

CCIE™ Service Provider Version 4

Written and Lab Exam Comprehensive Guide

**By: Nicholas J. Russo
CCIE™ #42518 (RS/SP)**

About the Author

Nicholas (Nick) Russo, CCIE™ #42518, holds active CCIE certifications in both Routing and Switching and Service Provider. Nick was among the first individuals to pass the CCIE Service Provider version 4 lab examination and this book represents his personal journey towards that end. Nick also holds a Bachelor's of Science in Computer Science, and a minor in International Relations, from the Rochester Institute of Technology (RIT). Nick lives in Maryland, USA with his wife, Carla. They are currently expecting their first child.



Dedications

This book is dedicated to my wife Carla, for without her support, I would have not even started this endeavor. Although I have spent years studying for multiple certifications, she continues to support me in every way. This is the mark of a true companion and I love her dearly for it.

Copyright 2016 Nicholas J. Russo

ISBN-10: 0-692-74737-0

ISBN-13: 978-0-692-74737-7

This material is not sponsored or endorsed by Cisco Systems, Inc. Cisco, Cisco Systems, CCIE and the CCIE Logo are trademarks of Cisco Systems, Inc. and its affiliates. The symbol ™ is included in the Logo artwork provided to you and should never be deleted from this artwork.

All Cisco products, features, or technologies mentioned in this document are trademarks of Cisco. This includes, but is not limited to, Cisco IOS®, Cisco IOS-XE®, and Cisco IOS-XR®. Within the body of this document, not every instance of the aforementioned trademarks are prepended with the symbols ® or ™ as they are demonstrated above.

The opinions expressed in this book belong to the author and are not necessarily those of Cisco.

THE INFORMATION HEREIN IS PROVIDED ON AN “AS IS” BASIS, WITHOUT ANY WARRANTIES OR REPRESENTATIONS, EXPRESS, IMPLIED OR STATUTORY, INCLUDING WITHOUT LIMITATION, WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Purpose: This book attempts to cover every topic in the CCIE Service Provider version 4 (SPv4) blueprint. The vast majority of technical topics, even topics only present on the written examination, have corresponding practical labs. In this way, the book is an educational resource focused more on developing true technical experts rather than training individuals to pass a test. By testing many advanced technologies in detail, such as Ethernet VPN and Segment Routing, the reader gains valuable insight as to the future of SP technologies.

Target audience: Individuals using this book should already have a strong understanding of core routing, switching, and SP technologies. The book does not detail the basics of routing, MPLS forwarding, or other topics considered “beneath” the scope of a CCIE certification. Very few of the labs in this book are single-technology focused. This is done intentionally to constantly exercise features working in concert (or disharmony) with one another. Readers should understand this and be knowledgeable on prerequisite topics as discussed in each chapter’s introduction.

Scope: This primarily focuses on CCIE SP version 4 topics as specified in the official blueprint published by Cisco. Other topics that do not appear on the blueprint, such as BGP customer multicast signaling and PPP over Ethernet (PPPoE), are documented briefly in this document as they are relevant for SP networking in general. Nonetheless, this focus remains on the core SP topics in the blueprint since this book is designed for CCIE SPv4 candidates and other SP networking professionals. The length and breadth of a section is often a good measure of how “important” it is. This is helpful for prioritizing one’s study time. Note that some blueprint topics may not be covered in an appropriate level of detail in this book; always consult the official blueprint to determine if a specific technology is testable or not.

How to use this book: The table of contents is hyper-linked to each chapter, so an ordinary “point-and-click” operation is an effective navigation tool. The table of contents is arrayed in a way that makes sense to the author, but this is not necessarily the best sequence to review the labs. Topologies are seldom recycled across major domains unless the topology is well-suited to a number of particular labs. Basic IP addressing and routing configurations will be briefly validated before each lab; this is done to conserve time. The core study topic at hand remains the focus of a particular section.

Reference material: An “*Additional Reading*” comment is included in every major technology area which identifies the suffix of the supporting document relating to a lab. This document will contain the original diagram embedded in the book, as well as all configuration files. These are included separately so that they may be viewed, printed, or modified. Below is a mapping of topic weights by Cisco for reference.

Topic	Written Weight (%)	Lab Weight (%)
Service Provider Architecture and Evolution	10	N/A
Core Routing	23	27
Service Provider Based Services	23	26
Access and Aggregation	17	17
High Availability and Fast Convergence	10	13
SP Security, SP Operation and Management	17	17

Contents

1.	SP architecture concepts	13
1.1	IPv6	13
1.1.1	Definitions	13
1.1.2	Neighbor Discovery details	16
1.2	Broadband Aggregation (BBA)	41
1.2.1	PPP over Ethernet (PPPoE) technology	42
1.2.2	Multi-service PPPoE and LAC/LNS architecture	70
1.3	MEF Ethernet Services Definitions (MEF 6.2)	93
1.4	Platform Architecture	94
1.4.1	Route-Switch Processor (RSP) and Route Processor (RP)	94
1.4.2	Line cards (LC)	95
1.4.3	Switching fabric / backplane and forwarding model	95
1.4.4	Multicast forwarding and hierarchical replication	96
1.4.5	Satellite operations (remote linecards)	96
3.1	WAN technologies	96
3.1.1	Packet over SONET/SDH	96
3.1.2	T1/E1 and T3/E3	97
3.1.3	Dense Wavelength Division Multiplexing (DWDM)	98
3.2	IP connectivity to the customer	99
3.2.1	Digital Subscriber Line (DSL)	99
3.2.2	Cable Internet	99
3.2.3	Wireline	99
4.	Virtualization concepts	100
4.1	SVR vs. HVR	100
4.2	Network Functions Virtualization (NFV)	101
4.3	Software Defined Networking (SDN)	101
5.	Mobility concepts	102
5.1	LTE	102
5.2	Backhaul	104
6.	Describe BGP path attributes	105
7.	Describe MPLS forwarding and control plane mechanisms	107

7.1	Label Distribution Protocol (LDP)	107
7.2	Static label bindings	166
7.3	MPLS IP and MTU minor options	170
8.	Describe MPLS advanced features	200
8.1	Segment Routing	200
8.2	Generalized MPLS (GMPLS)	212
8.3	MPLS Transport Profile (MPLS-TP)	213
8.4	Inter-AS MPLS	235
8.4.1	Option A (Back to back VRF exchange)	258
8.4.1.1	L3VPN	258
8.4.1.2	L2VPN	286
8.4.1.3	MVPN – GRE (Profile 0) and mLDP (Profile 1)	292
8.4.1.4	MPLS TE	310
8.4.1.5	Confederation variation	314
8.4.1.6	Carrier Supporting Carrier (CSC) variation	325
8.4.2	Option B (ASBR VPNv4/v6 eBGP)	331
8.4.2.1	L3VPN	333
8.4.2.2	L2VPN	368
8.4.2.3	mVPN – GRE (Profile 0)	379
8.4.2.4	MVPN – mLDP (Profile 17)	404
8.4.2.5	MPLS TE	413
8.4.2.6	Confederation variation	427
8.4.3	Option C (ASBR eBGP + Label, RR VPNv4 eBGP)	452
8.4.3.1	L3VPN	453
8.4.3.2	L2VPN	501
8.4.3.3	MVPN – GRE (Profile 0)	512
8.4.3.4	MVPN – mLDP (Profile 17)	519
8.4.3.5	MPLS TE	536
8.4.3.6	Confederation variation	563
8.4.4	Option AB Inter-AS hybrid (AKA Option D)	581
8.4.4.1	L3VPN	581
8.4.4.2	L2VPN	613

8.4.4.3	MVPN – GRE (Profile 0) and mLDP (Profile 1)	615
8.4.4.4	MPLS TE	623
8.4.5	Confederation variation	627
9.	Describe multicast P2MP TE	627
10.	Describe EVPN (EVPN and PBB-EVPN)	627
10.1	EVPN	630
10.2	PBB-EVPN	630
11.	Describe IEEE 802.1ad (QinQ), IEEE 802.1ah (Mac-in-Mac), and ITU G.8032 (REP)	646
11.1	802.1ad QinQ	646
11.2	802.1ah MAC in MAC (Provider Backbone Bridges)	648
11.3	Ethernet Ring loop-prevention	648
11.3.1	Cisco Resilient Ethernet Protocol (REP)	648
11.3.2	ITU G.8032	675
12.	Describe broadband forum TR-101 VLAN paradigms (N:1 and 1:1)	675
13.	Describe QoS link fragmentation (LFI), cRTP, and RTP	685
14.	Describe Multichassis/Clustering High Availability (HA)	694
14.1	High Availability (HA) Demonstration (NSF/NSR/GR)	696
14.1.1	IS-IS NSF and NSR	702
14.1.2	OSPFv2 NSF and NSR	707
14.1.3	OSPFv3 GR and NSR	710
14.1.4	BGP GR and NSR	712
14.1.5	LDP GR and NSR	720
14.1.6	RSVP-TE GR	726
14.1.7	EIGRP NSF	734
15.	Describe Layer 1 failure detection	737
16.	Describe BGPsec	740
17.	Describe backscatter traceback	740
18.	Describe lawful-intercept	740
19.	Describe BGP Flowspec	740
20.	Describe DDoS mitigation techniques	740
21.	Describe network event and fault management	741
22.	Describe performance management and capacity procedures	741

23.	Describe maintenance and operational procedures	744
24.	Describe the network inventory management process	745
25.	Describe network change, implementation, and rollback	745
25.1	Processes and best practices	745
25.2	NETCONF and YANG	747
26.	Describe the incident management process based on the ITILv3 framework	750
27.	Describe, implement, and troubleshoot advanced BGP features	751
27.1	Additional Paths (add-path) and Prefix Independent Convergence (PIC)	751
27.2	BGP RT-filter unicast / IPv4 RT-filter feature	818
27.3	BGP RR-group and Selective RT Retention	823
27.4	Accumulated IGP attribute	841
27.4.1	Basic AIGP	841
27.4.2	AIGP with cost-communities and BGP confederations	847
27.5	Cost-Community / Point Of Insertion (POI)	850
27.6	DMZ Link Bandwidth	865
27.7	BGP Multicast VPN (MVPN) Theory	881
27.8	BGP Link State AF and Path Computation Element (PCE)	884
28.	Describe, implement, and troubleshoot MVPN	890
28.1	Profile 0: Default MDT – GRE – PIM C–mcast Signaling (Traditional Draft-Rosen)	891
28.1.1	PIM-ASM in the core	893
28.1.2	PIM-SSM in the core	905
28.1.3	PIM-Bidir in the core	915
28.2	Profile 1: Default MDT – MLDP MP2MP – PIM C–mcast Signaling (Basic mLDLP)	924
28.3	Profile 3: Default MDT – GRE – BGP–AD – PIM C–mcast Signaling	951
28.4	Profile 6: VRF MLDP – In–band Signaling	960
28.5	Profile 7: Global MLDP In–band Signaling	969
28.6	Profile 8: Global Static – P2MP–TE	980
28.7	Profile 9: Default MDT – MLDP – MP2MP – BGP–AD – PIM C–mcast Signaling	987
28.8	Profile 10: VRF Static – P2MP TE - BGP–AD	993
28.9	Profile 11: Default MDT – GRE – BGP–AD – BGP C–mcast Signaling	1000
28.10	Profile 12: Default MDT – MLDP – P2MP – BGP–AD – BGP C–mcast Signaling	1011
28.11	Profile 13: Default MDT – MLDP – MP2MP – BGP–AD – BGP C–mcast Signaling	1030

28.12	Profile 14: Partitioned MDT – MLDP P2MP – BGP-AD – BGP C-mcast signaling	1061
28.13	Profile 17: Default MDT – MLDP P2MP – BGP-AD – PIM C-mcast signaling	1080
29.	Describe and optimize multicast scale and performance	1094
29.1	Inter-AS Multicast and Multicast Source Discovery Protocol (MSDP)	1094
29.2	Multicast Only Fast Re-Reroute (MoFRR)	1158
29.3	Protecting mLDP LSPs with Fast Re-Reroute (FRR)	1173
29.4	MVPN Extranet	1178
29.4.1	PIM/GRE	1179
29.4.2	mLDP	1205
30.	Describe, implement, and troubleshoot MPLS QoS models and related features	1233
30.1	Uniform	1234
30.2	Short pipe	1237
30.3	Pipe (AKA long pipe)	1238
30.4	QoS Policy Propagation through BGP (QPPB)	1240
30.5	QoS specifics on IOS XRv	1246
30.6	Network Based Application Recognition (NBAR) summary and configurations	1251
30.6.1	NBAR Custom Protocols	1253
30.6.2	NBAR Attributes	1258
30.6.3	NBAR Attributes with HTTP	1262
30.6.4	NBAR Protocol-ID	1267
30.6.5	NBAR Protocol Discovery	1268
31.	Describe, implement, and troubleshoot MPLS TE / QoS mechanisms	1270
31.1	MPLS RSVP-TE (General)	1270
31.1.1	TE Topology (TED) construction and RSVP-TE signaling	1270
31.1.2	TE attributes	1297
31.1.3	Directing traffic into TE tunnels and tunnel stitching	1338
31.2	TE Fast-ReRoute (FRR) and rapid provisioning	1363
31.2.1	Link (NHOP), Node (NNHOP), and Path protection – Manual	1363
31.2.2	Automatic tunnels (with OSPF)	1401
31.3	CBTS (IOS) and PBTS (XR)	1451
31.4	DiffServ-aware Traffic Engineering (DS-TE)	1469
31.4.1	Pre-standard Model	1470

31.4.2	IETF Russian Dolls Model (RDM)	1490
31.4.3	IETF Maximum Allocation Model (MAM)	1500
31.4.4	Per-VRF TE techniques	1507
32.	Describe, implement, and troubleshoot E-LAN and E-TREE (extended to general L2VPN)	1540
32.1	MPLS encapsulated L2VPN	1540
32.1.1	Static configuration	1540
32.1.1.1	E-LINE (VPWS)	1540
32.1.1.2	Advanced PW features (CW, Status, etc)	1562
32.1.1.3	E-LAN and E-TREE (VPLS)	1574
32.1.1.4	Multisegment PW (MS-PW) switching	1598
32.1.1.5	EVC rewrite operations	1622
32.1.2	BGP auto-discovery for VPWS/VPLS	1632
32.1.2.1	LDP signaling	1633
32.1.2.2	BGP signaling	1648
32.1.3	Hierarchical VPLS (H-VPLS)	1664
32.1.3.1	MPLS in the Access Network	1664
32.1.3.2	QinQ in the Access Network	1681
32.2	IP encapsulated L2VPN	1688
32.2.1	E-LINE with L2TP	1688
32.2.2	E-LAN and E-TREE using OTV	1714
33.	Describe, implement, and troubleshoot Unified MPLS and CSC	1731
33.1	Carrier Supporting Carrier (CSC)	1731
33.1.1	L3VPN	1739
33.1.2	L2VPN	1750
33.1.3	MVPN (Profile 0 with SSM)	1759
33.1.4	TE and TE-FRR	1768
33.2	Unified (seamless) MPLS	1780
33.2.1	IS-IS	1787
33.2.1.1	L3VPN	1797
33.2.1.2	L2VPN	1812
33.2.1.3	MVPN (mLDP profiles 1 and 17)	1816
33.2.1.4	Inter-area TE and TE-FRR	1824

33.2.2	OSPF (summarized)	1840
33.2.2.1	L3VPN	1843
33.2.2.2	L2VPN	1850
33.2.2.3	MVPN (mLDP profiles 1 and 17)	1856
33.2.2.4	MPLS TE and TE-FRR	1859
34.	Describe, implement, and troubleshoot LISP	1870
35.	Describe, implement, and troubleshoot GRE and mGRE-based VPN	1902
35.1	P2P GRE tunneling and GRE features	1902
35.2	Dynamic Multipoint VPN (DMVPN) basics	1916
35.2.1	Phase 1	1918
35.2.2	Phase 2	1938
35.2.3	Phase 3	1948
35.3	mGRE-based L3VPN	1964
36.	Describe, implement, and troubleshoot IPv6 transition mechanisms	1976
36.1	NAT44 and NAT444	1976
36.2	NAT64 and NAT464	1995
36.3	Dual stack lite (DS-lite)	2035
36.4	IPv6 tunneling over IPv4 networks	2037
36.4.1	GRE / Manual IPv6 tunnels	2038
36.4.2	6to4 automatic tunnels	2041
36.4.3	6 Rapid Deployment (6RD)	2045
36.4.4	Intra-Site Automatic tunnel Addressing Protocol (ISATAP)	2052
36.5	IPv4/IPv6 Internet Access over MPLS using NAT44	2055
37.	Describe, implement, and troubleshoot end-to-end fast convergence	2092
37.1	Loop Free Alternate (LFA) for IPv4	2092
37.1.1	OSPFv2	2092
37.1.1.1	Direct LFA	2092
37.1.1.2	Remote LFA	2106
37.1.2	IS-IS	2121
37.1.2.1	Direct LFA	2121
37.1.2.2	Remote LFA	2127
37.1.3	EIGRP	2131

37.2	Loop Free Alternate (LFA) for IPv6 (XR Only)	2136
37.2.1	OSPFv3	2136
37.2.1.1	Direct LFA	2136
37.2.1.2	Remote LFA	2140
37.2.2	IS-IS	2140
37.2.2.1	Direct LFA	2140
37.2.2.2	Remote LFA	2144
37.3	Convergence optimizations for BGP	2148
37.4	Convergence optimizations for IGPs	2174
37.4.1	IS-IS	2175
37.4.2	OSPFv2 and OSPFv3	2181
38.	Describe, implement, and troubleshoot multi-VRF CE and advanced VRF techniques	2194
38.1	Multi-VRF CE (VRF-Lite)	2195
38.1.1	Basic VRF-Lite	2195
38.1.2	OSPF and sham-links	2198
38.1.3	EIGRP and Site-of-Origin (SoO)	2233
38.1.4	IS-IS	2262
38.1.5	BGP and Site-of-Origin (SoO)	2266
38.1.6	Static routing	2289
38.1.7	RIP	2293
38.2	VRF label modes	2300
38.3	VRF selection for traffic leaking	2314
38.4	VRF route leaking	2318
38.5	L3VPN import/export maps	2338
38.6	Half-Duplex VRF (HDVRF)	2350
38.7	BGP Local Convergence (VRF Local Protection)	2363
39.	Describe, implement, and troubleshoot Layer 2 failure detection	2377
39.1	Link Aggregation Control Protocol (LACP)	2377
39.2	Uni-Directional Link Detection (UDLD)	2388
40.	Describe, implement, and troubleshoot Layer 3 failure detection	2396
40.1	Individual Protocol Hello packets	2396
40.2	Bidirectional Forwarding Detection (BFD)	2415

41.	Describe, implement, and troubleshoot control plane protection techniques	2444
41.1	Control Plane Policing (CPP) in XE and Local Packet Transport Services (LPTS) in XR	2444
42.	Describe, implement, and troubleshoot logging and SNMP security	2461
42.1	Logging	2461
42.2	SNMP security	2461
43.	Describe, implement, and troubleshoot timing	2461
43.1	Network Time Protocol (NTP)	2462
43.2	1588v2 (Precision Time Protocol(PTP))	2480
43.3	Synchronous Ethernet (SyncE)	2482
44.	Describe, implement, and troubleshoot SNMP traps, RMON, EEM, and EPC	2483
44.1	SNMP traps	2484
44.2	Remote Monitor (RMON) in XE and logging correlation in XR	2490
44.3	Embedded Event Manager (EEM)	2503
44.4	Embedded Packet Capture (EPC)	2512
45.	Describe, implement, and troubleshoot port mirroring protocols	2522
45.1	Switch port analyzer (SPAN)	2522
45.2	Remote SPAN (RSPAN)	2527
45.3	Encapsulated RSPAN (ERSPAN)	2530
46.	Describe, implement, and troubleshoot Netflow and IPFIX	2534
46.1	Flexible Netflow (FNF)	2536
46.2	IPFIX	2547
47.	Describe, implement, and troubleshoot IP SLA	2549
47.1	Basic IP SLA probes, responders, features, and configurations	2549
47.2	UDP-jitter and VOIP codec probes	2560
47.3	Advanced ICMP probes	2566
47.4	MPLS probes	2573
47.5	Ethernet probes including ITU-T Y.1731 Basics and Performing Monitoring (PM)	2577
47.6	Miscellaneous probes	2603
47.7	Aggregated statistics, history, group scheduling, and miscellaneous features	2610
47.8	Enhanced Object Tracking (EOT)	2622
47.9	IPv6 SLA	2637
47.10	IOS-XR IP SLA and EOT	2643

48.	Describe, implement, and troubleshoot MPLS OAM and Ethernet OAM	2667
48.1	MPLS ping, MPLS traceroute, and VCCV	2667
48.2	MPLS LSP Monitor (MPLSLM) / LSP Health Monitor	2690
48.3	Ethernet Management Tools (CFM, OAM, and E-LMI)	2703
48.3.1	Connectivity Fault Management (CFM) (802.3ag)	2703
48.3.2	Ethernet OAM (IEEE 802.3ah)	2733
48.3.3	Ethernet Local Management Interface (E-LMI) (MEF.16)	2748
48.3.4	Ethernet CFM, OAM, E-LMI, and Y.1731 on CSR1000v (Comprehensive)	2766
49.	Service Provider security best practices (Comprehensive)	2794
49.1	Control plane security best practices	2795
49.2	Management plane security best practices	2831
49.3	Data plane security best practices	2862
49.4	Advanced security techniques and features	2889

1. SP architecture concepts

1.1 IPv6

1.1.1 Definitions

Link-local address: Addressing within FE80::/10 (FE80:: through FEBF:FFFF...) to be used for communication on a link. The addressing is not routable and all routers must have LL addresses on all interfaces.

Site-local address: Addressing within FEC0::/10 (FEC0:: through FEFF:FFFF...) to be used within an organization. This is similar to RFC 1918 private addressing and is routable., but is discouraged. The unique-local addressing was meant to replace it.

Unique-local address (ULA): Addressing within FC00::/7 (FC00:: through FDFF:FFFF...) to be used within an organization. This replaced site-local addressing and serves the same function.

Multicast addresses: Addressing within FF00::/8 (anything starting with FF) to be used for multicast transport. Within the second byte, the first hex digit represents special flags while the second represents the scope. The flags, in binary, are "ORPT". The most significant bit is always 0 and means nothing.

1. 'R' indicates whether the IPv6 carries a PIM RP address. This is used for embedded RP and the RP address is signaled inside of the IPv6 multicast group.

2. 'P' indicates whether a multicast address is assigned based on the network prefix. This is used for embedded RP and the network prefix is embedded inside of the IPv6 address. If R is 1, P must also be 1, since the embedded RP construct implies that the network prefix is also carried in the IPv6 address. The

opposite is not true as 'P' could be 1 while 'R' is 0; a case may exist where network prefix information is carried in the multicast address but the function is unrelated to embedded RP.

3. 'T' indicates whether a multicast group is transient (dynamically/non-permanently assigned) or not. When T is 0, it assumes a well-known multicast address is used according to IANA. If P is 1, T must also be 1. The opposite is not true as 'T' could be 1 while 'P' is 0. This would represent a normal transient multicast group that does not carry any network prefix information.

The scopes are self-explanatory and are used to contain multicast into administrative regions.

1 - Interface local: Only useful for loopback transmission of multicast

2 - Link-local: Communication on a segment, typically used for IGP, PIM, neighbor discovery (ND), etc

4 - Admin-local: Smallest scope that can be administratively configured; that is, unlike node and link-local, this traffic is routable and the administrator decides what constitutes an "admin-local" boundary. This would be useful for limiting traffic to a set of devices within a site, such as access/distribution/core layers of a LAN-side routing architecture.

5 - Site-local: For use within a site. This would be useful for confining multicast traffic within local branch office. Although PIM dense-mode is not supported in IPv6 on Cisco platforms, a site-local sparse-mode domain may be a good alternative for local multicast confinement.

8 - Organization-local: Spans multiple sites within an organization, such as between branch offices. The information would typically not be allowed to be exchanged over the Internet.

E - Global scope: Sometimes called "VPN scope" by Cisco, this has no scoping limit.

Anycast addresses: Though the concept exists in IPv4, it does not exist on a LAN segment, and IPv6 enables this capability. Configuring an anycast address is essentially the same as a unicast address with duplicate address detection disabled (DAD is discussed later). When a host tries to resolve layer 2 addresses, any node may respond, hence the name anycast.

Solicited-node address: A link-local scope multicast address computed as a function of a node's unicast and anycast addresses. These addresses are formed by taking the low-order 24 bits of an IPv6 address and appending those bits to the prefix FF02::1:FF00::/104 (FF02::1:FF00:: to FF02::1:FFFF:FFFF). The network prefix length of 104 plus the low-order 24 bits of the unicast/anycast address on the interface creates the full 128-bit IPv6 solicited-node address. A node that has multiple prefixes but similar host addresses can therefore join less (hopefully only one) solicited-node multicast address. Every node must join a solicited-node multicast address for every unicast and any cast address on all interfaces, regardless of how they were configured (manual, DHCPv6, SLAAC, etc). This also reduces interrupts on nodes other than the target because the destination is not like an IPv4 ARP broadcast, or even an IPv6 all-nodes multicast. When a node sends traffic to a solicited-node address, it is like a semi-directed broadcast message that targets a very small set of nodes (again, hopefully only one).

Neighbor Solicitation (NS): ICMP type 135. The destination is the solicited-node multicast address of a specific host on the LAN, while the source is the link-local IPv6 address of the source interface. This is

used for LAN discovery and is directly comparable in function to an ARP request. The NS can also have a unicast destination when not being used for discovery. This is used to verify the reachability of a neighbor once discovered as a reachability probe and is known as Neighbor Unreachability Detection (NUD). NUD guarantees two-way communication in this way as well.

Neighbor Advertisement (NA): ICMP type 136. The destination is the link-local IPv6 address of the node that sent the NS (regular unicast packet) and the source is the LL address of the node sending the NA. The layer 2 address is contained within the packet's payload, and on Ethernet media this is the MAC address of the node sending the NA. If a node's layer 2 address changes, an unsolicited NA is sent to the all-nodes multicast address (FF02::1) to update their IPv6 neighbor tables. There is a solicit-flag that is 1 (true) only when the NA is sent in response to an NS, whereas the flag is 0 otherwise.

Router Solicitation (RS): ICMP type 133. These are sent by hosts to discover available routers on the segment. The source is the IPv6 link-local address of the sending interface (or :: if no address has been assigned yet) with a destination of the all-routers (FF02::2) multicast address. In this way, other IPv6 hosts will discard RS packets they receive since they are destined only for IPv6 routers, and because the source address can be unspecified (::), this facilitates SLAAC operation.

Router Advertisement (RA): ICMP type 134. These are periodically sent by routers with a source address of the interface LL address and destination of FF02::1. If sent in reply to an RS, it can also have a destination of the router's LL address that sent the RS. RA messages typically include: one or more prefixes for SLAAC (prefix-length must be 64 bits), prefix lifetime (validity), hop limit (TTL), MTU, and auto-configuration details. RA generation is enabled on Ethernet and FDDI interfaces by default and can be manually suppressed. On all other interfaces, it is disabled by default and can be manually enabled; one such use case of enabling it on a non-LAN interface would be to support ISATAP tunneling towards clients (discussed later). Two flags of are particular interest. The 'M' flag is the managed address configuration flag, which indicates that prefixes are available via DHCPv6. The 'O' flag indicates that other information, such as DNS, is available via DHCPv6 but addresses are not. If the 'M' flag is set, the 'O' flag is redundant/ignored, since all information is returned from DHCPv6 in that case. With both flags clear, this indicates that no information is available via DHCPv6. Regarding the router lifetime, a value of 0 indicates the router should not appear as a candidate default gateway; the lifetime only applies to the router's usefulness as a default gateway and no other RA components (prefixes have their own lifetimes).

Neighbor Redirect (NR): ICMP type 137. Used to notify a host of a better path to reach the destination. Same purpose as an IPv4 ICMP redirect, however the IPv6 NR must know the link-local address of the redirect target (i.e., the other router on the segment that is the better exit point). This LL address is contained in the payload of the NR message. An optional field that should be included, if known, is the target's layer 2 address as well. This saves the host receiving the redirect from having to use an NS to determine the next-hop, if it doesn't already have the information.

There are several validations that occur on these ND packets as well. For example, IPv6 nodes will

discard RA or RS messages that don't have a hop limit (TTL) of 255, which implies their origination was off-link and therefore is probably invalid.

Duplicate Address Detection (DAD): When a new address is configured on a link, DAD is typically run first before assigning the address to the link. The NS message is used with an unspecified source address (::) and the all-nodes multicast address (FF02::1). The tentative LL address that a node is checking for uniqueness is contained within the body of this NS. Two conditions will render the address "duplicate" and therefore unusable: reception of an NA from another node saying the address is already in use on the segment, or reception of an NS from another node that is concurrently trying to determine uniqueness. All IPv6 addresses (global and link-local) are subject to DAD, however DAD for LL addressing must happen first before progressing to additional IPv6 addresses. Cisco does not perform DAD on global or any cast addresses generated from 64-bit interface identifiers, such as EUI-64. It is assumed that these are unique and bypassing DAD is a minor optimization.

Default Router Preference (DRP): Signaled in unused bits within the RA message to provide low, medium, and high preference options for selecting a default gateway when multicast RAs offer it. Failure to evaluate/understand these bits results in a value of "medium".

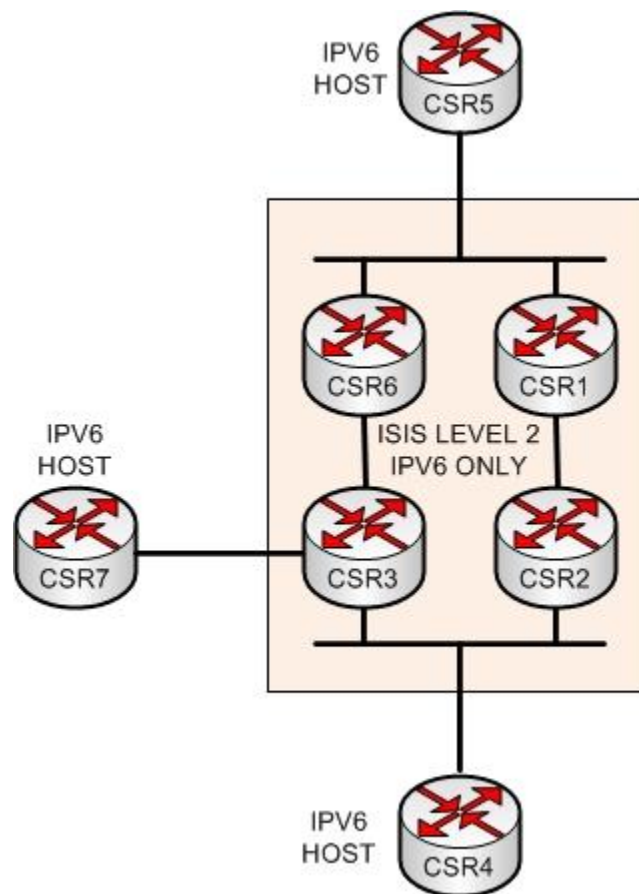
The IPv6 header differs from the IPv4 header in several ways. As expected, it is much larger at 40 bytes versus 20 bytes; each IPv6 address is 16 bytes by itself. The IPv4 TTL and IP protocol have been renamed to "hop limit" and "next header", but they are still both 1 byte fields with the same function. In IPv4, the TTL comes before the IP protocol, but in IPv6, the fields are reversed with "next header" coming before "hop limit". IPv6 also adds the concept of a flow label which assigns packets to a particular flow. It is 20 bits long, like an MPLS label, but has nothing to do with MPLS. The idea is that routers can do per-flow load sharing based on this information without having to look at higher layer protocols like TCP or UDP ports. Many protocols may have multiple flows but lack the concept of "ports" that TCP and UDP have. There is no way to verify the authenticity of the flow-label and it could be changed in transit, but since it is generally used for load-sharing, this may not be significant. A value of 0 indicates that the packet has not been assigned to a particular flow. Layer 2 (LAG) or layer 3 (CEF) mechanisms may use this for load-sharing.

The "next header" name is more appropriate for IPv6 since it can refer to one of two things. In a normal IPv6 packet, it would refer to the upper-layer protocol, such as 6 for TCP or 17 for UDP. It can also refer to IPv6 extension headers (EH) which immediately follow the normal IPv6 header. These are like IPv4 options that allow IPv6 to carry extra information. Some of these headers include the routing header (43), mobility header (135), fragment header (44), and destination options (60). IPv6 doesn't support fragmentation on the routers, but end hosts do, assuming they support these IPv6 EH options.

1.1.2 Neighbor Discovery details

This lab uses CSRs only since XRv does not appear to support sending RAs under any circumstance. Because XRv is modeled after an RSP, not a line-card, it cannot issue RA messages. I have included configurations for XRv1 and XRv2 that can be hot-swapped with CSR1 and CSR2 should the code be fixed

later. Basic IS-IS single-topology IPv6 routing is used for reachability across the network. CSR4, CSR5, and CSR7 represent end hosts with very little configuration.



First, we will examine the ND process using CSR1 and CSR2. This is simple because there are only two routers on the segment, making the RA/RS process unnecessary. In cases like this, disabling RAs makes sense to conserve resources and increase security. Although not necessary on transit links, I configured a global unicast address range as well. The relevant configuration from CSR1 is shown below; CSR2 has an identical configuration with different host addresses.

```
! CSR1
interface GigabitEthernet2.512
  ipv6 address FE80::11 link-local
  ipv6 address 2020:0:11:12::11/64
  ipv6 nd ra suppress all
```

Debugging ICMPv6 and ND on CSR1 allows us to see many details about what happens during the process described earlier. We bounce CSR1's link to CSR2 to see the full procedure. For clarity, the debug is broken into chunks and explained in line. First, IPv6 ND is notified that the layer 2 components of the link came up, which starts the ND process at layer 3. Before anything else, DAD must be run on the LL address of the link after a short delay. The DAD message is just an NS to the solicited-node

address of CSR1; this is a way to see if anyone else has the same low-order 24 bits of the host address as CSR1. DAD sees no response after 1 second (globally adjustable, as we see later) and declares the address unique. CSR1 then issues an unsolicited NA to the all-nodes multicast group to notify them of its MAC address binding to this IPv6 LL address.

```
R1#debug ipv6 icmp
R1#debug ipv6 nd
22:28:32.249: ICMPv6-ND: (GigabitEthernet2.512) L2 came up
22:28:32.249: IPv6-Addrmgr-ND: DAD request for FE80::11 on
GigabitEthernet2.512
22:28:32.249: ICMPv6-ND: Delay DAD for FE80::11 on GigabitEthernet2.512 by
200 msec
22:28:32.449: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) Sending DAD NS
[6F530]
22:28:32.450: ICMPv6: Sent N-Solicit, Src=::, Dst=FF02::1:FF00:11
22:28:33.449: IPv6-Addrmgr-ND: DAD: FE80::11 is unique.
22:28:33.449: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) Sending NA to
FF02::1
22:28:33.449: ICMPv6-ND: (GigabitEthernet2.512) L3 came up
22:28:33.449: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) Linklocal Up
22:28:33.450: ICMPv6: Sent N-Advert, Src=FE80::11, Dst=FF02::1
```

Next, DAD iterates through the rest of the unicast and anycast addresses on the link. The DAD process for subsequent addresses need not be delayed since there was not another link-up event. The solicited-node address happens to be the same in this case because the host addresses for the LL and global address are the same, but could be different. As expected, there are no duplicate addresses on the LAN between CSR1 and CSR2. CSR1 issues another unsolicited NA, this time sourced from the global address, to notify other nodes on the segment about its global address.

```
! CSR1
22:28:33.449: IPv6-Addrmgr-ND: DAD request for 2020:0:11:12::11 on
GigabitEthernet2.512
22:28:33.449: ICMPv6-ND: (GigabitEthernet2.512,2020:0:11:12::11) Sending DAD
NS [6F530]
22:28:33.451: ICMPv6: Sent N-Solicit, Src=::, Dst=FF02::1:FF00:11
22:28:34.449: IPv6-Addrmgr-ND: DAD: 2020:0:11:12::11 is unique.
22:28:34.449: ICMPv6-ND: (GigabitEthernet2.512,2020:0:11:12::11) Sending NA
to FF02::1
22:28:34.451: ICMPv6: Sent N-Advert, Src=2020:0:11:12::11, Dst=FF02::1
```

A few seconds later, IS-IS converges. Because IS-IS does not rely on IP, CSR1 has no idea about CSR2's existence and has no reason to resolve its layer 2 address. After convergence, IS-IS routes are learned via CSR2 and installed in the routing table, which prompts CSR1 to resolve the IPv6 next-hops, which are LL addresses. The ND state machine for FE80::12 (CSR2 address) transitions from deleted (nonexistent) to incomplete (INCMP). At this time, CSR1 send another NS to CSR2's solicited-node address; it gleans the solicited-node address from the low-order 24 bits of the address it is trying to resolve, which is

FE80::12. About 200 ms later, CSR2 responds with an NA which carries its MAC address. The NA packet is validated for security reasons, and the IPv6 neighbor entry transitions from incomplete to reachable.

```
! CSR1
22:28:40.936: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) ULP neighbour
22:28:40.936: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) DELETE -> INCMP
22:28:40.936: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) Sending NS
22:28:40.936: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) Set ULP NUD
22:28:40.937: ICMPv6: Sent N-Solicit, Src=FE80::11, Dst=FF02::1:FF00:12
22:28:41.147: ICMPv6: Received N-Advert, Src=FE80::12, Dst=FE80::11
22:28:41.147: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) Received NA from
FE80::12
22:28:41.147: ICMPv6-ND: Validating ND packet options: valid
22:28:41.147: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) LLA 0012.1212.1212
22:28:41.147: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) INCMP -> REACH
```

We can verify this entry by checking the IPv6 neighbor table, which is like the IPv4 ARP table. We see the IPv6 LL address and MAC address of CSR2 as reachable. Notice there is no entry for CSR2's global unicast address. The only reason CSR1 knew to look for CSR2 was because IS-IS next-hops necessitated it. CSR1 remains ignorant about anyone else on the LAN.

```
R1#show ipv6 neighbors gig 2.512
IPv6 Address                               Age Link-layer Addr State Interface
FE80::12                                   0 0012.1212.1212 REACH Gi2.512
```

```
R1#show ipv6 route isis | begin ^I2
I2  2020:0:3:6::/64 [115/30]
    via FE80::12, GigabitEthernet2.512
I2  2020:3:4:12::/64 [115/20]
    via FE80::12, GigabitEthernet2.512
I2  2020:5:6:11::/64 [115/40]
    via FE80::12, GigabitEthernet2.512
I2  FD00:3:4:12::/64 [115/20]
    via FE80::12, GigabitEthernet2.512
```

We can trick the router into trying to discover a new node in a few ways. The most obvious is to ping a new LL address out of that interface, which will trigger ND. A more subtle way is to configure a static route to a bogus next-hop, which like the IS-IS routes, will trigger ND. Below we configure a bogus default route; IPv6 ND makes three attempts to resolve the layer 2 address (one second apart), then gives up and delete the ND cache entry.

```
! CSR1
ipv6 route ::/0 GigabitEthernet2.512 FE80::BEEF

R1#debug ipv6 icmp
R1#debug ipv6 nd
22:52:28.302: ICMPv6-ND: (GigabitEthernet2.512,FE80::BEEF) DELETE -> INCMP
```

```

22:52:28.302: ICMPv6-ND: (GigabitEthernet2.512,FE80::BEEF) Sending NS
22:52:28.303: ICMPv6: Sent N-Solicit, Src=FE80::11, Dst=FF02::1:FF00:BEEF
22:52:29.393: ICMPv6-ND: (GigabitEthernet2.512,FE80::BEEF) Sending NS
22:52:29.393: ICMPv6: Sent N-Solicit, Src=FE80::11, Dst=FF02::1:FF00:BEEF
22:52:30.483: ICMPv6-ND: (GigabitEthernet2.512,FE80::BEEF) Sending NS
22:52:30.483: ICMPv6: Sent N-Solicit, Src=FE80::11, Dst=FF02::1:FF00:BEEF
22:52:31.573: ICMPv6-ND: (GigabitEthernet2.512,FE80::BEEF) INCMP -> DELETE
22:52:31.573: ICMPv6-ND: Remove ND cache entry

```

We can force ND for CSR2's global unicast address by pinging it. The debug also shows the ICMPv6 echo and echo-reply packets to confirm that everything worked. The first echo request is shown first, with the first echo reply shown last. Notice that CSR1 also receives an NS for its solicited-node address; this is because CSR2 also has to run ND to reach CSR1's global unicast address to send the echo-reply. CSR1 replies with a unicast NA to CSR2 (solicited), and after that, the ICMP flow succeeds.

```

! CSR1
22:56:36.499: ICMPv6: Sent echo request, Src=2020:0:11:12::11,
Dst=2020:0:11:12::12
22:56:36.501: ICMPv6-ND: (GigabitEthernet2.512,2020:0:11:12::12) DELETE ->
INCMP
22:56:36.503: ICMPv6-ND: (GigabitEthernet2.512,2020:0:11:12::12) Sending NS
22:56:36.503: ICMPv6-ND: (GigabitEthernet2.512,2020:0:11:12::12) Queued data
for resolution
22:56:36.504: ICMPv6: Sent N-Solicit, Src=FE80::11, Dst=FF02::1:FF00:12
22:56:36.507: ICMPv6: Received N-Advert, Src=2020:0:11:12::12, Dst=FE80::11
22:56:36.507: ICMPv6-ND: (GigabitEthernet2.512,2020:0:11:12::12) Received NA
from 2020:0:11:12::12
22:56:36.507: ICMPv6-ND: Validating ND packet options: valid
22:56:36.507: ICMPv6-ND: (GigabitEthernet2.512,2020:0:11:12::12) LLA
0012.1212.1212
22:56:36.507: ICMPv6-ND: (GigabitEthernet2.512,2020:0:11:12::12) INCMP ->
REACH
22:56:36.514: ICMPv6: Received N-Solicit, Src=2020:0:11:12::12,
Dst=FF02::1:FF00:11
22:56:36.514: ICMPv6-ND: (GigabitEthernet2.512,2020:0:11:12::11) Received NS
from 2020:0:11:12::12
22:56:36.514: ICMPv6-ND: Validating ND packet options: valid
22:56:36.514: ICMPv6-ND: (GigabitEthernet2.512,2020:0:11:12::11) Sending NA
to 2020:0:11:12::12
22:56:36.515: ICMPv6: Sent N-Advert, Src=2020:0:11:12::11,
Dst=2020:0:11:12::12
22:56:36.517: ICMPv6: Received echo reply, Src=2020:0:11:12::12,
Dst=2020:0:11:12::11

```

We can see the obvious differences in solicited-node addresses between CSR1 and CSR2 on this segment since they have different low-order 24 bit host addresses. We can try to trick DAD by configuring different host addresses on CSR1 and CSR2 but with the low-order 24 bits being equal. We will add these

as additional IPv6 address rather than replace the existing one. We quickly verify the solicited-node addresses on both routers to ensure they are the same for this new IPv6 address.

```
! CSR1
interface GigabitEthernet2.512
  ipv6 address 2020:0:11:12:0:11:0:1212/64

! CSR2
  ipv6 address 2020:0:11:12:0:12:0:1212/64

R1#show ipv6 interface gig2.512 | section group_add
  Joined group address(es):
    FF02::1
    FF02::1:FF00:11
    FF02::1:FF00:1212

R2#show ipv6 interface gig2.512 | section group_add
  Joined group address(es):
    FF02::1
    FF02::2
    FF02::1:FF00:12
    FF02::1:FF00:1212
```

The debug below shows that DAD is smart enough to determine if the address is unique or not. Even if CSR1 and CSR2 have the same solicited-node address, the actual IPv6 address in question is contained within the NS payload. CSR2 is joined to FF02::1:FF00:1212, as is CSR1, so CSR2 actually has to open the packet and process it. Had the host addresses been different, CSR2 would have discarded the packet at layer 3, saving it a little bit of CPU time. The solicited-node is not what DAD uses for its final decision, as it is really used as a CPU interrupt reduction technique. The timestamps are not perfectly synchronized between CSR1 and CSR2, but it is clear that CSR2 receives the DAD NS, does nothing, then receives the authoritative NA from CSR1 declaring the address unique. This is the correct behavior.

```
! CSR1
13:31:30.714: IPv6-Addrmgr-ND: DAD request for 2020:0:11:12:0:11:0:1212 on
GigabitEthernet2.512
13:31:30.715: ICMPv6-ND: (GigabitEthernet2.512,2020:0:11:12:0:11:0:1212)
Sending DAD NS [C0BB4]
13:31:30.716: ICMPv6: Sent N-Solicit, Src=::, Dst=FF02::1:FF00:1212
13:31:31.715: IPv6-Addrmgr-ND: DAD: 2020:0:11:12:0:11:0:1212 is unique.
13:31:31.715: ICMPv6-ND: (GigabitEthernet2.512,2020:0:11:12:0:11:0:1212)
Sending NA to FF02::1
13:31:31.717: ICMPv6: Sent N-Advert, Src=2020:0:11:12:0:11:0:1212,
Dst=FF02::1

! CSR2
13:31:31.303: ICMPv6: Received N-Solicit, Src=::, Dst=FF02::1:FF00:1212
13:31:31.303: ICMPv6-ND: (GigabitEthernet2.512,2020:0:11:12:0:11:0:1212)
```

```

Received NS from ::
13:31:32.303: ICMPv6: Received N-Advert, Src=2020:0:11:12:0:11:0:1212,
Dst=FF02::1
13:31:32.303: ICMPv6-ND: (GigabitEthernet2.512,2020:0:11:12:0:11:0:1212)
Received NA from 2020:0:11:12:0:11:0:1212
13:31:32.303: ICMPv6-ND: Validating ND packet options: valid

```

In the event there really is a duplicate address, DAD will detect this. Obviously, the solicited-node multicast addresses will be the same, and opening the packet for processing will alert DAD to the duplicated address on the segment. To save a little bit of memory, we can configure a duplicate address (2020::1212) that uses the same solicited-node address of something each router has already joined, which can reduce the number of solicited-node addresses a router must join. The duplicate address below also joins FF02::1:FF00:1212 since the low-order 24 bits of the host address are 0x001212 in hex. Assuming CSR2 has the address configured and we add it to CSR1 later, the debugs are shown below. CSR1 sends the DAD NS and immediately receives an NA back from CSR2; when the address is unique, the DAD NS should not receive an NA in response. Both routers also display a syslog message in case debugging is not enabled. CSR1 calls this a syslog warning (level 4) since it tried to use an address already in use. CSR2 calls this a syslog informational message (level 6) since someone else is attempting to use an address already valid on CSR2.

```

! CSR1
13:42:37.470: IPv6-Addrmgr-ND: DAD request for 2020::1212 on
GigabitEthernet2.512
13:42:37.470: ICMPv6-ND: (GigabitEthernet2.512,2020::1212) Sending DAD NS
[598ED]
13:42:37.471: ICMPv6: Sent N-Solicit, Src=::, Dst=FF02::1:FF00:1212
13:42:37.475: ICMPv6: Received N-Advert, Src=2020::1212, Dst=FF02::1
13:42:37.475: ICMPv6-ND: (GigabitEthernet2.512,2020::1212) Received NA from
2020::1212
13:42:37.475: ICMPv6-ND: Validating ND packet options: valid
13:42:37.475: %IPV6_ND-4-DUPLICATE: Duplicate address 2020::1212 on
GigabitEthernet2.512

! CSR2
13:42:38.057: ICMPv6: Received N-Solicit, Src=::, Dst=FF02::1:FF00:1212
13:42:38.057: ICMPv6-ND: (GigabitEthernet2.512,2020::1212) Received NS from
::
13:42:38.057: ICMPv6-ND: Packet contains no options
13:42:38.057: ICMPv6-ND: Validating ND packet options: valid
13:42:38.057: ICMPv6-ND: Packet contains no options
13:42:38.057: ICMPv6-ND: (GigabitEthernet2.512,2020::1212) Sending NA to
FF02::1
13:42:38.057: %IPV6_ND-6-DUPLICATE_INFO: DAD attempt detected for 2020::1212
on GigabitEthernet2.512
13:42:38.058: ICMPv6: Sent N-Advert, Src=2020::1212, Dst=FF02::1

```

We can verify this by checking the interface details on each router. The IPv6 address 2020::1212 is marked as [DUP] to indicate it is a duplicate address on CSR1. CSR2 does not show this because it had the address first; DAD honors “first come, first served” in terms of address claims. The syslog message priorities above seem to support this conclusion as well. We will see how to work around DAD issues later.

```
R1#show ipv6 interface gig2.512 | section Global_uni
Global unicast address(es):
  2020::1212, subnet is 2020::/64 [DUP]
  2020:0:11:12::11, subnet is 2020:0:11:12::/64
  2020:0:11:12:0:11:0:1212, subnet is 2020:0:11:12::/64
```

```
R2#show ipv6 interface gig2.512 | section Global_uni
Global unicast address(es):
  2020::1212, subnet is 2020::/64
  2020:0:11:12::12, subnet is 2020:0:11:12::/64
  2020:0:11:12:0:12:0:1212, subnet is 2020:0:11:12::/64
```

The ND state machine is quick to transition ND entries out of the REACH state. By default, this is 30,000 ms (30 seconds) on all interfaces. It can be tuned globally or at the interface level. Below, CSR1 runs ND to CSR2’s LL address. The entry was in the STALE state, and after the ping, ND transitions the entry to the REACH state. The ND process doesn’t need to occur again since we still had a STALE entry, and the successful ping shows us that the entry is still valid. ND now marks it as REACH, but after 30 seconds, the entry transitions back to the STALE state since no traffic is flowing through this address presently. This STALE state helps the administrator determine how long it has been since traffic was sent to an IPv6 peer.

```
! CSR1
13:52:45.031: ICMPv6: Sent echo request, Src=FE80::11, Dst=FE80::12
13:52:45.034: ICMPv6: Received echo reply, Src=FE80::12, Dst=FE80::11
13:52:45.034: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) ULP indication
13:52:45.034: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) STALE -> REACH
13:53:15.085: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) REACH -> STALE
```

We will adjust this timer on CSR1 facing CSR2 to have a 10 second transition so that ND cache entries are moved to the STALE state more quickly when not used. Running the same ND test again, we can see the transition happens in 10 seconds, as expected.

```
! CSR1
interface GigabitEthernet2.512
  ipv6 nd reachable-time 10000
```

```
R1#show ipv6 interface gig2.512 | include ND_reachable
ND reachable time is 10000 milliseconds (using 10000)
```

```

14:02:25.842: ICMPv6: Sent echo request, Src=FE80::11, Dst=FE80::12
14:02:25.846: ICMPv6: Received echo reply, Src=FE80::12, Dst=FE80::11
14:02:25.846: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) ULP indication
14:02:25.846: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) STALE -> REACH
14:02:35.934: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) REACH -> STALE

```

Entries are removed from the IPv6 ND cache after having been stale for 4 hours by default. This can also be adjusted globally or at the interface level. CSR1 will adjust this globally to delete entries that are stale for 50 seconds. Thus, on the interface to CSR2, an entry moves from REACH to STALE after 10 seconds, then STALE to DELETE after 50 seconds, meaning that there is one minute between traffic sent to a next-hop and the cache entry being totally removed.

```

! CSR1
ipv6 nd cache expire 50

```

There does not appear to be a show command to verify this, but we can check the IPv6 cache statistics to see the number of cache entries in each state. This command was run after the entries aged out (given the 50 second DELETE timer), so there are zero entries in the cache currently.

```

R1#show ipv6 neighbors statistics
IPv6 ND Statistics
  Entries 0, High-water 4, Gleaned 1, Scavenged 3, Static 0
Entry States
  INCMP 0 REACH 0 STALE 0 GLEAN 0 DELAY 0 PROBE 0
Resolutions
  Requested 11, timeouts 10, resolved 7, failed 3
  In-progress 0, High-water 2, Throttled 0, Data discards 0
NUD
  Requested 1, timeouts 0, resolved 1, failed 0
  in-progress 0, high-water 1, throttled 0, current queue 0, queue high-
water 0
  Delayed Queue 0, Delayed Queue High-water 4

```

Repeating the same test again, we confirm this behavior on CSR1 by verifying the timestamps.

```

! CSR1
14:04:53.505: ICMPv6: Sent echo request, Src=FE80::11, Dst=FE80::12
14:04:53.508: ICMPv6: Received echo reply, Src=FE80::12, Dst=FE80::11
14:04:53.508: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) ULP indication
14:04:53.508: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) STALE -> REACH
14:05:03.548: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) REACH -> STALE
14:05:53.600: ICMPv6-ND: STALE deleted: FE80::12
14:05:53.600: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) STALE -> DELETE
14:05:53.601: ICMPv6-ND: Remove ND cache entry

```

From the statistics show command, we see there are other states such as GLEAN, DELAY, and PROBE.

The GLEAN state doesn't actually show up in the ND neighbor cache, but it is a valid state. When an unsolicited NA is received on the segment, routers will ignore those entries (like ignoring a gratuitous ARP) to save memory. For example, if we bounce CSR2's interface, it will perform DAD on all of its addresses beginning with its LL address. CSR1 doesn't see the DAD NS because it isn't joined to the same solicited-node address, but it does see the NA that CSR2 sends once DAD declares the address unique. CSR1 does nothing with it; no further processing is done on its IPv6 cache. CSR1 will need it later for IS-IS routing, but that is beyond the scope of this test. Notice that CSR1 has no entry for FE80::12 in its cache.

```
! CSR1
14:13:09.122: ICMPv6: Received N-Advert, Src=FE80::12, Dst=FF02::1
14:13:09.122: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) Received NA from
FE80::12
14:13:09.122: ICMPv6-ND: Validating ND packet options: valid

R1#show ipv6 neighbors gig2.512
[no output]
```

We can configure CSR1 to record these unsolicited NA mappings on a per-interface basis. CSR1 can "glean" that information by snooping the LAN, which may speed convergence and reduce independent ND conversations later. The cost is larger ND caches (more memory) for address that may not be relevant for the traffic patterns on a given LAN. Once configured, we can verify it by checking the IPv6 interface details.

```
! CSR1
interface GigabitEthernet2.512
  ipv6 nd na glean

R1#show ipv6 interface gig2.512 | include glean
ND gleaning on unsolicited neighbor advertisements
```

This time, when CSR2 sends the NA onto the LAN, CSR1 is directed to glean the layer 2 address for this unsolicited NA. The entry is recorded as STALE in the ND cache, which makes sense since CSR1 has no idea if the address is actually reachable as it did not initiate an ND conversation with it, nor direct traffic to/through it. This entry is still subject to the ND expiration timer configured earlier.

```
! CSR1
14:16:45.688: ICMPv6: Received N-Advert, Src=FE80::12, Dst=FF02::1
14:16:45.688: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) Received NA from
FE80::12
14:16:45.688: ICMPv6-ND: Validating ND packet options: valid
14:16:45.688: ICMPv6-ND: Glean unsolicited NA
14:16:45.688: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) Glean
14:16:45.688: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) LLA 0012.1212.1212
14:16:45.688: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) INCMP -> STALE
```

This process happens for all of CSR2's addresses, and CSR1 gleans them all and records them as stale. The IPv6 cache statistics counts them as GLEAN entries, despite their operational capacity being "stale" in a sense.

```
R1#show ipv6 neighbors gig2.512
IPv6 Address                               Age Link-layer Addr State Interface
2020::1212                                0 0012.1212.1212 STALE Gi2.512
2020:0:11:12::12                           0 0012.1212.1212 STALE Gi2.512
2020:0:11:12:0:12:0:1212                   0 0012.1212.1212 STALE Gi2.512
FE80::12                                    0 0012.1212.1212 STALE Gi2.512
```

```
R1#show ipv6 neighbors statistics
IPv6 ND Statistics
Entries 4, High-water 4, Gleaned 9, Scavenged 8, Static 0
Entry States
  INCMP 0 REACH 0 STALE 0 GLEAN 4 DELAY 0 PROBE 0
Resolutions
  Requested 12, timeouts 10, resolved 7, failed 3
  In-progress 0, High-water 2, Throttled 0, Data discards 0
NUD
  Requested 1, timeouts 0, resolved 1, failed 0
  in-progress 0, high-water 1, throttled 0, current queue 0, queue high-
water 0
  Delayed Queue 0, Delayed Queue High-water 4
```

IPv6 NUD also accounts the presence of IGP to reduce ND traffic. When routes are learned from an IGP, the next-hop will be a LL address. As seen earlier, installation of those routes in the RIB triggers ND for the next-hops whether traffic is flowing to those destinations or not. The router needing to resolve the remote next-hop sends an NS to the target's solicited-node address. If there is an IGP neighbor with that node, NUD assumes there is reachability to it, and does not wait for the NA to return before identifying the cache entry as REACH. This is enabled by default on all interfaces; the debug below on CSR2 shows that the cache entry was moved to the REACH state before the NA was received from CSR1. CSR2 knows the MAC address for CSR1 only because it received an NS from CSR1 who was performing ND for CSR2 at the same time.

```
! CSR2
15:07:59.194: %CLNS-5-ADJCHANGE: ISIS: Adjacency to R1 (GigabitEthernet2.512)
Up, new adjacency
15:07:59.194: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) ULP neighbour
15:07:59.194: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) DELETE -> INCMP
15:07:59.194: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) Sending NS
15:07:59.194: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) Set ULP NUD
15:07:59.195: ICMPv6: Sent N-Solicit, Src=FE80::12, Dst=FF02::1:FF00:11
15:07:59.272: ICMPv6: Received N-Solicit, Src=FE80::11, Dst=FF02::1:FF00:12
15:07:59.272: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) Received NS from
FE80::11
15:07:59.272: ICMPv6-ND: Validating ND packet options: valid
```

```

15:07:59.272: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) LLA 0011.1111.1111
15:07:59.272: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) INCMP -> STALE
15:07:59.273: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) STALE -> REACH
15:07:59.279: ICMPv6: Received N-Advert, Src=FE80::11, Dst=FE80::12
15:07:59.279: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) Received NA from
FE80::11
15:07:59.279: ICMPv6-ND: Validating ND packet options: valid

```

We can disable this behavior on CSR2, which ignores that symmetric NS coming from CSR1 in terms of honoring the source MAC address. Notice that CSR2 still stores the MAC address from CSR1, carried in the NS message. However, this transitions the entry to the DELAY state once CSR2 responds to CSR1's NS with an NA, not the REACH state. While in the DELAY state, the assumption is that we have told our neighbor about our MAC address using a solicited NA, and we are simply waiting for the neighbor to do the same. Until then, the entry is not marked as REACH. NUD is waiting for the solicited NA to come back from CSR1 which authoritatively identifies CSR1's MAC address (and implies reachability without relying on IGP).

```

! CSR2
interface GigabitEthernet2.512
 no ipv6 nd nud igp

! CSR2
15:06:02.357: %CLNS-5-ADJCHANGE: ISIS: Adjacency to R1 (GigabitEthernet2.512)
Up, new adjacency
15:06:02.357: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) ULP neighbour
15:06:02.357: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) DELETE -> INCMP
15:06:02.358: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) Sending NS
15:06:02.358: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) Set ULP NUD
15:06:02.358: ICMPv6: Sent N-Solicit, Src=FE80::12, Dst=FF02::1:FF00:11
15:06:02.434: ICMPv6: Received N-Solicit, Src=FE80::11, Dst=FF02::1:FF00:12
15:06:02.434: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) Received NS from
FE80::11
15:06:02.434: ICMPv6-ND: Validating ND packet options: valid
15:06:02.434: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) LLA 0011.1111.1111
15:06:02.434: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) INCMP -> STALE
15:06:02.434: ICMPv6-ND: (GigabitEthernet2.512,FE80::12) Sending NA to
FE80::11
15:06:02.435: ICMPv6: Sent N-Advert, Src=FE80::12, Dst=FE80::11
15:06:02.436: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) STALE -> DELAY
15:06:02.443: ICMPv6: Received N-Advert, Src=FE80::11, Dst=FE80::12
15:06:02.443: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) Received NA from
FE80::11
15:06:02.443: ICMPv6-ND: Validating ND packet options: valid
15:06:02.443: ICMPv6-ND: (GigabitEthernet2.512,FE80::11) DELAY -> REACH

```

Next, we will examine the RS and RA messages. Cisco routers will always send RA messages out of their IPv6 LAN interfaces unless suppressed. Suppressing them makes sense on transit links, such as CSR1-

CSR2 and CSR3-CSR6, where there are no hosts. Leaving the “all” keyword off the command only suppresses unsolicited, periodic RA messages. The “all” keyword ensures that router does not respond to RS messages with a solicited RA, either. Only CSR2 is shown, but this is configured on all transit links.

```
! CSR2
interface GigabitEthernet2.512
  ipv6 nd ra suppress all

R2#show ipv6 interface gig2.512 | include ND_RA
ND RAs are suppressed (all)
```

RAs are allowed on the LAN segments upon which CSR4 and CSR5 are hosted. This allows to hosts to discover the routers and automatically obtain IPv6 addresses from the on-link prefix(es).

```
! CSR5
interface GigabitEthernet2.556
  ipv6 address autoconfig default
```

We will examine the basic ND process between a host requiring autoconfiguration and the routers on the segment by debugging on CSR5. As expected, the very first thing all IPv6 nodes do is run DAD for their LL address. Since CSR5 has no explicit LL address, the EUI-64 process is used. This takes the 48-bit MAC address, inserts the hex string 0xFFFE into the middle of it, and sets the U/L bit in the MAC address to 1. With a MAC address of 0055.5555.5555, the EUI-64 address becomes 0255:55FF:FE55:5555. The prefix is FE80::/10 as always; CSR5 ensures its EUI-64 address is unique before doing anything else. After 1 second of not seeing an NA in response, it assumes the address is unique, and sends an unsolicited NA onto the segment to announce it.

```
! CSR5
15:33:46.200: ICMPv6-ND: (GigabitEthernet2.556) L2 came up
15:33:46.200: IPv6-Addrmgr-ND: DAD request for FE80::255:55FF:FE55:5555 on
GigabitEthernet2.556
15:33:46.201: ICMPv6-ND: Delay DAD for FE80::255:55FF:FE55:5555 on
GigabitEthernet2.556 by 200 msec
15:33:46.401: ICMPv6-ND: (GigabitEthernet2.556,FE80::255:55FF:FE55:5555)
Sending DAD NS [A23BB]
15:33:46.402: ICMPv6: Sent N-Solicit, Src=::, Dst=FF02::1:FF55:5555
15:33:47.401: IPv6-Addrmgr-ND: DAD: FE80::255:55FF:FE55:5555 is unique.
15:33:47.401: ICMPv6-ND: (GigabitEthernet2.556,FE80::255:55FF:FE55:5555)
Sending NA to FF02::1
15:33:47.401: ICMPv6-ND: (GigabitEthernet2.556) L3 came up
15:33:47.402: ICMPv6-ND: (GigabitEthernet2.556,FE80::255:55FF:FE55:5555)
Linklocal Up
15:33:47.403: ICMPv6: Sent N-Advert, Src=FE80::255:55FF:FE55:5555,
Dst=FF02::1
```

CSR5 also needs a globally routable address, but it has no idea what the on-link prefixes are. It needs to

check for routers on the segment by issuing an RS message to the all-routers multicast group sourced from its LL address.

```
! CSR5
15:33:47.756: ICMPv6-ND: (GigabitEthernet2.556) Sending RS
15:33:47.763: ICMPv6: Sent R-Solicit, Src=FE80::255:55FF:FE55:5555,
Dst=FF02::2
```

CSR5 receives a solicited RA from CSR1 and CSR6 at the same time; we will examine CSR1's RA first. Upon receipt, the RA is validated (hop limit = 255, no bogus flags, etc). Because this was a solicited RA, the host gleans the MAC address based on the source MAC of the Ethernet frame. The entry is marked as STALE, not REACH, since traffic is not yet flowing through these routers. Next, there is a chatty process that is used for default router selection. Since CSR1 is the only router known, it is currently the best, and a default route is installed on CSR5. The RA also carries the on-link prefix 2020:5:6::11/64 which will be used for autoconfiguration soon.

```
! CSR5
15:33:47.769: ICMPv6: Received R-Advert, Src=FE80::11, Dst=FF02::1
15:33:47.769: ICMPv6-ND: (GigabitEthernet2.556,FE80::11) Received RA
15:33:47.769: ICMPv6-ND: Validating ND packet options: valid
15:33:47.769: ICMPv6-ND: (GigabitEthernet2.556,FE80::11) Glean
15:33:47.769: ICMPv6-ND: (GigabitEthernet2.556,FE80::11) LLA 0011.1111.1111
15:33:47.769: ICMPv6-ND: (GigabitEthernet2.556,FE80::11) INCMP -> STALE
15:33:47.769: ICMPv6-ND: [default] New router interface context
created/GigabitEthernet2.556
15:33:47.769: ICMPv6-ND: [default] New router interface context
created/7F2323D66078
15:33:47.769: ICMPv6-ND: [default] inserted router
FE80::11/GigabitEthernet2.556
15:33:47.769: ICMPv6-ND: [default] Select default router
15:33:47.769: ICMPv6-ND: [default] best rank is C11
15:33:47.769: ICMPv6-ND: [default] router FE80::11/GigabitEthernet2.556 is
new best
15:33:47.769: ICMPv6-ND: [default] Selected new default router
15:33:47.769: ICMPv6-ND: [default] Install default to
FE80::11/GigabitEthernet2.556
15:33:47.769: ICMPv6-ND: Prefix : 2020:5:6:11::, Length: 64, Vld Lifetime:
2592000, Prf Lifetime: 604800, PI Flags: C0
15:33:47.769: ICMPv6-ND: Created OL-prefix root for 0
15:33:47.769: ICMPv6-ND: New on-link prefix 2020:5:6:11::/64 on
GigabitEthernet2.556/FE80::11, lifetime 2592000
```

CSR5 also receives an RA from CSR6. CSR6 is advertising the same on-link prefix, so CSR5 annotates that the prefix is supported by CSR6 as well since it already tracked this existing prefix. CSR6's MAC address is gleaned just like CSR1's. CSR5 continues to use CSR1 as its default-router since there are no preferences configured and CSR1 is the older entry.

```

! CSR5
15:33:47.769: ICMPv6: Received R-Advert, Src=FE80::6, Dst=FF02::1
15:33:47.769: ICMPv6-ND: (GigabitEthernet2.556,FE80::6) Received RA
15:33:47.769: ICMPv6-ND: Validating ND packet options: valid
15:33:47.769: ICMPv6-ND: (GigabitEthernet2.556,FE80::6) Glean
15:33:47.769: ICMPv6-ND: (GigabitEthernet2.556,FE80::6) LLA 0066.6666.6666
15:33:47.769: ICMPv6-ND: (GigabitEthernet2.556,FE80::6) INCMP -> STALE
15:33:47.771: ICMPv6-ND: [default] New router interface context
created/7F2323D66078
15:33:47.771: ICMPv6-ND: [default] inserted router
FE80::6/GigabitEthernet2.556
15:33:47.771: ICMPv6-ND: [default] Select default router
15:33:47.771: ICMPv6-ND: [default] best rank is C11
15:33:47.771: ICMPv6-ND: Prefix : 2020:5:6:11::, Length: 64, Vld Lifetime:
2592000, Prf Lifetime: 604800, PI Flags: C0
15:33:47.771: ICMPv6-ND: Update on-link prefix 2020:5:6:11::/64 on
GigabitEthernet2.556/FE80::6, lifetime 2592000

```

As a quick aside, we can verify the default route installed by CSR5 points to CSR1 as an ND route, and that CSR5 sees both routers. All of the detailed RA information is contained there as well.

```

R5#show ipv6 route ::/0
Routing entry for ::/0
  Known via "ND", distance 2, metric 0
  Route count is 1/1, share count 0
  Routing paths:
    FE80::11, GigabitEthernet2.556
    Last updated 00:13:41 ago

R5#show ipv6 routers detail
IPV6 ND Routers (table: default)
Router FE80::11 on GigabitEthernet2.556, last update 2 min
  Rank 0xC11 (elegant), Default Router
  Hops 64, Lifetime 1800 sec, AddrFlag=0, OtherFlag=0, MTU=1500
  HomeAgentFlag=0, Preference=Medium, trustlevel = 0
  Reachable time 0 (unspecified), Retransmit time 0 (unspecified)
  Prefix 2020:5:6:11::/64 onlink autoconfig
  Valid lifetime 2592000, preferred lifetime 604800
Router FE80::6 on GigabitEthernet2.556, last update 1 min
  Rank 0xC11 (elegant)
  Hops 64, Lifetime 1800 sec, AddrFlag=0, OtherFlag=0, MTU=1500
  HomeAgentFlag=0, Preference=Medium, trustlevel = 0
  Reachable time 0 (unspecified), Retransmit time 0 (unspecified)
  Prefix 2020:5:6:11::/64 onlink autoconfig
  Valid lifetime 2592000, preferred lifetime 604800

```

Mixed in with the debugs above is the DAD process for the global address derived from

autoconfiguration. For clarity, I grouped those debug messages below. The computed autoconfiguration address uses EUI-64 as well, which means both the LL and global addresses have the same host address. Thus, only a single solicited-node multicast group must be joined, shown below. After sending the NS, DAD waits for 1 second, as usual, then declares this global unicast address unique. The ND process finishes with an unsolicited NA for other hosts on the segment; recall that routers will ignore this by default and not use them for gleaned adjacencies unless configured.

```
! CSR5
15:33:47.769: IPv6-Addrmgr-ND: DAD request for 2020:5:6:11:255:55FF:FE55:5555
on GigabitEthernet2.556
15:33:47.769: ICMPv6-ND:
(GigabitEthernet2.556,2020:5:6:11:255:55FF:FE55:5555) Sending DAD NS [A23BB]
15:33:47.769: ICMPv6-ND: Autoconfiguring 2020:5:6:11:255:55FF:FE55:5555 on
GigabitEthernet2.556
15:33:47.771: ICMPv6-ND: %GigabitEthernet2.556: OK: IPv6 Address Autoconfig
2020:5:6:11::/64 eui-64, 2020:5:6:11:255:55FF:FE55:5555
2020:5:6:11:255:55FF:FE55:5555/64 is existing
15:33:47.773: ICMPv6: Sent N-Solicit, Src=::, Dst=FF02::1:FF55:5555
15:33:48.769: IPv6-Addrmgr-ND: DAD: 2020:5:6:11:255:55FF:FE55:5555 is unique.
15:33:48.769: ICMPv6-ND:
(GigabitEthernet2.556,2020:5:6:11:255:55FF:FE55:5555) Sending NA to FF02::1
15:33:48.770: ICMPv6: Sent N-Advert, Src=2020:5:6:11:255:55FF:FE55:5555,
Dst=FF02::1

R5#show ipv6 interface gig2.556 | section (group|unicast)_add
Global unicast address(es):
  2020:5:6:11:255:55FF:FE55:5555, subnet is 2020:5:6:11::/64 [EUI/CAL/PRE]
    valid lifetime 2591947 preferred lifetime 604747
Joined group address(es):
  FF02::1
  FF02::2
  FF02::1:FF55:5555
```

Continuing with our verification, CSR5 maintains these two routers as STALE entries until it actually sends traffic through them. Since CSR1 is the default gateway, sending traffic off-link will move CSR1 from STALE to REACH via the NUD process.

```
R5#show ipv6 neighbors gig2.556
```

IPv6 Address	Age	Link-layer Addr	State	Interface
FE80::6	24	0066.6666.6666	STALE	Gi2.556
FE80::11	24	0011.1111.1111	STALE	Gi2.556

Since CSR1 is the default gateway, sending traffic off-link will move CSR1 from STALE to REACH via the ND process. The age also resets to 0, since the “age” column represents the last time an ND conversation occurred with the given cache entry. Because CSR1 does not have the MAC address of CSR5 mapped to CSR5’s global address, it issues an NS message for it, to which CSR5 responds.

```
R5#ping 2020:0:11:12::11 repeat 1
Type escape sequence to abort.
Sending 1, 100-byte ICMP Echos to 2020:0:11:12::11, timeout is 2 seconds:
!
```

```
Success rate is 100 percent (1/1), round-trip min/avg/max = 10/10/10 ms
```

```
! CSR5
```

```
16:00:05.181: ICMPv6-ND:
```

```
(GigabitEthernet2.556,2020:5:6:11:255:55FF:FE55:5555) Received NS from
FE80::11
```

```
16:00:05.181: ICMPv6-ND: Validating ND packet options: valid
```

```
16:00:05.181: ICMPv6-ND:
```

```
(GigabitEthernet2.556,2020:5:6:11:255:55FF:FE55:5555) Sending NA to FE80::11
```

```
16:00:05.182: ICMPv6-ND: (GigabitEthernet2.556,FE80::11) STALE -> DELAY
```

```
16:00:05.185: ICMPv6-ND: (GigabitEthernet2.556,FE80::11) ULP indication
```

```
16:00:05.185: ICMPv6-ND: (GigabitEthernet2.556,FE80::11) DELAY -> REACH
```

```
R5#show ipv6 neighbors
```

IPv6 Address	Age	Link-layer Addr	State	Interface
FE80::6	27	0066.6666.6666	STALE	Gi2.556
FE80::11	0	0011.1111.1111	REACH	Gi2.556

If CSR5 wants to send traffic to the segment between CSR6 and CSR3, it still sends traffic to CSR1 initially. This is suboptimal and is handled with redirect messages. First, we verify that CSR1 is actually routing to CSR6 via the host LAN to which CSR5 is joined. For clarity, many of the basic NA/NS messages are stripped from the debugs since that process has been examined thoroughly.

```
R1#show ipv6 route 2020:0:3:6::/64
```

```
Routing entry for 2020:0:3:6::/64
```

```
Known via "isis 2020", distance 115, metric 20, type level-2
```

```
Route count is 1/1, share count 0
```

```
Routing paths:
```

```
FE80::6, GigabitEthernet2.556
```

```
Last updated 00:32:25 ago
```

When CSR5 sends packets to this destination, they first go to CSR1. CSR1 issues a redirect message to CSR5 inform it of the better gateway. The target address is carried in the payload and identifies CSR6's LL address as the next-hop. CSR5 does not appear to honor the redirect, but I wanted to show the mechanism.

```
! CSR1
```

```
ICMPv6-ND: (GigabitEthernet2.556,2020:0:3:6::6) Sending REDIRECT, target
FE80::6
```

```
ICMPv6: Sent Redirect, Src=FE80::11, Dst=2020:5:6:11:255:55FF:FE55:5555
```

```
! CSR5
```

```
ICMPv6: Received Redirect, Src=FE80::11, Dst=2020:5:6:11:255:55FF:FE55:5555
```

Another interesting characteristic of the ND cache is the PROBE state. This is NUD in action, sending targeted (unicast) NS messages to verify reachability. The reason CSR5 performs this towards CSR6 is because the initial packet is asymmetrically routed. CSR5 sent it to CSR1, but the reply came from CSR6. CSR5 cannot guarantee that two-way reachability exists with CSR6 despite knowing it's MAC address from the RA. The cache entry transitions from the PROBE state once the solicited NA is received from the neighbor.

```
! CSR5
ICMPv6: Received echo reply, Src=2020:0:3:6::6,
Dst=2020:5:6:11:255:55FF:FE55:5555
ICMPv6-ND: (GigabitEthernet2.556,FE80::6) DELAY -> PROBE
ICMPv6-ND: (GigabitEthernet2.556,FE80::6) Sending NS
ICMPv6: Sent N-Solicit, Src=FE80::255:55FF:FE55:5555, Dst=FE80::6
ICMPv6: Received N-Advert, Src=FE80::6, Dst=FE80::255:55FF:FE55:5555
ICMPv6-ND: (GigabitEthernet2.556,FE80::6) Received NA from FE80::6
ICMPv6-ND: Validating ND packet options: valid
ICMPv6-ND: Packet contains no options
ICMPv6-ND: (GigabitEthernet2.556,FE80::6) PROBE -> REACH
```

We will briefly examine anycast addressing on the LAN as well. Earlier, we saw how DAD can determine if there are duplicate addresses on a LAN; clearly this does not make sense for anycast gateways where the same IPv6 address may exist on the LAN. In XE, we can append the “anycast” keyword to an IPv6 address that essentially disables DAD for the address. In XE and XR, we can disable DAD for the entire interface, which affects all prefixes. We will use both methods on CSR1 and CSR6 while adding a new anycast IPv6 address to the subnet. We will ensure this new address is within the same on-link prefix so the composition of the RA message need not change. The method used on CSR1 is the only way to configure anycast addresses on XR.

```
! CSR1
interface GigabitEthernet2.556
  ipv6 address 2020:5:6:11::611/64
  ipv6 nd dad attempts 0

! CSR6
interface GigabitEthernet2.556
  ipv6 address 2020:5:6:11::611/64 anycast

R1#show ipv6 interface gig2.556 | section Global_uni|DAD
Global unicast address(es):
  2020:5:6:11::11, subnet is 2020:5:6:11::/64
  2020:5:6:11::611, subnet is 2020:5:6:11::/64
  ND DAD is disabled

R6#show ipv6 interface gig2.556 | section Global_uni|DAD
```

```
Global unicast address(es):
  2020:5:6:11::6, subnet is 2020:5:6:11::/64
  2020:5:6:11::611, subnet is 2020:5:6:11::/64 [ANY]
ND DAD is enabled, number of DAD attempts: 1
```

Debugging ND on CSR1 shows that the DAD software process is invoked for all three addresses, but immediately returns that the addresses are unique without actually doing anything. The timestamps prove this.

```
! CSR1
16:40:22.787: IPv6-Addrmgr-ND: DAD request for FE80::11 on
GigabitEthernet2.556
16:40:22.787: IPv6-Addrmgr-ND: DAD: FE80::11 is unique.
16:40:22.788: IPv6-Addrmgr-ND: DAD request for 2020:5:6:11::11 on
GigabitEthernet2.556
16:40:22.788: IPv6-Addrmgr-ND: DAD: 2020:5:6:11::11 is unique.
16:40:22.788: IPv6-Addrmgr-ND: DAD request for 2020:5:6:11::611 on
GigabitEthernet2.556
16:40:22.788: IPv6-Addrmgr-ND: DAD: 2020:5:6:11::611 is unique.
```

The output is similar on CSR6, but only for the anycast address. The other addresses undergo the normal DAD process.

```
! CSR6
16:42:55.742: IPv6-Addrmgr-ND: DAD request for FE80::6 on
GigabitEthernet2.556
16:42:55.742: ICMPv6-ND: Delay DAD for FE80::6 on GigabitEthernet2.556 by 200
msec
16:42:55.941: ICMPv6-ND: (GigabitEthernet2.556,FE80::6) Sending DAD NS
[D9C50]
16:42:56.942: IPv6-Addrmgr-ND: DAD: FE80::6 is unique.
16:42:56.942: ICMPv6-ND: (GigabitEthernet2.556,FE80::6) Sending NA to FF02::1
16:42:56.942: ICMPv6-ND: (GigabitEthernet2.556) L3 came up
16:42:56.942: IPv6-Addrmgr-ND: DAD request for 2020:5:6:11::6 on
GigabitEthernet2.556
16:42:56.942: ICMPv6-ND: (GigabitEthernet2.556,2020:5:6:11::6) Sending DAD NS
[D9C50]
16:42:56.942: IPv6-Addrmgr-ND: DAD request for 2020:5:6:11::611 on
GigabitEthernet2.556
16:42:56.942: IPv6-Addrmgr-ND: DAD: 2020:5:6:11::611 is unique.
16:42:57.942: IPv6-Addrmgr-ND: DAD: 2020:5:6:11::6 is unique.
```

There are several other important RA options as well. We can have multiple on-link prefixes but only offer a subset of them to clients for autoconfiguration. For example, CSR2 and CSR3 are both routers serving network access to CSR4. They have a global unicast prefix as well as a unique-local prefix for intra-site routing. The client should typically not use ULA for autoconfiguration if its wants Internet reachability. Both CSR2 and CSR3 can suppress this prefix from their RA messages so that CSR4 is not

aware of its existence. CSR2 and CSR3 have nearly identical configurations, not counting the host addresses, so only CSR2 is shown. We can verify the prefixes advertised by an IPv6-enabled router interface as well; the ULA prefix has the 'N' flag to indicate it is not advertised, while the global unicast address is. CSR4 only sees the global prefix as a result.

```
! CSR2
interface GigabitEthernet2.542
  ipv6 address FE80::12 link-local
  ipv6 address 2020:3:4:12::12/64
  ipv6 address FD00:3:4:12::12/64
  ipv6 nd prefix FD00:3:4:12::/64 no-advertise

R2#show ipv6 interface gig2.542 prefix
IPv6 Prefix Advertisements GigabitEthernet2.542
Codes for 1st column:
  A - Address, P - Prefix-Advertisement, O - Pool
  U - Per-user prefix
Codes for 2nd column and above:
  D - Default
  N - Not advertised, C - Calendar
```

```
PD default [LA] Valid lifetime 2592000, preferred lifetime 604800
AD 2020:3:4:12::/64 [LA] Valid lifetime 2592000, preferred lifetime 604800
PAN FD00:3:4:12::/64 [LA] Valid lifetime 2592000, preferred lifetime 604800
```

CSR4 only sees the global prefix as a result and computes its EUI-64 address accordingly.

```
R4#show ipv6 routers | include ^Router|Prefix
Router FE80::12 on GigabitEthernet2.542, last update 1 min
  Prefix 2020:3:4:12::/64 onlink autoconfig
Router FE80::3 on GigabitEthernet2.542, last update 1 min
  Prefix 2020:3:4:12::/64 onlink autoconfig

R4#show ipv6 interface gig2.542 | section Global_uni
Global unicast address(es):
  2020:3:4:12:244:44FF:FE44:4444, subnet is 2020:3:4:12::/64 [EUI/CAL/PRE]
    valid lifetime 2591882 preferred lifetime 604682
```

We can adjust the unsolicited RA interval and the corresponding lifetime as well. The lifetime is relevant for how long the default routing can be considered valid, not the RA itself. Debugging on CSR4, we can see that every 20 seconds, an RA from CSR2 is received (green). Every 30 seconds, an RA from CSR3 is received (yellow). For debugging brevity, we look at the ICMPv6 packet exchange without examining the ND process. To prevent RA synchronization, the timers are randomized within a range; the values configured above are maximum values. The minimum is 75% of the maximum and the actual timer used is a random number in that range. The minimum can be adjusted as well, but 75% is a good value. Thus, CSR2's RAs are sent at a rate of 15 – 20 seconds while CSR3's RAs are sent at a rate of 22.5 – 30 seconds.

```

! CSR2
interface GigabitEthernet2.542
  ipv6 nd ra lifetime 200
  ipv6 nd ra interval 20

! CSR3
interface GigabitEthernet2.542
  ipv6 nd ra lifetime 300
  ipv6 nd ra interval 30

! CSR4
16:57:59.614: ICMPv6: Received R-Advert, Src=FE80::3, Dst=FF02::1
16:58:09.105: ICMPv6: Received R-Advert, Src=FE80::12, Dst=FF02::1
16:58:25.823: ICMPv6: Received R-Advert, Src=FE80::3, Dst=FF02::1
16:58:27.625: ICMPv6: Received R-Advert, Src=FE80::12, Dst=FF02::1
16:58:44.403: ICMPv6: Received R-Advert, Src=FE80::12, Dst=FF02::1
16:58:55.329: ICMPv6: Received R-Advert, Src=FE80::3, Dst=FF02::1
16:59:04.014: ICMPv6: Received R-Advert, Src=FE80::12, Dst=FF02::1

```

The Default Router Preference (DRP) feature provides basic “low, medium, high” priorities as a tiebreaker for selecting a default router. On the LAN with CSR4, both CSR2 and CSR3 are originating RAs. CSR2 is configured with a priority of “high” while CSR3 uses the default priority of “medium”. We can confirm this on both routers by checking the IPv6 interface details.

```

! CSR2
interface GigabitEthernet2.542
  ipv6 nd router-preference High

R2#show ipv6 interface gig2.542 | include preference
ND advertised default router preference is High

R3#show ipv6 interface gig2.542 | include preference
ND advertised default router preference is Medium

```

Debugging IPv6 ND on CSR4, it receives unsolicited RAs from both CSR2 and CSR3 periodically. CSR4 can see this DRP value and always select CSR2 when it is available. Notice that the RA lifetimes are shown in this output as well, which are different than the prefix lifetimes. The RA lifetime measures how long this router is useful as a default router; prefix lifetimes are examined next.

```

R4#show ipv6 routers detail
IPv6 ND Routers (table: default)
Router FE80::12 on GigabitEthernet2.542, last update 0 min
  Rank 0xC19 (elegant), Default Router
  Hops 64, Lifetime 200 sec, AddrFlag=0, OtherFlag=0, MTU=1500
  HomeAgentFlag=0, Preference=High, trustlevel = 0
  Reachable time 0 (unspecified), Retransmit time 0 (unspecified)

```



```

Prefix 2020:3:4:12::/64 onlink autoconfig
Valid lifetime 2592000, preferred lifetime 604800
Router FE80::3 on GigabitEthernet2.542, last update 0 min
Rank 0xC11 (eligible)
Hops 64, Lifetime 300 sec, AddrFlag=0, OtherFlag=1, MTU=1500
HomeAgentFlag=0, Preference=Medium, trustlevel = 0
Reachable time 0 (unspecified), Retransmit time 0 (unspecified)
Prefix 2020:3:4:12::/64 onlink autoconfig
Valid lifetime 2592000, preferred lifetime 604800

```

The valid and preferred lifetimes are used to denote how long a prefix can be used or preferred. The preferred lifetime cannot exceed the valid lifetime, and these values can be tuned per-prefix. However, all routers on the segment should agree on the values or else the router will display an error message showing the differences.

```

! CSR2
interface GigabitEthernet2.542
  ipv6 nd prefix 2020:3:4:12::/64 200 180

! CSR2
%IPV6_ND-3-CONFLICT: Router FE80::3 on GigabitEthernet2.542 conflicting ND
setting prefix 2020:3:4:12::/64 valid lifetime, difference 2591800 seconds

! CSR3
%IPV6_ND-3-CONFLICT: Router FE80::12 on GigabitEthernet2.542 conflicting ND
setting prefix 2020:3:4:12::/64 valid lifetime, difference 2591800 seconds

```

For consistency, we configure these settings on CSR3 as well (not shown), then verify it on both routers and the client (CSR4).

```

R2#show ipv6 interface gig2.542 prefix | include 2020
PA 2020:3:4:12::/64 [LA] Valid lifetime 200, preferred lifetime 180

R3#show ipv6 interface gig2.542 prefix | include 2020
PA 2020:3:4:12::/64 [LA] Valid lifetime 200, preferred lifetime 180

R4#sh ipv6 router | include ^Router|Prefix|Valid
Router FE80::12 on GigabitEthernet2.542, last update 0 min
  Prefix 2020:3:4:12::/64 onlink autoconfig
  Valid lifetime 200, preferred lifetime 180
Router FE80::3 on GigabitEthernet2.542, last update 0 min
  Prefix 2020:3:4:12::/64 onlink autoconfig
  Valid lifetime 200, preferred lifetime 180

```

There are several other options we can enable per-prefix as well, such as whether the prefix can be used for autoconfiguration, whether it is on-link, etc. We will configure a bogus prefix on CSR2 only which is not on-link and cannot be used for autoconfiguration. Notice that we do not need to configure an IPv6 address on CSR2 for this prefix.

```
! CSR2
interface GigabitEthernet2.542
  ipv6 nd prefix 2020:FFFF:FFFF:FFFF::/64 infinite infinite no-autoconfig no-
onlink
```

```
R2#show ipv6 interface gig2.542 prefix | include FFFF
P 2020:FFFF:FFFF:FFFF::/64 [] Valid lifetime infinite, preferred lifetime
infinite
```

CSR4 learns the prefix but it cannot use it for much, at present. There is no ND route for it (no connected, on-link route) and it cannot be used for auto-configuration.

```
R4#show ipv6 routers default
Router FE80::12 on GigabitEthernet2.542, last update 0 min
  Hops 64, Lifetime 200 sec, AddrFlag=0, OtherFlag=0, MTU=1500
  HomeAgentFlag=0, Preference=High, trustlevel = 0
  Reachable time 0 (unspecified), Retransmit time 0 (unspecified)
  Prefix 2020:3:4:12::/64 onlink autoconfig
    Valid lifetime 200, preferred lifetime 180
  Prefix 2020:FFFF:FFFF:FFFF::/64
    Valid lifetime infinite, preferred lifetime infinite
```

```
R4#show ipv6 route nd | begin ^ND
ND  ::/0 [2/0]
    via FE80::12, GigabitEthernet2.542
NDp 2020:3:4:12::/64 [2/0]
    via GigabitEthernet2.542, directly connected
```

Next, we will examine DHCPv6 for stateless autoconfiguration. DHCPv6's role in this design is to issue non-address related configuration, such as DNS servers, domain names, SNTP servers, etc. This information doesn't need to be bound to a host and can be handed out freely to SLAAC clients upon request. A router signals that it is capable of providing DHCPv6 "other" configurations by setting the 'O' flag in the RA, described earlier. CSR3 will be the DHCPv6 server and will notify CSR4 about some non-address configurations. CSR4 doesn't have to use CSR3 as a default gateway to use this service, either. We verify that CSR4 can see this 'O' flag set in the RA from CSR3 but not CSR2.

```
! CSR3
ipv6 dhcp pool DHCPV6_POOL
  dns-server 2020::BEEF
  domain-name lab.local
  sntp address 2001:0:3:7::3

interface GigabitEthernet2.542
  ipv6 nd other-config-flag
  ipv6 dhcp server DHCPV6_POOL
```

```
R4#show ipv6 routers | include Router|Other
Router FE80::12 on GigabitEthernet2.542, last update 0 min
  Hops 64, Lifetime 200 sec, AddrFlag=0, OtherFlag=0, MTU=1500
Router FE80::3 on GigabitEthernet2.542, last update 0 min
  Hops 64, Lifetime 300 sec, AddrFlag=0, OtherFlag=1, MTU=1500
```

When CSR4 receives a solicited RA from CSR3 after having sent an RS (assume a link-up event on CSR4), it will notice the 'O' flag and invoke the DHCPv6 process to send traffic to the DHCPv6 servers and relay agents multicast group (FF02::1:2). Configuring the DHCPv6 pool on CSR3 causes it to listen to this group as a DHCPv6 server. The other group, FF05::1:3 is for DHCPv6 servers only, not relay agents, but serves the same function except is routable based on the scope bits.

```
R3#show ipv6 interface gig2.542 | section group_add
Joined group address(es):
  FF02::1
  FF02::2
  FF02::1:2
  FF02::1:FF00:3
  FF05::1:3
```

```
R4#debug ipv6 nd
R4#debug ipv6 dhcp detail
17:28:48.238: ICMPv6-ND: (GigabitEthernet2.542) Sending RS
17:28:48.241: ICMPv6-ND: (GigabitEthernet2.542,FE80::3) Received RA
17:28:48.241: ICMPv6-ND: Validating ND packet options: valid
[snip, normal RA processing]
17:28:48.242: ICMPv6-ND: O-bit set; checking DHCP
17:28:48.242: IPv6 DHCP: detailed packet contents
17:28:48.242:   src FE80::244:44FF:FE44:4444
17:28:48.242:   dst FF02::1:2 (GigabitEthernet2.542)
17:28:48.242:   type INFORMATION-REQUEST(11), xid 13468421
17:28:48.242:   option ELAPSED-TIME(8), len 2
17:28:48.242:     elapsed-time 0
17:28:48.242:   option CLIENTID(1), len 10
17:28:48.242:     00030001001E4980B400
17:28:48.242:   option ORO(6), len 4
17:28:48.242:   DNS-SERVERS, DOMAIN-LIST
17:28:48.242: IPv6 DHCP: Sending INFORMATION-REQUEST to FF02::1:2 on
GigabitEthernet2.542
17:28:48.243: IPv6 DHCP: DHCPv6 changes state from IDLE to INFORMATION-
REQUEST (STATELESS) on GigabitEthernet2.542
```

CSR3 replies to the DHCP information request with the DNS servers and domain-list. CSR4 did not ask for the SNTP servers, as seen above, so the DHCPv6 server did not respond with it. The response is a unicast reply to the clients LL address; CSR4 then saves the new DNS and domain information.

```
! CSR4
17:28:48.245: IPv6 DHCP: Received REPLY message
```

```

17:28:48.245: IPv6 DHCP: Received REPLY from FE80::3 on GigabitEthernet2.542
17:28:48.245: IPv6 DHCP: detailed packet contents
17:28:48.245:   src FE80::3 (GigabitEthernet2.542)
17:28:48.245:   dst FE80::244:44FF:FE44:4444 (GigabitEthernet2.542)
17:28:48.245:   type REPLY(7), xid 13468421
17:28:48.245:   option SERVERID(2), len 10
17:28:48.245:     00030001001EE5A8FF00
17:28:48.245:   option CLIENTID(1), len 10
17:28:48.245:     00030001001E4980B400
17:28:48.245:   option DNS-SERVERS(23), len 16
17:28:48.245:     2020::BEEF
17:28:48.245:   option DOMAIN-LIST(24), len 11
17:28:48.245:     lab.local
17:28:48.245: IPv6 DHCP: Adding server FE80::3
17:28:48.245: IPv6 DHCP: Processing options
17:28:48.245: IPv6 DHCP: Configuring DNS server 2020::BEEF
17:28:48.245: IPv6 DHCP: Configuring domain name lab.local
17:28:48.245: IPv6 DHCP: DHCPv6 changes state from INFORMATION-REQUEST to
IDLE (REPLY_RECEIVED) on GigabitEthernet2.542

```

```

R4#show hosts
Name lookup view: Global
Default domain is not set
Domain list: lab.local
[snip]

```

We can also test the stateful DHCPv6 behavior. This allows a DHCPv6 server to hand out IPv6 addresses from a specific prefix (pool) as in DHCPv4. We can extend our DHCPv6 pool to add a prefix, then offer it to CSR7 on a new interface. The 'M' and 'O' flags are set on this interface which allows CSR7 to get all of its information, both addressing and "other", from the DHCPv6 server. We confirm that CSR7 sees both of these flags in the RA from CSR3.

```

! CSR3
ipv6 dhcp pool DHCPV6_POOL
  address prefix 2020:0:3:7::/64

interface GigabitEthernet2.537
  ipv6 dhcp server DHCPV6_POOL
  ipv6 nd managed-config-flag
  ipv6 nd other-config-flag

R7#show ipv6 routers detail
IPV6 ND Routers (table: default)
Router FE80::3 on GigabitEthernet2.537, last update 0 min
  Rank 0xA11 (elegant), Default Router
  Hops 64, Lifetime 1800 sec, AddrFlag=1, OtherFlag=1, MTU=1500
  HomeAgentFlag=0, Preference=Medium, trustlevel = 0
  Reachable time 0 (unspecified), Retransmit time 0 (unspecified)

```

```
Prefix 2020:0:3:7::/64 onlink autoconfig
Valid lifetime 2592000, preferred lifetime 604800
```

Unfortunately, XE does not appear to support stateful DHCPv6 client at this time. “ipv6 address dhcp” is not a supported option, but we will show the rest of CSR7’s configuration for completeness. With IPv6 enabled, a LL address can be obtained automatically. We also tell ND to automatically configure the prefix and default-route based on the address received, which would normally come from DHCPv6 in a functional design.

```
R7(config-subif)#ipv6 address ?
WORD                General prefix name
X:X:X:X::X          IPv6 link-local address
X:X:X:X::X/<0-128>   IPv6 prefix
autoconfig           Obtain address using autoconfiguration

! CSR7
interface GigabitEthernet2.537
  ipv6 enable
  ipv6 nd autoconfig prefix
  ipv6 nd autoconfig default-route
```

Additional Reading – Reference configurations “ipv6-nd”

1.2 Broadband Aggregation (BBA)

BBA is a sizable topics and only the basic concepts are covered here. Below are some example BBA architectures and definitions:

1. Direct connections from DSLAM/ANs to BNGs.
2. DSLAM/AN to an aggregate Ethernet switch, then to BNG, in a hub-spoke criss-form; classic design.
3. DSLAM/AN to an aggregate Ethernet switch, all of which are tied into a ring where the BNGs also reside.

BNG - Broadband Network Gateway. Sits between the DSLAM, or aggregator of DSL connections, and the IP network of the network service provider (NSP). It may encompass the BRAS, but the two are not the same. Some architectures may introduce dual BNGs, for example dedicating one to video services and another to all other. In dual-BNG scenarios, BOTH BNGs do not have to meet all requirements, as long as the union of the BNG capabilities does.

BRAS - Broadband remote access server. This is the aggregation point between the NSP and the access network, typically using IP. It is also an injection point for policies, such as IP QoS.

BBA - Broadband aggregation. This commonly relies on L2TP. The main component of L2TP is a reliable control channel that is responsible for session setup, negotiation, and teardown, and a forwarding plane that adds negotiated session IDs and forwards traffic. Layer 2 circuits terminate in a device called an

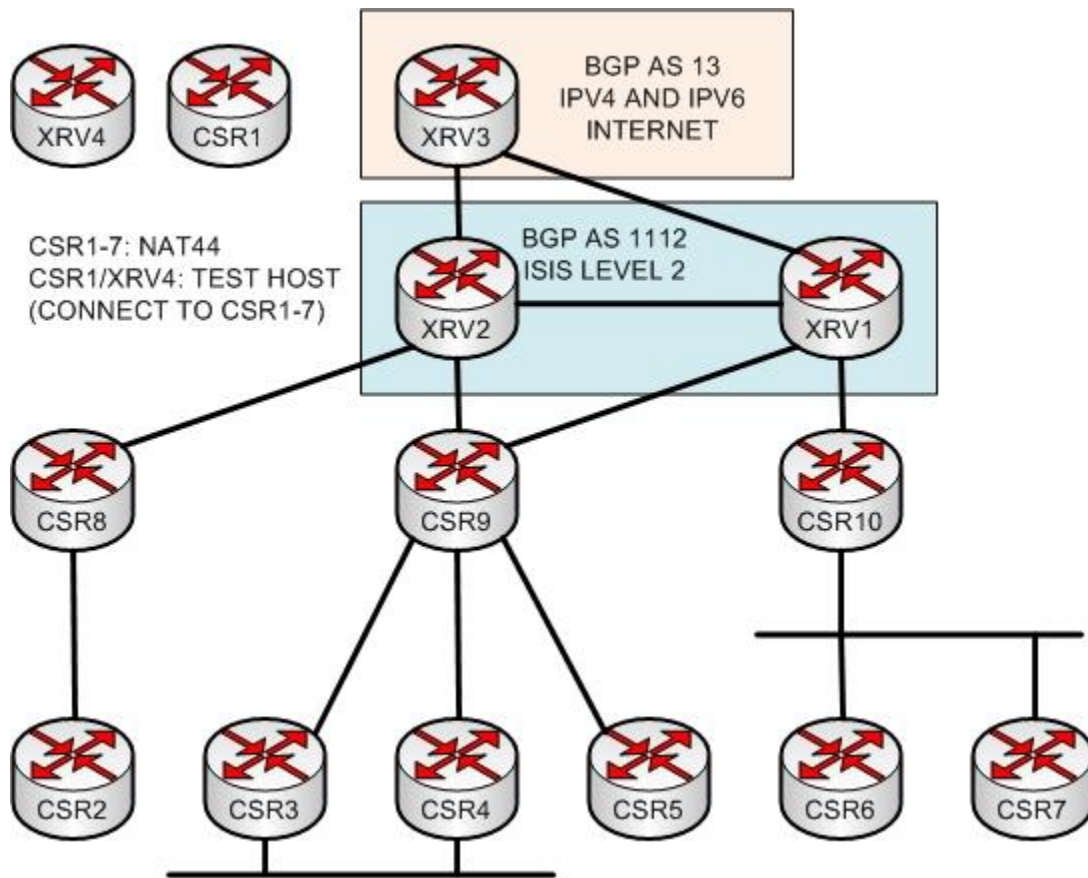
L2TP access concentrator (LAC), and the PPP sessions terminate in an L2TP network server (LNS). The LNS authenticates the user and is the endpoint for PPP negotiation. The LAC is closer to the customer than the LNS, and is the “downstream tail end” of the L2TP tunnel, whereas the LNS is the “upstream head end”. Thus, the PPPoE frames are tunneled inside L2TP to the LNS. The LAC connects to the LNS using a LAN or a WAN connection, and L2TP rides over the top of this. The LAC directs the subscriber session into L2TP tunnels based on the domain of each session.

1.2.1 PPP over Ethernet (PPPoE) technology

PPPoE is commonly used for BBA because it offers all of the benefits of PPP (authentication, directional call control, compression, encryption, etc) but can use Ethernet at layer 2 as transport. Many callers can “dial in” to the BNG on a shared segment and gain network connectivity in this way. Unlike Ethernet, clients cannot talk to one another, and this method works well with the N:1 VLAN paradigm discussed later, which further restricts peer-to-peer connectivity at layer 2. XR supports PPPoE server only, but XRV does not appear to support PPPoE at all. The CSR1000v supports both roles, but some features are unsupported. Currently, I have discovered that Microsoft Point to Point Encryption (MPPE) and compression (stac, predictor) are not supported. The CSR generates a log message when you try to configure these features to indicate that it’s virtual-access interface is incapable of supporting them.

```
%FMANRP_ESS-4-FULLVAI: Session creation failed due to Full Virtual-Access  
Interfaces not being supported. Check that all applied Virtual-Template and  
RADIUS features support Virtual-Access sub-interfaces. swidb= 0x7F1E9054C508,  
ifnum= 19
```

To represent a PPPoE-based BBA architecture that is somewhat realistic, we will use a hierarchical access/aggregation network (similar network used for NAT444, NAT464, etc). CSR8, CSR9, and CSR10 are the PPPoE servers while CSR2 through CSR7 are the PPPoE clients. The clients are like CPE routers in residential areas while the PPPoE servers are the BNGs. XRV1 and XRV2 are Internet gateways, and XRV3 is the Internet. Because XR does not support IPv6 SLAAC, CSR1 has several VRFs to similar a client behind each CPE router. Basic NAT44 is used to translate private CPE addressing to global addressing at the CPE; hierarchical NAT is discussed in a dedicated chapter. NAT is not the focus of this lab so very basic NAT techniques are used, otherwise the PPPoE design would be very unrealistic with IGP running everywhere.



First, we will configure PPPoE between CSR8 and CSR2; as the access concentrator (AC), CSR8 only has one client. The AC uses a virtual-template interface and the client uses a dialer interface. We will negotiate the client IP address using IP control protocol (IPCP) which is a function of PPP. The addresses issued to clients will be handed out from a local pool (not DHCP). We can also apply limits to the number of client sessions the server will accept; in this case, we say there can be only one per-MAC and per-VLAN. This prevents CSR2 from dialing into CSR8 multiple times. We must adjust the MTU on the PPPoE virtual interfaces to be 8 bytes less than the supported layer 3 MTU since PPPoE adds 8 bytes of encapsulation. To support IPv6, we create two pools. The first is meant to service the transit links (the configuration below has a tricky error; will be fixed later) and the second is a way to “delegate” a downstream IPv6 prefix for the client to use. In this way, DHCPv6 can offer CSR2 a LAN-side public prefix so that CSR2 doesn’t have to manually configure it. This prefix is exchanged using IPv6 ND, so we must unsuppress the RA advertisements on the BNG.

```
! CSR8
bba-group pppoe PPPOE_28
 virtual-template 28
  sessions per-mac limit 1
  sessions per-vlan limit 1
  ipv6 address FE80::8 link-local

ip local pool PPPOE_POOL_V4 209.2.8.100 209.2.8.149
```

```

ipv6 local pool PPPOE_POOL_V6 2001:10:2:80::/60 64
ipv6 local pool PD_POOL_V6 2001:192:168:80::/60 64

```

```

interface Virtual-Template28
  mtu 1492
  ip unnumbered Loopback28
  peer default ip address pool PPPOE_POOL_V4
  peer default ipv6 pool PPPOE_POOL_V6
  ipv6 enable
  no ipv6 nd ra suppress
  ipv6 nd ra lifetime 60
  ipv6 nd ra interval 10 5
  ipv6 dhcp server DHCP_POOL_V6

```

The client configuration is similar to the BNG. Dialers are not PPP-encapsulated by default, so we must specify this as well. NAT44 is enabled but it does not affect IPv6 traffic at all. We assign a dial-pool number which is applied at the interface level from which the session initiation occurs. We instruct the client to install a default route to the IPCP negotiated address, and for IPv6 we likewise install a default route to the BNG router discovered through IPv6 ND. The IPv6 prefix-delegation allows CSR2 to learn a prefix from the IPv6 local pool defined on CSR8 to use for its LAN segment.

```

! CSR2
interface Dialer28
  mtu 1492
  ip address negotiated
  ip nat outside
  encapsulation ppp
  dialer pool 28
  dialer idle-timeout 0
  dialer persistent
  ipv6 address autoconfig default
  ipv6 dhcp client pd PPPOE_ISP_PREFIX
  ppp ipcp route default

interface GigabitEthernet2.528
  pppoe-client dial-pool-number 28

```

First, we will enable PPP and PPPoE debugging on the client and server to watch the sequence of events. The PPP debugging is not specific to PPPoE at all and shows many low-level details. The PPPoE discovery packets seen below are described now. For clarity, the packets from the debug are shown in-line with the descriptions below.

```

! CSR2 and CSR8
debug pppoe events
debug pppoe packets
debug ppp negotiation

```


1. PPPoE Active Discovery Initiation (PADI): Sent to the Ethernet broadcast address (ffff.ffff.ffff) with a source MAC of the client. This is used to discover all ACs on the segment. Later, we will examine service-names, and only ACs with a matching service name should respond. This is similar to a DHCPDISCOVER and the PADI is sent from CSR2 to CSR8 as shown below. The destination MACs are shown in pink with source MACs in green. Notice that the PPPoE discovery ethertype is 0x8863, which is non-IP traffic (cyan, only shown once). Upon receipt, CSR8's debug shows a nice summary of the PADI header information to include remote (R) and local (L) MAC addresses, VLAN ID, and interface. The "I" before the word PADI means incoming. The server also annotates that the client's service tag is null, so no special treatment is being requested, and any server may respond. The code for PADI is 0x09 (codes are discussed later).

```
! CSR2
Sending PADI: Interface = GigabitEthernet2.528
pppoe_send_padi:
contiguous pak, size 64
FF FF FF FF FF FF 00 50 56 A9 BE 8A 81 00 0D C8
88 63 11 09 00 00 00 10 01 01 00 00 01 03 00 08
D2 00 00 06 00 00 0F 20 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

! CSR8
PPPoE 0: I PADI R:0050.56a9.be8a L:ffff.ffff.ffff 3528 Gi2.528
contiguous pak, size 40
FF FF FF FF FF FF 00 50 56 A9 BE 8A 81 00 0D C8
88 63 11 09 00 00 00 10 01 01 00 00 01 03 00 08
D2 00 00 06 00 00 0F 20
Service tag: NULL Tag
```

2. PAD Offer (PADO): The response from the AC is a unicast Ethernet frame back to the source and is sent from ACs that are capable of servicing the client. This is similar to a DHCP OFFER. CSR8 originates this message and sends it to CSR2 as a unicast frame, again with a null service tag. The PADO is outbound as denoted by the "O". CSR2 receives this PADO; we can tell because the "I" before PADO means incoming and the local MAC address is CSR2's Ethernet interface (destination of the frame). The PADO code is 0x07.

```
! CSR8
PPPoE 0: O PADO, R:0050.56a9.fb1c L:0050.56a9.be8a 3528 Gi2.528

Service tag: NULL Tag
```

```
contiguous pak, size 66
00 50 56 A9 BE 8A 00 50 56 A9 FB 1C 81 00 0D C8
88 63 11 07 00 00 00 2A 01 01 00 00 01 03 00 08
D2 00 00 06 00 00 0F 20 01 02 00 02 52 38 01 04
00 10 97 58 88 C2 01 8F 8A 96 23 D2 F7 0E E3 54
```

F4 D5

! CSR2

PPPoE 0: I PADO R:0050.56a9.fb1c L:0050.56a9.be8a 3528 Gi2.528

contiguous pak, size 66

```
00 50 56 A9 BE 8A 00 50 56 A9 FB 1C 81 00 0D C8
88 63 11 07 00 00 00 2A 01 01 00 00 01 03 00 08
D2 00 00 06 00 00 0F 20 01 02 00 02 52 38 01 04
00 10 97 58 88 C2 01 8F 8A 96 23 D2 F7 0E E3 54
F4 D5
```

3. PAD Request (PADR): The client selects one of the ACs and sends a unicast Ethernet frame to it requesting to connect. This is like a DHCPREQUEST. For some reason, the output on CSR2 is inconsistent with the format seen with the PADI and PADO thus far. It simply says the PADR was sent but doesn't parse any details for us. We can easily pick out the MAC addresses and see that this is destined as a unicast frame to CSR8. When CSR8 receives it, it prepares an encapsulation string for the session. This includes the full layer 2 encapsulation of the PPPoE data frames; notice the ethertype is 0x8864 now, also non-IP, and is used for PPPoE bearer traffic. first two bytes of the PPPoE header represent this new ethertype (green). The first 4 bits in the next byte represents version and the second 4 bits represents type (cyan, both must be 1). This is the third byte of the PPPoE header. The next byte represents a code used for discovery and session stages, and is zero here (pink). The next 2 bytes (0x0013) represents the session ID, which is decimal 19 in this case (grey). The last 2 bytes represent the length of the packet at layer 3, which varies per packet and is shown as zero in the debug (red). The PADR code is 0x19.

! CSR2

OUT PADR from PPPoE Session

contiguous pak, size 66

```
00 50 56 A9 FB 1C 00 50 56 A9 BE 8A 81 00 0D C8
88 63 11 19 00 00 00 2A 01 03 00 08 D2 00 00 06
00 00 0F 20 01 02 00 02 52 38 01 04 00 10 97 58
88 C2 01 8F 8A 96 23 D2 F7 0E E3 54 F4 D5 01 01
00 00
```

! CSR8

PPPoE 0: I PADR R:0050.56a9.be8a L:0050.56a9.fb1c 3528 Gi2.528

contiguous pak, size 66

```
00 50 56 A9 FB 1C 00 50 56 A9 BE 8A 81 00 0D C8
88 63 11 19 00 00 00 2A 01 03 00 08 D2 00 00 06
00 00 0F 20 01 02 00 02 52 38 01 04 00 10 97 58
88 C2 01 8F 8A 96 23 D2 F7 0E E3 54 F4 D5 01 01
00 00
```

Service tag: NULL Tag

PPPoE : encap string prepared

contiguous pak, size 24

```
00 50 56 A9 BE 8A 00 50 56 A9 FB 1C 81 00 0D C8
88 64 11 00 00 13 00 00
```

4. PAD Session (PADS): The server acknowledges and accepts the offer which completes the PPPoE session. This packet also contains the session ID. This is like a DHCPACK. Now that the session ID has been established, the debug messages pertinent to this session will include the number (in decimal) within the debug logs. CSR8 sends the PADS back to CSR2 with this number embedded in the PPPoE header; previously it was zero for the initial PAD exchanges. The PPPoE header is shown in yellow; we can see the layer 3 packet length is 0x2A (42 in decimal). CSR2 receives this PADS packet and decodes the encapsulation string, which is identical to what CSR8 generated upon receipt of the PADR. The PADS code is 0x65.

```
! CSR8
[19]PPPoE 19: O PADS R:0050.56a9.be8a L:0050.56a9.fb1c Gi2.528
contiguous pak, size 66
  00 50 56 A9 BE 8A 00 50 56 A9 FB 1C 81 00 0D C8
  88 63 11 65 00 13 00 2A 01 03 00 08 D2 00 00 06
  00 00 0F 20 01 02 00 02 52 38 01 04 00 10 97 58
  88 C2 01 8F 8A 96 23 D2 F7 0E E3 54 F4 D5 01 01
  00 00

! CSR2
PPPoE 19: I PADS R:0050.56a9.fb1c L:0050.56a9.be8a 3528 Gi2.528
contiguous pak, size 66
  00 50 56 A9 BE 8A 00 50 56 A9 FB 1C 81 00 0D C8
  88 63 11 65 00 13 00 2A 01 03 00 08 D2 00 00 06
  00 00 0F 20 01 02 00 02 52 38 01 04 00 10 97 58
  88 C2 01 8F 8A 96 23 D2 F7 0E E3 54 F4 D5 01 01
  00 00
```

```
IN PADS from PPPoE Session
PPPoE: Virtual Access interface obtained.
PPPoE : encaps string prepared
contiguous pak, size 24
  00 50 56 A9 FB 1C 00 50 56 A9 BE 8A 81 00 0D C8
  88 64 11 00 00 13 00 00
```

Although not part of a successful PPPoE discovery process, a PAD Termination (PADT) is sent when the session should be torn down. From CSR2's perspective, we can see the PADT is exchanged mutually between client and server depending on who terminates the session. Session ID is 31 for this session only because I added this paragraph at the end of the testing. Most of the packet is padding since the termination message is carried as a message code (0xA7) inside of the PPPoE header, which of course includes the session ID (0x1F = 31).

```
! CSR2
PPPoE 31: I PADT R:0050.56a9.fb1c L:0050.56a9.be8a 3528 Gi2.528
contiguous pak, size 64
  00 50 56 A9 BE 8A 00 50 56 A9 FB 1C 81 00 0D C8
  88 63 11 A7 00 1F 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PPPoE : Shutting down client session
[0]PPPoE 31: O PADT R:0050.56a9.fb1c L:0050.56a9.be8a Gi2.528
contiguous pak, size 64
00 50 56 A9 FB 1C 00 50 56 A9 BE 8A 81 00 0D C8
88 63 11 A7 00 1F 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

Once the PPPoE discovery process is complete, the traditional PPP negotiation must occur. In our case, three protocols must be negotiated. As done earlier, the output is shown in sections.

1. Link Control Protocol (LCP): LCP negotiates basic PPP parameters such as packet size, method of transmission, authentication, etc. The detailed codes will not be examined here, but we can watch the LCP process. One of the first things the PPP/LCP process figures out is that, because it is within a PPPoE session, it establishes that the AC is being called (call-in) and the client is calling (call-out). This is shown in the debug logs during the magic number negotiation. This negotiation is just an agreement between both routers that the number selected can be used; MRU is also negotiated as part of determining the packet sizes. The “I” and “O”, as with PPPoE discovery, represent inbound and outbound packets. Each router sends both inbound and outbound configuration requests (CONFREQ) and acknowledgements (CONFACK). Rejected configurations generate configuration reject (CONFREJ) messages, which is often due to authentication failures. Four colors are used to show the same messages displayed on CSR2 and CSR8 for mapping purposes. Once LCP is “open”, other higher-layer protocols can begin negotiating over PPP.

```

! CSR2
Vi1 PPP: Using dialer call direction
Vi1 PPP: Treating connection as a callout
Vi1 PPP: Session handle[CC00000D] Session id[13]
Vi1 LCP: Event[OPEN] State[Initial to Starting]
Vi1 PPP: No remote authentication for call-out
Vi1 LCP: O CONFREQ [Starting] id 1 len 14
Vi1 LCP: MRU 1492 (0x010405D4)
Vi1 LCP: MagicNumber 0x230EBFC3 (0x0506230EBFC3)
Vi1 LCP: Event[UP] State[Starting to REQsent]
Vi1 LCP: I CONFREQ [REQsent] id 1 len 14
Vi1 LCP: MRU 1492 (0x010405D4)
Vi1 LCP: MagicNumber 0x28BA1EA3 (0x050628BA1EA3)
Vi1 LCP: O CONFACK [REQsent] id 1 len 14
Vi1 LCP: MRU 1492 (0x010405D4)
Vi1 LCP: MagicNumber 0x28BA1EA3 (0x050628BA1EA3)
Vi1 LCP: Event[Receive ConfReq+] State[REQsent to ACKsent]
Vi1 LCP: I CONFACK [ACKsent] id 1 len 14
Vi1 LCP: MRU 1492 (0x010405D4)
Vi1 LCP: MagicNumber 0x230EBFC3 (0x0506230EBFC3)
Vi1 LCP: Event[Receive ConfAck] State[ACKsent to Open]

```

```

Vi1 PPP: Phase is FORWARDING, Attempting Forward
Vi1 LCP: State is Open

! CSR8
ppp19 PPP: Using vpn set call direction
ppp19 PPP: Treating connection as a callin
ppp19 PPP: Session handle[EC000013] Session id[19]
ppp19 LCP: Event[OPEN] State[Initial to Starting]
ppp19 PPP: No remote authentication for call-in
ppp19 PPP LCP: Enter passive mode, state[Stopped]
ppp19 LCP: I CONFREQ [Stopped] id 1 len 14
ppp19 LCP: MRU 1492 (0x010405D4)
ppp19 LCP: MagicNumber 0x230EBFC3 (0x0506230EBFC3)
ppp19 LCP: O CONFREQ [Stopped] id 1 len 14
ppp19 LCP: MRU 1492 (0x010405D4)
ppp19 LCP: MagicNumber 0x28BA1EA3 (0x050628BA1EA3)
ppp19 LCP: O CONFACK [Stopped] id 1 len 14
ppp19 LCP: MRU 1492 (0x010405D4)
ppp19 LCP: MagicNumber 0x230EBFC3 (0x0506230EBFC3)
ppp19 LCP: Event[Receive ConfReq+] State[Stopped to ACKsent]
ppp19 LCP: I CONFACK [ACKsent] id 1 len 14
ppp19 LCP: MRU 1492 (0x010405D4)
ppp19 LCP: MagicNumber 0x28BA1EA3 (0x050628BA1EA3)
ppp19 LCP: Event[Receive ConfAck] State[ACKsent to Open]
ppp19 PPP: Queue IPCP code[1] id[1]
ppp19 PPP: Queue IPV6CP code[1] id[1]
ppp19 PPP: Phase is FORWARDING, Attempting Forward
ppp19 LCP: State is Open

```

2. IP control protocol (IPCP): Since both routers want to use IPv4 on the link, those parameters must be negotiated as well. In this case, CSR2 has no IP address, and indicates this in its initial outbound CONFREQ. Interestingly, CSR8 offers the address of 209.2.8.107 using a CONFNACK (negative ACK) message, which triggers a CONFREQ from CSR2 to request that same address. CSR8 confirms it with a CONFACK. This process is shown and yellow, and the simpler exchange of CSR2 learning CSR8's static address is shown in green. After IPCP is open, each one installs a connected host route to the remote peer via the PPP interface. This allows hosts in different subnets to communicate over PPP.

```

! CSR2
Vi1 IPCP: Protocol configured, start CP. state[Initial]
Vi1 IPCP: Event[OPEN] State[Initial to Starting]
Vi1 IPCP: O CONFREQ [Starting] id 1 len 10
Vi1 IPCP: Address 0.0.0.0 (0x030600000000)
Vi1 IPCP: Event[UP] State[Starting to REQsent]
Vi1 IPCP: I CONFREQ [REQsent] id 1 len 10
Vi1 IPCP: Address 209.2.8.8 (0x0306D1020808)
Vi1 IPCP: O CONFACK [REQsent] id 1 len 10
Vi1 IPCP: Address 209.2.8.8 (0x0306D1020808)
Vi1 IPCP: Event[Receive ConfReq+] State[REQsent to ACKsent]

```

```

Vi1 IPCP: I CONFNAK [ACKsent] id 1 len 10
Vi1 IPCP: Address 209.2.8.107 (0x0306D102086B)
Vi1 IPCP: O CONFREQ [ACKsent] id 2 len 10
Vi1 IPCP: Address 209.2.8.107 (0x0306D102086B)
Vi1 IPCP: Event[Receive ConfNak/Rej] State[ACKsent to ACKsent]
Vi1 IPCP: I CONFACK [ACKsent] id 2 len 10
Vi1 IPCP: Address 209.2.8.107 (0x0306D102086B)
Vi1 IPCP: Event[Receive ConfAck] State[ACKsent to Open]
Vi1 IPCP: State is Open
Di28 IPCP: Install default route thru 209.2.8.8
Di28 Added to neighbor route AVL tree: topoid 0, address 209.2.8.8
Di28 IPCP: Install route to 209.2.8.8

! CSR8
Vi2.1 IPCP: Protocol configured, start CP. state[Initial]
Vi2.1 IPCP: Event[OPEN] State[Initial to Starting]
Vi2.1 IPCP: O CONFREQ [Starting] id 1 len 10
Vi2.1 IPCP: Address 209.2.8.8 (0x0306D1020808)
Vi2.1 IPCP: Event[UP] State[Starting to REQsent]
Vi2.1 PPP: Process pending ncp packets
Vi2.1 IPCP: Redirect packet to Vi2.1
Vi2.1 IPCP: I CONFREQ [REQsent] id 1 len 10
Vi2.1 IPCP: Address 0.0.0.0 (0x030600000000)
Vi2.1 IPCP AUTHOR: Done. Her address 0.0.0.0, we want 0.0.0.0
Vi2.1 IPCP: Pool returned 209.2.8.107
Vi2.1 IPCP: O CONFNAK [REQsent] id 1 len 10
Vi2.1 IPCP: Address 209.2.8.107 (0x0306D102086B)
Vi2.1 IPCP: Event[Receive ConfReq+] State[REQsent to REQsent]
Vi2.1 IPCP: I CONFACK [REQsent] id 1 len 10
Vi2.1 IPCP: Address 209.2.8.8 (0x0306D1020808)
Vi2.1 IPCP: Event[Receive ConfAck] State[REQsent to ACKrcvd]
Vi2.1 IPCP: I CONFREQ [ACKrcvd] id 2 len 10
Vi2.1 IPCP: Address 209.2.8.107 (0x0306D102086B)
Vi2.1 IPCP: O CONFACK [ACKrcvd] id 2 len 10
Vi2.1 IPCP: Address 209.2.8.107 (0x0306D102086B)
Vi2.1 IPCP: Event[Receive ConfReq+] State[ACKrcvd to Open]
Vi2.1 IPCP: State is Open
Vi2.1 Added to neighbor route AVL tree: topoid 0, address 209.2.8.107
Vi2.1 IPCP: Install route to 209.2.8.107

```

3. IPV6CP: Like IPCP, IPv6 information is negotiated over the link also. In this case, 64-bit interface-IDs are exchanged over the link which make up the host portion of an IPv6 LL address. The address don't make it into the IPv6 RIB but are tracked internally by PPP for forwarding. CSR2 informs CSR8 that it wants to use the address ending in DB00 (Green) and CSR8 informs CSR2 that it wants to use the address ending in 4D00 (yellow).

```

! CSR2
Vi1 IPV6CP: Protocol configured, start CP. state[Initial]

```

```

Vi1 IPV6CP: Event[OPEN] State[Initial to Starting]
Vi1 IPV6CP: O CONFREQ [Starting] id 1 len 14
Vi1 IPV6CP: Interface-Id 021E:14FF:FE15:DB00 (0x010A021E14FFFE15DB00)
Vi1 IPV6CP: Event[UP] State[Starting to REQsent]
Vi1 IPV6CP: I CONFREQ [REQsent] id 1 len 14
Vi1 IPV6CP: Interface-Id 021E:E6FF:FE4D:4D00 (0x010A021EE6FFFE4D4D00)
Vi1 IPV6CP: O CONFACK [REQsent] id 1 len 14
Vi1 IPV6CP: Interface-Id 021E:E6FF:FE4D:4D00 (0x010A021EE6FFFE4D4D00)
Vi1 IPV6CP: Event[Receive ConfReq+] State[REQsent to ACKsent]
Vi1 IPV6CP: I CONFACK [ACKsent] id 1 len 14
Vi1 IPV6CP: Interface-Id 021E:14FF:FE15:DB00 (0x010A021E14FFFE15DB00)
Vi1 IPV6CP: Event[Receive ConfAck] State[ACKsent to Open]
Vi1 IPV6CP: State is Open

! CSR8
Vi2.1 IPV6CP: Protocol configured, start CP. state[Initial]
Vi2.1 IPV6CP: Event[OPEN] State[Initial to Starting]
Vi2.1 IPV6CP: O CONFREQ [Starting] id 1 len 14
Vi2.1 IPV6CP: Interface-Id 021E:E6FF:FE4D:4D00 (0x010A021EE6FFFE4D4D00)
Vi2.1 IPV6CP: Event[UP] State[Starting to REQsent]
Vi2.1 IPV6CP: Redirect packet to Vi2.1
Vi2.1 IPV6CP: I CONFREQ [REQsent] id 1 len 14
Vi2.1 IPV6CP: Interface-Id 021E:14FF:FE15:DB00 (0x010A021E14FFFE15DB00)
Vi2.1 IPV6CP: O CONFACK [REQsent] id 1 len 14
Vi2.1 IPV6CP: Interface-Id 021E:14FF:FE15:DB00 (0x010A021E14FFFE15DB00)
Vi2.1 IPV6CP: Event[Receive ConfReq+] State[REQsent to ACKsent]
Vi2.1 IPV6CP: I CONFACK [ACKsent] id 1 len 14
Vi2.1 IPV6CP: Interface-Id 021E:E6FF:FE4D:4D00 (0x010A021EE6FFFE4D4D00)
Vi2.1 IPV6CP: Event[Receive ConfAck] State[ACKsent to Open]
Vi2.1 IPV6CP: State is Open

```

At this point, we will verify everything with show commands. The client and server both show the summary information for each PPPoE session in similar formats. Note: The session bounced once during the course of documenting the feature so the session ID incremented from 19 to 20. This output shows us the remote and local MAC addresses, port, VLAN, virtual interface, session ID, and state. It is the most valuable PPPoE show command. The string “PTA” means locally terminated and is present only the AC; it stands for PPP Termination and Aggregation.

```

R2#show pppoe session
    1 client session

```

Uniq ID	PPPoE SID	RemMAC LocMAC	Port	VT	VA VA-st	State Type
N/A	20	0050.56a9.fb1c 0050.56a9.be8a	Gi2.528	Di28	Vi1 UP	UP

```

R8#show pppoe session
    1 session in LOCALLY_TERMINATED (PTA) State

```

```

1 session total

Uniq ID  PPPoE  RemMAC      Port      VT  VA      State
      SID  LocMAC      Port      VA-st    Type
      20    20    0050.56a9.be8a  Gi2.528  28  Vi2.1    PTA
                0050.56a9.fb1c  VLAN:3528  UP

```

Some outputs/commands reference the session ID, so it is important to understand that concept. It may be useful to look at packet counters as well, shown below. ACs also maintain a summary view of all PPPoE sessions, included those forwarded past the AC or in a transient state.

```

R8#show pppoe session packets
Total PPPoE sessions 1

```

```

SID  Pkts-In      Pkts-Out  Bytes-In  Bytes-Out
20    882          1448      13651     50867

```

```

R8#show pppoe summary

```

```

PTA : Locally terminated sessions

```

```

FWDED: Forwarded sessions

```

```

TRANS: All other sessions (in transient state)

```

	TOTAL	PTA	FWDED	TRANS
TOTAL	1	1	0	0
GigabitEthernet2	1	1	0	0

The PPP show commands also give additional information which is specific to PPP. This includes PPP subprotocol negotiation details. We can see a summary of all PPP sessions on both routers and their negotiated protocols. Notice the peer name is blank since there is no authentication happening presently. Both CSR2 and CSR8 show that LCP, IPCP, and IPV6CP were successfully negotiated.

```

R2#show ppp all

```

Interface/ID	OPEN+	Nego*	Fail-	Stage	Peer Address	Peer Name
Vi1	LCP+	IPCP+	IPV6CP+	LocalT	209.2.8.8	

```

R8#show ppp all

```

Interface/ID	OPEN+	Nego*	Fail-	Stage	Peer Address	Peer Name
Vi2.1	LCP+	IPCP+	IPV6CP+	LocalT	209.2.8.108	

Looking at the details on CSR8, we can see there is a ton of PPP information for each sub-protocol. The items of greatest significance are highlighted. Note that the IPv6 address exchanges are not visible with any other show command to my knowledge. CSR2's output is very similar and is omitted for brevity.

```

R8#show ppp interface virtual-access2.1

```

```

Vi2.1 No PPP serial context

```

```

PPP Session Info

```



```

-----
Interface      : Vi2.1
PPP ID         : 0x4C000014
Phase          : UP
Stage          : Local Termination
Peer Name      :
Peer Address   : 209.2.8.108
Control Protocols: LCP[Open] IPCP[Open] IPV6CP[Open]
Session ID     : 20
AAA Unique ID  : 31
SSS Manager ID : 0x7C000029
SIP ID         : 0x61000028
PPP_IN_USE     : 0x11

Vi2.1 LCP: [Open]
Our Negotiated Options
Vi2.1 LCP:      MRU 1492 (0x010405D4)
Vi2.1 LCP:      MagicNumber 0x28BACAD4 (0x050628BACAD4)
Peer's Negotiated Options
Vi2.1 LCP:      MRU 1492 (0x010405D4)
Vi2.1 LCP:      MagicNumber 0x230F6BF6 (0x0506230F6BF6)

Vi2.1 IPCP: [Open]
Our Negotiated Options
Vi2.1 IPCP:      Address 209.2.8.8 (0x0306D1020808)
Peer's Negotiated Options
Vi2.1 IPCP:      Address 209.2.8.108 (0x0306D102086C)

Vi2.1 IPV6CP: [Open]
Our Negotiated Options
Vi2.1 IPV6CP:      Interface-Id 021E:E6FF:FE4D:4D00 (0x010A021EE6FFFE4D4D00)
Peer's Negotiated Options
Vi2.1 IPV6CP:      Interface-Id 021E:14FF:FE15:DB00 (0x010A021E14FFFE15DB00)

```

One particularly important piece of the IPV6CP debugging not shown above indicates an issue with assigning a prefix to the PPPoE client. This is poorly documented and not well known, so I highlight it. CSR8 says that it cannot allocate a prefix from the local pool since CSR2 has no remote name. The PPP show commands earlier prove this. We can rectify this by configuring a PAP username on CSR2 and tell CSR8 to use PAP authentication. PAP details are examined later.

```

! CSR8
Vi2.1 IPV6CP: Cannot use a pool without remote name

! CSR2
interface Dialer28
 ppp pap sent-username R2 password 0 PAP

! CSR8

```

```
username R2 password 0 PAP
interface Virtual-Template28
 ppp authentication pap callin
```

Although the IPV6CP debugs don't show the local IPv6 prefix being allocated to CSR2, it did actually work. CSR8 shows that one of its local prefixes was allocated for this purpose and CSR2 shows it as a global unicast address on its dialer interface.

```
R8#show ipv6 local pool
```

Pool	Prefix	Free	In use
PPPOE_POOL_V6	2001:10:2:80::/60	15	1
PD_POOL_V6	2001:192:168:80::/60	15	1

```
R2#show ipv6 interface dialer 28 | section Global
Global unicast address(es):
  2001:10:2:80:21E:14FF:FE15:DB00, subnet is 2001:10:2:80::/64
[EUI/CAL/PRE]
  valid lifetime 2591997 preferred lifetime 604797
```

Interestingly, IPV6CP relies on ordinary IPv6 ND to issue this prefix. CSR8 includes this prefix in an RA on the PPPoE virtual interface, which was pulled from the IPV6CP local pool. Upon receipt of the RA from CSR8, CSR2 uses this as the "on-link" prefix for the dialer interface.

```
CSR8#debug ipv6 nd
ICMPv6-ND: (Virtual-Access2.1,FE80::21E:E6FF:FE4D:4D00) Sending RA (60) to
FF02::1
ICMPv6-ND: MTU = 1492
ICMPv6-ND: prefix 2001:10:2:80::/64 [LA] 2592000/604800
CSR2#debug ipv6 nd
ICMPv6-ND: (Dialer28,FE80::21E:E6FF:FE4D:4D00) Received RA
ICMPv6-ND: Validating ND packet options: valid
ICMPv6-ND: Prefix : 2001:10:2:80::, Length: 64, Vld Lifetime: 2592000, Prf
Lifetime: 604800, PI Flags: C0
ICMPv6-ND: Update on-link prefix 2001:10:2:80::/64 on
Dialer28/FE80::21E:E6FF:FE4D:4D00, lifetime 2592000
```

This is not the same as prefix delegation, which is shown next. This process relies on DHCPv6 and not IPv6 ND to distribute those delegated prefixes.

```
! CSR2 and CSR8
debug ipv6 dhcp detailed

! CSR8
IPv6 DHCP: Received REBIND from FE80::21E:14FF:FE15:DB00 on Virtual-Access2.1
IPv6 DHCP: detailed packet contents
  src FE80::21E:14FF:FE15:DB00 (Virtual-Access2.1)
  dst FF02::1:2
```

```

type REBIND(6), xid 7320700
option ELAPSED-TIME(8), len 2
    elapsed-time 0
option CLIENTID(1), len 10
    00030001001E1415DB00
option ORO(6), len 6
    IA-PD,DNS-SERVERS,DOMAIN-LIST
option IA-PD(25), len 41
    IAID 0x000C0001, T1 0, T2 0
    option IAPREFIX(26), len 25
        preferred 0, valid 0, prefix 2001:192:168:80::/64
IPv6 DHCP: Using interface pool DHCP_POOL_V6
IPv6 DHCP: REBIND: Client has moved from unassigned to Virtual-Access2.1
IPv6 DHCP: Route added: 2001:192:168:80::/64 via FE80::21E:14FF:FE15:DB00
dist 1 iaid 000C0001 vrf default

```

When CSR8 selects a prefix from its local pool, it also installs a static route on the AC to reach that prefix. This is very useful because it can be redistributed into IGP as needed to provide Internet connectivity. This is redistributed into IS-IS (configuration not shown), as verified below. Short of running IGP, this is an excellent, dynamic approach to issuing IPv6 prefixes to PPPoE clients.

```

R8#show ipv6 route 2001:192:168:80::/64
Routing entry for 2001:192:168:80::/64
    Known via "static", distance 1, metric 0
    Redistributing via isis 1112
    Route count is 1/1, share count 0
    Routing paths:
        FE80::21E:14FF:FE15:DB00, Virtual-Access2.1
    Last updated 00:09:40 ago

```

```

R8#show isis database 12 R8.00-00 detail | begin IPv6_Add
IPv6 Address: 2001:10:8:12::8
Metric: 0          IPv6 (MT-IPv6) 2001:192:168:80::/64

```

Below, we see that CSR2 receives this PD prefix from CSR8 and binds it to the string "PPPOE_ISP_PREFIX" which can be used elsewhere.

```

! CSR2
IPv6 DHCP: Received REPLY from FE80::21E:E6FF:FE4D:4D00 on Dialer28
IPv6 DHCP: detailed packet contents
    src FE80::21E:E6FF:FE4D:4D00 (Dialer28)
    dst FE80::21E:14FF:FE15:DB00 (Dialer28)
    type REPLY(7), xid 7320700
    option SERVERID(2), len 10
        00030001001EE64D4D00
    option CLIENTID(1), len 10
        00030001001E1415DB00
    option IA-PD(25), len 41

```

```

IAID 0x000C0001, T1 302400, T2 483840
option IAPREFIX(26), len 25
  preferred INFINITY, valid INFINITY, prefix 2001:192:168:80::/64
IPv6 DHCP: Processing options
IPv6 DHCP: Adding prefix 2001:192:168:80::/64 to PPPOE_ISP_PREFIX

```

CSR2 can use this as another IPv6 prefix on its LAN interface. The configuration is similar to the IPV6 general prefix construct and this prefix will be included in the RA messages by default. I also add a static IPv6 prefix to this link also to support non-SLAAC capable clients, such as XRv4, but remove it from the RA so that SLAAC-capable clients don't select addresses from that prefix. The "N" flag in the show command indicates that it is not included in the RA. Also note the NAT44 inside interface, which is unrelated to IPv6 but important for IPv4 connectivity to the Internet.

```

! CSR2
interface GigabitEthernet2.524
 ip nat inside
 ipv6 address FE80::2 link-local
 ipv6 address 2001:192:168:2::2/64
 ipv6 address PPPOE_ISP_PREFIX ::2/64
 ipv6 nd prefix 2001:192:168:2::/64 no-advertise
 ipv6 nd ra lifetime 30
 ipv6 nd ra interval 10 5

```

```
R2#show ipv6 interface gigabitEthernet 2.524 prefix
```

```
IPv6 Prefix Advertisements GigabitEthernet2.524
```

Codes for 1st column:

A - Address, P - Prefix-Advertisement, O - Pool

U - Per-user prefix

Codes for 2nd column and above:

D - Default

N - Not advertised, C - Calendar

```

PD default [LA] Valid lifetime 2592000, preferred lifetime 604800
PAN 2001:192:168:2::/64 [LA] Valid lifetime 2592000, preferred lifetime
604800
AD 2001:192:168:80::/64 [LA] Valid lifetime 2592000, preferred lifetime
604800

```

CSR2 has a classic CPE configuration with NAT44 (seen already) is configured with a DHCP pool to service its hosts. Since NAT44 occurs, CSR8 only needs to reach the post-NAT public address on CSR2's dialer interface.

```

! CSR2
ip dhcp excluded-address 192.168.2.0 192.168.2.20
ip dhcp pool DHCP_POOL_V4
 network 192.168.2.0 255.255.255.0
 default-router 192.168.2.2

```

CSR1 is configured as a DHCPv4 client and an IPv6 SLAAC client. AS such, it receives an address/prefix for both protocols along with a default route. CSR1 now has full Internet connectivity for IPv4 and IPv6, and this represents a typical DSL deployment.

```
R1#show ip interface brief gigabitEthernet 2.524
Interface                IP-Address      OK? Method Status      Protocol
GigabitEthernet2.524    192.168.2.21    YES DHCP    up          up
```

```
R1#show ipv6 interface brief gigabitEthernet 2.524
GigabitEthernet2.524    [up/up]
FE80::250:56FF:FEA9:1AAA
2001:192:168:80:250:56FF:FEA9:1AAA
```

```
R1#show ip route vrf 2 0.0.0.0
Routing Table: 2
Routing entry for 0.0.0.0/0, supernet
  Known via "static", distance 254, metric 0, candidate default path
  Routing Descriptor Blocks:
  * 192.168.2.2
    Route metric is 0, traffic share count is 1
```

```
R1#show ipv6 route vrf 2 ::/0
Routing entry for ::/0
  Known via "ND", distance 2, metric 0
  Route count is 1/1, share count 0
  Routing paths:
  FE80::2, GigabitEthernet2.524
  Last updated 1d00h ago
```

We quickly confirm connectivity to the Internet from CSR1.

```
R1#ping vrf 2 13.144.2.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 13.144.2.1, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/6/15 ms
```

```
R1#ping vrf 2 2bad:beef:13:aaaa::a
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2BAD:BEEF:13:AAAA::A, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 5/7/13 ms
```

We will briefly examine service tags. We know this information is carried back and forth in the PPPoE discovery packets when negotiating a session between client and AC. If a client specifies a service tag that no ACs can service, they can respond with a PADO for that client since a “Null” service at the AC

essentially means “any service”. The client requests “BLUE” and the server responds with “BLUE”, despite “BLUE” not being configured anywhere on CSR8.

```
! CSR2
interface GigabitEthernet2.528
  pppoe-client dial-pool-number 28 service-name "BLUE"

! CSR8
PPPoE 0: I PADI R:0050.56a9.be8a L:ffff.ffff.ffff 3528 Gi2.528
contiguous pak, size 44
  FF FF FF FF FF FF 00 50 56 A9 BE 8A 81 00 0D C8
  88 63 11 09 00 00 00 14 01 01 00 04 42 4C 55 45
  01 03 00 08 87 00 00 0B 00 00 23 1D
Service tag: BLUE
PPPoE 0: O PADO, R:0050.56a9.fb1c L:0050.56a9.be8a 3528 Gi2.528
Service tag: BLUE
contiguous pak, size 70
  00 50 56 A9 BE 8A 00 50 56 A9 FB 1C 81 00 0D C8
  88 63 11 07 00 00 00 2E 01 01 00 04 42 4C 55 45
  01 03 00 08 87 00 00 0B 00 00 23 1D 01 02 00 02
  52 38 01 04 00 10 97 58 88 C2 01 8F 8A 96 23 D2
  F7 0E E3 54 F4 D5
```

Configuring CSR8 with service “RED” under the BBA means that it will only service clients with a service containing the string “RED”. The “contains” keyword indicates we can use a partial match. CSR2 will keep sending PADI messages to CSR8, which never responds with a PADO.

```
! CSR8
bba-group pppoe PPPOE_28
  virtual-template 28
  service name contains RED
PPPoE 0: I PADI R:0050.56a9.be8a L:ffff.ffff.ffff 3528 Gi2.528
contiguous pak, size 44
  FF FF FF FF FF FF 00 50 56 A9 BE 8A 81 00 0D C8
  88 63 11 09 00 00 00 14 01 01 00 04 42 4C 55 45
  01 03 00 08 87 00 00 0B 00 00 23 1D
PPPoE 0: Requested service-name BLUE has no partial match with RED,
discarding PADI R:0050.56a9.be8a L:ffff.ffff.ffff 3528 Gi2.528
```

Updating CSR8 to include the string “BLU” means that it can service CSR2, since “BLU” is contained within “BLUE”. CSR8 shows the match and sends a PADO back to CSR2.

```
! CSR8
bba-group pppoe PPPOE_28
  virtual-template 28
  service name contains BLU
PPPoE 0: I PADI R:0050.56a9.be8a L:ffff.ffff.ffff 3528 Gi2.528
```

contiguous pak, size 44

```
FF FF FF FF FF FF 00 50 56 A9 BE 8A 81 00 0D C8
88 63 11 09 00 00 00 14 01 01 00 04 42 4C 55 45
01 03 00 08 87 00 00 0B 00 00 23 1D
```

PPPoE 0: Requested service-name BLUE partial match with BLU

Service tag: BLUE

PPPoE 0: O PADO, R:0050.56a9.fb1c L:0050.56a9.be8a 3528 Gi2.528

Service tag: BLUE

However, if the client has a null service but the BBA point specifies a string, the session cannot form. CSR8 is still expecting a partial match with the “BLU” string. The BBA has restrictive logic whereby only clients requesting that specific service can be serviced by the BBA in question. This can be used for simple load-sharing, where different strings can be used by different ACs on a LAN segment. This can be overridden with the “accept-null-service” option under the BBA configuration if needed.

! CSR2

interface GigabitEthernet2.528

no pppoe-client dial-pool-number 28 service-name "BLUE"

pppoe-client dial-pool-number 28

! CSR8

PPPoE 0: I PADI R:0050.56a9.be8a L:ffff.ffff.ffff 3528 Gi2.528

contiguous pak, size 40

```
FF FF FF FF FF FF 00 50 56 A9 BE 8A 81 00 0D C8
88 63 11 09 00 00 00 10 01 01 00 00 01 03 00 08
32 00 00 0C 00 00 26 58
```

PPPoE 0: Discarding PADI with empty service-name R:0050.56a9.be8a

L:ffff.ffff.ffff 3528 Gi2.528

Before continuing, we fix the client to be back in “BLUE” service again. There are other ways to setup PPP networks, although CSR2/CSR8 is the most common. CSR9 is an AC that has clients CSR3, CSR4, and CSR5. CSR9 uses DHCP to hand out addresses to PPP, which is similar to the local-pool but uses a centralized DHCP process. The pool is still local to CSR9, but other hosts can also use addresses from this pool, not just PPP. IPCP is still used to issue IP addresses to clients, but DHCP or static addressing could technically be used. Rather than use IPv6 DHCP prefix delegation (PD), we can run IGP over the link to exchange IPV6 prefixes. This is not common but certainly works. We also enable PAP and CHAP authentication with a custom AAA method list. If RADIUS/TACACS were in play, the PPP sessions could be authenticated against a remote AAA server. In this case, the method-list just uses the local database. CSR9 prefers to use CHAP but can fallback to PAP. Notice that CSR9 enables ISIS on this link for IPv4; this is only to advertise the /24 connected prefix into ISIS for routing reachability. Passive-interface cannot be used on the virtual-template since it is always down. Each client has a separate VLAN for connectivity to the BNG, which conforms to the TR-101 1:1 VLAN paradigm.

! CSR9

bba-group pppoe PPPOE_P2MP

virtual-template 345

```

sessions per-mac limit 1
sessions per-vlan limit 1

aaa new-model
aaa authentication login default none
aaa authentication ppp PPPOE local-case

ip dhcp excluded-address 209.34.59.0 209.34.59.20
ip dhcp pool DHCP_POOL_PPPOE_NETWORK
  network 209.34.59.0 255.255.255.0
  default-router 209.34.59.9

interface Virtual-Template345
  mtu 1492
  ip address 209.34.59.9 255.255.255.0
  ip router isis 1112
  peer default ip address dhcp-pool DHCP_POOL_PPPOE_NETWORK
  ipv6 enable
  ospfv3 9 ipv6 area 0
  ospfv3 9 ipv6 network point-to-point
  ppp authentication chap pap callin PPPOE

interface GigabitEthernet2.539
  pppoe enable group PPPOE_P2MP

interface GigabitEthernet2.549
  pppoe enable group PPPOE_P2MP

interface GigabitEthernet2.559
  pppoe enable group PPPOE_P2MP

```

Aside from interface enumerations, CSR3 and CSR4 have identical configurations. They both use the same CHAP hostname as well, and refuse to use the insecure PAP method. They install default routes for IPv4 and IPv6 negotiated addresses.

```

! CSR3 and CSR4
interface Dialer3
  description CPE OUTSIDE
  mtu 1492
  ip address negotiated
  ip nat outside
  encapsulation ppp
  dialer pool 3
  dialer idle-timeout 0
  dialer persistent
  ipv6 address autoconfig default
  ipv6 enable
  ospfv3 9 ipv6 area 0

```



```

ospfv3 9 ipv6 network point-to-point
ppp chap hostname CHAP
ppp chap password 0 CHAP
ppp pap refuse
ppp ipcp route default

interface GigabitEthernet2.539
  pppoe-client dial-pool-number 3

```

CSR5 is very similar except it refuses CHAP and uses PAP. CHAP refusal is necessary since it is the preferred authentication method on the AC; failing to explicitly refuse CHAP means that authentication will fail and PAP will not be used for fallback.

```

! CSR5
interface Dialer5
  mtu 1492
  ip address negotiated
  ip nat outside
  encapsulation ppp
  dialer pool 5
  ipv6 address autoconfig default
  ospfv3 9 ipv6 area 0
  ospfv3 9 ipv6 network point-to-point
  ppp chap refuse
  ppp pap sent-username PAP_R5 password 0 PAP_R5
  ppp ipcp route default

interface GigabitEthernet2.559
  pppoe-client dial-pool-number 5

```

The only new thing to research with this design is the authentication, since IPv6 prefix-delegation is not in play, and OSPFv3 over a P2P link is not new. CSR3 and CSR9 negotiate CHAP authentication and it is successful. CSR3 receives the inbound challenge from R9, sends a response using username CHAP (which has a valid local-database entry on CSR9). CSR9 then responds that authentication was successful.

```

! CSR3
Vi2 CHAP: Redirect packet to Vi2
Vi2 CHAP: I CHALLENGE id 1 len 23 from "R9"
Vi2 LCP: State is Open
Vi2 CHAP: Using hostname from interface CHAP
Vi2 CHAP: Using password from interface CHAP
Vi2 CHAP: O RESPONSE id 1 len 25 from "CHAP"
Vi2 CHAP: I SUCCESS id 1 len 4
Vi2 PPP: Phase is FORWARDING, Attempting Forward
Vi2 PPP: Phase is ESTABLISHING, Finish LCP

! CSR9
ppp45 PPP: Phase is AUTHENTICATING, by this end

```

```

ppp45 CHAP: O CHALLENGE id 1 len 23 from "R9"
ppp45 LCP: State is Open
ppp45 CHAP: I RESPONSE id 1 len 25 from "CHAP"
ppp45 PPP: Phase is FORWARDING, Attempting Forward
ppp45 PPP: Phase is AUTHENTICATING, Unauthenticated User
ppp45 PPP: Phase is FORWARDING, Attempting Forward
Vi2.4 PPP: Phase is AUTHENTICATING, Authenticated User
Vi2.4 CHAP: O SUCCESS id 1 len 4

```

CSR5 and CSR9 fail to negotiate CHAP since CSR5 refuses it, and instead authenticate with PAP. The CHAP failure is not shown in the debug since the messages sent by CSR5 (low level PPP information) carried it. PAP has less chatter than CHAP but still has an explicit authentication request from the client and response to the server. PPP authenticate in general can be done in either direction or bidirectionally, but normally the AC will authenticate the client only. For additional security, the client can authenticate the server since PPP is a peer-to-peer protocol, generally speaking. PPPoE is not used in this fashion but authentication is transport-independent.

```

! CSR5
Vi3 PPP: Phase is AUTHENTICATING, by the peer
Vi3 PAP: Using hostname from interface PAP
Vi3 PAP: Using password from interface PAP
Vi3 PAP: O AUTH-REQ id 1 len 18 from "PAP_R5"
Vi3 LCP: State is Open
Vi3 PAP: I AUTH-ACK id 1 len 5

```

```

! CSR9
ppp46 PPP: Queue PAP code[1] id[1]
ppp46 PPP: Phase is AUTHENTICATING, by this end
ppp46 PAP: Redirect packet to ppp46
ppp46 PAP: I AUTH-REQ id 1 len 18 from "PAP_R5"
ppp46 PAP: Authenticating peer PAP_R5
ppp46 PPP: Phase is FORWARDING, Attempting Forward
ppp46 LCP: State is Open
Vi2.1 PAP: O AUTH-ACK id 1 len 5

```

Verifying the PPPoE sessions on CSR9 shows 3 subscribers, each on a different VLAN, but all in the PTA state. This shows that PPPoE is working properly. Since each PPPoE client has a statically-configured IPv6 LAN prefix, we use OSPFv3 to learn them at the BNG. CSR9 has OSPFv3 neighbors with all subscribers through the PPPoE session as well.

```

! CSR9
R9#show pppoe session
    3 sessions in LOCALLY_TERMINATED (PTA) State
    3 sessions total

```

Uniq ID	PPPoE	RemMAC	Port	VT	VA	State
	SID	LocMAC			VA-st	Type

45	45	0050.56a9.8ccf	Gi2.539	345	Vi2.4	PTA
		0050.56a9.d672	VLAN:3539		UP	
40	40	0050.56a9.2c57	Gi2.549	345	Vi2.2	PTA
		0050.56a9.d672	VLAN:3549		UP	
46	46	0050.56a9.dc63	Gi2.559	345	Vi2.1	PTA
		0050.56a9.d672	VLAN:3559		UP	

```
R9#show ospfv3 ipv6 neighbor
```

```
OSPFv3 9 address-family ipv6 (router-id 209.19.85.11)
```

Neighbor ID	Pri	State	Dead Time	Interface ID	Interface
192.168.5.5	0	FULL/ -	00:00:37	18	Virtual-Access2.1
192.168.34.3	0	FULL/ -	00:00:38	12	Virtual-Access2.4
192.168.34.4	0	FULL/ -	00:00:34	13	Virtual-Access2.2

We can verify the DHCP-issued addresses on CSR9 as well. The PPP information on CSR10 shows which address maps to which client. Notice that PAP and CHAP are also considered PPP sub-protocols and are shown in the PPP summary. The client-ID is actually the hostname in ASCII using hex values: 4348.4150 spells CH.AP and 5041.505f.5235 spells PA.P_R5.

```
R9#show ip dhcp binding
```

```
Bindings from all pools not associated with VRF:
```

IP address	Client-ID/ Hardware address/ User name	Lease expiration	Type	State	Interface
209.34.59.35	4348.4150	Infinite		On-demand	Selecting Vi2.2
209.34.59.37	4348.4150	Infinite		On-demand	Selecting Vi2.4
209.34.59.38	5041.505f.5235	Infinite		On-demand	Selecting Vi2.1

```
R9#show ppp all
```

Interface/ID	OPEN+	Nego*	Fail-	Stage	Peer Address	Peer Name
Vi2.1	LCP+	PAP+	IPCP+	IPV6>	LocalT	209.34.59.38
Vi2.4	LCP+	CHAP+	IPCP+	IPV>	LocalT	209.34.59.37
Vi2.2	LCP+	CHAP+	IPCP+	IPV>	LocalT	209.34.59.35

CSR1 in the client VRF behind CSR3 and CSR4 (tied together with HSRP, which is examined in greater detail in the NAT44/NAT444 section), we use all static addressing and static routing for IPv4 and IPv6. CSR1 has reachability within the client VRF as desired.

```
! CSR1
```

```
interface GigabitEthernet2.534
```

```
 vrf forwarding 34
```

```
 ip address 192.168.34.1 255.255.255.0
```

```
 ipv6 address 2001:192:168:34::1/64
```

```
ip route vrf 34 0.0.0.0 0.0.0.0 192.168.34.254
```

```
ipv6 route vrf 34 ::/0 GigabitEthernet2.534 FE80::254
```

```

R1#ping vrf 34 13.144.2.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 13.144.2.1, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/6/15 ms

R1#ping vrf 34 2bad:beef:13:aaaa::a
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2BAD:BEEF:13:AAAA::A, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 5/6/10 ms

```

Last, we examine the DHCPv4 proxy service in conjunction with PPP. The equivalent for IPv6 would be using some AAA server to issue IPv6 prefixes for prefix-delegation, but that is not tested here. In this example, PPPoE clients CSR6 and CSR7 dial-in to CSR10 and request addresses via IPCP. Because CSR10 has neither local pools nor DHCP pools configured, it sends the requests to a remote DHCP server (CSR9) much like a DHCP relay. That DHCP server will issue addresses on a per hostname basis, which implies that one of two things must happen: authentication must be used so the server sees hostnames, or the server must be configured to assign different hostnames behind the scenes to each client. Failure to do this will result in the same client-ID presented to the DHCP server, which results in the same IPCP address allocated to multiple clients. This ultimately breaks routing. CSR10 uses local DHCPv6 for PD but rather than call a local pool, it assigns specific prefixes to CSR6 and CSR7. The hexadecimal values are the DHCPv6 unique ID (DUID) of each client, which we verify later. I also add several other minor PPP IPCP and IPV6CP options to enforce uniqueness among addressing for clients, but the “username” uniqueness is what allows this design to actually work. The BBA configuration includes 2 sessions per VLAN since all PPPoE speakers are on the same segment in this broadband design, but each client can only have one session. The throttling ensures that a PPPoE client cannot try to initiate more than 5 sessions in 5 minutes, and if it does, it is blocked for 0 minutes (not blocked at all). We can set the 802.1p bits in the 802.1q VLAN tag to be CoS 7 so that the PPPoE control traffic is less likely to be dropped during times of congestion. The reason two DHCP servers are defined is because the addressing is somewhat asymmetric; CSR10 can reach CSR9’s loopback, but when CSR9 responds, it does so from a suppressed transit link as the source. CSR10 needs to account for this source or else the DHCP OFFER is automatically rejected, which explains the two DHCP server commands.

```

! CSR10
ipv6 dhcp pool DHCP_PD_R6_R7_SPECIFIC
 prefix-delegation 2001:192:168:7::/64 00030001001E49CAA400 lifetime infinite
infinite
 prefix-delegation 2001:192:168:6::/64 00030001001EBD696200 lifetime infinite
infinite

ip dhcp-server 10.9.9.9
ip dhcp-server 10.9.11.9
ip address-pool dhcp-proxy-client

```

```

bba-group pppoe PPPOE_VLAN
  virtual-template 67
  sessions per-mac limit 1
  sessions per-vlan limit 2
  sessions per-vlan throttle 5 5 0
  control-packets vlan cos 7

interface Virtual-Template67
  mtu 1492
  ip unnumbered Loopback67
  peer ip address forced
  peer default ip address dhcp
  ipv6 enable
  no ipv6 nd ra suppress
  ipv6 nd ra lifetime 60
  ipv6 nd ra interval 10 5
  ipv6 dhcp server DHCP_PD_R6_R7_SPECIFIC
  ppp ipcp mask reject
  ppp ipcp username unique
  ppp ipcp address required
  ppp ipcp address unique
  ppp ipv6cp address unique

interface GigabitEthernet2.556
  pppoe enable group PPPOE_VLAN

```

The client configuration is nothing special, and is nearly identical on CSR6 and CSR7; only CSR6 is shown. Both clients receive IPCP addresses and perform NAT44 to give access for their LAN hosts. They also learn IPv6 prefixes via DHCPv6 PD for their LANs to access the IPv6 Internet.

```

! CSR6
interface Dialer6
  description CPE OUTSIDE
  mtu 1492
  ip address negotiated
  ip nat outside
  encapsulation ppp
  dialer pool 6
  dialer idle-timeout 0
  dialer persistent
  ipv6 address autoconfig default
  ipv6 dhcp client pd PREFIX_FROM_ISP
  ppp ipcp route default

interface GigabitEthernet2.556
  pppoe-client dial-pool-number 6

```

To support this design, we also need to add a new DHCP pool to service CSR10's PPPoE clients. The pool is configured on CSR9.

```
! CSR9
ip dhcp excluded-address 209.56.70.0 209.56.70.20
ip dhcp pool DHCP_PROXY_V4
 network 209.56.70.0 255.255.255.0
 default-router 209.56.70.10
```

As a general comment, the debug below shows what happens if the DHCP server from which the DHCP OFFER is received is not explicitly configured on the proxy router. The DHCP OFFER arriving on CSR10 is automatically rejected if 10.9.11.9 is not configured as an explicit DHCP server.

```
CSR10#debug dhcp
DHCP: offer received from 10.9.11.9
DHCP: offer: server 10.9.11.9 not in approved list
```

On CSR10, we examine the debugs to see how a client dials in. The initial PPP LCP process is unchanged as the DHCPv4 process is invoked by IPCP, and upper layer PPP sub-protocol. IPCP is "stalled" waiting for an address.

```
! CSR10
Vi2.1 IPCP: Stalled on pool request
Vi2.1 IPCP: CP stalled on event[IPCP Allocate Address]
Vi2.1 IPCP: Stalled on option [Address]
```

At this point, the DHCP process sends a DHCPDISCOVER to the DHCP server. The discover is sent twice, once to each server, but we know 10.9.11.9 is unroutable. The reply comes from 10.9.11.9 (CSR9 transit link) and contains the address 209.56.70.22.

```
! CSR10
DHCP: proxy allocate request
DHCP: new entry. add to queue
DHCP: SDiscover attempt # 1 for entry:
DHCP: SDiscover: sending 276 byte length DHCP packet
DHCP: SDiscover 276 bytes
DHCP: SDiscover 276 bytes

DHCP: Received a BOOTREP pkt
DHCP: offer received from 10.9.11.9
DHCP: SRequest attempt # 1 for entry:
DHCP: SRequest- Server ID option: 10.9.11.9
DHCP: SRequest- Requested IP addr option: 209.56.70.22
DHCP: SRequest placed lease len option: 75144
DHCP: SRequest: 294 bytes
DHCP: SRequest: 294 bytes
```

```

DHCP: SRequest: 294 bytes
DHCP: XID MATCH in dhcp_for_us()
DHCP: Received a BOOTREP pkt
DHCP Proxy Client Pooling: ***Allocated IP address: 209.56.70.22

```

This address is returned to the IPCP process for allocation to the client. IPCP is now “unstalled”. An inbound CONFREQ arrives with all zeroes, essentially requesting an address. The AC uses the CONFNAK message, sent outbound to the client, as a method of offering an address. The client then formally request the address and the AC confirms it. This is the same mechanism seen earlier for local-pool IPCP address allocation.

```

! CSR10
Vi2.1 IPCP: CP unsta1l
Vi2.1 IPCP: Continue processing stalled packet:
Vi2.1 IPCP: I CONFREQ [ACKrcvd] id 1 len 10
Vi2.1 IPCP: Address 0.0.0.0 (0x030600000000)
Vi2.1 PPP/IPAM: ipcp_req_addr: s_data=C000056 r=0 a=0 ans=0
Vi2.1 IPCP AUTHOR: Done. Her address 0.0.0.0, we want 0.0.0.0
Vi2.1 IPCP: Pool returned 209.56.70.22
Vi2.1 IPCP: O CONFNAK [ACKrcvd] id 1 len 10
Vi2.1 IPCP: Address 209.56.70.22 (0x0306D1384616)
Vi2.1 IPCP: Event[Receive ConfReq-] State[ACKrcvd to ACKrcvd]
Vi2.1 IPCP: I CONFREQ [ACKrcvd] id 2 len 10
Vi2.1 IPCP: Address 209.56.70.22 (0x0306D1384616)
Vi2.1 PPP/IPAM: ipcp_req_addr: s_data=0 r=0 a=0 ans=0
Vi2.1 IPCP: O CONFACK [ACKrcvd] id 2 len 10
Vi2.1 IPCP: Address 209.56.70.22 (0x0306D1384616)

```

The DHCP server now shows two addresses allocated to clients CSR6 and CSR7. The PPP details on CSR10 can show which address went to which host. Notice the single digit difference in the client-ID, which was done by CSR10 by making the usernames unique on call-in. Without this, the DHCP server thinks that the same client keeps asking for an address, so it responds with the same address over and over, which is not valid as it breaks routing on CSR10.

```

R9#show ip dhcp binding 209.56.70.22
IP address      Client-ID/      Lease expiration  Type      State      Interface
                Hardware address/
                User name
209.56.70.22    003d.3230.392e.3536.  MON 09 2015 08:48 PM  Automatic  Active      Gig2.591
                2e37.302e.3130.3d56.
                6932.2e31

R9#show ip dhcp binding 209.56.70.23
IP address      Client-ID/      Lease expiration  Type      State      Interface
                Hardware address/
                User name
209.56.70.23    003d.3230.392e.3536.  MON 09 2015 08:54 PM  Automatic  Active      Gig2.591
                2e37.302e.3130.3d56.
                6932.2e32

```

```
R10#show ppp all
```

Interface/ID	OPEN+	Nego*	Fail-	Stage	Peer Address	Peer Name
Vi2.2	LCP+	IPCP+	IPV6CP+	LocalT	209.56.70.23	
Vi2.1	LCP+	IPCP+	IPV6CP+	LocalT	209.56.70.22	

We can also verify the DUIDs on the clients, which do not appear configurable. To issue specific IPv6 prefixes to clients, we can map these DUIDs to manual prefixes inside the DHCPv6 pool on CSR10. This is less dynamic but more granular than using a local pool. AAA attributes allow for this functionality as well, but that is not tested here. One would have to do this first on a router if trying to assign specific PD prefixes to a CPE device via DHCPv6.

```
R6#show ipv6 dhcp
```

```
This device's DHCPv6 unique identifier(DUID): 00030001001EBD696200
```

```
R7#show ipv6 dhcp
```

```
This device's DHCPv6 unique identifier(DUID): 00030001001E49CAA400
```

To verify the CoS markings applied to CSR10, we can enable PPPoE packet debugging on CSR10 and CSR9 to compare the differences. Within CSR9's dot1q PADO header, the 4 bits preceding the VLAN ID are 0000; the first 3 bits represent the CoS which is 000, or 0 in decimal. The packet PADO from CSR10, however, has bits 1110 (0xE) which is 111, or 7. This CoS marking applies to PADO and PADS packets for PPPoE discovery, as well as PPP's LCP, NCP sub-protocols (IPCP, IPV6CP, etc), keepalives, and authentication. CSR10's PADO, PADS and PADT are shown to prove this; notice that the PADT does not have this marking set.

```
! CSR9
```

```
PPPoE 0: O PADO, R:0050.56a9.d672 L:0050.56a9.8ccf 3539 Gi2.539
```

```
Service tag: NULL Tag
```

```
contiguous pak, size 66
```

```
00 50 56 A9 8C CF 00 50 56 A9 D6 72 81 00 0D D3
88 63 11 07 00 00 00 2A 01 01 00 00 01 03 00 08
C3 00 00 02 00 00 26 8C 01 02 00 02 52 39 01 04
00 10 6A EC 54 7B 52 F4 6B 8F 8D AA 32 83 7D 6E
B1 0D
```

```
! CSR10
```

```
PPPoE 0: O PADO, R:0050.56a9.f961 L:0050.56a9.ea77 3556 Gi2.556
```

```
Service tag: NULL Tag
```

```
contiguous pak, size 67
```

```
00 50 56 A9 EA 77 00 50 56 A9 F9 61 81 00 ED E4
88 63 11 07 00 00 00 2B 01 01 00 00 01 03 00 08
A9 00 00 05 00 00 1C CB 01 02 00 03 52 31 30 01
04 00 10 D3 77 CD ED 4B 6B AF E9 12 94 4A 4D F4
0C 92 34
```

```
[52] PPPoE 52: O PADS R:0050.56a9.ea77 L:0050.56a9.f961 Gi2.556
```


contiguous pak, size 67

```
00 50 56 A9 EA 77 00 50 56 A9 F9 61 81 00 ED E4
88 63 11 65 00 34 00 2B 01 03 00 08 A9 00 00 05
00 00 1C CB 01 02 00 03 52 31 30 01 04 00 10 D3
77 CD ED 4B 6B AF E9 12 94 4A 4D F4 0C 92 34 01
01 00 00
```

[50]PPPoE 50: O PADT R:0050.56a9.ea77 L:0050.56a9.f961 Gi2.556

contiguous pak, size 64

```
00 50 56 A9 EA 77 00 50 56 A9 F9 61 81 00 0D E4
88 63 11 A7 00 32 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

For client connectivity, we use the same method on CSR6 and CSR7 as we did on CSR2. This involves a local DHCPv4 pool and NAT44 for IPv4 connectivity, with IPv6 PD for the IPv6 hosts. The configuration on all devices, include the CSR1 client VRFs, is not shown. We can verify connectivity for IPv4 and IPv6 using both CSR6 and CSR7 below. Unfortunately, despite being in different VRFs, XE does not let us configure multiple IPv6 ND defaults on multiple interfaces. We manually configure static routes for VRF 6 and 7 as a result (not shown).

R1#ping vrf 6 13.144.2.1

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 13.144.2.1, timeout is 2 seconds:

!!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 4/6/13 ms

R1#ping vrf 6 2bad:beef:13:aaaa::a

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 2BAD:BEEF:13:AAAA::A, timeout is 2 seconds:

!!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 6/8/16 ms

R1#ping vrf 7 13.144.2.1

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 13.144.2.1, timeout is 2 seconds:

!!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 4/6/15 ms

R1#ping vrf 7 2bad:beef:13:aaaa::a

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 2BAD:BEEF:13:AAAA::A, timeout is 2 seconds:

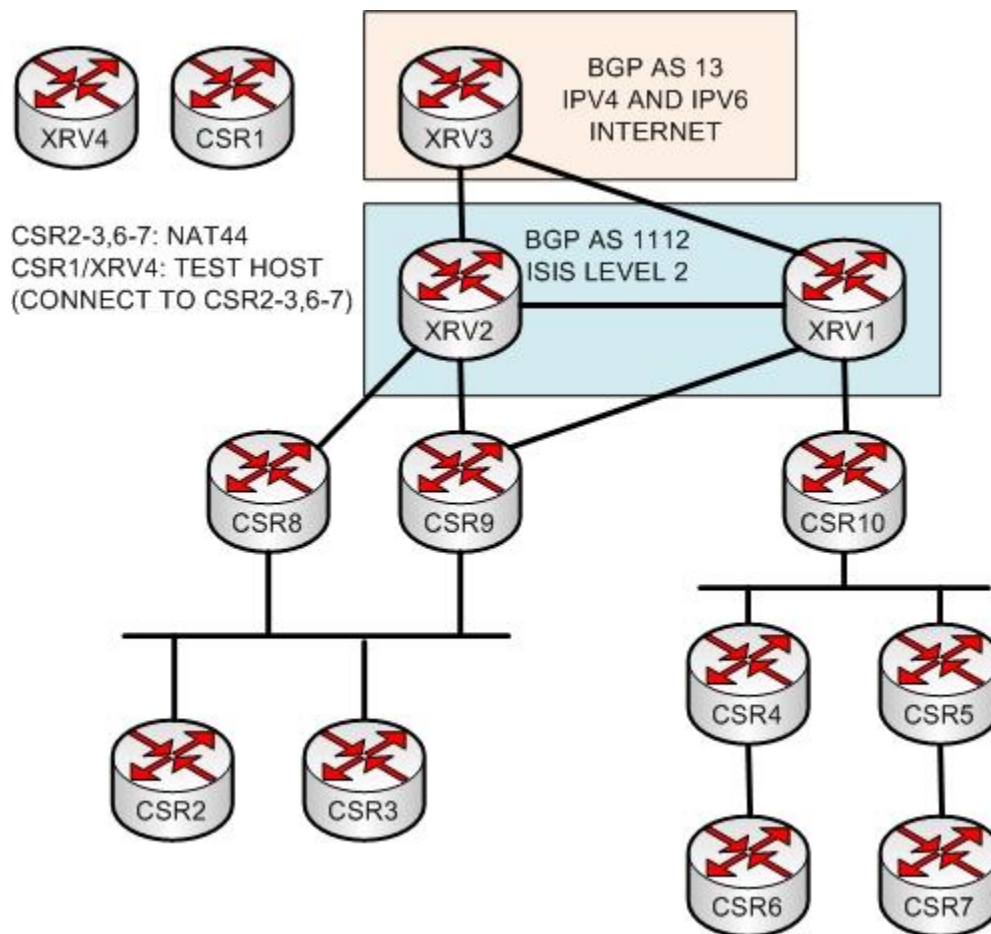
!!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 6/7/12 ms

Additional Reading – Reference configurations “pppoe-tech”

1.2.2 Multi-service PPPoE and LAC/LNS architecture

This section tests using “smart” PPPoE server selection as well as a basic LAC/LNS architecture using PPPoE as the access technology. The technical PPPoE and L2TP details are summarized since the architecture is the focus in this section; those technologies have their own sections. The network is shown below which is very similar to the PPPoE technology architecture. This time, CSR2 and CSR3 are PPPoE clients with CSR8 and CSR9 as ACs on the same LAN. CSR10 is an LNS with CSR4 and CSR5 as LACs. CSR6 and CSR7 are PPPoE clients like CSR2 and CSR3. This allows us to test PPPoE in conjunction with L2TP VPDN technologies. The upper-level architecture is still BGP-oriented since XRv does not support PPPoE or any L2VPN features, and CSR1 is generally used for most tests because it supports IPv6 SLAAC (XR in general does not).



CSR2 wants to join the "RED" service and states this in its PADI as seen in the PPPoE technology section. Both CSR8 and CSR9 offer "RED" service, but CSR9's PADO is delayed by about 1 second. Cisco will round these backoff timers to the closest multiple of 256 ms, which is why I chose 1024 ms. The PADI is a layer 2 broadcast, and both ACs will respond, but CSR2 will use CSR8's PADO since it was received first. The PADO delay timer, combined with service-name selection, is how one can achieve granularity/load-sharing with BNG nodes. Basic features like DHCPv6 PD and local pools are not shown again on the ACs since they are the same as the earlier examples. No new complexity is being introduced with those

technologies. The interface to which the VTs are unnumbered is advertised into IS-IS passively, and the netmask is big enough to encompass the entire local pool. In this way, routing to the PPPoE clients is cleanly achieved without manual configuration. IPv6 static routes generated by DHCPv6 for PD are also redistributed into IS-IS.

```
! CSR8
bba-group pppoe PPPOE_RED
  virtual-template 89
  service name contains RED
  sessions per-vlan limit 5
  pado delay 0
  control-packets vlan cos 7

interface GigabitEthernet2.589
  pppoe enable group PPPOE_RED

interface Virtual-Template89
  mtu 1492
  ip unnumbered Loopback208
  peer default ip address pool PPPOE_RED_IPV4
  ipv6 enable
  no ipv6 nd ra suppress
  ipv6 nd ra interval 30
  ipv6 dhcp server DHCPV6_PD

! CSR9
bba-group pppoe PPPOE_RED
  virtual-template 89
  service name contains RED
  sessions per-vlan limit 5
  pado delay 1024
  control-packets vlan cos 7

bba-group pppoe PPPOE_BLUE
  virtual-template 99
  service name contains BLUE accept-null-service
  sessions per-vlan limit 5
  pado delay 0
  control-packets vlan cos 6

interface Virtual-Template89
  mtu 1492
  ip unnumbered Loopback209
  peer default ip address pool PPPOE_RED_IPV4
  ipv6 enable
  no ipv6 nd ra suppress
  ipv6 nd ra interval 30
  ipv6 dhcp server DHCPV6_PD
```

```

interface Virtual-Template99
  mtu 1492
  ip unnumbered Loopback209
  peer default ip address pool PPPOE_BLUE_IPV4
  ipv6 enable
  no ipv6 nd ra suppress
  ipv6 nd ra interval 30
  ipv6 dhcp server DHCPV6_PD

interface GigabitEthernet2.589
  pppoe enable group PPPOE_RED

interface GigabitEthernet3.589
  pppoe enable group PPPOE_BLUE

```

The client configurations are shown below. Basic features like NAT44 and DHCPv4 are not shown again since they are the same as the earlier examples. No new complexity is being introduced with those technologies. CSR3's dialer is identical to CSR2's with the exception of numbering, so it is not shown again. The only difference is the service-name.

```

! CSR2
interface Dialer2
  mtu 1492
  ip address negotiated
  ip nat outside
  encapsulation ppp
  dialer pool 2
  dialer idle-timeout 0
  dialer persistent
  ipv6 address autoconfig default
  ipv6 dhcp client pd PPPOE_ISP_PREFIX
  ppp ipcp route default

interface GigabitEthernet2.589
  pppoe-client dial-pool-number 2 service-name "RED"

! CSR3
interface GigabitEthernet2.589
  pppoe-client dial-pool-number 3 service-name "BIG_BLUE_HOUSE"

```

We can watch the session establish when CSR2 initiates the discovery process. The debug on CSR2 (with timestamps) clearly shows the outgoing PADI, followed by two incoming PADOs.

```

R2#debug pppoe event
R2#debug pppoe packet
00:04:51.747: pppoe_send_padi
00:04:51.750: PPPoE 0: I PADO R:0050.56a9.fb1c L:0050.56a9.be8a 3589 Gi2.589

```

```

00:04:53.011: PPPoE 0: I PADO R:0050.56a9.d672 L:0050.56a9.be8a 3589 Gi2.589
00:04:53.796: PPPOE: we've got our pado and the pado timer went off
00:04:53.796: OUT PADR from PPPoE Session
00:04:53.802: PPPoE 1: I PADS R:0050.56a9.fb1c L:0050.56a9.be8a 3589 Gi2.589
00:04:53.802: IN PADS from PPPoE Session

```

CSR8 receives the PADI and matches it to the RED service. The PADO is immediately sent in reply, to which CSR2 then issues a PADR. The PPPoE discovery process continues normally and the session is formed between CSR2 and CSR8 (debug is trimmed).

```

! CSR8
00:04:51.155: PPPoE 0: I PADI R:0050.56a9.be8a L:ffff.ffff.ffff 3589 Gi2.589
00:04:51.155: PPPoE 0: Requested service-name RED partial match with RED
00:04:51.155: Service tag: RED
00:04:51.155: PPPoE 0: O PADO, R:0050.56a9.fb1c L:0050.56a9.be8a 3589 Gi2.589
00:04:51.155: Service tag: RED
00:04:53.204: PPPoE 0: I PADR R:0050.56a9.be8a L:0050.56a9.fb1c 3589 Gi2.589
00:04:53.204: Service tag: RED
00:04:53.206: [5]PPPoE 1: O PADS R:0050.56a9.be8a L:0050.56a9.fb1c Gi2.589

```

CSR9 also receives the PADI and matches it to the RED service. It sees the PADI twice, once for RED and one for BLUE matching, and fails the BLUE match as expected. The PADO is sent about 1 second later, and CSR2 never replies with a PADR back to CSR9. CSR9 creates no PPPOE state and acts as if nothing happened.

```

! CSR9
00:04:51.212: PPPoE 0: I PADI R:0050.56a9.be8a L:ffff.ffff.ffff 3589 Gi2.589
00:04:51.212: PPPoE 0: Requested service-name RED partial match with RED
00:04:51.212: Service tag: RED
00:04:51.212: PPPoE: PADO id 0: Starting timer for 1024 msec
00:04:51.212: PPPoE 0: I PADI R:0050.56a9.be8a L:ffff.ffff.ffff 3589 Gi3.589
00:04:51.212: PPPoE 0: Requested service-name RED has no partial match with
BLUE, discarding PADI R:0050.56a9.be8a L:ffff.ffff.ffff 3589 Gi3.589
00:04:52.474: PPPoE: Sending PADO for pado id 0
00:04:52.474: PPPoE 0: O PADO, R:0050.56a9.d672 L:0050.56a9.be8a 3589 Gi2.589
00:04:52.474: Service tag: RED

```

We can also verify this with show commands. CSR2 is connected to the MAC address of CSR8; we can see the IP address, assuming IPCP is negotiated, by checking the PPP details.

```

R2#show pppoe session
      1 client session

```

Uniq ID	PPPoE	RemMAC	Port	VT	VA	State
	SID	LocMAC			VA-st	Type
N/A	1	0050.56a9.fb1c	Gi2.589	Di2	Vi2	UP
		0050.56a9.be8a			UP	

```
R2#show ppp all
```

Interface/ID	OPEN+	Nego*	Fail-	Stage	Peer Address	Peer Name
Vi2	LCP+	IPCP+	IPV6CP+	LocalT	209.8.8.8	

CSR2 successfully received an IP address via IPCP (CSR8 local pool) and an IPv6 address via autoconfiguration, which was exchanged with IPV6CP. We also validate that IPv6 prefix delegation worked using DHCPv6; CSR2 receives a prefix from the pool of prefixes and CSR8 creates a static route to be redistributed into IGP.

```
R2#show ppp interface vi2 | begin IPCP
```

```
Vi2 IPCP: [Open]
```

```
Our Negotiated Options
```

```
Vi2 IPCP: Address 209.8.9.50 (0x0306D1080932)
```

```
Peer's Negotiated Options
```

```
Vi2 IPCP: Address 209.8.8.8 (0x0306D1080808)
```

```
Vi2 IPV6CP: [Open]
```

```
Our Negotiated Options
```

```
Vi2 IPV6CP: Interface-Id 021E:14FF:FE15:DB00 (0x010A021E14FFFE15DB00)
```

```
Peer's Negotiated Options
```

```
Vi2 IPV6CP: Interface-Id 021E:E6FF:FE4D:4D00 (0x010A021EE6FFFE4D4D00)
```

```
R2#show ipv6 general-prefix
```

```
IPv6 Prefix PPPOE_ISP_PREFIX, acquired via DHCP PD
```

```
2001:192:168:80::/64 Valid lifetime 2591986, preferred lifetime 604786  
GigabitEthernet2.524 (Address command)
```

```
R8#show ipv6 route static | begin ^S
```

```
S 2001:192:168:80::/64 [1/0]  
via FE80::21E:14FF:FE15:DB00, Virtual-Access2.1
```

Looking at CSR3 as a client, CSR8 does not offer the BLUE service but CSR9 does. Since CSR3 wants to join the BLUE service, only one PADO will be received from CSR9 since CSR8 cannot support it. CSR9 also supports clients with the "null" service to catch clients by default. For brevity we will debug only PPPoE events, not packets.

```
R3#debug pppoe packet
```

```
00:27:23.143: padi timer expired
```

```
00:27:23.143: Sending PADI: Interface = GigabitEthernet2.589
```

```
00:27:23.147: PPPoE 0: I PADO R:0050.56a9.4c24 L:0050.56a9.8ccf 3589 Gi2.589
```

```
00:27:25.192: PPPOE: we've got our pado and the pado timer went off
```

```
00:27:25.192: OUT PADR from PPPoE Session
```

```
00:27:25.197: PPPoE 36: I PADS R:0050.56a9.4c24 L:0050.56a9.8ccf 3589  
Gi2.589
```

```
00:27:25.197: IN PADS from PPPoE Session
```

```

! CSR8
00:27:24.115: PPPoE 0: I PADI R:0050.56a9.8ccf L:ffff.ffff.ffff 3589 Gi2.589
00:27:24.115: PPPoE 0: Requested service-name BIG_BLUE_HOUSE has no partial
match with RED, discarding PADI R:0050.56a9.8ccf L:ffff.ffff.ffff 3589
Gi2.589

! CSR9
00:27:24.172: PPPoE 0: I PADI R:0050.56a9.8ccf L:ffff.ffff.ffff 3589 Gi2.589
00:27:24.172: PPPoE 0: Requested service-name BIG_BLUE_HOUSE has no partial
match with RED, discarding PADI R:0050.56a9.8ccf L:ffff.ffff.ffff 3589
Gi2.589
00:27:24.172: PPPoE 0: I PADI R:0050.56a9.8ccf L:ffff.ffff.ffff 3589 Gi3.589
00:27:24.172: PPPoE 0: Requested service-name BIG_BLUE_HOUSE partial match
with BLUE
00:27:24.172: Service tag: BIG_BLUE_HOUSE
00:27:24.172: PPPoE 0: O PADO, R:0050.56a9.4c24 L:0050.56a9.8ccf 3589 Gi3.589
00:27:24.172: Service tag: BIG_BLUE_HOUSE
00:27:26.221: PPPoE 0: I PADR R:0050.56a9.8ccf L:0050.56a9.4c24 3589 Gi3.589
00:27:26.222: Service tag: BIG_BLUE_HOUSE
00:27:26.222: PPPoE : encap string prepared
00:27:26.222: [327]PPPoE 36: O PADS R:0050.56a9.8ccf L:0050.56a9.4c24
Gi3.589

```

A quick PPPoE/PPP verification shows that CSR3 is working properly.

```

R3#show pppoe session
    1 client session

```

Uniq ID	PPPoE	RemMAC	Port	VT	VA	State
	SID	LocMAC			VA-st	Type
N/A	38	0050.56a9.4c24	Gi2.589	Di3	Vi3	UP
		0050.56a9.8ccf			UP	

```

R3#show ppp all

```

Interface/ID	OPEN+	Nego*	Fail-	Stage	Peer Address	Peer Name
Vi3	LCP+	IPCP+	IPV6CP+	LocalT	209.9.9.9	

As a final verification to ensure NAT44 is working on CSR2 and CSR3, as well as IPv6 global unicast routing, we will send traffic from CSR1 to the Internet via both clients for both protocols. This feature is sometimes called the “smart” PPPoE server selection mechanism.

```

R1#ping vrf 2 13.144.2.1
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 3/6/16 ms

```

```

R1#ping vrf 3 13.144.2.1
[snip]

```

```
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/6/16 ms
```

```
R1#ping vrf 2 2bad:beef:13:dddd::d  
[snip]
```

```
Success rate is 100 percent (5/5), round-trip min/avg/max = 5/6/9 ms
```

```
R1#ping vrf 3 2bad:beef:13:dddd::d  
[snip]
```

```
Success rate is 100 percent (5/5), round-trip min/avg/max = 7/10/25 ms
```

Next, we progress to the LAC/LNS configuration. CSR4 and CSR5 are LACs that create L2TP tunnels to the LNS. The PPPoE sessions terminate on the LACs, but all of the intelligent PPP negotiation happens with the LNS. The LAC simply forward the PPP connections onto the LNS inside of an L2TP tunnel. This allows the CPE routers to appear directly connected to the LNS. For brevity, only CSR4 and CSR6 will be analyzed in terms of LAC and CPE; CSR5 and CSR7 are configured almost identically. IPCP and IPV6CP work exactly as one would expect; the only caveat is that all of this logic is centralized on the LNS, not the LACs. First, we will examine the CPE configurations, which are literally identical to the configurations on CSR2 and CSR3. The differences are shown below; the PAP hostname includes the domain-name so that the Virtual Private Dial-up Network (VPDN) process can match this user to a vpdn-group. All other CPE settings, such as NAT, IPCP, DHCPv6 PD, MTU, etc are the same as CSR2 and CSR3.

```
! CSR6  
interface Dialer6  
  ppp pap sent-username R6@lab.local password 0 R6  
  
interface GigabitEthernet2.546  
  pppoe-client dial-pool-number 6  
  
! CSR7  
interface Dialer6  
  ppp pap sent-username R7@lab.local password 0 R7  
  
interface GigabitEthernet2.557  
  pppoe-client dial-pool-number 7
```

The LAC configuration is a little more involved. This is where the PPPoE logic terminates as there is a BBA configured on the interfaces towards the CPEs. The LAC will initiate L2TP tunnels to CSR10's LAN interface IP address (any reachable address is fine) provided the dialing user is within the lab.local domain. This is why the PAP username is valuable (CHAP can be used also). Before configuring any VPDN features, we must enable the process globally, and in this case, we want to perform domain-based matching. Notice that the LAC must be configured to authenticate CPE's via PAP despite not actually doing it. When creating L2TP tunnels, both CSR4 and CSR5 will use the name "LAC45" to identify themselves.

```
! CSR4
```



```

vpdn enable
vpdn search-order domain

vpdn-group LAC
 request-dialin
 protocol l2tp
 domain lab.local
 initiate-to ip 10.45.10.10
 local name LAC45
 l2tp tunnel password 0 L2TP_AUTH

bba-group pppoe LAC
 virtual-template 10
 sessions per-mac limit 2

interface Virtual-Template10
 mtu 1492
 no ip address
 ppp authentication pap callin

interface GigabitEthernet2.546
 pppoe enable group LAC

```

Last, we configure the LNS. AAA must be enabled or else authentication will fail, but we can simply use the local database. Usernames are manually configured for R6 and R7, which must include the domain-name as well since that is part of the PAP hostname string. The LACs requested dial-in, and the LNS accepts dial-in. It will terminate any L2TP tunnel from devices with hostname “LAC45”, which is both CSR4 and CSR5. Like the BBA object, the VPDN-group on the LNS will reference a virtual-template. This is configured just like CSR8 and CSR9 in terms of PPP options and protocols. Here, we can configure IPCP and IPV6CP to enable IPv4/v6 reachability to the CPEs. PAP authentication is also enabled on this interface. The local pools and other unrelated objects are not shown.

```

! CSR10
aaa new-model
aaa authentication login default none
aaa authentication ppp default local

username R6@lab.local password 0 R6
username R7@lab.local password 0 R7

vpdn enable
vpdn-group LNS
 accept-dialin
 protocol l2tp
 virtual-template 10
 terminate-from hostname LAC45
 l2tp tunnel password 0 L2TP_AUTH

```

```

interface Virtual-Template10
  mtu 1492
  ip unnumbered Loopback209
  peer default ip address pool LNS_POOL
  ipv6 enable
  no ipv6 nd ra suppress
  ipv6 nd ra interval 30
  ipv6 dhcp server DHCPV6_PD
  ppp authentication pap callin

```

With CSR7 disabled for now, we will debug the PPPoE, PPP, L2TP, and VPDN activities as necessary on CSR6, CSR4, and CSR10. First, the PPPoE exchange happens between the CPE and the LAC, and is limited to only those nodes. The LNS is unaware of what flavor of PPP is used on the access link. The PPPoE basic exchange is shown below.

```

R6#debug pppoe event
R6#debug ppp negotiation
02:22:07.093: Sending PADI: Interface = GigabitEthernet2.546
02:22:07.096: PPPoE 0: I PADO R:0050.56a9.2c57 L:0050.56a9.de0d 3546 Gi2.546
02:22:09.141: PPPOE: we've got our pado and the pado timer went off
02:22:09.141: OUT PADR from PPPoE Session
02:22:09.144: PPPoE 24: I PADS R:0050.56a9.2c57 L:0050.56a9.de0d 3546
Gi2.546
02:22:09.144: IN PADS from PPPoE Session
02:22:09.149: PPPoE: Virtual Access interface obtained.
02:22:09.149: PPPoE : encap string prepared

```

```

R4#debug pppoe event
R4#debug ppp negotiation
R4#debug l2tp brief
R4#debug vpdn event
02:22:06.423: PPPoE 0: I PADI R:0050.56a9.de0d L:ffff.ffff.ffff 3546 Gi2.546
02:22:06.424: Service tag: NULL Tag
02:22:06.424: PPPoE 0: O PADO, R:0050.56a9.2c57 L:0050.56a9.de0d 3546 Gi2.546
02:22:06.424: Service tag: NULL Tag
02:22:08.471: PPPoE 0: I PADR R:0050.56a9.de0d L:0050.56a9.2c57 3546 Gi2.546
02:22:08.471: Service tag: NULL Tag
[snip]
02:22:08.471: [30]PPPoE 24: O PADS R:0050.56a9.de0d L:0050.56a9.2c57 Gi2.546

```

Next, the CPE and LAC begin the LCP negotiation within PPP. This exchange is also limited to the CPE and LAC as shown in the debugs. The messages within the two concurrent CONFREQ/CONFACK conversations are highlighted in yellow and green for clarity. So far, this is nothing new.

```

! CSR6
02:22:09.150: Vi2 PPP: Using dialer call direction
02:22:09.150: Vi2 PPP: Treating connection as a callout

```

```

02:22:09.150: Vi2 PPP: Session handle[1000001D] Session id[29]
02:22:09.150: Vi2 LCP: Event[OPEN] State[Initial to Starting]
02:22:09.150: Vi2 PPP: No remote authentication for call-out
02:22:09.150: Vi2 LCP: O CONFREQ [Starting] id 1 len 14
02:22:09.150: Vi2 LCP: MRU 1492 (0x010405D4)
02:22:09.150: Vi2 LCP: MagicNumber 0x2F35DBB0 (0x05062F35DBB0)
02:22:09.150: Vi2 LCP: Event[UP] State[Starting to REQsent]
02:22:09.152: Vi2 LCP: I CONFREQ [REQsent] id 1 len 18
02:22:09.153: Vi2 LCP: MRU 1492 (0x010405D4)
02:22:09.153: Vi2 LCP: AuthProto PAP (0x0304C023)
02:22:09.153: Vi2 LCP: MagicNumber 0x2AB61AF0 (0x05062AB61AF0)
02:22:09.153: Vi2 LCP: O CONFACK [REQsent] id 1 len 18
02:22:09.153: Vi2 LCP: MRU 1492 (0x010405D4)
02:22:09.153: Vi2 LCP: AuthProto PAP (0x0304C023)
02:22:09.153: Vi2 LCP: MagicNumber 0x2AB61AF0 (0x05062AB61AF0)
02:22:09.153: Vi2 LCP: Event[Receive ConfReq+] State[REQsent to ACKsent]
02:22:09.153: Vi2 LCP: I CONFACK [ACKsent] id 1 len 14
02:22:09.153: Vi2 LCP: MRU 1492 (0x010405D4)
02:22:09.153: Vi2 LCP: MagicNumber 0x2F35DBB0 (0x05062F35DBB0)

```

! CSR4

```

02:22:08.471: ppp30 PPP: Using vpn set call direction
02:22:08.471: ppp30 PPP: Treating connection as a callin
02:22:08.471: ppp30 PPP: Session handle[8000001E] Session id[30]
02:22:08.471: ppp30 LCP: Event[OPEN] State[Initial to Starting]
02:22:08.471: ppp30 PPP LCP: Enter passive mode, state[Stopped]
02:22:08.480: ppp30 LCP: I CONFREQ [Stopped] id 1 len 14
02:22:08.480: ppp30 LCP: MRU 1492 (0x010405D4)
02:22:08.480: ppp30 LCP: MagicNumber 0x2F35DBB0 (0x05062F35DBB0)
02:22:08.480: ppp30 LCP: O CONFREQ [Stopped] id 1 len 18
02:22:08.480: ppp30 LCP: MRU 1492 (0x010405D4)
02:22:08.480: ppp30 LCP: AuthProto PAP (0x0304C023)
02:22:08.480: ppp30 LCP: MagicNumber 0x2AB61AF0 (0x05062AB61AF0)
02:22:08.481: ppp30 LCP: O CONFACK [Stopped] id 1 len 14
02:22:08.481: ppp30 LCP: MRU 1492 (0x010405D4)
02:22:08.481: ppp30 LCP: MagicNumber 0x2F35DBB0 (0x05062F35DBB0)
02:22:08.481: ppp30 LCP: Event[Receive ConfReq+] State[Stopped to ACKsent]
02:22:08.482: ppp30 LCP: I CONFACK [ACKsent] id 1 len 18
02:22:08.482: ppp30 LCP: MRU 1492 (0x010405D4)
02:22:08.482: ppp30 LCP: AuthProto PAP (0x0304C023)
02:22:08.482: ppp30 LCP: MagicNumber 0x2AB61AF0 (0x05062AB61AF0)

```

Next, CSR6 tries to authenticate via PAP. At this point, LCP is open and CSR6 is waiting for a response back from the LNS, so the LAC cannot respond to this. CSR4 claims there is no method list to authenticate this user after the AUTH-REQ is received from CSR6.

! CSR6

```

02:22:09.153: Vi2 LCP: Event[Receive ConfAck] State[ACKsent to Open]

```

```

02:22:09.169: Vi2 PPP: Phase is AUTHENTICATING, by the peer
02:22:09.169: Vi2 PAP: Using hostname from interface PAP
02:22:09.169: Vi2 PAP: Using password from interface PAP
02:22:09.169: Vi2 PAP: O AUTH-REQ id 1 len 20 from "R6@lab.local"
02:22:09.169: Vi2 LCP: State is Open

! CSR4
02:22:08.482: ppp30 LCP: Event[Receive ConfAck] State[ACKsent to Open]
02:22:08.498: ppp30 PPP: Phase is AUTHENTICATING, by this end
02:22:08.498: ppp30 LCP: State is Open
02:22:08.504: ppp30 PAP: I AUTH-REQ id 1 len 20 from "R6@lab.local"
02:22:08.504: ppp30 PAP: Authenticating peer R6@lab.local
02:22:08.504: ppp30 PPP: Phase is FORWARDING, Attempting Forward
02:22:08.504: PPPoE : Method list does not exists

```

At this point, CSR4 knows that LCP was successful and that it must generate an L2TP tunnel to the LNS to complete the authentication process. The VPDN process matches the LAC group and initiates the tunnel to 10.45.10.10 as user LAC45. The L2TP tunnels comes up shortly thereafter. Most of the L2TP and VPDN debug isn't very helpful, but the key parts are shown below. CSR4 "forwards" the PPP session into the tunnel after negotiating LCP (to some extent) and forwarding the other PPP protocols (PAP, etc) onto the LNS.

```

! CSR4
02:22:08.505: VPDN L2X: ADD class VPDN group LAC ip addr 10.45.10.10 client
LAC45 (group LAC)
[snip]
02:22:08.505: [30]PPPoE 24: State LCP_NEGOTIATION      Event PPP FORWARDING
02:22:08.505: [30]PPPoE 24: Segment (SSS class): UPDATED
02:22:08.505: [30]PPPoE 24: SSS switch updated
02:22:08.511: L2TP 0001E:080E6:0000F3DE: APP<-L2TP: remote circuit status
sock 81000018 serv 000080E4 UP
[snip]
02:22:08.514: L2TP 0001E:080E6:0000F3DE: APP<-L2TP: Connected sock 81000018
serv 000080E4
02:22:08.515: VPDN Received L2TUN socket message Connected
02:22:08.515: VPDN uid:30 VPDN session up
02:22:08.515: ppp30 PPP: Phase is FORWARDED, Session Forwarded
02:22:08.515: [30]PPPoE 24: State LCP_NEGOTIATION      Event PPP FORWARDED
02:22:08.515: [30]PPPoE 24: Connected Forwarded

```

```

R10#debug ppp negotiation
R10#debug l2tp brief
R10#debug vpdn event
02:22:07.336: VPDN L2X: ADD class AAA author, group "LNS" (group LNS)
02:22:07.338: L2TP ____:081B1:00000A51: APP<-L2TP: Incoming sock 00000000
serv 000081B3
02:22:07.338: VPDN Received L2TUN socket message Incoming
02:22:07.338: VPDN uid:88 L2TUN socket session accept requested

```

```
[snip]
02:22:07.338: L2TP 00058:081B1:00000A51: APP->L2TP: Setup dataplane sock
2200001A serv 000081B3 replied on same socket
02:22:07.345: L2TP 00058:081B1:00000A51: APP<-L2TP: Connected sock 2200001A
serv 000081B3
02:22:07.345: VPDN Received L2TUN socket message Connected
02:22:07.345: VPDN uid:88 VPDN session up
```

CSR10 is now actively participating in the PPP connections with CSR6, the CPE. “FORCED” messages are exchanged within LCP to identify the handoff, and then the PAP process continues. The LNS then authenticates the user via PAP and sends the AUTH-ACK back towards the CPE. AT this point, the LAC does nothing except pass traffic back and forth.

```
! CSR10
02:22:07.345: ppp88 PPP: Phase is ESTABLISHING
02:22:07.345: ppp88 LCP: Event[Jam Start] State[Initial to Closed]
02:22:07.345: ppp88 LCP: I FORCED rcvd CONFACK len 18
02:22:07.346: ppp88 LCP: MRU 1492 (0x010405D4)
02:22:07.346: ppp88 LCP: AuthProto PAP (0x0304C023)
02:22:07.346: ppp88 LCP: MagicNumber 0x2AB61AF0 (0x05062AB61AF0)
02:22:07.346: ppp88 LCP: I FORCED sent CONFACK len 14
02:22:07.346: ppp88 LCP: MRU 1492 (0x010405D4)
02:22:07.346: ppp88 LCP: MagicNumber 0x2F35DBB0 (0x05062F35DBB0)
02:22:07.346: ppp88 LCP: Event[Jam UP] State[Closed to Open]
02:22:07.355: ppp88 PPP: Phase is FORWARDING, Attempting Forward
02:22:07.355: ppp88 LCP: State is Open
02:22:07.355: ppp88 PPP: Phase is AUTHENTICATING, Unauthenticated User
02:22:07.355: ppp88 PPP: Phase is FORWARDING, Attempting Forward
02:22:07.363: VPDN uid:88 Virtual interface created for R6@lab.local
bandwidth 1000000 Kbps
02:22:07.363: VPDN Vi3.1 Virtual interface created for R6@lab.local,
bandwidth 1000000 Kbps
02:22:07.363: L2TP 00058:081B1:00000A51: APP->L2TP: Session updated sock
2200001A serv 000081B3 replied on same socket
02:22:07.364: L2TP 00058:081B1:00000A51: APP<-L2TP: Dataplane up sock
2200001A serv 000081B3
02:22:07.364: VPDN Received L2TUN socket message Data UP
02:22:07.365: Vi3.1 PPP: Phase is AUTHENTICATING, Authenticated User
02:22:07.365: Vi3.1 PAP: O AUTH-ACK id 1 len 5
02:22:07.365: Vi3.1 PPP: No AAA accounting method list
02:22:07.365: Vi3.1 PPP: Phase is UP
```

The CPE receives this AUTH-ACK several milliseconds later since the VPDN/L2TP process took time. The remaining IPCP and IPV6CP debugs are not interesting since this is normal PPP negotiation at this point; the CPE is not aware of anything special and sees only a basic PPPoE session. Likewise, there is nothing else interesting to see on CSR10 since it is negotiating these protocols with CSR6.

```

! CSR6
02:22:09.210: Vi2 PAP: I AUTH-ACK id 1 len 5
02:22:09.210: Vi2 PPP: Phase is FORWARDING, Attempting Forward
02:22:09.210: Vi2 PPP: Queue IPCP code[1] id[1]
02:22:09.210: Vi2 PPP: Queue IPV6CP code[1] id[1]
02:22:09.211: Vi2 PPP: Phase is ESTABLISHING, Finish LCP
02:22:09.212: Vi2 PPP: Phase is UP
[snip]

```

We can verify the connectivity in stages. First, we verify that PPPoE is functional. The CPE sees the session as UP, but the LAC sees it as forwarded and not PTA as seen earlier. This makes sense because although the LAC “terminates” the PPPoE session, it is only terminating the transport; all of the PPP intelligent negotiation happens with the LNS. The MAC addresses shown below are the MACs of CSR6 (DE0D) and CSR4 (2C57), since the LNS is not aware that PPPoE is used.

```

R6#show pppoe session
      1 client session

```

Uniq ID	PPPoE	RemMAC	Port	VT	VA	State
	SID	LocMAC			VA-st	Type
N/A	24	0050.56a9.2c57	Gi2.546	Di6	Vi2	UP
		0050.56a9.de0d			UP	

```

R4#show pppoe session
      1 session in FORWARDED (FWDED) State
      1 session total

```

Uniq ID	PPPoE	RemMAC	Port	VT	VA	State
	SID	LocMAC			VA-st	Type
30	24	0050.56a9.de0d	Gi2.546	10	N/A	FWDED
		0050.56a9.2c57	VLAN:3546			

Next, we verify PPP connectivity. CSR6 and CSR10 show normal output; this makes sense because they negotiated all of the PPP upper-layer protocols required for communication. In passing, we can also see that an IPv4 address was issued to CSR6 from the local pool on the LNS via IPCP. The details on CSR10 reveal that LCP is “jammed”, meaning that it is being forced open not because it was negotiated with the peer, but because of another process. In this case, VPDN/L2TP is holding it open. There is otherwise nothing special about these PPP parameters.

```

R6#show ppp all

```

Interface/ID	OPEN+	Nego*	Fail-	Stage	Peer Address	Peer Name
Vi2	LCP+	IPCP+	IPV6CP+	LocalT	209.10.10.10	

```

R10#show ppp all

```

Interface/ID	OPEN+	Nego*	Fail-	Stage	Peer Address	Peer Name

```
Vi3.1          LCP+ PAP+ IPCP+ IPV6> LocalT 209.10.10.57 R6@lab.local
```

```
R10#show ppp interface vi3.1 | begin LCP:
```

```
Vi3.1 LCP: [Open] JAMMED
```

```
Our Negotiated Options
```

```
Vi3.1 LCP: MRU 1492 (0x010405D4)
```

```
Vi3.1 LCP: AuthProto PAP (0x0304C023)
```

```
Vi3.1 LCP: MagicNumber 0x2AB61AF0 (0x05062AB61AF0)
```

```
Peer's Negotiated Options
```

```
Vi3.1 LCP: MRU 1492 (0x010405D4)
```

```
Vi3.1 LCP: MagicNumber 0x2F35DBB0 (0x05062F35DBB0)
```

```
[snip]
```

CSR4 shows new information; it was initially involved in negotiating LCP and LCP was shown as “open” in the debugs, so it is shown as such. PAP never really completed since it was forwarded to the LN, so it is listed as “negotiating”. We can look at the details of this PPP session by hex ID since there isn’t an interface associated with it.

```
R4#show ppp all
```

Interface/ID	OPEN+ Nego*	Fail-	Stage	Peer Address	Peer Name
0x8000001E	LCP+ PAP*		Fwded	0.0.0.0	R6@lab.local

```
R4#show ppp id 8000001E | begin ^PPP Session
```

```
PPP Session Info
```

```
-----
```

```
Interface      : ppp30
```

```
PPP ID         : 0x8000001E
```

```
Phase          : FORWARDED
```

```
Stage          : Forwarded
```

```
Peer Name      : R6@lab.local
```

```
Peer Address   : 0.0.0.0
```

```
Control Protocols: LCP[Open] PAP*
```

```
Session ID     : 30
```

```
AAA Unique ID  : 41
```

```
SSS Manager ID : 0xCE000037
```

```
SIP ID         : 0x88000036
```

```
PPP_IN_USE     : 0x10
```

```
ppp30 LCP: [Open]
```

```
Our Negotiated Options
```

```
ppp30 LCP: MRU 1492 (0x010405D4)
```

```
ppp30 LCP: AuthProto PAP (0x0304C023)
```

```
ppp30 LCP: MagicNumber 0x2AB61AF0 (0x05062AB61AF0)
```

```
Peer's Negotiated Options
```

```
ppp30 LCP: MRU 1492 (0x010405D4)
```

```
ppp30 LCP: MagicNumber 0x2F35DBB0 (0x05062F35DBB0)
```

Checking the L2TP details on the LAC, we can see there is a single, locally-initiated tunnel from the LAC to the LNS. The tunnel actually uses UDP port 1701, not IP protocol 115, but the result is the same.

```
R4#show l2tp tunnel summary
L2TP Tunnel Information Total tunnels 1 sessions 1
LocTunID  RemTunID  Remote Name  State  Remote Address  Sessn L2TP Class/
Count VPDN Group
62075      39858      R10          est    10.45.10.10     1      LAC
```

```
R4#show l2tp tunnel transport
L2TP Tunnel Information Total tunnels 1 sessions 1
LocTunID  Type Prot  Local Address  Port  Remote Address  Port
62075      UDP  17    10.45.10.4     1701  10.45.10.10     1701
```

Within the control channel, a single session exists, where the user is R6 and the session relies on the tunnel seen earlier (ID 62075). This represents a “call” and one exists for each dial-in connection.

```
R4#show l2tp session brief
L2TP Session Information Total tunnels 1 sessions 1
LocID      TunID      Peer-address  State      Username, Intf/
sess/cir   Vcid, Circuit
62430      62075      10.45.10.10  est,UP     R6@lab.local, Gi2.546:3546
```

The information is similar on the LNS. The difference is that the remote name is LAC45, which was configured under the VPDN-group on the LAC and matched by the LNS. CSR10 sees the session to CSR6 via CSR4’s transport address. These 10.45.10.0/24 addresses are the tunnel endpoints.

```
R10#show l2tp tunnel summary
L2TP Tunnel Information Total tunnels 1 sessions 1
LocTunID  RemTunID  Remote Name  State  Remote Address  Sessn L2TP Class/
Count VPDN Group
39858      62075      LAC45        est    10.45.10.4      1      LNS
```

```
R10#show l2tp session brief
L2TP Session Information Total tunnels 1 sessions 1
LocID      TunID      Peer-address  State      Username, Intf/
sess/cir   Vcid, Circuit
2641      39858      10.45.10.4    est,UP     R6@lab.local, Vi3.1
```

The VPDN show commands are just wrappers for the L2TP commands (or whatever protocol is used). One quick example is shown below on the LNS, which offers no new information.

```
R10#show vpdn
L2TP Tunnel and Session Information Total tunnels 1 sessions 1
LocTunID  RemTunID  Remote Name  State  Remote Address  Sessn L2TP Class/
Count VPDN Group
39858      62075      LAC45        est    10.45.10.4      1      LNS
```


LocID	RemID	TunID	Username, Intf/ Vcid, Circuit	State	Last Chg	Uniq ID
2641	62430	39858	R6@lab.local, Vi3.1	est	00:44:34	88

With CSR6 connected, we will bring up CSR7 next. This time, we will enable more detailed L2TP debugs CSR10 (LNS) without enabling any PPPoE, PPP, or VPDN debugs. Since CSR6 and CSR7 are logically equivalent, we use this debugging approach for variety. This will allow us to see the details of the L2TP tunnel and session construction. The debugs are verbose so only the most critical parts are shown. First, the LAC determines that a tunnel is needed to 10.45.10.10, which is the LNS IP address identified in the VPDN group. Soon, the LAC determines its source IP address of 10.45.10.5 and uses L2TP over UDP, along with its local hostname LAC45, to initiate the tunnel. Because there are no existing sessions between these routers, a new L2TP control channel must be created. This requires the LAC (initiator) to send an SCCRP to the LNS.

```
R5#debug l2tp events
19:25:22.049: L2X _____:_____ : class [VPDN group LAC ip addr
10.45.10.10 client LAC45]
19:25:22.049: L2X _____:_____ : created
19:25:22.049: L2X _____:_____ : class [VPDN group LAC ip addr
10.45.10.10 client LAC45]
[snip]
19:25:22.049: L2TP 0000D:_____:_____ : L2TPoUDP session needed between
19:25:22.049: L2TP 0000D:_____:_____ : <unset>:0<->10.45.10.10:0
[snip]
19:25:22.049: L2TP _____:_____ : 10.45.10.5<->10.45.10.10
19:25:22.049: L2TP _____:_____ : with class: VPDN group LAC ip addr
10.45.10.10 client LAC45
19:25:22.049: L2TP _____:_____ : and group: "
19:25:22.049: L2TP _____:_____ : and group: "VPDN group LAC ip
addr 10.45.10.10 client LAC45..."
19:25:22.049: L2TP _____:_____ : and IP proto: L2TPoUDP
19:25:22.049: L2TP _____:_____ : and framing type: sync
19:25:22.049: L2TP _____:_____ : and bearer type: none
19:25:22.049: L2TP _____:_____ : and version: V2
19:25:22.049: L2TP _____:_____ : and local hostname: LAC45
[snip]
19:25:22.049: L2TP _____:_____ : Need to instigate control channel
[snip]
19:25:22.049: L2TP tn1 08037:0000AD60: Open sock 10.45.10.5:1701-
>10.45.10.10:1701
19:25:22.049: L2TP tn1 08037:0000AD60: FSM-CC ev Sock-Ready
19:25:22.049: L2TP tn1 08037:0000AD60: FSM-CC Wt-Sock->Wt-SCCRP
19:25:22.049: L2TP tn1 08037:0000AD60: FSM-CC do Tx-SCCRP
```

CSR10 receives the SCCRQ and continues signaling the control channel back to 10.45.10.5. The SCCRQ is processed so that LAC45 can be matched to a VPDN group. After that, the LNS sends the SCCRP back to the LAC.

```
R10#debug l2tp events
19:25:20.881: L2X tnl 101BA:_____ : Create logical tunnel
19:25:20.881: L2TP tnl 101BA:_____ : Create tunnel
19:25:20.881: L2TP tnl 101BA:_____ : version set to V2[1]
19:25:20.881: L2TP tnl 101BA:_____ : remote ip set to 10.45.10.5
19:25:20.881: L2TP tnl 101BA:_____ : local ip set to 10.45.10.10
19:25:20.881: L2TP tnl 101BA:00007B9E: FSM-CC ev Rx-SCCRQ
19:25:20.881: L2TP tnl 101BA:00007B9E: FSM-CC Idle->Proc-SCCRQ
19:25:20.881: L2TP tnl 101BA:00007B9E: FSM-CC do Rx-SCCRQ
19:25:20.881: L2TP tnl 101BA:00007B9E: ACCT(0000007B): UID allocated
19:25:20.881: L2X _____ : _____ : Tunnel author started for LAC45
[snip]
19:25:20.881: L2TP tnl 101BA:00007B9E: FSM-CC ev SCCRQ-OK
19:25:20.881: L2TP tnl 101BA:00007B9E: FSM-CC Proc-SCCRQ->Wt-SCCCN
19:25:20.881: L2TP tnl 101BA:00007B9E: FSM-CC do Tx-SCCRP
```

CSR5 receives the SCCRP, authenticates the LNS, and replies with a SCCCN to indicate the control channel tunnel is built. CSR10 receives the SCCCN, authenticates the LAC, and moves the control channel to the established state, just as CSR5 did when it sent the SCCCN.

```
! CSR5
19:25:22.055: L2TP tnl 08037:0000AD60: FSM-CC ev Rx-SCCRP
19:25:22.055: L2TP tnl 08037:0000AD60: FSM-CC Wt-SCCRP->Proc-SCCRP
19:25:22.055: L2TP tnl 08037:0000AD60: FSM-CC do Rx-SCCRP
19:25:22.055: L2TP tnl 08037:0000AD60: Tunnel Authentication success
19:25:22.055: L2TP tnl 08037:0000AD60: FSM-CC ev SCCRP-OK
19:25:22.055: L2TP tnl 08037:0000AD60: FSM-CC Proc-SCCRP->established
19:25:22.055: L2TP tnl 08037:0000AD60: FSM-CC do Tx-SCCCN
19:25:22.055: L2TP tnl 08037:0000AD60: Tunnel accounting send not possible
- no id
19:25:22.055: L2TP tnl 08037:0000AD60: Control channel up

! CSR10
19:25:20.885: L2TP tnl 101BA:00007B9E: FSM-CC ev Rx-SCCCN
19:25:20.885: L2TP tnl 101BA:00007B9E: FSM-CC Wt-SCCCN->Proc-SCCCN
19:25:20.885: L2TP tnl 101BA:00007B9E: FSM-CC do Rx-SCCCN
19:25:20.885: L2TP tnl 101BA:00007B9E: Tunnel Authentication success
19:25:20.885: L2TP tnl 101BA:00007B9E: FSM-CC ev SCCCN-OK
19:25:20.885: L2TP tnl 101BA:00007B9E: FSM-CC Proc-SCCCN->established
19:25:20.885: L2TP tnl 101BA:00007B9E: FSM-CC do Established
19:25:20.885: L2TP tnl 101BA:00007B9E: Tunnel accounting send not possible
- no mlist
19:25:20.885: L2TP tnl 101BA:00007B9E: Control channel up
```

Now, CSR5 must initiate a call to CSR10 for this particular PPP session. It sends an ICRQ to CSR10 who matches this to the VPDN application. It checks the access circuits (which I assume are the connected interfaces) then replies with an ICRP.

```
! CSR5
19:25:22.055: L2TP 0000D:08037:00000C6B: FSM-Sn do Tx-ICRQ

! CSR10
19:25:20.885: L2TP _____:101BA:00009B8F: FSM-Sn do Rx-ICRQ
19:25:20.885: L2TP _____:101BA:00009B8F: Chose application VPDN
19:25:20.885: L2TP _____:101BA:00009B8F: App type set to VPDN
19:25:20.885: L2TP _____:101BA:00009B8F: VPDN: process AVPs
19:25:20.885: L2TP _____:101BA:00009B8F: Set HA epoch to 0
19:25:20.885: L2TP _____:101BA:00009B8F: Local AC is now UP
19:25:20.885: L2TP _____:101BA:00009B8F: Remote AC is now UP
[snip]
19:25:20.886: L2TP 00059:101BA:00009B8F: FSM-Sn do Tx-ICRP
```

CSR5 receives the ICRP and replies with an ICCN. CSR5 now considers the session up once it checks the local/remote ACs as CSR10 did.

```
! CSR5
19:25:22.059: L2TP 0000D:08037:00000C6B: FSM-Sn do Rx-ICRP
19:25:22.059: L2TP 0000D:08037:00000C6B: MTU is 65535
19:25:22.059: L2TP 0000D:08037:00000C6B: Dataplane provisioned, segment 8249
19:25:22.059: L2TP 0000D:08037:00000C6B: Remote AC is now UP
19:25:22.059: L2TP 0000D:08037:00000C6B: Local AC is now UP
[snip]
19:25:22.060: L2TP 0000D:08037:00000C6B: FSM-Sn do Tx-ICCN
19:25:22.062: L2TP 0000D:08037:00000C6B: FSM-Sn ev Established
19:25:22.062: L2TP 0000D:08037:00000C6B: FSM-Sn in established
19:25:22.062: L2TP 0000D:08037:00000C6B: FSM-Sn do Established
19:25:22.062: L2TP 0000D:08037:00000C6B: Session up
19:25:22.062: L2TP 0000D:08037:00000C6B: 10.45.10.5<->10.45.10.10

! CSR10
19:25:20.891: L2TP 00059:101BA:00009B8F: FSM-Sn do Rx-ICCN
19:25:20.891: L2TP 00059:101BA:00009B8F: MTU is 65535
19:25:20.891: L2TP 00059:101BA:00009B8F: Dataplane provisioned, segment 12736
19:25:20.891: L2TP 00059:101BA:00009B8F: VPDN: process AVPs
19:25:20.891: L2TP 00059:101BA:00009B8F: APP<-L2TP: Connected sock 6900001B
serv 000101BC
19:25:20.891: L2TP 00059:101BA:00009B8F: FSM-Sn ev ICCN-OK
19:25:20.891: L2TP 00059:101BA:00009B8F: FSM-Sn Proc-ICCN->established
19:25:20.891: L2TP 00059:101BA:00009B8F: FSM-Sn do Established
19:25:20.891: L2TP 00059:101BA:00009B8F: Session up
19:25:20.891: L2TP 00059:101BA:00009B8F: 10.45.10.10<->10.45.10.5
```

We can verify this similarly as we did for CSR4 and CSR6. Checking the LAC (CSR5), we can see the PPPoE session is forwarded and that PPP LCP has negotiated completely. The remote PPPoE MAC (EA77) is CSR7 and the local MAC (DC63) is CSR5.

```
R5#show pppoe session
      1 session  in FORWARDED (FWDED) State
      1 session  total
```

Uniq ID	PPPoE SID	RemMAC LocMAC	Port	VT	VA VA-st	State Type
13	3	0050.56a9.ea77 0050.56a9.dc63	Gi2.557 VLAN:3557	10	N/A	FWDED

```
R5#show ppp all
Interface/ID OPEN+ Nego* Fail- Stage Peer Address Peer Name
-----
0xAB00000D LCP+ PAP* Fwded 0.0.0.0 R7@lab.local
```

The L2TP tunnel (control channel) is formed to CSR10. The session uses this control channel as it references the tunnel ID of 44384 and is also terminated on CSR10, the LNS. The session shows the username of R7 so it is clear that this session is associated with a given user.

```
R5#show l2tp tunnel summary
L2TP Tunnel Information Total tunnels 1 sessions 1
LocTunID RemTunID Remote Name State Remote Address Sessn L2TP Class/
Count VPDN Group
44384 31646 R10 est 10.45.10.10 1 LAC
```

```
R5#show l2tp session brief
L2TP Session Information Total tunnels 1 sessions 1
LocID TunID Peer-address State Username, Intf/
sess/cir Vcid, Circuit
3179 44384 10.45.10.10 est,UP R7@lab.local, Gi2.557:3557
```

The LNS shows two L2TP tunnels and two sessions. There will always be one tunnel per LAC and one session per user; the number of tunnels will always be less than or equal to the number of users as a result. Since the tunnels used the same remote name for the control channel, the only way to tell the difference is by examining the remote IP address.

```
R10#show l2tp tunnel
L2TP Tunnel Information Total tunnels 2 sessions 2
LocTunID RemTunID Remote Name State Remote Address Sessn L2TP Class/
Count VPDN Group
31646 44384 LAC45 est 10.45.10.5 1 LNS
39858 62075 LAC45 est 10.45.10.4 1 LNS
```

```
R10#show l2tp session brief
```

```
L2TP Session Information Total tunnels 2 sessions 2
LocID      TunID      Peer-address      State      Username, Intf/
sess/cir   Vcid, Circuit
39823      31646      10.45.10.5        est,UP     R7@lab.local, Vi3.2
2641      39858      10.45.10.4        est,UP     R6@lab.local, Vi3.1
```

To create additional calls through a LAC, we can add a new dialer to CSR7. We won't use this for routing or anything intelligent; we don't even have to configure IPv4 or IPv6. Just building the PPPoE session to the LAC is enough to cause a new L2TP session to establish. L2TP achieves scalability by mapping new sessions to existing tunnels between a common set of endpoints.

```
! CSR7
interface Dialer77
  mtu 1492
  encapsulation ppp
  dialer pool 77
  dialer idle-timeout 0
  dialer persistent
  ppp pap sent-username R77@lab.local password 0 R77

interface GigabitEthernet2.557
  pppoe-client dial-pool-number 77

! CSR10
username R77@lab.local password 0 R77
```

Now, the LAC sees two PPPoE sessions between the same pair of MAC addresses. The only difference is the session ID which is included in the encapsulation string as a demultiplexer.

```
R5#show pppoe session
  2 sessions in FORWARDED (FWDED) State
  2 sessions total
```

Uniq ID	PPPoE SID	RemMAC LocMAC	Port	VT	VA VA-st	State Type
13	3	0050.56a9.ea77	Gi2.557	10	N/A	FWDED
		0050.56a9.dc63	VLAN:3557			
14	4	0050.56a9.ea77	Gi2.557	10	N/A	FWDED
		0050.56a9.dc63	VLAN:3557			

There is still only a single L2TP control channel, but there are two sessions that rely on it now. The sessions have different local IDs and usernames, but otherwise go to the same LNS.

```
R5#show l2tp tunnel
L2TP Tunnel Information Total tunnels 1 sessions 2
LocTunID  RemTunID  Remote Name  State  Remote Address  Sessn L2TP Class/
Count VPDN Group
```

```
44384      31646      R10          est      10.45.10.10      2      LAC
```

```
R5#show l2tp session brief
```

```
L2TP Session Information Total tunnels 1 sessions 2
```

LocID	TunID	Peer-address	State	Username, Intf/ sess/cir	Vcid, Circuit
21902	44384	10.45.10.10	est,UP	R77@lab.local,	Gi2.557:3557
3179	44384	10.45.10.10	est,UP	R7@lab.local,	Gi2.557:3557

As expected, the LNS sees two control channels (one per LAC) but three sessions (one per user). The sessions for R7 and R77 use tunnel ID 31646 which maps to CSR5, the LAC with two calls behind it. For variety, we use the “show vpdn” command, which displays the same L2TP information arrayed differently. Each control channel is displayed separately with all supported sessions beneath it. The control-channel to 10.45.10.5 has 2 sessions which are shown next. Then, the control-channel to CSR4 is shown with its session as well. The usernames are shown with the sessions for clarity.

```
R10#show vpdn
```

```
L2TP Tunnel and Session Information Total tunnels 2 sessions 3
```

LocTunID	RemTunID	Remote Name	State	Remote Address	Sessn Count	L2TP VPDN	Class/Group
31646	44384	LAC45	est	10.45.10.5	2	LNS	

LocID	RemID	TunID	Username, Intf/ Vcid, Circuit	State	Last Chg	Uniq ID
39823	3179	31646	R7@lab.local, Vi3.2	est	00:45:25	89
39190	21902	31646	R77@lab.local, Vi3.3	est	00:04:22	90

LocTunID	RemTunID	Remote Name	State	Remote Address	Sessn Count	L2TP VPDN	Class/Group
39858	62075	LAC45	est	10.45.10.4	1	LNS	

LocID	RemID	TunID	Username, Intf/ Vcid, Circuit	State	Last Chg	Uniq ID
2641	62430	39858	R6@lab.local, Vi3.1	est	17:48:38	88

At this point, we can validate the IPv4 and IPv6 unicast routing. We saw earlier that IPv4 addresses were properly exchanged using IPCP by checking the PPP details. We will verify the routing table by checking connected host routes on CSR6 and CSR7. These address were issued from the LNS’ local pool.

```
R6#show ip route connected | include Dialer
```

```
C      209.10.10.10 is directly connected, Dialer6
C      209.10.10.61 is directly connected, Dialer6
```

```
R7#show ip route connected | include Dialer
```

```
C      209.10.10.10 is directly connected, Dialer7
C      209.10.10.59 is directly connected, Dialer7
```

Additionally, we verify that both CSR6 and CSR7 were able to receive IPv6 prefixes from DHCPv6. They are applied to the CPE LAN interfaces to support SLAAC.

```
R6#show ipv6 general-prefix
IPv6 Prefix PPPOE_ISP_PREFIX, acquired via DHCP PD
2001:192:168:A1::/64 Valid lifetime 2527643, preferred lifetime 540443
GigabitEthernet2.564 (Address command)
```

```
R7#show ipv6 general-prefix
IPv6 Prefix PPPOE_ISP_PREFIX, acquired via DHCP PD
2001:192:168:A0::/64 Valid lifetime 2589055, preferred lifetime 601855
GigabitEthernet2.574 (Address command)
```

Both CSR6 and CSR7 should have IPv4 and IPv6 default routes to the LNS as well. The IPv4 default route was automatically added by IPCP, and the IPv6 default route was added by IPv6 ND (SLAAC).

```
R6#show ip route 0.0.0.0
Routing entry for 0.0.0.0/0, supernet
  Known via "static", distance 1, metric 0, candidate default path
  Routing Descriptor Blocks:
  * 209.10.10.10
    Route metric is 0, traffic share count is 1
```

```
R6#show ipv6 route ::/0
Routing entry for ::/0
  Known via "ND", distance 2, metric 0
  Route count is 1/1, share count 0
  Routing paths:
  FE80::21E:14FF:FE6F:8300, Dialer6
    Last updated 00:07:58 ago
```

```
R7#show ip route 0.0.0.0
Routing entry for 0.0.0.0/0, supernet
  Known via "static", distance 1, metric 0, candidate default path
  Routing Descriptor Blocks:
  * 209.10.10.10
    Route metric is 0, traffic share count is 1
```

```
R7#show ipv6 route ::/0
Routing entry for ::/0
  Known via "ND", distance 2, metric 0
  Route count is 1/1, share count 0
  Routing paths:
  FE80::21E:14FF:FE6F:8300, Dialer7
    Last updated 00:08:46 ago
```

A quick confirmation of routing on the LNS is helpful as well. The routing table is not clear as to which IPv4 address was assigned to which client, so we can check the PPP IPCP details. Thanks to PAP, we can

see the username in the PPP summary, and conclude that Vi3.1 is for R6 and Vi3.2 is for R7. The third session to “R77” uses Vi3.3 but failed to negotiate IPCP, has no IPv4 address, and thus is not used for IPv4 routing.

```
R10#show ip route connected | include Virtual-Access
C      209.10.10.59/32 is directly connected, Virtual-Access3.2
C      209.10.10.61/32 is directly connected, Virtual-Access3.1
```

```
R10#show ppp all
```

Interface/ID	OPEN+	Nego*	Fail-	Stage	Peer Address	Peer Name
Vi3.1	LCP+	PAP+	IPCP+	IPV6>	LocalT	209.10.10.61 R6@lab.local
Vi3.3	LCP+	PAP+	IPCP-	IPV6>	LocalT	0.0.0.0 R77@lab.local
Vi3.2	LCP+	PAP+	IPCP+	IPV6>	LocalT	209.10.10.59 R7@lab.local

We also confirm that the LNS installed static routes for the two IPv6 prefixes issued to the CPEs via DHCPv6 PD. 2001:192:168:A0::/64 was issued to CSR7 and 2001:192:168:A1::/64 was issued to CSR6, according to the outgoing interfaces in the routing table. This is consistent with the CPE verifications we did earlier. These static routes are redistributed into ISIS, and we can verify this in the ISIS LSPDB.

```
R10#show ipv6 route static | begin ^S
S      2001:192:168:A0::/64 [1/0]
      via FE80::21E:49FF:FECA:A400, Virtual-Access3.2
S      2001:192:168:A1::/64 [1/0]
      via FE80::21E:BDFF:FE69:6200, Virtual-Access3.1
```

```
R10#show isis database detail level-2 R10-00.00 | include MT-IPv6
Metric: 10          IS (MT-IPv6) XRv1.00
Metric: 0           IPv6 (MT-IPv6) 2001:192:168:A0::/64
Metric: 0           IPv6 (MT-IPv6) 2001:192:168:A1::/64
```

Note that the LACs play no role in routing whatsoever. Looking briefly at CSR4, it generally only has routes to the LNS; in this case, a connected LAN. For IPv6, it has no routes at all, and it is not even enabled for IPv6. There is no reason for it to be unless the L2TP endpoint was an IPv6 address, which may not be supported in all versions of IOS.

```
R4#show ip route | begin Gateway
Gateway of last resort is not set

      10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C      10.45.10.0/24 is directly connected, GigabitEthernet2.545
L      10.45.10.4/32 is directly connected, GigabitEthernet2.545

R4#show ipv6 route | begin Applic
a - Application
L      FF00::/8 [0/0]
```


via Null0, receive

Finally, we confirm connectivity from the client simulator (CSR1) for both CPE routers using both IPv4 and IPv6.

```
R1#ping vrf 6 13.144.2.1
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 6/9/19 ms

R1#ping vrf 7 13.144.2.1
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 5/9/21 ms

R1#ping vrf 6 2bad:beef:13:dddd::d
Success rate is 100 percent (5/5), round-trip min/avg/max = 8/19/61 ms

R1#ping vrf 7 2bad:beef:13:dddd::d
Success rate is 100 percent (5/5), round-trip min/avg/max = 7/8/12 ms
```

Additional Reading – Reference configurations “pppoe-arch”

1.3 MEF Ethernet Services Definitions (MEF 6.2)

Ethernet services as defined by MEF are shown below. For each type of service, there are port-based (private) and VLAN-based (virtual private) variations.

E-LINE: “Point-to-point” Ethernet virtual circuit. It functions just like an Ethernet cable in terms of its layer 2 capabilities. Only the low-level, layer-1 signaling is not shared between customer devices across an E-LINE.

- a. **EPL:** Ethernet Private Line. Simple P2P service with low delay and low loss. No service multiplexing or CoS applications is allowed (except for basic CIR/PIR policing). This is literally a P2P connection between two sites and multiple C-VLAN tags are not allowed for service mapping. Bundling is also disallowed and the maximum number of EVC is fixed at 1, but there is no limit on source MAC addresses that can be used. Technically, each node at the end of the link is identified as a “root” node and a maximum of two UNIs can exist in the EPL. The CE-VLAN IDs, including their CoS markings, must be preserved across the EVC.
- b. **EVPL:** Ethernet Virtual Private Line. P2P Ethernet service where service multiplexing (more than one EVC) is allowed. The individual EVCs can be given special CoS parameters. An individual EVC would be created by allowing multiple C-VLAN IDs and mapping each to a different EVC. It’s like a collection of P2P links, but the EVCs are separate P2P links, so not like a LAN. This is loosely analogous to OSPF P2MP network type in a hub-and-spoke design. Many of the same capabilities/limitations apply for EVPL as they did for EPL, except bundling is possible. The CE-VLAN IDs, including their CoS markings, do not have to be preserved across the EVC, which implies C-VLAN rewrite operations are permitted.

E-LAN: MP2MP EVC. It's like an emulated LAN design. Because this is a true LAN, every node is set to have "root" status so it can talk to every other node directly. The maximum number of UNIs is three or greater, otherwise it would just be an E-LINE. The EVC type is classified as "Multipoint-to-multipoint".

- a. **EP-LAN:** LAN services with similar behavior as EPL. Bundling and service multiplexing are disabled as the LAN service is port-based, and an EVC maximum of 1 is imposed. CE-VLAN ID and CoS values must be preserved across the private LAN service.
- b. **EVP-LAN:** LAN services with similar behavior as EVPL. Bundling is possible and service multiplexing is enabled as the LAN service is VLAN-based with no EVC maximum. However, C-VLAN ID and CoS values don't have to be preserved across the private LAN service.

E-TREE: P2MP E-LAN service where the leaves/spokes can communicate with the root/hub but not with one another. It's like an emulated private VLAN design. The most common use is in franchise operations where small offices/sites need not communicate directly. Technically, it could also be a MP2MP partial mesh if there are multiple root nodes. The root nodes must be set to "root" mode so they can talk to all nodes. The remaining nodes are placed in "leaf" mode so they only have connectivity to roots. An E-TREE must have at least two leaves; the only exception is starting with two "roots" and adding leaves later. The EVC is classified as "rooted-multipoint".

- a. **EP-TREE:** This has the same characteristics of an EP-LAN with the additional reachability restrictions determined by the placement of root/leaf nodes.
- b. **EVP-TREE:** This has the same characteristics of an EVP-LAN with the additional reachability restrictions determined by the placement of root/leaf nodes.

1.4 Platform Architecture

This section focuses primarily on the ASR9000 architecture and its forwarding processes.

1.4.1 Route-Switch Processor (RSP) and Route Processor (RP)

The ASR9000 series router will have a set of RSPs or RPs, depending on its size. The purpose of the RSP/RP is to be the centralized control-plane of the router. Packets punted by line-cards or information directed to the router itself (such as BGP updates) is processed here. They are not designed for forwarding traffic, but are capable of doing it slowly. These cards also contain coaxial ports for precision time sources/methods (SyncE, IEEE 1588, GPS, etc). They also contain the traditional management interfaces such as USB, console, auxiliary, and network management Ethernet ports. RSP cards can be clustered together (there are dedicated ports on the RSP for it) which allow multiple routers to be tied together to create one large logical entity. The difference between an RSP and RP is the presence or absence of the switching fabric. To connect remote linecards, an internal switching fabric is used as a high-speed transport between ingress and egress linecards. On platforms that use RSPs, such as ASR9001, ASR9006, and ASR9010, this switching fabric is part of the RSP. This doesn't mean the traffic is processed-switched nor punted to the routing engine, just that the high-speed backplane is physically on the card. On the ASR9922, the largest router, RPs are used, not RSPs. The RPs only serve the routing control-plane function since 7 additional fabric cards can be added for additional resiliency. This large router decouples the switching fabric from the routing engine from a hardware perspective.

1.4.2 Line cards (LC)

Linecards are modules that can be added to a router to add specific capabilities. Typically, linecards will be densely populated with low speed ports, sparsely populated with high speeds ports, or somewhere in between. They also can support additional media types such as frame-relay, ATM, SONET/SDH, T1/T3, E1/E3, etc. Linecards have their own set of computing resources which are optimized for forwarding frames in and out. Most linecards have some limited control-plane intelligence so that basic functions like ARP, BFD, ACL, QoS, and Netflow can be offloaded from the RSP/RP. The linecards have network processors (NP) which are mapped to a number of individual ports. Often times the port to NP ratio is between 3:1 and 6:1, which means that there are still a lot of NPs on a linecard to process transit network traffic. The NPs are connected to fabric interface ASIC (FIA) which are self-explanatory; they connect linecards to the switching fabric. This is how inter-LC traffic transits across a router. Typically there will be fewer FIAs than NPs, just as there are fewer NPs than physical ports, so there is some hierarchy to the forwarding model within the LC. The NP is not the same as the LC-CPU, which is only consulted occasionally as discussed later.

1.4.3 Switching fabric / backplane and forwarding model

The switching fabric is used to connect all components of a router. It is physically present on RSPs in the ASR9010 and smaller, yet is a dedicated card on the ASR9922. Unlike some other routers, there are no “shortcut” paths where linecards can exchange information directly. All packets must traverse the switch fabric as it is an integral part of the 3 stage forwarding process. Shortcuts are unnecessary since the bandwidth of the switch fabric is so high, which simplifies the design. Forwarding happens in 3 stages: ingress linecard, switch fabric, and egress linecard. When packets arriving at the ingress LC, the traffic is classified one of two ways: transit, host, or exception.

1. Transit traffic is simple as the LC will consult its FIB on the NP (LCs have a copy of the FIB so the RSP is not interrupted), perform a rewrite (MAC address, MPLS label, etc), and forward to the switch fabric via the FIA. This is true even if the egress LC is the same as the ingress LC. This “hairpin” isn’t a big deal; there are no shortcuts within a linecard in the ASR9000 platforms. Once on the switch fabric, which is the second stage, the packet is forwarded to the appropriate egress LC. Even if traffic is going between two adjacent ports that share a linecard’s NP, the traffic still transits the switching fabric. Once on the fabric, the second stage of forwarding occurs as the traffic is delivered to the proper egress LC. The third stage involve the egress LC delivering the packet to the proper NP, and ultimately the proper physical port. Sometimes this is described as “2 phase” forwarding where only the ingress and egress LC decisions are made, and the switching fabric forwarding doesn’t count as a stage. An additional feature of this feature is that egress NPs can signal “backpressure” to the upstream FIA during periods of congestion. The ingress FIA facing the fabric can buffer traffic as necessary.
2. Host-traffic destined “for us” will either be punted to the LC-CPU or RSP, depending on the update. An ARP update would go to the LC-CPU while a BGP update would go to the RSP. This traffic is also subject to LPTS, which is described later. Link-local traffic, such as IPv6 ND or IGPs, also falls into this category. Any kind of management traffic, such as telnet/SSH, is also forwarded to the RSP/RP for processing since the intent is to manage the router itself.

3. Exception traffic is traffic that should have been transit traffic but wasn't due to some condition, such as TTL expiration. This is forwarded to the LC-CPU. The only exception to this rule is IGMP snooping, which is punted to the RSP CPU.

1.4.4 Multicast forwarding and hierarchical replication

The ASR9000 replicates multicast traffic as close to the egress point as possible, even inside the router itself. When a multicast packet is received, the ingress LC tags the packet (in software, no change to the packet) with a fabric group ID and multicast group ID. When the packet ultimately arrives at the fabric, because all transit packets must cross the switching fabric, the FGID is used to determine which egress LCs need the packet. The fabric then replicates the packet as necessary to each egress LC, keeping in mind the ingress LC might be in the set of egress LCs as well. The LC switch fabric uses the MGID to replicate the packet to downstream FIAs, and the FIAs use the MGID as well to replicate the packet down to egress NPs. The egress NPs consult their MFIBs to replicate the packets to multiple ports, if necessary. In this way, there are 4 stages of replication: switching fabric, LC fabric, FIA, and egress NP.

1.4.5 Satellite operations (remote linecards)

The ASR9000 platforms also support the concept of a remote linecard, known as the ASR9000v. This is somewhat similar to a Nexus 2000 fabric extender (FEX) which connects to a Nexus 5000 switch. The ASR9000v can be connected to the ASR9000 using up to (4) 10 GbE links (can be bundled). The remaining (44) ports can support 1 GbE connections. There is no local switching done on the satellite, much like the N2K FEX. All operations, to include QoS, is done on the host device (ASR9000 router). The ASR9000v discovery mechanism works like CDP since "NV" has its own discovery mechanism. The satellite heartbeat is once per second to ensure it remains reachable. The satellite is configured from the ASR9000 and certain satellite ports can be assigned to specific 10 GbE uplinks or ether-channel ports that connect the host to the satellite.

3.1 WAN technologies

3.1.1 Packet over SONET/SDH

Synchronous Optical Networking (SONET) and Synchronous Digital Hierarchy (SDH) send multiple digital signals over fiber concurrently. SONET is prevalent in the USA and Canada and is defined by ANSI. SDH is prevalent everywhere else and is an ITU standard. One major difference in these technologies is that the header information (such as IP or Ethernet headers) are not necessarily transmitted first, but are interleaved with the payload at layer 1. Some bytes from the header are sent, then bytes from the payload, and this process is repeated multiple times until the packet is totally sent. Graphically, the packet would look like a rectangle if each of the transmission were placed atop one another, and this is how SONET reassembles the packet (at a high level).

SONET and SDH both support many low-level alarms which obviate overlay failure detection techniques like BFD. Many of these alarms are self explanatory, but I add a brief comment in parenthesis after some of them.

```
RTR12410-1(config-if)#pos report ?  
all      all Alarms/Signals
```

b1-tca B1 BER threshold crossing alarm
 b2-tca B2 BER threshold crossing alarm
 b3-tca B3 BER threshold crossing alarm
 lais Line Alarm Indication Signal (if SLOF or SLOS, set at remote end)
 lrdi Line Remote Defect Indication
 pais Path Alarm Indication Signal (defect noticed on peer signal; minor)
 plop Path Loss of Pointer
 prdi Path Remote Defect Indication (issue with a node two sites away)
 rdool Receive Data Out Of Lock
 sd-ber LBIP BER in excess of SD threshold
 sf-ber LBIP BER in excess of SF threshold
 slof Section Loss of Frame (errors in the framing pattern/alignment)
 slos Section Loss of Signal (0->1 or 1->0 bit transitions not seen)

SONET keepalives are also independent between peers. One side can have it enabled, and the other can have it disabled. The timers can also be mismatched. The CRC for a POS interface defaults to 16 bits but can be increased to 32 for extra resiliency. The automatic protection switching (APS) feature allows for a pair of SONET links to serve as active/standby. The Working (W) link is backed up by the Protect (P) link and the failover time is about 50 ms. The two links must be in the same APS group. The routers communicate APS information using the Protect Group Protocol (PGP). These concepts of working/protect links are also extended to MPLS Transport Profile (MPLS-TP) as discussed later.

For OAM functionality, a Data Communication Channel (DCC) exists over SONET/SDH as well. It can also be used for remote provisioning over a SONET link. This is somewhat similar to Ethernet LMI (E-LMI) and other OAM protocols add to Ethernet to support service provider operations.

The chart below summarizes the speeds for common circuit names. Remember that SONET OC-1 is 51.84 Mbps and SDH STM-1 is 155.52 Mbps, and achieve the rest of the number is simple multiplication. For example, an OC-3 is 3 OC-1s, and $51.84 * 3 = 155.52$. An OC-24 is 8 OC-3s and an STM-64 is 16 OC-4s, for example. The OC designation refers to a signal in its optimal form and the frame format represents the size of data carried. An OC-3, technically speaking, consists of 3 STS-1s, not 3 OC-1s. Of note, a SONET OC-192 is often time compared against 10 GbE because their speeds are almost identical (cyan).

SONET OC Level	SONET frame format	SDH level and frame format	Line rate
OC-1	STS-1 (810 bytes)	STM-0	51.84 Mbps
OC-3	STS-3	STM-1	155.52 Mbps
OC-12	STS-12	STM-4	622.08 Mbps
OC-24	STS-24	N/A	1.244 Gbps
OC-48	STS-48	STM-16	2.488 Gbps
OC-192	STS-192	STM-64	9.953.28 Gbps
OC-768	STS-768	STM-256	39.813 Gbps

3.1.2 T1/E1 and T3/E3

T-carrier and E-carrier technologies have been around for many years and are typical WAN circuit designations. They are time division multiplexing (TDM) based and are similar in many ways. Each of

them are a collection of 64 kbps channels, called DS0s, which are aggregated into a larger bundle to form these specifications. Each DS0 carries 8 bits every 125 us, which is 64 kbps. T-carriers are common in North America, Japan, and South Korea. E-carriers are common in Europe. A T1 consists of 24 DS0s while an E1 consists of 32, yielding 1.536 Mbps and 2.048 Mbps, respectively. However, once all 24 of the DS0s carry their data, an extra framing bit as adding for OAM functionality, yielding a 193 bit T-1 frame. For this reason, a T1 is said to have 1.544 Mbps line rate speed. The logic is extended to T3 circuits where even more framing bits are used, and the logic is similar for E3. The math behind it is beyond the scope of this summary.

Certain network devices may allow the network administrator to break out DS0s individual for various purposes. For example, the 8 kbps channels could each carry a voice phone call, and some DS0s within a T1/E1 could be dedicated for that. Remaining channels could be bonded for data transmission. The initial function of these circuits was to carry phone calls.

The chart below summarizes the speeds for the 4 main circuit types above.

Circuit / Carrier	Composition	Data rate
T1 / DS1	24 DS0 + 1 frame bit	1.544 Mbps
T3 / DS3	28 DS1 + 69 frame bit	44.736 Mbps
E1	30 DS0 (or E0)	2.048 Mbps
E3	16 E1 + frame bits	34.368

3.1.3 Dense Wavelength Division Multiplexing (DWDM)

Wavelength division multiplexing (WDM) is a method of transmitting many different wavelengths of light onto a single fiber media. This is the same thing as frequency division multiplexing (FDM), which is a term used in radio frequency networking, but the concept is the same. WDM is typically used when referring to optical carriers. Dense WDM is an enhancement to the original WDM (coarse WDM) to stuff more wavelengths onto a single medium, which increases the bandwidth. The two ends of a link will have a multiplexer and demultiplexer to combine and restore the signal, respectively. An optical supervisor channel (OSC) can also be transmitted over the same optical medium to serve OAM purposes; it is analogous to SONET's DCC. Many types of modulation are supported over this medium, to include AM, FM, PSK, QAM, and others. The major benefit of DWDM is that it can expand optical capacity without having to lay more fiber. The channel spacing between wavelengths becomes smaller as technology matures and more wavelengths can be "stuffed" onto a fiber pair. DWDM is most commonly used for commercial long-haul systems and often uses C-band frequencies. Most DWDM deployments run on single-mode fiber, which is built for long-haul transmissions at higher data rates and has a diameter of 9 um. SMF only allows a single ray (mode) of light, which requires more precise lasers. This also increases the range significantly as light does not bounce around within the fiber core as reflected by the cladding. Multi-mode fiber is more common at the premises with a core diameter of ~62.5 um. MMF requires less precise LEDs as light sources since rays can bounce around within the core, but distance is severely limited when compared to SMF. The light bounding inside the MMF causes distortion which limits range, but is much more affordable than SMF.

3.2 IP connectivity to the customer

Because broadband implementation can be a sizable topic, the entire BBA and PPPoE sections demonstrate some of the network connectivity techniques and architectures. This section gives a brief overview of the access technologies only.

3.2.1 Digital Subscriber Line (DSL)

DSL is a widely deployed “last-mile” access technology, typically for residential and small/medium enterprise (SME) customers. It relies on existing telephone lines which are already widespread across the world. DSL passes digital data over telephone lines by using a different set of frequencies than are used to carry phone conversations. Intelligent filtering prevents the voice and data frequencies from interfering with one another. A DSL connection is generally comprised of a DSL modem at the customer end and a DSL access multiplexer (DSLAM) at the provider end. The DSLAM aggregates many DSL connections and, using some kind of transport media like ATM or Ethernet, connects to the BRAS described earlier. There are many types of DSL, most of which are just newer versions and better speeds, but there are two main variants.

SDSL: Symmetric DSL means that the download (from provider to customer) and upload (to provider from customer) speeds are the same. If two sites are considered “peer”, that is, being used as a WAN mechanism to link offices, this would be an appropriate choice. It also may be appropriate for a SME HQ site that is sending a fair amount of data to the Internet.

ADSL: Asymmetric DSL means that the download speed is much faster than the upload speed. Technically the opposite also qualifies as ADSL but doesn’t really exist. This is the most common form of DSL deployed for residential access as most consumers want to download much more information than they upload.

3.2.2 Cable Internet

Like DSL, cable Internet access uses an existing infrastructure that is very common in many homes, which is cable television. This uses coaxial cables as compared to phone lines, but still requires a modem to transmit digital data over the cable lines, much like DSL. At the cable TV provider facility, there are a series of “splitter” devices that ensure data traffic can flow bidirectionally. That is to say, users can upload and download information. This is similar to telephony where calls can be placed or answered. Cable TV however, is receive-only, so TV traffic is still only allowed downstream.

3.2.3 Wireline

Any delivery mechanism that relies on wires, which includes DSL and cable Internet, is considered wireline. Those aforementioned technologies are subsets of this topic, so rather than discuss them again, I will discuss other wireline technologies. Although considered prohibitively expensive and unnecessary years ago, “fiber to the premises” is becoming more popular. This is a dedicated fiber connection to each residence or business, which generally provides superior service. It also commands a premium price (at the time of this writing) in some areas, and is not available everywhere. Rather than re-use existing telephone or cable TV lines, this builds a dedicated network connection. The benefit is

that, with service providers offering IP-based TV and telephony solutions, the existing phone and cable lines may become obsolete. The single fiber connection could conceivably be the only network connection necessary in the future to provide TV, telephone, and Internet service to customers.

4. Virtualization concepts

4.1 SVR vs. HVR

Software-Isolated Virtual Router (SVR): Achieves isolation between different routing instances in software exclusively. This means that the VRs contend for the same set of physical resources. There are three models for achieving this, but the underlying point is that hardware resources are always shared in the data plane. The most obvious and practical example of an SVR in the Cisco world is a VRF. Other vendors may reference to these SVR constructs as routing-instances.

- a. **Overlay:** Guest OSes overlay atop a host operating system. Scales poorly as it introduces resource contention issues. The host OS would be a Type 1 hypervisor, by loose definition.
- b. **Kernel:** Integrates virtualization into the kernel itself (like a Type 2 hypervisor). This essentially turns the kernel into an OS that provides an interface by which VMs communicate with hardware. Introduces extra complexity and instability into the kernel.
- c. **Application:** Doesn't use multiple OSes but virtualizes individual applications. Lower overhead but complicates design, testing, and management of the SVRs. Applications would also need to understand some virtualization aspects, requiring application rework.

Hardware-Isolated Virtual Router (HVR): Dedicated hardware components (cards in a chassis) to both the control and data planes of a router. The only thing shared in an HVR system is "sheet metal" and potentially blowers, electrical lines, and other basic service components. No virtualization is needed in either plane and eliminates contention between VRs. They are more resilient (an attack targeting one does not affect another), easier to manage (clear separation of management boundaries), and scale better (adding more HVRs mean more HW, but also more performance).

For data centers, SVR makes more sense. Rack space/power tend to be premium resources, while router scale is of little consequence. In a DC, routers are typically gateways to provide DC services and are not bearing the load of east-west traffic in a DC. Routing tables are relatively small and transit bandwidth is also. The routers in a DC also tend to need the same or similar feature sets, and since SVRs are hosted on a single platform, this is achieved automatically. DCs are also managed by common entities with multiple administrative domains, so using SVRs is a good choice.

For SP POPs, the situation is almost the opposite. Rack space/cooling are typically easier to come by since the provider probably owns the premises, where DC's tend to rent it. The routers need to be very powerful in both the control and data planes (not including non-transit devices like BGP route-reflectors or out-of-band management devices). In terms of features, a PE and P router have very different roles, so enabling/optimizing for certain feature performance matters based on placement. Management-wise, the scope of administration is more stove-piped so having HVRs is a better option in an SP environment.

Cisco routers running IOS-XR (ASR9K, CRS, etc) can have their cards allocated to secure domain routers (SDRs). Each router is effectively an HVR, sharing only the chassis and the low-level control mechanisms inside of it. They are otherwise totally isolated with their own RPs, forwarding line cards, and other components allocated by the administrator. Connections across SDRs would be external (use cables, no backplane magic). The HVR approach scales linearly with the number of SDRs; with SVRs, the capability of the system is divided every time a new SVR is introduced. The main downside to HVRs is that more routers requires more money, where SVRs are typically free.

4.2 Network Functions Virtualization (NFV)

NFV decouples network functions from hardware appliances and puts them into software, typically as virtual machines. The idea is that provisioning new network components/services, such as firewall, router, load balancer, etc, becomes much faster and easier. This is basically a fancy way of using vFirewalls and vRouters in a network to virtualize some/all of the network functions. The term is quite self-explanatory. Product-wise, the Cisco CSR1000v, XRv, ASAv, vWLC, and several others all contribute to the concept of NFV in some capacity. Almost all of the studying in this lab was done using NFV by that definition. The specific benefits of NFV come from being able to rapidly provision network functions, string them together in a customer-desired topology, and offer this as a network service for a fee. Simply virtualizing network devices isn't very exciting by itself, but the ability to rapidly provision network services for customers is impossible with physical appliances.

4.3 Software Defined Networking (SDN)

Not to be confused with NFV, SDN's goal is to remove the control-plane (brains) from network devices and centralize it in software. The forwarding devices in the data plane (muscles) would be control-plane-less (or with a limited, distributed control-plane for failover). There are many different opinions/designs being proposed for SDN. One extreme is to have all of the brains centralized in a controller while the forwarding devices are commodity items with no intelligence whatsoever (complete centralization). Another extreme is the current deployment of a "legacy" network, which is a totally distributed control-plane. Hybrid approaches tend to Cisco's focus area whereby a centralized controller can optimize particular application flows but devices are still intelligent enough to operate autonomously if required. This forms the basis of their Application-Centric Infrastructure (ACI) model commonly used in Cisco-based data centers. Many SDN standards are still emerging and most new products today claim support for some of these SDN interfaces that allow them to control or be controlled.

Cisco's Performance Routing (PfR) is, in my opinion, the first actually-deployed quasi-SDN solution on the market. It has matured significantly over the past several years and is used extensively in intelligent WAN (IWAN) deployments to reduce costs/optimize bandwidth for enterprise networks. PfR is beyond the scope of CCIE SP and is not evaluated in detail here.

While the "white box" model of total centralization is considered the target architecture by many, it brings other challenges. Packets with IP router alert options or MPLS router alert labels must be punted to the SDN controller, which is over the network. The controller is then responsible for making a

forwarding decision, which puts it in the transit path for some flows. This could be considered a security risk and certainly comes with a performance impact. It may complicate basic MPLS OAM functionality as it relies on IP and MPLS router alert mechanisms. The concern is valid for low-level protocols like IPv4 ARP, IPv6 ND, VFD, QoS, ACL, etc; The ASR9000 offloads these to line-cards to protect the RSP/RP. Other challenges include maintaining a very robust and high-speed management network required to sustain the chatter between SDN controllers and client devices.

5. Mobility concepts

5.1 LTE

Long Term Evolution (LTE) architecture consists of many various components and interfaces shown in the diagram below. The individual parts and their interactions are described here.

UE: User Equipment. This would be an end-user device like a cell phone. Each UE contains a Universal Integrated Circuit Card (UICC). Within the context of LTE, this is also called the SIM (Subscriber Identity Module). In a cell phone, the SIM identifies a phone's number, billing plan, and all other network-related information.

eNodeB: Also called eNB. These are base stations that control the mobile nodes in one or more calls. A base station that is supporting a specific mobile node is reference as the mobile node's "serving eNB". LTE mobile nodes can only communicate with one base station at a time. The eNB has two primary functions: send/receive radio transmissions and to control low-level signaling such as handover commands. eNBs are connected to one another to support mobility events (for packet forwarding and handover) using the X2 interface, and connect to upstream data networks via the S1 interface. The eNodeBs do not need to be fully meshed. The UEs talk to eNodeBs via the LTE-Uu interface.

RAN: Radio Access Network. This is a way of providing backhaul from access networks to the provider's core network. Backhaul is discussed later.

UMTS: Universal Mobile Telecommunication System. This was the third generation (3G) network upon which 4G LTE was built. This was a combination of circuit and packet switched architectures which was more hierarchical (sometimes called "lumpy" by those who prefer "flat" networks) than the current 4G LTE architecture.

E-UTRAN: Evolved UMTS Terrestrial RAN. This encompasses the entire LTE access network, which general consists of eNodeB radios. The E-UTRAN is responsible for mobility control, radio admission control, eNB configuration and provisioning, and dynamic resource allocation (scheduling). LTE is designed to be all-IP (only packet switched) with a flatter architecture. Peak download rates are ~299.6 Mbps with peak upload rates of 75.4 Mbps (highly dependent on equipment, environment, etc). There are many standardized "cell widths" as well: 1.4 MHz, 3 MHz, 5 MHz, 10 MHz, 15 MHz and 20 MHz.

EPC: Evolved Packet Core. This is a network that contains several sub-components described below. It is responsible for forwarding traffic, handover events, filtering, billing, and accounting.

HSS: Home Subscriber Server is a central database that contains information about all of the subscribers within a given network.

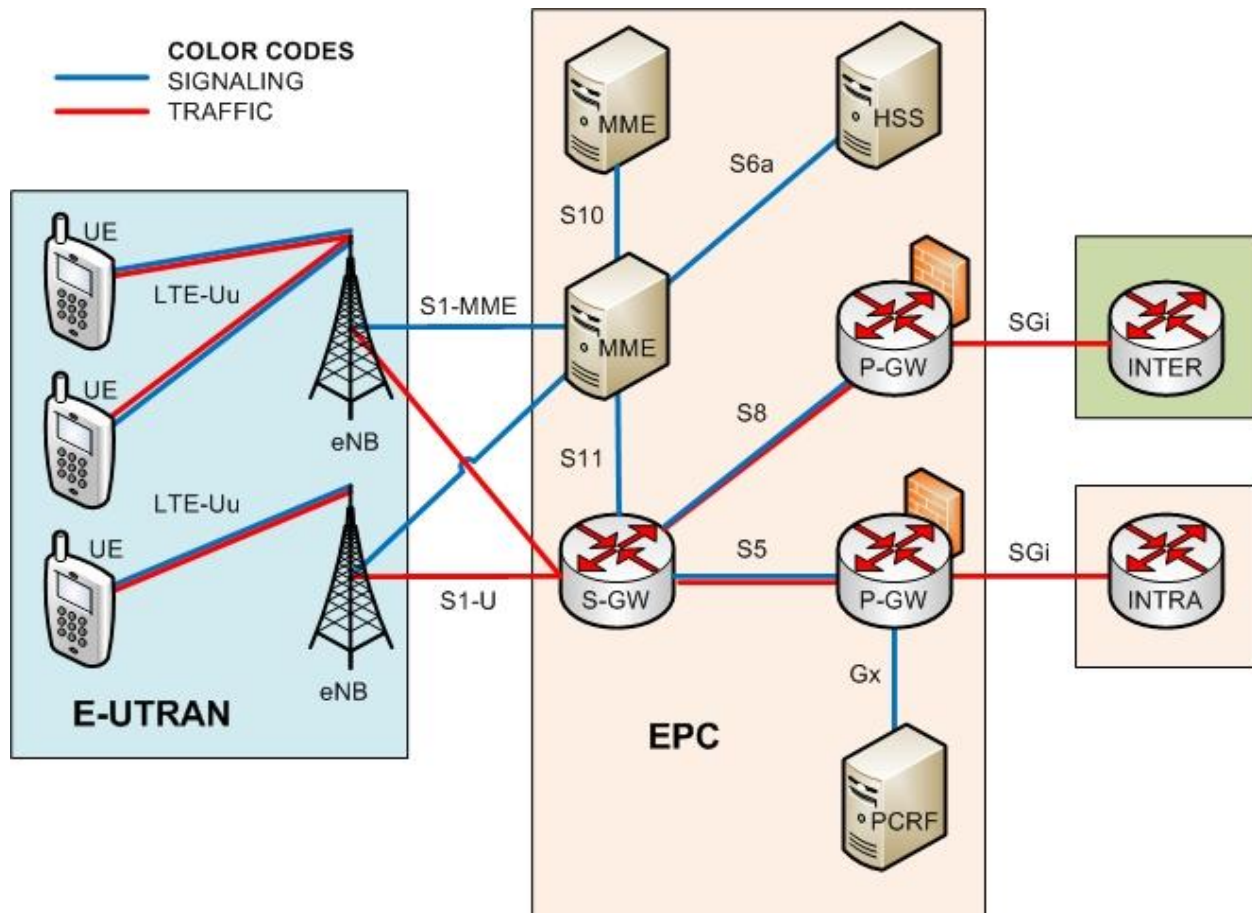
PDN Packet Data Network. Any external network beyond the LTE architecture, such as the Internet.

P-GW: PDN Gateway. This communicates with external PDNs using the SGi interface. Each PDN is identified by a different access point name so that multiple PDNs can exist. The P-GW allocates IP address to the UEs and performs packet filtering for security purposes. The IP addresses allocated to UE's is likely dependent on the PDN to which the P-GW is connected.

S-GW: Serving Gateway. This acts as a router and forwards data between the eNodeB and the PDN Gateway. Only the S-GW and P-GW actually forward bearer traffic; the majority of LTE components are for signaling/control only. The interface between S-GW and P-GW will be either S5 or S8. S5 is used when the S-GW and P-GW are in the same network, and S8 is used when they are in different networks. The S-GW communicates to the E-UTRAN via the S1-U interface. The S-GW is also the mobility anchor point, which is used for encapsulating (tunneling) traffic between S-GWs when mobility events occur.

MME: Mobility Management Entity. Facilitates mobility-related signaling between the HSS and the E-UTRAN devices. It is the main control entity for the entire E-UTRAN and is also responsible for authentication services. The MME communicates to the HSS using the S6a interface and to the E-UTRAN using the S1-MME interface. MMEs can communicate to one another using the S10 interface.

PCRF: Policy Control and Charging Rules function. Mainly responsible for QoS policy, flow-based charging functionality, and policy control enforcement function (PCEF). It connects to the P-GWs via the Gx interface so the edge of the EPC can appropriately bill subscribers in the E-UTRAN, treat their traffic according to SLAs, etc.



5.2 Backhaul

Generally speaking, a backhaul link is any link that connects the small subnetworks at the edges (access or aggregation networks) to the core network. Within the context of LTE, this would provide the transport for the eNodeB to the S-GW, or the S1-U interface described in the LTE section. It could also carry inter-eNodeB traffic/signaling for mobility events (X2 interface) or eNodeB to MME signaling (S1-MME interface). Traditionally, backhaul links have been TDM-based, such as T1/E3. Multiple TDM links could be bundled together to support higher bandwidth backhaul links, but over time this became less profitable. Ethernet has been used very successfully given its lower cost and higher bandwidth compared to traditional TDM technologies. SONET/SDH can also be used but is less common. Wireless backhaul can be popular for a number of reasons, but comes with drawbacks as well. The benefits of wireless backhaul using microwave links is that they are easy/fast to deploy and allow moving POPs as necessary. They tend to be slower than wired connections (less bandwidth) and are viewed as a temporary measure. Assuming high towers are available, microwave links are more desirable, cheaper, and more scalable than copper links, but not as desirable as fiber. Cell towers, for example, are migrating from wireless to fiber optic connections. For smaller nodes where POPs require mobility, wireless backhaul is the best option for the RAN. Wireless links can be licensed or unlicensed; the FCC regulates power output restraints both, but the difference is that spectrum for unlicensed bands is not managed by anyone.

6. Describe BGP path attributes

This section will not belabor the extensively documented BGP best-path selection algorithm. Instead, I will comment heavily on the lesser known caveats. Before best-path runs, there are some pre-checks:

1. **Next-hop reachability:** Mandatory, well-known, and transitive. There must be a route to the BGP next-hop. It can be BGP for recursive lookups also, but ultimately a connected route is at the bottom of every recursive route lookup anyway provided there isn't a fault. Failure to meet this reachability condition results in (inaccessible) appearing next to the next-hop value.
2. **iBGP synchronization:** Often off by default, this rule states that for an iBGP route to be considered for best-path, there must be a matching IGP route in the routing table. Matching means exact prefix length match, so less specific aggregates cannot satisfy the iBGP synchronization condition. Static routes can also satisfy this rule, but IGP routes are typically used. It's original purpose was to ensure one did not create multihop iBGP peerings where routers in the transit path were not running BGP at all. **Note: if the underlying IGP is OSPF and synchronization is enabled, the OSPF and BGP RIDs must match for the iBGP route to be synchronized.**
3. **Pre-bestpath cost-community:** Optional, non-transitive. This feature is tested heavily in this document. If the pre-bestpath point of insertion (POI) is passed in a prefix via extended communities, it is considered before any other best-path attribute. The rules for its operation are defined in the appropriate section of this document, but in summary, it is the ultimate "trump card" for influencing the best-path selection short of route filtering.

The selection process begins with the following steps, summarized from the attached Cisco reference:

1. **Weight:** Optional, local only. Higher is better, and locally originated prefixes are assigned a value of 32,768 by default. This default weight optimization makes the "local origination" step moot since local prefixes are almost always preferred.
2. **Local preference:** Mandatory, non-transitive. Higher is better with a default value of 100. Typically assigned inbound to an eBGP peer to affect traffic flows outbound. A number greater than 100 forces traffic in an AS to exit towards a particular eBGP peer, and a number less than 100 makes an egress point less desirable. This attribute is maintained across confed-external boundaries.
3. **Accumulated IGP (AIGP):** Documented heavily in its own chapter, this feature is allows BGP to add the IGP metric to the BGP next-hop with the remote ASes metric value. It's like MED except higher in the best-path selection but also accounts for the local IGP costs, too. Effectively, it is an end-to-end cost carried inside of BGP.
4. **Locally originated better than BGP learned:** Given the weight assigned by Cisco routers for locally originated prefixes, local origination beats even local preference by default. Otherwise, routes locally originated by a router ("sourced") are preferred over any learned BGP routes.
5. **AS-path length:** Mandatory, transitive, and well-known. The local AS is appended to an UPDATE message when routes are advertised out of an AS. Within a confederation, the values are placed into a parenthesized list and treated as a single AS. When existing a confederation, this list is

collapsed into the AS number for the entire AS. AS path pre-pending is commonly set outbound to influence traffic flows inbound (opposite utility as local preference).

6. **Origin:** Mandatory, transitive, and well-known. IGP implies the route was derived from IGP (network statement), EGP is a legacy option soon to be deprecated, and incomplete implies unknown origin (redistribution). It isn't commonly used for path selection but can be used either inbound or outbound to influence flows out or in, respectively.
7. **Multi-exit discriminator (MED):** Optional, non-transitive. Used to carry the IGP metric to remote ASes to "hint" at the best path within the source AS network. Can be set outbound to influence flows inbound, similar to AS-path pre-pending. AIGP extends the idea of MED by taking the same (or similar) value and evaluating it sooner in concert with the BGP next-hop metric. In a multi-homed environment, if there are multiple peer ASes, this feature cannot be used since MED is non-transitive, so AS-path prepending would work instead.
8. **Neighbor type:** eBGP preferred over iBGP. Confed-external is treated the same as confed-internal, so this would be a tie in that case. The idea is to have "hot potato" routing by default where getting traffic out of the AS is preferable.
9. **IGP metric to the BGP next hop:** Computed locally based on the recursive route lookups. Lower numbers are preferred since that implies a shortly IGP path to the next BGP router in the topology. Summed with AIGP, if configured, to determine lowest path-cost metric end to end.

The following steps are considered "tie breakers" because after this point, from a performance and optimal routing perspective, BGP cannot tell if there is one path obviously better than the others.

10. **IGP cost-community:** Optional, non-transitive. This feature is tested heavily in this document. If the IGP point of POI (which is the default) is passed in a prefix via extended communities, it is considered as the first "tie breaker". You can use this in your routes to further "hint" the best path without changing real BGP attributes. The rules for its operation are defined in the appropriate section of this document.
11. **Multipath:** Not really a selection criterion, but this is normally where multipath determinations would go. Multipath rules can be relaxed for iBGP unequal cost (where the IGP metric can be unequal), as well as the AS-path numbers.
12. **For eBGP only, select the oldest route:** This appears at the bottom of the route details when using the "show bgp afi safi x.x.x.x" command. The idea is to reduce churn in the eBGP topology by selecting the most stable route.
13. For iBGP or eBGP with the "**always compare RID**" configured, select the route coming from the lowest BGP RID. It's generally not used, because selecting the oldest eBGP route helps reduce churn in the BGP process for eBGP peers. For iBGP routes coming from a route-reflector (or had an RR anywhere in the path), a field known as the "originator" is compared instead. The value of this option is for more deterministic eBGP routing when evaluating tie-breakers.
14. **For iBGP, select the route with the lowest cluster-list length:** The idea is to pick the route that was reflected the fewest number of times.

15. **Lowest peer address:** This is the final tie-breaker and is a totally arbitrary selection criterion. This is not the lowest peer BGP RID; it is the lowest peer address where the TCP session is established.

7. Describe MPLS forwarding and control plane mechanisms

There are many control-plane components to MPLS. The details of LDP and static bindings are described here since other methods like BGP, RSVP-TE, and SR are detailed in their own sections. These protocols are discussed very briefly here for completeness.

LDP is a very common method for allocating labels for IP prefixes within an MPLS core. It typically runs one-to-one with the underlying IGP so that all IP-enabled links are also MPLS-enabled, provided the destination is an IP prefix.

BGP can also be used to distribute transport labels. This is commonly called the “labeled-unicast” address family and is relevant for IPv4 and IPv6. It is commonly used to support 6PE, Unified MPLS, and Carrier supporting Carrier (CSC) architectures. Each of these topics is examined in great detail, so this is just meant to be a summary. The additional IPv4/v6 label capability is advertised during the BGP peer negotiation to determine whether peers can exchange labels or not. Failure to negotiate this AFI means that the IPv4/v6 session can still form and distribute IPv4/v6 prefixes, but not MPLS labels.

RSVP with traffic engineering (TE) extensions can be used to build LSPs in a network enabled for TE. This is explained in great detail later in this book. The reason it is mentioned here is because, like BGP, it could theoretically be used to completely replace LDP.

Segment Routing (SR) is designed to actually replace LDP by carrying the prefix-to-label bindings inside of IGP messages, such as IS-IS LSPs and OSPF LSAs. This removes the need for advanced LDP features like IGP synchronization which is described later.

7.1 Label Distribution Protocol (LDP)

As mentioned earlier, LDP is commonly used within a core to provide transport service between MPLS service endpoints. First, we begin with some definitions.

Label distribution modes:

Downstream on Demand (DoD): Each LSR requests a label binding for a FEC following that IP routing path. There is only one binding per FEC received and only from its downstream LSR. The LIB only shows one remote binding. This is only used on Label Controlled (LC) ATM interfaces.

Unsolicited downstream (UD): Each LSR distributes a label binding for all IGP routes to all neighboring LSRs without being asked to do so. The LIB will likely show a binding from each neighbor. This is used on all interfaces except LC-ATM.

Label retention modes:

Liberal Label Retention (LLR): All labels are stored in the LIB for a given FEC. Only the label from the downstream LSR, according to the FIB, is installed in the LFIB. The others are for backup only and this facilitates FRR. Better for HA and used on all interfaces except LC-ATM.

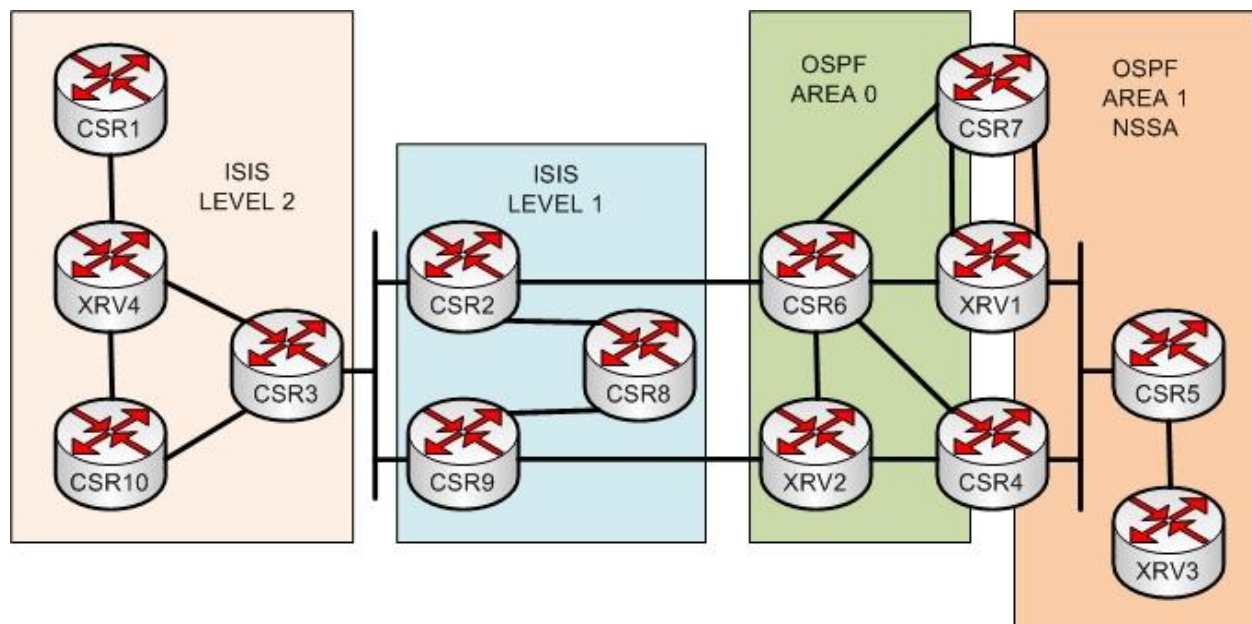
Conservative Label Retention (CLR): Only the label from the downstream LSR is stored in the LIB. Better for memory conservation and only used on LC-ATM interfaces.

LSP control modes:

Independent: Each LSR creates a local binding for a FEC as soon as it recognizes the FEC (that is, the IP prefix is in the FIB via IGP). No other LSR is involved. Disadvantage is that some LSRs forward packets before the LSP is set up. Used in most Cisco platforms.

Ordered: LSR only creates a binding for a FEC if it realizes that it is the egress LSR, or if it received a label binding from the next-hop for this FEC. Only allocates labels for connected routes or IGP routes for which it has received a binding from the next-hop LSR. Used on Cisco ATM switches.

The focus of this test is LDP and basic MPLS forwarding, so advanced MPLS services (L3VPN, L2VPN, TE, etc) are not examined in any detail. The network is a large IS-IS and OSPF domain with multiple levels and areas. Although not relevant for LDP (and probably a bad design), I illustrate this to show that LDP is IGP-agnostic and LSPs can be built across IGP boundaries provided there is IP reachability. Note that CSR7 has connections to XRv1 in both areas 0 and 1.



The basic interface and IGP configurations are not shown, but the highlights are shown below. Of note, CSR3 is an L1/L2 router that leaks /32 level-2 routes into level-1 to complete the LSPs. Failure to do this will break LSPs as we will see later as LDP bindings are prefix-specific. With the exception of the L2-into-

L1 leaking, the full IS-IS configuration is shown since it is near-identical on all IS-IS routers. CSR4 and XRv1 are OSPF ABRs but have no special filtering configured since area 1 is a non-stub area.

```
! CSR3
ip prefix-list PL_HOST_ROUTES seq 5 permit 0.0.0.0/0 ge 32

route-map RM_L2_INT0_L1 permit 10
  match ip address prefix-list PL_HOST_ROUTES

router isis LDP
  net 00.0000.0000.0003.00
  advertise passive-only
  metric-style wide
  log-adjacency-changes all
  redistribute isis ip level-2 into level-1 route-map RM_L2_INT0_L1
  passive-interface Loopback0
  address-family ipv6
    multi-topology
  advertise passive-only
```

CSR6 and XRv2 mutually redistribute between IS-IS and OSPF, and use administrative distance (AD) to ensure IS-IS routes are never learned via OSPF, which could cause loops. I use a parameterized RPL structure on XRv2 for modularity, and re-use route-maps on CSR6. There are better and more strict ways to accomplish the redistribution, but since it isn't the focus of this lab, I use a fast method.

```
! XRv2
prefix-set PS_HOST_ROUTES
  92.0.0.0/24 ge 32
end-set

route-policy RPL_MATCH_IF_DEST($PS)
  if destination in $PS then
    pass
  endif

router isis LDP
  address-family ipv4 unicast
  redistribute ospf 92 level-1 route-policy RPL_MATCH_IF_DEST(PS_HOST_ROUTES)

router ospf 92
  distance ospf external 255
  redistribute isis LDP route-policy RPL_MATCH_IF_DEST(PS_HOST_ROUTES)

! CSR6
ip prefix-list PL_HOST_ROUTES seq 5 permit 92.0.0.0/24 ge 32

route-map RM_REDIST_FILTER permit 10
  match ip address prefix-list PL_HOST_ROUTES
```

```

router ospf 92
 redistribute isis LDP level-1 subnets route-map RM_REDIST_FILTER

router isis LDP
 redistribute ospf 92 route-map RM_REDIST_FILTER level-1

```

Once all of the basic interfaces, routing, and redistribution are configured, every router should be able to see the loopbacks of every other router. I will test this at CSR1 and XRv3, since those are at the edges of the network. Fortunately, the routing table is sorted so we can quickly scan to see 14 loopbacks on each. Reachability could be broken, but since we will be configuring MPLS, we don't care right now.

```

R1#show ip route 92.0.0.0 255.255.255.0 longer-prefixes | begin Gateway
Gateway of last resort is not set

```

```

          92.0.0.0/8 is variably subnetted, 16 subnets, 2 masks
C          92.0.0.1/32 is directly connected, Loopback0
i L2      92.0.0.2/32 [115/30] via 92.1.14.14, 01:52:12, GigabitEthernet2.514
i L2      92.0.0.3/32 [115/20] via 92.1.14.14, 01:52:12, GigabitEthernet2.514
i L2      92.0.0.4/32 [115/40] via 92.1.14.14, 01:52:12, GigabitEthernet2.514
i L2      92.0.0.5/32 [115/40] via 92.1.14.14, 01:52:12, GigabitEthernet2.514
i L2      92.0.0.6/32 [115/40] via 92.1.14.14, 01:52:12, GigabitEthernet2.514
i L2      92.0.0.7/32 [115/40] via 92.1.14.14, 01:52:12, GigabitEthernet2.514
i L2      92.0.0.8/32 [115/40] via 92.1.14.14, 01:52:12, GigabitEthernet2.514
i L2      92.0.0.9/32 [115/30] via 92.1.14.14, 01:52:12, GigabitEthernet2.514
i L2      92.0.0.10/32 [115/20] via 92.1.14.14, 01:52:12, GigabitEthernet2.514
i L2      92.0.0.11/32 [115/40] via 92.1.14.14, 01:52:12, GigabitEthernet2.514
i L2      92.0.0.12/32 [115/40] via 92.1.14.14, 01:52:12, GigabitEthernet2.514
i L2      92.0.0.13/32 [115/40] via 92.1.14.14, 01:52:12, GigabitEthernet2.514
i L2      92.0.0.14/32 [115/10] via 92.1.14.14, 01:52:12, GigabitEthernet2.514

```

```

RP/0/0/CPU0:XRv3#show route ipv4 longer-prefixes 92.0.0.0/24 | begin ^O
O E2 92.0.0.1/32 [110/20] via 92.5.13.5, 00:09:03, GigabitEthernet0/0/0/0.553
O E2 92.0.0.2/32 [110/20] via 92.5.13.5, 02:05:14, GigabitEthernet0/0/0/0.553
O E2 92.0.0.3/32 [110/20] via 92.5.13.5, 02:05:14, GigabitEthernet0/0/0/0.553
O IA 92.0.0.4/32 [110/3] via 92.5.13.5, 02:05:14, GigabitEthernet0/0/0/0.553
O   92.0.0.5/32 [110/2] via 92.5.13.5, 02:05:14, GigabitEthernet0/0/0/0.553
O IA 92.0.0.6/32 [110/4] via 92.5.13.5, 02:05:14, GigabitEthernet0/0/0/0.553
O IA 92.0.0.7/32 [110/4] via 92.5.13.5, 02:05:14, GigabitEthernet0/0/0/0.553
O E2 92.0.0.8/32 [110/20] via 92.5.13.5, 02:05:14, GigabitEthernet0/0/0/0.553
O E2 92.0.0.9/32 [110/20] via 92.5.13.5, 02:05:14, GigabitEthernet0/0/0/0.553
O E2 92.0.0.10/32 [110/20] via 92.5.13.5, 00:09:03, GigabitEthernet0/0/0/0.553
O IA 92.0.0.11/32 [110/3] via 92.5.13.5, 02:05:14, GigabitEthernet0/0/0/0.553
O IA 92.0.0.12/32 [110/4] via 92.5.13.5, 02:05:14, GigabitEthernet0/0/0/0.553
L   92.0.0.13/32 is directly connected, 13:06:38, Loopback0
O E2 92.0.0.14/32 [110/20] via 92.5.13.5, 00:09:03, GigabitEthernet0/0/0/0.553

```

There are two main ways to enable LDP: at the interface level, or automatically on a 1:1 basis with IGP. I use a combination of methods throughout the topology to make things interesting, but the effect is the same regardless of which method is used. I personally prefer auto-config because its less typing and automatically enables LDP wherever IGP is enabled; in many cases this can be tuned on a per-level or per-area basis. It also accounts for any new IGP-enabled interfaces in the future, which helps maintain IGP/LDP synchronization. First, we will look at the simpler manual method. This is configured on CSR10, and while the command makes no explicit reference to LDP, it enables MPLS for IP prefixes. The default protocol used for IP label bindings is LDP, which is different than Cisco's Tag Distribution Protocol (TDP) which is older and inferior. We also need LDP enabled for static label bindings as seen later. After configuring this, we can quickly verify that LDP is enabled for this interfaces as shown below.

```
! CSR6
interface GigabitEthernet2.530
  mpls ip
```

```
interface GigabitEthernet2.504
  mpls ip
```

```
R10#show mpls interfaces
```

Interface	IP	Tunnel	BGP	Static	Operational
GigabitEthernet2.530	Yes (ldp)	No	No	No	Yes
GigabitEthernet2.504	Yes (ldp)	No	No	No	Yes

The alternative method is to use auto-config, shown on CSR3. We can optionally specify either level-1 or level-2 if we want to be granular, but since CSR3 is the L1/L2 router, we enable it for both levels. Then, we verify it is enabled on all IS-IS interfaces.

```
! CSR3
router isis LDP
  mpls ldp autoconfig
```

```
R3#show mpls interfaces
```

Interface	IP	Tunnel	BGP	Static	Operational
GigabitEthernet2.523	Yes (ldp)	No	No	No	Yes
GigabitEthernet2.530	Yes (ldp)	No	No	No	Yes
GigabitEthernet2.534	Yes (ldp)	No	No	No	Yes

If we look at the detailed interface command on both CSR3 and CSR10, it will tell us whether LDP has been enabled via the interface command (mpls ip) or auto-config via IGP. We will look at the link that CSR3 and CSR10 share to show the difference.

```
R10#show mpls interfaces gigabitEthernet 2.530 detail
```

```
Interface GigabitEthernet2.530:
  Type Unknown
  IP labeling enabled (ldp) :
```

```

    Interface config
    LSP Tunnel labeling not enabled
    IP FRR labeling not enabled
    BGP labeling not enabled
    MPLS operational
    MTU = 1500

R3#show mpls interfaces gigabitEthernet 2.530 detail
Interface GigabitEthernet2.530:
    Type Unknown
    IP labeling enabled (ldp) :
    IGP config
    LSP Tunnel labeling not enabled
    IP FRR labeling not enabled
    BGP labeling not enabled
    MPLS operational
    MTU = 1500

```

Auto-config may not be appropriate for all interfaces. For example, a router may have 100 interfaces but only 90 should be MPLS enabled. Rather than configure “mpls ip” 90 times, you can select individual interfaces to disable auto-config. As an example, CSR7 and XRv1 have two parallel links between themselves. VLAN 517 is MPLS-enabled by VLAN 571 should not be, and both routers use auto-config. In XE, we can disable this on a per-link basis at the interface level. On XR, we perform the same logic under the LDP process.

```

! CSR7
interface GigabitEthernet2.571
    no mpls ldp igp autoconfig

! XRv1
mpls ldp
interface GigabitEthernet0/0/0/0.571
    address-family ipv4
    auto-config disable

```

To verify it, we can check to see if MPLS is enabled on those specific interfaces. XE shows no output, which is implicit confirmation that MPLS is not enabled on this link. XR gives explicit confirmation to arrive at the same conclusion.

```

R7#show mpls interfaces gigabitEthernet 2.571
Interface          IP          Tunnel    BGP Static Operational
[no output]

RP/0/0/CPU0:XRv1#show mpls interfaces gigabitEthernet 0/0/0/0.571
Interface is not MPLS-enabled: 'GigabitEthernet0/0/0/0.571'

```

Before continuing with any advanced topics, we will examine the LDP discovery and neighbor formation process. First, LDP sends hello packets to the all-routers multicast group of 224.0.0.2 using UDP port 646. Unlike many other protocols, LDP does not introduce a new IP protocol for its operation, using UDP and TCP only. During the initial session discovery, we can reveal these details with a debug command. Below, CSR3 sends an LDP hello on the link towards CSR10, and CSR10 does the same. Upon receiving the LDP hello, CSR3 creates a new session for neighbor 92.0.0.10:0, where the last 0 represents the label space. System-wide label space is represented with a value of 0, while interface-specific label spaces use other numbers but only have relevance for ATM. These are not examined here.

```
! CSR3
debug mpls ldp transport events interface g2.530

ldp: Send ldp hello; GigabitEthernet2.530, src/dst 92.3.10.3/224.0.0.2,
inst_id 0
ldp: Rcvd ldp hello; GigabitEthernet2.530, from 92.3.10.10 (92.0.0.10:0),
intf_id 0, opt 0xC
ldp: ldp Hello from 92.3.10.10 (92.0.0.10:0) to 224.0.0.2, opt 0xC
ldp: New adj 0x7F97D1A384C0 for 92.0.0.10:0, GigabitEthernet2.530
ldp:      adj_addr/xport_addr 92.3.10.10/92.0.0.10
```

The next batch of debug indicates that TCP port 646 has been opened for packets from 92.0.0.10. The link address 92.3.10.10 was the source of the hello packets, but the TCP session is sourced from the LDP router-ID by default, which is set to loopback0.

```
! CSR3
ldp: Request adj send hello back on GigabitEthernet2.530 to (xport addr
92.0.0.10) in 1 msec
ldp: local interface = GigabitEthernet2.530, holdtime = 15000, peer
92.3.10.10 holdtime = 15000
ldp: Link intvl min cnt 2, intvl 5000, interface GigabitEthernet2.530
ldp: Opening listen port 646 for 92.0.0.10 (for hellos from 92.3.10.10)
```

Most of the debugs are noisy and not terribly useful, but here we can see the TCP session forming. CSR10 uses source port 19889 with destination port 646, which was opened on CSR3. The TCP TCB process finds an adjacency which means the session can be locally processed. Once data is transported across the TCP session, the session is assumed to be up.

```
! CSR3
ldp: Incoming {ldp conn 92.0.0.3:646=>92.0.0.10:19889} with normal priority
ldp: Process work item for incoming call
ldp: Found adj 0x7F97D1A384C0 for 92.0.0.10 (Hello xport addr opt)
[snip]
ldp: Data received for adj 0x7F97D1A384C0 from 92.3.10.10!
created dhcb: tableid 0, local 92.0.0.3, target 92.0.0.10
ldp: Setup directed hello for 92.0.0.10, holding_timer = 0
%LDP-5-NBRCHG: LDP Neighbor 92.0.0.10:0 (1) is UP
```

We can check to see the neighbor being up on CSR3. This output is verbose but very important. First we see the peer identity, which is its router-ID, along with our local router-ID. Then, we see the TCP connection endpoints and their TCP ports; by default, this is between the router-IDs. The discovery sources show all methods by which CSR10 was discovered, and in this case, it was from LDP hello packets on Gig2.530 from 92.3.10.10. Last, the addresses local to CSR10 are considered “bound” to this LDP peer, which is critical for MPLS forwarding. If CSR3 has an IGP route with a next-hop of any CSR10 interface, it must use the LDP label from CSR10.

```
R3#show mpls ldp neighbor 92.0.0.10
Peer LDP Ident: 92.0.0.10:0; Local LDP Ident 92.0.0.3:0
TCP connection: 92.0.0.10.19889 - 92.0.0.3.646
State: Oper; Msgs sent/rcvd: 57/32; Downstream
Up time: 00:11:53
LDP discovery sources:
  GigabitEthernet2.530, Src IP addr: 92.3.10.10
Addresses bound to peer LDP Ident:
  92.3.10.10      92.0.0.10      92.10.14.10
```

The LDP neighbor table will show fully-negotiated neighbors with operational TCP sessions. Sometimes, when troubleshooting, the neighbor isn’t fully operational, or there are problems with its formation. In those cases, we can also see the LDP discovery methods. For each interface. The “xmit” and “recv” options show where LDP hellos are being sent and received, respectively. Based on CSR3’s location in the network, it should ideally have 4 LDP neighbors since it has bidirectionally discovered 4 other LDP speakers across its 3 connected interfaces.

```
R3#show mpls ldp discovery
Local LDP Identifier:
  92.0.0.3:0
Discovery Sources:
Interfaces:
  GigabitEthernet2.523 (ldp): xmit/recv
    LDP Id: 92.0.0.9:0
    LDP Id: 92.0.0.2:0
  GigabitEthernet2.530 (ldp): xmit/recv
    LDP Id: 92.0.0.10:0
  GigabitEthernet2.534 (ldp): xmit/recv
    LDP Id: 92.0.0.14:0
```

We can drill into more details with this command as well. This shows interface-level details such as the hello/hold interval, transport address (discussed later), and authentication options.

```
R3#show mpls ldp discovery detail | begin 530
  GigabitEthernet2.530 (ldp): xmit/recv
    Enabled: IGP config;
```

```

Hello interval: 5000 ms; Transport IP addr: 92.0.0.3
LDP Id: 92.0.0.10:0
  Src IP addr: 92.3.10.10; Transport IP addr: 92.0.0.10
  Hold time: 15 sec; Proposed local/peer: 15/15 sec
  Reachable via 92.0.0.10/32
  Password: not required, fallback, in use
Clients: IPv4, mLDP

```

[snip]

After the sessions are formed, LDP will be exchanging labels with the peer. To see this, we will flap the neighbor with CSR10 and enable another debug. The first thing LDP does after the neighbor comes up is advertises its local interfaces without any label bindings. This is what populates the “peer identity bindings” seen in the LDP neighbor table. Again, this is critical for MPLS forwarding since LDP must be aware of all local interfaces enabled for MPLS when selecting labels for forwarding.

```

! CSR3
R3#debug mpls ldp advertisements

```

```

%LDP-5-NBRCHG: LDP Neighbor 92.0.0.10:0 (1) is UP
lcon: Send initial advertisements to peer 92.0.0.10:0
lcon: peer 92.0.0.10:0 (pp 0x7F97D1D0C2A0): advertise 92.0.0.3
lcon: peer 92.0.0.10:0 (pp 0x7F97D1D0C2A0): advertise 92.3.10.3
lcon: peer 92.0.0.10:0 (pp 0x7F97D1D0C2A0): advertise 92.2.3.3
lcon: peer 92.0.0.10:0 (pp 0x7F97D1D0C2A0): advertise 92.3.14.3

```

Next, LDP actually begins to distribute labels. Every prefix for which label allocation and advertisement is allowed (controlled by filters which are seen later) is advertised. This is coupled with the local labels from CSR3. Notice that all of CSR3’s transit links are assigned labels as well, since those count as IGP routes. All connected interfaces are advertised with some kind of null label, typically implicit-null (label 3). I have omitted some of the output because it is highly repetitive since CSR3 allocates a label for all loopbacks, of which there are 14. When the label exchange is complete, CSR3 “deassigns” CSR10 from its workflow for label advertising. Notice that remote prefixes, such as CSR10 and XRv4 loopbacks, are assigned labels from CSR3’s global pool of labels, which is 3000 – 3999.

```

! CSR3
lcon: peer 92.0.0.10:0 (pp 0x7F97D1D0C2A0): advertise 92.0.0.3/32, label 3
      (imp-null) (#2)
lcon: peer 92.0.0.10:0 (pp 0x7F97D1D0C2A0): advertise 92.0.0.10/32, label
      3000 (#4)
lcon: peer 92.0.0.10:0 (pp 0x7F97D1D0C2A0): advertise 92.0.0.14/32, label
      3002 (#58)
[snip]
lcon: peer 92.0.0.10:0 (pp 0x7F97D1D0C2A0): advertise 92.2.3.0/24, label 3
      (imp-null) (#79)
lcon: peer 92.0.0.10:0 (pp 0x7F97D1D0C2A0): advertise 92.3.10.0/24, label 3
      (imp-null) (#81)

```

```
lcon: peer 92.0.0.10:0 (pp 0x7F97D1D0C2A0): advertise 92.3.14.0/24, label 3
(imp-null) (#83)
[snip]
lcon: (default) Deassign peer id; 92.0.0.10:0: id 0
```

We can verify that CSR10 received these label bindings by checking the label information base (LIB). The label values in the LIB match what CSR3 advertised via LDP.

```
R10#show mpls ldp bindings 92.0.0.3 32 neighbor 92.0.0.3
lib entry: 92.0.0.3/32, rev 94
remote binding: lsr: 92.0.0.3:0, label: imp-null
R10#show mpls ldp bindings 92.0.0.10 32 neighbor 92.0.0.3
lib entry: 92.0.0.10/32, rev 101
remote binding: lsr: 92.0.0.3:0, label: 3000
R10#show mpls ldp bindings 92.0.0.14 32 neighbor 92.0.0.3
lib entry: 92.0.0.14/32, rev 144
remote binding: lsr: 92.0.0.3:0, label: 3002
```

This is where the LDP/CEF interaction becomes important. Now that CSR10 learned labels from CSR3, it needs to be able select the proper label when sending traffic to a destination. For example, let's assume CSR10 wants to send traffic to CSR8. The first thing it does is consult its routing table (ignore the FIB for now). The route is an IGP route from CSR3 with a next-hop of 92.3.10.3. This means we MUST use an LDP label (cannot be BGP, RSVP-TE, etc) that was learned via whichever LDP peer is bound to address 92.3.10.3.

```
R10#show ip route 92.0.0.8
Routing entry for 92.0.0.8/32
Known via "isis", distance 115, metric 30, type level-2
Redistributing via isis LDP
Last update from 92.3.10.3 on GigabitEthernet2.530, 00:29:20 ago
Routing Descriptor Blocks:
* 92.3.10.3, from 92.0.0.3, 00:29:20 ago, via GigabitEthernet2.530
Route metric is 30, traffic share count is 1
```

Assuming we didn't know which router was bound to 92.3.10.3, we can check the LDP neighbor table. CSR10 has two neighbors: XRv4 and CSR3. We clearly see that 92.3.10.3 is bound to CSR3.

```
R10#show mpls ldp neighbor
Peer LDP Ident: 92.0.0.14:0; Local LDP Ident 92.0.0.10:0
TCP connection: 92.0.0.14.19636 - 92.0.0.10.646
State: Oper; Msgs sent/rcvd: 51/52; Downstream
Up time: 00:31:09
LDP discovery sources:
GigabitEthernet2.504, Src IP addr: 92.10.14.14
Targeted Hello 92.0.0.10 -> 92.0.0.14, active, passive
Addresses bound to peer LDP Ident:
92.10.14.14      92.1.14.14      92.3.14.14      92.0.0.14
```



```

Peer LDP Ident: 92.0.0.3:0; Local LDP Ident 92.0.0.10:0
TCP connection: 92.0.0.3.646 - 92.0.0.10.24379
State: Oper; Msgs sent/rcvd: 21/36; Downstream
Up time: 00:15:04
LDP discovery sources:
  GigabitEthernet2.530, Src IP addr: 92.3.10.3
Addresses bound to peer LDP Ident:
  92.0.0.3          92.3.10.3          92.2.3.3          92.3.14.3

```

CSR10 must consult its LIB to find out what label it should use to reach 92.0.0.8/32 via 92.3.10.3. This label would have been allocated by CSR3, and we find value 3011. The LIB must be consulted after the route lookup occurs, because if we jumped right to this step, it is not clear whether label 3011 or 94008 should be used. Liberal label retention, discussed earlier, means that routers will hold onto all labels they learn even if they aren't programmed into the LFIB. This is to support fast reconvergence without having to readvertise labels constantly. Coupled with IP fast-reroute, this removes the need for LDP to have an FRR capability with respect to IP prefix label binding advertisement.

```

R10#show mpls ldp bindings 92.0.0.8 32
lib entry: 92.0.0.8/32, rev 143
  local binding: label: 10011
  remote binding: lsr: 92.0.0.14:0, label: 94008
  remote binding: lsr: 92.0.0.3:0, label: 3011

```

The combination of the IP next-hop and associated label are programmed into the FIB. The FIB is consulted when an IP packet arrives (or is locally generated) at a router, and the act of adding labels atop an IP packet is called “imposing” or “pushing” the label. The word “imposition” is used to describe the process as well. Part of the reason MPLS is very efficient is that the label operations happen in CEF, so the RIB/LIB lookups are bypassed for normal transit traffic along an LSP.

```

R10#show ip cef 92.0.0.8
92.0.0.8/32
  nexthop 92.3.10.3 GigabitEthernet2.530 label 3011

```

We can go into great detail with the FIB adjacency information. The internal FIB details show an “output chain”, or process of encapsulation events, which shows the label push occurring just before the layer 2 encapsulation. Since it is an Ethernet interface, the outer-most encapsulation is still a standard Ethernet header. It is using VLAN 3530 so a dot1q VLAN tag is added as well. At the end of the encapsulation string is 0x8847, which indicates an MPLS unicast packet being transported. The label value is not shown here (this is a generic output not specific to any LSP) but we know it would be 3011 in our case.

```

R10#show ip cef 92.0.0.8 internal | begin output_chain
output chain:
  label 3011
  TAG adj out of GigabitEthernet2.530, addr 92.3.10.3 7FBC3F46FAC0

```

```

R10#show adjacency gigabitEthernet 2.530 link mpls 92.3.10.3 encapsulation
Protocol Interface Address
TAG GigabitEthernet2.530 92.3.10.3(14)
Encap length 18
005056A98CCF005056A9F96181000DCA
8847
L2 destination address byte offset 0
L2 destination address byte length 6
Link-type after encap: dot1Q
Provider: ARPA

```

Assuming CSR10 encapsulates the packet correctly, CSR3 will consult its LFIB upon receiving the packet. This is because an MPLS packet, not an IP packet, was received. When label 3011 arrives, CSR3 is performing ECMP to reach CSR8. This is because the routing table has two ECMP paths, so the RIB (and therefore the FIB/LFIB) installs both. Load-sharing is done like it is for IPv4, where the source/destination IPv4 addresses are inputs to the LFIB sharing mechanism. The same is true for IPv6 packets. For non-IP traffic (Ethernet frames inside L2VPN, etc), the bottom label is used.

```

R3#show mpls forwarding-table labels 3011 detail
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id Switched      interface
3011       2002       92.0.0.8/32 0             Gi2.523    92.2.3.2
MAC/Encaps=18/22, MRU=1500, Label Stack{2002}
005056A9BE8A005056A98CCF81000DC38847 007D2000
No output feature configured
Per-destination load-sharing, slots: 0 2 4 6 8 10 12 14
9003       92.0.0.8/32 0             Gi2.523    92.2.3.9
MAC/Encaps=18/22, MRU=1500, Label Stack{9003}
005056A9D672005056A98CCF81000DC38847 0232B000
No output feature configured
Per-destination load-sharing, slots: 1 3 5 7 9 11 13 15

```

Specifically, since the source is CSR10's loopback and the destination is CSR8's loopback, we can find the exact path using a show command. The packet will be sent to CSR9 in this specific case.

```

CSR3#show mpls forwarding-table exact-route label 3011 ipv4 source 92.0.0.10
destination 92.0.0.8
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id Switched      interface
3011       9003       92.0.0.8/32 0             Gi2.523    92.2.3.9

```

CSR9 performs PHP since it is the next-to-last hop towards CSR8. CSR8 instructs CSR9 to pop the topmost label by advertising an implicit-null label for this prefix. We can confirm this by checking the LIB as well.

```

R9#show mpls forwarding-table labels 9003
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop

```

Label	Label	or Tunnel Id	Switched	interface	
9003	Pop Label	92.0.0.8/32	3114	Gi2.589	92.8.9.8

```
R9#show mpls ldp bindings 92.0.0.8 32 neighbor 92.0.0.8
lib entry: 92.0.0.8/32, rev 10
remote binding: lsr: 92.0.0.8:0, label: imp-null
```

A quick traceroute on CSR10 shows the LSP as we traced it. It uses label 3011 to send traffic to CSR3 for prefix 92.0.0.8/32, then CSR3 swaps it to label 9003 based on the ECMP hash algorithm.

```
R10#traceroute 92.0.0.8 source 92.0.0.10
Type escape sequence to abort.
Tracing the route to 92.0.0.8
VRF info: (vrf in name/id, vrf out name/id)
 1 92.3.10.3 [MPLS: Label 3011 Exp 0] 4 msec 4 msec 5 msec
 2 92.2.3.9 [MPLS: Label 9003 Exp 0] 20 msec 20 msec 20 msec
 3 92.8.9.8 20 msec 11 msec 11 msec
```

We quickly look at CSR7 and XRv1 where there is a link that is not MPLS enabled. The route to 92.0.0.11/32 is labeled (implicit-null, still counts as labeled) yet the route to 92.0.0.13/32 is not. This is because of the way IGP converged; CSR7's route to XRv3 is via an OSPF area 1 route which traverses a non-MPLS link. CSR7's route to XRv1's loopback is via an OSPF area 0 route which traverses an MPLS enabled link.

Local	Outgoing	Prefix	Bytes	Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched		interface	
7009	Pop Label	92.0.0.11/32	4578		Gi2.517	92.11.7.11

Local	Outgoing	Prefix	Bytes	Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched		interface	
7011	No Label	92.0.0.13/32	0		Gi2.571	92.7.11.11

Even though CSR7 has labeled paths to XRv3, it cannot use these labels since IGP directs the route over a non-MPLS enabled interface. It would be great if CSR7 could use label 91005 to direct traffic through XRv1, since traffic is being sent to XRv1 anyway.

```
R7#show mpls ldp bindings 92.0.0.13 32
lib entry: 92.0.0.13/32, rev 46
local binding: label: 7011
remote binding: lsr: 92.0.0.11:0, label: 91005
remote binding: lsr: 92.0.0.6:0, label: 6012
```

```
R7#show ip route 92.0.0.13
Routing entry for 92.0.0.13/32
Known via "ospf 92", distance 110, metric 4, type intra area
```

```
Last update from 92.7.11.11 on GigabitEthernet2.571, 03:50:01 ago
Routing Descriptor Blocks:
* 92.7.11.11, from 92.0.0.13, 03:50:01 ago, via GigabitEthernet2.571
  Route metric is 4, traffic share count is 1
```

As a quick fix, we can apply a static route to CSR7 to direct traffic to this loopback towards XRv1 using the MPLS-enabled link. This wouldn't be possible with OSPF dynamically due to the intra-area vs. inter-area route preference. Static routes count as IGP routes from the perspective of LDP, which means LDP-bound labels can be used when a static route for a given prefix is installed in the RIB. Both the FIB and LFIB are updated to reflect this change.

```
! CSR7
ip route 92.0.0.13 255.255.255.255 GigabitEthernet2.517 92.11.7.11
```

```
R7#show ip route 92.0.0.13
Routing entry for 92.0.0.13/32
  Known via "static", distance 1, metric 0
  Routing Descriptor Blocks:
  * 92.11.7.11, via GigabitEthernet2.517
    Route metric is 0, traffic share count is 1
```

```
R7#show ip cef 92.0.0.13
92.0.0.13/32
  nexthop 92.11.7.11 GigabitEthernet2.517 label 91005
```

```
R7#show mpls forwarding-table 92.0.0.13 32
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id  Switched     interface
7011      91005      92.0.0.13/32    0            Gi2.517    92.11.7.11
```

We quickly confirm the LSP from CSR7 to XRv3 via XRv1 (VLAN 517) which overrides the OSPF topology. When XRv1 receives packets with label 91005, it swaps this label for 5005 which was CSR5's local label for 92.0.0.13/32. CSR5 pops the topmost label since it is the penultimate hop, which exposes the IP packet to XRv3. We use traceroute to confirm the full path and label operations along the way.

```
RP/0/0/CPU0:XRv1#show mpls forwarding labels 91005
Local  Outgoing  Prefix      Outgoing  Next Hop      Bytes
Label  Label      or ID       Interface  Hop           Switched
-----
91005  5005      92.0.0.13/32  Gi0/0/0/0.541  92.4.11.5    792
```

```
R5#show mpls forwarding-table labels 5005
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id  Switched     interface
5005      Pop Label  92.0.0.13/32    7194         Gi2.553    92.5.13.13
```

```
R7#traceroute 92.0.0.13 source 92.0.0.7
```

```
Type escape sequence to abort.
Tracing the route to 92.0.0.13
VRF info: (vrf in name/id, vrf out name/id)
  1 92.11.7.11 [MPLS: Label 91005 Exp 0] 9 msec 5 msec 5 msec
  2 92.4.11.5 [MPLS: Label 5005 Exp 0] 14 msec 15 msec 15 msec
  3 92.5.13.13 19 msec 13 msec 13 msec
```

An important concept in LDP is the RID. This has to be a routable address as we have seen earlier. Unlike most other protocols, this is used for session establishment and is more than just a unique ID formatted like an IPv4 address. This is how the TCP session is formed (can be adjusted as discussed later), but for clarity we can force the RID to be tied to a specific interface. On all XE routers, the following command is applied. If you specify the “force” keyword, it will reset the current session to use the new address. Excluding that option means that it will only change when it has the opportunity. (router reload, etc).

```
! All XE routers
mpls ldp router-id Loopback0 force
```

XR doesn’t give you the choice of interfaces, but allows you to specify the RID by IPv4 address. We also do this on every node, except the last octet changes on each router. Only XRv1 is shown for brevity.

```
! XRv1
mpls ldp
  router-id 92.0.0.11
```

We can tune a few options with the LDP with respect to timers. LDP maintains two separate sets of timers: discovery and maintenance. The discovery timers include the hello and hold down timers which is similar to OSPF, EIGRP, or IS-IS. These can be adjusted globally. Focusing on CSR8, we can see the timers are 5 seconds for hello and 15 seconds for hold down. We also see that there is some kind of negotiation for holddown timers since both the local and peer timers are shown.

```
R8#show mpls ldp discovery detail
Local LDP Identifier:
  92.0.0.8:0
Discovery Sources:
Interfaces:
  GigabitEthernet2.528 (ldp): xmit/recv
    Enabled: IGP config;
    Hello interval: 5000 ms; Transport IP addr: 92.0.0.8
    LDP Id: 92.0.0.2:0
    Src IP addr: 92.2.8.2; Transport IP addr: 92.0.0.2
    Hold time: 15 sec; Proposed local/peer: 15/15 sec
    [snip]
  GigabitEthernet2.589 (ldp): xmit/recv
    Enabled: IGP config;
    Hello interval: 5000 ms; Transport IP addr: 92.0.0.8
    LDP Id: 92.0.0.9:0
```

```
Src IP addr: 92.8.9.9; Transport IP addr: 92.0.0.9
Hold time: 15 sec; Proposed local/peer: 15/15 sec
[snip]
```

We can adjust these timers globally. On CSR8, we will reduce the hello timer to 3 and the hold time to 12. Looking at the discovery details again, we can see that the lower hold time is preferred, and routers can still use independent hello timers. This is somewhat similar to BGP or BFD timer negotiation since timers can be mismatched, but are ultimately converged on a common set of values for at least some of the timers.

```
! CSR8
mpls ldp discovery hello interval 3
mpls ldp discovery hello holdtime 12

R8#show mpls ldp discovery detail
Local LDP Identifier:
  92.0.0.8:0
Discovery Sources:
Interfaces:
  GigabitEthernet2.528 (ldp): xmit/recv
    Enabled: IGP config;
    Hello interval: 3000 ms; Transport IP addr: 92.0.0.8
    LDP Id: 92.0.0.2:0
    Src IP addr: 92.2.8.2; Transport IP addr: 92.0.0.2
    Hold time: 12 sec; Proposed local/peer: 12/15 sec
    [snip]
  GigabitEthernet2.589 (ldp): xmit/recv
    Enabled: IGP config;
    Hello interval: 3000 ms; Transport IP addr: 92.0.0.8
    LDP Id: 92.0.0.9:0
    Src IP addr: 92.8.9.9; Transport IP addr: 92.0.0.9
    Hold time: 12 sec; Proposed local/peer: 12/15 sec
    [snip]
```

Looking at CSR2 to confirm this, we can see the local hold time is 15 seconds and the remote is 12 seconds, yet 12 was selected. The routers will always agree on the discovery hold time. Also notice that CSR2's hello timer was automatically adjusted (but not in the configuration). The LDP discovery hello timer must be at least three times as frequent as the hold time. Because CSR2 had to reduce its hold time from 15 to 12 seconds, a hello timer of 5 seconds was too slow.

```
R2#show mpls ldp discovery detail
[snip]
GigabitEthernet2.528 (ldp): xmit/recv
  Enabled: IGP config;
  Hello interval: 4000 ms; Transport IP addr: 92.0.0.2
  LDP Id: 92.0.0.8:0
  Src IP addr: 92.2.8.8; Transport IP addr: 92.0.0.8
```

```
Hold time: 12 sec; Proposed local/peer: 15/12 sec
[snip]
```

We can also see this by looking at the neighbor details. This command also reveals the maintenance hold down timers as well, also known as the keep-alive or KA timer of 180 seconds (3 minutes) by default. The KA interval is also shown, which is 60 seconds by default.

```
R8#show mpls ldp neighbor 92.0.0.2 detail | include time
Up time: 00:41:42; UID: 7; Peer Id 2
    holdtime: 12000 ms, hello interval: 3000 ms
Peer holdtime: 180000 ms; KA interval: 60000 ms; Peer state: estab
```

We can change this in global configuration mode as well. This only affects new sessions, not existing ones, and the parser tells you this. We will clear the session between CSR8 and CSR2 to see the difference. Like the discovery hold down timer, the lower value is negotiated between the peers, and the KA interval is always one third of the KA hold time (not configurable). Both CSR2 and CSR8 are using the value of 120 seconds (2 minutes) for KA hold time and 40 seconds for KA interval.

```
R8(config)#mpls ldp holdtime 120
% Previously established sessions may not use the new holdtime.
```

```
R8#show mpls ldp neighbor 92.0.0.2 detail | include time
Up time: 00:00:23; UID: 8; Peer Id 0
    holdtime: 12000 ms, hello interval: 3000 ms
Peer holdtime: 120000 ms; KA interval: 40000 ms; Peer state: estab
```

```
R2#show mpls ldp neighbor 92.0.0.8 detail | include time
Up time: 00:00:45; UID: 12; Peer Id 3
    holdtime: 12000 ms, hello interval: 4000 ms
    holdtime: infinite, hello interval: 10000 ms
Peer holdtime: 120000 ms; KA interval: 40000 ms; Peer state: estab
```

The feature works similarly on XR. We configure some new values on XRv3; of note, the new discovery hold time is greater than the default on CSR5, so we expect 15 seconds to be used. The session holdtime is also too slow, so the default of 180 should be negotiated. XRv3 can still use its custom discovery hello interval of 4 seconds since this more than 3 times as frequent as the lowest holdtime.

```
! XRv3
mpls ldp
  session holdtime 240
  discovery
    hello holdtime 20
    hello interval 4
```

```
RP/0/0/CPU0:XRv3#show mpls ldp neighbor 92.0.0.5 detail | include KA
```

```
Peer holdtime: 180 sec; KA interval: 60 sec; Peer state: Estab
```

```
RP/0/0/CPU0:XRv3#show mpls ldp discovery detail
```

```
Local LDP Identifier: 92.0.0.13:0
```

```
Discovery Sources:
```

```
Interfaces:
```

```
GigabitEthernet0/0/0/0.553 (0xf00) : xmit/recv
```

```
VRF: 'default' (0x60000000)
```

```
Source address: 92.5.13.13; Transport address: 92.0.0.13
```

```
Hello interval: 4 sec (due in 7 msec)
```

```
Quick-start: Enabled
```

```
LDP Id: 92.0.0.5:0
```

```
Source address: 92.5.13.5; Transport address: 92.0.0.5
```

```
Hold time: 15 sec (local:20 sec, peer:15 sec)
```

```
(expiring in 11.4 sec)
```

There is also a protection mechanism built into LDP to prevent two LSRs from constantly trying to establish LDP peerings when they are incompatible. For example, perhaps there are significant version differences or other negotiated parameters/capabilities preventing the peer from forming. Constantly trying to establish sessions can tax an LSR's resources, so tuning the backoff timers is an option. The initial backoff timer (the first failure) is 15 seconds and the longest backoff time is 120 seconds, by default. We will adjust these to more aggressive values on CSR5 and XRv3 as a demonstration, although we can't reliably test it. The configuration and show commands are simple, and XR appears to display a table of backed-off sessions, as applicable.

```
! CSR5
```

```
mpls ldp backoff 5 60
```

```
! XRv3
```

```
mpls ldp
```

```
session backoff 5 60
```

```
R5#show mpls ldp backoff all
```

```
LDP initial/maximum backoff: 5/60 sec
```

```
RP/0/0/CPU0:XRv3#show mpls ldp backoff
```

```
Backoff Time:
```

```
Initial:5 sec, Maximum:60 sec
```

```
Backoff Table:
```

```
No Entry
```

Next, we will configure additional LDP features. The first and most commonly used is authentication. This gives MD5 protection to the TCP sessions between LDP peers. I will configure it everywhere in the topology using the simplest method to start. This says that a single "fallback" password is used for all

peers (as opposed to peer-specific passwords). XE also requires the operator to identify this security option as “required” or else it is considered optional.

```
! All XE routers
mpls ldp password fallback LDP_AUTH
mpls ldp password required

! All XR routers
mpls ldp
 neighbor
  password clear LDP_AUTH
```

Looking at CSR10, we can see that the MD5 password is required and in use. The fallback password is used since there are no specific passwords defined.

```
R10#show mpls ldp neighbor password
Peer LDP Ident: 92.0.0.14:0; Local LDP Ident 92.0.0.10:0
TCP connection: 92.0.0.14.19636 - 92.0.0.10.646
Password: required, fallback, in use
State: Oper; Msgs sent/rcvd: 73/74
Peer LDP Ident: 92.0.0.3:0; Local LDP Ident 92.0.0.10:0
TCP connection: 92.0.0.3.646 - 92.0.0.10.24379
Password: required, fallback, in use
State: Oper; Msgs sent/rcvd: 44/58
```

We can also quickly check the status of MD5 authentication by checking the neighbor details and filtering on the TCP sessions. This will show each neighbor in one line, along with its MD5 status. This also works on XR and is my favorite way to quickly check LDP neighbors for being up and authenticated.

```
R10#show mpls ldp neighbor detail | include TCP
TCP connection: 92.0.0.14.19636 - 92.0.0.10.646; MD5 on
TCP connection: 92.0.0.3.646 - 92.0.0.10.24379; MD5 on
```

```
RP/0/0/CPU0:XRv4#show mpls ldp neighbor detail | include TCP
TCP connection: 92.0.0.1:646 - 92.0.0.14:21418; MD5 on
TCP connection: 92.0.0.3:646 - 92.0.0.14:23273; MD5 on
TCP connection: 92.0.0.10:646 - 92.0.0.14:58993; MD5 on
```

Alternatively, we can go directly to the TCB and look for the MD5 option for a given TCP session. The TCP table brief reveals the TCBs, and we select the one representing the connection to CSR3. We can clearly see that the MD5 option was negotiated. Note that the “lossless password switchover” feature is enabled, which means that we can apply a key-chain with time constraints for automatic rollover if we wish.

```
R10#show tcp brief
TCB          Local Address          Foreign Address          (state)
```

```

7FBC417C11F8  92.0.0.10.646          92.0.0.14.19636      ESTAB
7FBC4148F918  92.0.0.10.24379          92.0.0.3.646         ESTAB

```

```

R10#show tcp tcb 7FBC4148F918 | section Option
Option Flags: non-blocking reads, non-blocking writes,
MD5 lossless password switchover, Retrans timeout

```

We can also specify per neighbor passwords. On CSR10, we will configure a customer password for the peer XRv4. Since we haven't configured it on XRv4 yet, the LDP neighbor will eventually fail. Both routers will immediately begin generating log messages since the MD5 authentication no longer matches. The key to this log message is port 646; this is how we can tell it is an LDP session versus any other TCP session (BGP, MSDP, etc). Once the password matches, the error messages cease.

```

! CSR10
mpls ldp neighbor 92.0.0.14 password LDP_AUTH_CUSTOM

! CSR6
%TCP-6-BADAUTH: Invalid MD5 digest from 92.0.0.14(58815) to 92.0.0.10(646)
tableid - 0

! XRv4
tcp[389]: %IP-TCP-3-BADAUTH : Invalid MD5 digest from 92.0.0.10:646 to
92.0.0.14:58815

```

The configuration on XR is very straightforward; we just need to define a more specific per-neighbor password for CSR10. XR also requires the label space (0 for system-wide) to be included for this command. If we check the neighbor password details on CSR10, we now see that XRv4 has a "neighbor" password as opposed to "fallback". CSR3 still uses the fallback password since no specific password was defined for that peer.

```

! XRv4
mpls ldp
neighbor
92.0.0.10:0 password clear LDP_AUTH_CUSTOM

R10#show mpls ldp neighbor password
Peer LDP Ident: 92.0.0.3:0; Local LDP Ident 92.0.0.10:0
TCP connection: 92.0.0.3.646 - 92.0.0.10.24379
Password: required, fallback, in use
State: Oper; Msgs sent/rcvd: 56/69
Peer LDP Ident: 92.0.0.14:0; Local LDP Ident 92.0.0.10:0
TCP connection: 92.0.0.14.58993 - 92.0.0.10.646
Password: required, neighbor, in use
State: Oper; Msgs sent/rcvd: 17/18

```

XR does not appear to support key-chains for LDP passwords, so we will implement password rollover between CSR6 and CSR7. These routers are running OSPF and have an LDP session between them. Both of them currently use the fallback password with all peers.

```
R6#show mpls ldp neighbor password
  Peer LDP Ident: 92.0.0.2:0; Local LDP Ident 92.0.0.6:0
    TCP connection: 92.0.0.2.646 - 92.0.0.6.13774
    Password: required, fallback, in use
    State: Oper; Msgs sent/rcvd: 1102/1093
  Peer LDP Ident: 92.0.0.7:0; Local LDP Ident 92.0.0.6:0
    TCP connection: 92.0.0.7.11800 - 92.0.0.6.646
    Password: required, fallback, in use
    State: Oper; Msgs sent/rcvd: 1066/1042
  Peer LDP Ident: 92.0.0.4:0; Local LDP Ident 92.0.0.6:0
    TCP connection: 92.0.0.4.646 - 92.0.0.6.22345
    Password: required, fallback, in use
    State: Oper; Msgs sent/rcvd: 1067/1070
[snip, lots of neighbors]
```

```
R7#show mpls ldp neighbor password
  Peer LDP Ident: 92.0.0.6:0; Local LDP Ident 92.0.0.7:0
    TCP connection: 92.0.0.6.646 - 92.0.0.7.11800
    Password: required, fallback, in use
    State: Oper; Msgs sent/rcvd: 1043/1068
  Peer LDP Ident: 92.0.0.11:0; Local LDP Ident 92.0.0.7:0
    TCP connection: 92.0.0.11.39991 - 92.0.0.7.646
    Password: required, fallback, in use
    State: Oper; Msgs sent/rcvd: 1028/1042
```

CSR6 and CSR7 will both implement a simple key-chain that uses two passwords. One is good for five minutes, while the second is good forever. This allows us to simply roll their clocks back to the time when the first key is valid, then not have to worry about it every again once the second key is valid. We use a simple symmetric key design (we could use different send/accept lifetimes to authenticate TCP segments in either direction). There is a one minute carry-over time, and we can notify LDP about this using the rollover commands. This means that once the next-key is active, the “rollover” process will take up to 1 minute to complete, which is fine in this case.

```
! CSR6 and CSR7
mpls ldp password rollover duration 1

key chain KC_LDP_AUTH
  key 1
    key-string LDP_AUTH_1
    accept-lifetime 00:00:00 May 23 2005 00:05:00 May 23 2005
    send-lifetime 00:00:00 May 23 2005 00:05:00 May 23 2005
    cryptographic-algorithm md5
  key 2
```

```
key-string LDP_AUTH_2
accept-lifetime 00:04:00 May 23 2005 infinite
send-lifetime 00:04:00 May 23 2005 infinite
cryptographic-algorithm md5
```

The mechanism to apply this keychain is a little odd. You define LDP password “options” which invoke the key-chains. These options are tied to ACLs that match LDP router-IDs that indicate to which neighbors this should apply. Neighbors for whom there is no specified option will continue to use the fallback password. Only CSR6’s configuration is shown since CSR7’s configuration is identical except that it creates ACL_R6 and references CSR6’s LDP router-ID.

```
! CSR6
ip access-list standard ACL_R7
 permit 92.0.0.7

mpls ldp password option 1 for ACL_R7 key-chain KC_LDP_AUTH
```

Checking CSR6 and CSR7, we can see this new password option in use. Other neighbors, such as CSR2 and XRv1, are not using this new option and continue to use the fallback password.

```
R6#show mpls ldp neighbor password
[snip]
  Peer LDP Ident: 92.0.0.2:0; Local LDP Ident 92.0.0.6:0
    TCP connection: 92.0.0.2.646 - 92.0.0.6.28586
    Password: required, fallback, in use
    State: Oper; Msgs sent/rcvd: 29/28
  Peer LDP Ident: 92.0.0.7:0; Local LDP Ident 92.0.0.6:0
    TCP connection: 92.0.0.7.48539 - 92.0.0.6.646
    Password: required, option 1 (KC_LDP_AUTH), in use
    State: Oper; Msgs sent/rcvd: 24/11

R7#show mpls ldp neighbor password
  Peer LDP Ident: 92.0.0.11:0; Local LDP Ident 92.0.0.7:0
    TCP connection: 92.0.0.11.39991 - 92.0.0.7.646
    Password: required, fallback, in use
    State: Oper; Msgs sent/rcvd: 1061/1074
  Peer LDP Ident: 92.0.0.6:0; Local LDP Ident 92.0.0.7:0
    TCP connection: 92.0.0.6.646 - 92.0.0.7.48539
    Password: required, option 1 (KC_LDP_AUTH), in use
    State: Oper; Msgs sent/rcvd: 14/28
```

All of the relevant LDP password logging is enabled by default, but I will apply the commands to CSR6 and CSR7 just in case. These commands cause the system to generate syslog messages when passwords are rolled over.

```
! CSR6 and CSR7
```

```
mpls ldp logging password rollover
mpls ldp logging password configuration
```

Next, we will adjust the clocks to be midnight on 23 May 2005. We immediately check the key chain to see that key 1 is now valid on one of the routers.

```
! CSR6 and CSR7
R7#clock set 00:00:00 23 may 2005

R7#show key chain KC_LDP_AUTH
Key-chain KC_LDP_AUTH:
  key 1 -- text "LDP_AUTH_1"
    accept lifetime (00:00:00 UTC May 23 2005) - (00:05:00 UTC May 23
2005) [valid now]
    send lifetime (00:00:00 UTC May 23 2005) - (00:05:00 UTC May 23 2005)
[valid now]
  key 2 -- text "LDP_AUTH_2"
    accept lifetime (00:04:00 UTC May 23 2005) - (infinite)
    send lifetime (00:04:00 UTC May 23 2005) - (infinite)
```

In about 1 minute (slightly less), the password changes from LDP_AUTH_2 to LDP_AUTH_1 since we manually changed the clock. The log message shows that the change occurred, and we expect to see another change in about 4 minutes. We aren't focusing on this change since it's an artificial one; we care more about crossing the time boundary naturally.

```
! CSR6
May 23 00:00:49.888: %LDP-5-PWDCFG: Password configuration changed for
92.0.0.7:0
```

```
! CSR7
May 23 00:00:57.923: %LDP-5-PWDCFG: Password configuration changed for
92.0.0.6:0
```

While we wait for the time to hit the 5 minute mark, notice that while the clock is in between 4:00 and 5:00 minutes, both keys are valid. This isn't really necessary since LDP will be rolling over password as soon as 5 minutes is up, but I wanted to demonstrate it.

```
R6#show clock
00:04:35.905 UTC Mon May 23 2005

R6#show key chain KC_LDP_AUTH
Key-chain KC_LDP_AUTH:
  key 1 -- text "LDP_AUTH_1"
    accept lifetime (00:00:00 UTC May 23 2005) - (00:05:00 UTC May 23
2005) [valid now]
    send lifetime (00:00:00 UTC May 23 2005) - (00:05:00 UTC May 23 2005)
[valid now]
```

```
key 2 -- text "LDP_AUTH_2"
    accept lifetime (00:04:00 UTC May 23 2005) - (infinite) [valid now]
    send lifetime (00:04:00 UTC May 23 2005) - (infinite) [valid now]
```

At the 5 minute mark (approximately), we see additional log messages to show the password changing again. This is LDP using key 2 in addition to key 1 for the purpose of rollover.

```
! CSR6
May 23 00:04:49.889: %LDP-5-PWDCFG: Password configuration changed for
92.0.0.7:0
```

```
! CSR7
May 23 00:04:57.922: %LDP-5-PWDCFG: Password configuration changed for
92.0.0.6:0
```

We can check the “pending” passwords on both routers to see the rollover in progress. Both of them show LDP password option 1 and the associated key chain marked as “stale”, which implied rollover is occurring.

```
R6#show mpls ldp neighbor password pending
Peer LDP Ident: 92.0.0.7:0; Local LDP Ident 92.0.0.6:0
TCP connection: 92.0.0.7.48539 - 92.0.0.6.646
Password: required, option 1 (KC_LDP_AUTH), stale (rollover)
State: Oper; Msgs sent/rcvd: 37/23
```

```
R7#show mpls ldp neighbor password pending
Peer LDP Ident: 92.0.0.6:0; Local LDP Ident 92.0.0.7:0
TCP connection: 92.0.0.6.646 - 92.0.0.7.48539
Password: required, option 1 (KC_LDP_AUTH), stale (rollover)
State: Oper; Msgs sent/rcvd: 23/37
```

After exactly 1 more minute, which is the LDP rollover duration timer, we can see the passwords change again. This signals the completion of the rollover process since key 1 is no longer valid for LDP usage. There are no more pending password changes on CSR6 and CSR7 since the rollover has completed. We can see the “current” passwords on CSR7 to see that option 1 is no longer stale, but is actively in use.

```
! CSR6
May 23 00:05:49.888: %LDP-5-PWDCFG: Password configuration changed for
92.0.0.7:0
```

```
! CSR7
May 23 00:05:57.923: %LDP-5-PWDCFG: Password configuration changed for
92.0.0.6:0
```

```
R6#show mpls ldp neighbor password pending
[no output]
```

```
R7#show mpls ldp neighbor password pending
```

```
[no output]
```

```
R7#show mpls ldp neighbor password current
```

```
Peer LDP Ident: 92.0.0.11:0; Local LDP Ident 92.0.0.7:0
TCP connection: 92.0.0.11.39991 - 92.0.0.7.646
Password: required, fallback, in use
State: Oper; Msgs sent/rcvd: 1081/1094
Peer LDP Ident: 92.0.0.6:0; Local LDP Ident 92.0.0.7:0
TCP connection: 92.0.0.6.646 - 92.0.0.7.48539
Password: required, option 1 (KC_LDP_AUTH), in use
State: Oper; Msgs sent/rcvd: 34/47
```

Shortly after the last configuration change, each router generates a syslog message once the rollover is confirmed. As you can see, the process is fairly involved but the logging is excellent.

```
! CSR6
```

```
May 23 00:06:10.453: %LDP-5-PWDRO: Password rolled over for 92.0.0.7:0
```

```
! CSR7
```

```
May 23 00:05:55.464: %LDP-5-PWDRO: Password rolled over for 92.0.0.6:0
```

Earlier I mentioned the concept of a transport address. This is simply the TCP source address for the session; by default, the router-ID is used. In some corner cases, this may need manual adjustment. For example, let's assume the link between XRv2 and CSR4 has a transparent firewall that only allows link-local traffic on port 646 for UDP and TCP. The TCP session between the loopbacks would be blocked by this firewall, so to work around it, we can source the TCP session from the connected interfaces for that link only. This means that all other sessions can continue to use the router-ID for their TCP connection. The word "discovery" is relevant because this applies to dynamically-discovered LDP peers only.

```
! CSR4
```

```
interface GigabitEthernet2.542
 mpls ldp discovery transport-address interface
```

```
! XRv2
```

```
mpls ldp
interface GigabitEthernet0/0/0/0.542
 address-family ipv4
 discovery transport-address interface
```

Using my favorite LDP show command, we quickly confirm that the transport addresses have been changed for the session between CSR4 and XRv2 only. The other sessions remain unaffected since the modification is relevant only to neighbors discovered on a given interface. This change has no effect on MPLS forwarding, label bindings, or anything of the sort.

```
R4#show mpls ldp neighbor detail | include TCP
```

```
TCP connection: 92.4.12.12.20508 - 92.4.12.4.646; MD5 on
TCP connection: 92.0.0.11.12011 - 92.0.0.4.646; MD5 on
TCP connection: 92.0.0.5.43590 - 92.0.0.4.646; MD5 on
TCP connection: 92.0.0.6.12154 - 92.0.0.4.646; MD5 on
```

```
RP/0/0/CPU0:XRv2#show mpls ldp neighbor detail | include TCP
TCP connection: 92.4.12.4:646 - 92.4.12.12:20508; MD5 on
TCP connection: 92.0.0.9:646 - 92.0.0.12:33163; MD5 on
TCP connection: 92.0.0.6:646 - 92.0.0.12:19816; MD5 on
```

If we check the discovery details, we can see that the entry for XRv2 differs slightly from the others. The specific IP address is listed since this is used as the remote transport address on this interface. The other discovery entries need not display this since it is assumed the LDP router-ID is used for TCP transport. XR is more explicit and clearly shows the transport address for all neighbors, even though where the transport address and LDP router-ID are equal. We can clearly see CSR4's custom transport address in XRv2's output.

```
R4#show mpls ldp discovery
Local LDP Identifier:
  92.0.0.4:0
Discovery Sources:
Interfaces:
  GigabitEthernet2.546 (ldp): xmit/recv
    LDP Id: 92.0.0.6:0
  GigabitEthernet2.541 (ldp): xmit/recv
    LDP Id: 92.0.0.11:0
    LDP Id: 92.0.0.5:0
  GigabitEthernet2.542 (ldp): xmit/recv
    LDP Id: 92.0.0.12:0; IP addr: 92.4.12.12
```

```
RP/0/0/CPU0:XRv2#show mpls ldp discovery
Local LDP Identifier: 92.0.0.12:0
Discovery Sources:
Interfaces:
  GigabitEthernet0/0/0/0.542 : xmit/recv
    VRF: 'default' (0x60000000)
    LDP Id: 92.0.0.4:0, Transport address: 92.4.12.4
    Hold time: 15 sec (local:15 sec, peer:15 sec)

  GigabitEthernet0/0/0/0.562 : xmit/recv
    VRF: 'default' (0x60000000)
    LDP Id: 92.0.0.6:0, Transport address: 92.0.0.6
    Hold time: 15 sec (local:15 sec, peer:15 sec)

  GigabitEthernet0/0/0/0.592 : xmit/recv
    VRF: 'default' (0x60000000)
    LDP Id: 92.0.0.9:0, Transport address: 92.0.0.9
    Hold time: 15 sec (local:15 sec, peer:15 sec)
```


Next, we will examine LDP session protection (SP). We know that the LDP session is actually a TCP session between peers since the LDP multicast hellos are just used for discovery. All of the label exchanges happen within the TCP exchanges. If a link between two routers fails, the LDP hello messages are not seen, and the router deletes the LDP session. The consequence of this action is that all of the labels learned from that peer are purged from the LIB. Before configuring SP, we will demonstrate this on CSR10 and CSR3. Shutting down the link to CSR10 on CSR3 causes the LDP neighbor to fail, as seen below. I show the label bindings for the 14 loopbacks being withdrawn. The TIB is synonymous with LIB and is legacy terminology.

```
R10#debug mpls ldp bindings
LDP Label Information Base (LIB) changes debugging is on
lcon: tibent(92.0.0.1/32): label 3003 from 92.0.0.3:0 removed
lcon: tibent(92.0.0.2/32): label 3007 from 92.0.0.3:0 removed
lcon: tibent(92.0.0.3/32): label imp-null from 92.0.0.3:0 removed
lcon: tibent(92.0.0.4/32): label 3004 from 92.0.0.3:0 removed
lcon: tibent(92.0.0.5/32): label 3010 from 92.0.0.3:0 removed
lcon: tibent(92.0.0.6/32): label 3014 from 92.0.0.3:0 removed
lcon: tibent(92.0.0.7/32): label 3009 from 92.0.0.3:0 removed
lcon: tibent(92.0.0.8/32): label 3011 from 92.0.0.3:0 removed
lcon: tibent(92.0.0.9/32): label 3012 from 92.0.0.3:0 removed
lcon: tibent(92.0.0.10/32): label 3000 from 92.0.0.3:0 removed
lcon: tibent(92.0.0.11/32): label 3008 from 92.0.0.3:0 removed
lcon: tibent(92.0.0.12/32): label 3001 from 92.0.0.3:0 removed
lcon: tibent(92.0.0.13/32): label 3005 from 92.0.0.3:0 removed
lcon: tibent(92.0.0.14/32): label 3002 from 92.0.0.3:0 removed
```

Since CSR3 is no longer an LDP neighbor, there are no bindings learned from that peer. We could have stored those labels temporarily to allow the link to come back up, and due to liberal label retention, could potentially re-route alternative MPLS paths via XRv4, since those labels would still be in the LIB.

```
R10#show mpls ldp bindings neighbor 92.0.0.3
[no output]
```

In large topologies, this might be a lot of information and originally took some time to exchange in the first place. At the cost of leaving those labels in memory, we can configure the routers to sustain their TCP session provided there is IP reachability between their transport addresses. This is why using the LDP router-ID as the transport address is generally desirable. Like authentication, we will enable the feature everywhere as a starting point.

```
! All XE routers
mpls ldp session protection
```

```
! All XR routers
mpls ldp
```

session protection

Entering this command doesn't generate any log messages since it isn't used for discovery. In addition to the link hellos, it establishes a targeted LDP (tLDP) session with each neighbor as well. Once a neighbor is dynamically-discovered, SP automatically created a tLDP hello towards it. This is a unicast hello that is analogous to an OSPF or EIGRP unicast hello on an NBMA interface, except it has a larger TTL. Looking at CSR10's LDP discovery cache, we can see a new stanza at the bottom to show targeted sessions. It introduces the terms "active" and "passive". Active implies that the local router is configured to originate this session while xmit means the router is actually sending hellos. Passive is the opposite, where the router is not explicitly configured to receive the connection but accepts it passively, and rcv complements this state by saying hellos are received. Long story short, we can see bidirectional tLDP hello exchanges with XRv4 and CSR3.

```
R10#show mpls ldp discovery
Local LDP Identifier:
 92.0.0.10:0
Discovery Sources:
Interfaces:
  GigabitEthernet2.530 (ldp): xmit/rcv
    LDP Id: 92.0.0.3:0
  GigabitEthernet2.504 (ldp): xmit/rcv
    LDP Id: 92.0.0.14:0
Targeted Hellos:
 92.0.0.10 -> 92.0.0.3 (ldp): active/passive, xmit/rcv
    LDP Id: 92.0.0.3:0
 92.0.0.10 -> 92.0.0.14 (ldp): active/passive, xmit/rcv
    LDP Id: 92.0.0.14:0
```

If we look at the discovery details for these targeted sessions, we can see their origin. Later, we will see tLDP used for L2VPN, TE, and many other applications. In this case, the sessions were generated by the LDP session protection (LDP SP) feature. We can clearly see the transport addresses as shown below, and as an added benefit, these targeted sessions are authenticated. This is NOT an additional TCP connection, since there is only one TCP connection per set of neighbors regardless of how many adjacencies they have.

```
R10#show mpls ldp discovery detail | begin Target
Targeted Hellos:
 92.0.0.10 -> 92.0.0.3 (ldp): active/passive, xmit/rcv
  Enabled by: LDP SP,
    Hello interval: 10000 ms; Transport IP addr: 92.0.0.10
    LDP Id: 92.0.0.3:0
    Src IP addr: 92.0.0.3; Transport IP addr: 92.0.0.3
    Hold time: 90 sec; Proposed local/peer: 90/90 sec
    Reachable via 92.0.0.3/32
    Password: required, fallback, in use
 92.0.0.10 -> 92.0.0.14 (ldp): active/passive, xmit/rcv
```

```

Enabled by: LDP SP,
  Hello interval: 10000 ms; Transport IP addr: 92.0.0.10
  LDP Id: 92.0.0.14:0
    Src IP addr: 92.0.0.14; Transport IP addr: 92.0.0.14
  Hold time: 90 sec; Proposed local/peer: 90/90 sec
  Reachable via 92.0.0.14/32
  Password: required, neighbor, in use

```

Next, we shut down CSR3's interface to CSR10 again. CSR10 will re-route traffic to XRv4 in its LFIB, but still retains labels from CSR3. Technically, the LDP neighbor is still up, and the debug does not reveal any label purging. The neighbor discovery sources no longer include VLAN 530 as that link was shutdown, but the targeted hello continues to keep the session alive.

```

R10#show mpls ldp neighbor 92.0.0.3
  Peer LDP Ident: 92.0.0.3:0; Local LDP Ident 92.0.0.10:0
    TCP connection: 92.0.0.3.646 - 92.0.0.10.31875
    State: Oper; Msgs sent/rcvd: 7/24; Downstream
    Up time: 00:02:46
    LDP discovery sources:
      Targeted Hello 92.0.0.10 -> 92.0.0.3, active, passive
    Addresses bound to peer LDP Ident:
      92.0.0.3          92.2.3.3          92.3.14.3

```

A quick look at the LIB to query labels from CSR3 shows no changes. All of the labels are still present, but none will be used for forwarding as the LFIB shows.

```

R10#show mpls ldp bindings neighbor 92.0.0.3
  lib entry: 92.0.0.1/32, rev 145
    remote binding: lsr: 92.0.0.3:0, label: 3003
  lib entry: 92.0.0.2/32, rev 135
    remote binding: lsr: 92.0.0.3:0, label: 3007
  lib entry: 92.0.0.3/32, rev 94
    remote binding: lsr: 92.0.0.3:0, label: imp-null
  lib entry: 92.0.0.4/32, rev 140
    remote binding: lsr: 92.0.0.3:0, label: 3004
[snip]

```

```

R10#show mpls forwarding-table

```

Local	Outgoing	Prefix	Bytes Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched	interface	
10000	94005	92.0.0.3/32	0	Gi2.504	92.10.14.14
10001	94004	92.0.0.6/32	0	Gi2.504	92.10.14.14
10002	9413	92.0.0.13/32	0	Gi2.504	92.10.14.14
10003	94012	92.0.0.5/32	0	Gi2.504	92.10.14.14

```

[snip]

```

Assuming we had some debugs running when the link was shut down, we can actually see what SP was doing behind the scenes. A hold-up timer of 86400 seconds (24 hours) begins once the link fails, which means that after this amount of time, SP will stop holding the session active. SP moves from the ready state to the protecting state and starts the hold-up timer. At some point it makes sense to flush stale labels from the LIB, and 24 hours is the default so that routine link maintenance that makes more than a few minutes (cleaning fiber ports, rearranging cable runs, etc) doesn't introduce LIB churn. We can see these defaults by checking the neighbor details, which also shows the time remaining for the SP hold-up.

```
R10#debug mpls ldp session protection
LDP session protection events debugging is on

! CSR10
LDP SP: 92.0.0.3:0: last primary adj lost; starting session protection holdup
timer
LDP SP: 92.0.0.3:0: LDP session protection holdup timer started, 86400
seconds
LDP SP: 92.0.0.3:0: state change (Ready -> Protecting)
%LDP-5-SP: 92.0.0.3:0: session hold up initiated

R10#show mpls ldp neighbor 92.0.0.3 detail | begin Session Protect
LDP Session Protection enabled, state: Protecting
    duration: 86400 seconds
    holdup time remaining: 86155 seconds
```

When the link comes back up, CSR10 stops the SP hold-up timer and changes the protection state back to ready. The hold-up timer stops counting and is removed from the LDP neighbor output.

```
! CSR10
LDP SP: 92.0.0.3:0: primary adj restored; stopping session protection holdup
timer
LDP SP: 92.0.0.3:0: state change (Protecting -> Ready)
%LDP-5-SP: 92.0.0.3:0: session recovery succeeded

R10#show mpls ldp neighbor 92.0.0.3 detail | begin Session Protect
LDP Session Protection enabled, state: Ready
    duration: 86400 seconds
```

Let's pretend that CSR10 is low on memory and should not retain these labels for a long time. We can reduce the hold-up timer to a shorter time, say 60 seconds, so that a link outage longer than that will flush labels from the LIB. This timer is only locally significant and does not have to match throughout the network. It does, however, mean that CSR3 will continue to store all of CSR10's labels for 24 hours, which doesn't make sense from a design perspective when the peer uses a different timer. The minimum time is 30 seconds and the maximum time is infinite (never flush labels from the LIB, generally a bad idea). We quickly check to ensure the configuration worked and that SP is ready.

```
! CSR10
mpls ldp session protection duration 60
```

```
R10#show mpls ldp neighbor 92.0.0.3 detail | begin Session Protect
      LDP Session Protection enabled, state: Ready
      duration: 60 seconds
```

When CSR3 shuts down its interface to CSR10, SP activates. This is no different than earlier except we see the timer is now 60 seconds. CSR10 still has all of CSR3's labels in the LIB as well.

```
! CSR10
LDP SP: 92.0.0.3:0: last primary adj lost; starting session protection holdup
timer
LDP SP: 92.0.0.3:0: LDP session protection holdup timer started, 60 seconds
LDP SP: 92.0.0.3:0: state change (Ready -> Protecting)
%LDP-5-SP: 92.0.0.3:0: session hold up initiated
```

```
R10#show mpls ldp neighbor 92.0.0.3 detail | begin Session Protect
      LDP Session Protection enabled, state: Protecting
      duration: 60 seconds
      holdup time remaining: 41 seconds
```

```
R10#show mpls ldp bindings neighbor 92.0.0.3
lib entry: 92.0.0.1/32, rev 145
    remote binding: lsr: 92.0.0.3:0, label: 3003
lib entry: 92.0.0.2/32, rev 135
    remote binding: lsr: 92.0.0.3:0, label: 3007
lib entry: 92.0.0.3/32, rev 94
    remote binding: lsr: 92.0.0.3:0, label: imp-null
lib entry: 92.0.0.4/32, rev 140
[snip]
```

After the timer expires and the link doesn't come up, SP moves the session into state "none" which effectively tears down the session. This flushes all labels from the LIB, as designed, since the neighbor no longer exists.

```
! CSR10
LDP SP: 92.0.0.3:0: LDP session protection holdup timer expired
LDP SP: 92.0.0.3:0: disabling session protection: holdup timer expired
LDP SP: 92.0.0.3:0: state change (Protecting -> None)
%LDP-5-SP: 92.0.0.3:0: session recovery failed
```

```
R10#show mpls ldp bindings neighbor 92.0.0.3
[no output]
```

```
R10#show mpls ldp neighbor 92.0.0.3
[no output]
```

Since this feature works identically in XR, we won't test it in detail. However, the detailed LDP neighbor command shows similar output to XE; this shows the SP state, hold-up (duration) timer, and any ACLs applied. ACLs are discussed next.

```
RP/0/0/CPU0:XRv1#show mpls ldp neighbor 92.0.0.4 detail | begin Session Prot
Clients: Session Protection
Session Protection:
  Enabled, state: Ready
  Duration: 86400 sec
```

SP can also be configured to only protect certain peers by supplying an ACL. This makes sense on nodes where there is only one path, and the overhead from the targeted sessions adds no value. As such, we will remove session protection from CSR1 and XRv3 entirely (not shown). However, CSR5 and XRv4 will still try to offer SP to those routers by default since we did not filter it. Notice that both CSR5 and XRv4 are trying to run SP with XRv3 and CSR1 respectively. The sessions are active and the routers are sending tLDP hellos, but there is nothing coming back. This is a waste of resources on CSR5 and XRv4.

```
R5#show mpls ldp discovery | begin Target
Targeted Hellos:
  92.0.0.5 -> 92.0.0.4 (ldp): active/passive, xmit/recv
    LDP Id: 92.0.0.4:0
  92.0.0.5 -> 92.0.0.11 (ldp): active/passive, xmit/recv
    LDP Id: 92.0.0.11:0
  92.0.0.5 -> 92.0.0.13 (ldp): active, xmit
```

```
RP/0/0/CPU0:XRv4#show mpls ldp discovery | begin Target
Targeted Hellos:
  92.0.0.14 -> 92.0.0.1 (active), xmit

  92.0.0.14 -> 92.0.0.3 (active), xmit/recv
    LDP Id: 92.0.0.3:0
    Hold time: 90 sec (local:90 sec, peer:90 sec)

  92.0.0.14 -> 92.0.0.10 (active), xmit/recv
    LDP Id: 92.0.0.10:0
    Hold time: 90 sec (local:90 sec, peer:90 sec)
```

To correct this inefficiency, we configure XRv4 and CSR5 not to offer the feature to those stub routers. If the link hello adjacency fails, there is no other way to reach those routers, so SP is not valuable. I use different ACL logic for variety, where CSR5 shows a stricter control to ensure only loopback addresses can be SP targets.

```
! CSR5
ip access-list standard ACL_LDP_SESS_PROTECT
deny 92.0.0.13
permit 92.0.0.0 0.0.0.255
```

```
mpls ldp session protection for ACL_LDP_SESS_PROTECT
```

```
! XRv4
```

```
ipv4 access-list ACL_LDP_SESS_PROTECT
```

```
10 deny ipv4 host 92.0.0.1 any
```

```
20 permit ipv4 any any
```

```
mpls ldp
```

```
session protection for ACL_LDP_SESS_PROTECT
```

Before checking the LDP discovery updates, we can ensure the configurations were successful. Since the command is configured globally, we can select any neighbor to see the ACL applied.

```
R5#show mpls ldp neighbor 92.0.0.11 detail | begin Session Prot
```

```
LDP Session Protection enabled, state: Ready
```

```
acl: ACL_LDP_SESS_PROTECT, duration: 86400 seconds
```

```
RP/0/0/CPU0:XRv4#show mpls ldp neighbor 92.0.0.10 detail | begin Session Prot
```

```
Clients: Session Protection
```

```
Session Protection:
```

```
Enabled, state: Ready
```

```
ACL: 'ACL_LDP_SESS_PROTECT', Duration: 86400 sec
```

We can check CSR5 to ensure it is not “actively” sending targeted hellos to 92.0.0.13. Notice that only CSR4 and XRv1 are listed as targets for tLDP hellos. The same is true for XRv4 which only targets CSR3 and CSR10 now.

```
R5#show mpls ldp discovery | begin Target
```

```
Targeted Hellos:
```

```
92.0.0.5 -> 92.0.0.4 (ldp): active/passive, xmit/recv
```

```
LDP Id: 92.0.0.4:0
```

```
92.0.0.5 -> 92.0.0.11 (ldp): active/passive, xmit/recv
```

```
LDP Id: 92.0.0.11:0
```

```
RP/0/0/CPU0:XRv4#show mpls ldp discovery | begin Target
```

```
Targeted Hellos:
```

```
92.0.0.14 -> 92.0.0.3 (active), xmit/recv
```

```
LDP Id: 92.0.0.3:0
```

```
Hold time: 90 sec (local:90 sec, peer:90 sec)
```

```
92.0.0.14 -> 92.0.0.10 (active), xmit/recv
```

```
LDP Id: 92.0.0.10:0
```

```
Hold time: 90 sec (local:90 sec, peer:90 sec)
```

Although it may not seem obvious, you can emulate SP without enabling the specific feature. SP is a very simple logic: when a peer is dynamically discovered, create a tLDP session to it, and if the primary

adjacencies (links) fail, continue to use a targeted session just to maintain the LIB. SP does not attempt to sustain MPLS forwarding as this is the job of FRR. We could just manually configure a tLDP session between two routers to achieve the same effect. We will demonstrate this on XRv1 and CSR7. This also implies we should disable SP for those peers, which we accomplish with an ACL again. Even though these routers only share one MPLS link, the non-MPLS link can still backup a session.

```
! CSR7
ip access-list standard ACL_LDP_SESS_PROTECT
  deny 92.0.0.11
  permit 92.0.0.0 0.0.0.255

mpls ldp session protection for ACL_LDP_SESS_PROTECT

! XRv1
ipv4 access-list ACL_LDP_SESS_PROTECT
  10 deny ipv4 host 92.0.0.7 any
  20 permit ipv4 92.0.0.0 0.0.0.255 any

mpls ldp
  session protection for ACL_LDP_SESS_PROTECT
```

Next, we will manually configure the tLDP sessions. The configuration is very simple on both XE and XR.

```
! CSR7
mpls ldp neighbor 92.0.0.11 targeted ldp

! XRv1
mpls ldp
  address-family ipv4
  neighbor 92.0.0.7 targeted
```

Looking at the targeted hellos on CSR7, we can clearly see a difference between this manual session and the SP session to CSR6. The session to XRv1 is identified as “LDP config” which means it was manually configured. Other than that, everything else looks the same, which means it should perform the same function as SP.

```
R7#show mpls ldp discovery detail | begin Targeted
Targeted Hellos:
  92.0.0.7 -> 92.0.0.6 (ldp): active/passive, xmit/recvd
    Enabled by: LDP SP,
    Hello interval: 10000 ms; Transport IP addr: 92.0.0.7
    LDP Id: 92.0.0.6:0
    Src IP addr: 92.0.0.6; Transport IP addr: 92.0.0.6
    Hold time: 90 sec; Proposed local/peer: 90/90 sec
    Reachable via 92.0.0.6/32
    Password: required, option 1 (KC_LDP_AUTH), in use
  92.0.0.7 -> 92.0.0.11 (ldp): active/passive, xmit/recvd
```


Enabled by: LDP Config,

```
Hello interval: 10000 ms; Transport IP addr: 92.0.0.7
LDP Id: 92.0.0.11:0
  Src IP addr: 92.0.0.11; Transport IP addr: 92.0.0.11
  Hold time: 90 sec; Proposed local/peer: 90/90 sec
  Reachable via 92.0.0.11/32
  Password: required, fallback, in use
```

If the primary MPLS link between the two routers fail, CSR7 will still have other paths to reach XRv1, and vice versa. As such, the failure of the link should not affect the LDP session since a static tLDP hello exchange will concur to occur, provided there is IP reachability. Below, we can see that VLAN 517 is no longer a discovery source for 92.0.0.11:0, but the tLDP session remains. The same is true for XRv1 who can no longer discover CSR7 dynamically, but maintains the tLDP session.

```
R7#show mpls ldp discovery
Local LDP Identifier:
  92.0.0.7:0
Discovery Sources:
Interfaces:
  GigabitEthernet2.567 (ldp): xmit/recv
    LDP Id: 92.0.0.6:0
Targeted Hellos:
  92.0.0.7 -> 92.0.0.6 (ldp): active/passive, xmit/recv
    LDP Id: 92.0.0.6:0
  92.0.0.7 -> 92.0.0.11 (ldp): active/passive, xmit/recv
    LDP Id: 92.0.0.11:0
```

```
RP/0/0/CPU0:XRv1#show mpls ldp discovery 92.0.0.7:0
Local LDP Identifier: 92.0.0.11:0
Discovery Sources:
  Targeted Hellos:
    92.0.0.11 -> 92.0.0.7 (active), xmit/recv
      LDP Id: 92.0.0.7:0
      Hold time: 90 sec (local:90 sec, peer:90 sec)
```

A quick check of the LIB on both CSR7 and XRv1 shows that the labels have been exchanged. Unlike the SP default hold-up timer, this session will stay up forever, and could be a workaround for environments where not all routers support the dynamic SP feature.

```
R7#show mpls ldp bindings neighbor 92.0.0.11
lib entry: 92.0.0.2/32, rev 36
  remote binding: lsr: 92.0.0.11:0, label: 91007
lib entry: 92.0.0.3/32, rev 37
  remote binding: lsr: 92.0.0.11:0, label: 91008
lib entry: 92.0.0.4/32, rev 38
  remote binding: lsr: 92.0.0.11:0, label: 91000
```

```
RP/0/0/CPU0:XRv1#show mpls ldp bindings neighbor 92.0.0.7:0
92.0.0.2/32, rev 23
```

```
Local binding: label: 91007
```

```
Remote bindings: (4 peers)
```

Peer	Label
92.0.0.7:0	7001

```
92.0.0.3/32, rev 51
```

```
Local binding: label: 91008
```

```
Remote bindings: (4 peers)
```

Peer	Label
92.0.0.7:0	7002

Note: The targeted session is not a magic trick you can use to fix broken LSPs. The MPLS forwarding table is still based on IP routing, and there isn't a path to XRv1 anymore. Just because the labels are preserved doesn't mean they will be used; a snapshot of CSR7 confirms this as CSR6 is the preferred path for many of the router loopbacks now. Unless there is an MPLS-enabled interface to the neighbor, the tLDP session is just to preserve labels, not to directly fix forwarding issues.

```
R7#show mpls forwarding-table
```

Local	Outgoing	Prefix	Bytes	Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched		interface	
7001	6000	92.0.0.2/32	0		Gi2.567	92.6.7.6
7002	6007	92.0.0.3/32	0		Gi2.567	92.6.7.6
7003	6009	92.0.0.4/32	82646		Gi2.567	92.6.7.6

[snip]

As discussed earlier, there are discovery and maintenance timers, and these apply to targeted LDP sessions as well. The maintenance timer isn't specific to dynamic or targeted session since there is only one LDP session between routers, so we won't look at adjusting that again. However, XE and XR appear to have different behaviors with regards to targeted LDP session hold timers for discovery. XE claims the hold time is infinite while XR claims it is 90 seconds (or 9 times the hello interval of 10 seconds). I believe this may be a cosmetic error on XE because XR states that both the local and remote hold times are 90 seconds.

```
R7#show mpls ldp neighbor 92.0.0.11 detail | begin LDP disc
```

```
LDP discovery sources:
```

```
Targeted Hello 92.0.0.7 -> 92.0.0.11, active, passive;
```

```
holdtime: infinite, hello interval: 10000 ms
```

```
GigabitEthernet2.517; Src IP addr: 92.11.7.11
```

```
holdtime: 15000 ms, hello interval: 5000 ms
```

```
RP/0/0/CPU0:XRv1#show mpls ldp discovery 92.0.0.7:0 | begin Target
```

```
Targeted Hellos:
```

```
92.0.0.11 -> 92.0.0.7 (active), xmit/recv
```

```
LDP Id: 92.0.0.7:0
Hold time: 90 sec (local:90 sec, peer:90 sec)
```

By checking the LDP parameters, we can further solidify our claim that this is a cosmetic output issue on XE. CSR7 claims that its tLDP hello holdtime is 90 seconds by default. It makes sense that the two platforms would agree on these defaults.

```
R7#show mpls ldp parameters | include time
Session hold time: 180 sec; keep alive interval: 60 sec
Discovery hello: holdtime: 15 sec; interval: 5 sec
Discovery targeted hello: holdtime: 90 sec; interval: 10 sec
```

```
RP/0/0/CPU0:XRv1#show mpls ldp parameters | include time
Hold time: 180 sec
Link Hellos:      Holdtime:15 sec, Interval:5 sec
Targeted Hellos: Holdtime:90 sec, Interval:10 sec
Housekeeping periodic timer: 10 sec
```

If we change CSR7 to a lower value, we would expect XRv1 to negotiate to that lesser value. This is also a global configuration parameter. We can see that changing CSR7 adjusts the local parameters, and XRv1 negotiates this timer for the specific session (discovery only) with CSR7.

```
! CSR7
mpls ldp discovery targeted-hello holdtime 80

R7#show mpls ldp parameters | include time
Session hold time: 180 sec; keep alive interval: 60 sec
Discovery hello: holdtime: 15 sec; interval: 5 sec
Discovery targeted hello: holdtime: 80 sec; interval: 10 sec
```

```
RP/0/0/CPU0:XRv1#show mpls ldp discovery 92.0.0.7:0 | begin Target
Targeted Hellos:
  92.0.0.11 -> 92.0.0.7 (active), xmit/recv
    LDP Id: 92.0.0.7:0
      Hold time: 80 sec (local:90 sec, peer:80 sec)
```

If we configure an even lower timer on XRv1, we expect XRv1's local parameters to change and CSR7 to agree to this lower value. Because of XE's output format, the value always says "infinite" but we can check on XRv1 to see the 70 second and 80 second parameters being exchanged. 70 seconds is what is negotiated between the peers.

```
! XRv1
mpls ldp
  discovery
    targeted-hello holdtime 70
```

```
RP/0/0/CPU0:XRv1#show mpls ldp parameters | include time
  Hold time: 180 sec
  Link Hellos:      Holdtime:15 sec, Interval:5 sec
  Targeted Hellos:  Holdtime:70 sec, Interval:10 sec
  Housekeeping periodic timer: 10 sec

RP/0/0/CPU0:XRv1#show mpls ldp discovery 92.0.0.7:0 | begin Target
  Targeted Hellos:
    92.0.0.11 -> 92.0.0.7 (active), xmit/recv
    LDP Id: 92.0.0.7:0
      Hold time: 70 sec (local:70 sec, peer:80 sec)
```

Next, we will look at another powerful LDP feature known as IGP synchronization or simply “IGP sync”. IGP sync is meant to solve the problem of IGP and LDP having two different views, or being in two different states, of a particular link. Packet loss can occur if IGP and LDP are not synchronized; I will illustrate an example. A new IGP adjacency forms on a link. The IGP converges quickly and runs SPF for all remote prefixes before the LDP session fully forms (distributes labels, etc). Traffic will be routed out of this link as raw IP traffic until the LFIB is programmed with the proper labels, and this will break any LSPs using this interface for a short time. We can demonstrate this on CSR4; the shortest-path to CSR7 is via the directly connected link to CSR6. If LDP fails on this link, but OSPF remains intact, the RIB will still prefer this link despite it not being MPLS-capable.

```
R4#show ip cef 92.0.0.7
92.0.0.7/32
  nexthop 92.4.6.6 GigabitEthernet2.546 label 6008
```

We can kill the LDP session many ways. One way is to apply an ACL that denies UDP/TCP 646 on the interface. Another option is to create a static null route for 92.0.0.6/32, which is the transport address for the LDP session. I will use the latter approach. We can manually clear the neighbor to speed things up.

```
! CSR6
ip route 92.0.0.6 255.255.255.255 null0

R4#clear mpls ldp neighbor 92.0.0.6
%LDP-5-NBRCHG: LDP Neighbor 92.0.0.6:0 (5) is DOWN (User cleared session manually)
```

Despite this, OSPF remains intact, and CSR7 is still reachable via this link. We can see that the traffic is not MPLS encapsulated since the FIB no longer has a label binding for this prefix out of this interface. Although this particular fabricated failure isn’t realistic, it gives us time to verify the fault; IGP sync is meant to protect against short-term black-holes due to convergence issues, but also protects against blatant mis-configurations like this as well. If the entire link went down, OSPF would converge, but as of now the network is in a broken state.

```

R4#show ip cef 92.0.0.7
92.0.0.7/32
  nexthop 92.4.6.6 GigabitEthernet2.546

R4#traceroute 92.0.0.7 source 92.0.0.4
Type escape sequence to abort.
Tracing the route to 92.0.0.7
VRF info: (vrf in name/id, vrf out name/id)
 1 92.4.6.6 3 msec 3 msec 3 msec
 2 92.6.7.7 4 msec 4 msec 3 msec

```

Before continuing, we will remove the static route so the network is repaired. IGP sync can be enabled within the OSPF or IS-IS processes to apply to all IGP-enabled links. Like LDP auto-config, it makes sense to use this approach when there are many interfaces requiring synchronization. In XR, this is only supported for OSPF, as IS-IS requires it explicitly on a per-link basis. We will use the process-level approach on many routers. XRv2 shows IGP sync enabled on a per-interface basis for both IS-IS and OSPF. XE only allows IGP sync to be enabled per-process. Basically, sync is enabled everywhere except towards CSR1 and XRv3. CSR5 explicitly disables IGP sync on this interface, whereas XRv4 never enabled it towards CSR1 in the first place. The same is true for CSR7 and XRv1 on the non-MPLS link they share.

```

! CSR4, CSR5, CSR6, CSR7, XRv1
router ospf 92
  mpls ldp sync

! CSR5 only
interface GigabitEthernet2.553
  no mpls ldp igp sync

! CSR7 only
interface GigabitEthernet2.571
  no mpls ldp igp sync

! XRv1 only
router ospf 92
  area 1
    interface GigabitEthernet0/0/0/0.571
      mpls ldp sync disable

! CSR2, CSR3, CSR8, CSR9, CSR10
router isis LDP
  mpls ldp sync

! XRv4
router isis LDP
  interface GigabitEthernet0/0/0/0.504
    address-family ipv4 unicast
      mpls ldp sync
  interface GigabitEthernet0/0/0/0.534

```

```

address-family ipv4 unicast
  mpls ldp sync

! XRV2
router ospf 92
  area 0
    interface GigabitEthernet0/0/0/0.542
      mpls ldp sync

    interface GigabitEthernet0/0/0/0.562
      mpls ldp sync

router isis LDP
  interface GigabitEthernet0/0/0/0.592
    address-family ipv4 unicast
      mpls ldp sync

```

We can spot-check a few nodes to ensure IGP sync is working properly. Looking at CSR3, we can see 3 interfaces enabled for IGP sync even though there are 4 LDP neighbors. Since this is a per-interface configuration, the LDP peer router-IDs are listed in each interface stanza. We can see that sync is enabled and that the peers are reachable. This is the expected state for IGP sync when the network is stable.

```

R3#show mpls ldp igp sync
  GigabitEthernet2.523:
    LDP configured; LDP-IGP Synchronization enabled.
    Sync status: sync achieved; peer reachable.
    Sync delay time: 0 seconds (0 seconds left)
    IGP holddown time: infinite.
    Peer LDP Ident: 92.0.0.9:0; 92.0.0.2:0
    IGP enabled: ISIS LDP
  GigabitEthernet2.530:
    LDP configured; LDP-IGP Synchronization enabled.
    Sync status: sync achieved; peer reachable.
    Sync delay time: 0 seconds (0 seconds left)
    IGP holddown time: infinite.
    Peer LDP Ident: 92.0.0.10:0
    IGP enabled: ISIS LDP
  GigabitEthernet2.534:
    LDP configured; LDP-IGP Synchronization enabled.
    Sync status: sync achieved; peer reachable.
    Sync delay time: 0 seconds (0 seconds left)
    IGP holddown time: infinite.
    Peer LDP Ident: 92.0.0.14:0
    IGP enabled: ISIS LDP

```

On XRV4, we see similar results. Sync is working to CSR10 and CSR3 but not CSR1. It was never configured to CSR1 since there are no redundant paths. The parenthesis message is a little misleading,

but the overall status of “not ready” means that sync isn’t enabled to CSR1 as expected. A quick look at CSR5 shows similar output as IGP sync is disabled towards XRv3 as well.

```
RP/0/0/CPU0:XRv4#show mpls ldp igp sync
GigabitEthernet0/0/0/0.504:
  VRF: 'default' (0x60000000)
  Sync delay: Disabled
  Sync status: Ready
  Peers:
    92.0.0.10:0

GigabitEthernet0/0/0/0.514:
  VRF: 'default' (0x60000000)
  Sync delay: Disabled
  Sync status: Not ready (Initial update to peer not done yet)

GigabitEthernet0/0/0/0.534:
  VRF: 'default' (0x60000000)
  Sync delay: Disabled
  Sync status: Ready
  Peers:
    92.0.0.3:0

R5#show mpls ldp igp sync
GigabitEthernet2.541:
  LDP configured; LDP-IGP Synchronization enabled.
  Sync status: sync achieved; peer reachable.
  Sync delay time: 0 seconds (0 seconds left)
  IGP holddown time: infinite.
  Peer LDP Ident: 92.0.0.11:0; 92.0.0.4:0
  IGP enabled: OSPF 92
  GigabitEthernet2.553:
    LDP configured; LDP-IGP Synchronization not enabled.
```

The mechanics of IGP sync are very simple. When an LDP session fails on a link where IGP is still enabled, the sync process will raise the link cost to the maximum so that the link is less preferred than any other alternative. Going back to CSR4, we re-enable the static null route to 92.0.0.6/32 which breaks the LDP session. We can debug IGP sync at the same time to watch what happens in the background. After flapping the neighbor, it never comes back but sync takes action by notifying OSPF about the change.

```
R4#debug mpls ldp igp sync
LDP-IGP Synchronization debugging is on

LDP-SYNC: Gi2.546, OSPF 92: notify status (required, not achieved, delay,
holddown infinite) internal status (not achieved, timer not running)
LDP-SYNC: Gi2.546, 92.0.0.6: Adj being deleted, sync_achieved goes down
```

We can see that CSR4 now prefers a valid labeled path via XRv2. However, when we check the OSPF interface costs, nothing has changed. Without digging deeper, IGP sync seems like magic.

```
R4#show ip cef 92.0.0.7
92.0.0.7/32
  nexthop 92.4.12.12 GigabitEthernet2.542 label 92009
```

```
R4#show ip ospf interface brief
```

Interface	PID	Area	IP Address/Mask	Cost	State	Nbrs	F/C
Lo0	92	0	92.0.0.4/32	1	LOOP	0/0	
Gi2.542	92	0	92.4.12.4/24	1	P2P	1/1	
Gi2.546	92	0	92.4.6.4/24	1	P2P	1/1	
Gi2.541	92	1	92.4.11.4/24	1	BDR	2/2	

The secret lies within the OSPF router LSA (LSA1). This is where IGP sync makes it changes; it doesn't actually change the configuration, but rather manipulates the SPF inputs by adjusting the LSA1. Looking at the details, we can see CSR4 has two transit links in area 0. One of them now has a cost of 65535, which is the link to CSR6. This is the result of IGP sync, and now the path through XRv2 is the shortest path.

```
R4#show ip ospf 92 0 database router self-originate | begin Number_of
Number of Links: 3
```

```
Link connected to: a Stub Network
```

```
(Link ID) Network/subnet number: 92.0.0.4
```

```
(Link Data) Network Mask: 255.255.255.255
```

```
Number of MTID metrics: 0
```

```
TOS 0 Metrics: 1
```

```
Link connected to: another Router (point-to-point)
```

```
(Link ID) Neighboring Router ID: 92.0.0.12
```

```
(Link Data) Router Interface address: 92.4.12.4
```

```
Number of MTID metrics: 0
```

```
TOS 0 Metrics: 1
```

```
Link connected to: another Router (point-to-point)
```

```
(Link ID) Neighboring Router ID: 92.0.0.6
```

```
(Link Data) Router Interface address: 92.4.6.4
```

```
Number of MTID metrics: 0
```

```
TOS 0 Metrics: 65535
```

If we check the IGP sync details, we can see that the peer remains reachable (IGP works) but synchronization with LDP has not been achieved on this link. This is an indication that this particular link is out of sync. However, traceroute reveals that the path to CSR7 is MPLS enabled, which means customer traffic (L3VPN, L2VPN, etc) can still flow properly.


```

R4#show mpls ldp igp sync interface gig2.546
    GigabitEthernet2.546:
        LDP configured; LDP-IGP Synchronization enabled.
        Sync status: sync not achieved; peer reachable.
        Sync delay time: 0 seconds (0 seconds left)
        IGP holddown time: infinite.
        IGP enabled: OSPF 92

R4#traceroute 92.0.0.7 source 92.0.0.4
Type escape sequence to abort.
Tracing the route to 92.0.0.7
VRF info: (vrf in name/id, vrf out name/id)
  1 92.4.12.12 [MPLS: Label 92009 Exp 0] 6 msec 5 msec 5 msec
  2 92.6.12.6 [MPLS: Label 6008 Exp 0] 15 msec 15 msec 15 msec
  3 92.6.7.7 19 msec 9 msec 11 msec

```

When we restore the LDP session on this link, IGP sync restores the original OSPF metric once the LDP session is complete. That is to say, once the labels have been exchanged and programmed to CSR4's LFIB, normal forwarding can continue and IGP sync will re-synchronize. Initially, when the session starts and no LDP updates have been sent, IGP sync ignores this because it doesn't qualify as being fully up.

```

! CSR4
LDP-SYNC: Gi2.546: No session or session has not send initial update, ignore
adj joining event.
%LDP-5-NBRCHG: LDP Neighbor 92.0.0.6:0 (3) is UP

```

Very shortly thereafter, LDP begins the label exchange which means IGP sync honors the adjacency change and deactivates from this link.

```

! CSR4
LDP-SYNC: Gi2.546: session 92.0.0.6:0 came up, sync achieved up
LDP-SYNC: Gi2.546, OSPF 92: notify status (required, achieved, no delay,
holddown infinite) internal status (achieved, timer not running)

```

A quick check of the LSA1 shows the restoration of the original OSPF cost, and checking the FIB shows that the labeled traffic is forward through CSR6 again.

```

R4#show ip ospf 92 0 database router self-originate | begin Number of
Number of Links: 3

Link connected to: a Stub Network
(Link ID) Network/subnet number: 92.0.0.4
(Link Data) Network Mask: 255.255.255.255
Number of MTID metrics: 0
TOS 0 Metrics: 1

Link connected to: another Router (point-to-point)

```

```
(Link ID) Neighboring Router ID: 92.0.0.12
(Link Data) Router Interface address: 92.4.12.4
Number of MTID metrics: 0
TOS 0 Metrics: 1
```

```
Link connected to: another Router (point-to-point)
(Link ID) Neighboring Router ID: 92.0.0.6
(Link Data) Router Interface address: 92.4.6.4
Number of MTID metrics: 0
TOS 0 Metrics: 1
```

```
R4#show ip cef 92.0.0.7
92.0.0.7/32
  nexthop 92.4.6.6 GigabitEthernet2.546 label 6008
```

Next, we will test the feature with IS-IS on XR. We will configure XRv2 with an ACL that blocks UDP/TCP 646 on the interface to CSR9. This will break the session, but both routers will still have an IGP adjacency (and thus an IP route for one another's transport addresses).

```
! XRv2
ipv4 access-list ACL_DENY_LDP
 10 deny udp any any eq ldp
 20 deny tcp any any eq ldp
 30 permit ipv4 any any

interface GigabitEthernet0/0/0/0.592
  ipv4 access-group ACL_DENY_LDP ingress
```

To speed things up, we will clear the LDP session manually only XRv2. We also enable IGP sync debugging so we can see the failures occur. IGP sync moves the adjacency out of the synchronized state and notifies IS-IS of the issue. We can see that this interface is not synchronized.

```
! XRv2
debug mpls ldp igp sync

mpls_ldp[1048]: DBG-ISync[1], Intf GigabitEthernet0_0_0_0.592: Adj 92.9.12.9
being deleted, sync achieved goes down
mpls_ldp[1048]: DBG-ISync[1], ldp_isync_announce_status: Intf
GigabitEthernet0_0_0_0.592 (ifh 0x900); notify 1, (sync 0, nsf 0) -> (sync 0,
nsf 0)

RP/0/0/CPU0:XRv2#show mpls ldp igp sync interface gig0/0/0/0.592
GigabitEthernet0/0/0/0.592:
  VRF: 'default' (0x60000000)
  Sync delay: Disabled
  Sync status: Not ready (No hello adjacency)
```

Notice that IGP sync causes the IS-IS cost on the interface to CSR9 to be increased to the maximum metric within the IS-IS LSP. This is the equivalent behavior in IS-IS as we saw in OSPF. Because there are no alternative paths, XRv2 still installs this route in the RIB, but it doesn't matter since there is no LDP reachability on the link due to the traffic filter. If there were alternative paths, they would have been selected when IS-IS ran SPF. The link isn't invalidated, it just isn't preferred.

```
RP/0/0/CPU0:XRv2#show isis database level 1 XRv2.00-00 detail
IS-IS LDP (Level-1) Link State Database
LSPID                LSP Seq Num  LSP Checksum  LSP Holdtime  ATT/P/OL
XRv2.00-00           * 0x00000071  0x89be        987           0/0/0
  Area Address: 00
  NLPID:         0xcc
  NLPID:         0x8e
  MT:            Standard (IPv4 Unicast)
  MT:            IPv6 Unicast                               0/0/0
  Hostname:      XRv2
  IP Address:    92.0.0.12
  Metric: 16777214  IS-Extended R9.00
  Metric: 0        IP-Extended 92.0.0.4/32
  Metric: 0        IP-Extended 92.0.0.5/32
  Metric: 0        IP-Extended 92.0.0.6/32
[snip]
```

```
RP/0/0/CPU0:XRv2#show route ipv4 92.0.0.9
Routing entry for 92.0.0.9/32
  Known via "isis LDP", distance 115, metric 16777214, type level-1
  Routing Descriptor Blocks
    92.9.12.9, from 92.0.0.9, via GigabitEthernet0/0/0/0.592
    Route metric is 16777214
  No advertising protos.
```

Something similar happens on CSR9. IGP sync sees there is a fault and increases the IS-IS cost to XRv2 to the max-metric. However, CSR9 has an alternate route via CSR2 (CSR6 is the ASBR) which is a more realistic use for IGP sync.

```
R9#show mpls ldp igp sync interface gig2.592
GigabitEthernet2.592:
  LDP configured; LDP-IGP Synchronization enabled.
  Sync status: sync not achieved; peer reachable.
  Sync delay time: 0 seconds (0 seconds left)
  IGP holddown time: infinite.
  IGP enabled: ISIS LDP

R9#show isis database level-1 R9.00-00 detail
Tag LDP:
IS-IS Level-1 LSP R9.00-00
LSPID                LSP Seq Num  LSP Checksum  LSP Holdtime  ATT/P/OL
```

```

R9.00-00          * 0x0000006A    0x2175          708          0/0/0
Area Address: 00
NLPID:           0xCC 0x8E
Topology:         IPv4 (0x0)
                  IPv6 (0x2)
Hostname: R9
Metric: 10        IS-Extended R8.00
Metric: 16777214  IS-Extended XRv2.00
Metric: 10        IS-Extended R3.01
[snip]

```

```

R9#show ip route 92.0.0.12
Routing entry for 92.0.0.12/32
  Known via "isis", distance 115, metric 20, type level-1
  Redistributing via isis LDP
  Last update from 92.2.3.2 on GigabitEthernet2.523, 00:06:51 ago
  Routing Descriptor Blocks:
    * 92.2.3.2, from 92.0.0.6, 00:06:51 ago, via GigabitEthernet2.523
      Route metric is 20, traffic share count is 1

```

We can adjust the IGP sync hold-down time as well. This is a global setting on XE that tells LDP how long to wait for synchronization, and there does not appear to be an XR equivalent. By default, it will wait forever, but my experience shows me that that is always the case. I've never personally seen this value do anything, but the configuration is applied to CSR9. I also cannot think of a case where you would want this to be anything other than infinity, since failing to synchronize a link quickly doesn't mean IGP should suddenly start using it. I set the timer to 30000ms (30ms) on CSR9, then verify any IGP sync interface to show the change.

```

! CSR9
mpls ldp igp sync holddown 30000

R9#show mpls ldp igp sync int gig2.589
GigabitEthernet2.589:
  LDP configured; LDP-IGP Synchronization enabled.
  Sync status: sync achieved; peer reachable.
  Sync delay time: 0 seconds (0 seconds left)
  IGP holddown time: 30000 milliseconds.
  Peer LDP Ident: 92.0.0.8:0
  IGP enabled: ISIS LDP

```

There is also a sync delay parameter which is supported on XE and XR. This tells LDP how long to wait before declaring a link synchronized after the link comes back (and subsequently restoring the cost to the normal value). By default, this is 0 seconds, which means that as soon as a link comes back online, synchronization is immediately announced. This is generally not desired and could introduce churn into the IGP process. Continuing to use the backup path is better than rushing to a new link that might flap

again 5 seconds later. I set it to 45 seconds on both CSR9 and XRv2, and then verify it quickly on both sides.

```
! CSR9
interface GigabitEthernet2.592
  mpls ldp igp sync delay 45

mpls ldp
  interface GigabitEthernet0/0/0/0.592
    igp sync delay on-session-up 45

R9#show mpls ldp igp sync interface gig2.592
  GigabitEthernet2.592:
    LDP configured; LDP-IGP Synchronization enabled.
    Sync status: sync achieved; peer reachable.
    Sync delay time: 45 seconds (0 seconds left)
    IGP holddown time: 30000 milliseconds.
    Peer LDP Ident: 92.0.0.12:0
    IGP enabled: ISIS LDP

RP/0/0/CPU0:XRv2#show mpls ldp igp sync interface gig0/0/0/0.592
GigabitEthernet0/0/0/0.592:
  VRF: 'default' (0x60000000)
  Sync delay: 45 sec
  Sync status: Ready
  Peers:
    92.0.0.9:0
```

If we reapply the ACL on XRv2 to break the LDP session then remove it after the session fails (basically, flap the LDP session), we will see that the IGP sync process will wait 45 seconds on both sides before declaring the link synchronized after the link comes back up. This is the first set of log messages we see.

```
! CSR9
%LDP-5-NBRCHG: LDP Neighbor 92.0.0.12:0 (4) is UP
LDP-SYNC: Gi2.592: session 92.0.0.12:0 came up, sync_achieved up
LDP-SYNC: Gi2.592: Delay notifying IGP of sync achieved for 45 seconds

! XRv2
%ROUTING-LDP-5-NBR_CHANGE : VRF 'default' (0x60000000), Neighbor 92.0.0.9:0
is UP (IPv4 connection)
mpls_ldp[1048]: DBG-ISync[1], Intf GigabitEthernet0_0_0_0.592:
ldp_isync_up_adj_core delay_sync 1 delay_cfgd 1 gr_enabled 0 gr_recon 0
event 0x1 isync_flag 0
mpls_ldp[1048]: DBG-ISync[1], Intf 'GigabitEthernet0_0_0_0.592': Tmr started:
'IGP-Sync Intf Delay' (45s,0ms)
```

During this 45 minute period, we quickly check the IGP sync status on both routers. Both of them are counting down from 45 to 0, and until then, the IS-IS link is still carrying the max-metric. XE says synchronization is achieved, which is technically true, but the countdown is the hint that max-metric is still advertised. XR has better output and uses the word “deferred” to explicitly suggest that full synchronization waiting for the delay timer to expire.

```
R9#show mpls ldp igp sync interface gig2.592
GigabitEthernet2.592:
  LDP configured; LDP-IGP Synchronization enabled.
  Sync status: sync achieved; peer reachable.
  Sync delay time: 45 seconds (29 seconds left)
  IGP holddown time: 30000 milliseconds.
  Peer LDP Ident: 92.0.0.12:0
  IGP enabled: ISIS LDP

RP/0/0/CPU0:XRv2#show mpls ldp igp sync interface gig0/0/0/0.592
GigabitEthernet0/0/0/0.592:
  VRF: 'default' (0x60000000)
  Sync delay: 45 sec
  Sync status: Deferred (35 sec remaining)
```

The next set of debugs, occurring 45 seconds after the first batch, indicates that full synchronization has been announced to the IGP. Using the sync delay can protect against consistently unstable links.

```
! CSR9
LDP-SYNC: Gi2.592: Delay timer expired, notify IGP of sync achieved
LDP-SYNC: Gi2.592, ISIS LDP: notify status (required, achieved, no delay,
holddown 30000) internal status (achieved, timer not running)

! XRv2
DBG-ISync[1], Intf 'GigabitEthernet0_0_0_0.592': IGP Sync up (delay tmr
expired)
mpls_ldp[1048]: DBG-ISync[1], ldp_isync_announce_status: Intf
GigabitEthernet0_0_0_0.592 (ifh 0x900); notify 1, (sync 1, nsf 0) -> (sync 1,
nsf 0)
```

Next, we will quickly examine what happens when we have the opposite problem. That is, IS-IS fails to form on a link but LDP forms just fine. We can break IS-IS in many ways, but the simplest would be a network type mismatch. This problem is much more rare and less significant, but it is worth examining.

```
! CSR9
interface GigabitEthernet2.592
  no isis network point-to-point
```

Interestingly, this does not cause any problems nor does it trigger IGP sync at all. Since IGP will obviously converge around it, there is no possibility of blackholing traffic over this link, so IGP sync can remain

blind to this condition. CSR9 has chosen a valid alternative path for all of its IGP prefixes. Eventually, the LDP session will time out and IGP sync will be lost, but at this point in time, the network has already converged so IGP sync doesn't need to act quickly, even though sync isn't achieved.

```
R9#show isis neighbors
```

```
Tag LDP:
```

System Id	Type	Interface	IP Address	State	Holdtime	Circuit Id
R2	L1	Gi2.523	92.2.3.2	UP	24	R3.01
R3	L1	Gi2.523	92.2.3.3	UP	6	R3.01
R8	L1	Gi2.589	92.8.9.8	UP	23	01

```
R9#show mpls ldp igp sync interface gig2.592
```

```
GigabitEthernet2.592:
```

```
LDP configured; LDP-IGP Synchronization enabled.
```

```
Sync status: sync achieved; peer reachable.
```

```
Sync delay time: 0 seconds (0 seconds left)
```

```
IGP holddown time: 30000 milliseconds.
```

```
Peer LDP Ident: 92.0.0.12:0
```

```
IGP enabled: ISIS LDP
```

Before continuing, all broken network types, ACLs, and null routes from previous tests are removed so the network is stable. Next, we will look at label allocation and filtering/advertising. Label allocation is the process of assigning local labels to prefixes. We can control this by applying prefix-lists (XE) or ACLs (XR) to the LDP process to determine for which prefixes we allocate labels. This can greatly reduce LIB size if, for example, labels are only allocated for host-routes. Bear in mind that local labels must be allocated for any remote prefix that can be an LSP endpoint. First, we show output on CSR9 with the command disabled, and LIB entries are created for all transit links. This is generally worthless unless there are hosts on those LAN segments that require their traffic to be MPLS-encapsulated as it transits the network.

```
R9#show mpls ldp bindings 92.8.9.0 24
```

```
lib entry: 92.8.9.0/24, rev 40
```

```
local binding: label: imp-null
```

```
R9#show mpls ldp bindings 92.9.12.0 24
```

```
lib entry: 92.9.12.0/24, rev 42
```

```
local binding: label: imp-null
```

Like setting the RID, this is a technique I almost always configure by default. There is even a keyword for host-routes that makes it very easy. I enable this on all XE and XR routers to start.

```
! All XE routers
```

```
mpls ldp label
```

```
allocate global host-routes
```

```
! All XR routers
```

```
mpls ldp
 address-family ipv4
  label
  local
  allocate for host-routes
```

After enabling this command, a quick check on CSR9 shows a LIB that contains only host-route entries. There are no local bindings for these prefixes anymore.

```
R9#show mpls ldp bindings 92.8.9.0 24
 lib entry: 92.8.9.0/24, rev 45
    no local binding
```

```
R9#show mpls ldp bindings 92.9.12.0 24
 lib entry: 92.9.12.0/24, rev 46
    no local binding
```

We can also apply list-based filters rather than use the “host-routes” keyword. For simplicity, I configure prefix and access-lists that accomplish the same thing except explicitly. Personally, I like the prefix-list more because it lets me match the mask value, whereas the ACL in XR does not. Technically a prefix like 92.0.0.0/30 would get a label on XRv2, but not CSR7. This is just a CLI limitation on XR.

```
! CSR7
ip prefix-list PL_LOOPBACKS seq 5 permit 92.0.0.0/24 ge 32

mpls ldp label
 allocate global prefix-list PL_LOOPBACKS

! XRv2
ipv4 access-list ACL_92_NET
 10 permit ipv4 92.0.0.0 0.0.0.255 any

mpls ldp
 address-family ipv4
  label
  local
  allocate for ACL_92_NET
```

To verify these configurations, we will select a transit link for which CSR7 and XRv2 have IGP routes and expect to see no corresponding labels in the LIB.

```
R7#show mpls ldp bindings 92.4.12.0 24
 lib entry: 92.4.12.0/24, rev 61
    no local binding
```

```
RP/0/0/CPU0:XRv2#show mpls ldp bindings 92.4.12.0/24
[no output]
```


Once labels are allocated, they can be selectively advertised to neighbors, as well as selectively accepted on ingress from other neighbors. This might be used to reduce overall LIB size or force traffic to some destinations to never be MPLS-encapsulated. Intelligent label filtering is usually focused on reducing LIB size. For example, every LDP router will allocate and advertise a label for every IGP route it learns. The labels are advertised to all peers and retained in their LIBs. In our topology, there is never a case where XRv4 would need to learn any label from CSR1 except for CSR1's local prefixes. The same is true for the relationship between CSR5 and XRv3. However, XRv4 and CSR5 still need to advertise all of the IGP routes with their corresponding labels to CSR1 and XRv3, respectively, so the filters are not always obvious to visualize. Outbound filtering is simpler on XR than XE, so we start there first. XRv4 will advertise labels for CSR8 and XRv3 loopbacks to CSR1. This means that CSR1 will only be able to push labels if LSP goes to CSR8 or XRv3.

```
! XRv4
ipv4 access-list ACL_PERMIT_LOOPBACKS
 10 permit ipv4 host 92.0.0.8 any
 20 permit ipv4 host 92.0.0.13 any

mpls ldp
 address-family ipv4
  label
  local
  advertise
    to 92.0.0.1:0 for ACL_PERMIT_LOOPBACKS
```

When we check CSR1's LIB, we only see label bindings for the specified prefixes. Quickly checking the FIB, we can see that CSR8's loopback has the label bound properly, but CSR9's loopback is untagged. This can be used as a memory saving technique to keep the LIB as small as possible.

```
R1#show mpls ldp bindings neighbor 92.0.0.14
 lib entry: 92.0.0.8/32, rev 127
   remote binding: lsr: 92.0.0.14:0, label: 94007
 lib entry: 92.0.0.13/32, rev 132
   remote binding: lsr: 92.0.0.14:0, label: 94008

R1#show ip cef 92.0.0.8
92.0.0.8/32
  nexthop 92.1.14.14 GigabitEthernet2.514 label 94007

R1#show ip cef 92.0.0.9
92.0.0.9/32
  nexthop 92.1.14.14 GigabitEthernet2.514
```

We can also configure inbound label filtering on XRv4. For example, there is never a case where XRv4 would need to learn a label for 92.0.0.1/32 from XRv3 or CSR10. This is because those routers will never

have an alternate path to CSR1, so retaining the labels is worthless. First, we check the LIB and see that XRv4 has labels from all three of its LDP neighbors when it really only needs the implicit-null label from CSR1.

```
RP/0/0/CPU0:XRv4#show mpls ldp bindings 92.0.0.1/32
92.0.0.1/32, rev 16
  Local binding: label: 94003
  Remote bindings: (3 peers)
    Peer                Label
    -----
    92.0.0.1:0          ImpNull
    92.0.0.3:0          3003
    92.0.0.10:0         10012
```

We apply the configuration below so that XRv4 will not learn these labels for 92.0.0.1/32 from CSR3 and CSR10. This makes sense because there is never a case when XRv4 would route traffic to 92.0.0.1/32 via those LSRs. In short, the rule for inbound filtering on XR is configured under the “label remote accept” stanza, while outbound filtering is configured under the “label local advertise” stanza.

```
! XRv4
ipv4 access-list ACL_R1_LOOPBACK
 10 deny ipv4 host 92.0.0.1 any
 20 permit ipv4 any any

mpls ldp
 address-family ipv4
  label
   remote
    accept
     from 92.0.0.3:0 for ACL_R1_LOOPBACK
     from 92.0.0.10:0 for ACL_R1_LOOPBACK
```

After applying this configuration, XRv4 rejects the labels from the specified peers. It only have one label in the LIB for this prefix which was learned from CSR1.

```
RP/0/0/CPU0:XRv4#show mpls ldp bindings 92.0.0.1/32
92.0.0.1/32, rev 16
  Local binding: label: 94003
  Remote bindings: (1 peers)
    Peer                Label
    -----
    92.0.0.1:0          ImpNull
```

The configuration in XE is less straightforward. We will configure CSR5 to only advertise labels for 92.0.0.8/32 and 92.0.0.1/32 towards XRv3. The configuration appears simple at first.

```

! CSR5
ip access-list standard ACL_LOOPBACKS
  permit 92.0.0.1
  permit 92.0.0.8

ip access-list standard ACL_XRV3
  permit 92.0.0.13

mpls ldp advertise-labels for ACL_LOOPBACKS to ACL_XRV3

```

However, XRv3 still has all of the labels for all loopbacks from CSR5 despite configuring advertisement of only specific prefixes towards XRv3 on CSR5.

```

RP/0/0/CPU0:XRv3#show mpls ldp bindings neighbor 92.0.0.5:0
92.0.0.1/32, rev 77
  Local binding: label: 93008
  Remote bindings: (1 peers)
    Peer          Label
    -----
    92.0.0.5:0    5008
92.0.0.2/32, rev 43
  Local binding: label: 93013
  Remote bindings: (1 peers)
    Peer          Label
    -----
    92.0.0.5:0    5010
[snip]

```

The issue is that XE assumes that you want to advertise all labels with all prefixes to all peers until you explicitly tell it to stop. This means that you have to explicitly advertise labels to other peers as well. To disable label advertisement in general, we use the command below. Now, XRv3 only has the two labels to which it is entitled, and no more.

```

! CSR5
no mpls ldp advertise-labels

RP/0/0/CPU0:XRv3#show mpls ldp bindings neighbor 92.0.0.5:0
92.0.0.1/32, rev 77
  Local binding: label: 93008
  Remote bindings: (1 peers)
    Peer          Label
    -----
    92.0.0.5:0    5008
92.0.0.8/32, rev 44
  Local binding: label: 93014
  Remote bindings: (1 peers)
    Peer          Label
    -----

```

92.0.0.5:0 5015

This introduces a new problem. Since CSR5 totally stopped advertising labels (except for the few to XRv3), XRv1 and CSR4 now learn no labels from CSR5. This obviously breaks MPLS transport to most of the nodes in the network.

```
RP/0/0/CPU0:XRv1#show mpls ldp bindings neighbor 92.0.0.5:0
[no output]
```

```
R4#show mpls ldp bindings neighbor 92.0.0.5
[no output]
```

We can correct this by instructing CSR5 to advertise labels for all prefixes to all neighbors that are not XRv3. Personally, I feel like this is overkill and difficult to maintain. That is probably why the XR configuration is much simpler as the CLI syntax is newer (XE behavior is carried over from classic IOS).

```
! CSR5
ip access-list standard ACL_ANY
 permit any
ip access-list standard ACL_NOT_XRV3
 deny 92.0.0.13
 permit any

mpls ldp advertise-labels for ACL_ANY to ACL_NOT_XRV3
```

Now, we confirm XRv3 only has the 2 labels it's supposed to, while XRv1 and CSR4 have them all.

```
RP/0/0/CPU0:XRv3#show mpls ldp bindings neighbor 92.0.0.5:0
92.0.0.1/32, rev 77
  Local binding: label: 93008
  Remote bindings: (1 peers)
    Peer          Label
    -----
    92.0.0.5:0    5008
92.0.0.8/32, rev 44
  Local binding: label: 93014
  Remote bindings: (1 peers)
    Peer          Label
    -----
    92.0.0.5:0    5015
```

```
RP/0/0/CPU0:XRv1#show mpls ldp bindings neighbor 92.0.0.5:0
92.0.0.1/32, rev 77
  Local binding: label: 91011
  Remote bindings: (4 peers)
    Peer          Label
    -----
```

```

          92.0.0.5:0          5008
92.0.0.2/32, rev 23
  Local binding: label: 91007
  Remote bindings: (4 peers)
      Peer          Label
      -----
92.0.0.5:0          5010
[snip]

R4#show mpls ldp bindings neighbor 92.0.0.5
  lib entry: 92.0.0.1/32, rev 83
    remote binding: lsr: 92.0.0.5:0, label: 5008
  lib entry: 92.0.0.2/32, rev 28
    remote binding: lsr: 92.0.0.5:0, label: 5010
  lib entry: 92.0.0.3/32, rev 49
    remote binding: lsr: 92.0.0.5:0, label: 5014
[snip]

```

There is a special command we can use on XE to see which advertisement ACLs apply to which prefixes. I show two examples below. First, the output shows all configured prefix and peer ACLs as reference, and then iterates over matching LIB entries. For 92.0.0.7/32, the prefix ACL is the one that matches all prefixes, and the peer ACL matches everything except XRv3. This “match” means that the label for this prefix can be advertised according to those rules. The LIB entry for 92.0.0.8/32 is also shown. This matches a more specific ACL containing CSR8 and CSR1 loopbacks only, and is specific to XRv3.

```

R5#show mpls ldp bindings 92.0.0.7 32 advertisement-acls
Advertisement spec:
  Prefix acl = ACL_LOOPBACKS; Peer acl = ACL_XRV3
  Prefix acl = ACL_ANY; Peer acl = ACL_NOT_XRV3

  lib entry: 92.0.0.7/32, rev 166
    Advert acl(s): Prefix acl ACL_ANY; Peer acl ACL_NOT_XRV3

R5#show mpls ldp bindings 92.0.0.8 32 advertisement-acls
Advertisement spec:
  Prefix acl = ACL_LOOPBACKS; Peer acl = ACL_XRV3
  Prefix acl = ACL_ANY; Peer acl = ACL_NOT_XRV3

  lib entry: 92.0.0.8/32, rev 126
    Advert acl(s): Prefix acl ACL_LOOPBACKS; Peer acl ACL_XRV3

```

The command also exists on XR but is less valuable. Here is the sample output from XRv4. It just lists prefixes with no valuable information provided regarding advertisement ACLs.

```

RP/0/0/CPU0:XRv4#show mpls ldp bindings advertisement-acls
Advertisement Spec: None

```

Local Label Allocation Spec:
Host routes only

92.0.0.1/32, rev 16
92.0.0.2/32, rev 25
92.0.0.3/32, rev 17
[snip]

XE inbound filtering is more straightforward. CSR5 receives labels for all IGP routes learned by XRv3, which is all of the loopbacks. CSR5 has no reason to learn any label from XRv3 except for XRv3's loopback (some kind of null label). We can filter these other labels from the LIB. Below is a "before" snapshot of CSR5's LIB entries learned from XRv3.

```
R5#show mpls ldp bindings neighbor 92.0.0.13
lib entry: 92.0.0.1/32, rev 160
remote binding: lsr: 92.0.0.13:0, label: 93008
lib entry: 92.0.0.2/32, rev 161
remote binding: lsr: 92.0.0.13:0, label: 93013
lib entry: 92.0.0.3/32, rev 162
remote binding: lsr: 92.0.0.13:0, label: 93015
[snip]
```

The configuration on CSR5 is below. We can be efficient and re-use an ACL from earlier that simply matches 92.0.0.13. Since the neighbor is specified in the command, we only have one ACL as input. Now, CSR5 only learns one label from XRv3, which is for its loopback prefix using implicit-null.

```
! CSR5
mpls ldp neighbor 92.0.0.13 labels accept ACL_XRV3

R5#show mpls ldp bindings neighbor 92.0.0.13
lib entry: 92.0.0.13/32, rev 131
remote binding: lsr: 92.0.0.13:0, label: imp-null
```

The idea behind these label filters is to sustain the LSPs between CSR1, CSR8, and XRv3. As such, these routers should be able to send MPLS-encapsulated traffic to one another with no breaks in the LSP. We will test a few paths to ensure that our label filters did not break these paths. Since no label filtering was done towards CSR8, we won't verify LSPs sourced from CSR8 since it is highly unlikely they are broken. Below, we can see all 4 traceroutes are fully MPLS-encapsulated.

```
R1#traceroute 92.0.0.8 source 92.0.0.1
Type escape sequence to abort.
Tracing the route to 92.0.0.8
VRF info: (vrf in name/id, vrf out name/id)
 1 92.1.14.14 [MPLS: Label 94007 Exp 0] 7 msec 6 msec 6 msec
 2 92.3.14.3 [MPLS: Label 3011 Exp 0] 28 msec 30 msec 30 msec
 3 92.2.3.2 [MPLS: Label 2002 Exp 0] 21 msec 19 msec 20 msec
```

```
4 92.2.8.8 21 msec 11 msec 10 msec
```

```
R1#traceroute 92.0.0.13 source 92.0.0.1
```

```
Type escape sequence to abort.
```

```
Tracing the route to 92.0.0.13
```

```
VRF info: (vrf in name/id, vrf out name/id)
```

```
1 92.1.14.14 [MPLS: Label 94008 Exp 0] 12 msec 10 msec 8 msec
2 92.3.14.3 [MPLS: Label 3005 Exp 0] 21 msec 31 msec 30 msec
3 92.2.3.9 [MPLS: Label 9006 Exp 0] 30 msec 29 msec 30 msec
4 92.9.12.12 [MPLS: Label 92012 Exp 0] 31 msec 31 msec 28 msec
5 92.4.12.4 [MPLS: Label 4005 Exp 0] 31 msec 32 msec 29 msec
6 92.4.11.5 [MPLS: Label 5005 Exp 0] 14 msec 15 msec 20 msec
7 92.5.13.13 19 msec 21 msec 15 msec
```

```
RP/0/0/CPU0:XRv3#traceroute 92.0.0.1 source 92.0.0.13
```

```
Type escape sequence to abort.
```

```
Tracing the route to 92.0.0.1
```

```
1 92.5.13.5 [MPLS: Label 5008 Exp 0] 9 msec 9 msec 9 msec
2 92.4.11.4 [MPLS: Label 4015 Exp 0] 9 msec 0 msec 0 msec
3 92.4.12.12 [MPLS: Label 92005 Exp 0] 0 msec 0 msec 0 msec
4 92.9.12.9 [MPLS: Label 9011 Exp 0] 9 msec 0 msec 0 msec
5 92.2.3.3 [MPLS: Label 3003 Exp 0] 9 msec 9 msec 0 msec
6 92.3.14.14 [MPLS: Label 94003 Exp 0] 0 msec 59 msec 9 msec
7 92.1.14.1 19 msec 9 msec 79 msec
```

```
RP/0/0/CPU0:XRv3#traceroute 92.0.0.8 source 92.0.0.13
```

```
Type escape sequence to abort.
```

```
Tracing the route to 92.0.0.8
```

```
1 92.5.13.5 [MPLS: Label 5015 Exp 0] 9 msec 0 msec 0 msec
2 92.4.11.4 [MPLS: Label 4009 Exp 0] 0 msec 0 msec 0 msec
3 92.4.6.6 [MPLS: Label 6003 Exp 0] 0 msec 0 msec 0 msec
4 92.2.6.2 [MPLS: Label 2002 Exp 0] 0 msec 0 msec 0 msec
5 92.2.8.8 0 msec 0 msec 0 msec
```

However, if we traceroute to some other destination for which CSR1 and XRv3 do not have labels, the LSP will be incomplete at the first hop (at a minimum). This would break all MPLS services, such as L3VPN and L2VPN.

```
R1#traceroute 92.0.0.7 source 92.0.0.1
```

```
Type escape sequence to abort.
```

```
Tracing the route to 92.0.0.7
```

```
VRF info: (vrf in name/id, vrf out name/id)
```

```
1 92.1.14.14 5 msec 1 msec 2 msec
2 92.3.14.3 [MPLS: Label 3009 Exp 0] 9 msec 8 msec 7 msec
3 92.2.3.2 [MPLS: Label 2010 Exp 0] 25 msec 30 msec 30 msec
4 92.2.6.6 [MPLS: Label 6008 Exp 0] 15 msec 15 msec 16 msec
```

```
5 92.6.7.7 21 msec 12 msec 11 msec
```

```
RP/0/0/CPU0:XRv3#traceroute 92.0.0.9 source 92.0.0.13
```

```
Type escape sequence to abort.
```

```
Tracing the route to 92.0.0.9
```

```
1 92.5.13.5 0 msec 0 msec 0 msec
2 92.4.11.11 [MPLS: Label 91006 Exp 0] 0 msec 0 msec 0 msec
3 92.6.11.6 [MPLS: Label 6001 Exp 0] 0 msec 0 msec 0 msec
4 92.2.6.2 [MPLS: Label 2003 Exp 0] 0 msec 0 msec 0 msec
5 92.2.3.9 109 msec 0 msec 0 msec
```

The last LDP feature we examine is LDP implicit withdraw. In newer IOS releases, when LDP needs to change a label binding for a particular prefix, that update is preceded by a label withdraw message. This is an explicit message which informs the peer that the label you had before is no longer valid, and then the new one is advertised. In older releases, this process was implicit and there was no explicit label withdraw. This command is enabled on a per neighbor basis to enable the old-style behavior. This does not appear to be supported in XR at all. We can enable LDP message reception debugging on XRv3 to see this.

```
RP/0/0/CPU0:XRv3#debug mpls ldp messages received
```

On CSR5, we will further constrain the label advertisement by removing the ability to advertise labels for 92.0.0.1/32. This will trigger an explicit label withdraw message from CSR5 to XRv3. The message contents are unreadable but we clearly see the message being received.

```
R5(config)#ip access-l standard ACL_LOOPBACKS
```

```
R5(config-std-nacl)#no 20 permit 92.0.0.1
```

```
! XRv3
```

```
mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0): Rcvd 'LABEL-  
WITHDRAW' msg (size 24, seq 7); 'Prefix' FEC
```

```
mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0): TLVs: (2)
```

```
mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0): #1:
```

```
idx=18, type=0x100, U/F=0/0
```

```
mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0): #2:
```

```
idx=30, type=0x200, U/F=0/0
```

When we add the ACL entry back for 92.0.0.1/32 (not shown), CSR5 sends a label mapping message, as discussed earlier, which advertises a particular label for the prefix.

```
! XRv3
```

```
mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0): Rcvd 'LABEL-  
MAPPING' msg (size 24, seq 11); 'Prefix' FEC
```

```
mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0): TLVs: (2)
```



```
mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0):      #1:
idx=18, type=0x100, U/F=0/0
mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0):      #2:
idx=30, type=0x200, U/F=0/0
```

Removing entries from an ACL is always going to trigger a withdraw since there is no new label being advertised. Instead, we will configure a static local label on CSR5. This new value is guaranteed to be different than the existing value. Static labels are covered in detail in the next section, but in this case, we just configure the in-label (local label) on CSR5. We also must define a static label range first as well.

```
! CSR5
mpls label range 5000 5999 static 500 599
mpls static binding ipv4 92.0.0.1 255.255.255.255 501
```

The result is that XRv3 receives a label withdraw immediately followed by a label mapping message. This is expected since implicit-withdraw is not enabled by default. This is CSR5 withdrawing the old LDP-allocated label and advertising the statically configured one in an LDP mapping message.

```
! XRv3
mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0): Rcvd 'LABEL-
WITHDRAW' msg (size 24, seq 11); 'Prefix' FEC
mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0):      TLVs: (2)
mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0):      #1:
idx=18, type=0x100, U/F=0/0
mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0):      #2:
idx=30, type=0x200, U/F=0/0
mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0): Rcvd 'LABEL-
MAPPING' msg (size 24, seq 12); 'Prefix' FEC
mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0):      TLVs: (2)
mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0):      #1:
idx=18, type=0x100, U/F=0/0
mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0):      #2:
idx=30, type=0x200, U/F=0/0
```

Now, we will enable implicit withdraw on CSR5 and change the static label once again. This time, CSR5 only sends a label mapping, which overwrites the existing LIB entry. This new label mapping message serves as an implicit withdraw to save LDP overhead. Because the label was overwritten, the need for an explicit withdrawal is eliminated.

```
! CSR5
mpls ldp neighbor 92.0.0.13 implicit-withdraw
mpls static binding ipv4 92.0.0.1 255.255.255.255 502

! XRv3
mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0): Rcvd 'LABEL-
MAPPING' msg (size 24, seq 17); 'Prefix' FEC
```

```

mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0): TLVs: (2)
mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0): #1:
idx=18, type=0x100, U/F=0/0
mpls_ldp[1048]: DBG-MsgRcv[1], VRF(default): Peer(92.0.0.5:0): #2:
idx=30, type=0x200, U/F=0/0

```

The next section contains the configurations for this section since it builds on this lab.

7.2 Static label bindings

This section is a continuation from the LDP lab above. Despite all of the advanced LDP options we configured, we can also manually build LSPs. This is similar to MPLS transport profile (MPLS-TP) where the LSPs must be provisioned statically on all LSRs along the LSP. However, the static LSPs in XE and XR are still somewhat reliant on LDP being enabled, which is unlike MPLS-TP. The static label feature allows us to custom-define local labels, as well as label swapping for forwarding. This can interwork with LDP labels so the path can be static for a few hops, then dynamic via LDP for the remaining hops. We will examine building a static LSP from CSR1 to CSR3, transiting XRv4. The idea is to sustain the MPLS connectivity between CSR1, CSR8, and XRv3 which was the theme in the last lab. This is configuration intensive but not difficult. Beginning with CSR1, we need to allocate local labels for both 92.0.0.8 and 92.0.0.13. We also need to identify what are the remote labels for this prefix and towards which next-hop. These out-labels on CSR1 must be statically defined as local labels on XRv4, which mimics the behavior of LDP where this happens automatically. As a final measure to ensure LDP bindings are not advertised, we totally disable label advertisement on CSR1.

```

! CSR1
no mpls ldp advertise-labels
mpls static binding ipv4 92.0.0.8 255.255.255.255 103
mpls static binding ipv4 92.0.0.8 255.255.255.255 output 92.1.14.14 9408
mpls static binding ipv4 92.0.0.13 255.255.255.255 113
mpls static binding ipv4 92.0.0.13 255.255.255.255 output 92.1.14.14 9413

```

After adding this configuration, the first issue we see indicates a label conflict. Because we have learned dynamic labels from XRv4, those are preferred over the statically defined out-labels. We can correct this one of two ways: filter labels inbound on CSR1, or filter them outbound on XRv4.

```

R1(config)# mpls static binding ipv4 92.0.0.8 255.255.255.255 output
92.1.14.14 9408
% Next hop 92.1.14.14 is an LDP peer (92.0.0.14:0)
% Label learned from peer, if any, takes precedence
% Continuing with configuration of the label

```

For simplicity, I will filter all label advertisements on XRv4 temporarily. Additionally, I will add in the appropriate static label bindings for the prefixes we are testing. Notice that the local labels 9413 and 9408 are the same out-labels we configured on CSR1. XRv4 will swap these labels for new labels that are local on CSR3, which we will also configure. To reach XRv4, since it is the penultimate hop, we configure

it to pop the topmost label when traffic arrives with label 9401. This is part of the reverse LSP but we configure it now for completeness.

```
! XRv4
mpls ldp
  address-family ipv4
    label
      local
        advertise
        disable

mpls static
  address-family ipv4 unicast
    local-label 9401 allocate per-prefix 92.0.0.1/32
    forward
      path 1 nexthop GigabitEthernet0/0/0/0.514 92.1.14.1 out-label pop

    local-label 9408 allocate per-prefix 92.0.0.8/32
    forward
      path 1 nexthop GigabitEthernet0/0/0/0.534 92.3.14.3 out-label 308

    local-label 9413 allocate per-prefix 92.0.0.13/32
    forward
      path 1 nexthop GigabitEthernet0/0/0/0.534 92.3.14.3 out-label 313
```

After these changes, we will perform some verification. Now that CSR1 does not learn LDP labels from XRv4, we should see the static labels in the LFIB. On XRv4, we see the static label in its LFIB as well.

R1#show mpls forwarding-table labels 100 - 199

Local	Outgoing	Prefix	Bytes Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched	interface	
103	9408	92.0.0.8/32	0	Gi2.514	92.1.14.14
113	9413	92.0.0.13/32	0	Gi2.514	92.1.14.14

RP/0/0/CPU0:XRv4#show mpls forwarding labels 9400 9499

Local	Outgoing	Prefix	Outgoing	Next Hop	Bytes
Label	Label	or ID	Interface		Switched
9401	Pop	92.0.0.1/32	Gi0/0/0/0.514	92.1.14.1	528
9408	308	92.0.0.8/32	Gi0/0/0/0.534	92.3.14.3	0
9413	313	92.0.0.13/32	Gi0/0/0/0.534	92.3.14.3	0

Next, we need to configure CSR3 with some label bindings. At a minimum, we need to configure local labels for 92.0.0.8/32 and 92.0.0.13/32 using values 308 and 313, since that is what XRv4 is sending outbound towards CSR3. We don't have to define a local label for 92.0.0.1/32, since LDP can do that; other routers can use this dynamic label to send traffic to CSR3 who will swap it for a static label towards XRv4 (specifically value 9401).

```

! CSR3
mpls static binding ipv4 92.0.0.1 255.255.255.255 output 92.3.14.14 9401
mpls static binding ipv4 92.0.0.8 255.255.255.255 308
mpls static binding ipv4 92.0.0.13 255.255.255.255 313

```

CSR3's LFIB is very interesting because we see LDP and static LSP being connected properly in both directions. The first output shows the static local labels, which is traffic destined for CSR8 and XRv3. Traffic arrives with a static label and is swapped to one of two dynamic labels, depending on the ECMP decision. The second output shows a dynamic local label allocated to LDP peers, but it is swapped to a static label for transmission to XRv4. The static range on every router is the dynamic range divided by ten, which makes it easy to spot at a glance.

```

R3#show mpls forwarding-table labels 300 - 399

```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Outgoing interface	Next Hop
308	2002	92.0.0.8/32	384	Gi2.523	92.2.3.2
	9007	92.0.0.8/32	570	Gi2.523	92.2.3.9
313	2012	92.0.0.13/32	2514	Gi2.523	92.2.3.2
	9006	92.0.0.13/32	2670	Gi2.523	92.2.3.9

```

R3#show mpls forwarding-table 92.0.0.1 32

```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Outgoing interface	Next Hop
3003	9401	92.0.0.1/32	0	Gi2.534	92.3.14.14

Next, we will manually trace the paths to ensure the label stacks are built properly. From CSR1, we will trace the LSP to CSR8. Even though the label wasn't learned from LDP, it still shows up in the LIB using the LDP show commands (I presume this is why LDP is required to be enabled for static labels to work). Since the traffic sourced from CSR1 is IP, the FIB is consulted, and label 9408 is pushed.

```

R1#show mpls ldp bindings 92.0.0.8 32
  lib entry: 92.0.0.8/32, rev 151
    local binding:  label: 103
    remote binding: lsr: 92.0.0.14:0, label: 9408

R1#show ip cef 92.0.0.8
92.0.0.8/32
  nexthop 92.1.14.14 GigabitEthernet2.514 label 9408

```

XRv4 is an ordinary P router that performs a swap operation from label 9408 to label 308. Both of these are static labels.

```

RP/0/0/CPU0:XRv4#show mpls forwarding labels 9408

```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched

```
-----
9408    308          92.0.0.8/32          Gi0/0/0/0.534 92.3.14.3      0
```

CSR3 is also a P routers performing a label swap, but it also connected the static and LDP LSPs. The exact path selects CSR2 based on the IPv4 source/destination ECMP hash.

```
R3#show mpls forwarding-table labels 308
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
308	2002	92.0.0.8/32	384	Gi2.523	92.2.3.2
	9007	92.0.0.8/32	570	Gi2.523	92.2.3.9

```
R3#show mpls forwarding-table exact-route label 308 ipv4 source 92.0.0.1 destination 92.0.0.8
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
308	2002	92.0.0.8/32	384	Gi2.523	92.2.3.2

CSR2 performs a normal pop operation (PHP) and delivers the IP packet to CSR8. We use traceroute on CSR1 to confirm the path and label swaps along the way. We can clearly see that CSR3 is the point at which the static and LDP LSPs connect.

```
R2#show mpls forwarding-table labels 2002
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
2002	Pop Label	92.0.0.8/32	12248	Gi2.528	92.2.8.8

```
R1#traceroute 92.0.0.8 source 92.0.0.1
```

Type escape sequence to abort.

Tracing the route to 92.0.0.8

VRF info: (vrf in name/id, vrf out name/id)

```
 1 92.1.14.14 [MPLS: Label 9408 Exp 0] 7 msec 5 msec 6 msec
 2 92.3.14.3  [MPLS: Label 308 Exp 0] 28 msec 30 msec 30 msec
 3 92.2.3.2   [MPLS: Label 2002 Exp 0] 21 msec 20 msec 20 msec
 4 92.2.8.8   21 msec 11 msec 11 msec
```

Tracing in the reverse direction from CSR8, we see two labels in the LIB learned from LDP and two ECMP paths to reach 92.0.0.1/32. Based on the ECMP hash, CEF selects CSR9 using label 9011.

```
R8#show mpls ldp bindings 92.0.0.1 32
```

```
lib entry: 92.0.0.1/32, rev 55
```

```
local binding: label: 8013
```

```
remote binding: lsr: 92.0.0.9:0, label: 9011
```

```
remote binding: lsr: 92.0.0.2:0, label: 2013
```

```
R8#show ip cef 92.0.0.1
```

```
92.0.0.1/32
```

```

nexthop 92.2.8.2 GigabitEthernet2.528 label 2013
nexthop 92.8.9.9 GigabitEthernet2.589 label 9011

```

```

R8#show ip cef exact-route 92.0.0.8 92.0.0.1
92.0.0.8 -> 92.0.0.1 => label 9011TAG adj out of GigabitEthernet2.589, addr
92.8.9.9

```

CSR9 swaps 9011 for 3003 and sends the packet to CSR3. Nothing special here.

```

R9#show mpls forwarding-table labels 9011

```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
9011	3003	92.0.0.1/32	570	Gi2.523	92.2.3.3

Next, CSR3 receives the packet with LDP label 3003 and swaps it for the static label 9401. This is connecting an LDP LSP to a static LSP, the reverse of what we saw earlier.

```

R3#show mpls forwarding-table labels 3003

```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
3003	9401	92.0.0.1/32	1944	Gi2.534	92.3.14.14

We manually programmed XRv4 to pop the topmost label when sending traffic to CSR1 for prefix 92.0.0.1/32, so label 9401 is removed to reveal the raw IP packet to CSR1. We confirm with traceroute, keeping in mind that starting it from CSR8 is going to alternative between ECMP paths (process switched). Traffic that was actually CEF-switched would always go through CSR9, though.

```

RP/0/0/CPU0:XRv4#show mpls forwarding labels 9401

```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
9401	Pop	92.0.0.1/32	Gi0/0/0/0.514	92.1.14.1	4830

```

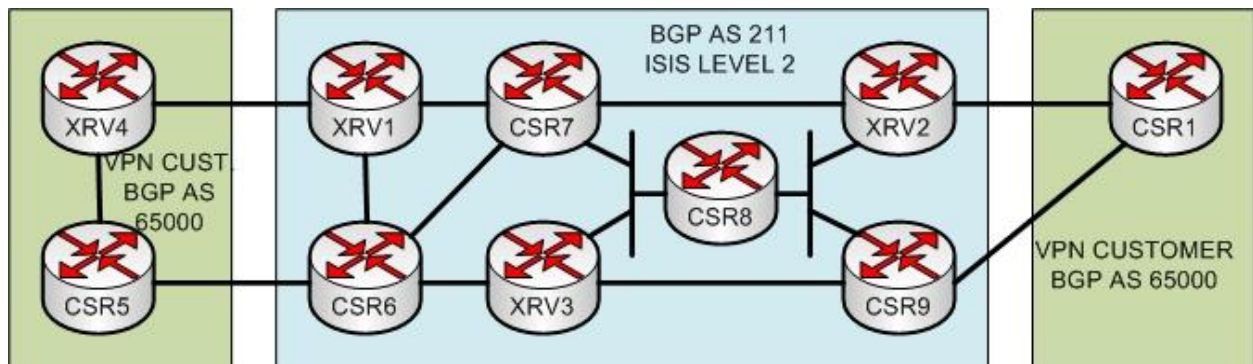
R8#traceroute 92.0.0.1 source 92.0.0.8 probe 1
Type escape sequence to abort.
Tracing the route to 92.0.0.1
VRF info: (vrf in name/id, vrf out name/id)
 1 92.8.9.9 [MPLS: Label 9011 Exp 0] 6 msec
 2 92.2.3.3 [MPLS: Label 3003 Exp 0] 8 msec
 3 92.3.14.14 [MPLS: Label 9401 Exp 0] 6 msec
 4 92.1.14.1 6 msec

```

Additional Reading – Reference configurations “mpls-ldp”

7.3 MPLS IP and MTU minor options

Rather than pollute the LDP lab by disabling the handy traceroute tool, I built another lab so we can demonstrate TTL handling and IP default route labeling supported in XE and XR. The lab will also focus on the extremely important but often overlooked issue of MPLS fragmentation and MTU adjustment. The network diagram is below and is a single BGP AS providing L3VPN (VPNv4/v6) to three customer sites. The customer sites use BGP as the PE-CE routing protocol and also for the backdoor link between XRv4 and CSR5. CSR1 is multi-homed to two PEs. The network core uses IS-IS with LDP and RSVP-TE, and both CSR7 and XRv3 are RR's with BGP add-path enabled for fast convergence. Most of these features are not relevant for testing the MPLS minor options, but it is good to have a moderately complex network rather than testing in isolation.



We will quickly skim the relevant configurations of the network. IS-IS configuration is L2 everywhere and is not examined. LDP is enabled on all IS-IS links with targeted session accepted everywhere for PE-P and P-P TE tunnel support. We can verify these things quickly; using a single command, I can see all of the IS-IS LSPs and their links. We can see 9 total vertices (7 routers and 2 DIS) with all of their links. This is very easy to read and it a good way to verify the IS-IS topology without relying on the RIB or ping/traceroute.

```
R7#show isis database detail level-2 | include Extended|^ [RX]
R6.00-00          0x0000000F  0x0A8B          873          0/0/0
Metric: 10        IS-Extended R7.00
Metric: 10        IS-Extended XRv1.00
Metric: 10        IS-Extended XRv3.00
R7.00-00          * 0x00000011  0x0FFA          680          0/0/0
Metric: 10        IS-Extended R6.00
Metric: 10        IS-Extended R8.02
Metric: 10        IS-Extended XRv1.00
Metric: 10        IS-Extended XRv2.00
R8.00-00          0x0000000F  0xD186          617          0/0/0
Metric: 10        IS-Extended R8.02
Metric: 10        IS-Extended R8.01
R8.01-00          0x0000000B  0x867E          510          0/0/0
Metric: 0         IS-Extended R8.00
Metric: 0         IS-Extended R9.00
Metric: 0         IS-Extended XRv2.00
R8.02-00          0x0000000C  0xB427          828          0/0/0
Metric: 0         IS-Extended R8.00
```

Metric: 0	IS-Extended R7.00		
Metric: 0	IS-Extended XRv3.00		
R9.00-00	0x0000000F 0xA9F0	693	0/0/0
Metric: 10	IS-Extended R8.01		
Metric: 10	IS-Extended XRv3.00		
XRv1.00-00	0x0000000B 0x7BA6	1001	0/0/0
Metric: 10	IS-Extended R6.00		
Metric: 10	IS-Extended R7.00		
XRv2.00-00	0x0000000E 0x33F2	1055	0/0/0
Metric: 10	IS-Extended R8.01		
Metric: 10	IS-Extended R7.00		
XRv3.00-00	0x0000000E 0xE1E7	1142	0/0/0
Metric: 10	IS-Extended R6.00		
Metric: 10	IS-Extended R8.02		
Metric: 10	IS-Extended R9.00		

We can also verify the LDP bindings very quickly. The summary command below shows that there are 7 prefixes with label bindings in the LIB, which equates to each loopback in the core. We can prove this by checking the LIB entries in summary form as well; we only see one prefix per router.

```
R7#show mpls ldp bindings summary
Total number of prefixes: 7
Generic label bindings
```

	assigned	learned
prefixes	in labels	out labels
7	7	35

```
Total tib route info allocated: 9
Previous tib remote label entries allocated Current/Total: 0/0
Previous tib remote label queues allocated Current/Total: 0/0

R7#show mpls ldp bindings | include lib
lib entry: 211.0.0.6/32, rev 2
lib entry: 211.0.0.7/32, rev 4
lib entry: 211.0.0.8/32, rev 24
lib entry: 211.0.0.9/32, rev 26
lib entry: 211.0.0.11/32, rev 28
lib entry: 211.0.0.12/32, rev 30
lib entry: 211.0.0.13/32, rev 32
```

MPLS-TE is enabled on every IGP enabled interface. We can quickly check the TED as well, observing that it is identical to the IS-IS database in terms of links. The first 7 entries are router nodes (actual routers), while the last 2 are network nodes (DIS). There are no TE tunnels in the network at present.

```
RP/0/0/CPU0:XRv3#show mpls traffic-eng topology brief | utility egrep '^IGP Id|Link'
Signalling error holddown: 10 sec Global Link Generation 2410
IGP Id: 0000.0000.0006.00, MPLS TE Id: 211.0.0.6 Router Node (IS-IS 211 level-2)
Link[0]:Point-to-Point, Nbr IGP Id:0000.0000.0007.00, Nbr Node Id:2, gen:2387
Link[1]:Point-to-Point, Nbr IGP Id:0000.0000.0011.00, Nbr Node Id:7, gen:2388
```



```

Link[2]:Point-to-Point, Nbr IGP Id:0000.0000.0013.00, Nbr Node Id:9, gen:2389
IGP Id: 0000.0000.0007.00, MPLS TE Id: 211.0.0.7 Router Node (IS-IS 211 level-2)
Link[0]:Point-to-Point, Nbr IGP Id:0000.0000.0006.00, Nbr Node Id:1, gen:2390
Link[1]:Broadcast, DR:0000.0000.0008.02, Nbr Node Id:5, gen:2391
Link[2]:Point-to-Point, Nbr IGP Id:0000.0000.0011.00, Nbr Node Id:7, gen:2392
Link[3]:Point-to-Point, Nbr IGP Id:0000.0000.0012.00, Nbr Node Id:8, gen:2393
IGP Id: 0000.0000.0008.00, MPLS TE Id: 211.0.0.8 Router Node (IS-IS 211 level-2)
Link[0]:Broadcast, DR:0000.0000.0008.02, Nbr Node Id:5, gen:2394
Link[1]:Broadcast, DR:0000.0000.0008.01, Nbr Node Id:4, gen:2395
IGP Id: 0000.0000.0009.00, MPLS TE Id: 211.0.0.9 Router Node (IS-IS 211 level-2)
Link[0]:Broadcast, DR:0000.0000.0008.01, Nbr Node Id:4, gen:2402
Link[1]:Point-to-Point, Nbr IGP Id:0000.0000.0013.00, Nbr Node Id:9, gen:2403
IGP Id: 0000.0000.0011.00, MPLS TE Id: 211.0.0.11 Router Node (IS-IS 211 level-2)
Link[0]:Point-to-Point, Nbr IGP Id:0000.0000.0006.00, Nbr Node Id:1, gen:2404
Link[1]:Point-to-Point, Nbr IGP Id:0000.0000.0007.00, Nbr Node Id:2, gen:2405
IGP Id: 0000.0000.0012.00, MPLS TE Id: 211.0.0.12 Router Node (IS-IS 211 level-2)
Link[0]:Broadcast, DR:0000.0000.0008.01, Nbr Node Id:4, gen:2406
Link[1]:Point-to-Point, Nbr IGP Id:0000.0000.0007.00, Nbr Node Id:2, gen:2407
IGP Id: 0000.0000.0013.00, MPLS TE Id: 211.0.0.13 Router Node (IS-IS 211 level-2)
Link[0]:Point-to-Point, Nbr IGP Id:0000.0000.0006.00, Nbr Node Id:1, gen:2408
Link[1]:Broadcast, DR:0000.0000.0008.02, Nbr Node Id:5, gen:2409
Link[2]:Point-to-Point, Nbr IGP Id:0000.0000.0009.00, Nbr Node Id:6, gen:2410
IGP Id: 0000.0000.0008.01, Network Node (IS-IS 211 level-2)
Link[0]:Broadcast, DR:0000.0000.0008.00, Nbr Node Id:3, gen:2396
Link[1]:Broadcast, DR:0000.0000.0009.00, Nbr Node Id:6, gen:2397
Link[2]:Broadcast, DR:0000.0000.0012.00, Nbr Node Id:8, gen:2398
IGP Id: 0000.0000.0008.02, Network Node (IS-IS 211 level-2)
Link[0]:Broadcast, DR:0000.0000.0008.00, Nbr Node Id:3, gen:2399
Link[1]:Broadcast, DR:0000.0000.0007.00, Nbr Node Id:2, gen:2400
Link[2]:Broadcast, DR:0000.0000.0013.00, Nbr Node Id:9, gen:2401

```

Next, we will quickly check the BGP details. Both CSR7 and XRv3 are RR's for VPNv4 and VPNv6. CSR7 is a "shadow RR" which advertises its second-best path to all clients, while XRv3 has no special BGP add-path/PIC configuration. This is mostly to support left-to-right traffic towards CSR1. First, we verify all of the sessions are up from the RR's for both AFIs. Each CE contributes 4 routes to the VPN, and since CSR1 is dual-homed, both XRv2 and CSR9 advertise the same prefixes. In total, 4 routes are learned from each PE.

```

R7#show bgp vpnv4 unicast all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State/PfxRcd
211.0.0.6      4      211     722    742      65    0    0 01:51:34      4
211.0.0.9      4      211     607    632      65    0    0 01:34:41      4
211.0.0.11     4      211     583    623      65    0    0 01:34:25      4
211.0.0.12     4      211     573    622      65    0    0 01:34:31      4

```

```

RP/0/0/CPU0:XRv3#show bgp vpnv4 unicast summary | begin ^Neighbor
Neighbor      Spk    AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  St/PfxRcd
211.0.0.6      0      211     592    566      45    0    0 01:31:17      4
211.0.0.9      0      211     582    566      45    0    0 01:31:15      4
211.0.0.11     0      211     564    566      45    0    0 01:31:14      4
211.0.0.12     0      211     554    566      45    0    0 01:31:16      4

```

The PE-CE links run independent IPv4/v6 BGP sessions for simplicity as XE and XR treat merged sessions differently. CSR1 uses weight from CSR9 to select it as the preferred exit point for outbound traffic. It also sets a high MED outbound to CSR9 to “hint” to AS 211 that XRv2 is the preferred ingress point into AS 65000. As an added bonus, I use an AS-path technique to ensure CSR1 does not become a transit router by only advertising routes with an AS-path length of the empty string (local routes only). We won’t verify every single aspect of this configuration for brevity.

```
! CSR1
ip as-path access-list 1 permit ^$

route-map RM_SET_MED permit 10
  set metric 111
route-map RM_SET_WEIGHT permit 10
  set weight 111

router bgp 65000
  address-family ipv4
    neighbor 10.1.9.9 route-map RM_SET_WEIGHT in
    neighbor 10.1.9.9 route-map RM_SET_MED out
    neighbor 10.1.9.9 filter-list 1 out

  address-family ipv6
    neighbor FD00:10:1:9::9 route-map RM_SET_WEIGHT in
    neighbor FD00:10:1:9::9 route-map RM_SET_MED out
    neighbor FD00:10:1:9::9 filter-list 1 out
```

The shadow RR configuration on CSR7 is shown below since it is worth revisiting (the details are in the BGP additional-paths section). We will see that the best-path to CSR1’s loopbacks is via XRv2, so CSR7 will advertise the alternative path via CSR9. We can see this by checking the outbound advertised-routes on the RR towards XRv1 in both AFIs.

```
! CSR7
address-family vpnv4
  bgp additional-paths select backup
  bgp additional-paths install
  no bgp recursion host
  neighbor IBGP advertise diverse-path backup

address-family vpnv6
  bgp additional-paths select backup
  bgp additional-paths install
  no bgp recursion host
  neighbor IBGP advertise diverse-path backup
```

Because the best path for routes from CSR1 is via XRv2, CSR7 selects the routes from CSR9 as backups and advertises those. Because the IGP cost was not a factor (MED was), we don’t need to tell the RRs to ignore the IGP cost in the best-path calculation as is often required.

```

R7#show bgp vpnv4 unicast rd 211:100 10.0.1.0/32
BGP routing table entry for 211:100:10.0.1.0/32, version 18
Paths: (2 available, best #1, no table)
  Additional-path-install
  Advertised to update-groups:
    2
  Refresh Epoch 1
  65000, (Received from a RR-client)
    211.0.0.12 (metric 10) (via default) from 211.0.0.12 (211.0.0.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:211:12
      mpls labels in/out nolabel/92006
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  65000, (Received from a RR-client)
    211.0.0.9 (metric 20) (via default) from 211.0.0.9 (211.0.0.9)
      Origin incomplete, metric 111, localpref 100, valid, internal,
      backup/repair
      Extended Community: RT:211:9
      mpls labels in/out nolabel/9001
      rx pathid: 0, tx pathid: 0

R7#show bgp vpnv4 unicast rd 211:100 neighbors 211.0.0.11 advertised-routes |
include bia
*bia10.0.1.0/32      211.0.0.9      111    100    0 65000 ?
*bia10.0.1.1/32     211.0.0.9      111    100    0 65000 ?
*bia10.0.1.2/32     211.0.0.9      111    100    0 65000 ?
*bia10.0.1.3/32     211.0.0.9      111    100    0 65000 ?

R7#show bgp vpnv6 unicast rd 211:100 neighbors 211.0.0.11 advertised-routes |
include bia
*bia::10:0:1:0/128  ::FFFF:211.0.0.9
*bia::10:0:1:1/128  ::FFFF:211.0.0.9
*bia::10:0:1:2/128  ::FFFF:211.0.0.9
*bia::10:0:1:3/128  ::FFFF:211.0.0.9

```

CSR6 and XRv1 both install these backup paths. This allows them to quickly failover if XRv2 fails or the primary route is otherwise lost. Again, none of this is directly relevant to MPLS minor options, but we want to make sure the network is functional before continuing.

```

! CSR6
router bgp 211
  address-family vpnv4
    bgp additional-paths select backup
    bgp additional-paths install
    no bgp recursion host

```

```

address-family vpnv6
  bgp additional-paths select backup
  bgp additional-paths install
  no bgp recursion host

! XRv1
route-policy RPL_ADD_PATH
  set path-selection backup 1 install
end-policy

router bgp 211
  address-family vpnv4 unicast
    additional-paths selection route-policy RPL_ADD_PATH

  address-family vpnv6 unicast
    additional-paths selection route-policy RPL_ADD_PATH

```

We quickly verify the BGP and CEF tables on CSR6 and to XRv1 to confirm proper operation.

```

R6#show bgp vpnv4 unicast vrf J 10.0.1.3/32
BGP routing table entry for 211:100:10.0.1.3/32, version 41
Paths: (2 available, best #1, table J)
  Additional-path-install
  Advertised to update-groups:
    3
  Refresh Epoch 1
  65000
    211.0.0.12 (metric 20) (via default) from 211.0.0.13 (211.0.0.13)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:211:12
      Originator: 211.0.0.12, Cluster list: 211.0.0.13
      mpls labels in/out nolabel/92009
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 3
  65000
    211.0.0.9 (metric 20) (via default) from 211.0.0.7 (211.0.0.7)
      Origin incomplete, metric 111, localpref 100, valid, internal,
      backup/repair
      Extended Community: RT:211:9
      Originator: 211.0.0.9, Cluster list: 211.0.0.7
      mpls labels in/out nolabel/9007
      rx pathid: 0, tx pathid: 0

R6#show ip cef vrf J 10.0.1.3 detail
10.0.1.3/32, epoch 0, flags [rib defined all labels]
  recursive via 211.0.0.12 label 92009
    nexthop 211.6.7.7 GigabitEthernet2.567 label 7001
  recursive via 211.0.0.9 label 9007, repair
    nexthop 211.6.13.13 GigabitEthernet2.563 label 93002

```

```

RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast vrf J 10.0.1.3/32 | begin Paths
Paths: (2 available, best #2)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  65000
    211.0.0.9 (metric 30) from 211.0.0.7 (211.0.0.9)
      Received Label 9007
      Origin incomplete, metric 111, localpref 100, valid, internal, backup,
add-path, import-candidate, imported
      Received Path ID 0, Local Path ID 2, version 99
      Extended community: RT:211:9
      Originator: 211.0.0.9, Cluster list: 211.0.0.7
      Source VRF: J, Source Route Distinguisher: 211:100
  Path #2: Received by speaker 0
  Not advertised to any peer
  65000
    211.0.0.12 (metric 20) from 211.0.0.13 (211.0.0.12)
      Received Label 92009
      Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, imported
      Received Path ID 0, Local Path ID 1, version 28
      Extended community: RT:211:12
      Originator: 211.0.0.12, Cluster list: 211.0.0.13
      Source VRF: J, Source Route Distinguisher: 211:100

RP/0/0/CPU0:XRv1#show cef vrf J 10.0.1.3/32
10.0.1.3/32, version 161, internal 0x5000001 0x0 (ptr 0xa1447bf4) [1], 0x0
(0x0), 0x208 (0xa187712c)
Prefix Len 32, traffic index 0, precedence n/a, priority 3
  via 211.0.0.9, 4 dependencies, recursive, backup [flags 0x6100]
    path-idx 0 NHID 0x0 [0xa15d5bf4 0x0]
    recursion-via-/32
    next hop VRF - 'default', table - 0xe0000000
    next hop 211.0.0.9 via 91005/0/21
      next hop 211.7.11.7/32 Gi0/0/0/0.571 labels imposed {7016 9007}
      next hop 211.6.11.6/32 Gi0/0/0/0.576 labels imposed {6000 9007}
  via 211.0.0.12, 5 dependencies, recursive [flags 0x6000]
    path-idx 1 NHID 0x0 [0xa15d6074 0x0]
    recursion-via-/32
    next hop VRF - 'default', table - 0xe0000000
    next hop 211.0.0.12 via 91002/0/21
      next hop 211.7.11.7/32 Gi0/0/0/0.571 labels imposed {7001 92009}

```

Since there is a backdoor between CSR5 and XRv4, the primary path should be the MPLS network. Rather than adjust BGP path selection, I use aggregation and longest-match routing to achieve this. CSR5 aggregates its loopbacks as a summary-only BGP aggregate. This also has the no-export community so

that it never gets advertised to the SP. CSR5 unsuppresses the longer matches and advertises those to CSR6, but only the summary goes to XRv4 since there is no unsuppress map on that peer. Also note that CSR5 uses an alternative technique to prevent transit-AS service. CSR1 used AS-path regex while CSR5 uses the no-export community inbound from CSR6.

```
! CSR5
route-map RM_SET_NO_EXPORT permit 10
  set community no-export additive
route-map RM_UNSUPPRESS permit 10

router bgp 65000
  address-family ipv4
    aggregate-address 10.0.5.0 255.255.255.252 summary-only route-map RM_SET_NO_EXPORT
    neighbor 10.5.6.6 unsuppress-map RM_UNSUPPRESS
    neighbor 10.5.6.6 route-map RM_SET_NO_EXPORT in

  address-family ipv6
    aggregate-address ::10:0:5:0/112 summary-only attribute-map RM_SET_NO_EXPORT
    neighbor FD00:10:5:6::6 unsuppress-map RM_UNSUPPRESS
    neighbor FD00:10:5:6::6 route-map RM_SET_NO_EXPORT in
```

Since XR does not appear to support an unsuppress map, we create the summary as an additional route then manually filter the longer-matches from the iBGP backdoor on XRv4. For practice, I use fancy RPLs which recycle prefix-sets that match all loopbacks regardless of mask, then apply a second inline prefix-set to ensure only host routes within that primary range are matched. These are dropped when routes are advertised to CSR5 via iBGP.

```
! XRv4
prefix-set PS_LOOPBACKS
  10.0.14.0/24 le 32
end-set

prefix-set PS_LOOPBACKS_V6
  ::10:0:14:0/112 le 128
end-set

route-policy RPL_DENY_HOST_ROUTES
  if destination in PS_LOOPBACKS and destination in (0.0.0.0/0 ge 32) then
    drop
  else
    pass
  endif
end-policy

route-policy RPL_DENY_HOST_ROUTES_V6
  if destination in PS_LOOPBACKS_V6 and destination in (:::/0 ge 128) then
    drop
  else
```

```

        pass
    endif
end-policy

router bgp 65000
  address-family ipv4 unicast
    aggregate-address 10.0.14.0/30 route-policy RPL_SET_NO_EXPORT

  address-family ipv6 unicast
    aggregate-address ::10:0:14:0/112 route-policy RPL_SET_NO_EXPORT

  neighbor 10.5.14.5
    address-family ipv4 unicast
    route-policy RPL_DENY_HOST_ROUTES out

  neighbor fd00:10:5:14::5
    address-family ipv6 unicast
    route-policy RPL_DENY_HOST_ROUTES_V6 out

```

Since this configuration is very involved, we will quickly check CSR5 advertised routes. Towards the provider, the host routes are allowed, but not the summary. Towards the backdoor peer, the summary is allowed, but not the local host routes. Other routes from CSR1 are still allowed as they were not suppressed by the summary nor explicitly filtered.

```

R5#show bgp ipv4 unicast neighbors 10.5.6.6 advertised-routes | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
s> 10.0.5.0/32      0.0.0.0          0        32768 ?
s> 10.0.5.1/32      0.0.0.0          0        32768 ?
s> 10.0.5.2/32      0.0.0.0          0        32768 ?
s> 10.0.5.3/32      0.0.0.0          0        32768 ?

```

Total number of prefixes 4

```

R5#show bgp ipv6 unicast neighbors FD00:10:5:6::6 advertised-routes | begin
Neighbor

```

	Network	Next Hop	Metric	LocPrf	Weight	Path
s>	::10:0:5:0/128	::	0		32768	?
s>	::10:0:5:1/128	::	0		32768	?
s>	::10:0:5:2/128	::	0		32768	?
s>	::10:0:5:3/128	::	0		32768	?

```

R5#show bgp ipv4 unicast neighbors 10.5.14.14 advertised-routes | begin
Network

```

	Network	Next Hop	Metric	LocPrf	Weight	Path
*>	10.0.1.0/32	10.5.6.6			0 211 211	?
*>	10.0.1.1/32	10.5.6.6			0 211 211	?
*>	10.0.1.2/32	10.5.6.6			0 211 211	?
*>	10.0.1.3/32	10.5.6.6			0 211 211	?
*>	10.0.5.0/30	0.0.0.0			32768	i
*>	10.0.14.0/32	10.5.6.6			0 211 211	?

```
*> 10.0.14.1/32      10.5.6.6              0 211 211 ?
*> 10.0.14.2/32      10.5.6.6              0 211 211 ?
*> 10.0.14.3/32      10.5.6.6              0 211 211 ?
```

R5#show bgp ipv6 unicast neighbors FD00:10:5:14::14 advertised-routes | begin Network

Network	Next Hop	Metric	LocPrf	Weight	Path
*> ::10:0:1:0/128	FD00:10:5:6::6			0	211 211 ?
*> ::10:0:1:1/128	FD00:10:5:6::6			0	211 211 ?
*> ::10:0:1:2/128	FD00:10:5:6::6			0	211 211 ?
*> ::10:0:1:3/128	FD00:10:5:6::6			0	211 211 ?
*> ::10:0:5:0/112	::			32768	i
*> ::10:0:14:0/128	FD00:10:5:6::6			0	211 211 ?
*> ::10:0:14:1/128	FD00:10:5:6::6			0	211 211 ?
*> ::10:0:14:2/128	FD00:10:5:6::6			0	211 211 ?
*> ::10:0:14:3/128	FD00:10:5:6::6			0	211 211 ?

We conduct a similar set of checks on XRv4. The results are nearly identical despite the configuration method being different. The host-routes are advertised to the provider and the summary is advertised to the iBGP backdoor peer, but not vice versa.

RP/0/0/CPU0:XRv4#show bgp ipv4 unicast neighbors 10.11.14.11 advertised-routes

Network	Next Hop	From	AS Path
10.0.14.0/32	10.11.14.14	Local	65000?
10.0.14.1/32	10.11.14.14	Local	65000?
10.0.14.2/32	10.11.14.14	Local	65000?
10.0.14.3/32	10.11.14.14	Local	65000?

RP/0/0/CPU0:XRv4#show bgp ipv6 unicast neighbors fd00:10:11:14::11 advertised-routes

Network	Next Hop	From	AS Path
::10:0:14:0/128	fd00:10:11:14::14	Local	65000?
::10:0:14:1/128	fd00:10:11:14::14	Local	65000?
::10:0:14:2/128	fd00:10:11:14::14	Local	65000?
::10:0:14:3/128	fd00:10:11:14::14	Local	65000?

RP/0/0/CPU0:XRv4#show bgp ipv4 unicast neighbors 10.5.14.5 advertised-routes

Network	Next Hop	From	AS Path
10.0.1.0/32	10.5.14.14	10.11.14.11	211 211?
10.0.1.1/32	10.5.14.14	10.11.14.11	211 211?
10.0.1.2/32	10.5.14.14	10.11.14.11	211 211?
10.0.1.3/32	10.5.14.14	10.11.14.11	211 211?
10.0.5.0/32	10.5.14.14	10.11.14.11	211 211?

10.0.5.1/32	10.5.14.14	10.11.14.11	211 211?
10.0.5.2/32	10.5.14.14	10.11.14.11	211 211?
10.0.5.3/32	10.5.14.14	10.11.14.11	211 211?
10.0.14.0/30	10.5.14.14	Local Aggregate i	

RP/0/0/CPU0:XRv4#show bgp ipv6 unicast neighbors fd00:10:5:14::5 advertised-routes

Network	Next Hop	From	AS Path
::10:0:1:0/128	fd00:10:5:14::14	fd00:10:11:14::11	211 211?
::10:0:1:1/128	fd00:10:5:14::14	fd00:10:11:14::11	211 211?
::10:0:1:2/128	fd00:10:5:14::14	fd00:10:11:14::11	211 211?
::10:0:1:3/128	fd00:10:5:14::14	fd00:10:11:14::11	211 211?
::10:0:5:0/128	fd00:10:5:14::14	fd00:10:11:14::11	211 211?
::10:0:5:1/128	fd00:10:5:14::14	fd00:10:11:14::11	211 211?
::10:0:5:2/128	fd00:10:5:14::14	fd00:10:11:14::11	211 211?
::10:0:5:3/128	fd00:10:5:14::14	fd00:10:11:14::11	211 211?
::10:0:14:0/112	fd00:10:5:14::14	Local Aggregate i	

We conduct 3 traceroutes to verify connectivity. From XRv4, we trace to CSR5 to ensure MPLS is used. Also from XRv4, we trace to CSR1 to ensure XRv2 is the ingress point into AS 65000. Last, we trace from CSR1 to CSR5 to ensure CSR9 is the egress point from AS 65000. This concludes the basic network verification.

RP/0/0/CPU0:XRv4#traceroute ::10:0:5:1 source ::10:0:14:3
Type escape sequence to abort.
Tracing the route to ::10:0:5:1

```

1  fd00:10:11:14::11 0 msec 0 msec 0 msec
2  fd00:10:5:6::6 [MPLS: Label 6005 Exp 0] 0 msec 0 msec 0 msec
3  fd00:10:5:6::5 0 msec 0 msec 0 msec

```

```
RP/0/0/CPU0:XRv4#traceroute ::10:0:1:0 source ::10:0:14:1
Type escape sequence to abort.
Tracing the route to ::10:0:1:0

 1  fd00:10:11:14::11 0 msec 0 msec 0 msec
 2  ::ffff:211.7.11.7 [MPLS: Labels 7001/92010 Exp 0] 0 msec 0 msec 49 msec
 3  fd00:10:1:12::12 [MPLS: Label 92010 Exp 0] 9 msec 0 msec 0 msec
 4  fd00:10:1:12::1 0 msec 0 msec 0 msec
```

```
R1#traceroute 10.0.5.0 source 10.0.1.2
Type escape sequence to abort.
Tracing the route to 10.0.5.0
VRF info: (vrf in name/id, vrf out name/id)
 1  10.1.9.9 4 msec 4 msec 4 msec
 2  211.9.13.13 [MPLS: Labels 93005/6023 Exp 0] 8 msec 6 msec 6 msec
 3  10.5.6.6 [MPLS: Label 6023 Exp 0] 15 msec 16 msec 15 msec
 4  10.5.6.5 20 msec 10 msec 10 msec
```

The first and most important thing to do is adjust MTU. We will focus on the link between CSR7 and XRv1 so we can see XE and XR commands together. The XE command to reveal the MPLS MTU is intuitive but the XR one is not. We must invoke the interface manager (IM) process, but this also shows us the other relevant MTUs such as IPv4, IPv6, CLNS, etc.

```
R7#show mpls interfaces GigabitEthernet2.571 detail
Interface GigabitEthernet2.571:
  Type Unknown
  IP labeling enabled (ldp) :
    IGP config
  LSP Tunnel labeling enabled
  IP FRR labeling not enabled
  BGP labeling not enabled
  MPLS operational
  MTU = 1500
```

```
RP/0/0/CPU0:XRv1#show im database interface gigabitEthernet 0/0/0/0.571 |
begin Protocol
  Protocol          Caps (state, mtu)
  -----
  None              vlan_jump (up, 1518)
  None              spio (up, 1518)
  None              dot1q (up, 1518)
  arp               arp (up, 1500)
  clns              clns (up, 1500)
  ipv4              ipv4 (up, 1500)
  mpls              mpls (up, 1500)
  ipv6              ipv6_preswitch (up, 1500)
  ipv6              ipv6 (up, 1500)
```

Considering the MPLS MTU is equal to the IPv4/v6 MTUs, this is going to cause a problem. If a customer sends 1500 byte packets into the network, we know that at least one 4-byte MPLS shim-header will be added to the packets in the case of L3VPN. In some designs, this could be closer to 5 labels as seen in the carrier supporting carrier (CSC) section, and for L2VPN, may include a 4-byte control-word (CW). While the traffic still flows, it will be fragmented. This is only true for IPv4 traffic; IPv6 traffic or non-IP traffic is simply discarded. First, we will prove the easier claim about IPv6. The PE imposes a two-label stack of {7001 92010} which is 8 bytes of encapsulation. This allows XRv4 to send packets up to 1492 bytes in size, but not 1493 bytes or larger.

```
RP/0/0/CPU0:XRv1#show cef vrf J ::10:0:1:0 | utility egrep 'via|labels'
  via ::ffff:211.0.0.9, 2 dependencies, recursive, backup [flags 0x6100]
    recursion-via-/128
    next hop ::ffff:211.0.0.9 via ::ffff:211.0.0.9:0
      next hop 211.7.11.7/32 Gi0/0/0/0.571 labels imposed {7016 9012}
      next hop 211.6.11.6/32 Gi0/0/0/0.576 labels imposed {6000 9012}
  via ::ffff:211.0.0.12, 3 dependencies, recursive [flags 0x6000]
    recursion-via-/128
    next hop ::ffff:211.0.0.12 via ::ffff:211.0.0.12:0
      next hop 211.7.11.7/32 Gi0/0/0/0.571 labels imposed {7001 92010}
```

```
RP/0/0/CPU0:XRv4#ping ::10:0:1:0 source ::10:0:14:1 size 1492
Type escape sequence to abort.
Sending 5, 1492-byte ICMP Echos to ::10:0:1:0, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/191/259 ms
```

```
RP/0/0/CPU0:XRv4#ping ::10:0:1:0 source ::10:0:14:1 size 1493
Type escape sequence to abort.
Sending 5, 1493-byte ICMP Echos to ::10:0:1:0, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
```

The situation is similar for IPv4. Going to CSR1, there are still 2 labels {7001 92006} which is 8 bytes of encapsulation. This time we perform 3 pings. First, we send packets of size 1492 which DF-bit set, which we expect to work. Next, we sent packets too large with DF-bit set and watch them fail. Third, we clear the DF-bit, which allows the packets to be fragmented before CEF imposes the labels, and this only works for IPv4. This is not a good practice since fragmentation is generally viewed unfavorably in IP networks due to increased CPU load and process switching on network devices. If fragmentation must occur, it makes more sense to offload it to end-hosts.

```
RP/0/0/CPU0:XRv1#show cef vrf J 10.0.1.0 | utility egrep 'via|labels'
  via 211.0.0.9, 4 dependencies, recursive, backup [flags 0x6100]
    recursion-via-/32
    next hop 211.0.0.9 via 91005/0/21
      next hop 211.7.11.7/32 Gi0/0/0/0.571 labels imposed {7016 9001}
```

```
next hop 211.6.11.6/32 Gi0/0/0/0.576 labels imposed {6000 9001}
via 211.0.0.12, 5 dependencies, recursive [flags 0x6000]
recursion-via-/32
next hop 211.0.0.12 via 91002/0/21
next hop 211.7.11.7/32 Gi0/0/0/0.571 labels imposed {7001 92006}
```

```
RP/0/0/CPU0:XRv4#ping 10.0.1.0 source 10.0.14.2 size 1492 df-bit
Type escape sequence to abort.
Sending 5, 1492-byte ICMP Echos to 10.0.1.0, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/83/129 ms
```

```
RP/0/0/CPU0:XRv4#ping 10.0.1.0 source 10.0.14.2 size 1493 df-bit
Type escape sequence to abort.
Sending 5, 1493-byte ICMP Echos to 10.0.1.0, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
```

```
RP/0/0/CPU0:XRv4#ping 10.0.1.0 source 10.0.14.2 size 1493
Type escape sequence to abort.
Sending 5, 1493-byte ICMP Echos to 10.0.1.0, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 29/121/189 ms
```

Since MPLS encapsulation is inserted after the layer 2 header but before the layer 3 header, we can conclude that the layer 2 MTU $\geq$ MPLS MTU $\geq$ layer 3 MTU. In our case, the layer 2 MTU is 1518 to account for 14 bytes of Ethernet and 4 bytes of 802.1q encapsulation. The MPLS and IPv4/v6 MTUs are both 1500, so the formula holds true. Ideally, none of these MTUs would ever be equal, as it introduces the problems shown above. If we adjust the interface MTU, this value is inherited by all protocols enabled on that interface as well. We will set it to 2000 on the physical interfaces of CSR7 and XRv1.

```
! CSR7
interface GigabitEthernet2
    mtu 2000

! XRv1
interface GigabitEthernet0/0/0/0
    mtu 2000
```

Now, the IPv4, IPv6, and MPLS MTUs are all 2000, which is better but still kind of sloppy. If for whatever reason an IP packet of size 2000 entered an ingress LSR, we would have the same issue. For that reason, it makes sense for the MTU formula to be layer 2 MTU $>$ MPLS MTU $>$ layer 3 MTU. The option of equality is removed which means the outer encapsulation MTU should be strictly greater than anything inside. XR displays the information differently, subtracting 14 bytes to account for the Ethernet header and allowing 4 extra bytes for 802.1q. We can still see that IPv4/v6 and MPLS MTUs are equal, though.

```
R7#show mpls interfaces GigabitEthernet2.571 detail | include MTU
MTU = 2000
```

```
R7#show ip interface gigabitEthernet 2.571 | include MTU
MTU is 2000 bytes
```

```
R7#show ipv6 interface gigabitEthernet 2.571 | include MTU
MTU is 2000 bytes
```

```
RP/0/0/CPU0:XRv1#show im database interface gigabitEthernet 0/0/0/0.571 |
begin Protocol
```

Protocol	Caps (state, mtu)
-----	-----
None	vlan_jump (up, 2004)
None	spio (up, 2004)
None	dot1q (up, 2004)
arp	arp (up, 1986)
clns	clns (up, 1986)
ipv4	ipv4 (up, 1986)
mpls	mpls (up, 1986)
ipv6	ipv6_preswitch (up, 1986)
ipv6	ipv6 (up, 1986)

Let's assume that our network supports 1500 byte IPv4 and IPv6 packets and that jumbo frames are not allowed. We are essentially going to allow "baby giants", which are frames just slightly larger than 1500 bytes, but not enormous 9000-byte packets. To do this, we will adjust the IP and IPv6 MTUs on our logical interfaces between CSR7 and XRv1 to 1500. Be mindful of adjusting MTUs as some protocols, such as OSPF, require them to match in most cases (can be ignored, but this is bad practice).

```
! CSR7
interface GigabitEthernet2.571
 ip mtu 1500
 ipv6 mtu 1500

! XRv1
interface GigabitEthernet0/0/0/0.571
 ipv4 mtu 1500
 ipv6 mtu 1500
```

When we verify the MTUs now, we can see the IPv4 and IPv6 MTUs are back to 1500, but the MPLS MTU remains unchanged. The MPLS MTU is still using the same value as was configured on the physical interface, which makes sense. Since MPLS encapsulates IPv4/IPv6, it wouldn't make sense for it to assume any MTU settings from those tunneled protocols.

```
R7#show mpls interfaces GigabitEthernet2.571 detail | include MTU
MTU = 2000
```

```
R7#show ip interface gigabitEthernet 2.571 | include MTU
MTU is 1500 bytes
```

```
R7#show ipv6 interface gigabitEthernet 2.571 | include MTU
MTU is 1500 bytes
```

```
RP/0/0/CPU0:XRv1#show im database interface gigabitEthernet 0/0/0/0.571 |
begin Protocol
  Protocol          Caps (state, mtu)
  -----          -
  None              vlan_jump (up, 2004)
  None              spio (up, 2004)
  None              dot1q (up, 2004)
  arp               arp (up, 1986)
  clns              clns (up, 1986)
  ipv4              ipv4 (up, 1500)
  mpls              mpls (up, 1986)
  ipv6              ipv6_preswitch (up, 1986)
  ipv6              ipv6 (up, 1500)
```

Let's consider our network for a moment. At present, all LSPs in this network use 1 or 2 labels in the stack. If we built a TE tunnel from PE-P or P-P, there could potentially be 3 labels. Without TE-FRR or CSC in the network, or any fancy features like flow-aware transport (FAT), entropy labels, control-words, etc, we can assume the maximum label stack depth is 3. Thus, it would make sense for the MPLS MTU to be at least 1512, where the additional 12 bytes accounts for 3 labels. Since the MPLS MTU is currently 2000, we wouldn't have the fragmentation issues anymore, but I adjust the MPLS MTU for completeness.

```
! CSR7
interface GigabitEthernet2.571
  mpls mtu 1512

! XRv1
interface GigabitEthernet0/0/0/0.571
  mpls
    mtu 1512
```

Now, we see the MTUs are properly adjusted. One note about MTU adjustment; the layer 2 network (in this case Ethernet) must also support MTUs larger than 1500 for this to work. Had we been dealing with L2VPNs, our MPLS MTU would have to be larger to account for tunneled Ethernet headers, control-words, and other components. The math follows the same logic and is not worth demonstrating. I set the MTU to 2000 as a demonstration, but the physical switched network in this lab supports MTUs of up to 9000 bytes.

```
R7#show mpls interfaces GigabitEthernet2.571 detail | include MTU
MTU = 1512
```

```
RP/0/0/CPU0:XRv1#show im database interface gigabitEthernet 0/0/0/0.571 |
begin Protocol
  Protocol          Caps (state, mtu)
  -----
None               vlan_jump (up, 2004)
None               spio (up, 2004)
None               dot1q (up, 2004)
arp                arp (up, 1986)
clns                clns (up, 1986)
ipv4                ipv4 (up, 1500)
mpls                mpls (up, 1512)
ipv6                ipv6_preswitch (up, 1986)
ipv6                ipv6 (up, 1500)
```

Because these changes were only made on a small part of the network, we will make the changes on all nodes in the MPLS core for consistency. I also adjust the IPv4/v6 MTUs on the PE-CE interfaces since they share the same physical interface as the core links. The configurations are not shown, but they set the interface MTU to 2000, MPLS MTU to 1512, and IPv4/v6 MTU to 1500. Now, we can verify that IPv6 connectivity works with 1500 byte packets. We also verify that IPv4 connectivity works with 1500 byte packets without fragmentation, which is ideal.

```
RP/0/0/CPU0:XRv4#ping ::10:0:1:0 source ::10:0:14:1 size 1500
Type escape sequence to abort.
Sending 5, 1500-byte ICMP Echos to ::10:0:1:0, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/185/249 ms
```

```
RP/0/0/CPU0:XRv4#ping 10.0.1.0 source 10.0.14.2 size 1500 df-bit
Type escape sequence to abort.
Sending 5, 1500-byte ICMP Echos to 10.0.1.0, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/75/179 ms
```

We can use EPC inbound on CSR7 to see these packets. The total size is 1526 bytes: 1500 IPv4 + 8 MPLS + 4 dot1q + 14 Ethernet. This packet only has a 2-label stack, which is highlighted. We saw earlier that the labels were {7001 92006} and this confirms it. We technically still have room for one more label since the size of the MPLS packet is 1508 bytes (starting with the topmost label) and the MPLS MTU is 1512 everywhere.

```
R7#show monitor capture CAP buffer detailed
 2 1526 2.350979 00:50:56:A9:2D:C6 -> 00:50:56:A9:EA:77 MPLS unicast
0000: 005056A9 EA770050 56A92DC6 81000DF3 .PV..w.PV.-.....
0010: 884701B5 90FE1676 61FE4500 05DC0000 .G.....va.E.....
0020: 4000FE01 541F0A00 0E020A00 01000800 @...T.....
0030: 6800A0B1 0000ABCD ABCDABCD ABCDABCD h.....
```

To ensure the network can support a third label without fragmentation, we build a simple TE tunnel from XRv1 (PE) to CSR8 (P) traversing CSR7. This tunnel will have LDP enabled on it, which triggers a targeted session to CSR8. This is necessary so that the head-end can push an additional label, which will be CSR8's LDP label to reach 211.0.0.12/32, the remote PE. The result will be a 3 label stack that we will verify shortly. We use this tunnel only for traffic going to XRv2, so a static route is appropriate.

```
! XRv1
explicit-path name EP_11_7_8
  index 10 next-address strict ipv4 unicast 211.0.0.7
  index 20 next-address strict ipv4 unicast 211.0.0.8

interface tunnel-te100
  description PE-P TUNNEL TO CSR8
  ipv4 unnumbered Loopback0
  logging events all
  destination 211.0.0.8
  path-option 10 explicit name EP_11_7_8

router static
  address-family ipv4 unicast
    211.0.0.12/32 tunnel-te100

mpls ldp
  interface tunnel-te100
```

We verify that the tunnel is up, is following the proper path, and had an LDP neighbor across it.

```
RP/0/0/CPU0:XRv1#show mpls traffic-eng tunnels 100 brief
```

TUNNEL NAME	DESTINATION	STATUS	STATE
tunnel-te100	211.0.0.8	up	up

```
Displayed 1 (of 1) heads, 0 (of 0) midpoints, 0 (of 0) tails
Displayed 1 up, 0 down, 0 recovering, 0 recovered heads

RP/0/0/CPU0:XRv1#show mpls traffic-eng tunnels 100 detail | begin Path Info
Path Info:
  Outgoing:
    Explicit Route:
      Strict, 211.7.11.7
      Strict, 211.7.8.7
      Strict, 211.7.8.8
      Strict, 211.0.0.8

RP/0/0/CPU0:XRv1#show mpls ldp neighbor 211.0.0.8:0
Peer LDP Identifier: 211.0.0.8:0
  TCP connection: 211.0.0.8:646 - 211.0.0.11:25230
  Graceful Restart: No
  Session Holdtime: 180 sec
  State: Oper; Msgs sent/rcvd: 13/14; Downstream-Unsolicited
```



```

Up time: 00:03:34
LDP Discovery Sources:
  IPv4: (1)
    Targeted Hello (211.0.0.11 -> 211.0.0.8, active)
[snip]

```

With the tunnel built properly, we will verify the label stack piece-by-piece for practice. The VPNv4 route uses label 92006 which was allocated by XRv2 to describe reachability to the final destination, 10.0.1.0/32.

```

RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast vrf J 10.0.1.0/32 | begin 211.0.0.12
211.0.0.12 from 211.0.0.13 (211.0.0.12)
  Received Label 92006
    Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, imported
    Received Path ID 0, Local Path ID 1, version 25
    Extended community: RT:211:12
    Originator: 211.0.0.12, Cluster list: 211.0.0.13
    Source VRF: J, Source Route Distinguisher: 211:100

```

Next, XRv1 looks up the path to 211.0.0.12 in the global table. It is a static route via a TE-tunnel, and since the tunnel destination is different than the BGP next-hop, the router consults its LDP LIB to find a label for 211.0.0.12/32. CSR8 allocates label 8001 to describe its IGP path to XRv2, which is the remote PE. By exposing this label to CSR8, transport to XRv2 is achieved. XRv1's FIB verifies the 2-label stack so far.

```

RP/0/0/CPU0:XRv1#show route ipv4 211.0.0.12
Routing entry for 211.0.0.12/32
  Known via "static", distance 1, metric 0 (connected)
  Routing Descriptor Blocks
    directly connected, via tunnel-te100
    Route metric is 0
  No advertising protos.

```

```

RP/0/0/CPU0:XRv1#show mpls ldp bindings 211.0.0.12/32 neighbor 211.0.0.8
211.0.0.12/32, rev 11
  Local binding: label: 91002
  Remote bindings: (3 peers)
    Peer          Label
    -----
    211.0.0.8:0   8001

```

```

RP/0/0/CPU0:XRv1#show cef vrf J ipv4 10.0.1.0/32 | begin 211.0.0.12
via 211.0.0.12, 5 dependencies, recursive [flags 0x6000]
  path-idx 1 NHID 0x0 [0xa15d6074 0x0]
  recursion-via-/32
  next hop VRF - 'default', table - 0xe0000000

```

```
next hop 211.0.0.12 via 91002/0/21
next hop 0.0.0.0/32 tt100 labels imposed {8001 92006}
```

Last, the TE label is pushed. Because the IGP route was via a TE-tunnel, the RSVP-TE label from CSR7 is used. This describes a path to CSR8 along the TE LSP, and uses value 7003. The full label stack becomes {7003 8001 92006}.

```
RP/0/0/CPU0:XRv1#show mpls traffic-eng tunnels 100 detail | include Label
Outgoing Interface: GigabitEthernet0/0/0/0.571, Outgoing Label: 7003
```

To verify it, we will ping inside the VPN again using 1500-byte packets (DF-bit set) with EPC enabled on CSR7. This reveals the 3-label stack and proves that fragmentation did not occur, confirming our MPLS MTU optimizations. Notice the size is exactly 4 bytes larger than the last test at 1530 bytes. I also highlight the 3-label stack we verified earlier to prove that the TE-tunnel from PE-P is working and traffic is not fragmented. Although EPC only shows the IPv4 packet, we also use ICMPv6 to verify that IPv6 traffic is not being dropped due to MTU violations.

```
RP/0/0/CPU0:XRv4#ping 10.0.1.0 source 10.0.14.2 size 1500 df-bit
Type escape sequence to abort.
Sending 5, 1500-byte ICMP Echos to 10.0.1.0, timeout is 2 seconds:
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/65/149 ms
```

```
RP/0/0/CPU0:XRv4#ping ::10:0:1:0 source ::10:0:14:1 size 1500
Type escape sequence to abort.
Sending 5, 1500-byte ICMP Echos to ::10:0:1:0, timeout is 2 seconds:
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/187/249 ms
```

```
R7#show monitor capture CAP buffer detailed
 1 1530 0.720015 00:50:56:A9:2D:C6 -> 00:50:56:A9:EA:77 MPLS unicast
0000: 005056A9 EA770050 56A92DC6 81000DF3 .PV..w.PV.-.....
0010: 884701B5 B0FE01F4 10FE1676 61FE4500 .G.....va.E.
0020: 05DC0000 4000FE01 541F0A00 0E020A00 ....@...T.....
0030: 01000800 4800C0B1 0000ABCD ABCDABCD ....H.....
```

Next, we will example the MPLS IP minor options. First, we know that LDP will allocate labels for all IGP prefixes by default. The exception is the default route, which LDP excludes from this rule. The idea is that if there is a default route in an MPLS core, chances are it is better to route traffic to the default destination using IP only in case MPLS forwarding is broken. This only makes sense if there is actually a default-route in the network, and in this case, XRv3 originates one inside IS-IS level-2. We verify that it works and that the information is carried inside the IS-IS LSP.

```
! XRv3
router isis 211
```

```
address-family ipv4 unicast
default-information originate
```

```
R7#show isis database level-2 detail XRv3.00-00 | include 0\.0\.0\.0
Metric: 0          IP 0.0.0.0/0
```

Before continuing, note that if you are using the handy “allocate global host-route” feature to only allocate labels for host routes, you must remove it if you want to also enable label switching for the default route. Below is an alternative configuration that only allocates labels for all hosts routes as well as the default route but nothing else. The second permit statement of this prefix-list does nothing until we actually enable the feature, so for now, XE behaves identically as it would with the “host-route” shortcut. Fortunately, XR doesn’t need this workaround since we can add multiple conditions to label advertisement, so the original configuration works (shown again for completeness).

```
! All XE LSRs
ip prefix-list PL_ALLOCATE_LABELS seq 5 permit 0.0.0.0/0 ge 32
ip prefix-list PL_ALLOCATE_LABELS seq 10 permit 0.0.0.0/0

mpls ldp label
allocate global prefix-list PL_ALLOCATE_LABELS

! All XR LSRs
address-family ipv4
label
local
allocate for host-routes
```

By default, both XE and XR actually do allocate labels for the default-route, but that label is a null label in accordance with LDP policies. That is to say, it is normally implicit-null, but could be explicit-null if LDP is configured as such. A quick look on CSR7 and XRv2 confirm this. In reality, the presence of all these null labels effectively means that label switching is disabled for default-routed traffic.

```
R7#show mpls ldp bindings 0.0.0.0 0
lib entry: 0.0.0.0/0, rev 62
local binding: label: imp-null
remote binding: lsr: 211.0.0.12:0, label: imp-null
remote binding: lsr: 211.0.0.6:0, label: imp-null
remote binding: lsr: 211.0.0.8:0, label: imp-null
remote binding: lsr: 211.0.0.11:0, label: imp-null
```

```
RP/0/0/CPU0:XRv1#show mpls ldp bindings 0.0.0.0/0
0.0.0.0/0, rev 28
Local binding: label: ImpNull
Remote bindings: (3 peers)
Peer          Label
-----
211.0.0.6:0   ImpNull
```

```

211.0.0.7:0      ImpNull
211.0.0.8:0      ImpNull

```

We can instruct LDP to allocate labels for the default route as shown below. The XE syntax is less obvious since this is actually a function of LDP, yet it isn't an LDP-related command. XR cleans this up and puts it in a more logical spot. Since XR can have multiple conditions for label allocation, we can add "default-route" along with "allocate for host-routes" and both will work. XE needed to allow the default route in the prefix-list, and enable the feature explicitly.

```

! All XE LSRs
mpls ip default-route

! All XR LSRs
mpls ldp
  address-family ipv4
    label
      local
        default-route

```

When we run the same commands on CSR7 and XRv1, now we see "real" labels for the default route, just like any other prefix. The benefit of this feature is that default-routed traffic can now be steered into TE-tunnels and protected by TE-FRR.

```

R7#show mpls ldp bindings 0.0.0.0 0
  lib entry: 0.0.0.0/0, rev 64
    local binding: label: 7010
    remote binding: lsr: 211.0.0.11:0, label: 91015
    remote binding: lsr: 211.0.0.8:0, label: 8000
    remote binding: lsr: 211.0.0.6:0, label: 6008
    remote binding: lsr: 211.0.0.12:0, label: 92014

```

```

RP/0/0/CPU0:XRv1#show mpls ldp bindings 0.0.0.0/0
0.0.0.0/0, rev 30
  Local binding: label: 91015
  Remote bindings: (3 peers)
    Peer          Label
    -----
    211.0.0.6:0   6008
    211.0.0.7:0   7010
    211.0.0.8:0   8000

```

One of the most common features that providers configure on their PE routers is TTL-propagation adjustment. Before continuing with these features, I briefly outline MPLS TTL behavior.

1. **Label swap:** The topmost label's TTL is decremented, and this new value is used for the swapped label. This is very similar to normal IP forwarding.

2. **Label push:** The topmost label's TTL is decremented, and this new value is used for the swapped label and any additional pushed labels. This would happen if, for example, a P router was routing traffic into a TE-FRR tunnel and was adding an additional label in the middle of the LSP.
3. **Label pop:** The topmost label's TTL is decremented, and this new value is applied to the inner label that was exposed as a result of the swap. This does not occur if the "new value" from the outer label is greater than the TTL of the inner label. For example, if the inner label has TTL 6 and the outer label has TTL 9, it would not make sense to increase the TTL of the inner packet by setting it to 8 (9 minus 1).

In the vast majority of this book, I use traceroute inside of L3VPNs to verify label stacks and routing paths. It could be highly undesirable/insecure for customers to see the detailed topology information of a provider's network, complete with IP hops and labels, by using a simple traceroute. At the same time, a provider should not prevent a customer from using traceroute to verify L3VPN connectivity between customer sites. By default, when IPv4/v6 packets enter an ingress LSR, their TTL/hop limits are copied onto all labels at imposition. Some other texts indicate that the TTL is only copied to the topmost label, which is false. We will quickly prove the claim that, at imposition, an ingress LSR will copy the IPv4 TTL or IPv6 hop-limit to all imposed labels. We will ping XRv4 to CSR1 again and use EPC inbound and outbound on CSR7 to confirm this. The first two stanzas represent an IPv6 packet and the second two represent an IPv4 packet. The difference in size is 4 bytes (1530 – 1526) and we see one less label in the stack. This is CSR7 performing PHP, but notice the TTLs in the labels is 0x3B, or decimal 59. This is true for all labels. When CSR7 label-switches the packet towards XRv3, it pops the topmost label, decrements TTL on the next label to 58 (0x3A), and forwards the packet. The same is true for the second pair of outputs which represent an IPv4 packet with TTL 254 (0xFE). This is applied to all labels at imposition, and the PHP/TTL reduction process is performed identically regardless of the tunneled protocol. In green, I highlight the original IPv4 TTL and IPv6 hop-limit to illustrate the equality.

R7#show monitor capture CAP buffer detailed

```

 3 1530      1.160987  00:50:56:A9:2D:C6 -> 00:50:56:A9:EA:77 MPLS unicast
0000:  005056A9 EA770050 56A92DC6 81000DF3  .PV..w.PV.-.....
0010:  884701B5 B03B01F4 103B1676 A13B6000  .G...;...;v.;`.
0020:  000005B4 3A3B0000 00000000 00000010  ....;.....
0030:  00000014 00010000 00000000 00000010  .....

 4 1526      1.160987  00:50:56:A9:EA:77 -> 00:50:56:A9:FB:1C MPLS unicast
0000:  005056A9 FB1C0050 56A9EA77 81000DFA  .PV....PV..w....
0010:  884701F4 103A1676 A13B6000 000005B4  .G...:v.;`.....
0020:  3A3B0000 00000000 00000010 00000014  :;.....
0030:  00010000 00000000 00000010 00000001  .....

 6 1530      2.183996  00:50:56:A9:2D:C6 -> 00:50:56:A9:EA:77 MPLS unicast
0000:  005056A9 EA770050 56A92DC6 81000DF3  .PV..w.PV.-.....
0010:  884701B5 B0FE01F4 10FE1676 61FE4500  .G.....va.E.
0020:  05DC0000 4000FE01 541F0A00 0E020A00  ....@...T.....
0030:  01000800 F80010B1 0000ABCD ABCDABCD  .....

```

```

 7 1526      2.183996  00:50:56:A9:EA:77 -> 00:50:56:A9:FB:1C MPLS unicast
0000:  005056A9 FB1C0050 56A9EA77 81000DFA  .PV....PV..w....
0010:  884701F4 10FD1676 61FE4500 05DC0000  .G.....va.E.....
0020:  4000FE01 541F0A00 0E020A00 01000800  @...T.....
0030:  F80010B1 0000ABCD ABCDABCD ABCDABCD  .....

```

By copying the TTL from the IPv4/v6 packets, the TTL could potentially expire in the middle of the MPLS core. For example, a packet with TTL=3 would expire at XRv3 along this LSP, which would generate an ICMP time-exceeded unreachable back to the source. We can disable this feature on XRv1 specifically for forwarded packets; we have the option of specifying “forwarded” or “local”, where “local” means locally generated traffic. It would be handy to allow XRv1 to use traceroute inside the MPLS core, but prevent the customers from doing so, which is why I commonly use the “forwarded” option. This is not a function of LDP or any specific protocol.

```

! XRv1
mpls ip-ttl-propagate disable forwarded

```

To verify it, I will send IPv4 and IPv6 pings from XRv4 to CSR1 again. CSR7 is still capturing inbound from XRv1 and outbound to XRv3 so we can see the difference. Notice that despite the command only specifying “ip”, it applies to all versions of IP. The TTL is fixed at 255 (0xFF) for both IPv4 and IPv6, and for all labels at imposition. The IPv4 TTL and IPv6 hop-limit is highlighted in green, which we clearly see is different; earlier it was the same as the MPLS shim-header TTL.

```

R7#show monitor capture CAP buffer detailed
 4 1530      0.913954  00:50:56:A9:2D:C6 -> 00:50:56:A9:EA:77 MPLS unicast
0000:  005056A9 EA770050 56A92DC6 81000DF3  .PV..w.PV.-.....
0010:  884701B5 B0FF01F4 10FF1676 61FF4500  .G.....va.E.
0020:  05DC0000 4000FE01 541F0A00 0E020A00  ....@...T.....
0030:  01000800 980070B1 0000ABCD ABCDABCD  .....p.....

 5 1526      0.913954  00:50:56:A9:EA:77 -> 00:50:56:A9:FB:1C MPLS unicast
0000:  005056A9 FB1C0050 56A9EA77 81000DFA  .PV....PV..w....
0010:  884701F4 10FE1676 61FF4500 05DC0000  .G.....va.E.....
0020:  4000FE01 541F0A00 0E020A00 01000800  @...T.....
0030:  980070B1 0000ABCD ABCDABCD ABCDABCD  ..p.....

11 1530      4.323988  00:50:56:A9:2D:C6 -> 00:50:56:A9:EA:77 MPLS unicast
0000:  005056A9 EA770050 56A92DC6 81000DF3  .PV..w.PV.-.....
0010:  884701B5 B0FF01F4 10FF1676 A1FF6000  .G.....v..`.
0020:  000005B4 3A3E0000 00000000 00000010  ....:;.....
0030:  00000014 00010000 00000000 00000010  .....

12 1526      4.323988  00:50:56:A9:EA:77 -> 00:50:56:A9:FB:1C MPLS unicast
0000:  005056A9 FB1C0050 56A9EA77 81000DFA  .PV....PV..w....
0010:  884701F4 10FE1676 A1FF6000 000005B4  .G.....v..`.

```

```

0020: 3A3B0000 00000000 00000010 00000014  ;;.....
0030: 00010000 00000000 00000010 00000001  .....

```

The ultimate test is performing a traceroute from the customer network. The entire SP topology is now hidden, with the exception of the ingress LSP hop, egress LSR hop, and corresponding remote VPN label.

```
RP/0/0/CPU0:XRv4#traceroute 10.0.1.0 source 10.0.14.2
```

```
Type escape sequence to abort.
```

```
Tracing the route to 10.0.1.0
```

```

 1  10.11.14.11 9 msec  0 msec  0 msec
 2  211.8.9.12 [MPLS: Label 92006 Exp 0] 0 msec  0 msec  0 msec
 3  10.1.12.1 0 msec  0 msec  0 msec

```

```
RP/0/0/CPU0:XRv4#traceroute ::10:0:1:0 source ::10:0:14:1
```

```
Type escape sequence to abort.
```

```
Tracing the route to ::10:0:1:0
```

```

 1  fd00:10:11:14::11 0 msec 0 msec 0 msec
 2  fd00:10:1:12::12 [MPLS: Label 92010 Exp 0] 0 msec 0 msec 0 msec
 3  fd00:10:1:12::1 0 msec 0 msec 0 msec

```

We demonstrate the benefit of not disabling “local” TTL propagation, because XRv1’s traceroute probe’s can be copied to the MPLS TTL. This means that the core routers can still use traceroute since the traffic was locally originated, so copying the IPv4 TTL or IPv6 hop limit into the MPLS TTL is acceptable.

```
RP/0/0/CPU0:XRv1#traceroute 211.0.0.9 source 211.0.0.11
```

```
Type escape sequence to abort.
```

```
Tracing the route to 211.0.0.9
```

```

 1  211.6.11.6 [MPLS: Label 6000 Exp 0] 0 msec  0 msec  0 msec
 2  211.6.13.13 [MPLS: Label 93002 Exp 0] 0 msec  0 msec  0 msec
 3  211.9.13.9 0 msec  0 msec  0 msec

```

If we also disable local TTL propagation on XRv1, traffic is tunneled inside MPLS all the way to the target, and the traceroute is less valuable inside the provider core network.

```
! XRv1
```

```
mpls ip-ttl-propagate disable local
```

```
RP/0/0/CPU0:XRv1#traceroute 211.0.0.9 source 211.0.0.11
```

```
Type escape sequence to abort.
```

```
Tracing the route to 211.0.0.9
```

```

 1  211.9.13.9 9 msec  0 msec  0 msec

```

When CSR5 traceroutes to CSR1, the MPLS core is revealed since CSR6 is still copying the customer IP TTL into the MPLS TTL.

```
R5#traceroute 10.0.1.3 source 10.0.5.0
Type escape sequence to abort.
Tracing the route to 10.0.1.3
VRF info: (vrf in name/id, vrf out name/id)
 1 10.5.6.6 6 msec 3 msec 3 msec
 2 211.6.7.7 [MPLS: Labels 7001/92009 Exp 0] 7 msec 7 msec 7 msec
 3 211.7.12.12 [MPLS: Label 92009 Exp 0] 6 msec 11 msec 20 msec
 4 10.1.12.1 19 msec 11 msec 10 msec
```

The command is similar on XE, and we will disable “forwarded” TTL propagation on CSR6 to prevent this. We have the same “forwarded” and “local” options as XR. When the customer devices attempt to traceroute, only the edge LSRs are revealed, along with the VPN label. Note: When TTL propagation is disabled on P-routers, the TTL reduction of the topmost label is not propagated to inner labels. This can effect traceroutes inside the core as it changes the way MPLS handles TTL decrementing.

```
! CSR6
no mpls ip propagate-ttl forwarded
```

```
R5#traceroute 10.0.1.3 source 10.0.5.0
Type escape sequence to abort.
Tracing the route to 10.0.1.3
VRF info: (vrf in name/id, vrf out name/id)
 1 10.5.6.6 5 msec 3 msec 3 msec
 2 211.7.12.12 [MPLS: Label 92009 Exp 0] 5 msec 6 msec 5 msec
 3 10.1.12.1 5 msec 8 msec 10 msec
```

Like XR, we can also disable “local” TTL propagation. Currently, CSR6 can traceroute through the network.

```
R6#traceroute 211.0.0.12 source 211.0.0.6
Type escape sequence to abort.
Tracing the route to 211.0.0.12
VRF info: (vrf in name/id, vrf out name/id)
 1 211.6.7.7 [MPLS: Label 7001 Exp 0] 3 msec 3 msec 3 msec
 2 211.7.12.12 4 msec 3 msec 3 msec
```

When we disable all TTL propagation, this applies to both locally-generated and forwarded traffic, so CSR6 can no longer see the SP network topology via traceroute. The probes are tunneled inside MPLS, so only the first probe reaches the target and an unreachable is returned.

```
! CSR6
no mpls ip propagate-ttl
```



```

R6#traceroute 211.0.0.12 source 211.0.0.6
Type escape sequence to abort.
Tracing the route to 211.0.0.12
VRF info: (vrf in name/id, vrf out name/id)
 1 211.7.12.12 5 msec 3 msec 3 msec

```

From a design perspective, one would normally disable TTL propagation for forwarded traffic on all PE devices that offer L3VPN service. In this design, I leave TTL propagation enabled on XRv2 and CSR9 for variety, though it doesn't make much sense. TTL propagation should be disabled on ingress and egress LSRs; if it isn't, the IP TTL may theoretically exit the network at a larger value than it entered. Cisco safeguards against this, must like it does with the label pop TTL handling, and the MPLS TTL is not copied to the IP TTL if it is greater than the IP TTL. CSR1 can still traceroute across the network and see the SP topology since CSR6 is copying the customer TTL into the MPLS TTL of all labels on imposition.

```

R1#traceroute 10.0.14.1 source 10.0.1.0
Type escape sequence to abort.
Tracing the route to 10.0.14.1
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.9.9 5 msec 4 msec 3 msec
 2 211.9.13.13 [MPLS: Labels 93001/91011 Exp 0] 7 msec 8 msec 8 msec
 3 211.7.8.7 [MPLS: Labels 7000/91011 Exp 0] 8 msec 8 msec 28 msec
 4 211.7.11.11 [MPLS: Label 91011 Exp 0] 20 msec 21 msec 21 msec
 5 10.11.14.14 20 msec 15 msec 14 msec

```

The last MPLS IP-related option is TTL expiration label handling. To understand this command, we first have to understand how traceroute works over MPLS. When an LSP receives an MPLS packet with TTL=1 and the destination is not local, this is considered a time-exceeded event just like with IPv4 or IPv6. We can see this happening on CSR7 if we traceroute from CSR1 to XRv4. We will send a single traceroute probe to reduce the debug output. The debug clearly shows the original destination; because CSR7 is a P router with no context for these VPN addresses, it has no choice but to add the original label stack of {91011} to the ICMP unreachable and send it towards the destination.

```

R7#debug ip icmp
ICMP packet debugging is on

R1#traceroute 10.0.14.1 source 10.0.1.0 probe 1
Type escape sequence to abort.
Tracing the route to 10.0.14.1
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.9.9 5 msec
 2 211.9.13.13 [MPLS: Labels 93001/91011 Exp 0] 8 msec
 3 211.7.8.7 [MPLS: Labels 7000/91011 Exp 0] 10 msec
 4 211.7.11.11 [MPLS: Label 91011 Exp 0] 9 msec
 5 10.11.14.14 7 msec

```

! CSR7

```
MPLS: ICMP: time exceeded (time to live) sent to 10.0.1.0 (dest was 10.0.14.1)
```

We use EPC on CSR7 outbound towards XRv1 to confirm the actual packet contents. This is an ICMP unreachable message; I assume this because the protocol number is 1, the TTL is 255, and I was not sending any pings in the network during this capture. These fields are highlighted in yellow for clarity. The MPLS label is 91011 and is highlighted in green, which is the full label stack that CSR7 would have used along the original LSP when sending traffic to 10.0.14.1 via XRv1. The ICMP packet encompasses the original traceroute probe with TTL=2 and protocol 17 (grey) with original addresses 10.0.1.0 to 10.0.14.1 (cyan). The magenta addresses are in the ICMP unreachable packet itself, which is 211.7.8.7 to 10.0.1.0; this is a little awkward since the addresses are in two different tables, but is important.

```
R7#show monitor capture CAP buffer dump
0000: 005056A9 2DC60050 56A9EA77 81000DF3 .PV.-..PV..w....
0010: 88471638 3DFF45C0 00ACDE24 0000FF01 .G.8=.E....$.
0020: F65DD307 08070A00 01000B00 181A0000 .].....
0030: 00004500 001C7C56 00000211 197B0A00 ..E...|V.....{..
0040: 01000A00 0E01C15D 829C0008 98E30000 .....].....
```

When XRv1 receives this packet, it will perform an LFIB lookup and forward the packet to the CE, which is XRv4. XRv4 reverses the source and destination addresses and forwards the packet back into the MPLS network, which makes its way back to CSR1. The pink addresses above are how CSR1 can see that 211.7.8.7 was a hop in the carrier's network since it was carried inside the ICMP unreachable. Thus, if the CE-to-CE connectivity is broken in any way, traceroute will not work for the customer. Even if XRv4's interface was shutdown, CSR1 will get no feedback. We test this quickly, and it is valuable to note that there is no workaround for this within L3VPN. The LSP must be functional end-to-end, including CE routers, for this kind of traceroute to work.

```
R1#traceroute 10.0.14.1 source 10.0.1.0 probe 1
Type escape sequence to abort.
Tracing the route to 10.0.14.1
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.9.9 6 msec
 2 * [snip]
```

However, addressing within the provider core is all in the same routing table. If, for example, there was a broken LSP in the SP core, the core routers could still communicate using IP. I temporarily disable the link between CSR7 and CSR6 for this example so the LSP is longer (not shown). A traceroute on XRv2 to CSR6 shows us that the network has MPLS transport.

```
RP/0/0/CPU0:XRv2#traceroute 211.0.0.6 source 211.0.0.12
Type escape sequence to abort.
Tracing the route to 211.0.0.6

 1 211.7.12.7 [MPLS: Label 7012 Exp 0] 0 msec 0 msec 0 msec
```

```

2  211.7.8.13 [MPLS: Label 93005] Exp 0] 0 msec  0 msec  0 msec
3  211.6.13.6 0 msec  0 msec  0 msec

```

However, the MPLS process is generally ignorant to the tunneled payload and treats this traceroute just like an L3VPN traceroute. When TTL=1 packets hit CSR7, CSR7 encapsulates the unreachable with the original label stack {93005} (in yellow) and sends the packet towards XRv3. The original source/destination addresses are shown in cyan, which is 211.0.0.12 to 211.0.0.6. In pink, I highlight the TTL=1 and protocol UDP (0x11 or decimal 17) which was the original probe. There isn't a really good reason to send this packet along the original LSP since CSR7 actually does know how to reach 211.0.0.6, the original source.

```
R7#show monitor capture CAP buffer dump
```

```

1
0000:  005056A9 EA540050 56A9EA77 81000DFA  .PV..T.PV..w....
0010:  884716B4 DDEF45C0 00A8E8C8 0000FF01  .G....E.....
0020:  1FB1D307 0C07D300 000C0B00 9B2C0000  .....
0030:  00004500 001CABB6 00000111 6808D300  ..E.....h...
0040:  000CD300 0006ABB6 829B0008 2B790000  .....+y..

```

We can set a label depth threshold on the routers which allow it to make more intelligent decisions regarding TTL expiration events. If an MPLS packet arrives with less than or equal to the number of labels we specify, the router will use the global routing table for a route lookup. If a packet arrives with more than the number of labels specified, the original label stack is used. By default, the threshold is 0, which means all ICMP unreachables are tunneled to their final destination if they were MPLS encapsulated. By increasing this threshold to 1, we instruct the router to treat things differently for singly-labeled packets. Since XRv2 and CSR6 have only one label at any point in the stack along the path, this is an appropriate value. Other realistic values might be higher if there is a lot of PE-P/P-P TE in the network, CSC or UMPLS, etc. In our case, 1 label generally means global routing table, while 2 or more means L3VPN.

```

! CSR7
mpls ip ttl-expiration pop 1

```

Using traceroute with debugging enabled on CSR7, we can confirm this new behavior. Because the in-label stack was less or equal to the threshold we set, the label stack is removed; the debug reveals this clearly. XRv3 is no longer receiving these packets.

```

! CSR7
MPLS: ICMP: time exceeded (time to live) sent to 211.0.0.12 (dest was
211.0.0.6)
Pop labels: num_in_labels (1) <= ttl_exp_labels (1)

```

The command is nearly identical on XR, and we enable on it XRv3 for completeness. There isn't a good way to verify this with debugs on XR, but it works identically as it does on XE. Also, the link between CSR6 and CSR7 is restored before ending this lab.

```
! XRv3
mpls ip-ttl-expiration-pop 1
```

Additional Reading – Reference configurations “mpls-ip mtu”

8. Describe MPLS advanced features

8.1 Segment Routing

Segment Routing (SR) is a relatively new technology pioneered by Cisco that is meant to reduce state in MPLS core networks. One can use SR to replace LDP and RSVP-TE wholesale provided it is supported. The idea is that individual nodes and adjacencies have segment IDs (SIDs), and each segment has label bindings. This allows traffic to traverse the network encapsulated inside MPLS and individual links can be elected by the headend by way of specifying specific segment labels. There are other SIDs, also, but those are the main two. Right now it is only supported in XR and only for IS-IS IPv4. A SR mapping server can be used for LDP/SR interworking during migrations or pilot scenarios, which is demonstrated later.

You cannot configure prefix-sids on transit links at this time. Support for this feature may be introduced in later code versions.

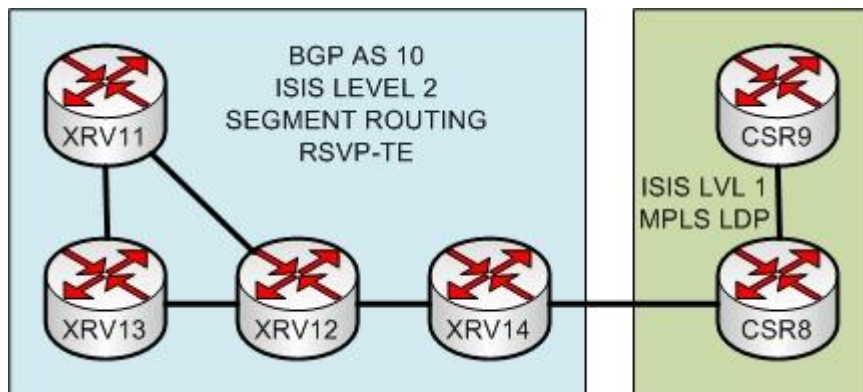
```
! XRv11
router isis 1
 interface GigabitEthernet0/0/0/0.512
   address-family ipv4 unicast
   prefix-sid index 512
```

!!% Not supported (Success): Nodal Segment configuration is only allowed for Loopback Interfaces

The SRGB (segment routing global block) must not overlap with the global MPLS label range allocation. The SRGB has a specific purpose which is explicitly different than the global MPLS label range and is discussed more later.

```
! XRv11
RP/0/0/CPU0: isis[1006]: %ROUTING-ISIS-4-SRGB_ALLOC_FAIL : SRGB allocation
failed: 'SRGB reservation not successful for [91000,91999], srgb=(91000
91999, SRGB_ALLOC_CONFIG_PENDING, 0x1) (So far 16 attempts). Make sure label
range is free'
```

A basic network diagram is shown below. All XR routers are SR-aware and are also configured for RSVP-TE. The XE routers are LDP-aware; XRv14 will perform the SR-to-LDP interworking.



The ISIS database will show all of the SR information. Some information is harder to see than others, and XRV11's LSP is shown below. Notice that the adjacency SID's are allocated from the global MPLS range, not the SRGB, and are placed in a simple table. The SRGB is designed for node SIDs only while the global MPLS label range supports the adjacency SIDs. The node SID is a little harder to decode; the "Prefix-SID Index" assigned to the router's loopback 11.11.11.11/32 is statically configured under the interface within IS-IS. It can be absolute as a label within the SRGB (like 81011) or a relative index value to be added to the SRGB lower bound (like 11). Every other router, when allocating a label for 11.11.11.11/32, will take its locally configured SRGB lower-bound number and add it to the destination index in question. For example, if XRV13's SRGB is 83000 - 83999 and traffic is destined for XRV11's loopback, XRV13 will allocate a label value of 83011 (83000 + 11) for the prefix 11.11.11.11/32. This process is repeated until labels are allocated for all other routers for which SR is enabled. The network diagram is shown below, along with the initial verifications.

```
RP/0/0/CPU0:XRv14#show isis database verbose XRv11.00-00
IS-IS 1 (Level-2) Link State Database
LSPID                LSP Seq Num  LSP Checksum  LSP Holdtime  ATT/P/OL
XRv11.00-00          0x0000000d   0xd5de        888           0/0/0
  Area Address: 00
  NLPID:         0xcc
  Hostname:      XRv11
  IP Address:    11.11.11.11
  Router Cap:    11.11.11.11, D:0, S:0
    Segment Routing: I:1 V:0, SRGB Base: 81000 Range: 1000
  Metric: 10      IS-Extended XRv12.01
  Metric: 10      IS-Extended XRv12.01
  LAN-ADJ-SID: F:0 B:0 V:1 L:1 S:0 weight:0
-----
| Hostname                | Adjacency Sid |
|-----|-----|
| XRv12                    | 91002         |
|-----|-----|
  Metric: 10      IS-Extended XRv13.03
  Metric: 10      IS-Extended XRv13.03
  LAN-ADJ-SID: F:0 B:0 V:1 L:1 S:0 weight:0
```

```

-----
| Hostname                               | Adjacency Sid |
|-----|-----|
| XRv13                                | 91005         |
|-----|-----|
Metric: 0                               IP-Extended 11.11.11.11/32
Prefix-SID Index: 11, R:0 N:1 P:0 E:0 V:0 L:0
Metric: 10                               IP-Extended 12.0.0.0/24
Metric: 10                               IP-Extended 13.0.0.0/24

```

The SRGB ranges do not need to be unique across routers, and in many cases should not be. This guarantees that the same label value can be used between pairs of nodes, since everyone adds the prefix-SID index with the SRGB lower bound. Here are two quick examples where XRv12 was configured with the same SRGB as XRv11 (81000 - 81999). Notice the same label value is used for each hop. The routers along the path just perform a swap for the same label value; it is possible there is some hardware optimization to perform no operation at all but this is beyond the scope of our analysis. For troubleshooting, having unique label values with meaningful numbers per node is desirable. This is probably not supportable in a large-scale network, but good for learning and lab use.

```
RP/0/0/CPU0:XRv14#traceroute 13.13.13.13
```

```
Type escape sequence to abort.
```

```
Tracing the route to 13.13.13.13
```

```

 1  24.0.0.12 [MPLS: Label 81013 Exp 0] 19 msec  29 msec  19 msec
 2  12.0.0.11 [MPLS: Label 81013 Exp 0] 19 msec  19 msec  19 msec
 3  13.0.0.13 19 msec  *   39 msec

```

```
RP/0/0/CPU0:XRv13#traceroute 14.14.14.14
```

```
Type escape sequence to abort.
```

```
Tracing the route to 14.14.14.14
```

```

 1  13.0.0.11 [MPLS: Label 81014 Exp 0] 109 msec  39 msec  29 msec
 2  12.0.0.12 [MPLS: Label 81014 Exp 0] 19 msec  29 msec  19 msec
 3  24.0.0.14 29 msec  *   29 msec

```

The prefix-SIDs must be unique based on the way labels are allocated. In this example, XRv14's index has been set to 11, the same as XRv11. Nothing looks wrong from XRv14's perspective, as it has a label for 11.11.11.11/32 via XRv12.

```
RP/0/0/CPU0:XRv14#show cef ipv4 11.11.11.11
```

```
[snip]
```

```

  via 24.0.0.12, GigabitEthernet0/0/0/0.524, 11 dependencies, weight 0,
 class 0 [flags 0x0]

```

```
  path-idx 0 NHID 0x0 [0xa0e87154 0x0]
```

```
  next hop 24.0.0.12
```

```
  local adjacency
```

```
local label 94004      labels imposed {82011}
```

When XRv12 receives traffic with label 82011, PHP is performed, and XRv11 receives the traffic.

```
P/0/0/CPU0:XRv12#sh mpls for labels 82011
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
82011	Pop	No ID	Gi0/0/0/0.512	12.0.0.11	3636

The reverse LSP is broken since XRv12 cannot program the same label value out of multiple interfaces at the same time, so XRv11 has no label to reach XRv14. Normally the label would have to be 82011, which is the same as what was used in the opposite direction. XRv12 allocated the correct label for XRv11 first, so the incorrect label for XRv14 (which should have been 82014) was not allocated.

```
RP/0/0/CPU0:XRv11#show cef ipv4 14.14.14.14
```

```
[snip]
```

```
via 12.0.0.12, GigabitEthernet0/0/0/0.512, 13 dependencies, weight 0,
class 0 [flags 0x0]
  path-idx 0 NHID 0x0 [0xa0fec2a4 0x0]
  next hop 12.0.0.12
  local adjacency
    local label 91000      labels imposed {None}
```

As soon as the discrepancy is fixed (XRv14's prefix-sid for loopback0 is set back to 14), the LSP is operational.

```
RP/0/0/CPU0:XRv11#show cef ipv4 14.14.14.14
```

```
[snip]
```

```
via 12.0.0.12, GigabitEthernet0/0/0/0.512, 11 dependencies, weight 0,
class 0 [flags 0x0]
  path-idx 0 NHID 0x0 [0xa0fec2f8 0x0]
  next hop 12.0.0.12
  local adjacency
    local label 91000      labels imposed {82014}
```

RSVP-TE can coexist with SR. In a basic configuration as we have currently, SR has replaced LDP but has not provided any TE capability (called SR-TE). If an RSVP-TE tunnel is built between any pair of routers that does not terminate on the remote PE, then LDP is typically enabled on the tunnel to create a targeted session between the head and tail ends. This is to exchange a label binding for the remote PE's loopback, which ensures the bottom label is not exposed to core routers too early. In this example, we build a basic RSVP-TE tunnel using an explicit path from XRv11 to XRv12 via XRv13. The IGP costs would normally not route this way, which necessitates the use of a TE tunnel.

```
! XRv11
```

```

explicit-path name XRV13
  index 10 next-address strict ipv4 unicast 13.13.13.13
  index 20 next-address strict ipv4 unicast 12.12.12.12

interface tunnel-te18
  ipv4 unnumbered Loopback0
  destination 12.12.12.12
  autoroute announce
  path-option 5 explicit name XRV13

```

XRv12 is the tail end and allocates implicit-null towards XRv13, telling it to perform PHP. This should reveal the SR label for XRv14's loopback, which was learned via IS-IS.

```

RP/0/0/CPU0:XRv12#show mpls traffic-eng tunnels
LSP Tunnel 11.11.11.11 18 [2] is signalled, Signaling State: up
  Tunnel Name: XRv11_t18 Tunnel Role: Tail
  InLabel: GigabitEthernet0/0/0/0.523, implicit-null
  [snip]

```

XRv13 is the midpoint and performs PHP. It allocates label 93003 towards the head end, which is XRv11.

```

RP/0/0/CPU0:XRv13#show mpls traffic-eng tunnels
LSP Tunnel 11.11.11.11 18 [2] is signalled, Signaling State: up
  Tunnel Name: XRv11_t18 Tunnel Role: Mid
  InLabel: GigabitEthernet0/0/0/0.513, 93003
  OutLabel: GigabitEthernet0/0/0/0.523, implicit-null

```

XRv11 is the head and pushes 93003 as the top label in the stack, assuming 14.14.14.14/32 is reachable via the TE tunnel. Notice that the "detail" keyword must be used with the show command to reveal the label on the head end (without looking directly at the RSVP RESV messages).

```

RP/0/0/CPU0:XRv11#sh mpls traffic-eng tunnels detail | begin Current LSP Info
Current LSP Info:
  Instance: 2, Signaling Area: IS-IS 1 level-2
  Uptime: 15:01:40 (since [snip])
  Outgoing Interface: GigabitEthernet0/0/0/0.513, Outgoing Label: 93003

```

Auto-route announce is configured on the tunnel, so the router is gleaned via IS-IS despite the LSPDB not revealing an adjacency.

```

RP/0/0/CPU0:XRv11#show route ipv4 unicast 14.14.14.14/32
Routing entry for 14.14.14.14/32
  Known via "isis 1", distance 115, metric 20, type level-2
  Routing Descriptor Blocks
    12.12.12.12, from 14.14.14.14, via tunnel-te18
    Route metric is 20
  No advertising protos.

```


The FIB shows the SR label for XRv14's loopback, which is the sum of XRv14's prefix SID for its loopback0 and the SRGB lower-bound of XRv12 (tunnel tail). This is the equivalent of an LDP label learned via the TE-tunnel, advertised by XRv12 to represent its label for XRv14's loopback.

```
RP/0/0/CPU0:XRv11#show cef ipv4 14.14.14.14/32
[snip]
  via 12.12.12.12, tunnel-te18, 9 dependencies, weight 0, class 0 [flags
0x0]
    path-idx 0 NHID 0x0 [0xa0fec544 0x0]
    next hop 12.12.12.12
    local adjacency
      local label 91000      labels imposed {82014}
```

When XRv11 sends traffic to XRv14, the SR label is pushed first (to get from XRv12 to XRv14), followed by the TE label (gets from XRv11 to XRv12).

```
RP/0/0/CPU0:XRv11#traceroute 14.14.14.14
Type escape sequence to abort.
Tracing the route to 14.14.14.14

 1  13.0.0.13 [MPLS: Labels 93003/82014 Exp 0] 29 msec  29 msec  29 msec
 2  23.0.0.12 [MPLS: Label 82014 Exp 0] 29 msec  29 msec  29 msec
 3  24.0.0.14 29 msec  *   19 msec
```

We can quickly test VPNv4 traffic also. We expect the label stack to remain the same except now the first label pushed (bottom-most) is the VPNv4 label allocated by XRv14 for its VPN route. The VPNv4 BGP topology is not discussed in detail.

```
RP/0/0/CPU0:XRv11#show bgp vpnv4 unicast vrf A 100.14.14.14/32
BGP routing table entry for 100.14.14.14/32, Route Distinguisher: 1:1
[snip]
 14.14.14.14 (metric 20) from 12.12.12.12 (14.14.14.14)
   Received Label 94007
   Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, imported
   Received Path ID 0, Local Path ID 1, version 26
   Extended community: RT:1:1
   Originator: 14.14.14.14, Cluster list: 12.12.12.12
   Source VRF: A, Source Route Distinguisher: 1:1
```

The bottom-most label is 94007 now, with the TE and SR labels remaining in the same sequence. In summary, SR has replaced LDP, and simplifies basic RSVP-TE configuration for TE tunnels not directly configured PE to PE. SR-TE was not made available until XR version 5.3.1, which was released after this was tested and documented.

```
RP/0/0/CPU0:XRv11#traceroute vrf A 100.14.14.14
Type escape sequence to abort.
Tracing the route to 100.14.14.14
```

```
 1  13.0.0.13 [MPLS: Labels 93003/82014/94007 Exp 0] 39 msec  39 msec  49
msec
 2  23.0.0.12 [MPLS: Labels 82014/94007 Exp 0] 29 msec  59 msec 129 msec
 3  24.0.0.14 89 msec  *   29 msec
```

When the route is learned via the TE tunnel but not via IS-IS (such as using a static route), the SR label cannot be used. Because SR is heavily synchronized with a specific IGP (not generic like LDP), routes from that exact IGP must be used when using TE.

```
RP/0/0/CPU0:XRv11#show route ipv4 unicast 14.14.14.14/32
Routing entry for 14.14.14.14/32
  Known via "static", distance 1, metric 0 (connected)
  Routing Descriptor Blocks
    directly connected, via tunnel-te18
      Route metric is 0
  No advertising protos.
```

```
RP/0/0/CPU0:XRv11#show cef ipv4 14.14.14.14/32
[snip]
  via tunnel-te18, 3 dependencies, weight 0, class 0 [flags 0x8]
    path-idx 0 NHID 0x0 [0xa0fec49c 0xa0fec544]
    local adjacency
      local label 91000      labels imposed {ImplNull}
```

As expected, XRv13 performs PHP of the RSVP-TE label, which will reveal either the raw IP packet or the VPNv4 label to XRv13. VPN connectivity is now broken. This could be considered a limitation of replacing LDP with SR as "autoroute destination" or static routing into a TE tunnel may not always work (autoroute destination is not supported on XRv 5.3.0, but works in XE). I theorize that the reason the CEF entry says implicit-null versus "None" is because the static route references the TE tunnel directly, making it appear attached. The lack of an MPLS at the second entry indicates a broken transport LSP.

```
RP/0/0/CPU0:XRv11#traceroute 14.14.14.14
Type escape sequence to abort.
Tracing the route to 14.14.14.14
```

```
 1  13.0.0.13 [MPLS: Label 93003 Exp 0] 19 msec  19 msec  59 msec
 2  23.0.0.12 79 msec  29 msec  69 msec
 3  24.0.0.14 19 msec  *   19 msec
```

```
RP/0/0/CPU0:XRv11#traceroute vrf A 100.14.14.14
Type escape sequence to abort.
Tracing the route to 100.14.14.14
```

```
1 * * *
2 [snip]
```

TE forwarding adjacency is an alternative to static routing and autoroute which creates a link in the LSPDB for IS-IS or LSDB for OSPF. It requires TE tunnels on both sides, so a TE tunnel must be added to XRv12 back to XRv11. The paths do not need to be symmetric, though. The IS-IS metric on the tunnel is reduced to 5 so that it is the preferred path between XRv11 and XRv12. This feature is documented in the TE section but is demonstrated here to test SR specifically. Only the tunnel on XRv11 is shown.

```
! XRv11
interface tunnel-te18
 forwarding-adjacency

router isis 1
 interface tunnel-te18
  address-family ipv4 unicast
  metric 5
```

```
RP/0/0/CPU0:XRv11#show isis topology systemid XRv12
IS-IS 1 paths to IPv4 Unicast (Level-2) routers
System Id      Metric  Next-Hop      Interface      SNPA
XRv12          5       XRv12         tt18           *PtoP*
```

```
RP/0/0/CPU0:XRv12#show isis topology systemid XRv11
IS-IS 1 paths to IPv4 Unicast (Level-2) routers
System Id      Metric  Next-Hop      Interface      SNPA
XRv11          5       XRv11         tt18           *PtoP*
```

As expected, VPN traffic works again because the route is learned via IS-IS. As long as the route is IS-IS (forwarding-adjacency or autoroute announce), RSVP-TE with SR will work.

```
RP/0/0/CPU0:XRv11#show route ipv4 unicast 14.14.14.14
Routing entry for 14.14.14.14/32
  Known via "isis 1", distance 115, metric 15, type level-2
  Routing Descriptor Blocks
    12.12.12.12, from 14.14.14.14, via tunnel-te18
    Route metric is 15
  No advertising protos.
```

```
RP/0/0/CPU0:XRv11#show cef ipv4 14.14.14.14/32
[snip]
  via 12.12.12.12, tunnel-te18, 9 dependencies, weight 0, class 0 [flags
0x0]
    path-idx 0 NHID 0x0 [0xa0fec544 0x0]
    next hop 12.12.12.12
    local adjacency
      local label 91000      labels imposed {82014}
```

```
RP/0/0/CPU0:XRv11#traceroute vrf A 100.14.14.14
```

```
Type escape sequence to abort.
```

```
Tracing the route to 100.14.14.14
```

```
 1 13.0.0.13 [MPLS: Labels 93003/82014/94007 Exp 0] 39 msec 49 msec 39 msec
 2 23.0.0.12 [MPLS: Labels 82014/94007 Exp 0] 39 msec 49 msec 69 msec
 3 24.0.0.14 39 msec * 39 msec
```

SR can also use a mapping server to allocate prefix-SIDs. In an SR deployment without a mapping server, prefix-SIDs are locally assigned based on the SRGB by each LSR, much like any other dynamic label allocation. Mappings cannot be shared between IS-IS processes and are not VRF aware (currently). The SR mapping server (SRMS) configuration is straightforward. A range of prefixes is configured starting with the first entry, followed by a starting SID value, followed by a range value. To be clear, the current SRMS support is for prefix-sid only; adjacency-sid is still allocated by each device (using the global MPLS label range) per IS-IS link and not per prefix. A common use case of the SRMS is SR/LDP interworking, where not all routers support SR or a migration is occurring. CSR8 and CSR9 are running LDP in IS-IS L1 behind XRv14, which is the L1/L2 router. So far, these CSRs have not participated in the demonstration. All L2 loopbacks are leaked into L1 and no default route (via AT-bit) exists as all routers share an IS-IS area. Routers in the SR domain still need to label-switch traffic to CSR9. Because XRv14 is running LDP with CSR8, the LSP from CSR9 to XRv11 works fine, but the opposite way does not.

```
CSR9#traceroute 11.11.11.11 source 9.9.9.9
```

```
Type escape sequence to abort.
```

```
Tracing the route to 11.11.11.11
```

```
VRF info: (vrf in name/id, vrf out name/id)
```

```
 1 89.0.0.8 [MPLS: Label 8000 Exp 0] 32 msec 25 msec 25 msec
 2 48.0.0.14 [MPLS: Label 94004 Exp 0] 25 msec 25 msec 25 msec
 3 24.0.0.12 [MPLS: Label 82011 Exp 0] 25 msec 25 msec 25 msec
 4 12.0.0.11 25 msec * 24 msec
```

```
RP/0/0/CPU0:XRv11#traceroute 9.9.9.9 source 11.11.11.11
```

```
Type escape sequence to abort.
```

```
Tracing the route to 9.9.9.9
```

```
 1 12.0.0.12 9 msec 0 msec 0 msec
 2 24.0.0.14 19 msec 39 msec 29 msec
 3 48.0.0.8 [MPLS: Label 8005 Exp 0] 29 msec 29 msec 39 msec
 4 89.0.0.9 29 msec * 19 msec
```

We can configure SRMS on any of the routers, even one that is out of band (kind of like PfR master controllers). Consider this example where XRv11 is the SRMS. Normally if your addresses are contiguous, you can use larger numbers with the "range" keyword to allocate prefix-sid values in bulk fashion rather than by individual prefix, as I did. I wanted to use easy numbers for demonstration. In this case, all

routers now use the indices of 88 and 99 to represent 8.8.8.8/32 and 9.9.9.9/32, respectively, and generate MPLS labels by adding this to their SRGB lower bound. A better example would be that all your loopbacks are /32s in the range of 10.10.10.0/24. You could use the “range 256” modifier to cover the range of 0 – 255, covering prefixes 10.10.10.0/32, 10.10.10.1/32 ... 10.10.10.255/32.

```
! XRv11
segment-routing
mapping-server
prefix-sid-map
address-family ipv4
  8.8.8.8/32 88 range 1
  9.9.9.9/32 99 range 1
```

A key component to SRMS working is configuring ISIS to advertise this information. Without this command on the SRMS, the local LSP will not contain the SID mappings.

```
! XRv11
router isis 1
address-family ipv4 unicast
segment-routing prefix-sid-map advertise-local
```

Likewise on all of the clients, they all must honor these markings. This also must be configured on XRv11 or else it cannot label switch to destinations identified in the SRMS (even though it is the same device).

```
! All SR routers
router isis 1
address-family ipv4 unicast
segment-routing prefix-sid-map receive
```

The mapping server adds these SID bindings to its IS-IS LSP via the prefix-sid Sub-TLV. Everyone else in IS-IS level 2 can see this, which encompasses the entire SR domain. Rather than advertise thousands of SIDs, it takes the mapping values (prefix, start index, and range) and advertises those only. Each router independently can calculate the proper prefix-SIDs based on this, assuming they have the routes. This is why loopback addresses allocated in contiguous blocks improves SR scalability when interworking with LDP.

```
RP/0/0/CPU0:XRv11#show isis database verbose XRv11.00-00 | begin SID Binding
SID Binding:  8.8.8.8/32 F:0 M:0 Weight:0 Range:1
SID: Start:88, R:0 N:0 P:0 E:0 V:0 L:0
SID Binding:  9.9.9.9/32 F:0 M:0 Weight:0 Range:1
SID: Start:99, R:0 N:0 P:0 E:0 V:0 L:0
```

Let's manually trace the LSP from XRv11 to CSR9. First, XRv11 should be using a label allocated by XRv12, since IS-IS routes us that way. 82099 is the SR label generated by XRv12 (82000 + 99, where 82000 is the SRGB lower bound for XRv12 and 99 is the SRMS' manual index for the prefix 9.9.9.9/32).

```
RP/0/0/CPU0:XRv11#show route ipv4 9.9.9.9/32
Routing entry for 9.9.9.9/32
  Known via "isis 1", distance 115, metric 40, type level-2
  Routing Descriptor Blocks
    12.0.0.12, from 14.14.14.14, via GigabitEthernet0/0/0/0.512
    Route metric is 40
  No advertising protos.
```

```
RP/0/0/CPU0:XRv11#show mpls forwarding prefix 9.9.9.9/32
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
91012	82099	9.9.9.9/32	Gi0/0/0/0.512	12.0.0.12	768

XRv12 performs a swap to the SR label allocated by XRv14, which is 84099 (84000 + 99). So far, this is basic SR label switching.

```
RP/0/0/CPU0:XRv12#show mpls forwarding labels 82099
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
82099	84099	No ID	Gi0/0/0/0.524	24.0.0.14	2790

Next, XRv14 performs a swap to the LDP label allocated by CSR8. We can see this is an LDP label by verifying the LDP bindings. XRv14 is the point of SR/LDP interworking in this design.

```
RP/0/0/CPU0:XRv14#show mpls forwarding labels 84099
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
84099	8005	No ID	Gi0/0/0/0.548	48.0.0.8	4032

```
RP/0/0/CPU0:XRv14#show mpls ldp bindings 9.9.9.9/32
9.9.9.9/32, rev 26
  Local binding: label: 94011
  Remote bindings: (1 peers)
    Peer          Label
    -----
    8.8.8.8:0     8005
```

Next, CSR8 performs PHP and forwards the traffic to CSR9. This is the end of the LSP.

```
CSR8#show mpls forwarding-table labels 8005
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
8005	Pop Label	9.9.9.9/32	5802	Gi2.589	89.0.0.9

Finally, we perform a trace route from XRv11 to verify the label stack at each hop.

```
RP/0/0/CPU0:XRv11#traceroute 9.9.9.9 source 11.11.11.11
Type escape sequence to abort.
Tracing the route to 9.9.9.9
```

```
 1  12.0.0.12 [MPLS: Label 82099 Exp 0] 49 msec  39 msec  39 msec
 2  24.0.0.14 [MPLS: Label 84099 Exp 0] 119 msec  39 msec  49 msec
 3  48.0.0.8  [MPLS: Label 8005 Exp 0]  59 msec  139 msec 109 msec
 4  89.0.0.9  79 msec   *    49 msec
```

SR has some minor options related to the prefix-SID that are worth mentioning. When enabling “prefix-sid” under a loopback, you can specify how to treat the node flag (n-flag) and explicit null e-flag). Explicit null is used for long pipe and uniform QoS models and is documented elsewhere in this book. LDP, BGP, and RSVP-TE all support explicit-null for this purpose as well. The N-flag is specific to SR and, if set, identifies the router itself. It's normally set on loopback interfaces and is meant to differentiate nodes from links. The mapping server seems to clear all flags and does not give options to set any (at this time), so adjusting these options with LDP interworking appears limited. The P-flag stands for “no-PHP” flag and does not appear directly configurable, and seems a little redundant since if exp-null is set, then no-PHP should also be set. If exp-null is clear, no-PHP should be clear. The output below shows a second loopback configured on XRv14 with the E-flag set and the N-flag clear (neither setting is a default). An interesting note about the N-flag is that the routers must ignore the N-flag if it is set on a prefix that is not /32 (IPv4) or /128 (IPv6), as it is meant to represent a stable transport address for LSPs.

```
! XRv14
router isis 1
 interface Loopback1
   address-family ipv4 unicast
     prefix-sid index 1 explicit-null n-flag-clear
```

A quick look at the database reveals the differences between the default-configuration on 14.14.14.14/32 versus the customized configuration on 14.14.14.1/32. The R, V, and L flags are defined in the draft RFCs and do not appear directly configurable in XR version 5.3.0. We also see a correlation between the P-flag and E-flag as discussed above.

```
RP/0/0/CPU0:XRv14#show isis database verbose XRv14.00-00
[snip]
Metric: 0          IP-Extended 14.14.14.1/32
Prefix-SID Index: 1, R:0 N:0 P:1 E:1 V:0 L:0
Metric: 0          IP-Extended 14.14.14.14/32
Prefix-SID Index: 14, R:0 N:1 P:0 E:0 V:0 L:0
```

Let's quickly trace the LSP to see if exp-null is being used when sending traffic toXRv14's new loopback. XRv11 shows the SR label for 14.14.14.1/32 via XRv12 (82000 + 1). The number 1 is the index used to identify that loopback (the prefix-sid index).

```
RP/0/0/CPU0:XRv11#show mpls forwarding prefix 14.14.14.1/32
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
91008	82001	14.14.14.1/32	Gi0/0/0/0.512	12.0.0.12	192

XRv12 performs a swap to label explicit-null (label 0 for IPv4), thus delivering the topmost EXP markings intact to XRv14 at the end of the LSP.

```
RP/0/0/CPU0:XRv12#show mpls forwarding labels 82001
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
82001	Exp-Null-v4	No ID	Gi0/0/0/0.524	24.0.0.14	654

Additional Reading – Reference configurations "sr"

8.2 Generalized MPLS (GMPLS)

GMPLS is an extension of the MPLS concept whereby any path attribute that can identify a flow can be specified. Specifically, GMPLS (sometimes called Multiprotocol Lambda Switching) targets optical networks for its use.

Given an all-optical network, traffic is often carried over these fibers in multiple different wavelengths. These different light waves are multiplexed (mux'ed) at the head-end and demultiplexed (demux'ed) at the tail end of the path. The links have enormous capacity as a single fiber strand can carry several wavelengths using Wavelength Division Multiplexing (WDM). It comes in Coarse and Dense (CWDM and DWDM) varieties, with the only difference being how closely two adjacent wavelengths are together. Dense means they are more tightly packed (smaller gaps) and is the only model deployed widely today in large carrier networks. CWDM is less expensive and might be appropriate in smaller networks. Note: The RF world calls this Frequency Division Multiplexing (FDM), but the two terms are identical. Changing frequency means changing wavelength no matter what. WDM technologies were discussed in a dedication chapter earlier in the book.

GMPLS seeks to provide a mechanism to set up "light paths" from end to end based on a set of constraints. Just like regular MPLS-TE, a specific amount of bandwidth or set of link colors might be preferred. GMPLS extends the idea to include anything else that might be relevant, including layer 1 components such as wavelength, fiber strand, etc. Provided the mechanism by which labels are allocated is aware of these characteristics, GMPLS labels can steer traffic accordingly. The main motivation for selecting an end-to-end wavelength is to guarantee connectivity. For example, each optical transport device in the network is certainly smart enough to dynamically determine which

available wavelengths exist on its links and select one accordingly. This is done using some kind of waveform assignment algorithm. Insufficient network resources may cause a light-path not to be established. Selecting a lambda explicitly for certain flows can simplify troubleshooting and also ensure the signal doesn't have to be "changed" multiple times in the network. The signal quality degrades over distance and time, but if the signal needed to be totally "changed" at each optical network device, it would be less efficient for the network to manage this than the occasional signal "clean up".

Another benefit of GMPLS is that it supports bidirectional LSPs, which is not supported in normal IP/MPLS networks. If using this feature, the requirements for the LSP are the same in both directions, which reduces latency during setup time. Explicit-paths can also be used to specify each hop/wavelength to use along the path in the optical network. There are some draft RFCs that discuss extending OSPF and IS-IS to flood this TE information, but I do not see Cisco claiming support for this on any platform.

XE has the command syntax for GMPLS interface but does not appear well-documented or widely used at the time of this writing. XR has a basic configuration guide but it does not mention using IGP to carry the TED.

8.3 MPLS Transport Profile (MPLS-TP)

MPLS TP is a mechanism to adjust typical MPLS behavior (technically IP/MPLS) to better emulate TDM networks. Carriers liked the idea of MPLS and saw obvious benefits, but wanted the additional OAM features and circuit-oriented approach within TDM and SONET/SDH architectures. MPLS TP supplants transport label allocation mechanisms, such as RSVP-TE, LDP, BGP, and SR. It stands in contrast to IP/MPLS in three key areas since these features are NOT supported in MPLS-TP:

1. **PHP:** Because the transport labels define the path, and MPLS-TP paths are statically configured and highly explicit, the PHP cannot remove the transport label. MPLS-TP is like extending a layer 2 circuit over an MPLS network, which includes all of the alarm signaling contained therein.
2. **ECMP:** MPLS-TP requires all paths to be congruent (symmetric) which is not a requirement in IP/MPLS. In IP/MPLS, paths can be asymmetric (non-congruent), LSPs can be unidirectional, etc. MPLS-TP requires congruence with bi-directional LSPs, much like a TDM/SONET/SDH circuit.
3. **Label merge:** When two LSPs reach a common LSR and have a common next-hop, their LSPs can be merged by using a single outgoing-label. In LDP, this is how MP2P trees can form and is highly efficient. This is not allowed in MPLS-TP since every LSP is entirely different, again, like a traditional circuit.

Despite not supporting these features, MPLS-TP supports several benefits over IP/MPLS.

1. **No need to configure IP:** There is no concept of binding labels to IP prefixes in MPLS-TP. While MPLS-TP can use IP addresses on a per-link basis for building its paths, this is optional. The lab shown later demonstrates this.
2. **Advanced OAM:** Rich set of tools to monitor and manage the MPLS-TP and the PWs that run through it. This includes the Generic Alert Label (GAL) and the Generic Associated Channel (G-ACH). The purpose of the GAL is to alert the router to the presence of the G-ACH within the

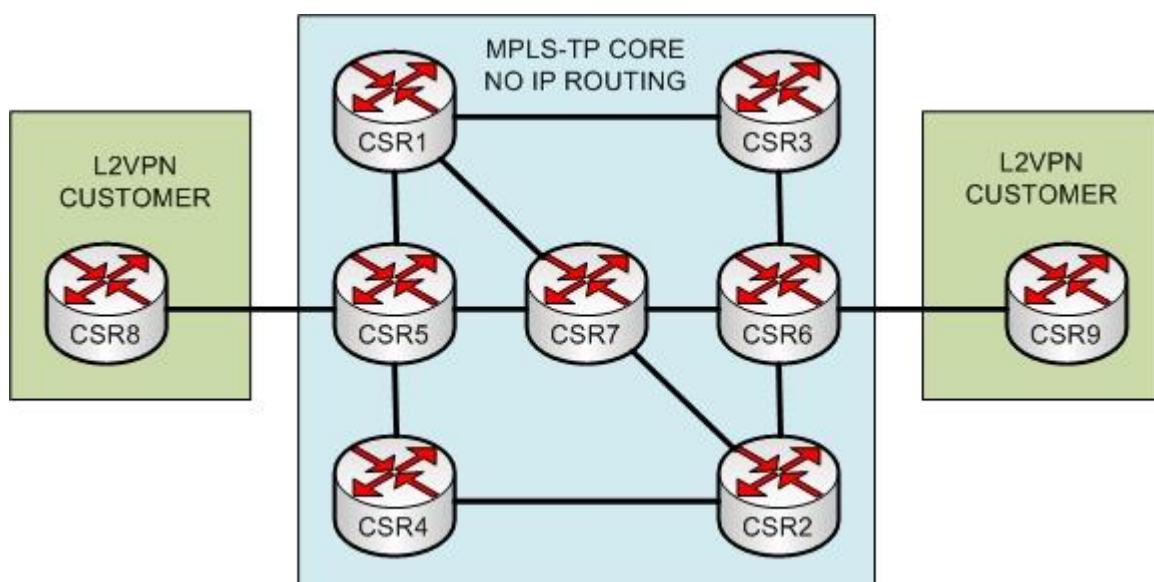
header. This is similar to the PC-ACH seen in the VCCV section and is discussed in detail later. In summary, this provides SONET/SDH-like features such as automatic protection switching (APS) and data communications channel (DCC). These are not available in IP/MPLS.

3. **Fault reporting:** As a component of OAM, there are 3 main message types used for fault management and reporting with MPLS-TP. Within the scope of the labs in this book, these are very similar in concept/behavior to Ethernet CFM (specifically ITU-T Y.1371 enhancements).
 - a. **Link Down Indicator (LDI):** Generated by a midpoint router when a failure occurs. Since the failure will break connectivity towards one end of the circuit, the message is sent back to the end that is still reachable. This will cause a switch from working to protect LSPs.
 - b. **Lock Report (LKR):** Generated by a midpoint router when an interface is administratively shutdown. Like the LDI, this message is sent back to the end that is still reachable. This will also cause a switch from working to protect LSPs.
 - c. **Alarm Indication Signal (AIS):** Not generated by Cisco, but is used to report general alarms along the LSP. Receipt of this message will NOT cause a switch from working to protect LSPs.

The network diagram is shown below. Since MPLS-TP is not supported in XRv, we use only CSR1000v routers in this lab. There is no IP routing configured anywhere. CSR5 and CSR6 are the PE routers while all other routers are P routers. CSR5 and CSR6 will provide AToM services for a number of VCs to connect customer routers CSR8 and CSR9. Below is quick proof that we cannot test MPLS-TP on XRv.

```
! XRv1
mpls traffic-eng
tp
node-id 11.11.11.11
```

!!% The requested operation is not supported: MPLS-TP is not supported on this platform



The link configurations are tedious but very simple. Here is an example from CSR5; note that only physical interfaces support MPLS-TP (no virtual interfaces, not even dot1q subinterfaces). Each link eligible to carry MPLS-TP LSPs must have a link number that must be unique on each router. You can specify the next-hop one of three ways:

1. **Next-hop MAC address:** On Ethernet networks, this is necessary since the router needs to know how to encapsulate the MPLS packet. I use this commonly throughout the lab and each interface has simple MAC addresses to facilitate easy reading (Gig1 and Gig2).
2. **Next-hop IPv4 address:** Assuming IPv4 is running in the network and is configured on a particular interface, you can specify the IP next-hop versus the MAC address. I only did this on the link between CSR5 and CSR4 for demonstration, nowhere else (Gig3).
3. **Treat as P2P link:** On P2P links, or Ethernet links identified as P2P using “medium p2p”, you can simply assign a TP link number without specifying any kind of next-hop (Gig7). As we will see later, this uses the destination MAC address of 0180.c200.0000, which is an IEEE reserved MAC typically used for STP. The idea is that there can be no STP-aware switches in the network between a pair of nodes, since MPLS-TP can’t really be sure these two routers are directly connected. You can obviously “hack” this by using switches that just flood frames sent to this multicast MAC address, but that defeats the purpose of MPLS-TP. Because my lab routers move between different physical hosts, there could be STP-aware switches in between, which will consume these frames. This method doesn’t work in my particular setup, which is why I use the next-easiest MAC address method.

For consistency I use the “tx-mac” method on all other links in the topology. We can immediately begin to see that MPLS-TP is very strict and non-dynamic. Thought it can interwork with targeted LDP and G-MPLS, that is beyond the scope of this test. The snippet below shows all three methods of enabling MPLS-TP on a link; no matter which method is used, you must specify a link ID.

```
! CSR5
interface GigabitEthernet1
  description TO R1
  mac-address 0000.0015.0005
  mpls tp link 1 tx-mac 0000.0015.0001

interface GigabitEthernet2
  description TO R7
  mac-address 0000.0057.0005
  mpls tp link 7 tx-mac 0000.0057.0007

interface GigabitEthernet3
  description TO R4
  ip address 10.4.5.5 255.255.255.0
  mpls tp link 4 ipv4 10.4.5.4
```

```

interface GigabitEthernet7
  description TO NOWHERE
  medium p2p
  mpls tp link 999

```

To verify this configuration, we can use a special show command. Notice that Gig1 and Gig2 identify a next-hop MAC and nothing else. Gig3 identifies a next-hop IP address, which implies ARP will be used to resolve the next-hop MAC address. Gig7 shows the reserved IEEE MAC 0180.c200.0000 as discussed earlier, which I show here just for demonstration.

```
R5#show mpls tp link-numbers
```

```
MPLS-TP Link Numbers:
```

Link	Interface	Next Hop	RX Macs
1	GigabitEthernet1	0000.0015.0001	
4	GigabitEthernet3	10.4.5.4	
7	GigabitEthernet2	0000.0057.0007	
999	GigabitEthernet7	0180.c200.0000	0180.c200.0000

```
R5#show ip arp 10.4.5.4
```

Protocol	Address	Age (min)	Hardware Addr	Type	Interface
Internet	10.4.5.4	88	0050.56a9.c765	ARPA	GigabitEthernet3

MPLS-TP also requires a static label range because the LSPs are manually (statically) provisioned. Each router has a static range equal to its dynamic range divided by ten. Below, CSR5 uses 5000 – 5999 for dynamic labels and 500 – 599 for static labels. In addition, an IP-like router-ID is required for each MPLS-TP node, although like OSPF, it does not have to be routable. This is required on all routers, to include P routers.

```

! CSR5
mpls label range 5000 5999 static 500 599
mpls tp
  logging events
  router-id 5.5.5.5

```

We can quickly verify these configurations by checking the MPLS-TP summary and MPLs label range. The MPLS-TP summary shows the router-ID, but all of the other fields are zero since we haven't configured any profiles or LSPs. Using similar terminology seen with SONTE/SDH, we can see the concepts of “working” and “protect” used in this context as well.

```
R2#show mpls tp summary
```

```

MPLS-TP: 0::5.5.5.5
  Path protection mode: 1:1 revertive
  PSC: Disabled
  Timers:  Fault OAM: 20 seconds  Wait-to-Restore: 10 seconds
           PSC: Fast-Timer: 1000 milli seconds, 3 messages
           Slow-Timer: 5 seconds

```

```

Endpoints: 0      up: 0      down: 0      shut: 0
  Working: 0      up: 0      down: 0
  Protect: 0      up: 0      down: 0
Midpoints: 0      working: 0      protect: 0
Platform max TP interfaces: 65536

```

```

R5#show mpls label range
Downstream Generic label region: Min/Max label: 5000/5999
Range for static labels: Min/Max label: 500/599

```

As a best practice, we will configure a basic single-hop BFD template for our MPLS-TP tunnels. This will allow MPLS-TP to determine when individual tunnels fails, which is important for failover to work correctly. This is required on all MPLS-TP endpoints, which include CSR5 and CSR6. I use a slow BFD timer to avoid the 100 kbps rate-limit on the CSR1000v.

```

! CSR5 and CSR6
bfd-template single-hop BT_MPLS_TP
  interval min-tx 900 min-rx 900 multiplier 3

```

Next, we will configure an MPLS-TP tunnel. This is somewhat like an MPLS-TE tunnel except it can have two tail ends. Specifically, the concepts of “working” and “protect” are carried over from other circuit-based protocols and configured here. This feels like manually building an LFIB since we are identifying local labels, remote labels, and outgoing interfaces. In this case, we will force traffic for the working LSP towards CSR7 with the protect LSP routing via CSR1. The difficult part is that you must track the label values manually, so I tried to use easy values. Traffic to CSR1 uses label 105 (protect) and traffic to CSR7 uses label 705 (working). The in-labels specified here are for the reverse LSP, which by definition MUST come from the same direction since congruence is required. The LSP numbers are used just to differentiate the paths, and the global-ID is used to make the MPLS-TP router-ID unique in a multi-provider environment. It isn’t necessary in this architecture but I demonstrate it anyway. Both router-ID and global-ID are carried in fault messages to assist the operator in finding the fault areas. BFD is applied directly to this profile while implies it is enabled for both working and protect LSPs.

```

! CSR5
interface Tunnel-tp56
  no ip address
  no keepalive
  tp source 5.5.5.5 global-id 0
  tp destination 6.6.6.6 global-id 0
  bfd BT_MPLS_TP
  working-lsp
    out-label 705 out-link 7
    in-label 507
    lsp-number 0
  protect-lsp
    out-label 105 out-link 1
    in-label 501

```

```
lsp-number 1
```

The configuration on CSR6 is nearly identical to CSR5 with reversed source/destination and a different set of labels and out-links. CSR6 sends traffic with label 607 towards CSR7 (working) and traffic with label 603 to CSR3 (protect).

```
! CSR6
interface Tunnel-tp56
  no ip address
  no keepalive
  tp source 6.6.6.6 global-id 0
  tp destination 5.5.5.5 global-id 0
  bfd BT_MPLS_TP
  working-lsp
    out-label 706 out-link 7
    in-label 607
    lsp-number 0
  protect-lsp
    out-label 306 out-link 3
    in-label 603
    lsp-number 1
```

Next, we have to manually configure every midpoint router. CSR7 is shown first, and remember that the MPLS-TP link numbers, router-ID, and static label range must be configured first (on all routers). The MPLS-TP on CSR5 indicates that label 705 is the out-label from CSR5, which means it is the in-label on CSR7. CSR7 connects this to label 607 outbound to CSR6, which is identified as the in-label on CSR6's MPLS-TP for the working LSP. You can see how this can get very tedious, which makes MPLS-TP a good target for automation. The reverse-LSP follows the same logic except in the opposite direction. The reverse LSP will swap labels in this sequence: 706 > 507.

```
! CSR7
mpls tp lsp source 5.5.5.5 tunnel-tp 56 lsp working destination 6.6.6.6
tunnel-tp 56
  forward-lsp
    in-label 705 out-label 607 out-link 6
  reverse-lsp
    in-label 706 out-label 507 out-link 5
```

Next, we quickly configure the protect path, which has two P routers, CSR1 and CSR3. The same logic applies, except we have to account for the additional routers. Following the forward LSP, the label swapping will be 105 > 301 > 603. The reverse LSP will be 306 > 103 > 501.

```
! CSR1
mpls tp lsp source 5.5.5.5 tunnel-tp 56 lsp protect destination 6.6.6.6
tunnel-tp 56
  forward-lsp
```

```

    in-label 105 out-label 301 out-link 3
reverse-lsp
    in-label 103 out-label 501 out-link 5

! CSR3
mpls tp lsp source 5.5.5.5 tunnel-tp 56 lsp protect destination 6.6.6.6
tunnel-tp 56
    forward-lsp
        in-label 301 out-label 603 out-link 6
    reverse-lsp
        in-label 306 out-label 103 out-link 1

```

Since these routers are just normal LSRs, we can check the LFIB to ensure these labels were programmed correctly. We need to verify the LSPs are correct along with the MAC next-hops. Once the LSPs are operational, we will also see byte counters increase as traffic flows along these LSPs.

```

R1#show mpls forwarding-table labels 100 - 199

```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
103	501	0::6.6.6.6::56::1::0::5.5.5.5	1016836	Gi1	0000.0015.0005
105	301	0::5.5.5.5::56::1::0::6.6.6.6	1014838	Gi2	0000.0013.0003

```

R3#show mpls forwarding-table labels 300 - 399

```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
301	603	0::5.5.5.5::56::1::0::6.6.6.6	1010152	Gi1	0000.0036.0006
306	103	0::6.6.6.6::56::1::0::5.5.5.5	1017086	Gi2	0000.0013.0001

Assuming everything was configured correctly, both the protect and working LSPs should come up. The debugs for MPLS-TP are verbose but not terribly useful. For completeness, we will analyze a few of them, specifically the LSP-endpoint and general event debugs.

```

! CSR5
debug mpls tp lsp-ep
debug mpls tp event

```

The first valuable piece of information we see is MPLS-TP trying to start building the working and protect paths using labels 705 and 105, respectively.

```

! CSR5
mpls_tp_forwarding_headend_update: tun 56 = 0x7FD99A5FA6B8: searching for
adjacencies for if (12)

```

```

mpls_tp_forwarding_headend_update: doing the adj walk? working 0x7FD99A620CF8
install protect 0x7FD99A620C60 install, mpls-tp is enabled
mpls_tp_tunnel_set_mfi_bind_pending: Setting bind pending to true for tp-
tunnel 56
mpls_tp_tunnel_outinfo_fill: tun = 0x7FD99A5FA6B8: work:7/705 prot:1/105
(working is active)
mpls_tp_lsp_outinfo_fill: building path protected output 7:28 0000.0057.0007
backup 6:28 0000.0015.0001

```

The next batch of output isn't very useful. It basically details the LSP construction and binds labels to LSPs.

```

! CSR5
mpls_tp_tunnel_clear_mfi_bind_pending: Clearing bind pending to true for tp-
tunnel 56
mpls_tp_lsp_ep_forwarding_tailend_update: tun = 0x7FD99A5FA6B8: build rec if
for 12
mpls_tp_lsp_ep_forwarding_tailend_update: tun 0x7FD99A5FA6B8: adding static
label 507 bind for lsp_ep 0x7FD99A620CF8 (0)
mpls_tp_handle_label_reply: found request for label 507 reply
TP-MFI:type 13: label table entry update, req 140572574747936, rc: success
allocated 507
mpls_tp_lsp_static_label_bind: 0:101058054:56:0 static binding 507/507 - rc
success
mpls_tp_lsp_ep_forwarding_tailend_update: tun = 0x7FD99A5FA6B8: build rec if
for 12
mpls_tp_lsp_ep_forwarding_tailend_update: tun 0x7FD99A5FA6B8: adding static
label 501 bind for lsp_ep 0x7FD99A620C60 (1)
mpls_tp_handle_label_reply: found request for label 501 reply
TP-MFI:type 13: label table entry update, req 140572574747784, rc: success
allocated 501
mpls_tp_lsp_static_label_bind: 0:101058054:56:1 static binding 501/501 - rc
success

```

Next, we can see MPLS-TP is bound to BFD and OAM. This applies to both LSP 0 and LSP 1, which we configured to be our working and protect LSPs, respectively. OAM in this context refers to the fault signaling that is exchanged between the MPLS-TP LSRs.

```

! CSR5
mpls_tp_lsp_ep_add_bfd_session: LSP added in BFD db if 12 tun 56 lsp 0
session handle 1
mpls_tp_lsp_ep_add_bfd_session: LSP added in BFD db if 12 tun 56 lsp 1
session handle 2
mpls_tp_lsp_ep_add_fault_session: Fault OAM added for EP LSP: tun 56 lsp 0
session hdl 2600468486
mpls_tp_lsp_ep_add_fault_session: Fault OAM added for EP LSP: tun 56 lsp 1
session hdl 1677721607

```



```
mpls_tp_bfd_session_notify_callback: BFD session notify event ADJ UP
Received for handle 1
mpls_tp_bfd_session_notify_callback: BFD session notify event ADJ UP
Received for handle 2
```

Filtering some of the unnecessary debugs, we ultimately see both the working and protect LSPs are fully up. As you can see, these debugs are difficult to read and aren't very telling.

```
! CSR5
mpls_tp_get_pp_event_and_run_pp_fsm: tunnel:56 lsp:working
lsp_fsm_event:BFD_UP sending pp_fsm_event:WRK_UP

mpls_tp_get_pp_event_and_run_pp_fsm: tunnel:56 lsp:protect
lsp_fsm_event:BFD_UP sending pp_fsm_event:PROT_UP
```

We can verify this by looking at the MPLS-TP tunnel. The summary form is shown first, followed by the detailed form. We can see that the working LSP is being used along the LSP with out-label 705. The details indicate there are no OAM faults as well.

```
R5#show mpls tp tunnel-tp 56
```

Tunnel Number	Peer global-id::node-id::tun	Active LSP	Local Label	Out Label	Out Interface	Oper State
56	0::6.6.6.6::56	work	507	705	Gi2	up

```
R5#show mpls tp tunnel-tp 56 detail
MPLS-TP tunnel 56:
  src global id: 0      node id: 5.5.5.5  tunnel: 56
  dst global id: 0      node id: 6.6.6.6  tunnel: 56
  description:
  Admin: up             Oper: up
  bandwidth:            0
  BFD template:  BT_MPLS_TP
  protection trigger: LDI LKR
  PSC: Disabled
  working-lsp: Active   lsp num 0
    BFD State: Up
    Lockout   : Clear
    Fault OAM: Clear
  protect-lsp: Standby  lsp num 1
    BFD State: Up
    Lockout   : Clear
    Fault OAM: Clear
```

We can achieve per-LSP granularity by using a different command. This shows both the working (0) and protect (1) LSPs together, and we see both are up. The details also provide information about bandwidth reservations (seen later) as well as the label values and out-links.

```
R5#show mpls tp lsps 5.5.5.5 tunnel-tp 56
```

```
MPLS-TP Endpoint LSPs:
```

LSP Identifier	Role	Local Label	Out Label	Out Interface	Oper State
0::5.5.5.5::56::0::6.6.6.6::56::0	actv	507	705	Gi2	up
0::5.5.5.5::56::0::6.6.6.6::56::1	stby	501	105	Gi1	up

```
R5#show mpls tp lsps 5.5.5.5 tunnel-tp 56 detail
```

```
MPLS-TP Endpoint LSPs:
```

```
0::5.5.5.5::56::0::6.6.6.6::56::0 (working/active)
  in label 507      label table 0      out label 705
  outgoing tp-link 7 interface Gi2
  Forwarding: Installed, Bandwidth: 0 Admitted
0::5.5.5.5::56::0::6.6.6.6::56::1 (protect/standby)
  in label 501      label table 0      out label 105
  outgoing tp-link 1 interface Gi1
  Forwarding: Installed, Bandwidth: 0 Admitted
```

MPLS-TP can also be a client of BFD. The command is very repetitive, but below we can see the BFD summary and detailed information about these sessions. BFD is also aware which is the protect and which is the working LSP.

```
R5#show bfd neighbors client mpls-tp mpls-tp tunnel-tp 56
```

```
MPLS-TP Sessions
```

Interface	LSP type	LD/RD	RH/RS	State
Tunnel-tp56	Protect	7/4	Up	Up
Tunnel-tp56	Working	6/3	Up	Up

```
R5#show bfd neighbors client mpls-tp mpls-tp tunnel-tp 56 details | include
Regist|Tunn
```

Tunnel-tp56	Protect	7/4	Up	Up
Registered protocols: MPLS-TP				
Tunnel-tp56	Working	6/3	Up	Up
Registered protocols: MPLS-TP				

We have our standard OAM options like ping and traceroute as well. The default is non-IP encapsulated messages within the G-ACH (ending in 0x0023) which is different than some of the PW-ACH formats we saw earlier (ending in 0x0021). Because we are not routing IP in this network, we will use the default G-ACH channel type, but I show the options below.

```
R5#ping mpls tp tunnel-tp 56 lsp working channel ?
```

```
cv use non-ip encapsulation with GACH channel 0x0025
```

```
ip use ip encapsulation with GACH channel 0x0021
```

First, we will check the working LSP. The keywords “working” and “protect” allow us to select which path to test. Notice that traceroute shows us the static LSP we provisioned which the labels hop-by-hop.

```
R5#ping mpls tp tunnel-tp 56 lsp working
Sending 5, 72-byte MPLS Echos to Tunnel-tp56,
    timeout is 2 seconds, send interval is 0 msec:
[snip]
```

Type escape sequence to abort.

!!!!

```
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/8 ms
Total Time Elapsed 24 ms
```

```
R5#traceroute mpls tp tunnel-tp 56 lsp working
Tracing MPLS TP Label Switched Path on Tunnel-tp56, timeout is 2 seconds
[snip]
```

Type escape sequence to abort.

```
0 0::5.5.5.5 MRU 1500 [Labels: 705 Exp: 0]
L 1 0::7.7.7.7 MRU 1500 [Labels: 607 Exp: 0] 7 ms
! 2 0::6.6.6.6 4 ms
```

We can also explicitly check the protect LSP as well. This should be a totally independent path from the working LSP. This LSP routes from CSR1 to CSR3 to CSR6 as expected.

```
R5#ping mpls tp tunnel-tp 56 lsp protect
Sending 5, 72-byte MPLS Echos to Tunnel-tp56,
    timeout is 2 seconds, send interval is 0 msec:
[snip]
```

Type escape sequence to abort.

!!!!

```
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/5/8 ms
Total Time Elapsed 27 ms
```

```
R5#traceroute mpls tp tunnel-tp 56 lsp protect
Tracing MPLS TP Label Switched Path on Tunnel-tp56, timeout is 2 seconds
[snip]
```

Type escape sequence to abort.

```
0 0::5.5.5.5 MRU 1500 [Labels: 105 Exp: 0]
L 1 0::1.1.1.1 MRU 1500 [Labels: 301 Exp: 0] 8 ms
L 2 0::3.3.3.3 MRU 1500 [Labels: 603 Exp: 0] 4 ms
! 3 0::6.6.6.6 5 ms
```

We will use EPC to look at the BFD packets being sent down the LSPs by capturing on both out-links at the same time from CSR5. Although BFD was configured once under the template, it is technically two independent sessions, so the sending of BFD packets may not be at exactly the same moment. The

working LSP timestamps are highlighted in green and the protect LSP timestamps are highlighted in yellow (the MAC addresses make it easy to tell). The MPLS label stacks are interesting since we have a new label 13 (0xD), which is the GAL label. The first label in the stack is the MPLS-TP label, which is 105 (0x69) or 705 (0x2C1) depending on which LSP is used. The bottom label is the GAL label, and notice that both labels carry EXP 6 (110). Following the label stacks is the G-ACH, like the PW-ACH. The presence of the GAL indicates that the G-ACH will follow. It begins with bits 0001 to show it is an associated channel, except has a new value of 0x0007. I assume this is specific to MPLS-TP; notice there is no IP traffic anywhere in this packet which means 0x0021 isn't used, and this isn't a G-ACH OAM test packet (like a ping) so 0x0025 isn't used either.

```
R5#show monitor capture CAP buffer detail
 0  50    0.000000  00:00:00:15:00:05 -> 00:00:00:15:00:01 MPLS unicast
0000:  00000015 00010000 00150005 88470006  .....G..
0010:  9CFF0000 DD011000 000720C8 03180000  .....
0020:  00070000 0004000D BBA0000D BBA00000  .....
0030:  0000                                     ..

 1  50    0.618956  00:00:00:57:00:05 -> 00:00:00:57:00:07 MPLS unicast
0000:  00000057 00070000 00570005 8847002C  ...W.....W...G.,
0010:  1CFF0000 DD011000 000720C8 03180000  .....
0020:  00060000 0003000D BBA0000D BBA00000  .....
0030:  0000                                     ..

 2  50    0.853945  00:00:00:15:00:05 -> 00:00:00:15:00:01 MPLS unicast
0000:  00000015 00010000 00150005 88470006  .....G..
0010:  9CFF0000 DD011000 000720C8 03180000  .....
0020:  00070000 0004000D BBA0000D BBA00000  .....
0030:  0000                                     ..

 3  50    1.438973  00:00:00:57:00:05 -> 00:00:00:57:00:07 MPLS unicast
0000:  00000057 00070000 00570005 8847002C  ...W.....W...G.,
0010:  1CFF0000 DD011000 000720C8 03180000  .....
0020:  00060000 0003000D BBA0000D BBA00000  .....
0030:  0000                                     ..
```

We will perform another capture, except this time we will look at the OAM packets. I sent one 500-byte OAM packet so that it stands out amongst the BFD packets. The G-ACH is highlighted in yellow and now shows 0x0025 as the channel header, which is correct. The GAL is highlighted in green and serves an identical purpose as before except the EXP is 0 since we did not specify a custom value in the MPLS ping.

```
R5#ping mpls tp tunnel-tp 56 lsp working size 500 repeat 1
Sending 1, 500-byte MPLS Echos to Tunnel-tp56,
    timeout is 2 seconds, send interval is 0 msec:
[snip]

Type escape sequence to abort.
!
```

Success rate is 100 percent (1/1), round-trip min/avg/max = 8/8/8 ms
Total Time Elapsed 9 ms

R5#show monitor capture CAP buffer detail

```
5 526 1.271974 00:00:00:57:00:05 -> 00:00:00:57:00:07 MPLS unicast
0000: 00000057 00070000 00570005 8847002C ...W.....W...G.,
0010: 10FF0000 D1011000 00250001 00000104 .....%.....
0020: 0000C013 93E60000 0002D9DF 8CAD6312 .....C.
0030: 6E970000 00000000 0000FC00 000C0000 n.....
```

Now that we have verified the MPLS-TP transport LSPs, we can configure a basic PW over this profile. Both legacy and L2VPN syntaxes are supported, but we will begin with the legacy syntax. I also define a static-OAM class which identifies a timeout setting for OAM messages. The PW-class enables the CW and disables LDP as a signaling protocol. It applies the OAM-class for status in lieu of LDP and assigns the MPLS-TP interface as a preferred path. This is similar to using MPLS-TE for PW support, except this time we use MPLS-TP.

```
! CSR5
pseudowire-static-oam class OAM_CLASS
    timeout refresh send 20

pseudowire-class PW_CLASS
    encapsulation mpls
    control-word
    protocol none
    preferred-path interface Tunnel-tp56
    status protocol notification static OAM_CLASS
```

The AC configuration is identical to a normal static PW. We define the labels and specify the neighbor ID (again, not reachable via IP, but just the remote MPLS-TP router-ID). The configurations are nearly identical on CSR6 and are not shown here.

```
! CSR5
interface GigabitEthernet6
    description CUSTOMER AC
    service instance 56 ethernet
    encapsulation dot1q 3558 second-dot1q 100
    rewrite ingress tag pop 2 symmetric
    xconnect 6.6.6.6 100 encapsulation mpls manual pw-class PW_CLASS
    mpls label 506 605
    mpls control-word
```

We verify that the PW comes up. The detailed view looks identical to a normal PW, but the details show that MPLS-TP is used as the preferred path. It also shows us the transport label of 705 which is used for the working LSP.

```
R5#show mpls l2transport vc 100
```

Local intf	Local circuit	Dest address	VC ID	Status
Gi6	Eth VLAN 3558/100	6.6.6.6	100	UP

```
R5#show mpls l2transport vc 100 detail | section Destination
Destination address: 6.6.6.6, VC ID: 100, VC status: up
Output interface: Tp56, imposed label stack {705 605}
Preferred path: Tunnel-tp56, active
Default path:
Next hop: point2point
```

Testing this LSP with OAM is tricky. If we use a basic MPLS ping syntax, it won't work. This may lead you to think (as I did) that the PW was somehow misconfigured despite it being UP.

```
R5#ping mpls pseudowire 6.6.6.6 100
Sending 5, 72-byte MPLS Echos to 6.6.6.6,
    timeout is 2 seconds, send interval is 0 msec:
[snip]
```

```
Type escape sequence to abort.
```

```
.....
```

```
Success rate is 0 percent (0/5)
```

```
Total Time Elapsed 9368 ms
```

The reason is because there is no IP routing in the network. When using MPLS OAM, the replies are IPv4 by default. Debugging on CSR6 can help reveal this.

```
! CSR6
```

```
debug mpls lspv event
```

```
LSPV: labelval 605
```

```
LSPV: FECVAL 1, table 0, label 605, adv_label 605, type 10
```

```
LSPV: FEC map info: advertised label 0x25D, retcode 2
```

```
LSPV: FEC Validation, PW FEC validated
```

```
LSPV: FEC Validation, fs-depth 1, fec_status 0, fec-rc 0, mapping retcode 2,
best_rc_old 3, best_rc 3
```

```
LSPV: Processing reply after jitter
```

```
LSPV: Reply sent via IP
```

In the OAM section, I demonstrate different reply options. When the CW is negotiated, the reply can be carried in the associated channel, and this technique does work. We will use EPC inbound on CSR5 to look at the reply packet. Just like in the OAM section, we see 0x0021 in the CW to identify this as an IPv4 packet, which is fine since it is being processed locally. The significant part is that the packet is MPLS-encapsulated inside the ACH on the way back. Without MPLS-TP, this was never an issue since IP routing guaranteed reachability between PW endpoints.

```
R5#ping mpls pseudowire 6.6.6.6 100 reply mode control-channel
Sending 5, 72-byte MPLS Echos to 6.6.6.6,
    timeout is 2 seconds, send interval is 0 msec:
[snip]

Type escape sequence to abort.
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 3/6/13 ms
Total Time Elapsed 33 ms
```

```

 4 126      2.799022  00:00:00:57:00:07 -> 00:00:00:57:00:05 MPLS unicast
0000:  00000057 00050000 00570007 8847001F  ...W.....W...G..
0010:  B0FE001F A1011000 002145C0 0064006D  .....!E..d.m
0020:  0000FF11 A4460606 06060505 05050DAF  .....F.....
0030:  0DAF0050 04410001 00000204 0301C013  ...P.A.....
```

We can also test the failover from working to protect LSP. If we administratively shutdown CSR7's Gig4, this will break the working LSP, and the protect LSP should become active after BFD detects the failure. Our definitions earlier suggest that CSR7 should send a Lock Report (LKR) back to CSR5 but not to CSR6. Both CSR5 and CSR6 show log messages to indicate these changes, since both of them are running BFD and will detect the failure that way. We can clearly see that the issue is with CSR7, TP link 6. The code of CC (continuity check) represents a BFD failure since CSR6 didn't get the detailed OAM alert. If we had multiple PWs, they could all be using this transport profile, which effectively becomes a FEC.

```
! CSR5
%MPLS_TP_LSP-3-UPDOWN: Working LSP 0::5.5.5.5::56::0::6.6.6.6::56::0 is down:
LKR:0::7.7.7.7::6
%MPLS_TP-5-REDUNDANCY: Tunnel-tp56, switched to Protect LSP as active

! CSR6
%MPLS_TP_LSP-3-UPDOWN: Working LSP 0::6.6.6.6::56::0::5.5.5.5::56::0 is down:
CC
%MPLS_TP-5-REDUNDANCY: Tunnel-tp56, switched to Protect LSP as active
```

Looking at the LSP details for this MPLS-TP instance, we can see that the protect LSP is now "active". This failover process took less than 3 seconds as we can see by the pings sent within the customer network between CSR8 and CSR9. Failure detection takes 2.7 seconds (900 ms times 3) which is consistent with the output below. In a production environment, BFD would have detected the failure much more quickly, which better approximates SONET/SDH failure detection behavior.

```
R5#show mpls tp lsps 5.5.5.5 tunnel-tp 56
MPLS-TP Endpoint LSPs:
```

LSP Identifier	Local Role	Out Label	Out Interface	Oper State
0::5.5.5.5::56::0::6.6.6.6::56::0	stby	507	Gi2	down
0::5.5.5.5::56::0::6.6.6.6::56::1	actv	501	Gi1	up

```

R8#ping 10.8.9.9 repeat 10000000
Type escape sequence to abort.
Sending 10000000, 100-byte ICMP Echos to 10.8.9.9, timeout is 2 seconds:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!..!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Success rate is 99 percent (303/305), round-trip min/avg/max = 3/22/31 ms

```

In case logging was disabled or we simply didn't see the message, we can see the fault OAM code by looking at the MPLS-TP details. CSR6 doesn't show the fault code since CSR7 has no reachability to CSR6 along this LSP anymore, so all faults appear "clear". The working LSP is still down thanks to BFD, so connectivity is restored using the protect LSP.

```

R5#show mpls tp tunnel-tp 56 detail
MPLS-TP tunnel 56:
  src global id: 0      node id: 5.5.5.5  tunnel: 56
  dst global id: 0      node id: 6.6.6.6  tunnel: 56
  description:
  Admin: up             Oper: up
  bandwidth:            0
  BFD template:  BT_MPLS_TP
  protection trigger:  LDI LKR
  PSC: Disabled
  working-lsp: Standby   lsp num 0
    BFD State: Down
    Lockout   : Clear
    Fault OAM: LKR
  protect-lsp: Active    lsp num 1
    BFD State: Up
    Lockout   : Clear
    Fault OAM: Clear

```

```

R6#show mpls tp tunnel-tp 56 detail | begin working
working-lsp: Standby   lsp num 0
  BFD State: Down
  Lockout   : Clear
  Fault OAM: Clear
protect-lsp: Active    lsp num 1
  BFD State: Up
  Lockout   : Clear
  Fault OAM: Clear

```

We can verify that the protect LSP is active by using OAM with the "active" keyword. This could be either the working or protect LSP, depending on which one is currently active. We can see that the LSP traverses through CSR1 and CSR3 as expected.


```
R5#traceroute mpls tp tunnel-tp 56 lsp active
Tracing MPLS TP Label Switched Path on Tunnel-tp56, timeout is 2 seconds
[snip]
```

Type escape sequence to abort.

```
0 0::5.5.5.5 MRU 1500 [Labels: 105 Exp: 0]
L 1 0::1.1.1.1 MRU 1500 [Labels: 301 Exp: 0] 8 ms
L 2 0::3.3.3.3 MRU 1500 [Labels: 603 Exp: 0] 4 ms
! 3 0::6.6.6.6 5 ms
```

Bringing CSR7's interface back up causes the working LSP to become active again, since the failover process is "revertive" by default. However, the reversion timer is 10 seconds, which is used to protect against flaps in the network. It is better to continue using a stable protect LSP rather than switch to the working LSP too quickly after a failure. This is loosely analogous to the LDP IGP sync-delay timer.

```
! CSR5
19:56:17.007: %MPLS_TP_LSP-3-UPDOWN: Working LSP
0::5.5.5.5::56::0::6.6.6.6::56::0 is up
19:56:27.007: %MPLS_TP-5-REDUNDANCY: Tunnel-tp56, switched to Working LSP as
active
```

```
! CSR6
19:56:27.609: %MPLS_TP_LSP-3-UPDOWN: Working LSP
0::6.6.6.6::56::0::5.5.5.5::56::0 is up
19:56:37.609: %MPLS_TP-5-REDUNDANCY: Tunnel-tp56, switched to Working LSP as
active
```

For completeness, we can verify this behavior again using the summary show command. The wait-to-restore (WTR) timer is 10 seconds by default and is controlled globally. The context-sensitive help is shown below as well.

```
R5#show mpls tp summary
MPLS-TP:      0::5.5.5.5
Path protection mode: 1:1 revertive
PSC: Disabled
Timers: Fault OAM: 20 seconds Wait-to-Restore: 10 seconds
        PSC: Fast-Timer: 1000 milli seconds, 3 messages
            Slow-Timer: 5 seconds
Endpoints: 1      up: 1      down: 0      shut: 0
          Working: 1      up: 1      down: 0
          Protect: 1      up: 1      down: 0
Midpoints: 0      working: 0      protect: 0
Platform max TP interfaces: 65536
```

```
R5(config-mpls-tp)#wtr-timer ?
<0-2147483647> Time in seconds to wait before restoring from protect to
```

working

We can use show commands along with OAM to confirm this change. Using the “active” option, we now see that the working LSP is active for this transport profile.

```
R5#show mpls tp lsp 6.6.6.6 tunnel-tp 56
MPLS-TP Endpoint LSPs:
```

LSP Identifier	Role	Local Label	Out Label	Out Interface	Oper State
0::5.5.5.5::56::0::6.6.6.6::56::0	actv	507	705	Gi2	up
0::5.5.5.5::56::0::6.6.6.6::56::1	stby	501	105	Gi1	up

```
R5#traceroute mpls tp tunnel-tp 56 lsp active
Tracing MPLS TP Label Switched Path on Tunnel-tp56, timeout is 2 seconds
[snip]
```

Type escape sequence to abort.

```
0 0::5.5.5.5 MRU 1500 [Labels: 705 Exp: 0]
L 1 0::7.7.7.7 MRU 1500 [Labels: 607 Exp: 0] 8 ms
! 2 0::6.6.6.6 4 ms
```

We will quickly demonstrate the LDI condition which occurs when a link actually goes down (not including admin-shut). We demonstrate this by disconnecting CSR7’s vNIC to CSR6 from within VMware so that the interface appears unplugged. While the log message shows CC as the error, the TP details show it (correctly) as LDI. Before continuing, we reconnect CSR7’s vNIC to CSR6.

```
R5#show mpls tp tunnel-tp 56 detail | begin working
      working-lsp: Standby      lsp num 0
      BFD State: Down
      Lockout : Clear
      Fault OAM: LDI
      protect-lsp: Active      lsp num 1
      BFD State: Up
      Lockout : Clear
      Fault OAM: Clear
```

Next, we will configure another MPLS-TP tunnel along the path CSR5 > CSR4 > CSR2 > CSR6. This will carry a second PW using VCID 200. It is possible to bind multiple PWs to a single MPLS-TP, which basically follows the logic of a FEC as mentioned earlier. However, we will map the second PW to a new MPLS-TP tunnel for variety. This TP will request 4 Mbps of bandwidth as well; the interesting part is that we must configure IP RSVP at the endpoints, but not the midpoints.

```
! CSR5 and CSR6
interface GigabitEthernet3
 ip rsvp bandwidth 5000
```

The MPLS-TP configurations are near identical on CSR5 and CSR6, so only CSR5 is shown. This requests 4 Mbps of bandwidth and also has a custom TP name. There is no protect LSP (not required) for brevity, but I would have configured it over one of the other 2 paths in the network not associated with this MPLS-TP. We can re-use the BFD template as well.

```
! CSR5
interface Tunnel-tp560
  no ip address
  no keepalive
  tp bandwidth 4000
  tp tunnel-name COOL_NAME
  tp source 5.5.5.5 global-id 0
  tp destination 6.6.6.6 global-id 0
  bfd BT_MPLS_TP
  working-lsp
    out-label 405 out-link 4
    in-label 504
    lsp-number 2
```

The midpoints of CSR4 and CSR2 are very similar to CSR1 and CSR3. They just connect the LSP together, and RSVP is not required on them since there isn't actually any RSVP signaling. The endpoints just look at their local egress interfaces for admission control, which isn't very comprehensive. The forward LSP uses labels 405 > 204 > 602. The reverse LSP uses labels 206 > 402 > 504.

```
! CSR4
mpls tp lsp source 5.5.5.5 tunnel-tp 560 lsp working destination 6.6.6.6
tunnel-tp 560
  forward-lsp
    in-label 405 out-label 204 out-link 2
  reverse-lsp
    in-label 402 out-label 504 out-link 5
```

```
! CSR2
mpls tp lsp source 5.5.5.5 tunnel-tp 560 lsp working destination 6.6.6.6
tunnel-tp 560
  forward-lsp
    in-label 204 out-label 602 out-link 6
  reverse-lsp
    in-label 206 out-label 402 out-link 4
```

We verify the tunnel is up by checking the TP summary and seeing a new working LSP (but no new protect LSPs).

```
R5#show mpls tp summary
MPLS-TP:      0::5.5.5.5
```

```

Path protection mode: 1:1 revertive
PSC: Disabled
Timers:  Fault OAM: 20 seconds  Wait-to-Restore: 10 seconds
        PSC: Fast-Timer: 1000 milli seconds, 3 messages
            Slow-Timer: 5 seconds
Endpoints: 2      up: 2      down: 0      shut: 0
Working: 2      up: 2      down: 0
Protect: 1      up: 1      down: 0
Midpoints: 0      working: 0      protect: 0
Platform max TP interfaces: 65536

```

We can also check the MPLS-TP LSPs. We can clearly see the TP number is 560 and the LSP is 2, which clearly differentiates it from the existing LSPs from earlier. The details for this new TP show that there is no protect LSP and that the working LSP is fully operational. It also shows the 4 Mbps bandwidth reservation.

```

R5#show mpls tp lsp
MPLS-TP Endpoint LSPs:

```

LSP Identifier	Local Role	Out Label	Out Interface	Oper State
0::5.5.5.5::56::0::6.6.6.6::56::0	actv	507	Gi2	up
0::5.5.5.5::56::0::6.6.6.6::56::1	stby	501	Gi1	up
0::5.5.5.5::560::0::6.6.6.6::560::2	actv	504	Gi3	up

```

R5#show mpls tp tunnel-tp 560 detail

```

```

MPLS-TP tunnel 560:
  src global id: 0      node id: 5.5.5.5  tunnel: 560
  dst global id: 0      node id: 6.6.6.6  tunnel: 560
  description:
  Admin: up            Oper: up
  bandwidth: 4000
  BFD template:  BT_MPLS_TP
  Name: COOL_NAME
  protection trigger: LDI LKR
  PSC: Disabled
  working-lsp: Active    lsp num 2
  BFD State: Up
  Lockout : Clear
  Fault OAM: Clear
  protect-lsp: none

```

We can see the bandwidth reservations per LSP as well. Here, we can see the working/protect LSPs with profile 56 requesting no bandwidth, but the working LSP with profile 560 has requested 4 Mbps.

```

R5#show mpls tp link-management admission-control
Admitted MPLS-TP Endpoint LSPs:
Tun  Dest                                     Out

```

Num	Global-id::Node-id	LSP	Intf	BW (kbps)
56	0::6.6.6.6	working-lsp:num 0	Gi2	0
56	0::6.6.6.6	protect-lsp:num 1	Gi1	0
560	0::6.6.6.6	working-lsp:num 2	Gi3	4000

When using OAM to trace this LSP from CSR5, we get some interesting output. The MPLS TP router-IDs aren't prefixed with the global-ID (specifically the string "0::") and one of the entries is marked as "unknown upstream index". The reason for this is because IPv4 is enabled on the link between CSR4 and CSR5. As such, the router assumes it can use the IP-encapsulated G-ACH (0x0021) versus the non-IP one (0x0025). The ping still works but gives us odd results. To clean up the output, we explicitly specify the "CV" mode.

```
R5#traceroute mpls tp tunnel-tp 560 lsp working
Tracing MPLS TP Label Switched Path on Tunnel-tp560, timeout is 2 seconds
```

```
Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
'R' - transit router, 'I' - unknown upstream index,
```

```
Type escape sequence to abort.
 0 10.4.5.5 MRU 1500 [Labels: 405 Exp: 0]
L 1 10.4.5.4 MRU 1500 [Labels: 204 Exp: 0] 8 ms
I 2 2.2.2.2 MRU 1500 [Labels: 602 Exp: 0] 6 ms
! 3 6.6.6.6 5 ms
```

```
R5#traceroute mpls tp tunnel-tp 560 lsp working channel cv
Tracing MPLS TP Label Switched Path on Tunnel-tp560, timeout is 2 seconds
[snip]
```

```
Type escape sequence to abort.
 0 0::5.5.5.5 MRU 1500 [Labels: 405 Exp: 0]
L 1 0::4.4.4.4 MRU 1500 [Labels: 204 Exp: 0] 8 ms
L 2 0::2.2.2.2 MRU 1500 [Labels: 602 Exp: 0] 3 ms
! 3 0::6.6.6.6 5 ms
```

Now that the LSP is working, we will create a PW to use it. For this, we can use the new L2VPN syntax, but the logic is the same. Only CSR5 is shown since CSR6 is nearly identical.

```
! CSR5
template type pseudowire PW_TEMP
 encapsulation mpls
 vc type ethernet
 signaling protocol none
 preferred-path interface Tunnel-tp560

interface pseudowire65
 source template type pseudowire PW_TEMP
```

```

encapsulation mpls
neighbor 6.6.6.6 200
signaling protocol none
label 516 615
pseudowire type 5

interface GigabitEthernet6
service instance 65 ethernet
encapsulation dot1q 3558 second-dot1q 200
rewrite ingress tag pop 2 symmetric

l2vpn xconnect context ATOM
member GigabitEthernet6 service-instance 65
member pseudowire65

```

Once the PW is configured, we can verify it is operational, along with revealing its full label stack. The PW label was statically configured as 615 and the MPLS-TP label used to send traffic to CSR4 is 405 (like the FEC). The PW uses MPLS-TP tunnel560 as configured in the PW template above.

```

R5#show l2vpn atom vc vcid 200 detail | section Destination
Destination address: 6.6.6.6 VC ID: 200
Output interface: Tp560, imposed label stack {405 615}
Preferred path: Tunnel-tp560, active
Default path:
Next hop: point2point

```

We can verify that the PW works using OAM. Again, remember to specify the control-channel (PW-ACH) for LSPV replies or else it will not work since IP routing is not enabled in this architecture.

```

R5#ping mpls pseudowire 6.6.6.6 200 reply mode control-channel
Sending 5, 72-byte MPLS Echos to 6.6.6.6,
    timeout is 2 seconds, send interval is 0 msec:
[snip]

Type escape sequence to abort.
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 5/7/10 ms
Total Time Elapsed 36 ms

```

A quick check in the customer network (CSR8 to CSR9) shows that this new VC works.

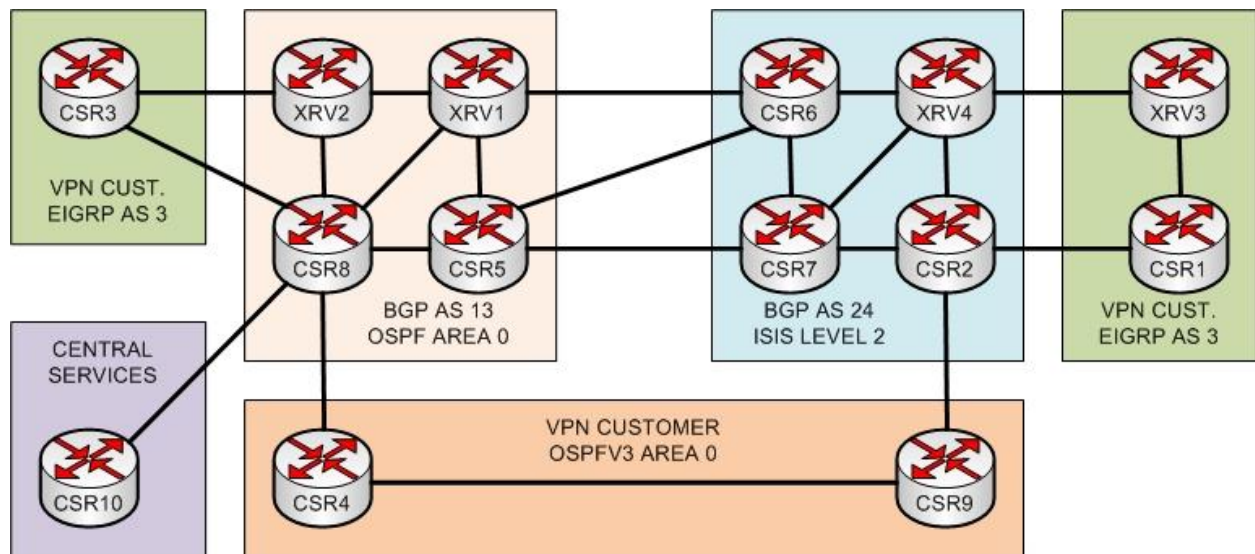
```

R8#ping 20.8.9.9
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 20.8.9.9, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 6/9/20 ms

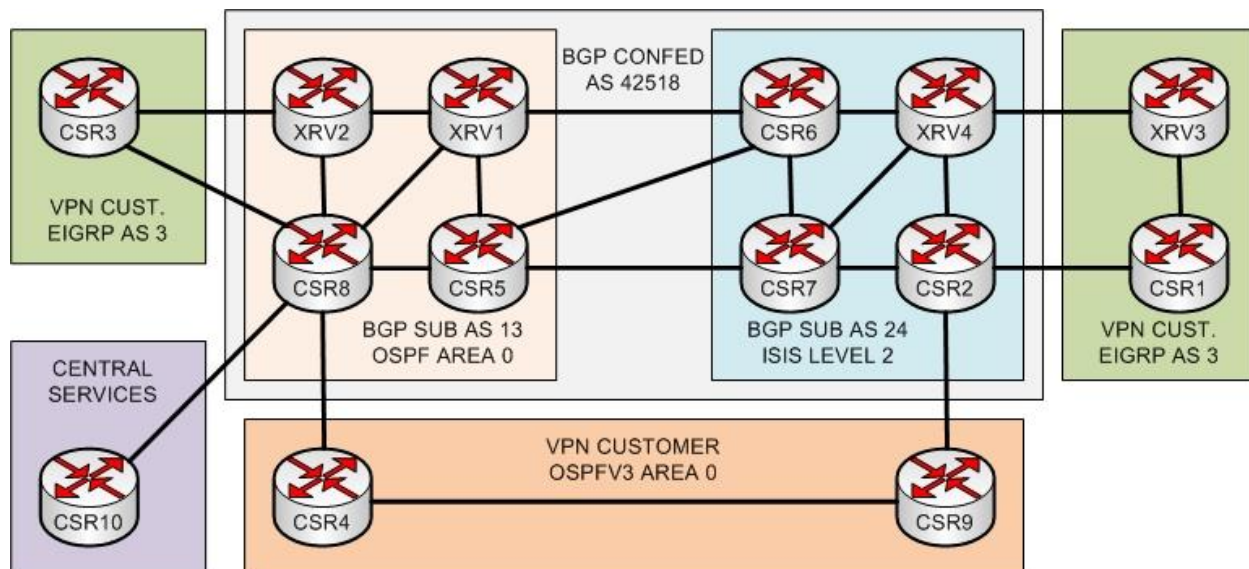
```

8.4 Inter-AS MPLS

This section discusses many options for providing MPLS services across AS boundaries. This includes L3VPN, L2VPN, MVPN, and TE functionality. Not all features are supported for all inter AS options; these details are discussed in subsequent sections. The network diagram includes two ASes. There are 3 transit links between the ASes and the exact configuration of these links changes with each option. There are 3 L3VPN customers, each one using a different routing protocol (OSPFv3, EIGRP, and BGP). The BGP customer is a central services VPN representing the Internet (CSR10). The EIGRP customer has an intra-AS backdoor link as well as a single remote site (CSR3, CSR1, and XRv3). The OSPFv3 customer has an inter-AS backdoor link and is singly-attached to each MPLS provider independently (CSR4/CSR9).



For the majority of tests, the eBGP topology shown above is used. Some of the inter-AS MPLS options support BGP confederations as well. The diagram is almost identical, except that the confederation ASN is 42518 and the existing ASes become sub-ASes. The diagram is shown here for reference but is only relevant for the confederation variations discussed later.



The intra-AS IGP, LDP, TE, and multicast infrastructure configurations don't change much between options so they are verified quickly. The intra-AS configurations are very basic so they are not shown here. Beginning with AS 13, we will view the OSPF database to ensure all links are properly formed. For brevity, I omit the router LSA entries not relevant for this verification. The summary view shows 4 router LSAs within area 0, which is correct. We see 14 opaque-area (type 10) LSAs which make up the TED. Each router creates one of these LSAs for its own node, which totals 4. Then, each node creates another for each link enabled for TE: 3 for CSR8, 2 for CSR5, 2 for XRV2, and 3 for XRV1, for a total of 10. Thus, the total of 14 type-10 LSAs is correct. Since all network types are point-to-point, there are no designated routers (network LSA).

```
R8#show ip ospf 13 0 database database-summary
```

```
OSPF Router with ID (13.0.0.8) (Process ID 13)
```

```
Area 0 database summary
```

LSA Type	Count	Delete	Maxage
Router	4	0	0
Network	0	0	0
Summary Net	0	0	0
Summary ASBR	0	0	0
Type-7 Ext	0	0	0
Prefixes redistributed in Type-7 0			
Opaque Link	0	0	0
Opaque Area	14	0	0
Subtotal	18	0	0

Each router has the proper point-to-point connections within the area as verified below. Although this output is long, it is a very fast and accurate way of checking the OSPF connectivity within an area.


```

R8#show ip ospf 13 0 database router | include Advertising|Neighboring_Router
Advertising Router: 13.0.0.5
  (Link ID) Neighboring Router ID: 13.0.0.11
  (Link ID) Neighboring Router ID: 13.0.0.8
Advertising Router: 13.0.0.8
  (Link ID) Neighboring Router ID: 13.0.0.11
  (Link ID) Neighboring Router ID: 13.0.0.12
  (Link ID) Neighboring Router ID: 13.0.0.5
Advertising Router: 13.0.0.11
  (Link ID) Neighboring Router ID: 13.0.0.12
  (Link ID) Neighboring Router ID: 13.0.0.5
  (Link ID) Neighboring Router ID: 13.0.0.8
Advertising Router: 13.0.0.12
  (Link ID) Neighboring Router ID: 13.0.0.11
  (Link ID) Neighboring Router ID: 13.0.0.8

```

We will also verify that the link between XRv1 and XRv2 has a higher cost. This will influence the traffic forwarding patterns for the LSPs tested later.

```

R8#show ip ospf 13 0 database router 13.0.0.11 | begin 13.0.0.12
(Link ID) Neighboring Router ID: 13.0.0.12
(Link Data) Router Interface address: 13.11.12.11
Number of MTID metrics: 0
TOS 0 Metrics: 50
[snip]

```

```

R8#show ip ospf 13 0 database router 13.0.0.12 | begin 13.0.0.11
(Link ID) Neighboring Router ID: 13.0.0.11
(Link Data) Router Interface address: 13.11.12.12
Number of MTID metrics: 0
TOS 0 Metrics: 50
[snip]

```

Next, we can verify that OSPF is carrying the loopbacks between the routers. We will look at the OSPF RIB to see them. From CSR8's perspective, one prefix is connected while the other 3 are OSPF-learned.

```

R8#show ip ospf rib | section /32
*> 13.0.0.5/32, Intra, cost 2, area 0
    via 13.5.8.5, GigabitEthernet2.558
*   13.0.0.8/32, Intra, cost 1, area 0, Connected
    via 13.0.0.8, Loopback0
*> 13.0.0.11/32, Intra, cost 2, area 0
    via 13.8.11.11, GigabitEthernet2.581
*> 13.0.0.12/32, Intra, cost 2, area 0
    via 13.8.12.12, GigabitEthernet2.582

```

Reachability to these loopbacks implies that LDP neighbors can form, assuming it is enabled. We ensure that LDP is enabled on all interfaces on XRv1 and CSR8 routers, then verify the LDP peers. Since XRv1

and CSR8 have the most interfaces (connect to all other nodes in the area), we will assume LDP is fully functional in verifying only those routers.

```
R8#show mpls interfaces
```

Interface	IP	Tunnel	BGP	Static	Operational
GigabitEthernet2.582	Yes (ldp)	Yes	No	No	Yes
GigabitEthernet2.558	Yes (ldp)	Yes	No	No	Yes
GigabitEthernet2.581	Yes (ldp)	Yes	No	No	Yes

```
RP/0/0/CPU0:XRv1#show mpls interfaces
```

Interface	LDP	Tunnel	Static	Enabled
GigabitEthernet0/0/0/0.521	Yes	Yes	No	Yes
GigabitEthernet0/0/0/0.551	Yes	Yes	No	Yes
GigabitEthernet0/0/0/0.581	Yes	Yes	No	Yes

```
R8#show mpls ldp neighbor | include Peer_LDP
```

```
Peer LDP Ident: 13.0.0.5:0; Local LDP Ident 13.0.0.8:0
Peer LDP Ident: 13.0.0.12:0; Local LDP Ident 13.0.0.8:0
Peer LDP Ident: 13.0.0.11:0; Local LDP Ident 13.0.0.8:0
```

```
RP/0/0/CPU0:XRv1#show mpls ldp neighbor brief
```

Peer	GR	NSR	Up Time	Discovery		Addresses		Labels	
				ipv4	ipv6	ipv4	ipv6	ipv4	ipv6
13.0.0.12:0	N	N	12:37:23	1	0	3	0	8	0
13.0.0.5:0	N	N	12:37:23	1	0	3	0	9	0
13.0.0.8:0	N	N	12:37:23	1	0	4	0	10	0

A quick look at the CSR8 and XRv1 LFIBs shows that labels have been learned for all remote loopbacks. Most of the time, the label is implicit-null, but this is dependent upon IGP. Since XRv1 routes to XRv2 via CSR8, CSR8's local label for 13.0.0.12/32 is used.

```
RP/0/0/CPU0:XRv1#show mpls forwarding | include Pop
```

91000	Pop	13.0.0.5/32	Gi0/0/0/0.551	13.5.11.5	86546
91001	Pop	13.0.0.8/32	Gi0/0/0/0.581	13.8.11.8	86456

```
RP/0/0/CPU0:XRv1#show mpls forwarding prefix 13.0.0.12/32
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
91002	8012	13.0.0.12/32	Gi0/0/0/0.581	13.8.11.8	701073

```
R8#show mpls forwarding-table | include Pop
```

8003	Pop Label	13.0.0.5/32	866602	Gi2.558	13.5.8.5
8012	Pop Label	13.0.0.12/32	1761822	Gi2.582	13.8.12.12
8014	Pop Label	13.0.0.11/32	835899	Gi2.581	13.8.11.11

We can see label 8012 being used when XRv1 sends traffic towards XRv2. This quick check shows us that MPLS forwarding is operational.

```
RP/0/0/CPU0:XRv1#traceroute 13.0.0.12 source 13.0.0.11
Type escape sequence to abort.
Tracing the route to 13.0.0.12
 1 13.8.11.8 [MPLS: Label 8012 Exp 0] 9 msec 0 msec 0 msec
 2 13.8.12.12 29 msec 0 msec 0 msec
```

Next, we will verify the TED. Like OSPF, we will look at the significant topology components such as vertices and edges. Since there is no fancy TE at this time, we don't need to perform a detailed verification. We simply need to ensure TE is enabled on all nodes and all links within the AS. The output is very similar to the OSPF database information we saw earlier, where we have 4 vertices and 10 edges (if you count each edge unidirectionally). These 14 lines of output map to the 14 type-10 LSAs we counted earlier.

```
R8#show mpls traffic-eng topology brief | include IGP_Id
IGP Id: 13.0.0.5, MPLS TE Id:13.0.0.5 Router Node (ospf 13 area 0)
  link[0]: Point-to-Point, Nbr IGP Id: 13.0.0.8, nbr_node_id:18, gen:71
  link[1]: Point-to-Point, Nbr IGP Id: 13.0.0.11, nbr_node_id:20, gen:71
IGP Id: 13.0.0.8, MPLS TE Id:13.0.0.8 Router Node (ospf 13 area 0)
  link[0]: Point-to-Point, Nbr IGP Id: 13.0.0.5, nbr_node_id:19, gen:62
  link[1]: Point-to-Point, Nbr IGP Id: 13.0.0.11, nbr_node_id:20, gen:62
  link[2]: Point-to-Point, Nbr IGP Id: 13.0.0.12, nbr_node_id:21, gen:62
IGP Id: 13.0.0.11, MPLS TE Id:13.0.0.11 Router Node (ospf 13 area 0)
  link[0]: Point-to-Point, Nbr IGP Id: 13.0.0.5, nbr_node_id:19, gen:68
  link[1]: Point-to-Point, Nbr IGP Id: 13.0.0.8, nbr_node_id:18, gen:68
  link[2]: Point-to-Point, Nbr IGP Id: 13.0.0.12, nbr_node_id:21, gen:68
IGP Id: 13.0.0.12, MPLS TE Id:13.0.0.12 Router Node (ospf 13 area 0)
  link[0]: Point-to-Point, Nbr IGP Id: 13.0.0.8, nbr_node_id:18, gen:69
  link[1]: Point-to-Point, Nbr IGP Id: 13.0.0.11, nbr_node_id:20, gen:69
```

We also verify that RSVP is enabled so that TE LSPs can be properly signaled. I limit the verification to CSR8 and XRv1 as they have the most links.

```
RP/0/0/CPU0:XRv1#show rsvp interface
*: RDM: Default I/F B/W % : 75% [default] (max resv/bc0), 0% [default] (bc1)
Interface    MaxBW (bps)    MaxFlow (bps)    Allocated (bps)    MaxSub (bps)
-----
Gi0/0/0/0.521    200M          200M          0 ( 0%)          0
Gi0/0/0/0.551    200M          200M          0 ( 0%)          0
Gi0/0/0/0.581    200M          200M          0 ( 0%)          0
```

```
R8#show ip rsvp interface
interface    rsvp    allocated    i/f max    flow max    sub max    VRF
Gi2          ena      0           750M       750M       0
Gi2.558      ena      0           200M       200M       0
```

Gi2.581	ena	0	200M	200M	0
Gi2.582	ena	0	200M	200M	0

Next, we verify the intra-AS multicast network. XRv2 and CSR2 are the RPs for all groups, and each AS uses BSR internally to disseminate RP information. For brevity, I simply verify CSR8 and XRv1 PIM neighbors to ensure they are up. Since PIM neighbors can form unidirectionally, I verify PIM neighbors on all devices but only show output from two routers.

```
R8#show ip pim neighbor | begin ^Neighbor
```

Neighbor Address	Interface	Uptime/Expires	Ver	DR Prio/Mode
13.5.8.5	GigabitEthernet2.558	12:05:52/00:01:36	v2	1 / S P G
13.8.11.11	GigabitEthernet2.581	12:03:33/00:01:40	v2	1 / DR P G
13.8.12.12	GigabitEthernet2.582	12:02:58/00:01:21	v2	1 / DR P G

```
RP/0/0/CPU0:XRv1#show pim neighbor | begin ^Neighbor
```

Neighbor Address	Interface	Uptime	Expires	DR pri	Flags
13.11.12.11*	GigabitEthernet0/0/0/0.521	12:04:30	00:01:29	1	B P E
13.11.12.12	GigabitEthernet0/0/0/0.521	12:03:50	00:01:36	1 (DR)	B P
13.5.11.5	GigabitEthernet0/0/0/0.551	12:04:24	00:01:33	1	P
13.5.11.11*	GigabitEthernet0/0/0/0.551	12:04:30	00:01:22	1 (DR)	B P E
13.8.11.8	GigabitEthernet0/0/0/0.581	12:04:25	00:01:17	1	P
13.8.11.11*	GigabitEthernet0/0/0/0.581	12:04:30	00:01:18	1 (DR)	B P E
13.0.0.11*	Loopback0	12:04:30	00:01:26	1 (DR)	B P E

For brevity, I also ensure that the RP information is being distributed. Both CSR8 and XRv4 learn the RP information via BSR within their respective ASes.

```
R8#show ip pim rp mapping
```

PIM Group-to-RP Mappings

Group(s) 224.0.0.0/4

RP 13.0.0.12 (?), v2

Info source: 13.0.0.12 (?), via bootstrap, priority 192, holdtime 150

Uptime: 11:49:43, expires: 00:02:08

```
RP/0/0/CPU0:XRv4#show pim rp mapping
```

PIM Group-to-RP Mappings

Group(s) 224.0.0.0/4

RP 24.0.0.2 (?), v2

Info source: 24.2.14.2 (?), elected via bsr, priority 0, holdtime 150

Uptime: 11:51:13, expires: 00:01:45

Before looking at the BGP topology related to VPNs, we will perform the same set of verifications in AS 24. Beginning with IS-IS, we look at the LSP details to see each vertex and its associated edges in the SPF graph. Unlike OSPFv2 in AS 13, IS-IS is running multi-topology IPv6 routing as well. It doesn't contribute to the inter-AS testing but is enabled for any future excursions using this topology. As such, the IS-IS LSPs

are larger to account for the multi-topology (MT) links for IPv6. CSR2 and CSR6 have 2 IPv4 peers, 2 IPv6 peers, and one loopback address. CSR7 and XRv4 have 3 IPv4 peers, 3 IPv6 peers, and one loopback address. We also see that for IPv4 only, the link between CSR2 and CSR7 has a high cost. With this single command, we can verify the most critical parts of the IS-IS topology from a single router. With OSPF, we used multiple commands for variety.

```
RP/0/0/CPU0:XRv4#show isis database detail | utility egrep '^[RX]|Extended'
```

```
R2.00-00          0x0000004f  0x14df          743          0/0/0
Metric: 50         IS-Extended R7.00
Metric: 10         IS-Extended XRv4.00
Metric: 10         MT (IPv6 Unicast) IS-Extended R7.00
Metric: 10         MT (IPv6 Unicast) IS-Extended XRv4.00
Metric: 0          IP-Extended 24.0.0.2/32

R6.00-00          0x0000004f  0xfbd5          819          0/0/0
Metric: 10         IS-Extended R7.00
Metric: 10         MT (IPv6 Unicast) IS-Extended R7.00
Metric: 10         MT (IPv6 Unicast) IS-Extended XRv4.00
Metric: 10         IS-Extended XRv4.00
Metric: 0          IP-Extended 24.0.0.6/32

R7.00-00          0x0000004e  0xb89f          658          0/0/0
Metric: 10         IS-Extended R6.00
Metric: 50         IS-Extended R2.00
Metric: 10         MT (IPv6 Unicast) IS-Extended R6.00
Metric: 10         MT (IPv6 Unicast) IS-Extended R2.00
Metric: 10         MT (IPv6 Unicast) IS-Extended XRv4.00
Metric: 10         IS-Extended XRv4.00
Metric: 0          IP-Extended 24.0.0.7/32

XRv4.00-00        * 0x0000004a  0x707e          1152         0/0/0
Metric: 10         IS-Extended R2.00
Metric: 10         IS-Extended R6.00
Metric: 10         IS-Extended R7.00
Metric: 0          IP-Extended 24.0.0.14/32
Metric: 10         MT (IPv6 Unicast) IS-Extended R2.00
Metric: 10         MT (IPv6 Unicast) IS-Extended R6.00
Metric: 10         MT (IPv6 Unicast) IS-Extended R7.00
```

We quickly verify that LDP is enabled on all interfaces and that LDP neighbors have formed. Because CSR7 and XRv4 have links to all nodes, we limit our verification to those routers.

```
RP/0/0/CPU0:XRv4#show mpls interfaces
```

Interface	LDP	Tunnel	Static	Enabled
GigabitEthernet0/0/0/0.524	Yes	Yes	No	Yes
GigabitEthernet0/0/0/0.564	Yes	Yes	No	Yes
GigabitEthernet0/0/0/0.574	Yes	Yes	No	Yes

```
R7#show mpls interfaces
```

Interface	IP	Tunnel	BGP	Static	Operational

GigabitEthernet2.567	Yes (ldp)	Yes	No	No	Yes
GigabitEthernet2.527	Yes (ldp)	Yes	No	No	Yes
GigabitEthernet2.574	Yes (ldp)	Yes	No	No	Yes

RP/0/0/CPU0:XRv4#show mpls ldp neighbor brief

Peer	GR	NSR	Up Time	Discovery		Addresses		Labels	
				ipv4	ipv6	ipv4	ipv6	ipv4	ipv6
24.0.0.2:0	N	N	13:16:37	1	0	3	0	7	0
24.0.0.6:0	N	N	13:16:37	1	0	3	0	6	0
24.0.0.7:0	N	N	13:16:37	1	0	4	0	7	0

R7#show mpls ldp neighbor | include Peer_LDP

```

Peer LDP Ident: 24.0.0.2:0; Local LDP Ident 24.0.0.7:0
Peer LDP Ident: 24.0.0.6:0; Local LDP Ident 24.0.0.7:0
Peer LDP Ident: 24.0.0.14:0; Local LDP Ident 24.0.0.7:0

```

To verify label exchanges, I will check CSR7's LIB for all of the remote loopbacks. CSR7 has learned labels for all relevant prefixes from all LDP peers, so we are fairly certain LDP is configured correctly.

R7#show mpls ldp bindings 24.0.0.6 32

```

lib entry: 24.0.0.6/32, rev 14
  local binding:  label: 7004
  remote binding: lsr: 24.0.0.2:0, label: 2002
  remote binding: lsr: 24.0.0.6:0, label: imp-null
  remote binding: lsr: 24.0.0.14:0, label: 94008

```

R7#show mpls ldp bindings 24.0.0.14 32

```

lib entry: 24.0.0.14/32, rev 20
  local binding:  label: 7002
  remote binding: lsr: 24.0.0.6:0, label: 6003
  remote binding: lsr: 24.0.0.2:0, label: 2000
  remote binding: lsr: 24.0.0.14:0, label: imp-null

```

R7#show mpls ldp bindings 24.0.0.2 32

```

lib entry: 24.0.0.2/32, rev 12
  local binding:  label: 7000
  remote binding: lsr: 24.0.0.2:0, label: imp-null
  remote binding: lsr: 24.0.0.6:0, label: 6004
  remote binding: lsr: 24.0.0.14:0, label: 94009

```

A quick traceroute shows a small LSP within the network, which shows that MPLS imposition works.

R2#traceroute 24.0.0.7 source 24.0.0.2

Type escape sequence to abort.

Tracing the route to 24.0.0.7

VRF info: (vrf in name/id, vrf out name/id)

```

 1 24.2.14.14 [MPLS: Label 94010 Exp 0] 5 msec 4 msec 4 msec

```

2 24.7.14.7 5 msec 4 msec 5 msec

Next, I verify the TED using the same technique we used earlier. The command is identical on XR and we use XRv4 for this verification. The IGP IDs are IS-IS NETs versus OSPF RIDs. The TE ID is still a dotted-decimal number derived from the loopback0 assigned to TE. CSR2 and CSR6 have 2 links while CSR7 and XRv4 have 3 links, which is correct.

```
RP/0/0/CPU0:XRv4#show mpls traffic-eng topology brief | include IGP Id
IGP Id: 0000.0000.0002.00, MPLS TE Id: 24.0.0.2 Router Node (IS-IS 24 level-2)
  Link[0]:Point-to-Point, Nbr IGP Id:0000.0000.0007.00, Nbr Node Id:4, gen:9668
  Link[1]:Point-to-Point, Nbr IGP Id:0000.0000.0014.00, Nbr Node Id:1, gen:9669
IGP Id: 0000.0000.0006.00, MPLS TE Id: 24.0.0.6 Router Node (IS-IS 24 level-2)
  Link[0]:Point-to-Point, Nbr IGP Id:0000.0000.0007.00, Nbr Node Id:4, gen:9666
  Link[1]:Point-to-Point, Nbr IGP Id:0000.0000.0014.00, Nbr Node Id:1, gen:9667
IGP Id: 0000.0000.0007.00, MPLS TE Id: 24.0.0.7 Router Node (IS-IS 24 level-2)
  Link[0]:Point-to-Point, Nbr IGP Id:0000.0000.0006.00, Nbr Node Id:2, gen:9670
  Link[1]:Point-to-Point, Nbr IGP Id:0000.0000.0002.00, Nbr Node Id:3, gen:9671
  Link[2]:Point-to-Point, Nbr IGP Id:0000.0000.0014.00, Nbr Node Id:1, gen:9672
IGP Id: 0000.0000.0014.00, MPLS TE Id: 24.0.0.14 Router Node (IS-IS 24 level-2)
  Link[0]:Point-to-Point, Nbr IGP Id:0000.0000.0002.00, Nbr Node Id:3, gen:9663
  Link[1]:Point-to-Point, Nbr IGP Id:0000.0000.0006.00, Nbr Node Id:2, gen:9664
  Link[2]:Point-to-Point, Nbr IGP Id:0000.0000.0007.00, Nbr Node Id:4, gen:9665
```

In order for paths to be signaled after PCALC completion, RSVP must be enabled. We quickly check this on XRv4 and CSR7 for brevity as they have the most links.

```
RP/0/0/CPU0:XRv4#show rsvp interface
*: RDM: Default I/F B/W % : 75% [default] (max resv/bc0), 0% [default] (bc1)
Interface    MaxBW (bps)   MaxFlow (bps) Allocated (bps)      MaxSub (bps)
-----
Gi0/0/0/0.524      200M          200M          0 ( 0%)              0
Gi0/0/0/0.564      200M          200M          0 ( 0%)              0
Gi0/0/0/0.574      200M          200M          0 ( 0%)              0
```

```
R7#show ip rsvp interface
interface    rsvp      allocated  i/f max  flow max  sub max  VRF
Gi2          ena        0          750M     750M      0
Gi2.527      ena        0          200M     200M      0
Gi2.567      ena        0          200M     200M      0
Gi2.574      ena        0          200M     200M      0
```

Next, we will verify the multicast configuration. Both XRv4 and CSR7 have 3 PIM neighbors which is a good indication that PIM is properly configured on all links.

```
RP/0/0/CPU0:XRv4#show pim neighbor | begin ^Neighbor
Neighbor Address  Interface          Uptime    Expires  DR pri  Flags
24.2.14.2         GigabitEthernet0/0/0/0.524 12:28:12  00:01:19 1      P
24.2.14.14*       GigabitEthernet0/0/0/0.524 12:29:57  00:01:31 1 (DR) B P E
```

```

24.6.14.6      GigabitEthernet0/0/0/0.564 12:28:55 00:01:24 1      P
24.6.14.14*   GigabitEthernet0/0/0/0.564 12:29:57 00:01:24 1 (DR) B P E
24.7.14.7      GigabitEthernet0/0/0/0.574 12:27:09 00:01:20 1      P
24.7.14.14*   GigabitEthernet0/0/0/0.574 12:29:57 00:01:38 1 (DR) B P E
24.0.0.14*    Loopback0                      12:29:57 00:01:35 1 (DR) B P E

R7#show ip pim neighbor | begin ^Neighbor
Neighbor      Interface      Uptime/Expires      Ver      DR
Address
24.2.7.2      GigabitEthernet2.527 12:27:30/00:01:18 v2      1 / S P G
24.6.7.6      GigabitEthernet2.567 12:27:30/00:01:19 v2      1 / S P G
24.7.14.14    GigabitEthernet2.574 12:27:27/00:01:17 v2      1 / DR P G

```

CSR2 is the BSR/RP for AS 24, and we check CSR7 and XRv4 to verify this.

```

R7#show ip pim rp mapping
PIM Group-to-RP Mappings
Group(s) 224.0.0.0/4
  RP 24.0.0.2 (?), v2
    Info source: 24.0.0.2 (?), via bootstrap, priority 0, holdtime 150
    Uptime: 12:26:05, expires: 00:02:21

RP/0/0/CPU0:XRv4#show ip pim rp mapping
PIM Group-to-RP Mappings
Group(s) 224.0.0.0/4
  RP 24.0.0.2 (?), v2
    Info source: 24.2.14.2 (?), elected via bsr, priority 0, holdtime 150
    Uptime: 12:26:15, expires: 00:02:12

```

The BGP topology varies significantly between the options so we only verify the most basic configurations now. XRv2 and CSR2 are the route-reflectors for all AFIs within their respective ASes for all labs. The RT policies will also change quite a bit, but initially, I identify export-only RTs for the BGP VRF on CSR8. The BGP configuration on CSR8 is a basic RR-client that peers to XRv2 and no other devices. VPNv4/v6 are negotiated as a base configuration; VPLS/MVPN AFIs are used as needed for specific options. Within the VPN, CSR8 peers with CSR10 to receive some Internet routes.

```

! CSR8
vrf definition BGP
  rd 13:1
  address-family ipv4
    route-target export 13:1
  address-family ipv6
    route-target export 13:1

router bgp 13
  no bgp default ipv4-unicast
  neighbor 13.0.0.12 remote-as 13

```



```

neighbor 13.0.0.12 password IBGP13
neighbor 13.0.0.12 update-source Loopback0
neighbor 13.0.0.12 timers 10 40

address-family vpnv4
  neighbor 13.0.0.12 activate

address-family vpnv6
  neighbor 13.0.0.12 activate

address-family ipv4 vrf BGP
  neighbor 10.8.10.10 remote-as 100
  neighbor 10.8.10.10 activate

address-family ipv6 vrf BGP
  neighbor FD00:10:8:10::10 remote-as 100
  neighbor FD00:10:8:10::10 activate

```

The RR configuration is also straightforward. I only show the configuration to CSR8, but the other routers in the AS use an identical configuration.

```

! XRv2
router bgp 13
  bgp cluster-id 13.0.0.12
  address-family vpnv4 unicast

  address-family vpnv6 unicast

  af-group VPNV4 address-family vpnv4 unicast
    route-reflector-client

  af-group VPNV6 address-family vpnv6 unicast
    route-reflector-client

  session-group IBGP
    remote-as 13
    timers 10 40
    password encrypted 11203B22274358
    update-source Loopback0

neighbor 13.0.0.8
  use session-group IBGP
  address-family vpnv4 unicast
    use af-group VPNV4

  address-family vpnv6 unicast
    use af-group VPNV6

```

Checking CSR8, we can see it has a BGP neighbor with XRv2 for VPNv4/v6 (yellow). It also VRF-aware IPv4/v6 neighbors with CSR10 (green), but the output doesn't make this explicit.

```
R8#show bgp vpnv4 unicast all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down  State/PfxRcd
10.8.10.10    4      100     874     887      86    0    0 13:08:18         4
13.0.0.12     4       13    5319    5585      86    0    0 14:31:03         8
```

```
R8#show bgp vpnv6 unicast all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down  State/PfxRcd
13.0.0.12     4       13    5319    5585     163    0    0 14:31:06         7
FD00:10:8:10::10  4      100     881     885     163    0    0 13:08:16         4
```

CSR8 learns several IPv4/v6 Internet routes. Checking the details of one IPv4 and one IPv6 route, we can see the proper export RT has been applied. All of the routes are reachable via CSR10, the Internet peering point. This proves that the basic PE-CE routing with CSR10 is functional.

```
R8#show bgp vpnv4 unicast vrf BGP | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 13:1 (default for vrf BGP)
*> 110.0.0.0/32    10.8.10.10              0              0 100 ?
*> 110.0.0.1/32    10.8.10.10              0              0 100 ?
*> 110.0.0.2/32    10.8.10.10              0              0 100 ?
*> 110.0.0.3/32    10.8.10.10              0              0 100 ?
```

```
R8#show bgp vpnv6 unicast vrf BGP | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 13:1 (default for vrf BGP)
*> ::110:0:0:0/128 FD00:10:8:10::10
                                0              0 100 ?
*> ::110:0:0:1/128  FD00:10:8:10::10
                                0              0 100 ?
*> ::110:0:0:2/128  FD00:10:8:10::10
                                0              0 100 ?
*> ::110:0:0:3/128  FD00:10:8:10::10
                                0              0 100 ?
```

```
R8#show bgp vpnv4 unicast vrf BGP 110.0.0.0/32
BGP routing table entry for 13:1:110.0.0.0/32, version 18
Paths: (1 available, best #1, table BGP)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  100
  10.8.10.10 (via vrf BGP) from 10.8.10.10 (110.0.0.0)
    Origin incomplete, metric 0, localpref 100, valid, external, best
    Extended Community: RT:13:1
    mpls labels in/out 8016/nolabel
```

```

    rx pathid: 0, tx pathid: 0x0

R8#show bgp vpnv6 unicast vrf BGP ::110:0:0:0/128
BGP routing table entry for [13:1]::110:0:0:0/128, version 153
Paths: (1 available, best #1, table BGP)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  100
  FD00:10:8:10::10 (FE80::10) (via vrf BGP) from FD00:10:8:10::10
  (110.0.0.0)
    Origin incomplete, metric 0, localpref 100, valid, external, best
    Extended Community: RT:13:1
    mpls labels in/out 8009/nolabel
    rx pathid: 0, tx pathid: 0x0

```

XRv2, as the RR, will learn and maintain these routes. Since they have not been imported into a VRF, we can reference them by RD since that is the mechanism by which BGP differentiates VPN routes. This proves that the intra-AS VPNv4/v6 advertisement is functional.

```

RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast rd 13:1 | begin Network
  Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 13:1
*>i110.0.0.0/32      13.0.0.8              0      100      0 100 ?
*>i110.0.0.1/32      13.0.0.8              0      100      0 100 ?
*>i110.0.0.2/32      13.0.0.8              0      100      0 100 ?
*>i110.0.0.3/32      13.0.0.8              0      100      0 100 ?

RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast rd 13:1 | begin Network
  Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 13:1
*>i::110:0:0:0/128    13.0.0.8              0      100      0 100 ?
*>i::110:0:0:1/128    13.0.0.8              0      100      0 100 ?
*>i::110:0:0:2/128    13.0.0.8              0      100      0 100 ?
*>i::110:0:0:3/128    13.0.0.8              0      100      0 100 ?

```

The EIGRP VRF inside AS 13 uses XRv2 as a PE. Routes are learned from CSR3 and redistributed into BGP, and vice versa. Since there are no import-RTs configured anywhere, no routes are being exchanged yet.

```

! XRv2
vrf EIGRP
  address-family ipv4 unicast
    export route-target
      13:3

  address-family ipv6 unicast
    export route-target
      13:3

```

```

router eigrp EIGRP
vrf EIGRP
address-family ipv4
log-neighbor-changes
autonomous-system 3
redistribute bgp 13
interface GigabitEthernet0/0/0/0.532

address-family ipv6
log-neighbor-changes
autonomous-system 3
redistribute bgp 13
interface GigabitEthernet0/0/0/0.532

router bgp 13
vrf EIGRP
rd 13:3
address-family ipv4 unicast
redistribute eigrp 3

address-family ipv6 unicast
redistribute eigrp 3

```

XRv2 originates CSR3's EIGRP-learned loopback into BGP, as well as the connected transit link. Since XRv2 is the RR, these routes will not exist anywhere else at present.

```

RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf EIGRP regexp ^$ | begin Network
      Network                Next Hop                Metric LocPrf Weight Path
Route Distinguisher: 13:3 (default for vrf EIGRP)
*> 10.3.3.3/32                10.3.12.3                10880                32768 ?
*> 10.3.12.0/24               0.0.0.0                   0                    32768 ?

RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast vrf EIGRP regexp ^$ | begin Network
      Network                Next Hop                Metric LocPrf Weight Path
Route Distinguisher: 13:3 (default for vrf EIGRP)
*> ::10:3:3:3/128            fe80::3                   10880                32768 ?
*> fd00:10:3:12::/64         ::                        0                    32768 ?

```

On the other side of the network, XRv4 and CSR2 are PEs for VRF EIGRP. CSR2 is also the RR for VPNv4/v6. CSR2's relevant BGP and VRF configurations are shown below; this is nothing complex but is displayed for completeness. On XRv4 is shown as a peer for brevity but CSR6/7 are also configured.

```

! CSR2
vrf definition EIGRP
rd 24:3
address-family ipv4
route-target export 24:3

```

```

address-family ipv6
  route-target export 24:3

router bgp 24
  template peer-session IBGP
    remote-as 24
    password IBGP24
    update-source Loopback0
    timers 10 40
  no bgp default ipv4-unicast
  neighbor 24.0.0.6 inherit peer-session IBGP
  neighbor 24.0.0.7 inherit peer-session IBGP
  neighbor 24.0.0.14 inherit peer-session IBGP

address-family vpnv4
  neighbor 24.0.0.14 activate
  neighbor 24.0.0.14 route-reflector-client

address-family vpnv6
  neighbor 24.0.0.14 activate
  neighbor 24.0.0.14 route-reflector-client

address-family ipv4 vrf EIGRP
  redistribute eigrp 3

address-family ipv6 vrf EIGRP
  redistribute eigrp 3

router eigrp EIGRP
  address-family ipv4 unicast vrf EIGRP autonomous-system 3
  topology base
    redistribute bgp 24
  network 10.1.2.2 0.0.0.0

address-family ipv6 unicast vrf EIGRP autonomous-system 3
  topology base
    redistribute bgp 24

```

Once this is complete (and assuming basic EIGRP has been configured on XRv3 and CSR1, not shown), CSR2 will learn EIGRP routes and redistribute them into BGP while adding RT:24:3. Since EIGRP is in use, all of the other advanced extended communities are applied as well; these are discussed in detail in the multi-VRF CE chapter.

```

R2#show bgp vpnv4 unicast vrf EIGRP | begin Network
      Network           Next Hop           Metric LocPrf Weight Path
Route Distinguisher: 24:3 (default for vrf EIGRP)
*> 10.1.1.1/32          10.1.2.1              10880           32768 ?

```

```
*> 10.1.2.0/24      0.0.0.0      0      32768 ?
```

```
R2#show bgp vpnv4 unicast vrf EIGRP 10.1.1.1/32
BGP routing table entry for 24:3:10.1.1.1/32, version 9
Paths: (1 available, best #1, table EIGRP)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local
    10.1.2.1 (via vrf EIGRP) from 0.0.0.0 (24.0.0.2)
    Origin incomplete, metric 10880, localpref 100, weight 32768, valid,
    sourced, best
    Extended Community: RT:24:3 Cost:pre-bestpath:128:10880 0x8800:32768:0
    0x8801:3:288 0x8802:65281:2560 0x8803:65281:1500 0x8806:0:167837953
    mpls labels in/out 2003/nolabel
    rx pathid: 0, tx pathid: 0x0
```

Next, we look at XRv4. The PE-CE configuration is similar to CSR2 where RT:24:3 is exported and nothing is imported. The routes are exchanged with BGP via redistribution and XRv4 peers with CSR2 inside the VPNv4/v6 AFIs.

```
! XRv4
vrf EIGRP
  address-family ipv4 unicast
    export route-target
    24:3

  address-family ipv6 unicast
    export route-target
    24:3

router eigrp EIGRP
  vrf EIGRP
    address-family ipv4
      log-neighbor-changes
      autonomous-system 3
      redistribute bgp 24
      interface GigabitEthernet0/0/0/0.534

    address-family ipv6
      log-neighbor-changes
      autonomous-system 3
      redistribute bgp 24
      interface GigabitEthernet0/0/0/0.534

router bgp 24
  address-family vpnv4 unicast
  address-family vpnv6 unicast
```

```

neighbor 24.0.0.2
  remote-as 24
  timers 10 40
  password encrypted 08086E69394B51
  update-source Loopback0
  address-family vpnv4 unicast
  address-family vpnv6 unicast

```

```

vrf EIGRP
  rd 24:3
  address-family ipv4 unicast
    redistribute eigrp 3

```

```

  address-family ipv6 unicast
    redistribute eigrp 3

```

XRv4 learns EIGRP routes locally from XRv3 within VRF EIGRP and advertises them to CSR2. Looking at a single route from each AFI, RT:24:3 was applied.

```

RP/0/0/CPU0:XRv4#show bgp vpnv4 unicast vrf EIGRP | begin Network
  Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 24:3 (default for vrf EIGRP)
*> 10.13.13.13/32   10.13.14.13          10752          32768 ?
*> 10.13.14.0/24    0.0.0.0              0              32768 ?

```

```

RP/0/0/CPU0:XRv4#show bgp vpnv6 unicast vrf EIGRP | begin Network
  Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 24:3 (default for vrf EIGRP)
*> ::10:13:13:13/128 fe80::13            10752          32768 ?
*> fd00:10:13:14::/64 ::                  0              32768 ?

```

```

RP/0/0/CPU0:XRv4#show bgp vpnv4 unicast vrf EIGRP 10.13.13.13/32 | begin
Paths
Paths: (1 available, best #1)
  Advertised to peers (in unique update groups):
    24.0.0.2
  Path #1: Received by speaker 0
  Advertised to peers (in unique update groups):
    24.0.0.2
  Local
    10.13.14.13 from 0.0.0.0 (24.0.0.14)
      Origin incomplete, metric 10752, localpref 100, weight 32768, valid,
      redistributed, best, group-best, import-candidate
      Received Path ID 0, Local Path ID 1, version 23
      Extended community: COST:128:128:10752 EIGRP route-info:0x8000:0 EIGRP
AD:3:282 EIGRP RHB:255:1:2560 EIGRP LM:0x0:1:1500 EIGRP VRR:0x0:13.13.13.10
RT:24:3

```

```

RP/0/0/CPU0:XRv4#show bgp vpnv6 unicast vrf EIGRP ::10:13:13:13/128 | begin
Paths
Paths: (1 available, best #1)
  Advertised to peers (in unique update groups):
    24.0.0.2
  Path #1: Received by speaker 0
  Advertised to peers (in unique update groups):
    24.0.0.2
  Local
    fe80::13 from 0.0.0.0 (24.0.0.14)
      Origin incomplete, metric 10752, localpref 100, weight 32768, valid,
      redistributed, best, group-best, import-candidate
      Received Path ID 0, Local Path ID 1, version 16
      Extended community: COST:128:128:10752 EIGRP route-info:0x8000:0 EIGRP
      AD:3:282 EIGRP RHB:255:1:2560 EIGRP LM:0x0:1:1500 EIGRP VRR:0x0:13.13.13.10
      RT:24:3

```

CSR2 retains them within the RD table, but they are not imported into VRF EIGRP on CSR2 yet. We can confirm that CSR2 received these routes from XRv4. The RT values were properly received, but since VRF EIGRP isn't importing this value, MPLS L3VPN connectivity doesn't exist yet.

```

R2#show bgp vpnv4 unicast rd 24:3 10.13.13.13
BGP routing table entry for 24:3:10.13.13.13/32, version 36
Paths: (1 available, best #1, table EIGRP)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local, (Received from a RR-client)
    24.0.0.14 (metric 10) (via default) from 24.0.0.14 (24.0.0.14)
      Origin incomplete, metric 10752, localpref 100, valid, internal, best
      Extended Community: RT:24:3 Cost:pre-bestpath:128:10752 0x8800:32768:0
      0x8801:3:282 0x8802:65281:2560 0x8803:1:1500 0x8806:0:168627469
      Connector Attribute: count=1
      type 1 len 12 value 24:3:24.0.0.14
      mpls labels in/out nlabel/94006
      rx pathid: 0, tx pathid: 0x0

```

```

R2#show bgp vpnv6 unicast rd 24:3 ::10:13:13:13/128
BGP routing table entry for [24:3>::10:13:13:13/128, version 32
Paths: (1 available, best #1, table EIGRP)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local, (Received from a RR-client)
    ::FFFF:24.0.0.14 (metric 10) (via default) from 24.0.0.14 (24.0.0.14)
      Origin incomplete, metric 10752, localpref 100, valid, internal, best
      Extended Community: RT:24:3 Cost:pre-bestpath:128:10752 0x8800:32768:0
      0x8801:3:282 0x8802:65281:2560 0x8803:1:1500 0x8806:0:168627469

```



```
Connector Attribute: count=1
  type 1 len 12 value 24:3:24.0.0.14
mpls labels in/out nolabel/94001
rx pathid: 0, tx pathid: 0x0
```

The OSPF VRF spans multiple ASes with CSR2 and CSR8 as PEs. The configuration on the PEs is nearly identical with the exception of exact RD/RT values. For brevity, only CSR2 is shown.

```
! CSR2
vrf definition OSPF
  rd 24:2
  address-family ipv4
    route-target export 24:2

  address-family ipv6
    route-target export 24:2

router ospfv3 2
  address-family ipv4 unicast vrf OSPF
    redistribute bgp 24
    prefix-suppression

  address-family ipv6 unicast vrf OSPF
    redistribute bgp 24
    prefix-suppression

router bgp 13
  address-family ipv4 vrf OSPF
    redistribute ospfv3 2

  address-family ipv6 vrf OSPF
    redistribute ospf 2

interface GigabitEthernet2.529
  encapsulation dot1Q 3529
  vrf forwarding OSPF
  ip address 10.2.9.2 255.255.255.0
  ipv6 address FE80::2 link-local
  ipv6 address FD00:10:2:9::2/64
  ospfv3 network point-to-point
  ospfv3 2 ipv6 area 0
  ospfv3 2 ipv4 area 0
```

We verify that CSR2 receives OSPF routes from CSR9 and applies RT:24:2. This is true for both IPv4 and IPv6 AFIs.

```
R2#show bgp vpnv4 unicast vrf OSPF 10.9.9.9/32
BGP routing table entry for 24:2:10.9.9.9/32, version 27
```

```

Paths: (1 available, best #1, table OSPF)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local
    10.2.9.9 (via vrf OSPF) from 0.0.0.0 (24.0.0.2)
      Origin incomplete, metric 1, localpref 100, weight 32768, valid,
sourced, best
      Extended Community: RT:24:2 OSPF ROUTER ID:10.2.9.2:0
      OSPF RT:0.0.0.0:2:0
      mpls labels in/out 2011/nolabel
      rx pathid: 0, tx pathid: 0x0

```

```

R2#show bgp vpnv6 unicast vrf OSPF ::10:9:9:9/128
BGP routing table entry for [24:2>::10:9:9:9/128, version 126
Paths: (1 available, best #1, table OSPF)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local
    FE80::9 (FE80::9) (via vrf OSPF) from 0.0.0.0 (24.0.0.2)
      Origin incomplete, metric 1, localpref 100, weight 32768, valid,
sourced, best
      Extended Community: RT:24:2 OSPF ROUTER ID:10.2.9.2:0
      OSPF RT:0.0.0.0:2:0
      mpls labels in/out 2010/nolabel
      rx pathid: 0, tx pathid: 0x0

```

On CSR8, the same is true, except we look at CSR4's local routes. RT:13:2 has been applied during the export process from the VRF into BGP.

```

R8#show bgp vpnv4 unicast vrf OSPF 10.4.4.4/32
BGP routing table entry for 13:2:10.4.4.4/32, version 28
Paths: (1 available, best #1, table OSPF)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local
    10.4.8.4 (via vrf OSPF) from 0.0.0.0 (13.0.0.8)
      Origin incomplete, metric 1, localpref 100, weight 32768, valid,
sourced, best
      Extended Community: RT:13:2 OSPF ROUTER ID:10.4.8.8:0
      OSPF RT:0.0.0.0:2:0
      mpls labels in/out 8000/nolabel
      rx pathid: 0, tx pathid: 0x0

```

```

R8#show bgp vpnv6 unicast vrf OSPF ::10:4:4:4/128
BGP routing table entry for [13:2>::10:4:4:4/128, version 170
Paths: (1 available, best #1, table OSPF)

```

```

Advertised to update-groups:
  1
Refresh Epoch 1
Local
  FE80::4 (FE80::4) (via vrf OSPF) from 0.0.0.0 (13.0.0.8)
    Origin incomplete, metric 1, localpref 100, weight 32768, valid,
sourced, best
    Extended Community: RT:13:2 OSPF ROUTER ID:10.4.8.8:0
    OSPF RT:0.0.0.0:2:0
    mpls labels in/out 8017/nolabel
    rx pathid: 0, tx pathid: 0x0

```

As an RR, XRv2 will retain these OSPF VPN routes despite not having the VRF locally configured. We reference them by RD; this retention allows XRv2 to advertise them to other routers running VPNv4/v6 that may need the prefixes.

```

RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast rd 13:2 10.4.4.4/32 | begin Local
Local, (Received from a RR-client)
  13.0.0.8 (metric 2) from 13.0.0.8 (13.0.0.8)
    Received Label 8000
    Origin incomplete, metric 1, localpref 100, valid, internal, best,
group-best, import-candidate, not-in-vrf
    Received Path ID 0, Local Path ID 1, version 20
    Extended community: OSPF router-id:10.4.8.8 OSPF route-type:0:2:0x0
RT:13:2

```

```

RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast rd 13:2 ::10:4:4:4/128 | begin Local
Local, (Received from a RR-client)
  13.0.0.8 (metric 2) from 13.0.0.8 (13.0.0.8)
    Received Label 8017
    Origin incomplete, metric 1, localpref 100, valid, internal, best,
group-best, import-candidate, not-in-vrf
    Received Path ID 0, Local Path ID 1, version 11
    Extended community: OSPF router-id:10.4.8.8 OSPF route-type:0:2:0x0
RT:13:2

```

Although no advanced MVPN has been configured, VRF-aware PIM has been configured for all L3VPN customers to support C-mcast signaling. First, I verify this on CSR8 for VRF BGP for both IPv4 and IPv6.

```

R8#show ip pim vrf BGP neighbor | begin ^Neigh
Neighbor      Interface      Uptime/Expires    Ver    DR
Address
10.8.10.10    GigabitEthernet2.580    14:36:02/00:01:41 v2      1 / DR S P G

R8#show ipv6 pim vrf BGP neighbor | begin ^Neigh
Neighbor Address      Interface      Uptime    Expires    Mode DR pri
FE80::10      Gi2.580        00:00:11   00:01:34   B G    DR 1

```

Next, I verify the same is true on CSR2 and CSR8 for VRF OSPF. CSR2 peers with CSR9 and CSR8 peers with CSR4 inside the VPN. It is important to verify both IPv4 and IPv6.

```
R2#show ip pim vrf OSPF neighbor | begin ^Neighbor
Neighbor      Interface      Uptime/Expires    Ver    DR
Address
10.2.9.9      GigabitEthernet2.529    00:00:40/00:01:33 v2      1 / DR S P G
```

```
R2#show ipv6 pim vrf OSPF neighbor | begin ^Neighbor
Neighbor Address      Interface      Uptime    Expires    Mode DR pri
FE80::9              Gi2.529       00:00:03   00:01:41   B G    DR 1
```

```
R8#show ip pim vrf OSPF neighbor | begin ^Neighbor
Neighbor      Interface      Uptime/Expires    Ver    DR
Address
10.4.8.4      GigabitEthernet2.548    00:00:36/00:01:37 v2      1 / S P G
```

```
R8#show ipv6 pim vrf OSPF neighbor | begin ^Neighbor
Neighbor Address      Interface      Uptime    Expires    Mode DR pri
FE80::4          Gi2.548       00:00:15   00:01:30   B G      1
```

Last, we verify PIM neighbors within VRF EIGRP. We see that each PE has exactly one neighbor as expected with each AFI.

```
RP/0/0/CPU0:XRv2#show pim vrf EIGRP neighbor | begin ^Neigh
Neighbor Address      Interface      Uptime    Expires    DR pri    Flags
10.3.12.3             GigabitEthernet0/0/0/0.532 14:36:12   00:01:35   1          P
10.3.12.12*           GigabitEthernet0/0/0/0.532 14:36:30   00:01:19   1 (DR)    B P E
```

```
RP/0/0/CPU0:XRv2#show pim vrf EIGRP ipv6 neighbor | begin ^Neigh
Neighbor Address      Uptime    Expires    DR pri    DR    Flags
fe80::3              00:00:03   00:01:41   1          B
fe80::12*            00:00:40   00:01:44   1          (DR) B P
```

```
RP/0/0/CPU0:XRv4#show pim vrf EIGRP neighbor | begin ^Neigh
Neighbor Address      Interface      Uptime    Expires    DR pri    Flags
10.13.14.13          GigabitEthernet0/0/0/0.534 00:07:01   00:01:22   1          B P
10.13.14.14*         GigabitEthernet0/0/0/0.534 14:39:22   00:01:43   1 (DR)    B P E
```

```
RP/0/0/CPU0:XRv4#show pim vrf EIGRP ipv6 neighbor | begin ^Neigh
Neighbor Address      Uptime    Expires    DR pri    DR    Flags
fe80::13             00:07:14   00:01:25   1          B P
fe80::14*            14:39:36   00:01:26   1          (DR) B P
```

```
R2#show ip pim vrf EIGRP neighbor | begin ^Neighbor
Neighbor      Interface      Uptime/Expires    Ver    DR
Address
Prio/Mode
```

10.1.2.1 GigabitEthernet2.512 00:07:44/00:01:21 v2 1 / S P G

```
R2#show ipv6 pim vrf EIGRP neighbor | begin ^Neighbor
```

Neighbor Address	Interface	Uptime	Expires	Mode	DR	pri
FE80::1	Gi2.512	00:04:22	00:01:18	B G		1

The last component to configure/verify are the L2VPN pieces. Since the L2VPN topology will change many timers between the options, I limit this to just the PE-CE access configurations. The VRF for VPLS on the CE routers is just used for ping testing so that there is no interference with the L3VPN/MVPN verification.

```
! CSR3
interface GigabitEthernet2.538
 encapsulation dot1Q 3538
 vrf forwarding VPLS
 ip address 10.0.0.3 255.255.255.0
 ipv6 address FE80::3 link-local

! CSR8
interface GigabitEthernet2
 service instance 3 ethernet
 encapsulation dot1q 3538 exact
 rewrite ingress tag pop 1 symmetric
```

On the other side of the network, the configuration is similar except different dot1q tags are used.

```
! CSR1
interface GigabitEthernet2.5123
 encapsulation dot1Q 3512 second-dot1q 3
 vrf forwarding VPLS
 ip address 10.0.0.1 255.255.255.0
 ipv6 address FE80::1 link-local

! CSR2
interface GigabitEthernet2
 service instance 3 ethernet
 encapsulation dot1q 3512 second-dot1q 3
 rewrite ingress tag pop 2 symmetric
```

A quick check shows that the service instances were configured correctly and are operational.

```
R2#show ethernet service instance
Identifier  Type      Interface      State      CE-Vlans
3           Static    GigabitEthernet2 Up
```

```
R8#show ethernet service instance
Identifier  Type      Interface      State      CE-Vlans
```

The following configuration is generic and is applied to any L2VPN PW termination point. The template enables the control word and sequence numbers. L2VPN logging is always nice to enable as well. This is on CSR2 and CSR8 initially, but may be added to other routers depending on the test.

```
! CSR2 and CSR8, others in the future
template type pseudowire TMP_VPLS
  encapsulation mpls
  sequencing both
  control-word include

l2vpn
  logging pseudowire status
```

The transit links have not been configured at this point. Like the VPN details, these change based on the option used.

8.4.1 Option A (Back to back VRF exchange)

Option A permits inter-AS MPLS connectivity by treating the ASBRs as ordinary PEs. These PEs will impose labels for ingress traffic and remote labels for egress traffic, just like a normal PE. The inter-AS traffic is therefore not MPLS encapsulated, which implies the ASBRs just treat one another as CE routers. VRF-aware BGP for IPv4/v6 exchanges allow routes to be exchanged back and forth, and extended-communities could be exchanged to keep certain attributes (OSPF/EIGRP custom communities, RTs, etc) intact during this exchange. The inter-AS traffic could be MPLS encapsulated for certain CSC architecture, which is a variation of this design discussed later.

There are many benefits to Option A. It is the simplest inter-AS MPLS option as it does not require any coordination of RDs, RTs, or other VPN information between providers. It is used in the vast majority of real-life deployments for this reason as it “just works”. It introduces no new technology into the SP network and works for all MPLS services (L3VPN, L2VPN, etc). The drawbacks of this option are that it scales poorly; for each customer VPN requiring inter-AS service, a new VRF (mapped to a specific interface) must be created on the ASBRs. This reduces the transparency of Option A and makes it very configuration-intensive. Many routers also have a limit on the number of BGP sessions they can support since each inter-AS connection is a new BGP peer. It also implies that LSPs are not end-to-end since the inter-AS transit traffic is regular IPv4/v6 (unless CSC variations are applied).

Additional Reading – Reference configurations “inter-as-mpls-a”

8.4.1.1 L3VPN

Before configuring Option A L3VPN, we will add basic import RTs to the existing VRFs. I added export-only RTs earlier just to demonstrate VRF-to-BGP route exports. Because this is a very basic task, I show a few examples for brevity. The exported RTs are identical to the imported RTs with option A since the RT

values don't need to be exchanged between ASes. CSR8 shows the central services RT being imported into VRF OSPF and the OSPF/EIGRP RT being imported into the central services VRF.

```
! XRv4
vrf EIGRP
  address-family ipv4 unicast
    import route-target
      24:3
    export route-target
      24:3

  address-family ipv6 unicast
    import route-target
      24:3
    export route-target
      24:3

! CSR8
vrf definition BGP
  rd 13:1
  address-family ipv4
    route-target export 13:1
    route-target import 13:3
    route-target import 13:2
  address-family ipv6
    route-target export 13:1
    route-target import 13:3
    route-target import 13:2

vrf definition OSPF
  rd 13:2
  address-family ipv4
    route-target export 13:2
    route-target import 13:2
    route-target import 13:1
  address-family ipv6
    route-target export 13:2
    route-target import 13:2
    route-target import 13:1
```

As a quick test, this means that CSR1 and XRv3 should have reachability over MPLS to one another (the backdoor link is currently down). CSR2 shows an iBGP route for 10.13.13.13/32 via XRv4's loopback using VPN label 94006. These routers had VPNv4/v6 configured in the basic configuration earlier and it is not related to inter-AS MPLS at all.

```
R2#show bgp vpnv4 unicast vrf EIGRP 10.13.13.13/32
BGP routing table entry for 24:3:10.13.13.13/32, version 36
Paths: (1 available, best #1, table EIGRP)
```

Advertised to update-groups:

1

Refresh Epoch 1

Local, (Received from a RR-client)

24.0.0.14 (metric 10) (via default) from 24.0.0.14 (24.0.0.14)
Origin incomplete, metric 10752, localpref 100, valid, internal, best
Extended Community: RT:24:3 Cost:pre-bestpath:128:10752 0x8800:32768:0
0x8801:3:282 0x8802:65281:2560 0x8803:1:1500 0x8806:0:168627469
Connector Attribute: count=1
type 1 len 12 value 24:3:24.0.0.14
mpls labels in/out nlabel/94006
rx pathid: 0, tx pathid: 0x0

The route to the BGP next-hop is an IGP route, so an LDP label is used. The routers are directly connected, making CSR2 the ingress LSR and PHP LSR, so only the VPN label is imposed.

R2#show ip route 24.0.0.14

Routing entry for 24.0.0.14/32

Known via "isis", distance 115, metric 10, type level-2

Redistributing via isis 24

Last update from 24.2.14.14 on GigabitEthernet2.524, 18:09:15 ago

Routing Descriptor Blocks:

* 24.2.14.14, from 24.0.0.14, 18:09:15 ago, via GigabitEthernet2.524
Route metric is 10, traffic share count is 1

R2#show mpls ldp bindings 24.0.0.14 32 neighbor 24.0.0.14

lib entry: 24.0.0.14/32, rev 15

remote binding: lsr: 24.0.0.14:0, label: imp-null

Thanks to the EIGRP extended communities, this is seen as an internal route on CSR1. The MPLS network collapses into a single, logical EIGRP router in this way. Traceroute reveals the VPN label of 94006 and the sites have reachability over MPLS. For brevity, I only one direction.

R1#show ip route 10.13.13.13

Routing entry for 10.13.13.13/32

Known via "eigrp 3", distance 90, metric 15880, type internal

Redistributing via eigrp 3

Last update from 10.1.2.2 on GigabitEthernet2.512, 16:08:52 ago

Routing Descriptor Blocks:

* 10.1.2.2, from 10.1.2.2, 16:08:52 ago, via GigabitEthernet2.512
Route metric is 15880, traffic share count is 1
Total delay is 21 microseconds, minimum bandwidth is 1000000 Kbit
Reliability 255/255, minimum MTU 1500 bytes
Loading 1/255, Hops 2

R1#traceroute 10.13.13.13 source 10.1.1.1

Type escape sequence to abort.

Tracing the route to 10.13.13.13


```
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.2.2 7 msec 3 msec 4 msec
 2 24.2.14.14 [MPLS: Label 94006 Exp 0] 6 msec 6 msec 4 msec
 3 10.13.14.13 4 msec 9 msec 13 msec
```

We will continue with the inter-AS configurations next. To support Option A with our baseline network, we must accomplish these general tasks:

1. Configure ASBRs as PEs for all VRFs that require inter-AS service; adjust RTs as needed
2. Create a new subinterface for each customer VRF to be exchanged
3. Configure IPv4/v6 BGP sessions between ASBRs within the VRF
4. Configure BGP VPNv4/v6 to all ASBRs within each AS

None of these tasks are particularly difficult and no new features are introduced. Beginning with the first task, we must configure VRFs OSPF and EIGRP on XRv1, CSR5, CSR6, and CSR7 (ASBRs). We could optionally configure VRF BGP inside AS 24 as well, but there is a simpler way to do inter-AS option A central services; this is discussed later. For simplicity, each AS uses RT's in the format RT:ASN:X where ASN is the BGP AS number and X is a number based on the VRF. BGP is 1, OSPF is 2, and EIGRP is 3. The RT's could be the same, but since I am exchanging extended-communities between the ASes to demonstrate other features, I make them different for clarity. For brevity, I only show CSR6 and XRv1, since the configurations on the other ASBRs are almost identical. Notice that XRv1 imports the central services RT into each VRF; this effectively allows the remote EIGRP and OSPF routers in AS 24 to access the central service networks. We do not have to extend VRF BGP into AS 24, which adds complexity.

```
! CSR6
vrf definition EIGRP
 rd 24:3
 address-family ipv4
  route-target export 24:3
  route-target import 24:3

 address-family ipv6
  route-target export 24:3
  route-target import 24:3

vrf definition OSPF
 rd 24:2
 address-family ipv4
  route-target export 24:2
  route-target import 24:2

 address-family ipv6
  route-target export 24:2
  route-target import 24:2

! XRv1
vrf OSPF
```

```

address-family ipv4 unicast
  import route-target
    13:1
    13:2
  export route-target
    13:2

address-family ipv6 unicast
  import route-target
    13:1
    13:2
  export route-target
    13:2

vrf EIGRP
  address-family ipv4 unicast
    import route-target
      13:1
      13:3
    export route-target
      13:3

  address-family ipv6 unicast
    import route-target
      13:1
      13:3
    export route-target
      13:3

```

Once these VRFs are defined, we can configure the transit links. Since we are extending 2 different VPNs across ASes, we must create 2 VRF-aware transit links between each set of neighbors. For brevity, I only show XRv1 and CSR6. Notice that CSR6 has 4 new interfaces as it has two inter-AS peers, and for redundancy, configures a link to each peer for each VRF. I use QinQ for VLAN conservation as well, but this is not a requirement. I also overlap IPv4/v6 addresses on the transit links to minimize configuration changes. Since we want to test MVPN later, we also enable PIM on all of these transit links.

```

! XRv1
multicast-routing
  vrf OSPF
    address-family ipv4
      interface all enable
    address-family ipv6
      interface all enable

  vrf EIGRP
    address-family ipv4
      interface all enable
    address-family ipv6

```

```

interface all enable

interface GigabitEthernet0/0/0/0.5612
 vrf OSPF
 ipv4 address 10.6.11.11 255.255.255.0
 ipv6 address fe80::11 link-local
 ipv6 address fd00:10:6:11::11/64
 encapsulation dot1q 3561 second-dot1q 2

interface GigabitEthernet0/0/0/0.5613
 vrf EIGRP
 ipv4 address 10.6.11.11 255.255.255.0
 ipv6 address fe80::11 link-local
 ipv6 address fd00:10:6:11::11/64
 encapsulation dot1q 3561 second-dot1q 3

! CSR6
interface GigabitEthernet2.5562
 encapsulation dot1Q 3556 second-dot1q 2
 vrf forwarding OSPF
 ip address 10.5.6.6 255.255.255.0
 ip pim sparse-mode
 ipv6 address FE80::6 link-local
 ipv6 address FD00:10:5:6::6/64

interface GigabitEthernet2.5563
 encapsulation dot1Q 3556 second-dot1q 3
 vrf forwarding EIGRP
 ip address 10.5.6.6 255.255.255.0
 ip pim sparse-mode
 ipv6 address FE80::6 link-local
 ipv6 address FD00:10:5:6::6/64

interface GigabitEthernet2.5612
 encapsulation dot1Q 3561 second-dot1q 2
 vrf forwarding OSPF
 ip address 10.6.11.6 255.255.255.0
 ip pim sparse-mode
 ipv6 address FE80::6 link-local
 ipv6 address FD00:10:6:11::6/64

interface GigabitEthernet2.5613
 encapsulation dot1Q 3561 second-dot1q 3
 vrf forwarding EIGRP
 ip address 10.6.11.6 255.255.255.0
 ip pim sparse-mode
 ipv6 address FE80::6 link-local
 ipv6 address FD00:10:6:11::6/64

```

Since we haven't configured BGP yet and it would make sense to ensure the links were configured correctly, we can verify all the PIM neighbors. This is a slow process but we need to verify it eventually, and it also implies that the VLAN tagging was done correctly. To speed things up, I verify this on CSR5 and CSR6 only since they have multiple inter-AS links.

```
R5#show ip pim vrf EIGRP neighbor | begin ^Neigh
Neighbor      Interface      Uptime/Expires    Ver    DR
Address
10.5.7.7      GigabitEthernet2.5573    15:07:02/00:01:37 v2      1 / DR S P G
10.5.6.6      GigabitEthernet2.5563    15:07:02/00:01:36 v2      1 / DR S P G

R5#show ipv6 pim vrf EIGRP neighbor | begin ^Neigh
Neighbor Address      Interface      Uptime    Expires    Mode DR pri
FE80::7      Gi2.5573      00:00:08    00:01:36 B G    DR 1
FE80::6      Gi2.5563      00:00:08    00:01:36 B G    DR 1

R6#show ip pim vrf EIGRP neighbor | begin ^Neigh
Neighbor      Interface      Uptime/Expires    Ver    DR
Address
10.5.6.5      GigabitEthernet2.5563    15:07:27/00:01:23 v2      1 / S P G
10.6.11.11    GigabitEthernet2.5613    00:08:58/00:01:18 v2      1 / DR P G

R6#show ipv6 pim vrf EIGRP neighbor | begin ^Neigh
Neighbor Address      Interface      Uptime    Expires    Mode DR pri
FE80::5      Gi2.5563      00:00:32    00:01:43 B G      1
FE80::11     Gi2.5613      00:02:31    00:01:25 B G    DR 1
```

Next, I will configure the BGP sessions between pairs of routers across these transit links. This is very basic so I only show XRv1 and CSR6. Again, CSR6 has two sets of peers per VRF as it is multi-homed to AS 13. Extended communities are explicitly enabled on all peers so that specific EIGRP and OSPF information can be exchanged; this is not going to impact any RT policies between ASes.

```
! XRv1
router bgp 13
  vrf EIGRP
    rd 13:3
    address-family ipv4 unicast
    address-family ipv6 unicast

  neighbor 10.6.11.6
    remote-as 24
    address-family ipv4 unicast
    route-policy RPL_PASS in
    route-policy RPL_PASS out
    send-extended-community-ebgp

  neighbor fd00:10:6:11::6
```

```

    remote-as 24
    address-family ipv6 unicast
        route-policy RPL_PASS in
        route-policy RPL_PASS out
        send-extended-community-ebgp

vrf OSPF
    rd 13:2
    address-family ipv4 unicast
    address-family ipv6 unicast

neighbor 10.6.11.6
    remote-as 24
    address-family ipv4 unicast
        route-policy RPL_PASS in
        route-policy RPL_PASS out
        send-extended-community-ebgp

neighbor fd00:10:6:11::6
    remote-as 24
    address-family ipv6 unicast
        route-policy RPL_PASS in
        route-policy RPL_PASS out
        send-extended-community-ebgp

! CSR6
router bgp 24
    address-family ipv4 vrf EIGRP
        neighbor 10.5.6.5 remote-as 13
        neighbor 10.5.6.5 activate
        neighbor 10.5.6.5 send-community extended
        neighbor 10.6.11.11 remote-as 13
        neighbor 10.6.11.11 activate
        neighbor 10.6.11.11 send-community extended

    address-family ipv6 vrf EIGRP
        neighbor FD00:10:5:6::5 remote-as 13
        neighbor FD00:10:5:6::5 activate
        neighbor FD00:10:5:6::5 send-community extended
        neighbor FD00:10:6:11::11 remote-as 13
        neighbor FD00:10:6:11::11 activate
        neighbor FD00:10:6:11::11 send-community extended

    address-family ipv4 vrf OSPF
        neighbor 10.5.6.5 remote-as 13
        neighbor 10.5.6.5 activate
        neighbor 10.5.6.5 send-community extended
        neighbor 10.6.11.11 remote-as 13
        neighbor 10.6.11.11 activate

```

```
neighbor 10.6.11.11 send-community extended
```

```
address-family ipv6 vrf OSPF
neighbor FD00:10:5:6::5 remote-as 13
neighbor FD00:10:5:6::5 activate
neighbor FD00:10:5:6::5 send-community extended
neighbor FD00:10:6:11::11 remote-as 13
neighbor FD00:10:6:11::11 activate
neighbor FD00:10:6:11::11 send-community extended
```

We can quickly verify these sessions come up (ignore the route counters for now) by checking CSR5 and CSR6 only. We can see that all back-to-back BGP sessions are currently operational.

```
R5#show bgp vpnv4 unicast vrf OSPF summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ  Up/Down  State/PfxRcd
10.5.6.6      4      24   1022   1026    40    0    0 15:18:33      2
10.5.7.7      4      24   1018   1020    40    0    0 15:18:38      2
```

```
R5#show bgp vpnv4 unicast vrf EIGRP summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ  Up/Down  State/PfxRcd
10.5.6.6      4      24   1025   1021    40    0    0 15:18:45      4
10.5.7.7      4      24   1024   1024    40    0    0 15:18:44      4
```

```
R6#show bgp vpnv4 unicast vrf OSPF summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ  Up/Down  State/PfxRcd
10.5.6.5      4      13   1026   1022    40    0    0 15:18:53      6
10.6.11.11    4      13    322    357    40    0    0 05:07:44      6
```

```
R6#show bgp vpnv4 unicast vrf EIGRP summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ  Up/Down  State/PfxRcd
10.5.6.5      4      13   1021   1025    40    0    0 15:19:01      6
10.6.11.11    4      13    320    359    40    0    0 05:07:46      6
```

Last, we configure the VPNv4 peerings from the ASBRs (PEs) to the local RR in each AS. I do not show the RR configurations as we verified those earlier. Each client has a very basic BGP RR-client configuration as well, and I show XRv1 and CSR6 for brevity.

```
! XRv1
router bgp 13
address-family vpnv4 unicast
address-family vpnv6 unicast

neighbor 13.0.0.12
remote-as 13
timers 10 40
password encrypted 143E302C3C5579
update-source Loopback0
address-family vpnv4 unicast
address-family vpnv6 unicast
```

```

! CSR6
router bgp 24
  no bgp default ipv4-unicast
  neighbor 24.0.0.2 remote-as 24
  neighbor 24.0.0.2 password IBGP24
  neighbor 24.0.0.2 update-source Loopback0
  neighbor 24.0.0.2 timers 10 40

address-family vpnv4
  neighbor 24.0.0.2 activate

address-family vpnv6
  neighbor 24.0.0.2 activate

```

We can verify that the sessions are operational by checking the RRs. XRv2 shows 3 neighbors for both VPNv4/v6 inside AS 13. CSR2 shows similar output for its 3 peers inside AS 24.

```

RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast summary | begin ^Neigh
Neighbor      Spk    AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
13.0.0.5       0    13    7034    6706      55    0    0  18:24:35      6
13.0.0.8       0    13    7082    6731      55    0    0  18:24:45      6
13.0.0.11      0    13    6649    6704      55    0    0  18:24:57      6

RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast summary | begin ^Neigh
Neighbor      Spk    AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
13.0.0.5       0    13    7034    6706      49    0    0  18:24:42      5
13.0.0.8       0    13    7082    6731      49    0    0  18:24:51      6
13.0.0.11      0    13    6649    6705      49    0    0  18:25:03      5

R2#show bgp vpnv4 unicast all summary | begin ^Neigh
Neighbor      V      AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  State/PfxRcd
24.0.0.6      4      24    7237    7302     114    0    0  18:49:54     12
24.0.0.7      4      24    7203    7327     114    0    0  18:49:49     12
24.0.0.14     4      24    6275    6744     114    0    0  17:24:33      2

R2#show bgp vpnv6 unicast all summary | begin ^Neigh
Neighbor      V      AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  State/PfxRcd
24.0.0.6      4      24    7237    7303     135    0    0  18:49:56     12
24.0.0.7      4      24    7204    7327     135    0    0  18:49:52     12
24.0.0.14     4      24    6275    6744     135    0    0  17:24:36      2

```

Ideally, this should complete the basic Option A configuration for L3VPN. Before introducing the complexities of backdoor connections, we will trace the LSP from CSR3 to XRv3. First, CSR3 has an internal EIGRP route to XRv3; right away we can see that the extended-communities used by EIGRP over MPLS were maintained between AS boundaries. We will prove this later as well.

```

R3#show ip route 10.13.13.13
Routing entry for 10.13.13.13/32
  Known via "eigrp 3", distance 90, metric 15880, type internal
  Redistributing via eigrp 3

```

```

Last update from 10.3.12.12 on GigabitEthernet2.532, 15:25:36 ago
Routing Descriptor Blocks:
* 10.3.12.12, from 10.3.12.12, 15:25:36 ago, via GigabitEthernet2.532
  Route metric is 15880, traffic share count is 1
  Total delay is 21 microseconds, minimum bandwidth is 1000000 Kbit
  Reliability 255/255, minimum MTU 1500 bytes
  Loading 1/255, Hops 2

```

Upon receiving traffic for this destination, XRv2 has multiple matching BGP routes with associated VPN labels. One path is from CSR5 and one is from XRv1. The routes are equal in every way so the tie is broken based on the BGP RID; CSR5 is the best path. The label is 5008; this was not exchanged across AS boundaries as this is a local label from CSR5. Additionally, the RT:13:3 is the RT specific to AS 13, so the RT was also not carried over. However, all of the important EIGRP information was transmitted, which is valuable for inter-AS backdoor links. This is true for both routes from XRv1 and CSR5.

```

RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf EIGRP 10.13.13.13/32 | begin
Paths
Paths: (2 available, best #1)
  Advertised to update-groups (with more than one peer):
    0.2
  Path #1: Received by speaker 0
  Advertised to update-groups (with more than one peer):
    0.2
  24, (Received from a RR-client)
    13.0.0.5 (metric 3) from 13.0.0.5 (13.0.0.5)
      Received Label 5008
      Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, imported
      Received Path ID 0, Local Path ID 1, version 45
      Extended community: EIGRP route-info:0x8000:0 EIGRP AD:3:282 EIGRP
RHB:255:1:2560 EIGRP LM:0x0:1:1500 EIGRP VRR:0x0:13.13.13.10 RT:13:3
      Source VRF: EIGRP, Source Route Distinguisher: 13:3
  Path #2: Received by speaker 0
  Not advertised to any peer
  24, (Received from a RR-client)
    13.0.0.11 (metric 3) from 13.0.0.11 (13.0.0.11)
      Received Label 91009
      Origin incomplete, localpref 100, valid, internal, import-candidate,
imported
      Received Path ID 0, Local Path ID 0, version 0
      Extended community: EIGRP route-info:0x8000:0 EIGRP AD:3:282 EIGRP
RHB:255:1:2560 EIGRP LM:0x0:1:1500 EIGRP VRR:0x0:13.13.13.10 RT:13:3
      Source VRF: EIGRP, Source Route Distinguisher: 13:3

```

In addition to label 5008, XRv2 adds a transport label. The route to the BGP next-hop is an IGP route via a non-TE interface, so an LDP label is added. The IGP routes via CSR8 to reach 13.0.0.5/32, so we consult

the LIB to find CSR8's local label for 13.0.0.5/32 and add it to the stack. The label stack becomes {8003 5008}.

```
RP/0/0/CPU0:XRv2#show route 13.0.0.5
Routing entry for 13.0.0.5/32
  Known via "ospf 13", distance 110, metric 3, type intra area
  Routing Descriptor Blocks
    13.8.12.8, from 13.0.0.5, via GigabitEthernet0/0/0/0.582
      Route metric is 3
  No advertising protos.
```

```
RP/0/0/CPU0:XRv2#show mpls ldp bindings 13.0.0.5/32 neighbor 13.0.0.8
13.0.0.5/32, rev 12
  Local binding: label: 92004
  Remote bindings: (2 peers)
    Peer          Label
    -----
    13.0.0.8:0    8003
```

Along this LSP, CSR8 is just a P router and performs PHP. This exposes label 5008 to CSR5, who also performs an LFIB lookup. All labels are removed and the IP traffic is forwarded to (what appears to be) the CE router, CSR6.

```
R8#show mpls forwarding-table labels 8003
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id  Switched     interface
8003      Pop Label  13.0.0.5/32  1160513     Gi2.558   13.5.8.5
```

```
R5#show mpls forwarding-table labels 5008 detail
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id  Switched     interface
5008      No Label  10.13.13.13/32 [V]  \
                                     1642     Gi2.5563   10.5.6.6
MAC/Encaps=22/22, MRU=1504, Label Stack{}
005056A9DE0D005056A9DC6381000DE4810000030800
VPN route: EIGRP
No output feature configured
```

Backtracking for a moment, CSR5 had two choices for forwarding. CSR5 selected CSR6 in this instance as it was the oldest route.

```
R5#show bgp vpnv4 unicast vrf EIGRP 10.13.13.13/32
BGP routing table entry for 13:3:10.13.13.13/32, version 14
Paths: (2 available, best #2, table EIGRP)
  Advertised to update-groups:
    2          1
  Refresh Epoch 1
```

24

```
10.5.7.7 (via vrf EIGRP) from 10.5.7.7 (24.0.0.7)
  Origin incomplete, localpref 100, valid, external
  Extended Community: RT:13:3 0x8800:32768:0 0x8801:3:282
    0x8802:65281:2560 0x8803:1:1500 0x8806:0:168627469
  mpls labels in/out 5008/nolabel
  rx pathid: 0, tx pathid: 0
```

Refresh Epoch 1

24

```
10.5.6.6 (via vrf EIGRP) from 10.5.6.6 (24.0.0.6)
  Origin incomplete, localpref 100, valid, external, best
  Extended Community: RT:13:3 0x8800:32768:0 0x8801:3:282
    0x8802:65281:2560 0x8803:1:1500 0x8806:0:168627469
  mpls labels in/out 5008/nolabel
  rx pathid: 0, tx pathid: 0x0
```

CSR6 performs an ordinary MPLS label imposition as a PE could. CSR5 looks like a CE from its perspective as raw IPv4/v6 packets are received. The VPNv4 route uses remote label 94006 from XRv4, the remote PE. The transport label is not imposed since CSR6 is also the penultimate hop, shown below.

```
R6#show bgp vpnv4 unicast vrf EIGRP 10.13.13.13/32
BGP routing table entry for 24:3:10.13.13.13/32, version 21
Paths: (1 available, best #1, table EIGRP)
  Advertised to update-groups:
    2
  Refresh Epoch 2
  Local
    24.0.0.14 (metric 10) (via default) from 24.0.0.2 (24.0.0.2)
      Origin incomplete, metric 10752, localpref 100, valid, internal, best
      Extended Community: RT:24:3 Cost:pre-bestpath:128:10752 0x8800:32768:0
        0x8801:3:282 0x8802:65281:2560 0x8803:1:1500 0x8806:0:168627469
      Originator: 24.0.0.14, Cluster list: 24.0.0.2
      Connector Attribute: count=1
        type 1 len 12 value 24:3:24.0.0.14
      mpls labels in/out nolabel/94006
      rx pathid: 0, tx pathid: 0x0
```

R6#show ip route 24.0.0.14

Routing entry for 24.0.0.14/32

```
Known via "isis", distance 115, metric 10, type level-2
Redistributing via isis 24
Last update from 24.6.14.14 on GigabitEthernet2.564, 19:01:11 ago
Routing Descriptor Blocks:
* 24.6.14.14, from 24.0.0.14, 19:01:11 ago, via GigabitEthernet2.564
  Route metric is 10, traffic share count is 1
```

R6#show mpls ldp bindings 24.0.0.14 32 neighbor 24.0.0.14

lib entry: 24.0.0.14/32, rev 19

```
remote binding: lsr: 24.0.0.14:0, label: imp-null
```

When XRv4 receives packets with label 94006, it removes the label stack and forwards the packets to XRv3 inside VRF EIGRP. In summary, there were two different and totally independent L3VPN processes we verified. Using traceroute on CSR3, we can verify the entire path. Notice that the PE-CE links at the beginning and the end of the traceroute are unlabeled as usual. The inter-AS transit link is also unlabeled since the ASBRs both consider this a PE-CE link. At each labeled hop, we confirm that the labels revealed by traceroute match the labels we verified above.

```
RP/0/0/CPU0:XRv4#show mpls forwarding vrf EIGRP prefix 10.13.13.13/32
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94006	Unlabelled	10.13.13.13/32 [V]	Gi0/0/0/0.534	10.13.14.13	3362

```
R3#traceroute 10.13.13.13 source 10.3.3.3
Type escape sequence to abort.
Tracing the route to 10.13.13.13
VRF info: (vrf in name/id, vrf out name/id)
 1 10.3.12.12 3 msec 3 msec 2 msec
 2 13.8.12.8 [MPLS: Labels 8003/5008 Exp 0] 5 msec 5 msec 12 msec
 3 10.5.6.5 [MPLS: Label 5008 Exp 0] 20 msec 16 msec 21 msec
 4 10.5.6.6 20 msec 12 msec 12 msec
 5 24.6.14.14 [MPLS: Label 94006 Exp 0] 11 msec 21 msec 21 msec
 6 10.13.14.13 20 msec 14 msec 16 msec
```

We will repeat the exercise for VRF OSPF using IPv6 in the opposite direction. CSR9 will be sending packets to CSR6 (backdoor link is down). CSR9 sees the route to CSR4 as inter-area, which like the EIGRP internal route on CSR3, immediately tells us that the OSPF extended-communities were correctly carried across the AS boundary. Had they not been, this route would have been external.

```
R9#show ipv6 route ::10:4:4:4
Routing entry for ::10:4:4:4/128
  Known via "ospf 2", distance 110, metric 2, type inter area
  Route count is 1/1, share count 0
  Routing paths:
    FE80::2, GigabitEthernet2.529
    Last updated 00:01:30 ago
```

As the ingress LSR, CSR2 will be imposing at least 1 label for L3VPN. It learns a pair VPNv6 routes from CSR6 and CSR7. CSR6 wins due to having a lower BGP RID, so label 6027 is pushed first. We also confirm that both routes carry the OSPF extended communities that were passed from AS 13.

```
R2#show bgp vpnv6 unicast vrf OSPF ::10:4:4:4/128
BGP routing table entry for [24:2>::10:4:4:4/128, version 145
Paths: (2 available, best #1, table OSPF)
```

```

Advertised to update-groups:
  1
Refresh Epoch 6
13, (Received from a RR-client)
::FFFF:24.0.0.6 (metric 20) (via default) from 24.0.0.6 (24.0.0.6)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
    OSPF RT:0.0.0.0:2:0
  mpls labels in/out nolabel/6027
  rx pathid: 0, tx pathid: 0x0
Refresh Epoch 6
13, (Received from a RR-client)
::FFFF:24.0.0.7 (metric 20) (via default) from 24.0.0.7 (24.0.0.7)
  Origin incomplete, metric 0, localpref 100, valid, internal
  Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
    OSPF RT:0.0.0.0:2:0
  mpls labels in/out nolabel/7027
  rx pathid: 0, tx pathid: 0

```

Since 6VPE is in use, the BGP next-hop is an IPv4 route so the router will perform a lookup in the IPv4 FIB. The route is IGP with a next-hop of XRv4, so XRv4's LDP label for 24.0.0.6/32 is pushed. The FIB shows this as label 94008. The full label stack is also revealed for this VPNv6 prefix as {94008 6027}.

```

R2#show ip route 24.0.0.6
Routing entry for 24.0.0.6/32
  Known via "isis", distance 115, metric 20, type level-2
  Redistributing via isis 24
  Last update from 24.2.14.14 on GigabitEthernet2.524, 16:12:42 ago
  Routing Descriptor Blocks:
  * 24.2.14.14, from 24.0.0.6, 16:12:42 ago, via GigabitEthernet2.524
    Route metric is 20, traffic share count is 1

```

```

R2#show ip cef 24.0.0.6
24.0.0.6/32
  nexthop 24.2.14.14 GigabitEthernet2.524 label 94008

```

```

R2#show ipv6 cef vrf OSPF ::10:4:4:4/128
::10:4:4:4/128
  nexthop 24.2.14.14 GigabitEthernet2.524 label 94008 6027

```

XRv4 is a P router along this LSP and performs PHP to expose label 6027 to CSR6, the remote PE. CSR6 strips all labels and forwards the traffic to CSR5.

```

RP/0/0/CPU0:XRv4#show mpls forwarding labels 94008

```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94008	Pop	24.0.0.6/32	Gi0/0/0/0.564	24.6.14.6	1083065

```

R6#show mpls forwarding-table labels 6027 detail
Local      Outgoing   Prefix           Bytes Label  Outgoing   Next Hop
Label      Label       or Tunnel Id     Switched     interface
6027       No Label    ::10:4:4:4/128[V] \
                                         0          Gi2.5562    FE80::5
MAC/Encaps=22/22, MRU=1504, Label Stack{}
005056A9DC63005056A9DE0D81000DE48100000286DD
VPN route: OSPF
No output feature configured

```

Just like CSR5 had earlier, CSR6 has two decisions for forwarding. It selects CSR5 due to being the oldest route, but both Xrv1 and CSR5 provide the same information.

```

R6#show bgp vpnv6 unicast vrf OSPF ::10:4:4:4/128
BGP routing table entry for [24:2>::10:4:4:4/128, version 187
Paths: (2 available, best #2, table OSPF)
  Advertised to update-groups:
    3          1
  Refresh Epoch 1
  13
  FD00:10:6:11::11 (FE80::11) (via vrf OSPF) from FD00:10:6:11::11
(13.0.0.11)
  Origin incomplete, localpref 100, valid, external
  Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
  OSPF RT:0.0.0.0:2:0
  mpls labels in/out 6027/nolabel
  rx pathid: 0, tx pathid: 0
  Refresh Epoch 2
  13
  FD00:10:5:6::5 (FE80::5) (via vrf OSPF) from FD00:10:5:6::5 (13.0.0.5)
  Origin incomplete, localpref 100, valid, external, best
  Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
  OSPF RT:0.0.0.0:2:0
  mpls labels in/out 6027/nolabel
  rx pathid: 0, tx pathid: 0x0

```

As the ingress PE, CSR5 will push a VPN label for this prefix as allocated by CSR8. Since CSR5 and CSR8 are directly connected, no LDP label is pushed.

```

R5#show bgp vpnv6 unicast vrf OSPF ::10:4:4:4/128
BGP routing table entry for [13:2>::10:4:4:4/128, version 11
Paths: (1 available, best #1, table OSPF)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  Local
  ::FFFF:13.0.0.8 (metric 2) (via default) from 13.0.0.12 (13.0.0.12)

```

```

Origin incomplete, metric 1, localpref 100, valid, internal, best
Extended Community: RT:13:2 OSPF ROUTER ID:10.4.8.8:0
  OSPF RT:0.0.0.0:2:0
Originator: 13.0.0.8, Cluster list: 13.0.0.12
mpls labels in/out nolabel/8017
rx pathid: 0, tx pathid: 0x0

```

```

R5#show ip route 13.0.0.8
Routing entry for 13.0.0.8/32
  Known via "ospf 13", distance 110, metric 2, type intra area
  Last update from 13.5.8.8 on GigabitEthernet2.558, 19:19:08 ago
  Routing Descriptor Blocks:
  * 13.5.8.8, from 13.0.0.8, 19:19:08 ago, via GigabitEthernet2.558
    Route metric is 2, traffic share count is 1

```

```

R5#show mpls ldp bindings 13.0.0.8 32 neighbor 13.0.0.8
  lib entry: 13.0.0.8/32, rev 12
    remote binding: lsr: 13.0.0.8:0, label: imp-null

```

When CSR8 receives packets labeled 8017, it removes all labels and forwards the traffic to CSR4 inside VRF OSPF. This completes the transit path.

```

R8#show mpls forwarding-table labels 8017 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
8017       No Label  ::10:4:4:4/128[V]  \
                                         0         Gi2.548    FE80::4
MAC/Encaps=0/0, MRU=1504, Label Stack{}
VPN route: OSPF
No output feature configured

```

Using extended traceroute on CSR9, we can confirm the label stacks along the way. Notice that the inter-AS link is raw IPv6 as MPLS is not enabled between providers.

```

R9#traceroute ipv6
Target IPv6 address: ::10:4:4:4
Source address: ::10:9:9:9
[snip]
Type escape sequence to abort.
Tracing the route to ::10:4:4:4

 1 FD00:10:2:9::2 15 msec 5 msec 4 msec
 2 2024:24:2:14::14 [MPLS: Labels 94008/6027 Exp 0] 5 msec 4 msec 16 msec
 3 FD00:10:5:6::6 [MPLS: Label 6027 Exp 0] 17 msec 19 msec 24 msec
 4 FD00:10:5:6::5 20 msec 16 msec 15 msec
 5 FD00:10:4:8::8 [MPLS: Label 8017 Exp 0] 15 msec 23 msec 21 msec
 6 FD00:10:4:8::4 32 msec 11 msec 15 msec

```

Next, we will verify central services connectivity. I will trace the LSPs much more quickly since the process is identical; again, no new technologies are introduced. As a sanity check, we ensure that intra-AS central services is working. In order to get these BGP routes into EIGRP, we must define a redistribution metric. We don't need to do this when the BGP route carries the EIGRP extended communities, but these routes are actually external. Rather than reset EIGRP metrics for all prefixes, I use a parameterized RPL for the XR PEs. Only the Internet routes have their metrics set, while other routes are allowed to pass transparently. The RPL can consume IPv4 or IPv6 prefix-sets as well.

```
! XRv1 and XRv2
prefix-set PS_INTERNET_ROUTES
  110.0.0.0/8 le 32
end-set

prefix-set PS_INTERNET_ROUTES_V6
  ::110:0:0:0/80 le 128
end-set

route-policy RPL_BGP_TO_EIGRP($PS)
  if destination in $PS then
    set eigrp-metric 100000 10 255 1 1500
  else
    pass
  endif
end-policy

router eigrp EIGRP
  vrf EIGRP
    address-family ipv4
      redistribute bgp 13 route-policy RPL_BGP_TO_EIGRP(PS_INTERNET_ROUTES)
    address-family ipv6
      redistribute bgp 13 route-policy RPL_BGP_TO_EIGRP(PS_INTERNET_ROUTES_V6)
```

On XRv2, we can verify this worked by checking the EIGRP topology inside the VRF. The VPNv4 sourced routes are clearly different than the truly external ones, as shown below. The metric values also differ which validates the RPL configuration.

```
RP/0/0/CPU0:XRv2#show eigrp vrf EIGRP topology 10.13.13.13/32
IPv4-EIGRP VR(EIGRP) AS(3) VRF EIGRP: Topology entry for 10.13.13.13/32
  State is Passive, Query origin flag is 1, 1 Successor(s), FD is 1377280,
  RIB is 10760
  Routing Descriptor Blocks:
    13.0.0.5, from VPNv4 Sourced, Send flag is 0x0
      Composite metric is (1377280/0), Route is Internal (VPNv4 Sourced)
      Vector metric:
        Minimum bandwidth is 1000000 Kbit
        Total delay is 11015625 picoseconds
        Reliability is 255/255
```

```

Load is 1/255
Minimum MTU is 1500
Hop count is 1
Originating router is 10.13.13.13

RP/0/0/CPU0:XRv2#show eigrp vrf EIGRP topology 110.0.0.2/32
IPv4-EIGRP VR(EIGRP) AS(3) VRF EIGRP: Topology entry for 110.0.0.2/32
State is Passive, Query origin flag is 1, 1 Successor(s), FD is 13107200,
RIB is 102400
Routing Descriptor Blocks:
13.0.0.8, from Redistributed, Send flag is 0x0
Composite metric is (13107200/0), Route is External
Vector metric:
  Minimum bandwidth is 100000 Kbit
  Total delay is 100000000 picoseconds
Reliability is 255/255
Load is 1/255
Minimum MTU is 1500
Hop count is 0
External data:
  Originating router is 13.0.0.12 (this system)
AS number of route is 3
External protocol is BGP, external metric is 0
Administrator tag is 100 (0x00000064)

```

On CSR2, I use a pair of prefix-lists and route-maps to accomplish the same thing.

```

! CSR2
ip prefix-list PL_INTERNET_ROUTES seq 5 permit 110.0.0.0/8 le 32
ipv6 prefix-list PL_INTERNET_ROUTES_V6 seq 5 permit ::110:0:0:0/80 le 128

route-map RM_BGP_TO_EIGRP permit 10
  match ip address prefix-list PL_INTERNET_ROUTES
  set metric 100000 10 255 1 1500
route-map RM_BGP_TO_EIGRP permit 100

route-map RM_BGP_TO_EIGRP_V6 permit 10
  match ipv6 address prefix-list PL_INTERNET_ROUTES_V6
  set metric 100000 10 255 1 1500
route-map RM_BGP_TO_EIGRP_V6 permit 100

router eigrp EIGRP
  address-family ipv4 unicast vrf EIGRP autonomous-system 3
  topology base
  redistribute bgp 24 route-map RM_BGP_TO_EIGRP

  address-family ipv6 unicast vrf EIGRP autonomous-system 3
  topology base
  redistribute bgp 24 route-map RM_BGP_TO_EIGRP_V6

```


We use the same verification method on CSR2 except I look at IPv6 Internet routes.

```
R2#show eigrp address-family ipv6 vrf EIGRP topology ::10:3:3:3/128
EIGRP-IPv6 VR(EIGRP) Topology Entry for AS(3)/ID(10.1.2.2)
    Topology(base) TID(0) VRF(EIGRP)
EIGRP-IPv6(3): Topology base(0) entry for ::10:3:3:3/128
    State is Passive, Query origin flag is 1, 1 Successor(s), FD is 1392640
    Descriptor Blocks:
    ::FFFF:24.0.0.6, from VPNv6 Sourced, Send flag is 0x0
        Composite metric is (1392640/0), route is Internal (VPNv6 Sourced)
    Vector metric:
        Minimum bandwidth is 1000000 Kbit
        Total delay is 11250000 picoseconds
        Reliability is 255/255
        Load is 1/255
        Minimum MTU is 1500
        Hop count is 1
        Originating router is 10.3.3.3

R2#show eigrp address-family ipv6 vrf EIGRP topology ::110:0:0:1/128
EIGRP-IPv6 VR(EIGRP) Topology Entry for AS(3)/ID(10.1.2.2)
    Topology(base) TID(0) VRF(EIGRP)
EIGRP-IPv6(3): Topology base(0) entry for ::110:0:0:1/128
    State is Passive, Query origin flag is 1, 1 Successor(s), FD is 13107200
    Descriptor Blocks:
    ::FFFF:24.0.0.6, from Redistributed, Send flag is 0x0
        Composite metric is (13107200/0), route is External
    Vector metric:
        Minimum bandwidth is 100000 Kbit
        Total delay is 100000000 picoseconds
        Reliability is 255/255
        Load is 1/255
        Minimum MTU is 1500
        Hop count is 0
    External data:
        Originating router is 10.1.2.2 (this system)
        AS number of route is 24
        External protocol is BGP, external metric is 0
        Administrator tag is 0 (0x00000000)
```

Moving back to AS 13, we will verify the intra-AS central services connectivity now. XRv2 has a VPN route for 100.0.0.0/32 which pushes a single VPN label allocated by CSR8. In the opposite direction, CSR8 pushes a label from XRv2 to reach CSR3's loopback.

```
RP/0/0/CPU0:XRv2#show cef vrf EIGRP 110.0.0.0/32
110.0.0.0/32, version 7, internal 0x5000001 0x0 (ptr 0xa142dc74) [1], 0x0
(0x0), 0x208 (0xa15a1140)
```

```
Prefix Len 32, traffic index 0, precedence n/a, priority 3
  via 13.0.0.8, 5 dependencies, recursive [flags 0x6000]
    path-idx 0 NHID 0x0 [0xa16099f4 0x0]
    recursion-via-/32
    next hop VRF - 'default', table - 0xe0000000
    next hop 13.0.0.8 via 92005/0/21
    next hop 13.8.12.8/32 Gi0/0/0/0.582 labels imposed {ImplNull 8016}
```

```
R8#show ip cef vrf BGP 10.3.3.3
10.3.3.3/32
  nexthop 13.8.12.12 GigabitEthernet2.582 label 92002
```

A quick traceroute in both directions confirms it is working.

```
R3#traceroute 110.0.0.0 source 10.3.3.3
Type escape sequence to abort.
Tracing the route to 110.0.0.0
VRF info: (vrf in name/id, vrf out name/id)
 1 10.3.12.12 1 msec 2 msec 2 msec
 2 10.8.10.8 [MPLS: Label 8016 Exp 0] 4 msec 3 msec 4 msec
 3 10.8.10.10 4 msec 4 msec 4 msec
```

```
R10#traceroute 10.3.3.3 source 110.0.0.0
Type escape sequence to abort.
Tracing the route to 10.3.3.3
VRF info: (vrf in name/id, vrf out name/id)
 1 10.8.10.8 3 msec 3 msec 2 msec
 2 13.8.12.12 [MPLS: Label 92002 Exp 0] 4 msec 3 msec 3 msec
 3 10.3.12.3 [AS 13] 8 msec 10 msec 16 msec
```

AS 24 has no knowledge of VRF BGP or RT:13:1 at all. Using CSR5 and XRv1, we saw earlier how RT:13:1 was imported into VRFs OSPF and EIGRP at these ASBRs. This allows CSR6 and CSR7 to learn the prefixes as normal “customer” routes without caring that it’s actually a central services extension. As an example, CSR5 imports the regular OSPF RT:13:2 from BGP into VRF OSPF, as well as RT:13:1 which represents the shared serviced VPN.

```
R5#show vrf detail OSPF | include ^Address|port_VPN|RT
Address family ipv4 unicast (Table ID = 0x2):
  Export VPN route-target communities
    RT:13:2
  Import VPN route-target communities
    RT:13:2 RT:13:1
Address family ipv6 unicast (Table ID = 0x1E000002):
  Export VPN route-target communities
    RT:13:2
  Import VPN route-target communities
    RT:13:2 RT:13:1
```

A quick look at the VPNv4 tables for VRFs OSPF and EIGRP on CSR5 shows that the routes have been correctly imported. We can verify their RTs as well.

```
R5#show bgp vpnv4 unicast vrf OSPF 110.0.0.0/32
BGP routing table entry for 13:2:110.0.0.0/32, version 31
Paths: (1 available, best #1, table OSPF)
  Advertised to update-groups:
    3
  Refresh Epoch 1
  100, imported path from 13:1:110.0.0.0/32 (global)
    13.0.0.8 (metric 2) (via default) from 13.0.0.12 (13.0.0.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:13:1
      Originator: 13.0.0.8, Cluster list: 13.0.0.12
      mpls labels in/out nolabel/8016
      rx pathid: 0, tx pathid: 0x0
```

```
R5#show bgp vpnv4 unicast vrf EIGRP 110.0.0.0/32
BGP routing table entry for 13:3:110.0.0.0/32, version 27
Paths: (1 available, best #1, table EIGRP)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  100, imported path from 13:1:110.0.0.0/32 (global)
    13.0.0.8 (metric 2) (via default) from 13.0.0.12 (13.0.0.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:13:1
      Originator: 13.0.0.8, Cluster list: 13.0.0.12
      mpls labels in/out nolabel/8016
      rx pathid: 0, tx pathid: 0x0
```

When these are received by AS 24 ASBRs and exported from the VRF to BGP, the RT is overwritten. AS 24 is none the wiser, and it doesn't matter. CSR7, for example, learns an eBGP path from CSR5 and an iBGP path from CSR6, but both have identical attributes.

```
R7#show bgp vpnv4 unicast vrf OSPF 110.0.0.0/32
BGP routing table entry for 24:2:110.0.0.0/32, version 35
Paths: (2 available, best #2, table OSPF)
  Advertised to update-groups:
    1
  Refresh Epoch 2
  13 100
    24.0.0.6 (metric 10) (via default) from 24.0.0.2 (24.0.0.2)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Extended Community: RT:24:2
      Originator: 24.0.0.6, Cluster list: 24.0.0.2
      mpls labels in/out 7019/6019
```

```

    rx pathid: 0, tx pathid: 0
Refresh Epoch 1
13 100
10.5.7.5 (via vrf OSPF) from 10.5.7.5 (13.0.0.5)
  Origin incomplete, localpref 100, valid, external, best
  Extended Community: RT:24:2
  mpls labels in/out 7019/nolabel
  rx pathid: 0, tx pathid: 0x0

```

Earlier, we cleaned up the BGP to EIGRP redistribution to ensure that both internal routes (inside the EIGRP VPN) and external routes (Internet, extranets, etc) could be redistributed properly. CSR1 and XRv3 both have EIGRP external routes to the Internet which shows that the central services control plane is operational.

```

R1#show ipv6 route ::110:0:0:3
Routing entry for ::110:0:0:3/128
  Known via "eigrp 3", distance 170, metric 107520, type external
  Route count is 1/1, share count 0
  Routing paths:
    FE80::2, GigabitEthernet2.512
    Last updated 06:07:49 ago

```

```

RP/0/0/CPU0:XRv3#show route ipv6 ::110:0:0:3
Routing entry for ::110:0:0:3/128
  Known via "eigrp 3", distance 170, metric 107520
  Tag 13, type external
  Routing Descriptor Blocks
    fe80::14, from fe80::14, via GigabitEthernet0/0/0/0.534
    Route metric is 107520
  No advertising protos.

```

We have already traced several LSPs, so I will use traceroute to verify connectivity instead. As always with option A, the inter-AS transit link traffic is unlabeled. This proves that central services is supported with option A.

```

R1#traceroute ipv6
Target IPv6 address: ::110:0:0:3
Source address: ::10:1:1:1
[snip]
Type escape sequence to abort.
Tracing the route to ::110:0:0:3

 1 FD00:10:1:2::2 4 msec 4 msec 3 msec
 2 2024:24:2:14::14 [MPLS: Labels 94008/6018 Exp 0] 5 msec 5 msec 10 msec
 3 FD00:10:5:6::6 [MPLS: Label 6018 Exp 0] 29 msec 22 msec 23 msec
 4 FD00:10:5:6::5 71 msec 5 msec 5 msec
 5 FD00:10:8:10::8 [MPLS: Label 8008 Exp 0] 8 msec 6 msec 17 msec
 6 FD00:10:8:10::10 23 msec 15 msec 15 msec

```

```
RP/0/0/CPU0:XRv3#traceroute ::110:0:0:3 source ::10:13:13:13
Type escape sequence to abort.
Tracing the route to ::110:0:0:3
```

```
 1  fd00:10:13:14::14 0 msec 0 msec 0 msec
 2  fd00:10:5:6::6 [MPLS: Label 6018 Exp 0] 0 msec 0 msec 0 msec
 3  fd00:10:5:6::5 9 msec 0 msec 0 msec
 4  fd00:10:8:10::8 [MPLS: Label 8008 Exp 0] 0 msec 0 msec 0 msec
 5  fd00:10:8:10::10 9 msec 0 msec 0 msec
```

Next, I will enable the backdoor links. The EIGRP backdoor isn't particularly interesting since it is intra-AS, so I will cover it quickly. I add new interfaces for this link and increase the delay so EIGRP does not prefer it.

```
! CSR1
interface GigabitEthernet2.513
 encapsulation dot1q 3513
 ip address 10.1.13.1 255.255.255.0
 ip pim sparse-mode
 delay 10000
 ipv6 address FE80::1 link-local
 ipv6 address FD00:10:1:13::1/64

router eigrp CUST
 address-family ipv4 unicast autonomous-system 3
  network 10.1.13.1 0.0.0.0

! XRv3
interface GigabitEthernet0/0/0/0.513
 ipv4 address 10.1.13.13 255.255.255.0
 ipv6 address fe80::13 link-local
 ipv6 address fd00:10:1:13::31/64
 encapsulation dot1q 3513

router eigrp CUST
 address-family ipv4
  interface GigabitEthernet0/0/0/0.513
   metric delay 10000
 address-family ipv6
  interface GigabitEthernet0/0/0/0.513
   metric delay 10000
```

I verify that EIGRP neighbors are up on CSR1 for both AFIs, then use traceroute to ensure MPLS is still preferred. Because extended-communities are in use, no loops can form when all links are operational, and the existing inter-AS/central services design is unaffected.

```
R1#show eigrp address-family ipv4 neighbors
```

```
EIGRP-IPv4 VR(CUST) Address-Family Neighbors for AS(3)
H   Address                               Interface      Hold Uptime    SRTT    RTO    Q    Seq
                               (sec)          (ms)          Cnt  Num
0   10.1.13.13                           Gi2.513        12 00:00:07    29    174    0    30
1   10.1.2.2                             Gi2.512        14 19:14:33     1    100    0    14
```

```
R1#show eigrp address-family ipv6 neighbors
```

```
EIGRP-IPv6 VR(CUST) Address-Family Neighbors for AS(3)
H   Address                               Interface      Hold Uptime    SRTT    RTO    Q    Seq
                               (sec)          (ms)          Cnt  Num
0   Link-local address:                   Gi2.513        11 00:00:08    17    102    0    34
    FE80::13
1   Link-local address:                   Gi2.512        11 18:16:28     4    100    0    12
    FE80::2
```

```
R1#traceroute 10.13.13.13 source 10.1.1.1
```

```
Type escape sequence to abort.
```

```
Tracing the route to 10.13.13.13
```

```
VRF info: (vrf in name/id, vrf out name/id)
```

```
 1 10.1.2.2 6 msec 4 msec 4 msec
 2 24.2.14.14 [MPLS: Label 94006 Exp 0] 6 msec 3 msec 4 msec
 3 10.13.14.13 4 msec 11 msec 13 msec
```

```
R1#traceroute ipv6
```

```
Target IPv6 address: ::10:13:13:13
```

```
Source address: ::10:1:1:1
```

```
[snip]
```

```
Tracing the route to ::10:13:13:13
```

```
 1 FD00:10:1:2::2 5 msec 4 msec 4 msec
 2 2024:24:2:14::14 [MPLS: Label 94001 Exp 0] 5 msec 5 msec 5 msec
 3 ::10:13:13:13 17 msec 15 msec 15 msec
```

Shutting down CSR1's link to XRv3 causes the backdoor to be used once EIGRP and BGP converge. Of course, we verify this in both directions but I only show one way for brevity.

```
R1#traceroute 10.13.13.13 source 10.1.1.1
```

```
Type escape sequence to abort.
```

```
Tracing the route to 10.13.13.13
```

```
VRF info: (vrf in name/id, vrf out name/id)
```

```
 1 10.1.13.13 3 msec 1 msec 2 msec
```

```
R1#traceroute 10.13.13.13 source 10.1.1.1
```

```
Type escape sequence to abort.
```

```
Tracing the route to 10.13.13.13
```

```
VRF info: (vrf in name/id, vrf out name/id)
```

```
 1 10.1.13.13 3 msec 1 msec 2 msec
```

```
R1#traceroute ipv6
```

```
Target IPv6 address: ::10:13:13:13
Source address: ::10:1:1:1
[snip]
Tracing the route to ::10:13:13:13
```

```
1 ::10:13:13:13 2 msec 3 msec 2 msec
```

Of greater interest is the OSPF backdoor. The configuration is shown below and is very basic. After applying this configuration, the backdoor link is preferred over the MPLS link despite the high cost. This is the classic use-case for the sham-link so that the MPLS-reachable routes are also OSPF intra-area. I only show CSR4 for brevity since the interfaces are identical, IPv4/v6 addresses notwithstanding.

```
! CSR4
interface GigabitEthernet2.549
 encapsulation dot1Q 3549
 ip address 10.4.9.4 255.255.255.0
 ip pim sparse-mode
 ipv6 address FE80::4 link-local
 ipv6 address FD00:10:4:9::4/64
 ospfv3 network point-to-point
 ospfv3 2 cost 500
 ospfv3 2 ipv4 area 0
 ospfv3 2 ipv6 area 0
```

```
R4#show ip route 10.9.9.9
Routing entry for 10.9.9.9/32
  Known via "ospfv3 2", distance 110, metric 500, type intra area
  Last update from 10.4.9.9 on GigabitEthernet2.549, 00:00:49 ago
  Routing Descriptor Blocks:
    * 10.4.9.9, from 10.4.9.9, 00:00:49 ago, via GigabitEthernet2.549
      Route metric is 500, traffic share count is 1
```

```
R4#traceroute 10.9.9.9 source 10.4.4.4
Type escape sequence to abort.
Tracing the route to 10.9.9.9
VRF info: (vrf in name/id, vrf out name/id)
 1 10.4.9.9 4 msec 3 msec 3 msec
```

Since we already have ordinary MPLS service between AS boundaries, creating a sham-link requires no new steps. I apply route-maps to the redistribution points to ensure that the sham-link endpoints don't get exposed to the customer. This is not necessary for IPv4 since the sham-link endpoints are IPv6 addresses. This topic is covered in detail in the multi-VRF CE chapter. I only show the configuration for CSR2 since CSR8 is identical, except for minor details like BGP AS, loopback IPv6 address, and sham-link source/destination.

```
! CSR2
interface Loopback2
```

```

vrf forwarding OSPF
ipv6 address FD00::2/128

ipv6 prefix-list PL_SHAM seq 5 permit FD00::/124 ge 128

route-map RM_BGP_TO_OSPF deny 10
  match ipv6 address prefix-list PL_SHAM
route-map RM_BGP_TO_OSPF permit 100

router ospfv3 2
  address-family ipv4 unicast vrf OSPF
  redistribute bgp 24
  area 0 sham-link FD00::2 FD00::8

address-family ipv6 unicast vrf OSPF
  redistribute bgp 24 route-map RM_BGP_TO_OSPF
  area 0 sham-link FD00::2 FD00::8

router bgp 13
  address-family ipv6 vrf OSPF
  network FD00::2/128

```

Since OSPFv3 IPv4 and IPv6 are two separate protocol instances, we created two separate sham-links. On CSR2, I verify that they are both operational. Unlike virtual-links, sham-links are just ordinary P2P adjacencies in the OSPF graph. I verify this by looking at CSR2's local router LSA; it reveals a direct connection to CSR8 and CSR9. The default sham-link cost is 1 but this is adjustable.

```

R2#show ospfv3 vrf OSPF sham-links | include ^Sham
Sham Link OSPFv3_SL0 to address FD00::8 is up
Sham Link OSPFv3_SL1 to address FD00::8 is up

R2#show ospfv3 vrf OSPF ipv4 database router adv-router 10.2.9.2
[snip]
  Number of Links: 2

    Link connected to: another Router (point-to-point)
      Link Metric: 1
      Local Interface ID: 18
      Neighbor Interface ID: 32
      Neighbor Router ID: 10.4.8.8

    Link connected to: another Router (point-to-point)
      Link Metric: 1
      Local Interface ID: 14
      Neighbor Interface ID: 27
      Neighbor Router ID: 10.4.9.9

```


We can verify that BGP is transporting these prefixes correctly (it must be, or else the sham-link would be down). CSR2 sees both its locally originated endpoint and the BGP-learned endpoint representing CSR8.

```
R2#show bgp vpnv6 unicast vrf OSPF FD00::/124 longer-prefixes | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 24:2 (default for vrf OSPF)
*> FD00::2/128          ::          0          32768 i
*>i FD00::8/128          ::FFFF:24.0.0.6 0      100      0 13 i
* i                     ::FFFF:24.0.0.7 0      100      0 13 i
```

For brevity, I use traceroute from CSR4 to demonstrate the inter-AS sham-link. The inter-AS transit links are unlabeled as usual, but traffic still follows the proper path.

```
R4#traceroute 10.9.9.9 source 10.4.4.4
Type escape sequence to abort.
Tracing the route to 10.9.9.9
VRF info: (vrf in name/id, vrf out name/id)
 1 10.4.8.8 5 msec 4 msec 4 msec
 2 10.5.6.5 [MPLS: Label 5004 Exp 0] 3 msec 3 msec 3 msec
 3 10.5.6.6 6 msec 10 msec 10 msec
 4 24.6.14.14 [MPLS: Labels 94009/2009 Exp 0] 11 msec 20 msec 18 msec
 5 10.2.9.2 [MPLS: Label 2009 Exp 0] 25 msec 20 msec 17 msec
 6 10.2.9.9 21 msec 8 msec 12 msec
```

```
R4#traceroute ipv6
Target IPv6 address: ::10:9:9:9
Source address: ::10:4:4:4
[snip]
Tracing the route to ::10:9:9:9

 1 FD00:10:4:8::8 5 msec 4 msec 4 msec
 2 FD00:10:5:6::5 [MPLS: Label 5015 Exp 0] 3 msec 3 msec 3 msec
 3 FD00:10:5:6::6 23 msec 14 msec 15 msec
 4 2024:24:6:14::14 [MPLS: Labels 94009/2010 Exp 0] 15 msec 23 msec 22 msec
 5 FD00:10:2:9::2 [MPLS: Label 2010 Exp 0] 23 msec 22 msec 21 msec
 6 FD00:10:2:9::9 23 msec 14 msec 15 msec
```

As a final test, I shut down CSR4's link to CSR8 and wait for OSPF/BGP to converge. The backdoor link can successfully be used for failover if the L3VPN is not available.

```
R4#traceroute 10.9.9.9 source 10.4.4.4
Type escape sequence to abort.
Tracing the route to 10.9.9.9
VRF info: (vrf in name/id, vrf out name/id)
 1 10.4.9.9 4 msec 3 msec 3 msec
```

```

R4#traceroute ipv6
Target IPv6 address: ::10:9:9:9
Source address: ::10:4:4:4
[snip]
Tracing the route to ::10:9:9:9

 1 FD00:10:4:9::9 3 msec 3 msec 3 msec

```

8.4.1.2 L2VPN

L2VPN over option A is very similar to L3VPN in terms of the general logic. Traffic enters the PE and the layer 2 frame is wrapped inside MPLS. The end of the virtual circuit is the ASBR, which like L3VPN, simply removes the labels and forwards the frame out of the access circuit (AC). The ASBR on the other end of the AC treats this incoming frame like it came from a customer and encapsulates it inside MPLS as expected. Like option A, the two providers can use totally different L2VPN mechanisms. In this example, AS 13 uses LDP signaled VPLS via BGP auto-discovery while AS 24 uses BGP signaled VPLS via BGP auto-discovery. Any kind of EVPN variations or static VPLS configurations are also acceptable. The basic PE-CE access configurations were validated earlier, so we will add the specific L2VPN details. First, we configure LDP-based VPLS on CSR8. This includes activating the proper AFI towards XRv2, the RR for AS 13. The bridge-domain ties the VFI to the service-instance. L2VPN has a dedicated chapter which covers the operational details.

```

! CSR8
l2vpn vfi context VPLS
  vpn id 3
  autodiscovery bgp signaling ldp template TMP_VPLS
  rd 13:30

bridge-domain 3
  member GigabitEthernet2 service-instance 3
  member vfi VPLS

router bgp 13
  address-family l2vpn vpls
    neighbor 13.0.0.12 activate
    neighbor 13.0.0.12 prefix-length-size 2

```

To avoid creating a bridging loop, the L2VPN traffic will only use one of the inter-AS links. Specifically, I will configure the L2VPN ASBRs to be CSR5 and CSR6. CSR5 terminates the PW inside AS 13 using an almost identical configuration to CSR8. I use dot1q tag 30 to demonstrate that the tags can be different at every layer 2 hop since the EFPs are stripping all tags before MPLS encapsulation occurs. This access circuit was not verified earlier since CSR5 wasn't initially considered an L2VPN PE. Using the option A model, it would be.

```

! CSR5
interface GigabitEthernet2

```

```

service instance 30 ethernet
  encapsulation dot1q 3556 second-dot1q 30
  rewrite ingress tag pop 2 symmetric

l2vpn vfi context VPLS
  vpn id 3
  autodiscovery bgp signaling ldp template TMP_VPLS
  rd 13:30

bridge-domain 3
  member GigabitEthernet2 service-instance 30
  member vfi VPLS

router bgp 13
  address-family l2vpn vpls
    neighbor 13.0.0.12 activate
    neighbor 13.0.0.12 prefix-length-size 2

```

Last, we will configure XRv2 as the RR for the L2VPN-VPLS AFI. It doesn't do any local processing; it only needs to reflect the L2VPN AD routes and disable BGP signaling as LDP should be used.

```

! XRv2
router bgp 13
  address-family l2vpn vpls-vpws

  af-group L2VPN address-family l2vpn vpls-vpws
    route-reflector-client
    Signalling bgp disable

  neighbor 13.0.0.5
    address-family l2vpn vpls-vpws
    use af-group L2VPN

  neighbor 13.0.0.8
    address-family l2vpn vpls-vpws
    use af-group L2VPN

```

Once this is complete, we can verify that the L2VPN BGP routes were properly exchanged. Notice that there are no labels associated with these routes. BGP is only used for discovery while tLDP is used for signaling, so no labels need to be exchanged. XRv2 learns both routes via iBGP and reflects them appropriately.

```

RP/0/0/CPU0:XRv2#show bgp l2vpn vpls rd 13:30 | begin Network
  Network          Next Hop          Rcvd Label          Local Label
Route Distinguisher: 13:30
*>i13.0.0.5/32      13.0.0.5          nolabel             nolabel
*>i13.0.0.8/32      13.0.0.8          nolabel             nolabel

```

Upon receiving the peer route, both CSR5 and CSR8 are able to create the tLDP session. We verify that the tLDP neighbor comes up and that the MPLS PW follows. Don't be fooled by the RT:13:3 value; although the same set of numbers used for EIGRP L3VPN, this is auto-generated. The 13 is the BGP ASN and the 3 is the VPN ID. I did this intentionally to make the lab tricky. The MPLS labels are zeroed out as well (explicit null is 0x00000).

```
R5#show bgp l2vpn vpls rd 13:30 13.0.0.8
BGP routing table entry for 13:30:13.0.0.8/96, version 3
Paths: (1 available, best #1, table L2VPN-VPLS-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    13.0.0.8 (metric 2) from 13.0.0.12 (13.0.0.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best, AGI
version(4194304002)
      Extended Community: RT:13:3 L2VPN AGI:13:3
      Originator: 13.0.0.8, Cluster list: 13.0.0.12
      mpls labels in/out exp-null/exp-null
      rx pathid: 0, tx pathid: 0x0
```

```
R5#show l2vpn atom vc
```

Interface	Peer ID	VC ID	Service		Status
			Type	Name	
pw100002	13.0.0.8	3	vfi	VPLS	UP

Next, we will configure VPLS auto-discovery and signaling using BGP inside AS 24. The configuration is very similar. Since CSR2 is the RR, we only have to enable the session directly between CSR2 and CSR6.

```
! CSR2
l2vpn vfi context VPLS
  vpn id 3
  autodiscovery bgp signaling bgp template TMP_VPLS
  ve id 2
  rd 24:30

bridge-domain 3
  member GigabitEthernet2 service-instance 3
  member vfi VPLS

router bgp 24
  address-family l2vpn vpls
    neighbor 24.0.0.6 activate
    neighbor 24.0.0.6 suppress-signaling-protocol ldp
```

Like CSR5, CSR6 needs to define an EFP for the inter-AS Ethernet frames. Otherwise, the configuration is almost identical to CSR2.

```
! CSR6
interface GigabitEthernet2
  service instance 30 ethernet
  encapsulation dot1q 3556 second-dot1q 30
  rewrite ingress tag pop 2 symmetric

l2vpn vfi context VPLS
  vpn id 3
  autodiscovery bgp signaling bgp template TMP_VPLS
  ve id 6
  rd 24:30

bridge-domain 3
  member GigabitEthernet2 service-instance 30
  member vfi VPLS

router bgp 24
  address-family l2vpn vpls
  neighbor 24.0.0.2 activate
  neighbor 24.0.0.2 suppress-signaling-protocol ldp
```

BGP-based VPLS signaling is much more complex than LDP in terms of mathematical evaluation. The labels are not carried in the BGP route, but a series of parameters are used to compute the label. This process is covered in detail in the L2VPN section. In short, CSR6 receives the route from CSR2, builds the PW, and derives a label from the information carried in the BGP route.

```
R6#show bgp l2vpn vpls rd 24:30 ve-id 2 block-offset 1
BGP routing table entry for 24:30:VEID-2:Blk-1/136, version 10
Paths: (1 available, best #1, table L2VPN-VPLS-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 2
  Local
    24.0.0.2 (metric 20) from 24.0.0.2 (24.0.0.2)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      AGI version(0), VE Block Size(10) Label Base(2026)
      Extended Community: RT:24:3 L2VPN L2:0x0:MTU-1500
      mpls labels in/out exp-null/2026
      rx pathid: 0, tx pathid: 0x0
```

```
R6#show l2vpn atom vc
```

		Service			
Interface	Peer ID	VC ID	Type	Name	Status
pw100002	2	3	vfi	VPLS	UP

Tracing the path from CSR3 to CSR1 over L2VPN, we can see the label that CSR8 imposes is 5019. this is the PW label allocated by CSR5 via tLDP.

```
R8#show l2vpn atom binding 13.0.0.5
Destination Address: 13.0.0.5, VC ID: 3
Local Label: 8017
  Cbit: 1, VC Type: Ethernet, GroupID: n/a
  MTU: 1500, Interface Desc: n/a
  VCCV: CC Type: RA [2], TTL [3]
        CV Type: LSPV [2]
Remote Label: 5019
  Cbit: 1, VC Type: Ethernet, GroupID: n/a
  MTU: 1500, Interface Desc: n/a
  VCCV: CC Type: RA [2], TTL [3]
        CV Type: LSPV [2]
```

Since CSR8 and CSR5 are directly connected, no LDP label is imposed (imp-null is signaled). The MPLS PW details show a single label being imposed which is the PW label.

```
R8#show l2vpn atom vc vcid 3 detail | include label_stack
Output interface: Gi2.558, imposed label stack {5019}
```

```
R8#show mpls ldp bindings 13.0.0.5 32
lib entry: 13.0.0.5/32, rev 8
local binding: label: 8002
remote binding: lsr: 13.0.0.5:0, label: imp-null
remote binding: lsr: 13.0.0.11:0, label: 91000
remote binding: lsr: 13.0.0.12:0, label: 92004
```

When CSR5 receives labeled packets, it removes all labels and sends the traffic onto the service-instance tied to CSR6.

```
R5#show mpls forwarding-table labels 5019
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
5019	No Label	l2ckt(1)	902	none	point2point

```
R5#show l2vpn atom vc vcid 3 detail | section VPLS
Member of vfi service VPLS
Bridge-Domain id: 3
Service id: 0x88000001
```

```
R5#show bridge-domain 3
Bridge-domain 3 (2 ports in all)
State: UP                               Mac learning: Enabled
Aging-Timer: 300 second(s)
GigabitEthernet2 service instance 30
```

```

vfi VPLS neighbor 13.0.0.8 3
AED MAC address      Policy Tag      Age Pseudoport
0  0050.56A9.1AAA forward dynamic 200 GigabitEthernet2.EFP30
1  FFFF.FFFF.FFFF flood  static  0  OLIST_PTR:0xe808c400
0  0050.56A9.8CCF forward dynamic 202 VPLS.1004011

```

CSR6 is the ingress LSR and receives the Ethernet frame as if it came from the customer. Two labels are imposed: the bottom label was derived from BGP to represent the PW, and the top label was derived from LDP. Specifically, the top label is XRv4's local label for 24.0.0.2/32 as shown below.

```

R6#show l2vpn atom vc vcid 3 detail | include label_stack
Output interface: Gi2.564, imposed label stack {94009 2031}

R6#show mpls ldp bindings 24.0.0.2 32 neighbor 24.0.0.14
lib entry: 24.0.0.2/32, rev 14
remote binding: lsr: 24.0.0.14:0, label: 94009

```

XRv4 is a P router along this LSP and performs PHP to expose label 2031 to CSR2.

```

RP/0/0/CPU0:XRv4#show mpls forwarding labels 94009
Local   Outgoing   Prefix           Outgoing   Next Hop        Bytes
Label   Label       or ID           Interface  Next Hop        Switched
-----
94009 Pop          24.0.0.2/32     Gi0/0/0/0.524 24.2.14.2      3552757

```

Upon receipt, CSR2 removes all labels and forwards the frame to CSR1, the end customer.

```

R2#show mpls forwarding-table labels 2031
Local   Outgoing   Prefix           Bytes Label   Outgoing   Next Hop
Label   Label       or Tunnel Id     Switched      interface
2031    No Label    lbl-blk-id(2:5)  1090          none       point2point

```

Using MPLS OAM, we can quickly verify the data plane for each PW individually before attempting to test end-to-end connectivity.

```

R8#ping mpls pseudowire 13.0.0.5 3
Sending 5, 72-byte MPLS Echos to 13.0.0.5,
    timeout is 2 seconds, send interval is 0 msec:
[snip]

Type escape sequence to abort.
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/4/8 ms
Total Time Elapsed 21 ms

```

Since the CW is not supported over BGP signaled PWs within XE, we can use the IPv4 FEC to at least trace the transport path inside AS 24.

```
R2#ping mpls ipv4 24.0.0.6/32 source 24.0.0.2
Sending 5, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
    timeout is 2 seconds, send interval is 0 msec:
[snip]

Type escape sequence to abort.
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/5/8 ms
Total Time Elapsed 28 ms
```

Last, we can verify that CSR3 and CSR1 can communicate over the inter-AS L2VPN. Traceroute reveals that the two routers are one hop away (directly connected at layer 3), which is the design goal.

```
R3#traceroute vrf VPLS 10.0.0.1
Type escape sequence to abort.
Tracing the route to 10.0.0.1
VRF info: (vrf in name/id, vrf out name/id)
 1 10.0.0.1 11 msec 9 msec 9 msec
```

8.4.1.3 MVPN – GRE (Profile 0) and mLDP (Profile 1)

As we have seen in the L3VPN and L2VPN sections, the two ASes can use completely different mechanisms for transporting traffic between their PE and AS boundary routers. For this test, we will transport multicast traffic within VRFs EIGRP and OSPF between customer sites. AS 13 will use MVPN profile 0 which relies on PIM for c-mcast signaling and data-plane encapsulation. AS 24 will use MVPN profile 1 which relies on PIM for c-mcast signaling but uses MPLS for transporting label switched multicast (LSM). This is a simple MVPN profile that doesn't require BGP auto-discovery as the LSM root is hard-coded for the MP2MP tree. These delivery trees are build using mLDP; all of these MVPN technologies are covered in detail later. Initially, we will not enable data MDT support in AS 13 nor P2MP S-PMSI support in AS 24; all multicast transport will follow the default MDT.

First, I define default MDT groups for VRFs OSPF and EIGRP. Using ASM obviates the need for IPv4 MDT or IPv4 MVPN AFI negotiation with BGP, but adds more complexity to the MVPN. I will use the ASM method as it is less common and more difficult. VRF OSPF will use ASM group 225.0.0.2 while VRF EIGRP uses ASM group 225.0.0.3.

```
! CSR8 and CSR5
vrf definition OSPF
  address-family ipv4
    mdt default 225.0.0.2
  address-family ipv6
    mdt default 225.0.0.2
```



```

! CSR5 only
vrf definition EIGRP
  address-family ipv4
    mdt default 225.0.0.3
  address-family ipv6
    mdt default 225.0.0.3

! XRv2 and XRv1
multicast-routing
vrf EIGRP
  address-family ipv4
    mdt default ipv4 225.0.0.3
  address-family ipv6
    mdt default ipv4 225.0.0.3

! XRv1 only
multicast-routing
vrf OSPF
  address-family ipv4
    mdt default ipv4 225.0.0.2
  address-family ipv6
    mdt default ipv4 225.0.0.2

```

We can do a quick spot-check to ensure the default MDTs are built properly. On CSR5, we look at 225.0.0.2 with XRv1 and CSR8 as sources. Both of them are inside VRF OSPF, but XRv2 is not. This makes sense as XRv2 is not a PE for the OSPF VPN.

```

R5#show ip mroute 225.0.0.2 13.0.0.11 | begin \
(13.0.0.11, 225.0.0.2), 00:06:16/00:02:37, flags: JTZ
  Incoming interface: GigabitEthernet2.551, RPF nbr 13.5.11.11
  Outgoing interface list:
    MVRF OSPF, Forward/Sparse, 00:06:16/00:02:43

```

```

R5#show ip mroute 225.0.0.2 13.0.0.8 | begin \
(13.0.0.8, 225.0.0.2), 00:12:14/00:01:05, flags: JTZ
  Incoming interface: GigabitEthernet2.558, RPF nbr 13.5.8.8
  Outgoing interface list:
    MVRF OSPF, Forward/Sparse, 00:12:14/00:02:45

```

The same set of checks applies to 225.0.0.3 which services VRF EIGRP, but the members include XRv1 and XRv2 now (not CSR8). Each loopback joined the proper (*,G) tree rooted at XRv2, but once the c-mcast PIM signaling was sent, the SPT switchover occurred for all multicast state. That is why the 'J' flag is set on all of these entries.

```

R5#show ip mroute 225.0.0.3 13.0.0.11 | begin \
(13.0.0.11, 225.0.0.3), 00:09:26/00:02:13, flags: JTZ
  Incoming interface: GigabitEthernet2.551, RPF nbr 13.5.11.11

```

Outgoing interface list:

MVRF EIGRP, Forward/Sparse, 00:09:26/00:02:33

```
R5#show ip mroute 225.0.0.3 13.0.0.12 | begin \  
(13.0.0.12, 225.0.0.3), 00:14:35/00:02:42, flags: JTZ  
Incoming interface: GigabitEthernet2.558, RPF nbr 13.5.8.8  
Outgoing interface list:  
MVRF EIGRP, Forward/Sparse, 00:14:35/00:00:24
```

With the MDTs properly built, we will verify c-mcast PIM neighbors within each VPN. For EIGRP, we expect to see XRv1 and XRv2. For OSPF, we expect to see XRv1 and CSR8.

```
R5#show ip pim vrf EIGRP neighbor | begin ^Neigh  
Neighbor      Interface      Uptime/Expires  Ver  DR  
Address                               Prio/Mode  
10.5.6.6      GigabitEthernet2.5563  01:37:08/00:01:28 v2    1 / DR S P G  
10.5.7.7      GigabitEthernet2.5573  01:37:08/00:01:33 v2    1 / DR S P G  
13.0.0.11     Tunnel5               00:11:13/00:01:20 v2    1 / P G  
13.0.0.12     Tunnel5               00:16:17/00:01:43 v2    1 / DR P G
```

```
R5#show ip pim vrf OSPF neighbor | begin ^Neigh  
Neighbor      Interface      Uptime/Expires  Ver  DR  
Address                               Prio/Mode  
10.5.6.6      GigabitEthernet2.5562  01:37:11/00:01:35 v2    1 / DR S P G  
10.5.7.7      GigabitEthernet2.5572  01:37:11/00:01:32 v2    1 / DR S P G  
13.0.0.11     Tunnel4               00:11:19/00:01:27 v2    1 / DR P G  
13.0.0.8      Tunnel4               00:17:14/00:01:44 v2    1 / S P G
```

We perform the same verification for IPv6 PIM neighbors. This will be IPv6 customer multicast tunneled inside IPv4 provider multicast. This shows that the IPv6 c-mcast hellos are transiting the SP core properly.

```
R5#show ipv6 pim vrf EIGRP neighbor | begin ^Neigh  
Neighbor Address      Interface      Uptime    Expires  Mode  DR pri  
FE80::6              Gi2.5563      01:37:18  00:01:21 B G    DR 1  
FE80::7              Gi2.5573      01:37:18  00:01:20 B G    DR 1  
::FFFF:13.0.0.11     Tunnel5       00:09:05  00:01:32 B G    1  
::FFFF:13.0.0.12     Tunnel5       00:09:03  00:01:44 B G    DR 1
```

```
R5#show ipv6 pim vrf OSPF neighbor | begin ^Neigh  
Neighbor Address      Interface      Uptime    Expires  Mode  DR pri  
FE80::6              Gi2.5562      01:37:22  00:01:39 B G    DR 1  
FE80::7              Gi2.5572      01:37:22  00:01:36 B G    DR 1  
::FFFF:13.0.0.8      Tunnel4       00:09:10  00:01:27 B G    1  
::FFFF:13.0.0.11     Tunnel4       00:09:09  00:01:22 B G    DR 1
```

Next, I will configure AS 24 with mLDP. XRv4 is the root for all VPNs within the AS. The mLDP configuration is identical on CSR2, CSR6, and CSR7 as all three routers host both VRF EIGRP and OSPF. The critical step to configuring mLDP is to identify a VPN ID. Without it, mLDP will not work, and there isn't an error message to help reveal it.

```
! CSR2, CSR6, CSR7
vrf definition EIGRP
  vpn id 24:3
  address-family ipv4
    mdt preference mldp
    mdt default mpls mldp 24.0.0.14
  address-family ipv6
    mdt preference mldp
    mdt default mpls mldp 24.0.0.14

vrf definition OSPF
  vpn id 24:2
  address-family ipv4
    mdt preference mldp
    mdt default mpls mldp 24.0.0.14
  address-family ipv6
    mdt preference mldp
    mdt default mpls mldp 24.0.0.14
```

XRv4 is the MP2MP root and also a PE. Since XRv doesn't support LSM in the PE role, the configuration below does nothing (reference only).

```
! XRv4
vrf EIGRP
  vpn id 24:3

multicast-routing
  vrf EIGRP
    address-family ipv4
      mdt default mldp ipv4 24.0.0.14
    address-family ipv6
      mdt default mldp ipv4 24.0.0.14
```

A quick way to verify the mLDP configuration is to check the root. It has no upstream peers; the root of the tree never does. For the 24:2 VPN (OSPF), it has 3 downstream peers. These would be CSR2, CSR6, and CSR7. For 24:3 (EIGRP), it has 4 downstream peers. This includes the 3 routers plus the VPN customer interface (PMSI), generally speaking.

```
RP/0/0/CPU0:XRv4#show mpls mldp database brief
```

LSM ID	Type	Root	Up	Down	Decoded	Opaque	Value
0x00002	MP2MP	24.0.0.14	0	3	[mdt	24:2	0]
0x00001	MP2MP	24.0.0.14	0	4	[mdt	24:3	0]

Since MP2MP trees are bidirectional, we can use OAM to verify the MDT from any PE. From CSR2, we can use this as a discovery mechanism to find all members of a given MVPN instance. Inside of VPN 24:2, only the ASBRs respond, since the third member is CSR2 (the source). Inside of VPN 24:3, the ASBRs and XRv4 respond, because all of them are PEs for this MVPN instance.

```
R2#ping mpls mldp mp2mp 24.0.0.14 mdt 24:2 0
mp2mp Root node addr 24.0.0.14
Opaque type MDT, oui:index 0x24:02, mdtnum 0
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
    timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]
```

```
Request #1
! reply addr 24.7.14.7
! reply addr 24.6.14.6
```

```
R2#ping mpls mldp mp2mp 24.0.0.14 mdt 24:3 0
mp2mp Root node addr 24.0.0.14
Opaque type MDT, oui:index 0x24:03, mdtnum 0
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
    timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]
```

```
Request #1
! reply addr 24.2.14.14
! reply addr 24.6.14.6
! reply addr 24.7.14.7
```

As a final check, we verify that the PEs have PIM neighbors within their respective VPNs. From CSR7's perspective, there are 2 neighbors in VPN 24:2 (OSPF) which include CSR2 and CSR6 only. VPN 24:3 (EIGRP) includes XRv4 as well, for a total of 3.

```
R7#show ip pim vrf OSPF neighbor | begin ^Neigh
Neighbor          Interface          Uptime/Expires    Ver    DR
Address
10.5.7.5           GigabitEthernet2.5572  02:00:38/00:01:37 v2      1 / S P G
24.0.0.6           Lspvif1              00:15:15/00:01:16 v2      1 / S P G
24.0.0.2           Lspvif1              00:15:15/00:01:44 v2      1 / S P G

R7#show ip pim vrf EIGRP neighbor | begin ^Neigh
Neighbor          Interface          Uptime/Expires    Ver    DR
Address
10.5.7.5           GigabitEthernet2.5573  02:00:42/00:01:33 v2      1 / S P G
24.0.0.14          Lspvif0            00:13:07/00:01:36 v2      1 / DR P G
```

24.0.0.6	Lspvif0	00:14:58/00:01:24	v2	1 / S P G
24.0.0.2	Lspvif0	00:14:58/00:01:24	v2	1 / S P G

For completeness, we quickly check IPv6 as well and notice the same neighbors.

```
R7#show ipv6 pim vrf OSPF neighbor | begin ^Neigh
```

Neighbor Address	Interface	Uptime	Expires	Mode	DR	pri
FE80::5	Gi2.5572	02:00:45	00:01:32	B G		1
::FFFF:24.0.0.2	Lspvif1	00:15:25	00:01:39	B G		1
::FFFF:24.0.0.6	Lspvif1	00:15:25	00:01:39	B G		1

```
R7#show ipv6 pim vrf EIGRP neighbor | begin ^Neigh
```

Neighbor Address	Interface	Uptime	Expires	Mode	DR	pri
FE80::5	Gi2.5573	02:00:49	00:01:36	B G		1
::FFFF:24.0.0.2	Lspvif0	00:15:29	00:01:23	B G		1
::FFFF:24.0.0.6	Lspvif0	00:15:29	00:01:19	B G		1
::FFFF:24.0.0.14	Lspvif0	00:13:15	00:01:23	B G	DR	1

To prepare for customer ASM testing, we will configure CSR3 as the RP inside the EIGRP VPN for IPv4 and IPv6. Because XRv4 cannot support LSM PE functions, we also must modify RPF on XRv3 so that it can learn the RP information from CSR1. Without the static multicast route per AFI, the BSR messages from CSR1 are dropped due to RPF failure.

```
! CSR3
ip pim bsr-candidate Loopback0 0
ip pim rp-candidate Loopback0
ipv6 pim bsr candidate bsr ::10:3:3:3
ipv6 pim bsr candidate rp ::10:3:3:3
```

```
! XRv3
router static
 address-family ipv4 multicast
  10.3.3.3/32 10.1.13.1
 address-family ipv6 multicast
  ::10:3:3:3/128 fd00:10:1:13::1
```

```
R1#show ip pim rp mapping
PIM Group-to-RP Mappings
Group(s) 224.0.0.0/4
  RP 10.3.3.3 (?), v2
    Info source: 10.3.3.3 (?), via bootstrap, priority 0, holdtime 150
    Uptime: 00:03:35, expires: 00:01:52
```

```
RP/0/0/CPU0:XRv3#show pim rp mapping
PIM Group-to-RP Mappings
Group(s) 224.0.0.0/4
  RP 10.3.3.3 (?), v2
    Info source: 10.1.13.1 (?), elected via bsr, priority 0, holdtime 150
```

Uptime: 00:01:48, expires: 00:01:42

```
R1#show ipv6 pim group-map info-source bsr
IP PIM Group Mapping Table
(* indicates group mappings being used)
FF00::/8*
  SM, RP: ::10:3:3:3
  RPF: Gi2.512, FE80::2
  Info source: BSR From: ::10:3:3:3 (00:01:33), Priority: 192
  Uptime: 00:00:56, Groups: 0
```

```
RP/0/0/CPU0:XRv3#show pim ipv6 rp mapping ::10:3:3:3
PIM Group-to-RP Mappings
Group(s) ff00::/8
  RP ::10:3:3:3 (?), v2
  Info source: fe80::1 (?), elected via bsr, priority 192, holdtime 150
  Uptime: 00:04:27, expires: 00:02:03
```

Just by virtue of the BSR messages being passed from CSR3 to CSR1/XRv3 means that the inter-AS MVPN is probably working. However, we will test actual data traffic as well to ensure all PIM signaling functions (join, prune, etc) are working between ASes. XRv3 will be a receiver for group 225.13.13.13 on its loopback interface. This is an ASM membership report as the mode is EXCLUDE and no sources are specified.

```
! XRv3
router igmp
  interface Loopback0
    join-group 225.13.13.13
```

```
RP/0/0/CPU0:XRv3#show igmp groups 225.13.13.13 detail
Interface:      Loopback0
Group:          225.13.13.13
Uptime:         00:00:52
Router mode:    EXCLUDE (Expires: never)
Host mode:      EXCLUDE
Last reporter:  10.13.13.13
Source list is empty
```

XRv3 originates the C(*,G) join towards the RP towards CSR3 (RP) via CSR1. The static multicast route identifies CSR1 as the RPF neighbor towards 10.3.3.3/32, and we can confirm this by checking the RPF details.

```
RP/0/0/CPU0:XRv3#show pim topology 225.13.13.13 | begin 13\13
(*,225.13.13.13) SM Up: 00:00:25 RP: 10.3.3.3
JP: Join(00:00:24) RPF: GigabitEthernet0/0/0/0.513, 10.1.13.1 Flags: LH
  Loopback0                                00:00:25 fwd LI II LH
```

```
RP/0/0/CPU0:XRv3#show pim rpf 10.3.3.3
Table: IPv4-Multicast-default
* 10.3.3.3/32 [1/0]
    via GigabitEthernet0/0/0/0.513 with rpf neighbor 10.1.13.1
```

CSR1 sends the C(*,G) join to CSR2, the PE, as its RPF is based on the unicast EIGRP route to 10.3.3.3/32.

```
R1#show ip mroute 225.13.13.13 | begin \
(*, 225.13.13.13), 00:00:18/00:03:11, RP 10.3.3.3, flags: S
    Incoming interface: GigabitEthernet2.512, RPF nbr 10.1.2.2
    Outgoing interface list:
        GigabitEthernet2.513, Forward/Sparse, 00:00:18/00:03:11
```

```
R1#show ip rpf 10.3.3.3
RPF information for ? (10.3.3.3)
    RPF interface: GigabitEthernet2.512
    RPF neighbor: ? (10.1.2.2)
    RPF route/mask: 10.3.3.3/32
    RPF type: unicast (eigrp 3)
    Doing distance-preferred lookups across tables
    RPF topology: ipv4 multicast base, originated from ipv4 unicast base
```

CSR2 receives the C(*,G) join in the VPN and sends it over the default MP2MP tree towards CSR6. Since the MVPN instances are confined to each AS, CSR2 is not aware that XRv2 is the egress PE, and instead views CSR6 as having this role.

```
R2#show ip mroute vrf EIGRP 225.13.13.13 | begin \
(*, 225.13.13.13), 00:00:59/00:03:29, RP 10.3.3.3, flags: S
    Incoming interface: Lspvif0, RPF nbr 24.0.0.6
    Outgoing interface list:
        GigabitEthernet2.512, Forward/Sparse, 00:00:59/00:03:29
```

CSR6 receives the C(*,G) join and forwards it towards XRv1. CSR5 was not selected due to XRv1's eBGP route being older as shown below.

```
R6#show ip mroute vrf EIGRP 225.13.13.13 | begin \
(*, 225.13.13.13), 00:02:12/00:03:14, RP 10.3.3.3, flags: S
    Incoming interface: GigabitEthernet2.5613, RPF nbr 10.6.11.11
    Outgoing interface list:
        Lspvif0, Forward/Sparse, 00:02:12/00:03:14
```

```
R6#show bgp vpnv4 unicast vrf EIGRP 10.3.3.3/32
BGP routing table entry for 24:3:10.3.3.3/32, version 60
Paths: (2 available, best #2, table EIGRP)
    Advertised to update-groups:
        2          5
    Refresh Epoch 1
```

13

```
10.5.6.5 (via vrf EIGRP) from 10.5.6.5 (13.0.0.5)
  Origin incomplete, localpref 100, valid, external
  Extended Community: RT:24:3 0x8800:32768:0 0x8801:3:288
    0x8802:65281:2560 0x8803:1:1500 0x8806:0:167971843
  mpls labels in/out 6005/nolabel
  rx pathid: 0, tx pathid: 0
```

Refresh Epoch 1

13

```
10.6.11.11 (via vrf EIGRP) from 10.6.11.11 (13.0.0.11)
  Origin incomplete, localpref 100, valid, external, best
  Extended Community: RT:24:3 0x8800:32768:0 0x8801:3:288
    0x8802:65281:2560 0x8803:1:1500 0x8806:0:167971843
  mpls labels in/out 6005/nolabel
  rx pathid: 0, tx pathid: 0x0
```

XRv1 views this C(*,G) join as if it came from a CE device, so it is wrapped inside the default MDT and signaled towards XRv2. When XRv2 receives it, the join is passed back into the customer network towards CSR3.

```
RP/0/0/CPU0:XRv1#show pim vrf EIGRP topology 225.13.13.13 | begin 225
(*,225.13.13.13) SM Up: 00:05:05 RP: 10.3.3.3
JP: Join(00:00:46) RPF: mdtEIGRP,13.0.0.12 Flags:
  GigabitEthernet0/0/0/0.5613 00:05:05 fwd Join(00:03:17)
```

```
RP/0/0/CPU0:XRv2#show pim vrf EIGRP topology 225.13.13.13 | begin 225
(*,225.13.13.13) SM Up: 00:05:56 RP: 10.3.3.3
JP: Join(now) RPF: GigabitEthernet0/0/0/0.532,10.3.12.3 Flags:
  mdtEIGRP 00:05:56 fwd Join(00:02:35)
```

In the customer network, CSR3 is the RP and therefore the root of the customer shared tree. This shows the correct C(*,G) construction.

```
R3#show ip mroute 225.13.13.13 | begin \
(*, 225.13.13.13), 00:06:44/00:02:43, RP 10.3.3.3, flags: S
  Incoming interface: Null, RPF nbr 0.0.0.0
  Outgoing interface list:
    GigabitEthernet2.532, Forward/Sparse, 00:06:44/00:02:43
```

For simplicity, the host address 10.3.3.3 will also be the source of the multicast traffic. This will allow XRv3 to issue C(S,G) joins towards CSR1 as we added RPF fix-up routes on XRv3. The PIM registration and SPT switchover processes are not evaluated in detail as this is documented in many other places.

```
R3#ping ip
Target IP address: 225.13.13.13
Repeat count [1]: 100000
Datagram size [100]:
```



```

Timeout in seconds [2]: 1
Extended commands [n]: y
Interface [All]: loopback0
Time to live [255]:
Source address or interface: loopback0
[snip]

```

XRv3 gets the first few packets along the C(*,G) tree and then performs the SPT switchover. XRv3 shows no OIL interfaces but we know the traffic is being delivered to the loopback (process switched).

```

RP/0/0/CPU0:XRv3#show pim topology 225.13.13.13 10.3.3.3 | begin 3,225
(10.3.3.3,225.13.13.13)SPT SM Up: 00:02:25
JP: Join(00:00:24) RPF: GigabitEthernet0/0/0/0.513,10.1.13.1 Flags:
KAT(00:01:05) RA
No interfaces in immediate olist

```

Looking briefly at the packet counters, we see exactly 1 100-byte packet received along the C(*,G) tree which triggered the SPT switchover. All future packets arrive along the C(S,G) tree. The size of each packet is exactly 100 bytes as expected. Seeing packets here is a sign that the test is working. For practice, we will verify the entire path though we know the signaling must be correct.

```

RP/0/0/CPU0:XRv3#show mfib route 225.13.13.13 | begin 225
(*,225.13.13.13),    Flags:  C
  Up: 00:17:08
  Last Used: 00:03:06
  SW Forwarding Counts: 1/1/100
  SW Replication Counts: 1/0/0
  SW Failure Counts: 0/0/0/0/0
  Loopback0 Flags:  IC NS EG, Up:00:17:08
  GigabitEthernet0/0/0/0.513 Flags:  A NS, Up:00:17:08

(10.3.3.3,225.13.13.13),    Flags:
  Up: 00:03:06
  Last Used: 00:00:00
  SW Forwarding Counts: 186/186/18600
  SW Replication Counts: 186/0/0
  SW Failure Counts: 0/0/0/0/0
  Loopback0 Flags:  IC NS EG, Up:00:03:06
  GigabitEthernet0/0/0/0.513 Flags:  A, Up:00:03:06

```

CSR1 has the proper C(S,G) state and its packet counters are incrementing along this tree. Again, only one packet traversed the C(*,G) tree. XE accounts for the layer 2 encapsulation (14 bytes Ethernet + 4 bytes dot1q) in its packet counters, where XR does not. Otherwise, we can see the packet is exactly 100 bytes as expected.

```

R1#show ip mroute 225.13.13.13 10.3.3.3 | begin \

```

```
(10.3.3.3, 225.13.13.13), 00:05:12/00:02:22, flags: T
Incoming interface: GigabitEthernet2.512, RPF nbr 10.1.2.2
Outgoing interface list:
GigabitEthernet2.513, Forward/Sparse, 00:05:12/00:03:17
```

```
R1#show ip mroute 225.13.13.13 count | begin ^Group
Group: 225.13.13.13, Source count: 1, Packets forwarded: 324, Packets
received: 324
RP-tree: Forwarding: 1/0/118/0, Other: 1/0/0
Source: 10.3.3.3/32, Forwarding: 323/1/118/0, Other: 323/0/0
```

CSR2 now has the C(S,G) within the VPN and receives packets from the PMSI. The mLDP database shows traffic coming down from the MP2MP root using label 2011 from XRv4.

```
R2#show ip mroute vrf EIGRP 225.13.13.13 10.3.3.3 | begin \((
(10.3.3.3, 225.13.13.13), 00:07:41/00:01:56, flags: T
Incoming interface: Lspvif0, RPF nbr 24.0.0.6
Outgoing interface list:
GigabitEthernet2.512, Forward/Sparse, 00:07:41/00:02:40

R2#show mpls mldp database opaque_type mdt 24:3 0
LSM ID : 1 (RNR LSM ID: 2)    Type: MP2MP    Uptime : 14:40:11
FEC Root      : 24.0.0.14
Opaque decoded : [mdt 24:3 0]
Opaque length  : 11 bytes
Opaque value   : 02 000B 0000240000000300000000
RNR active LSP : (this entry)
Upstream client(s) :
  24.0.0.14:0    [Active]
    Expires      : Never          Path Set ID   : 1
    Out Label (U) : 94003         Interface    : GigabitEthernet2.524*
    Local Label (D): 2011         Next Hop      : 24.2.14.14
Replication client(s):
MDT (VRF EIGRP)
  Uptime        : 14:40:11       Path Set ID   : 2
  Interface     : Lspvif0
```

A quick EPC on the link to XRv4 shows singly-labeled multicast packets entering CSR2. CSR2 removes the labels and forwards the raw IP multicast towards CSR1 as a “replication client”. The packet is 122 bytes which accounts for the 18 bytes of Ethernet/dot1q encapsulation and a single MP2MP label. The label value 2011 (0x7DB) is shown in yellow with the IP source/destination shown in green.

```
R2#show monitor capture CAP buffer detailed
 3  122    0.719008  00:50:56:A9:86:2A -> 00:50:56:A9:BE:8A MPLS unicast
0000: 005056A9 BE8A0050 56A9862A 81000DC4 .PV....PV...*....
0010: 8847007D B1FB4500 006405B3 0000FC01 .G.}..E..d.....
0020: BDC50A03 0303E10D 0D0D0800 F9B20018 .....
0030: 03100000 00008575 FBF9ABCD ABCDABCD .....u.....
```

As the ASBR, CSR6 receives IP multicast from AS 13 and encapsulates it inside MPLS along the MP2MP tree. This uses label 94012 towards XRv4 as this traffic is being sent upstream towards the root.

```
R6#show ip mroute vrf EIGRP 225.13.13.13 10.3.3.3 | begin \((
(10.3.3.3, 225.13.13.13), 00:19:56/00:03:25, flags: T
  Incoming interface: GigabitEthernet2.5613, RPF nbr 10.6.11.11
  Outgoing interface list:
    Lspvif0, Forward/Sparse, 00:19:56/00:03:15

R6#show mpls mldp database opaque_type mdt 24:3 0
LSM ID : 1 (RNR LSM ID: 2)   Type: MP2MP   Uptime : 14:45:06
  FEC Root      : 24.0.0.14
  Opaque decoded : [mdt 24:3 0]
  Opaque length  : 11 bytes
  Opaque value   : 02 000B 000024000000003000000000
  RNR active LSP : (this entry)
  Upstream client(s) :
    24.0.0.14:0      [Active]
      Expires        : Never          Path Set ID   : 1
      Out Label (U)   : 94012         Interface    : GigabitEthernet2.564*
      Local Label (D) : 6017          Next Hop      : 24.6.14.14
  Replication client(s):
    MDT (VRF EIGRP)
      Uptime          : 14:45:06      Path Set ID   : 2
      Interface       : Lspvif0
```

For completeness, we use EPC outbound on CSR6 towards XRv4 to verify this. The label 94012 is shown in yellow with the VPN multicast information in green. Also notice that the packet is 122 bytes, so there is no label stacking in this design.

```
R6#show monitor capture CAP buffer detailed
  2  122  1.000000  00:50:56:A9:DE:0D -> 00:50:56:A9:86:2A MPLS unicast
0000:  005056A9 862A0050 56A9DE0D 81000DEC  .PV...*.PV.....
0010:  884716F3 C1FC4500 0064069D 0000FC01  .G....E..d.....
0020:  BCD0A03 0303E10D 0D0D0800 663B0018  .....f;...
0030:  03FA0000 00008579 8E83ABCD ABCDABCD  .....y.....
```

Like CSR2, XRv1 receives traffic from the PMSI and forwards it towards the “customer” router CSR6. The traffic is following the default MDT since data MDTs were not configured for this test. This uses the P(S,G) of (13.0.0.2, 225.0.0.3) from XRv2. The packet counters indicate 0 packets in and many packets out; this is because traffic technically arrives on a global P-multicast group and goes out on a VPN c-multicast group. Only 1 100-byte packet was seen on the shared customer tree as expected.

```
RP/0/0/CPU0:XRv1#show pim vrf EIGRP topology 225.13.13.13 | begin 3,225
(10.3.3.3,225.13.13.13)SPT SM Up: 00:18:39
JP: Join(00:00:12) RPF: mdtEIGRP,13.0.0.12 Flags:
```

```
GigabitEthernet0/0/0/0.5613 00:18:39 fwd Join(00:02:32)
```

```
RP/0/0/CPU0:XRv1#show mfib vrf EIGRP route 225.13.13.13 | begin 225
(*,225.13.13.13), Flags: C
```

```
Up: 00:32:55
```

```
Last Used: never
```

```
SW Forwarding Counts: 0/1/100
```

```
SW Replication Counts: 0/1/100
```

```
SW Failure Counts: 0/0/0/0/0
```

```
mdtEIGRP Flags: A MI, Up:00:32:55
```

```
GigabitEthernet0/0/0/0.5613 Flags: NS EG, Up:00:32:55
```

```
(10.3.3.3,225.13.13.13), Flags:
```

```
Up: 00:23:56
```

```
Last Used: never
```

```
SW Forwarding Counts: 0/1435/143500
```

```
SW Replication Counts: 0/1435/143500
```

```
SW Failure Counts: 0/0/0/0/0
```

```
mdtEIGRP Flags: A MI, Up:00:23:56
```

```
GigabitEthernet0/0/0/0.5613 Flags: NS EG, Up:00:23:56
```

Looking at the P(S,G) for the default MDT, we see the opposite effect. Many packets come in but nothing goes out. The reason this number is much greater than the C(S,G) counters is because this accounts for all of the PIM signaling messages plus all other VPN flows.

```
RP/0/0/CPU0:XRv1#show mfib route 225.0.0.3 13.0.0.12 | begin 225
```

```
(13.0.0.12,225.0.0.3), Flags: MD MH CD DT DTV6
```

```
Up: 15:14:26
```

```
Last Used: 00:00:00
```

```
SW Forwarding Counts: 6087/0/0
```

```
SW Replication Counts: 6087/0/0
```

```
SW Failure Counts: 0/0/0/0/0
```

```
Loopback0 Flags: NS EG, Up:15:14:26
```

```
GigabitEthernet0/0/0/0.581 Flags: A, Up:15:14:26
```

XRv2 has the C(S,G) join for this group and receives packets from the CE, CSR3. Packets sent to the PMSI on the ingress PE are accounted for in the in and out directions as shown below. This is just the way XR accounts for packets; the egress PE only showed the outbound packet counter increasing.

```
RP/0/0/CPU0:XRv2#show pim vrf EIGRP topology 225.13.13.13 | begin 3,225
```

```
(10.3.3.3,225.13.13.13)SPT SM Up: 00:28:24
```

```
JP: Join(00:00:23) RPF: GigabitEthernet0/0/0/0.532,10.3.12.3 Flags:
```

```
mdtEIGRP 00:28:24 fwd Join(00:03:07)
```

```
RP/0/0/CPU0:XRv2#show mfib vrf EIGRP route 225.13.13.13 | begin 225
```

```
(*,225.13.13.13), Flags: C
```

```
Up: 00:38:09
```

```
Last Used: 00:29:09
```

```

SW Forwarding Counts: 1/1/100
SW Replication Counts: 1/0/0
SW Failure Counts: 0/0/0/0/0
mdtEIGRP Flags: F NS MI, Up:00:38:09
GigabitEthernet0/0/0/0.532 Flags: A, Up:00:38:09

```

```

(10.3.3.3,225.13.13.13), Flags:
Up: 00:29:09
Last Used: 00:00:00
SW Forwarding Counts: 1748/1748/174800
SW Replication Counts: 1748/0/0
SW Failure Counts: 0/0/0/0/0
mdtEIGRP Flags: F NS MI, Up:00:29:09
GigabitEthernet0/0/0/0.532 Flags: A, Up:00:29:09

```

If we stop the ping on CSR3 to quickly check the signaling details, we can see the C(S,G) along with the appropriate packet counters. This concludes the inter-AS option A ASM test.

```

R3#show ip mroute 225.13.13.13 10.3.3.3 | begin \
(10.3.3.3, 225.13.13.13), 00:31:35/00:03:21, flags: T
Incoming interface: Loopback0, RPF nbr 0.0.0.0
Outgoing interface list:
GigabitEthernet2.532, Forward/Sparse, 00:31:35/00:02:54

```

```

R3#show ip mroute 225.13.13.13 10.3.3.3 count | begin ^Group
Group: 225.13.13.13, Source count: 1, Packets forwarded: 1886, Packets
received: 1886
Source: 10.3.3.3/32, Forwarding: 1886/0/100/0, Other: 1886/0/0

```

Since VRF OSPF has no RP to support ASM, we will use SSM there. CSR4 will issue an MLDv2 report to receive traffic from group FF33::9 from CSR9's loopback. We verify the details of the MLDv2 join to see it is operating in INCLUDE mode with source ::10:9:9:9.

```

! CSR4
interface Loopback0
ipv6 mld join-group FF33::9 ::10:9:9:9

```

```

R4#show ipv6 mld groups ff33::9 detail
Interface: Loopback0
Group: FF33::9
Uptime: 00:00:36
Router mode: INCLUDE
Host mode: INCLUDE
Last reporter: FE80::21E:49FF:FE80:B400
Group source list:
Source Address      Uptime    Expires    Fwd  Flags
::10:9:9:9          00:00:36  00:03:44  Yes  Remote Local 2D

```

Since CSR4 is also an IPv6 multicast router, it issues the C(S,G) join towards CSR8, which is in the reverse path towards CSR9. This implies the OSPFv3 sham-link tested earlier is still operational so that traffic prefers the MPLS network over the slow backdoor link.

```
R4#show ipv6 mroute ff33::9 ::10:9:9:9 | begin \  
(::10:9:9:9, FF33::9), 00:03:00/never, flags: sLTI  
Incoming interface: GigabitEthernet2.548  
RPF nbr: FE80::8  
Immediate Outgoing interface list:  
Loopback0, Forward, 00:03:00/never
```

CSR8 receives this C(S,G) join and sends it onward towards CSR5. At this point, it isn't exactly clear why CSR8 selected CSR5 over XRv1, so we check the BGP table to only find one route for ::10:9:9:9.

```
R8#show ipv6 mroute vrf OSPF ff33::9 ::10:9:9:9 | begin \  
(::10:9:9:9, FF33::9), 00:04:12/00:03:19, flags: sT  
Incoming interface: Tunnel4  
RPF nbr: ::FFFF:13.0.0.5  
Immediate Outgoing interface list:  
GigabitEthernet2.548, Forward, 00:04:12/00:03:19
```

```
R8#show bgp vpnv6 unicast vrf OSPF ::10:9:9:9/128  
BGP routing table entry for [13:2>::10:9:9:9/128, version 112  
Paths: (1 available, best #1, table OSPF)  
Not advertised to any peer  
Refresh Epoch 1  
24  
::FFFF:13.0.0.5 (metric 2) (via default) from 13.0.0.12 (13.0.0.12)  
Origin incomplete, metric 0, localpref 100, valid, internal, best  
Extended Community: RT:13:2 OSPF ROUTER ID:10.2.9.2:0  
OSPF RT:0.0.0.0:2:0  
Originator: 13.0.0.5, Cluster list: 13.0.0.12  
Connector Attribute: count=1  
type 1 len 12 value 13:2:13.0.0.5  
mpls labels in/out nlabel/5021  
rx pathid: 0, tx pathid: 0x0
```

The reason for this reduced visibility within the AS is the route-reflector. XRv2 receives both paths but selects CSR5 over XRv1 due to a lower BGP RID. Only this path is advertised by the RR because add-path is not in use. This isn't specific to inter-AS MVPN but is worth noting that the unicast VPN routing influences the MVPN routing heavily.

```
RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast rd 13:2 ::10:9:9:9/128 | begin 24,  
24, (Received from a RR-client)  
13.0.0.5 (metric 3) from 13.0.0.5 (13.0.0.5)  
Received Label 5021
```

```

Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, not-in-vrf
Received Path ID 0, Local Path ID 1, version 305
Extended community: OSPF router-id:10.2.9.2 OSPF route-type:0:2:0x0
RT:13:2
Connector: type: 1, Value:13:2:13.0.0.5
Path #2: Received by speaker 0
Not advertised to any peer
24, (Received from a RR-client)
13.0.0.11 (metric 3) from 13.0.0.11 (13.0.0.11)
Received Label 91003
Origin incomplete, localpref 100, valid, internal, import-candidate,
not-in-vrf
Received Path ID 0, Local Path ID 0, version 0
Extended community: OSPF router-id:10.2.9.2 OSPF route-type:0:2:0x0
RT:13:2
Connector: type: 1, Value:13:2:13.0.0.11

```

CSR5 receives the C(S,G) join over the default MDT and sends it forward to CSR6. CSR6 is chosen over CSR7 since it is the oldest eBGP route.

```

R5#show ipv6 mroute vrf OSPF ff33::9 ::10:9:9:9 | begin \
(::10:9:9:9, FF33::9), 00:10:47/00:02:41, flags: sT
Incoming interface: GigabitEthernet2.5562
RPF nbr: FE80::6
Immediate Outgoing interface list:
Tunnel4, Forward, 00:10:47/00:02:41

R5#show bgp vpnv6 unicast vrf OSPF ::10:9:9:9/128
BGP routing table entry for [13:2>::10:9:9:9/128, version 178
Paths: (2 available, best #2, table OSPF)
Advertised to update-groups:
2 3
Refresh Epoch 1
24
FD00:10:5:7::7 (FE80::7) (via vrf OSPF) from FD00:10:5:7::7 (24.0.0.7)
Origin incomplete, localpref 100, valid, external
Extended Community: RT:13:2 OSPF ROUTER ID:10.2.9.2:0
OSPF RT:0.0.0.0:2:0
mpls labels in/out 5021/nolabel
rx pathid: 0, tx pathid: 0
Refresh Epoch 2
24
FD00:10:5:6::6 (FE80::6) (via vrf OSPF) from FD00:10:5:6::6 (24.0.0.6)
Origin incomplete, localpref 100, valid, external, best
Extended Community: RT:13:2 OSPF ROUTER ID:10.2.9.2:0
OSPF RT:0.0.0.0:2:0
mpls labels in/out 5021/nolabel
rx pathid: 0, tx pathid: 0x0

```

CSR6 sends the C(S,G) join to CSR2 over the mLDP MP2MP tree. This uses the same MP2MP tree we traced earlier, so we know the transit path is across XRv4. CSR2 receives the C(S,G) join and sends it to CSR9, which is the CE.

```
R6#show ipv6 mroute vrf OSPF ff33::9 ::10:9:9:9 | begin \  
(::10:9:9:9, FF33::9), 00:12:26/00:03:03, flags: sT  
Incoming interface: Lspvif1  
RPF nbr: ::FFFF:24.0.0.2  
Immediate Outgoing interface list:  
GigabitEthernet2.5562, Forward, 00:12:26/00:03:03
```

```
R2#show ipv6 mroute vrf OSPF ff33::9 ::10:9:9:9 | begin \  
(::10:9:9:9, FF33::9), 00:12:57/00:02:35, flags: sT  
Incoming interface: GigabitEthernet2.529  
RPF nbr: FE80::9  
Immediate Outgoing interface list:  
Lspvif1, Forward, 00:12:57/00:02:35
```

CSR9 is the root of the C-SPT and expects to see traffic enter from loopback0.

```
R9#show ipv6 mroute ff33::9 ::10:9:9:9 | begin \  
(::10:9:9:9, FF33::9), 00:13:58/00:02:31, flags: sT  
Incoming interface: Loopback0  
RPF nbr: FE80::21E:E5FF:FEA2:5700  
Immediate Outgoing interface list:  
GigabitEthernet2.529, Forward, 00:13:58/00:02:31
```

One benefit of SSM is that once the delivery tree is built, there is no more signaling. We don't need to trace the control path again since we know the SPT is built; there was never a C(*,G) tree or RP, implying no need for the registration or SPT switchover processes. We can originate traffic on CSR9 as shown below.

```
R9#ping ipv6  
Target IPv6 address: ff33::9  
Repeat count [5]: 100000  
Datagram size [100]:  
Timeout in seconds [2]: 1  
Extended commands? [no]: y  
Source address or interface: ::10:9:9:9  
[snip]  
Output Interface: loopback0
```

For brevity, I will verify a few key routers. The ingress PE is CSR2 and is showing packets received from the customer. These are being MPLS-encapsulated and sent towards XRv4 along the MP2MP mLDP tree. The label used is 94005 since traffic is flowing upstream.


```

R2#show ipv6 mroute vrf OSPF ff33::9 ::10:9:9:9 count | begin ^Group
Group: FF33::9
  Source: ::10:9:9:9,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 15/1/118/0, Other: 0/0/0
  Totals - Source count: 1, Packet count: 15

R2#show mpls mldp database opaque_type mdt 24:2
LSM ID : 3 (RNR LSM ID: 4)   Type: MP2MP   Uptime : 15:29:47
FEC Root      : 24.0.0.14
Opaque decoded : [mdt 24:2 0]
Opaque length  : 11 bytes
Opaque value   : 02 000B 000024000000002000000000
RNR active LSP : (this entry)
Upstream client(s) :
  24.0.0.14:0      [Active]
    Expires        : Never           Path Set ID   : 3
    Out Label (U)  : 94005           Interface    : GigabitEthernet2.524*
    Local Label (D): 2014            Next Hop     : 24.2.14.14
Replication client(s):
  MDT (VRF OSPF)
    Uptime         : 15:29:47        Path Set ID   : 4
    Interface      : Lspvif1

```

We confirm this using EPC on CSR2. Of note, there is always an IPv6-exp-null label (value 2) as the bottom label. I assume this is used as a shim to indicate that the packet is IPv6. Since IPv4 and IPv6 would otherwise use the same (and only) MP2MP mLDP labels, this can be used to identify an MPLS packet as carrying IPv6 so the proper MFIB lookup can occur. The labels are in green; the length is 126 bytes which is 4 bytes larger than the IPv4 packets; this is a result of exp-null being imposed. This is shown in yellow.

```

R2#show monitor capture CAP buffer detail
  1 126 0.145988 00:50:56:A9:BE:8A -> 00:50:56:A9:86:2A MPLS unicast
0000: 005056A9 862A0050 56A9BE8A 81000DC4 .PV...*.PV.....
0010: 884716F3 503E0000 213E6000 0000003C .G..P>...!>`....<
0020: 3A3E0000 00000000 00000010 00090009 :>.....
0030: 0009FF33 00000000 00000000 00000000 ...3.....

```

CSR6 receives the packet from the PMSI and forwards it on to CSR5 as raw IPv6 multicast. The counters are increasing.

```

R6#show ipv6 mroute vrf OSPF ff33::9 ::10:9:9:9 count | begin ^Group
Group: FF33::9
  Source: ::10:9:9:9,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 1071/1/126/0, Other: 0/0/0

```

Totals - Source count: 1, Packet count: 1071

CSR5 forwards them into the default MDT within AS 13, which reaches all routers in the MVPN instance. Only CSR8 needs it; both routers show increased packet counts.

```
R5#show ipv6 mroute vrf OSPF ff33::9 ::10:9:9:9 count | begin ^Group
Group: FF33::9
Source: ::10:9:9:9,
SW Forwarding: 0/0/0/0, Other: 0/0/0
HW Forwarding: 1083/1/122/0, Other: 0/0/0
Totals - Source count: 1, Packet count: 1083
```

```
R8#show ipv6 mroute vrf OSPF ff33::9 ::10:9:9:9 count | begin ^Group
Group: FF33::9
Source: ::10:9:9:9,
SW Forwarding: 0/0/0/0, Other: 0/0/0
HW Forwarding: 1125/1/142/1, Other: 0/0/0
Totals - Source count: 1, Packet count: 1125
```

To see the encapsulation on CSR8, we use EPC inbound from CSR5. This shows IPv6 multicast tunneled inside IPv4 multicast (default MDT). The GRE encapsulation is shown in yellow which specifies the IPv4 source/destination addresses. The IPv6 ethertype and first nibble of the IPv6 packet is shown in green, which shows IPv6 inside IPv4. The packet is 142 bytes long (cyan) which accounts for 14 bytes Ethernet, 4 bytes dot1q, and 24 bytes GRE.

```
R8#show monitor capture CAP buffer detail
 4  142    0.440972  13.0.0.5      -> 225.0.0.2      GRE
0000:  01005E00 00020050 56A9DC63 81000DE6  ..^....PV..c....
0010:  08004500 007C304C 0000FF2F 9CFF0D00  ..E...|0L.../....
0020:  0005E100 00020000 86DD6000 0000003C  .....`.....<
0030:  3A3B0000 00000000 00000010 00090009  :;.....
```

Finally, we check CSR4 who is receiving the packets. This shows that IPv6 multicast can also work across option A.

```
R4#show ipv6 mroute ff33::9 ::10:9:9:9 count | begin ^Group
Group: FF33::9
Source: ::10:9:9:9,
SW Forwarding: 0/0/0/0, Other: 0/0/0
HW Forwarding: 1181/1/118/0, Other: 0/0/0
Totals - Source count: 1, Packet count: 1181
```

8.4.1.4 MPLS TE

TE with option A is not very interesting as the tunnels are confined to their own ASes. This will be a brief test to ensure the feature works with option A. This is somewhat similar to using TE with CSC since you can only move traffic between PE, P, and ASBR routers within the AS. I will build a tunnel in each AS to

demonstrate the feature. First, I build a basic TE tunnel to CSR5 that traverses XRv1 directly, which is a high-cost path the OSPF did not select. I add a bandwidth reservation as well although it is not significant. This uses a simple explicit-path; once configured, we verify the tunnel is up.

```
! XRv2
explicit-path name EP_12_11_5
  index 10 next-address strict ipv4 unicast 13.0.0.11
  index 20 next-address strict ipv4 unicast 13.0.0.5

interface tunnel-te100
  ipv4 unnumbered Loopback0
  logging events all
  signalled-bandwidth 5000
  autoroute announce
  destination 13.0.0.5
  path-option 10 explicit name EP_12_11_5
```

```
RP/0/0/CPU0:XRv2#show mpls traffic-eng tunnels brief
```

TUNNEL NAME	DESTINATION	STATUS	STATE
tunnel-te100	13.0.0.5	up	up

Displayed 1 (of 1) heads, 0 (of 0) midpoints, 0 (of 0) tails
Displayed 1 up, 0 down, 0 recovering, 0 recovered heads

Before continuing, we expect to see some multicast issues as a result of building this tunnel. With auto-route, the path to 13.0.0.5/32 is now via this tunnel which is not PIM enabled. This will break the MDT in use by AS 13.

```
RP/0/0/CPU0:XRv2#show pim rpf 13.0.0.5
Table: IPv4-Unicast-default
* 13.0.0.5/32 [110/3]
  via Null with rpf neighbor 0.0.0.0
```

We can fix this by telling IGP to keep the multicast topology intact by effectively ignoring TE tunnels as RPF interfaces. This repairs RPF and allows the GRE traffic to flow following the ordinary IGP paths. These considerations are not specific to inter-AS VPN service, but are general issues. This handy feature is only supported when the TE tunnel uses “autoroute announce”.

```
! XRv2
router ospf 13
  mpls traffic-eng multicast-intact
```

```
RP/0/0/CPU0:XRv2#show pim rpf 13.0.0.5
Table: IPv4-Multicast-default
* 13.0.0.5/32 [110/3]
  via GigabitEthernet0/0/0/0.582 with rpf neighbor 13.8.12.8
```

We verify the outgoing label, which was allocated by XRv1. This allows XRv2 to tunnel traffic towards XRv1 inside the TE LSP.

```
RP/0/0/CPU0:XRv2#show mpls traffic-eng tunnels 100 detail | include Label
    Outgoing Interface: GigabitEthernet0/0/0/0.521, Outgoing Label: 91017
```

Any traffic that normally relied on 13.0.0.5/32 as a next-hop will now use this tunnel. Specifically, this will affect VRF EIGRP traffic traversing ASes. Traffic to CSR1's loopback will be sent towards CSR5, which means the existing LDP label is no longer used.

```
RP/0/0/CPU0:XRv2#show route vrf EIGRP 10.1.1.1
Routing entry for 10.1.1.1/32
  Known via "bgp 13", distance 200, metric 0
  Tag 24, type internal
  Routing Descriptor Blocks
    13.0.0.5, from 13.0.0.5
      Nexthop in Vrf: "default", Table: "default", IPv4 Unicast, Table Id:
0xe0000000
      Route metric is 0
      No advertising protos.
```

```
RP/0/0/CPU0:XRv2#show route 13.0.0.5
Routing entry for 13.0.0.5/32
  Known via "ospf 13", distance 110, metric 3, type intra area
  Routing Descriptor Blocks
    13.0.0.5, from 13.0.0.5, via tunnel-te100
      Route metric is 3
      No advertising protos.
```

Using traceroute inside VRF EIGRP, we can see that the TE LSP is being used for transport within AS 13.

```
R3#traceroute 10.1.1.1 source 10.3.3.3
Type escape sequence to abort.
Tracing the route to 10.1.1.1
VRF info: (vrf in name/id, vrf out name/id)
 1 10.3.12.12 3 msec 1 msec 2 msec
 2 13.11.12.11 [MPLS: Labels 91017/5006 Exp 0] 6 msec 5 msec 5 msec
 3 10.5.6.5 [MPLS: Label 5006 Exp 0] 4 msec 5 msec 5 msec
 4 10.5.6.6 10 msec 7 msec 7 msec
 5 24.6.14.14 [MPLS: Labels 94009/2012 Exp 0] 10 msec 11 msec 15 msec
 6 10.1.2.2 [MPLS: Label 2012 Exp 0] 15 msec 15 msec 15 msec
 7 10.1.2.1 15 msec 9 msec 9 msec
```

Since CSR5 is preferring CSR6 due to it advertising the oldest eBGP route (verified earlier), we also build a tunnel from CSR6 to CSR2. This tunnel will route via CSR7 and CSR2, also using the high cost path not chosen by IS-IS. We verify that the tunnel comes up correctly.

```
! CSR6
ip explicit-path name EP_6_7_2 enable
  next-address 24.0.0.7
  next-address 24.0.0.2

interface Tunnel100
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 24.0.0.2
  tunnel mpls traffic-eng autoroute announce
  tunnel mpls traffic-eng path-option 10 explicit name EP_6_7_2

R6#show mpls traffic-eng tunnels tunnel 100 brief | begin TUNNEL
TUNNEL NAME                DESTINATION        UP IF      DOWN IF
STATE/PROT
R6_t100                    24.0.0.2          -          Gi2.567    up/up
```

Following the route recursion, BGP points to 24.0.0.2 as the next-hop. The RIB says this is reachable via a TE tunnel, so the RSVP label is used. The label stack becomes {7064 2012}.

```
R6#show bgp vpnv4 unicast vrf EIGRP 10.1.1.1/32
BGP routing table entry for 24:3:10.1.1.1/32, version 150
Paths: (1 available, best #1, table EIGRP)
  Advertised to update-groups:
    2
  Refresh Epoch 2
  Local
    24.0.0.2 (metric 20) (via default) from 24.0.0.2 (24.0.0.2)
      Origin incomplete, metric 10880, localpref 100, valid, internal, best
      Extended Community: RT:24:3 Cost:pre-bestpath:128:10880 0x8800:32768:0
        0x8801:3:288 0x8802:65281:2560 0x8803:65281:1500 0x8806:0:167837953
      mpls labels in/out nolabel/2012
      rx pathid: 0, tx pathid: 0x0
```

```
R6#show ip route 24.0.0.2
Routing entry for 24.0.0.2/32
  Known via "isis", distance 115, metric 20, type level-2
  Redistributing via isis 24
  Last update from 24.0.0.2 on Tunnel100, 00:01:53 ago
  Routing Descriptor Blocks:
  * 24.0.0.2, from 24.0.0.2, 00:01:53 ago, via Tunnel100
    Route metric is 20, traffic share count is 1
```

```
R6#show ip rsvp reservation detail filter session-type 7 destination 24.0.0.2
| include Label
Label: 7064 (outgoing)
```

Traffic is now tunnel across CSR7 to CSR2 in AS 24. Using TE tunnels in this way, we can influence left-to-right traffic to use high cost paths, as an example. There are no multicast concerns with mLDP in AS 24 as we never configured mLDP to use TE tunnels in the first place. If we had, LDP would need to be configured on those tunnels so mLDP can exchange labels via those paths.

```
R3#traceroute 10.1.1.1 source 10.3.3.3
Type escape sequence to abort.
Tracing the route to 10.1.1.1
VRF info: (vrf in name/id, vrf out name/id)
 1 10.3.12.12 3 msec 1 msec 2 msec
 2 13.11.12.11 [MPLS: Labels 91017/5006 Exp 0] 6 msec 5 msec 5 msec
 3 10.5.6.5 [MPLS: Label 5006 Exp 0] 5 msec 5 msec 5 msec
 4 10.5.6.6 7 msec 8 msec 7 msec
 5 24.6.7.7 [MPLS: Labels 7064/2012 Exp 0] 10 msec 9 msec 11 msec
 6 10.1.2.2 [MPLS: Label 2012 Exp 0] 15 msec 16 msec 15 msec
 7 10.1.2.1 15 msec 9 msec 10 msec
```

8.4.1.5 Confederation variation

Inter-AS MPLS option A also works with BGP confederations. This might occur when one SP acquires or merges with another. In this case, sub-ASes 13 and 24 form into confederation AS 42518. The configuration is very simple on all 8 LSRs. The configurations for XE and XR are identical as well. Only the ASBRs need to enumerate the confederation peers, as the true iBGP peers just need to identify the confederation ASN.

```
! CSR2 and XRv4
router bgp 24
  bgp confederation identifier 42518
```

```
! CSR6 and CSR7
router bgp 24
  bgp confederation identifier 42518
  bgp confederation peers 13
```

```
! CSR8 and XRv2
router bgp 13
  bgp confederation identifier 42518
```

```
! CSR5 and XRv1
router bgp 13
  bgp confederation identifier 42518
  bgp confederation peers 24
```

If we make the changes quickly enough, the BGP peers may not even flap. I quickly check all 4 ASBRs for brevity to ensure the BGP sessions are operational. This will include the iBGP links to the sub-AS RRs and the intraconfederation (inter-subAS) links utilizing option A. All I do here is scan the last column for any number, indicating some prefix exchanges. Of note, to view VRF-aware BGP peers in XR that are not actually running VPNv4, you must use the specific VRF/AFI show command. This is technically more correct than the way XE does it, but takes longer to verify.

```
R5#show bgp vpnv4 unicast all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
10.5.6.6      4      24     40     38     213   0    0 00:18:26      5
10.5.6.6      4      24     93     34     213   0    0 00:18:33      3
10.5.7.7      4      24     44     43     213   0    0 00:18:31      5
10.5.7.7      4      24    100     35     213   0    0 00:18:30      3
13.0.0.12     4      13    1710    1844    213   0    0 04:24:01      9
```

```
R5#show bgp vpnv6 unicast all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
13.0.0.12     4      13    1710    1844    163   0    0 04:24:01      9
FD00:10:5:6::6 4      24     344     348    163   0    0 04:24:13      5
FD00:10:5:6::6 4      24     336     323    163   0    0 04:24:04      3
FD00:10:5:7::7 4      24     327     347    163   0    0 04:24:18      5
FD00:10:5:7::7 4      24     318     321    163   0    0 04:24:16      3
```

```
R6#show bgp vpnv4 unicast all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
10.5.6.5      4      13      38      40     294   0    0 00:18:30      6
10.5.6.5      4      13      34      93     294   0    0 00:18:38      7
10.6.11.11    4      13      15      15     294   0    0 00:07:50      6
10.6.11.11    4      13      16      13     294   0    0 00:07:49      7
24.0.0.2      4      24    1927    1838    294   0    0 04:24:13      8
```

```
R6#show bgp vpnv6 unicast all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
24.0.0.2      4      24    1927    1838    192   0    0 04:24:13      8
FD00:10:5:6::5 4      13     348     344    192   0    0 04:24:18      6
FD00:10:5:6::5 4      13     323     336    192   0    0 04:24:08      7
FD00:10:6:11::11
4      13      15      15    192   0    0 00:07:49      6
FD00:10:6:11::11
4      13      17      13    192   0    0 00:07:50      7
```

```
R7#show bgp vpnv4 unicast all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
10.5.7.5      4      13      43      45     842   0    0 00:18:39      6
10.5.7.5      4      13      35     100     842   0    0 00:18:38      7
24.0.0.2      4      24    1920    1837     842   0    0 04:24:21      8
```

```
R7#show bgp vpnv6 unicast all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
24.0.0.2      4      24    1920    1837    1536   0    0 04:24:21      8
FD00:10:5:7::5 4      13     348     327    1536   0    0 04:24:26      6
FD00:10:5:7::5 4      13     321     319    1536   0    0 04:24:24      7
```

```
RP/0/0/CPU0:XRv1#show bgp vrf all ipv4 unicast summary | utility egrep 'Neigh|10.6'
```

Neighbor	Spk	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	St/PfxRcd
10.6.11.6	0	24	24	26	53	0	0	00:16:48	3

Neighbor	Spk	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	St/PfxRcd
10.6.11.6	0	24	26	25	53	0	0	00:16:50	5

```
RP/0/0/CPU0:XRv1#show bgp vrf all ipv6 unicast summary | utility egrep 'Neigh|fd00'
```

Neighbor	Spk	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	St/PfxRcd
fd00:10:6:11::6	0	24	25	28	53	0	0	00:17:22	3

Neighbor	Spk	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	St/PfxRcd
fd00:10:6:11::6	0	24	26	26	53	0	0	00:17:20	5

Since only one router, CSR10, was running BGP as the PE-CE protocol, only that one router needs its BGP configuration updated. The remote-AS is now 42518 which represents the entire confederation. Checking CSR2, we can see that it's PE-CE eBGP peer is up, as well as the iBGP peer to the RR, XRv2.

```
! CSR10
router bgp 100
 neighbor 10.8.10.8 remote-as 42518
 neighbor FD00:10:8:10::8 remote-as 42518
```

```
R8#show bgp vpnv4 unicast all summary | begin ^Neigh
```

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
10.8.10.10	4	100	23	27	205	0	0	00:17:01	4
13.0.0.12	4	13	1755	1796	205	0	0	04:35:15	2

```
R8#show bgp vpnv6 unicast all summary | begin ^Neigh
```

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
13.0.0.12	4	13	1755	1797	260	0	0	04:35:18	2
FD00:10:8:10::10	4	100	23	27	260	0	0	00:16:58	4

Checking the neighbor details for the IPv4 session inside VRF OSPF between CSR6 and XRv1, we see some additional information. The output says the neighbors are “under common administration”, which is displayed only when the external peer is a confed-external peer. The output is consistent across both XE and XR platforms.

```
RP/0/0/CPU0:XRv1#show bgp vrf OSPF ipv4 unicast neighbors 10.6.11.6
BGP neighbor is 10.6.11.6, vrf OSPF
 Remote AS 24, local AS 13, external link
 Remote router ID 24.0.0.6
 Neighbor under common administration
 BGP state = Established, up for 00:26:57
[snip]
```

```
R6#show bgp vpnv4 unicast vrf OSPF neighbors 10.6.11.11
```



```
BGP neighbor is 10.6.11.11, vrf OSPF, remote AS 13, external link
BGP version 4, remote router ID 13.0.0.11
Neighbor under common administration
BGP state = Established, up for 00:27:15
[snip]
```

Another benefit of using confederations with XR is that the mandatory eBGP RPL filters are no longer required. Since it is assumed that these peers are under a “common administrator”, such filtering may not be appropriate. XR also automatically sends communities to confed-external peers, following the same logic. I quickly clean up the configuration on XRv1 as a result. The filters were harmless since they passed everything, and the sending of communities was also implied, but removing these commands from the configuration is preferred as they add no value.

```
! XRv1
router bgp 13
vrf OSPF
neighbor 10.6.11.6
address-family ipv4 unicast
no route-policy RPL_PASS in
no route-policy RPL_PASS out
no send-extended-community-ebgp

neighbor fd00:10:6:11::6
address-family ipv6 unicast
no route-policy RPL_PASS in
no route-policy RPL_PASS out
no send-extended-community-ebgp

vrf EIGRP
neighbor 10.6.11.6
address-family ipv4 unicast
no route-policy RPL_PASS in
no route-policy RPL_PASS out
no send-extended-community-ebgp

neighbor fd00:10:6:11::6
address-family ipv6 unicast
no route-policy RPL_PASS in
no route-policy RPL_PASS out
no send-extended-community-ebgp
```

Checking a VPN route on CSR6, we can see that it is successfully receiving routes, along with extended communities, from XRv1 and CSR5. This proves that the RPL/community configurations removed above were indeed unnecessary. The configurations below also reveal a problem. Confed-external peers behave similar to iBGP peers with respect to next-hop adjustments. That is to say, routes exchanged intraconfederation (inter-subAS) have their next-hops left intact. Most of the time, sub-ASes will run their own IGPs, so this default behavior is often undesirable. The exception to this would be if both ASes

were in a common IGP domain and confederations were only used to minimize iBGP full mesh requirements in lieu of route-reflection.

```
R6#show bgp vpnv4 unicast vrf OSPF 110.0.0.2/32
BGP routing table entry for 24:2:110.0.0.2/32, version 293
Paths: (2 available, no best path)
  Not advertised to any peer
  Refresh Epoch 1
  (13) 100
    13.0.0.8 (inaccessible) (via vrf OSPF) from 10.6.11.11 (13.0.0.11)
      Origin incomplete, metric 0, localpref 100, valid, confed-external
      Extended Community: RT:24:2
      mpls labels in/out 6040/nolabel
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  (13) 100
    13.0.0.8 (inaccessible) (via vrf OSPF) from 10.5.6.5 (13.0.0.5)
      Origin incomplete, metric 0, localpref 100, valid, confed-external
      Extended Community: RT:24:2
      mpls labels in/out 6040/nolabel
      rx pathid: 0, tx pathid: 0
```

Within the context of VPNv4, we cannot advertise addresses like 13.0.0.8 into the VPN since these are global routes used for VPN next-hop recursion. Next-hop-self is the best option, and I configure it on all ASBRs. For brevity, I show the configuration on CSR5 and XRv1.

```
! XRv1
router bgp 13
  vrf OSPF
    neighbor 10.6.11.6
      address-family ipv4 unicast
      next-hop-self

    neighbor fd00:10:6:11::6
      address-family ipv6 unicast
      next-hop-self

  vrf EIGRP
    neighbor 10.6.11.6
      address-family ipv4 unicast
      next-hop-self

    neighbor fd00:10:6:11::6
      address-family ipv6 unicast
      next-hop-self

! CSR5
router bgp 13
```

```

address-family ipv4 vrf EIGRP
  neighbor 10.5.6.6 next-hop-self
  neighbor 10.5.7.7 next-hop-self

address-family ipv6 vrf EIGRP
  neighbor FD00:10:5:6::6 next-hop-self
  neighbor FD00:10:5:7::7 next-hop-self

address-family ipv4 vrf OSPF
  neighbor 10.5.6.6 next-hop-self
  neighbor 10.5.7.7 next-hop-self

address-family ipv6 vrf OSPF
  neighbor FD00:10:5:6::6 next-hop-self
  neighbor FD00:10:5:7::7 next-hop-self

```

At this point, the routes learned inside the transit link VRFs are now showing valid next-hop values. The network should be logically identical to option A with eBGP now. The only exception is that the AS paths now contain the sub-ASes in parenthesis to indicate that they are part of the same confederation.

```

R6#show bgp vpnv4 unicast vrf OSPF 10.4.4.4/32
BGP routing table entry for 24:2:10.4.4.4/32, version 311
Paths: (2 available, best #2, table OSPF)
  Advertised to update-groups:
    6          3
  Refresh Epoch 1
  (13)
    10.6.11.11 (via vrf OSPF) from 10.6.11.11 (13.0.0.11)
      Origin incomplete, metric 1, localpref 100, valid, confed-external
      Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
        OSPF RT:0.0.0.0:2:0
      mpls labels in/out 6036/nolabel
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 2
  (13)
    10.5.6.5 (via vrf OSPF) from 10.5.6.5 (13.0.0.5)
      Origin incomplete, metric 1, localpref 100, valid, confed-external,
best
      Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
        OSPF RT:0.0.0.0:2:0
      mpls labels in/out 6036/nolabel
      rx pathid: 0, tx pathid: 0x0

```

The existing sham-link automatically forms once there is VPN connectivity between the PEs participating in the sham-link. This is a good indication that MPLS forwarding will work for the customer traffic as well.

```
R8#show ospfv3 vrf OSPF sham-links | include ^Sham
```

```
Sham Link OSPFv3_SL0 to address FD00::2 is up
Sham Link OSPFv3_SL1 to address FD00::2 is up
```

Tracing the path from CSR4 to CSR9, we see that CSR4 has both the PE-CE and backdoor links up. The route to 10.9.9.9/32 is an intra-area route via the MPLS network, proving that the sham-link is working.

```
R4#show ospfv3 neighbor
```

```
OSPFv3 2 address-family ipv4 (router-id 10.4.8.4)
```

Neighbor ID	Pri	State	Dead Time	Interface ID	Interface
10.4.9.9	0	FULL/ -	00:00:33	26	Gig2.549
10.4.8.8	0	FULL/ -	00:00:39	12	Gig2.548

```
OSPFv3 2 address-family ipv6 (router-id 10.4.8.4)
```

Neighbor ID	Pri	State	Dead Time	Interface ID	Interface
10.4.9.9	0	FULL/ -	00:00:34	26	Gig2.549
10.4.8.8	0	FULL/ -	00:00:35	12	Gig2.548

```
R4#show ip route 10.9.9.9
```

```
Routing entry for 10.9.9.9/32
```

```
Known via "ospfv3 2", distance 110, metric 3, type intra area
```

```
Last update from 10.4.8.8 on GigabitEthernet2.548, 10:15:30 ago
```

```
Routing Descriptor Blocks:
```

```
* 10.4.8.8, from 10.4.9.9, 10:15:30 ago, via GigabitEthernet2.548
```

```
Route metric is 3, traffic share count is 1
```

CSR8 learns the VPN route to 10.9.9.9/32 via CSR5 (reflected best-path by XRv2) with a label of 5007. The AS path contains sub-AS 24 because the route was originated inside that sub-AS.

```
R8#show bgp vpnv4 unicast vrf OSPF 10.9.9.9/32
```

```
BGP routing table entry for 13:2:10.9.9.9/32, version 236
```

```
Paths: (1 available, best #1, table OSPF)
```

```
Not advertised to any peer
```

```
Refresh Epoch 1
```

```
(24)
```

```
13.0.0.5 (metric 2) (via default) from 13.0.0.12 (13.0.0.12)
```

```
Origin incomplete, metric 1, localpref 100, valid, confed-internal,
best
```

```
Extended Community: RT:13:2 OSPF ROUTER ID:10.2.9.2:0
```

```
OSPF RT:0.0.0.0:2:0
```

```
Originator: 13.0.0.5, Cluster list: 13.0.0.12
```

```
Connector Attribute: count=1
```

```
type 1 len 12 value 13:2:13.0.0.5
```

```
mpls labels in/out nlabel/5007
```

```
rx pathid: 0, tx pathid: 0x0
```

Only the VPN label of 5007 is added to the label stack since CSR8 and CSR5 are connected. The route to 13.0.0.5/32 is an IGP route, so the LDP label of implicit-null is used.

```
R8#show ip route 13.0.0.5
Routing entry for 13.0.0.5/32
  Known via "ospf 13", distance 110, metric 2, type intra area
  Last update from 13.5.8.5 on GigabitEthernet2.558, 15:14:49 ago
  Routing Descriptor Blocks:
  * 13.5.8.5, from 13.0.0.5, 15:14:49 ago, via GigabitEthernet2.558
    Route metric is 2, traffic share count is 1
```

```
R8#show mpls ldp bindings 13.0.0.5 32 neighbor 13.0.0.5
lib entry: 13.0.0.5/32, rev 8
remote binding: lsr: 13.0.0.5:0, label: imp-null
```

Packets arriving with label 5007 have their label stack removed and raw IP traffic forwarded to 10.5.6.6 (CSR6) inside the OSPF VPN.

```
R5#show mpls forwarding-table labels 5007 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
5007       No Label  10.9.9.9/32[V]  600          Gi2.5562   10.5.6.6
MAC/Encaps=22/22, MRU=1504, Label Stack{}
005056A9DE0D005056A9DC6381000DE4810000020800
VPN route: OSPF
No output feature configured
```

Checking CSR5's VPN route, we can see that it is marked as confed-external as opposed to confed-internal. This isn't terribly significant, but the transit links are the only places in the network where confed-external peers exist.

```
R5#show bgp vpnv4 unicast vrf OSPF 10.9.9.9/32
BGP routing table entry for 13:2:10.9.9.9/32, version 9
Paths: (2 available, best #2, table OSPF)
  Advertised to update-groups:
    12      13
  Refresh Epoch 1
  (24)
    10.5.7.7 (via vrf OSPF) from 10.5.7.7 (24.0.0.7)
      Origin incomplete, metric 1, localpref 100, valid, confed-external
      Extended Community: RT:13:2 OSPF ROUTER ID:10.2.9.2:0
      OSPF RT:0.0.0.0:2:0
      mpls labels in/out 5007/nolabel
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  (24)
    10.5.6.6 (via vrf OSPF) from 10.5.6.6 (24.0.0.6)
```

```

Origin incomplete, metric 1, localpref 100, valid, confed-external,
best
Extended Community: RT:13:2 OSPF ROUTER ID:10.2.9.2:0
OSPF RT:0.0.0.0:2:0
mpls labels in/out 5007/nolabel
rx pathid: 0, tx pathid: 0x0

```

CSR6 adds two labels to the incoming traffic from CSR5. 2015 is the VPN label allocated by CSR2 and 94009 is XRv4's label towards 24.0.0.2/32, bound by LDP. XRv4 is a P router that performs PHP to expose label 2015 to CSR2. CSR2 removes all labels and delivers the packet to CSR9 inside the OSPF VPN.

```

R6#show ip cef vrf OSPF 10.9.9.9/32
10.9.9.9/32
  nexthop 24.6.14.14 GigabitEthernet2.564 label 94009 2015

```

```

RP/0/0/CPU0:XRv4#show mpls forwarding labels 94009

```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94009	Pop	24.0.0.2/32	Gi0/0/0/0.524	24.2.14.2	2114005

```

R2#show mpls forwarding-table labels 2015 detail

```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
2015	No Label	10.9.9.9/32[V]	20706	Gi2.529	10.2.9.9

```

MAC/Encaps=18/18, MRU=1504, Label Stack{
005056A9D672005056A9BE8A81000DC90800
VPN route: OSPF
No output feature configured

```

Like option A, the transit traffic is untagged between ASes. Otherwise, the behavior is identical to option A.

```

R4#traceroute 10.9.9.9 source 10.4.4.4
Type escape sequence to abort.
Tracing the route to 10.9.9.9
VRF info: (vrf in name/id, vrf out name/id)
 1 10.4.8.8 6 msec 5 msec 2 msec
 2 10.5.6.5 [MPLS: Label 5007 Exp 0] 4 msec 4 msec 5 msec
 3 10.5.6.6 7 msec 11 msec 9 msec
 4 24.6.14.14 [MPLS: Labels 94009/2015 Exp 0] 13 msec 32 msec 23 msec
 5 10.2.9.2 [MPLS: Label 2015 Exp 0] 16 msec 20 msec 19 msec
 6 10.2.9.9 22 msec 11 msec 9 msec

```

I quickly spot-check some key nodes along the IPv6 central services LSP from XRv3 to CSR10. XRv3 sees this as an external route via XRv4 since it came from a non-EIGRP domain across the VPN.

```
RP/0/0/CPU0:XRv3#show route ipv6 ::110:0:0:2
Routing entry for ::110:0:0:2/128
  Known via "eigrp 3", distance 170, metric 107520
  Tag 13, type external
  Routing Descriptor Blocks
    fe80::14, from fe80::14, via GigabitEthernet0/0/0/0.534
      Route metric is 107520
  No advertising protos.
```

This route shows AS 100 as the originator, and since it is not in parenthesis, we assume it is a true eBGP peer connected via sub-AS 13.

```
RP/0/0/CPU0:XRv4#show bgp vpnv6 unicast vrf EIGRP ::110:0:0:2/128 | begin 13
(13) 100
  24.0.0.6 (metric 10) from 24.0.0.2 (24.0.0.6)
    Received Label 6031
    Origin incomplete, metric 0, localpref 100, valid, confed-internal,
best, group-best, import-candidate, imported
    Received Path ID 0, Local Path ID 1, version 7420
    Extended community: RT:24:3
    Originator: 24.0.0.6, Cluster list: 24.0.0.2
    Source VRF: EIGRP, Source Route Distinguisher: 24:3
```

On the other side of the network, CSR10 shows only AS 42518 in the path. The entire confederation AS path list collapses into the confederation ID once the route is advertised to a true eBGP peer. From CSR8's perspective, the ASN of 100 is a true external AS and the prefixes from that AS are labeled as such. This is not specific to inter-AS MPLS but is the general behavior of BGP confederations.

```
R10#show bgp ipv6 unicast ::10:13:13:13/128
BGP routing table entry for ::10:13:13:13/128, version 5328
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  42518
    FD00:10:8:10::8 (FE80::8) from FD00:10:8:10::8 (13.0.0.8)
      Origin incomplete, localpref 100, valid, external, best
      rx pathid: 0, tx pathid: 0x0
```

```
R8#show bgp vpnv6 unicast vrf BGP ::110:0:0:2/128
BGP routing table entry for [13:1]::110:0:0:2/128, version 497
Paths: (1 available, best #1, table BGP)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  100
    FD00:10:8:10::10 (FE80::10) (via vrf BGP) from FD00:10:8:10::10
(110.0.0.0)
      Origin incomplete, metric 0, localpref 100, valid, external, best
```

```
Extended Community: RT:13:1
mpls labels in/out 8012/nolabel
rx pathid: 0, tx pathid: 0x0
```

Without tracing the LSP manually, we can see no issue with connectivity. Traffic is unlabeled on the transit links as expected.

```
RP/0/0/CPU0:XRv3#traceroute ::110:0:0:2 source ::10:13:13:13
Type escape sequence to abort.
Tracing the route to ::110:0:0:2
 1  fd00:10:13:14::14 0 msec 0 msec 0 msec
 2  fd00:10:5:6::6 [MPLS: Label 6031 Exp 0] 0 msec 0 msec 0 msec
 3  fd00:10:5:6::5 0 msec 0 msec 0 msec
 4  fd00:10:8:10::8 [MPLS: Label 8012 Exp 0] 0 msec 0 msec 0 msec
 5  fd00:10:8:10::10 9 msec 0 msec 0 msec
```

As a quick MVPN test, we can see that the BSR information inside VRF EIGRP has traversed the network. If any of the default MDTs were broken in either AS, this would not be possible, so this cursory check implies the default MDTs are still operational from the option A configuration. Once the intraconfederation BGP next-hops are set to “self” on the ASBRs, everything “just works” when migrating from eBGP to confederations.

```
RP/0/0/CPU0:XRv3#show pim rp mapping
PIM Group-to-RP Mappings
Group(s) 224.0.0.0/4
  RP 10.3.3.3 (?), v2
    Info source: 10.1.13.1 (?), elected via bsr, priority 0, holdtime 150
    Uptime: 10:50:28, expires: 00:02:23
```

Sending traffic from CSR3, we confirm that the network is still capable of transporting MVPN flows. XRv3 is receiving this traffic as a result of having joined 225.13.13.13 as the counters indicate.

```
R3#ping ip
Target IP address: 225.13.13.13
Repeat count [1]: 10000
Datagram size [100]:
Timeout in seconds [2]: 1
Extended commands [n]: y
Interface [All]: loopback0
Time to live [255]:
Source address or interface: loopback0
[snip]
```

```
RP/0/0/CPU0:XRv3#show mfib route 225.13.13.13 10.3.3.3 | begin 225
(10.3.3.3,225.13.13.13),    Flags:
  Up: 00:00:45
  Last Used: 00:00:00
```



```
SW Forwarding Counts: 45/45/4500
SW Replication Counts: 45/0/0
SW Failure Counts: 0/0/0/0/0
Loopback0 Flags: IC NS EG, Up:00:00:45
GigabitEthernet0/0/0/0.513 Flags: A, Up:00:00:45
```

Additional Reading – Reference configurations “inter-as-mpls-a-confed”

8.4.1.6 Carrier Supporting Carrier (CSC) variation

This variation uses the original option A lab (no confederations) to demonstrate how to have an end-to-end MPLS forwarding path. Like option B, the LSPs change frequently along the path as the VPN label is swapped, but this can be used for CSC support. For example, if VRF OSPF and VRF EIGRP were renamed to CUST_CARRIER1 and CUST_CARRIER2, VPN routes could be exchanged between two core carriers that team up to provide CSC services. Since options B and C already support MPLS paths end-to-end, this is not a consideration for them as CSC support is inherent in those designs. Option AB, discussed later, has a specific CSC feature as well. Using basic IPv4 labeled-unicast can be used to enhance option A, making it a viable solution for inter-AS MPLS encapsulation between two core carriers. The configuration changes are very simple and are shown below. All we must do is enable labeled-unicast where CSC is required. In this example, VRF OSPF requires CSC (MPLS on transit link) while VRF EIGRP does not (IP on transit link, classic design). As such, there is no reason to exchange BGP labels inside the EIGRP VPN. This assumes that the configuration baseline begins with the standard eBGP option A design.

```
! XRv1
router bgp 13
  vrf OSPF
    address-family ipv4 unicast
      allocate-label all

  neighbor 10.6.11.6
    no address-family ipv4 unicast
    address-family ipv4 labeled-unicast
    route-policy RPL_PASS in
    route-policy RPL_PASS out
    send-extended-community-ebgp

! CSR6
router bgp 24
  address-family ipv4 vrf OSPF
    neighbor 10.5.6.5 send-label
    neighbor 10.6.11.11 send-label

! CSR5
router bgp 13
  address-family ipv4 vrf OSPF
    neighbor 10.5.6.6 send-label
    neighbor 10.5.7.7 send-label
```

```
! CSR7
router bgp 24
  address-family ipv4 vrf OSPF
    neighbor 10.5.7.5 send-label
```

Once the BGP sessions comes up, we see the usual syslog message about “mpls bgp forwarding” being configured on the transit links on XE routers. This is a good sign that BGP IPv4 labeled-unicast has been successfully negotiated.

```
! CSR5 and CSR6
%BGP_LMM-6-AUTOGEN1: The mpls bgp forwarding command has been configured on
interface: GigabitEthernet2.5562
```

We quickly verify that labeled-unicast was negotiated with all peers by checking CSR5 and CSR6 only. Since they peer with all remote ASBRs, this is faster than checking all 4 routers. Technically, XE calls this VPNv4 labeled-unicast since the labels are exchanged inside of a VPN, but the configuration is still similar to IPv4 labeled-unicast on all platforms; this is not the same as VPNv4 unicast. The capability is advertised and received on all routers, which implies bidirectionally capability negotiation.

```
R5#show bgp vpnv4 unicast vrf OSPF neighbors 10.5.6.6 | include
^BGP|vpnv4_MPLS
BGP neighbor is 10.5.6.6, vrf OSPF, remote AS 24, external link
vpnv4 MPLS Label capability: advertised and received
```

```
R5#show bgp vpnv4 unicast vrf OSPF neighbors 10.5.7.7 | include
^BGP|vpnv4_MPLS
BGP neighbor is 10.5.7.7, vrf OSPF, remote AS 24, external link
vpnv4 MPLS Label capability: advertised and received
```

```
R6#show bgp vpnv4 unicast vrf OSPF neighbors 10.5.6.5 | include
^BGP|vpnv4_MPLS
BGP neighbor is 10.5.6.5, vrf OSPF, remote AS 13, external link
vpnv4 MPLS Label capability: advertised and received
```

```
R6#show bgp vpnv4 unicast vrf OSPF neighbors 10.6.11.11 | include
^BGP|vpnv4_MPLS
BGP neighbor is 10.6.11.11, vrf OSPF, remote AS 13, external link
vpnv4 MPLS Label capability: advertised and received
```

Although this new CSC configuration has nothing to do with the OSPF sham-links, we verify that they are up. This will allow traffic between CSR4 and CSR9 to prefer to MPLS network over the low-speed backdoor link.

```
R2#show ospfv3 vrf OSPF sham-links | include ^Sham
Sham Link OSPFv3_SL0 to address FD00::8 is up
```

Sham Link OSPFv3_SL1 to address FD00::8 is up

We will trace the LSP from CSR9 to CSR4 using IPv4. The focus is on the ASBRs so the intra-AS trace will be fast. CSR2's best VPN route is via CSR6 using label 6025 with a transport label of 94008.

```
R2#show bgp vpnv4 unicast vrf OSPF 10.4.4.4/32 bestpath
BGP routing table entry for 24:2:10.4.4.4/32, version 542
Paths: (2 available, best #2, table OSPF)
  Advertised to update-groups:
    5
  Refresh Epoch 1
  13, (Received from a RR-client)
    24.0.0.6 (metric 20) (via default) from 24.0.0.6 (24.0.0.6)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
        OSPF RT:0.0.0.0:2:0
        mpls labels in/out nolaabel/6205
        rx pathid: 0, tx pathid: 0x0
```

```
R2#show ip cef 24.0.0.6
24.0.0.6/32
  nexthop 24.2.14.14 GigabitEthernet2.524 label 94008
```

XRv4 removes label 94008 to expose label 6025 to CSR6. CSR6 swaps this for label 5054, which was received from CSR5 for prefix 10.4.4.4/32 inside the OSPF VPN, and forwards traffic towards CSR5 in that same VPN. MPLS traffic essentially moves from the global table to a customer VPN, just like CSC. Although the LFIB gives us all the details we require for a trace, I show the BGP route for comparison. This is the primary difference between traditional option A and CSC-supported option A. Normally label 6205 would be removed and raw IPv4 traffic would be forwarded towards 10.5.6.5.

```
RP/0/0/CPU0:XRv4#show mpls forwarding labels 94008
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94008	Pop	24.0.0.6/32	Gi0/0/0/0.564	24.6.14.6	50292

```
R6#show mpls forwarding-table labels 6205 detail
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
6205	5054	10.4.4.4/32 [V]	0	Gi2.5562	10.5.6.5

```
MAC/Encaps=22/26, MRU=1500, Label Stack{5054}
005056A9DC63005056A9DE0D81000DE4810000028847 013BE000
VPN route: OSPF
No output feature configured
```

```
R6#show bgp vpnv4 unicast vrf OSPF 10.4.4.4/32 bestpath
BGP routing table entry for 24:2:10.4.4.4/32, version 277
```

```

Paths: (2 available, best #2, table OSPF)
  Advertised to update-groups:
    8          7
  Refresh Epoch 1
  13
    10.5.6.5 (via vrf OSPF) from 10.5.6.5 (13.0.0.5)
      Origin incomplete, localpref 100, valid, external, best
      Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
      OSPF RT:0.0.0.0:2:0
      mpls labels in/out 6205/5054
      rx pathid: 0, tx pathid: 0x0

```

CSR5 swaps label 5054 for label 8015, which is CSR8's original VPN label for 10.4.4.4/32. No additional transport label is pushed because CSR5 and CSR8 are directly connected, so the LDP label bound to 13.0.0.8/32 is implicit-null.

```

R5#show mpls forwarding-table labels 5054
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id Switched     interface
5054       8015       10.4.4.4/32[V]  0           Gi2.558    13.5.8.8

```

```

R5#show bgp vpnv4 unicast vrf OSPF 10.4.4.4/32
BGP routing table entry for 13:2:10.4.4.4/32, version 253
Paths: (1 available, best #1, table OSPF)
  Advertised to update-groups:
    6
  Refresh Epoch 1
  Local
    13.0.0.8 (metric 2) (via default) from 13.0.0.12 (13.0.0.12)
      Origin incomplete, metric 1, localpref 100, valid, internal, best
      Extended Community: RT:13:2 OSPF ROUTER ID:10.4.8.8:0
      OSPF RT:0.0.0.0:2:0
      Originator: 13.0.0.8, Cluster list: 13.0.0.12
      Connector Attribute: count=1
      type 1 len 12 value 13:2:13.0.0.8
      mpls labels in/out 5054/8015
      rx pathid: 0, tx pathid: 0x0

```

```

R5#show ip cef 13.0.0.8
13.0.0.8/32
  nexthop 13.5.8.8 GigabitEthernet2.558

```

Using traceroute within the VPN shows this connectivity. Although we did not manually trace the reverse LSP, we can see that it is also MPLS-encapsulated for its entire journey. This would allow CSR4 and CSR9 to send MPLS packets into AS 13 or 24 (assuming those PE-CE links were configured to support it).

```

R9#traceroute 10.4.4.4 source 10.9.9.9

```

```
Type escape sequence to abort.
Tracing the route to 10.4.4.4
VRF info: (vrf in name/id, vrf out name/id)
 1 10.2.9.2 5 msec 4 msec 4 msec
 2 24.2.14.14 [MPLS: Labels 94008/6205 Exp 0] 6 msec 6 msec 7 msec
 3 10.5.6.6 [MPLS: Label 6205 Exp 0] 29 msec 32 msec 37 msec
 4 10.5.6.5 [MPLS: Label 5054 Exp 0] 23 msec 32 msec 32 msec
 5 10.4.8.8 [MPLS: Label 8015 Exp 0] 18 msec 21 msec 20 msec
 6 10.4.8.4 76 msec 8 msec 8 msec
```

```
R4#traceroute 10.9.9.9 source 10.4.4.4
Type escape sequence to abort.
Tracing the route to 10.9.9.9
VRF info: (vrf in name/id, vrf out name/id)
 1 10.4.8.8 6 msec 3 msec 3 msec
 2 10.5.6.5 [MPLS: Label 5053 Exp 0] 8 msec 10 msec 10 msec
 3 10.5.6.6 [MPLS: Label 6211 Exp 0] 20 msec 30 msec 32 msec
 4 24.6.7.7 [MPLS: Labels 7010/2025 Exp 0] 29 msec 31 msec 31 msec
 5 10.2.9.2 [MPLS: Label 2025 Exp 0] 15 msec 16 msec 16 msec
 6 10.2.9.9 19 msec 11 msec 10 msec
```

Of note, since the central services routes were merged with the OSPF VPN routes inside AS 13, CSC is supported towards CSR10 as well. This may not be particularly useful, especially if CSR10 actually is a “central services” router, but certain CSC architectures may require something similar.

```
R9#traceroute 110.0.0.1 source 10.9.9.9
Type escape sequence to abort.
Tracing the route to 110.0.0.1
VRF info: (vrf in name/id, vrf out name/id)
 1 10.2.9.2 4 msec 4 msec 4 msec
 2 24.2.14.14 [MPLS: Labels 94008/6208 Exp 0] 7 msec 7 msec 7 msec
 3 10.5.6.6 [MPLS: Label 6208 Exp 0] 29 msec 30 msec 32 msec
 4 10.5.6.5 [MPLS: Label 5057 Exp 0] 30 msec 31 msec 31 msec
 5 10.8.10.8 [MPLS: Label 8040 Exp 0] 19 msec 21 msec 21 msec
 6 10.8.10.10 19 msec 12 msec 11 msec
```

Quickly using traceroute inside the EIGRP VPN, we can see the traffic on the transit links is still raw IP. This is by design as BGP labels were not exchanged within the EIGRP VPN. This allows option A to remain flexible on a per-customer basis, which can preserve labels on the ASBRs as CSC can be selectively enabled.

```
RP/0/0/CPU0:XRv3#traceroute 10.3.3.3 source 10.13.13.13
Type escape sequence to abort.
Tracing the route to 10.3.3.3
 1 10.13.14.14 9 msec 0 msec 0 msec
 2 10.5.6.6 [MPLS: Label 6203 Exp 0] 0 msec 0 msec 0 msec
 3 10.5.6.5 0 msec 0 msec 0 msec
```

```

4 13.5.8.8 [MPLS: Labels 8000/92004 Exp 0] 0 msec 0 msec 0 msec
5 13.8.12.12 [MPLS: Label 92004 Exp 0] 0 msec 0 msec 0 msec
6 10.3.12.3 0 msec 0 msec 0 msec

```

```
R3#traceroute 10.1.1.1 source 10.3.3.3
```

```
Type escape sequence to abort.
```

```
Tracing the route to 10.1.1.1
```

```
VRF info: (vrf in name/id, vrf out name/id)
```

```

1 10.3.12.12 3 msec 2 msec 2 msec
2 13.11.12.11 [MPLS: Labels 91008/5036 Exp 0] 9 msec 6 msec 5 msec
3 10.5.6.5 [MPLS: Label 5036 Exp 0] 16 msec 16 msec 15 msec
4 10.5.6.6 20 msec 10 msec 11 msec
5 24.6.7.7 [MPLS: Labels 7010/2026 Exp 0] 28 msec 21 msec 19 msec
6 10.1.2.2 [MPLS: Label 2026 Exp 0] 19 msec 19 msec 19 msec
7 10.1.2.1 22 msec 10 msec 9 msec

```

Regarding IPv6, it will remain raw IPv6 across both VPNs at this point. We did not explicitly configure IPv6 labeled-unicast inside of the OSPF VPN. Using traceroute inside the OSPF VPN proves this.

```
R4#traceroute ipv6
```

```
Target IPv6 address: ::10:9:9:9
```

```
Source address: ::10:4:4:4
```

```
[snip]
```

```

1 FD00:10:4:8::8 5 msec 4 msec 1 msec
2 FD00:10:5:6::5 [MPLS: Label 5049 Exp 0] 4 msec 5 msec 4 msec
3 FD00:10:5:6::6 23 msec 13 msec 16 msec
4 ::FFFF:24.6.7.7 [MPLS: Labels 7010/2032 Exp 0] 15 msec 24 msec 27 msec
5 FD00:10:2:9::2 [MPLS: Label 2032 Exp 0] 22 msec 22 msec 22 msec
6 FD00:10:2:9::9 23 msec 16 msec 13 msec

```

Below are snippets that would probably work if XE supported VPNv6 labeled-unicast exchange. It is not currently supported, so this configuration is for demonstration only. This is generally acceptable since the CSC endpoints would normally be IPv4 loopbacks, and the customer carrier could run 6VPE inside of the existing MPLS tunnel. XR appears to support the feature, though.

```
! XRv1 (future testing)
```

```
router bgp 13
```

```
vrf OSPF
```

```
address-family ipv6 unicast
```

```
allocate-label all
```

```
neighbor fd00:10:6:11::6
```

```
no address-family ipv6 unicast
```

```
address-family ipv6 labeled-unicast
```

```
route-policy RPL_PASS in
```

```
route-policy RPL_PASS out
```

```
send-extended-community-ebgp
```

```

! CSR6 (future testing)
router bgp 24
  address-family ipv6 vrf OSPF
    neighbor fd00:10:5:6::5 send-label
    neighbor fd00:10:6:11::11 send-label

! CSR5 (future testing)
router bgp 13
  address-family ipv6 vrf OSPF
    neighbor fd00:10:5:6::6 send-label
    neighbor fd00:10:5:7::7 send-label

! CSR7 (future testing)
router bgp 24
  address-family ipv6 vrf OSPF
    neighbor fd00:10:5:7::5 send-label

```

Applying these configurations to CSR7, as an example, generate the following error message. I suspect this will be supported in the future as IPv6 becomes common in the carrier IGP domains, along with LDPv6.

```

R7(config-router-af)#neighbor fd00:10:5:7::5 send-label
%BGP-4-BGP_LABELS_NOT_SUPPORTED: BGP neighbor FD00:10:5:7::5 does not support
sending labelsend

```

Additional Reading – Reference configurations “inter-as-mpls-a-csc”

8.4.2 Option B (ASBR VPNv4/v6 eBGP)

Inter-AS Option B uses direct VPNv4/v6 sessions between ASBRs to exchange VPN routes. This means that the VPN sessions are eBGP and exist in the global table; thus, a single global transit link can be used. This greatly enhances the scalability of inter-AS VPNs over Option A as all ASBRs no longer need to define VRFs locally or configure per-customer BGP sessions. It also means that inter-AS traffic can be MPLS-encapsulated implying that technologies like TE, mLDP, and CSC can be extended across AS boundaries. In terms of configuration, the ASBRs still must run VPNv4/v6 for L3VPN, so those intra-AS BGP sessions shown in option A are still needed. I do not show the basic intra-AS VPNv4/v6 configurations again for these AFIs here. The same is true for the L2VPN AFI, except that now the two ASes must agree on the auto-discovery and signaling methods. One of the biggest downsides to option B is that the providers must agree on the exact RT policies per customer as these values are exchanged between ASes. This additional coordination, in real life, can be very burdensome and problematic. Failing to do this would require manual RT rewriting at the ASBRs which scales poorly and is considered a workaround in many cases. The common configuration for option B for all MPLS services is the transit link configuration. These configurations are incomplete in terms of MPLS TE but that is discussed later as appropriate. First we begin with the AS 13 transit links. Two of them connect to CSR6 and one of them connects to CSR7. These are basic interfaces with no new technologies; BSR-border is required so that

RP information does not leak between AS boundaries. This was not a concern in option A since the transit links were in separate VRFs.

```
! XRv1
interface GigabitEthernet0/0/0/0.561
  ipv4 address 10.6.11.11 255.255.255.0
  ipv6 address fe80::11 link-local
  ipv6 address fd00:10:6:11::11/64
  encapsulation dot1q 3561

router pim
  address-family ipv4
    interface GigabitEthernet0/0/0/0.561
      bsr-border

! CSR5
interface GigabitEthernet2.556
  encapsulation dot1Q 3556
  ip address 10.5.6.5 255.255.255.0
  ip pim bsr-border
  ip pim sparse-mode
  ipv6 address FE80::5 link-local
  ipv6 address FD00:10:5:6::5/64
  ip rsvp bandwidth 200000
  no ipv6 pim

interface GigabitEthernet2.557
  encapsulation dot1Q 3557
  ip address 10.5.7.5 255.255.255.0
  ip pim bsr-border
  ip pim sparse-mode
  ipv6 address FE80::5 link-local
  ipv6 address FD00:10:5:7::5/64
  no ipv6 pim
```

Next, the transit links in AS 24 are shown. There is nothing special here either; the configurations are for reference only.

```
! CSR6
interface GigabitEthernet2.556
  encapsulation dot1Q 3556
  ip address 10.5.6.6 255.255.255.0
  ip pim bsr-border
  ip pim sparse-mode
  ipv6 address FE80::6 link-local
  ipv6 address FD00:10:5:6::6/64
  no ipv6 pim
```



```

interface GigabitEthernet2.561
  encapsulation dot1Q 3561
  ip address 10.6.11.6 255.255.255.0
  ip pim bsr-border
  ip pim sparse-mode
  ipv6 address FE80::6 link-local
  ipv6 address FD00:10:6:11::6/64
  no ipv6 pim

! CSR7
interface GigabitEthernet2.557
  encapsulation dot1Q 3557
  ip address 10.5.7.7 255.255.255.0
  ip pim bsr-border
  ip pim sparse-mode
  ipv6 address FE80::7 link-local
  ipv6 address FD00:10:5:7::7/64
  no ipv6 pim

```

To verify that the transit links were configured correctly, we can use the same method we used for option A. Checking the PIM neighbors is a two-fold benefit: it ensures there is IP reachability between ASes and also verifies the PIM neighbors (required for multicast). I limit the verification to CSR5 and CSR6 since they have neighbors with all peer AS routers.

```

R5#show ip pim neighbor | begin ^Neighbor
Neighbor      Interface      Uptime/Expires    Ver    DR
Address
13.5.11.11    GigabitEthernet2.551    19:04:12/00:01:37 v2      1 / DR P G
13.5.8.8      GigabitEthernet2.558    19:04:15/00:01:43 v2      1 / DR S P G
10.5.6.6      GigabitEthernet2.556    00:03:47/00:01:22 v2      1 / DR S P G
10.5.7.7      GigabitEthernet2.557    00:03:38/00:01:32 v2      1 / DR S P G

```

```

R6#show ip pim neighbor | begin ^Neighbor
Neighbor      Interface      Uptime/Expires    Ver    DR
Address
24.6.14.14    GigabitEthernet2.564    1d14h/00:01:25    v2      1 / DR P G
24.6.7.7      GigabitEthernet2.567    1d14h/00:01:32    v2      1 / DR S P G
10.5.6.5      GigabitEthernet2.556    00:06:01/00:01:36 v2      1 / S P G
10.6.11.11    GigabitEthernet2.561    00:00:34/00:01:41 v2      1 / DR P G

```

Additional Reading – Reference configurations “inter-as-mpls-b”

8.4.2.1 L3VPN

With the transit links properly configured, we will verify that the VPNv4/v6 sessions are enabled between the RR and each PE/ASBR. Like option A, we still need to extend VPNv4/v6 to each PE/ASBR as the ASBRs are now exchanging these routes directly with the peer AS. We will verify XRv2 (AS 13) and

CSR2 (AS 24) to ensure the VPNv4/v6 sessions are up. As expected, no VPN routes are received from the ASBRs presently for either AFI.

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast summary | begin ^Neighbor
Neighbor      Spk    AS  MsgRcvd  MsgSent    TblVer  InQ  OutQ  Up/Down  St/PfxRcd
13.0.0.5      0     13   16164   15357      314     0    0  19:52:21      0
13.0.0.8      0     13   16048   15370      314     0    0  19:52:21      7
13.0.0.11     0     13   15196   15353      314     0    0    1d17h      0

RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast summary | begin ^Neighbor
Neighbor      Spk    AS  MsgRcvd  MsgSent    TblVer  InQ  OutQ  Up/Down  St/PfxRcd
13.0.0.5      0     13   16172   15365      325     0    0  19:53:36      0
13.0.0.8      0     13   16056   15378      325     0    0  19:53:36      7
13.0.0.11     0     13   15204   15360      325     0    0    1d17h      0

R2#show bgp vpnv4 unicast all summary | begin ^Neighbor
Neighbor      V      AS  MsgRcvd  MsgSent    TblVer  InQ  OutQ  Up/Down  State/PfxRcd
24.0.0.6      4      24    7988    8045      417     0    0  20:23:02      0
24.0.0.7      4      24   16240   16405      417     0    0    1d18h      0
24.0.0.14     4      24   14620   15812      417     0    0    1d16h      3

R2#show bgp vpnv6 unicast all summary | begin ^Neighbor
Neighbor      V      AS  MsgRcvd  MsgSent    TblVer  InQ  OutQ  Up/Down  State/PfxRcd
24.0.0.6      4      24    7995    8053      487     0    0  20:24:17      0
24.0.0.7      4      24   16248   16413      487     0    0    1d18h      0
24.0.0.14     4      24   14628   15820      487     0    0    1d16h      4
```

After receiving VPN routes from the PE routers in each AS (and locally originated, since each RR is also a PE for at least one VPN), these are advertised to the ASBRs. This is true for all RDs. On CSR2, we see that all of the routes with RD 24:2 and RD 24:3 are advertised to CSR6 as an example.

```
R2#show bgp vpnv4 unicast all neighbors 24.0.0.6 advertised-routes | begin
Network

      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 24:2 (default for vrf OSPF)
*> 10.2.9.0/24      0.0.0.0          0          32768 ?
*> 10.4.4.4/32      10.2.9.9         501         32768 ?
*> 10.9.9.9/32      10.2.9.9          1          32768 ?
Route Distinguisher: 24:3 (default for vrf EIGRP)
*> 10.1.1.1/32      10.1.2.1        10880         32768 ?
*> 10.1.2.0/24      0.0.0.0          0          32768 ?
*> 10.1.13.0/24     10.1.2.1        51210240      32768 ?
*>i 10.13.13.13/32   24.0.0.14       10752        100          0 ?
*>i 10.13.14.0/24   24.0.0.14          0        100          0 ?
```

However, CSR6 does not have any of these routes. If the ASBR doesn't learn the VPN routes, there is no possible way that they can be advertised across the AS boundary. The same issue exists on all ASBRs; I show a few other ASBRs as well, along with VPNv6 AFI outputs. This isn't an AS-specific or router-specific problem at this point.

```
R6#show bgp vpnv4 unicast all  
[no output]
```

```
RP/0/0/CPU0:XRv1#show bgp vpnv6 unicast  
[no output]
```

```
R5#show bgp vpnv4 unicast all  
[no output]
```

```
R7#show bgp vpnv6 unicast all  
[no output]
```

To troubleshoot the issue, I enable BGP update debugging for VPNv4 on CSR6. We can see routes arriving from both the OSPF and EIGRP VPNs (RD 24:2 and 24:3 respectively) and both are rejected. This output means that the RTs were not imported into any local VRF on CSR6, so there is no logical reason for CSR6 to retain these prefixes. Doing so, in the majority of cases, is a waste of memory.

```
R6#debug bgp vpnv4 unicast updates in  
BGP updates debugging is on (inbound) for address family: VPNv4 Unicast
```

```
BGP(4): 24.0.0.2 rcvd UPDATE w/ attr: nexthop 24.0.0.14, origin ?, localpref  
100, metric 0, originator 24.0.0.14, clusterlist 24.0.0.2, extended community  
RT:24:3 Cost:pre-bestpath:128:10240 0x8800:32768:0 0x8801:3:256  
0x8802:65280:2560 0x8803:1:1500 0x8806:0:402653198  
BGP(4): 24.0.0.2 rcvd 24:3:10.13.14.0/24, label 94007 -- DENIED due to:  
extended community not supported;
```

```
BGP(4): 24.0.0.2 rcvd UPDATE w/ attr: nexthop 24.0.0.2, origin ?, localpref  
100, metric 501, extended community RT:24:2 OSPF ROUTER ID:10.2.9.2:0 OSPF  
RT:0.0.0.0:2:0  
BGP(4): 24.0.0.2 rcvd 24:2:10.4.4.4/32, label 2006 -- DENIED due to:  
extended community not supported;
```

```
BGP(4): 24.0.0.2 rcvd UPDATE w/ attr: nexthop 24.0.0.2, origin ?, localpref  
100, metric 1, extended community RT:24:2 OSPF ROUTER ID:10.2.9.2:0 OSPF  
RT:0.0.0.0:2:0  
BGP(4): 24.0.0.2 rcvd 24:2:10.9.9.9/32, label 2009 -- DENIED due to:  
extended community not supported;
```

There are three solutions to this problem, and we will demonstrate all three. The first and most obvious solution is to configure the VRFs locally on the ASBRs. This reduces scalability and it introduces some of the limitations present in option A. While there need not be any interfaces inside of the VRF, having the VRF locally to import the RTs in question will permit BGP to retain the routes. We will configure this on CSR6; note that no RTs are exported. The VRFs are essentially importing routes just to appease BGP filtering policies.

```

! CSR6
vrf definition EIGRP
  rd 24:3
  address-family ipv4
    route-target import 24:3
  address-family ipv6
    route-target import 24:3

vrf definition OSPF
  rd 24:2
  address-family ipv4
    route-target import 24:2
  address-family ipv6
    route-target import 24:2

```

To confirm proper operation, we look at all of the VPN routes received. We now see VPN routes inside VRF EIGRP and OSPF, which is exactly what the RR saw. This is true for IPv4 and IPv6 and is a valid solution for option B. Note that, since the VRFs are configured on the ASBR, BGP is able to identify an RD to a VRF and displays the VRF name in the output below.

```

R6#show bgp vpnv4 unicast all | begin Network

```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 24:2 (default for vrf OSPF)					
*>i 10.2.9.0/24	24.0.0.2	0	100	0 ?	
*>i 10.4.4.4/32	24.0.0.2	501	100	0 ?	
*>i 10.9.9.9/32	24.0.0.2	1	100	0 ?	
Route Distinguisher: 24:3 (default for vrf EIGRP)					
*>i 10.1.1.1/32	24.0.0.2	10880	100	0 ?	
*>i 10.1.2.0/24	24.0.0.2	0	100	0 ?	
*>i 10.1.13.0/24	24.0.0.2	51210240	100	0 ?	
*>i 10.13.13.13/32	24.0.0.14	10752	100	0 ?	
*>i 10.13.14.0/24	24.0.0.14	0	100	0 ?	

```

R6#show bgp vpnv6 unicast all | begin Network

```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 24:2 (default for vrf OSPF)					
*>i ::10:4:4:4/128	::FFFF:24.0.0.2	501	100	0 ?	
*>i ::10:9:9:9/128	::FFFF:24.0.0.2	1	100	0 ?	
*>i FD00::2/128	::FFFF:24.0.0.2	0	100	0 i	
Route Distinguisher: 24:3 (default for vrf EIGRP)					
*>i ::10:1:1:1/128	::FFFF:24.0.0.2	10880	100	0 ?	
*>i ::10:13:13:13/128	::FFFF:24.0.0.14	10752	100	0 ?	
*>i FD00:10:1:2::/64	::FFFF:24.0.0.14	51215360	100	0 ?	
*>i FD00:10:1:13::/64	::FFFF:24.0.0.2	51210240	100	0 ?	

```
*>i FD00:10:13:14::/64
::FFFF:24.0.0.14
0      100      0 ?
```

The second option is to configure the ASBR as a route-reflector. Specifically, the ASBR would identify the RR as an RR-client, which seems awkward since ASBRs have no iBGP routes to reflect. Since RRs for VPNv4/v6 are required to distribute VPN routes to all iBGP peers within the AS, they cannot be selective (unless a specific RT constraint advertise feature is enabled, which it presently is not) on which VPN routes are advertised. VPN routes from an RR-client are installed in the BGP table and the best path within each RD is advertised onward. CSR7 only has one iBGP peer, so making it an RR doesn't have any negative network effects in this particular topology. The drawback of approach is revealed when multiple RRs are present in each AS, which is very common. The ASBR will be reflecting VPN routes back towards the RRs; although the originator-ID or the cluster-ID will eventually break the advertisement loop, it is a sloppy design and wastes memory. It is, however, a valid solution for configuring option B.

```
! CSR7
router bgp 24
 address-family vpnv4
   neighbor 24.0.0.2 route-reflector-client
 address-family vpnv6
   neighbor 24.0.0.2 route-reflector-client
```

After configuring this and waiting for BGP to converge, CSR7 now has all of the VPN routes for IPv4/v6. Unlike the previous option of configuring each VRF locally, this solution scales better. The output does not list the VRF alongside the RD because CSR7 has no idea what the VRF names are; it only sees the RD. This solution is more scalable than configuring every VRF locally on the option B ASBRs.

```
R7#show bgp vpnv4 unicast all | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 24:2
*>i 10.2.9.0/24      24.0.0.2          0      100      0 ?
*>i 10.4.4.4/32      24.0.0.2          501     100      0 ?
*>i 10.9.9.9/32      24.0.0.2          1      100      0 ?
Route Distinguisher: 24:3
*>i 10.1.1.1/32      24.0.0.2          10880   100      0 ?
*>i 10.1.2.0/24      24.0.0.2          0       100      0 ?
[snip]
```

```
R7#show bgp vpnv6 unicast all | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 24:2
*>i ::10:4:4:4/128   ::FFFF:24.0.0.2    501     100      0 ?
*>i ::10:9:9:9/128   ::FFFF:24.0.0.2    1       100      0 ?
*>i FD00::2/128      ::FFFF:24.0.0.2    0       100      0 i
Route Distinguisher: 24:3
*>i ::10:1:1:1/128   ::FFFF:24.0.0.2    10880   100      0 ?
```

```
*>i ::10:13:13:13/128
::FFFF:24.0.0.14
10752 100 0 ?
[snip]
```

The third and most preferred option is to simply instruct the ASBRs to retain all VPN routes regardless of whether the RTs are imported or not. This feature is specific to option B and has little value outside of an option B ASBR. We will configure this option on CSR5 for both VPNv4/v6. The command effectively instructs the router to not automatically filter RTs that are not locally imported. The double-negative in XE syntax means that all routes are retained.

```
! CSR5
router bgp 13
 address-family vpnv4
   no bgp default route-target filter
 address-family vpnv6
   no bgp default route-target filter
```

Unlike the other solutions, this requires a soft route refresh to take effect since the configuration is local-only. AS 13 has three VRFs, but CSR5 is not aware of them. Like the second option, the VRFs are not defined locally, so CSR5 only has RD-level visibility. Compared to the other solutions, this option has no limitations in terms of scalability or loops. A general limitation of option B is that the ASBRs must always retain these VPN routes, which means they must be large routers with abundant memory. They also need to explicitly negotiate every AFI that must be supported between ASes (VPNv4, VPNv6, L2VPN, IPv4 MDT, etc). Normally these requirements were only relevant for RRs but the requirement is extended to ASBRs. This is not specific to the three options discussed and is basic option B architecture.

```
R5#clear bgp vpnv4 unicast * soft
R5#clear bgp vpnv6 unicast * soft
```

```
R5#show bgp vpnv4 unicast all | begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 13:1					
*>i 110.0.0.0/32	13.0.0.8	0	100	0	100 ?
*>i 110.0.0.1/32	13.0.0.8	0	100	0	100 ?
*>i 110.0.0.2/32	13.0.0.8	0	100	0	100 ?
*>i 110.0.0.3/32	13.0.0.8	0	100	0	100 ?
Route Distinguisher: 13:2					
*>i 10.4.4.4/32	13.0.0.8	1	100	0	?
*>i 10.4.8.0/24	13.0.0.8	0	100	0	?
*>i 10.9.9.9/32	13.0.0.8	501	100	0	?
Route Distinguisher: 13:3					
*>i 10.3.3.3/32	13.0.0.12	10880	100	0	?
*>i 10.3.12.0/24	13.0.0.12	0	100	0	?

```
R5#show bgp vpnv6 unicast all | begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 13:1					
*>i ::110:0:0:0/128	::FFFF:13.0.0.8	0	100	0	100 ?
*>i ::110:0:0:1/128	::FFFF:13.0.0.8	0	100	0	100 ?
*>i ::110:0:0:2/128	::FFFF:13.0.0.8	0	100	0	100 ?
*>i ::110:0:0:3/128	::FFFF:13.0.0.8	0	100	0	100 ?
Route Distinguisher: 13:2					
*>i ::10:4:4:4/128	::FFFF:13.0.0.8	1	100	0	?
*>i ::10:9:9:9/128	::FFFF:13.0.0.8	501	100	0	?
*>i FD00::8/128	::FFFF:13.0.0.8	0	100	0	i
Route Distinguisher: 13:3					
*>i ::10:3:3:3/128	::FFFF:13.0.0.12	10880	100	0	?
*>i FD00:10:3:12::/64	::FFFF:13.0.0.12	0	100	0	?

We have not yet looked at IOS XR and how the same problem is solved. All three options still exist, but we will only configure the third. Configuring the VRFs locally or making XRv1 an RR is nothing new in terms of configuration, and logically, we know it will work identically as it does on XE. Instead, we configure XRv1 to retain VPN routes for VPNv4 and VPNv6. The command syntax in XR is easier than XE, removing the double negative and simply saying “retain the routes” rather than “don’t disallow the routes”. XR also introduces an RPL attach point; we can specify which routes we want to retain based on RT, or simply specify all routes. XE does not have this capability. I demonstrate both techniques below.

```

! XRv1
extcommunity-set rt RT_OSPF
  13:2
end-set

route-policy RPL_RETAIN_RT_V6
  if extcommunity rt matches-any RT_OSPF then
    drop
  else
    pass
  endif
end-policy

router bgp 13
  address-family vpnv4 unicast
    retain route-target all
  address-family vpnv6 unicast
    retain route-target route-policy RPL_RETAIN_RT_V6

```

For VPNv4, we see routes from all three RDs in the VPNv4 table. For VPNv6, we only see routes from RDs 13:1 and 13:2; this implies that XRv1 cannot be an ASBR for VRF OSPF. We cannot match the RD at this attach point, but since I matched the RT to the RD artificially, I am effectively removing all routes

from an entire VPN by filtering route targets. This can be used for traffic engineering since the OSPF VPN traffic must always traverse CSR5.

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast | begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 13:1					
*>i110.0.0.0/32	13.0.0.8	0	100	0	100 ?
*>i110.0.0.1/32	13.0.0.8	0	100	0	100 ?
*>i110.0.0.2/32	13.0.0.8	0	100	0	100 ?
*>i110.0.0.3/32	13.0.0.8	0	100	0	100 ?
Route Distinguisher: 13:2					
*>i10.4.4.4/32	13.0.0.8	1	100	0	?
*>i10.4.8.0/24	13.0.0.8	0	100	0	?
*>i10.9.9.9/32	13.0.0.8	501	100	0	?
Route Distinguisher: 13:3					
*>i10.3.3.3/32	13.0.0.12	10880	100	0	?
*>i10.3.12.0/24	13.0.0.12	0	100	0	?


```
RP/0/0/CPU0:XRv1#show bgp vpnv6 unicast | begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 13:1					
*>i::110:0:0:0/128	13.0.0.8	0	100	0	100 ?
*>i::110:0:0:1/128	13.0.0.8	0	100	0	100 ?
*>i::110:0:0:2/128	13.0.0.8	0	100	0	100 ?
*>i::110:0:0:3/128	13.0.0.8	0	100	0	100 ?
Route Distinguisher: 13:3					
*>i::10:3:3:3/128	13.0.0.12	10880	100	0	?
*>ifd00:10:3:12::/64	13.0.0.12	0	100	0	?

Now that the ASBRs have the VPN routes, we must configure eBGP VPNv4/v6 peers. This is a straightforward process and introduces no new technologies or techniques. For brevity, I only show the configurations for CSR5 and XRv1 inside AS 13.

```
! CSR5
router bgp 13
 neighbor 10.5.6.6 remote-as 24
 neighbor 10.5.7.7 remote-as 24
 address-family vpnv4
  neighbor 10.5.6.6 activate
  neighbor 10.5.7.7 activate
 address-family vpnv6
  neighbor 10.5.6.6 activate
  neighbor 10.5.7.7 activate

! XRv1
router bgp 13
 neighbor 10.6.11.6
  remote-as 24
```



```

address-family vpnv4 unicast
  route-policy RPL_PASS in
  route-policy RPL_PASS out
address-family vpnv6 unicast
  route-policy RPL_PASS in
  route-policy RPL_PASS out

```

As soon as the BGP peers come up, the XE routers will display a new syslog message on every interface that establishes a VPN eBGP peer. Below is a message from CSR5 as an example. It states that a new command was automatically configured on the transit links to support BGP-based MPLS forwarding. This is explained in more detail later.

```

! CSR5
%BGP_LMM-6-AUTOGEN1: The mpls bgp forwarding command has been configured on
interface: GigabitEthernet2.556

```

To verify it, we check the interface configuration and see this new command. We also check the MPLS interfaces to see that it has been enabled for BGP forwarding as well. When we verify the data plane, we will be relying on BGP to perform label-swaps of the VPN label between ASes, which effectively changes the LSP.

```

! CSR5
interface GigabitEthernet2.556
  encapsulation dot1Q 3556
  ip address 10.5.6.5 255.255.255.0
[snip]
mpls bgp forwarding

```

```
R5#show mpls interfaces
```

Interface	IP	Tunnel	BGP	Static	Operational
GigabitEthernet2.551	Yes (ldp)	Yes	No	No	Yes
GigabitEthernet2.558	Yes (ldp)	Yes	No	No	Yes
GigabitEthernet2.556	No	No	Yes	No	Yes
GigabitEthernet2.557	No	No	Yes	No	Yes

Before we do any data-plane verifications with BGP forwarding, we will ensure the VPNv4/v6 neighbors came up on all routers. A quick check on CSR5 and CSR6 confirms this, and we should see a number greater than zero for all inter-AS VPN peers. CSR6 does not appear to be receiving any inter-AS VPN routes, so this indicates a problem.

```

R5#show bgp vpnv4 unicast all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ  Up/Down
State/PfxRcd
10.5.6.6      4      24      29      25      80    0    0 00:07:53    8
10.5.7.7      4      24      28      42      80    0    0 00:07:07    8
13.0.0.12     4      13     7596    8096     80    0    0 20:47:47    9

```

```
R5#show bgp vpnv6 unicast all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down
State/PfxRcd
10.5.6.6      4      24      29      25      250    0    0 00:08:03    8
10.5.7.7      4      24      28      42      250    0    0 00:07:17    8
13.0.0.12     4      13     7597    8097    250    0    0 20:47:57    9
```

```
R6#show bgp vpnv6 unicast all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down
State/PfxRcd
10.5.6.5      4      13      40      49      942    0    0 00:09:09    0
10.6.11.11    4      13      29      61      942    0    0 00:07:42    0
24.0.0.2     4      24     8453    8350    942    0    0 21:19:18    8
```

```
R6#show bgp vpnv4 unicast all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down
State/PfxRcd
10.5.6.5      4      13      40      49      266    0    0 00:09:39    0
10.6.11.11    4      13      29      61      266    0    0 00:08:11    0
24.0.0.2     4      24     8456    8353    266    0    0 21:19:48    8
```

We saw no issue on CSR5, so we check CSR7 and XRv11. They also appear to be learning routes correctly, making CSR6 the only router that is not.

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast summary | begin ^Neighbor
Neighbor      Spk    AS MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
10.6.11.6      0     24      64      32      403    0    0 00:10:59    8
13.0.0.12     0     13    15729    15571    403    0    0 1d18h      9
```

```
RP/0/0/CPU0:XRv1#show bgp vpnv6 unicast summary | begin ^Neighbor
Neighbor      Spk    AS MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
10.6.11.6      0     24      64      32      408    0    0 00:11:06    8
13.0.0.12     0     13    15730    15572    408    0    0 1d18h      6
```

```
R7#show bgp vpnv4 unicast all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down  State/PfxRcd
10.5.7.5      4      13      47      34      316    0    0 00:12:06    9
24.0.0.2     4      24      270      263      316    0    0 00:39:28    8
```

```
R7#show bgp vpnv6 unicast all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down  State/PfxRcd
10.5.7.5      4      13      47      34      796    0    0 00:12:06    9
24.0.0.2     4      24      271      263      796    0    0 00:39:29    8
```

Debugging BGP updates for VPNv4 reveals the problem. This is the same issue we noticed when CSR6 was trying to learn intra-AS routes when it was not importing the RTs locally. Since all other ASBRs are either configured as RRs or to retain RTs, CSR6 must import AS 13's RTs for each VPN. Hopefully this illustrates why the option B ASBR solution on CSR6 is not a good choice; in addition to scaling poorly, it is difficult to maintain.

```
R6#debug bgp vpnv4 unicast updates in
```

BGP updates debugging is on (inbound) for address family: VPNv4 Unicast

```
BGP(4): 10.6.11.11 rcvd UPDATE w/ attr: nexthop 10.6.11.11, origin ?, merged
path 13, AS_PATH , extended community RT:13:2 OSPF ROUTER ID:10.4.8.8:0 OSPF
RT:0.0.0.0:2:0
```

```
BGP(4): 10.6.11.11 rcvd 13:2:10.9.9.9/32, label 91009 -- DENIED due to:
extended community not supported;
```

```
BGP(4): 10.6.11.11 rcvd UPDATE w/ attr: nexthop 10.6.11.11, origin ?, merged
path 13, AS_PATH , extended community RT:13:3 0x8800:32768:0 0x8801:3:288
0x8802:65281:2560 0x8803:1:1500 0x8806:0:167971843
```

```
BGP(4): 10.6.11.11 rcvd 13:3:10.3.3.3/32, label 91010 -- DENIED due to:
extended community not supported;
```

We quickly import these AS 13 RTs into VRFs OSPF and EIGRP for both AFIs. Then, we see the routes are accepted from the eBGP peer.

```
! CSR6
vrf definition EIGRP
  address-family ipv4
    route-target import 13:3
  address-family ipv6
    route-target import 13:3
```

```
vrf definition OSPF
  address-family ipv4
    route-target import 13:2
  address-family ipv6
    route-target import 13:2
```

```
R6#show bgp vpnv6 unicast all summary | begin ^Neighbor
```

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
10.5.6.5	4	13	97	78	950	0	0	00:22:59	5
10.6.11.11	4	13	56	90	950	0	0	00:21:31	2
24.0.0.2	4	24	8577	8449	963	0	0	21:33:08	8

```
R6#show bgp vpnv4 unicast all summary | begin ^Neighbor
```

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
10.5.6.5	4	13	97	78	270	0	0	00:23:00	5
10.6.11.11	4	13	56	90	270	0	0	00:21:32	5
24.0.0.2	4	24	8577	8449	278	0	0	21:33:09	8

The number of routes we received from the eBGP peers is less than we expected; other neighbors showed 8 or 9 routes. With debugging still on, we see that the central services routes have not been imported.

```
! CSR6
BGP(4): 10.5.6.5 rcvd 13:1:110.0.0.0/32, label 5021 -- DENIED due to:
extended community not supported;
```

```
BGP(4): 10.5.6.5 rcvd 13:1:110.0.0.1/32, label 5013 -- DENIED due to:
extended community not supported;
BGP(4): 10.5.6.5 rcvd 13:1:110.0.0.2/32, label 5014 -- DENIED due to:
extended community not supported;
BGP(4): 10.5.6.5 rcvd 13:1:110.0.0.3/32, label 5015 -- DENIED due to:
extended community not supported;
```

Even worse, CSR6 now has to worry about the central services VPN. We can configure the central service VPN locally so that it can import those RTs, which is highly undesirable. The best way to salvage this situation is to import RT:13:1 into VRFs OSPF and EIGRP rather than create a new VRF. Now, we can see that CSR6 is importing all of the routes from AS 13. Clearly this approach is undesirable but might be the only option if the (low-end) ASBR does not support RT retention or RR capabilities.

```
! CSR6
vrf definition EIGRP
  address-family ipv4
    route-target import 13:1
  address-family ipv6
    route-target import 13:1
```

```
vrf definition OSPF
  address-family ipv4
    route-target import 13:1
  address-family ipv6
    route-target import 13:1
```

```
R6#show bgp vpnv4 unicast all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
10.5.6.5       4      13     130     100     290    0    0 00:27:19      9
10.6.11.11     4      13      68     111     290    0    0 00:25:51      9
24.0.0.2       4      24    8623    8480     290    0    0 21:37:27      8
```

```
R6#show bgp vpnv6 unicast all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
10.5.6.5       4      13     130     100     979    0    0 00:27:19      9
10.6.11.11     4      13      68     111     979    0    0 00:25:51      6
24.0.0.2       4      24    8623    8480     979    0    0 21:37:28      8
```

Upon receiving the VPN routes, the ASBRs ultimately need to advertise the routes to the remote PEs where the customers attach. In both ASes, this is done via the RRs. I select a VPN route and check the RR in AS 24 for its presence; we are immediately presented with next-hop reachability problems.

```
R2#show bgp vpnv4 unicast rd 13:1 110.0.0.0/32
BGP routing table entry for 13:1:110.0.0.0/32, version 0
Paths: (2 available, no best path)
  Not advertised to any peer
  Refresh Epoch 3
  13 100, (Received from a RR-client)
```

```

10.5.6.5 (inaccessible) (via default) from 24.0.0.6 (24.0.0.6)
  Origin incomplete, metric 0, localpref 100, valid, internal
  Extended Community: RT:13:1
  mpls labels in/out nolabel/5021
  rx pathid: 0, tx pathid: 0
Refresh Epoch 1
13 100, (Received from a RR-client)
10.5.7.5 (inaccessible) (via default) from 24.0.0.7 (24.0.0.7)
  Origin incomplete, metric 0, localpref 100, valid, internal
  Extended Community: RT:13:1
  mpls labels in/out nolabel/5021
  rx pathid: 0, tx pathid: 0

```

This is the classic iBGP behavior of not modifying the next-hop of any BGP routes by default. The transit links were never advertised into IGP and the BGP routers never used “next-hop-self” towards the RR. Either solution is valid for option B but the LSPs change slightly depending on which option we use. Inside AS 24, I will advertise the transit links into IGP. CSR6 uses IS-IS passive interfaces and CSR7 uses redistribution for variety.

```

! CSR6
router isis 24
  passive-interface GigabitEthernet2.556
  passive-interface GigabitEthernet2.561

! CSR7
ip prefix-list PL_R5R7 seq 5 permit 10.5.7.0/24

route-map RM_CONN_TO_ISIS permit 10
  match ip address prefix-list PL_R5R7

router isis 24
  redistribute connected route-map RM_CONN_TO_ISIS

```

To confirm the IS-IS advertisement was successful, I check the LSP details. This is done locally on each ASBR.

```

R6#show isis database detail | section R6.00-00
R6.00-00          0x000000D1    0x591E          971          0/0/0
[snip]
  IP Address:    24.0.0.6
  Metric: 0      IP 24.0.0.6/32
  Metric: 0      IP 10.5.6.0/24
  Metric: 0      IP 10.6.11.0/24
  IPv6 Address:  ::24:0:0:6
  Metric: 0      IPv6 (MT-IPv6) ::24:0:0:6/128
  Metric: 0      IPv6 (MT-IPv6) FD00:10:5:6::/64

```

```

R7#show isis database detail | section R7.00-00
R7.00-00          * 0x000000D0    0x732C          1099          0/0/0
[snip]
  IP Address:      24.0.0.7
  Metric: 0         IP 24.0.0.7/32
  Metric: 0         IP 10.5.7.0/24
  IPv6 Address:    ::24:0:0:7
  Metric: 0         IPv6 (MT-IPv6) ::24:0:0:7/128

```

Now CSR2 has valid routes for remote VPN destinations. CSR2 selects the route via CSR6 as best due to having a lower BGP RID and advertises it only XRv4. It would also be used for redistribution into EIGRP/OSPF since this is a central services route. Because CSR6 and CSR7 did not adjust the VPN next-hop, the VPN label of 5021 remains unchanged as allocated by CSR5.

```

R2#show bgp vpnv4 unicast rd 13:1 110.0.0.0/32
BGP routing table entry for 13:1:110.0.0.0/32, version 422
Paths: (2 available, best #1, no table)
  Advertised to update-groups:
    1
  Refresh Epoch 3
  13 100, (Received from a RR-client)
    10.5.6.5 (metric 20) (via default) from 24.0.0.6 (24.0.0.6)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:13:1
      mpls labels in/out nolaabel/5021
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  13 100, (Received from a RR-client)
    10.5.7.5 (metric 20) (via default) from 24.0.0.7 (24.0.0.7)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Extended Community: RT:13:1
      mpls labels in/out nolaabel/5021
      rx pathid: 0, tx pathid: 0

```

When we check XRv4 for this central services route, it has nothing from RD 13:1. This is because it is not importing RT:13:1, which is an AS 13 route target. This is one of the option B limitations; the providers must agree on the RT policies.

```

RP/0/0/CPU0:XRv4#show bgp vpnv4 unicast rd 13:1
[no output]

```

If we expect the AS 24 VPN customers to have reachability to the AS 13 VPN customers, AS 24 must import the RTs exported by AS 13. On XRv4, this means importing RT:13:3 for EIGRP VPN reachability as well as RT:13:1 for central services reachability. Once we import these AS 13 route targets, we quickly check XRv4 to ensure it imports the EIGRP VPN and central services routes from AS 13.

```

! XRv4
vrf EIGRP
  address-family ipv4 unicast
    import route-target
      13:1
      13:3

  address-family ipv6 unicast
    import route-target
      13:1
      13:3

```

```
RP/0/0/CPU0:XRv4#show bgp vpnv4 unicast rd 13:1 | begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 13:1					
*>i110.0.0.0/32	10.5.6.5	0	100	0	13 100 ?
*>i110.0.0.1/32	10.5.6.5	0	100	0	13 100 ?
*>i110.0.0.2/32	10.5.6.5	0	100	0	13 100 ?
*>i110.0.0.3/32	10.5.6.5	0	100	0	13 100 ?

```
RP/0/0/CPU0:XRv4#show bgp vpnv4 unicast rd 13:3 | begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 13:3					
*>i10.3.3.3/32	10.5.6.5	0	100	0	13 ?
*>i10.3.12.0/24	10.5.6.5	0	100	0	13 ?

We must do the same thing on CSR2 for both the EIGRP and OSPF VPNs. Don't be fooled; just because CSR2 is an RR and retains all VPN routes, this does not automatically mean the customer VPNs on the RR have reachability. The output below proves it as CSR2 is preferring CSR1 (CE) as the next-hop towards the remote EIGRP VPN. If CSR2 actually had a BGP route in its VRF for this, the BGP-learned route would be preferred (this one is BGP originated).

```
R2#show bgp vpnv4 unicast vrf EIGRP 10.3.3.3/32
```

```
BGP routing table entry for 24:3:10.3.3.3/32, version 3662
```

```
Paths: (1 available, best #1, table EIGRP)
```

```
  Advertised to update-groups:
```

```
    1
```

```
  Refresh Epoch 1
```

```
  Local
```

```
    10.1.2.1 (via vrf EIGRP) from 0.0.0.0 (24.0.0.2)
```

```
      Origin incomplete, metric 1229056640, localpref 100, weight 32768,
```

```
valid, sourced, best
```

```
      Extended Community: RT:24:3
```

```
        Cost:pre-bestpath:128:1229056640 (default-918427007) 0x8800:32768:0
```

```
        0x8801:3:61452576 0x8802:65353:2560 0x8803:65281:1500
```

```
        0x8806:0:167971843
```

```
      mpls labels in/out 2072/nolabel
```

```
      rx pathid: 0, tx pathid: 0x0
```

Below are the updated RT policies on CSR2 needed to enable inter-AS reachability. Both EIGRP and OSPF VPNs import the central services RT of 13:1, along with their specific VPN RTs per VPN. Notice that the OSPF VPN no longer needs to import RT:24:2 as CSR2 is the only router exporting it. For cleanup, we remote this from the RT policy, although leaving it there has no operational effect.

```
! CSR2
vrf definition EIGRP
  address-family ipv4
    route-target import 13:3
    route-target import 13:1
  address-family ipv6
    route-target import 13:3
    route-target import 13:1

vrf definition OSPF
  address-family ipv4
    route-target import 13:2
    route-target import 13:1
    no route-target import 24:2
  address-family ipv6
    route-target import 13:2
    route-target import 13:1
    no route-target import 24:2
```

Using the same example as above, we can now see the VPN routes pointing in the right direction.

```
R2#show bgp vpnv4 unicast vrf EIGRP 10.3.3.3/32
BGP routing table entry for 24:3:10.3.3.3/32, version 3612
Paths: (1 available, best #1, table EIGRP)
  Not advertised to any peer
  Refresh Epoch 4
  13, (Received from a RR-client), imported path from 13:3:10.3.3.3/32
(global)
  10.5.6.5 (metric 20) (via default) from 24.0.0.6 (24.0.0.6)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    Extended Community: RT:13:3 0x8800:32768:0 0x8801:3:288
      0x8802:65281:2560 0x8803:1:1500 0x8806:0:167971843
    Connector Attribute: count=1
      type 1 len 12 value 13:3:13.0.0.12
    mpls labels in/out nolabel/5020
    rx pathid: 0, tx pathid: 0x0
```

We quickly check the EIGRP VPN to see the routes being populated for IPv4 and IPv6. We will do more detailed verification once AS 13 has been properly configured. The EIGRP extended communities that allow the VPN routes to appear internal are transitive as seen earlier in option A tests. The route to CSR3

is an internal route, but the central services routes as external as these are BGP originated. This is true for IPv4 and IPv6 and is the expected result.

```
R1#show ip route eigrp | begin Gate
Gateway of last resort is not set
```

```
10.0.0.0/8 is variably subnetted, 9 subnets, 2 masks
D 10.3.3.3/32 [90/16000] via 10.1.2.2, 00:03:12, GigabitEthernet2.512
D 10.3.12.0/24 [90/15360] via 10.1.2.2, 00:03:12, GigabitEthernet2.512
D 10.13.13.13/32
[90/15880] via 10.1.2.2, 23:20:55, GigabitEthernet2.512
D 10.13.14.0/24 [90/15360] via 10.1.2.2, 23:20:55,
GigabitEthernet2.512
110.0.0.0/32 is subnetted, 4 subnets
D EX 110.0.0.0
[170/51307520] via 10.1.13.13, 00:15:12, GigabitEthernet2.513
D EX 110.0.0.1
[170/51307520] via 10.1.13.13, 00:15:12, GigabitEthernet2.513
D EX 110.0.0.2
[170/51307520] via 10.1.13.13, 00:15:12, GigabitEthernet2.513
D EX 110.0.0.3
[170/51307520] via 10.1.13.13, 00:15:12, GigabitEthernet2.513
```

```
R1#show ipv6 route eigrp | begin Appl
a - Application
D ::10:3:3:3/128 [90/16000]
via FE80::2, GigabitEthernet2.512
D ::10:13:13:13/128 [90/15880]
via FE80::2, GigabitEthernet2.512
EX ::110:0:0:0/128 [170/51307520], tag 13
via FE80::13, GigabitEthernet2.513
[snip]
```

Shifting our attention to AS 13's RR, we expect to see next-hop inaccessibility for the VPN routes on XRv2. In AS 24, we corrected this by advertising the transit links into IGP.

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast rd 24:2 10.9.9.9 | begin 24,
24, (Received from a RR-client)
10.5.6.6 (inaccessible) from 13.0.0.5 (13.0.0.5)
Received Label 6036
Origin incomplete, metric 0, localpref 100, valid, internal, not-in-vrf
Received Path ID 0, Local Path ID 0, version 0
Extended community: OSPF router-id:10.2.9.2 OSPF route-type:0:2:0x0
RT:24:2
Path #2: Received by speaker 0
Not advertised to any peer
24, (Received from a RR-client)
10.6.11.6 (inaccessible) from 13.0.0.11 (13.0.0.11)
```

```
Received Label 6036
Origin incomplete, localpref 100, valid, internal, not-in-vrf
Received Path ID 0, Local Path ID 0, version 0
Extended community: OSPF router-id:10.2.9.2 OSPF route-type:0:2:0x0
RT:24:2
```

The AS 24 approach is less desirable than “next-hop-self”, at least in this context, as we will discover later when it comes to MPLS forwarding limitations on certain platforms. AS 13 will use the simpler “next-hop-self” approach on XRv1 and CSR5.

```
! CSR5
router bgp 13
 address-family vpnv4
   neighbor 13.0.0.12 next-hop-self
 address-family vpnv6
   neighbor 13.0.0.12 next-hop-self
```

```
! XRv1
router bgp 13
 neighbor 13.0.0.12
 address-family vpnv4 unicast
   next-hop-self
 address-family vpnv6 unicast
   next-hop-self
```

Checking XRv2, we can now see that the VPN routes are reachable. Because CSR5 and XRv1 changed the VPN next-hops, they also allocated new local labels.

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast rd 24:2 10.9.9.9 | begin 24,
24, (Received from a RR-client)
 13.0.0.5 (metric 3) from 13.0.0.5 (13.0.0.5)
   Received Label 5037
   Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, not-in-vrf
   Received Path ID 0, Local Path ID 1, version 334
   Extended community: OSPF router-id:10.2.9.2 OSPF route-type:0:2:0x0
RT:24:2
  Path #2: Received by speaker 0
  Not advertised to any peer
 24, (Received from a RR-client)
 13.0.0.11 (metric 3) from 13.0.0.11 (13.0.0.11)
   Received Label 91021
   Origin incomplete, localpref 100, valid, internal, import-candidate,
not-in-vrf
   Received Path ID 0, Local Path ID 0, version 0
   Extended community: OSPF router-id:10.2.9.2 OSPF route-type:0:2:0x0
RT:24:2
```

We will also run into the same inter-AS RT issue observed in AS 24. CSR8, for example, is a PE for the OSPF VPN. Looking at the BGP table, we do see a route, but upon further inspection we see that it was locally originated. This is because of the inter-AS OSPF backdoor; the route was learned from CSR4 and redistributed into BGP, which is not ideal. There are many ways to see this: the next-hop is CSR4, the MED is 501 (backdoor plus transit link), the route is sourced, the weight is 32,768, and the MPLS label is a local label (should be an outbound label from the ASBR). The route was not iBGP learned as it is not marked with “internal” as we would expect.

```
R8#show bgp vpnv4 unicast vrf OSPF 10.9.9.9/32
BGP routing table entry for 13:2:10.9.9.9/32, version 125
Paths: (1 available, best #1, table OSPF)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  Local
    10.4.8.4 (via vrf OSPF) from 0.0.0.0 (13.0.0.8)
      Origin incomplete, metric 501, localpref 100, weight 32768, valid,
sourced, best
      Extended Community: RT:13:2 OSPF ROUTER ID:10.4.8.8:0
        OSPF RT:0.0.0.0:2:0
        mpls labels in/out 8009/nolabel
      rx pathid: 0, tx pathid: 0x0
```

Below are the VRF RT updates on CSR8. For the shared services VPN, we must import the AS 24 routes for both EIGRP and OSPF VPNs. For the OSPF VPN, there is no need to import RT:13:2 since CSR8 is the only router exporting this RT now. Instead, we must import RT:24:2 which was exported by CSR2. This is the same cleanup we performed on CSR2. The RT policies become more complex when they are coordinated across AS boundaries.

```
! CSR8
vrf definition BGP
  address-family ipv4
    route-target import 24:2
    route-target import 24:3
  address-family ipv6
    route-target import 24:2
    route-target import 24:3

vrf definition OSPF
  address-family ipv4
    no route-target import 13:2
    route-target import 24:2
  address-family ipv6
    no route-target import 13:2
    route-target import 24:2
```

BGP shows the iBGP route now learned in the VPN table for OSPF, but there is a routing issue. We will resolve this later, but for now, we can see the proper path was imported across AS boundaries. The iBGP route is not preferred over the backdoor-originated route due to the weight attribute.

```
R8#show bgp vpnv4 unicast vrf OSPF 10.9.9.9/32
BGP routing table entry for 13:2:10.9.9.9/32, version 125
Paths: (2 available, best #2, table OSPF)
  Advertised to update-groups:
    2
  Refresh Epoch 1
    24, imported path from 24:2:10.9.9.9/32 (global)
      13.0.0.5 (metric 2) (via default) from 13.0.0.12 (13.0.0.12)
        Origin incomplete, metric 0, localpref 100, valid, internal
        Extended Community: RT:24:2 OSPF ROUTER ID:10.2.9.2:0
          OSPF RT:0.0.0.0:2:0
          Originator: 13.0.0.5, Cluster list: 13.0.0.12
          mpls labels in/out 8009/5037
          rx pathid: 0, tx pathid: 0
    Refresh Epoch 1
  Local
    10.4.8.4 (via vrf OSPF) from 0.0.0.0 (13.0.0.8)
      Origin incomplete, metric 501, localpref 100, weight 32768, valid,
sourced, best
      Extended Community: RT:13:2 OSPF ROUTER ID:10.4.8.8:0
        OSPF RT:0.0.0.0:2:0
        mpls labels in/out 8009/nolabel
        rx pathid: 0, tx pathid: 0x0
```

XRv2 is a PE servicing the EIGRP VPN and it must import RT:24:3 and stop importing RT:13:3 as a result of the new architecture.

```
! XRv2
vrf EIGRP
  address-family ipv4 unicast
    import route-target
    no 13:3
    24:3

  address-family ipv6 unicast
    import route-target
    no 13:3
    24:3
```

At this point, we will manually trace the path from CSR3 to XRv3 before sending any traffic. We just configured XRv2 to import the EIGRP VPN routes, so CSR3 should have an EIGRP internal route to XRv3's loopback.

```

R3#show ip route 10.13.13.13
Routing entry for 10.13.13.13/32
  Known via "eigrp 3", distance 90, metric 15880, type internal
  Redistributing via eigrp 3
  Last update from 10.3.12.12 on GigabitEthernet2.532, 00:00:27 ago
  Routing Descriptor Blocks:
  * 10.3.12.12, from 10.3.12.12, 00:00:27 ago, via GigabitEthernet2.532
    Route metric is 15880, traffic share count is 1
    Total delay is 21 microseconds, minimum bandwidth is 1000000 Kbit
    Reliability 255/255, minimum MTU 1500 bytes
    Loading 1/255, Hops 2

```

XRv2 performs a lookup in its VRF-aware BGP table for this route. The best-path is via 13.0.0.5 in the global table, and XRv2 adds label 5032 to the label stack.

```

RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf EIGRP 10.13.13.13/32 | begin 24,
[snip]
  24, (Received from a RR-client)
    13.0.0.5 (metric 3) from 13.0.0.5 (13.0.0.5)
      Received Label 5032
      Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, imported
      Received Path ID 0, Local Path ID 1, version 343
      Extended community: EIGRP route-info:0x8000:0 EIGRP AD:3:282 EIGRP
RHB:255:1:2560 EIGRP LM:0x0:1:1500 EIGRP VRR:0x0:13.13.13.10 RT:24:3
      Source VRF: default, Source Route Distinguisher: 24:3

```

The transport label is from LDP as the route to the BGP next-hop is IGP learned. The label stack becomes {8002 5032}.

```

RP/0/0/CPU0:XRv2#show ip route 13.0.0.5
Routing entry for 13.0.0.5/32
  Known via "ospf 13", distance 110, metric 3, type intra area
  Routing Descriptor Blocks:
  13.8.12.8, from 13.0.0.5, via GigabitEthernet0/0/0/0.582
    Route metric is 3
  No advertising protos.

```

```

RP/0/0/CPU0:XRv2#show mpls ldp bindings 13.0.0.5/32 neighbor 13.0.0.8
13.0.0.5/32, rev 12
  Local binding: label: 92004
  Remote bindings: (2 peers)
    Peer          Label
    -----
    13.0.0.8:0    8002

```

CSR8 performs PHP to expose label 5032 to CSR5.

```
R8#show mpls forwarding-table labels 8002
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
8002	Pop Label	13.0.0.5/32	1390704	Gi2.558	13.5.8.5

CSR5 receives label 5032 and performs a label swap to 6030. Notice that the prefix shows the full 96-bit VPN prefix including the RD. This is BGP VPNv4 influencing a label swap, which is something we have not seen yet.

```
R5#show mpls forwarding-table labels 5032
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
5032	6030	24:3:10.13.13.13/32	\		
			0	Gi2.556	10.5.6.6

CSR5 learned two eBGP VPNv4 paths to 10.13.13.13/32 within RD 24:3. CSR6 is preferred as it is the older eBGP route, so CSR6's label is used as the outgoing label.

```
R5#show bgp vpnv4 unicast rd 24:3 10.13.13.13/32
```

```
BGP routing table entry for 24:3:10.13.13.13/32, version 76
```

```
Paths: (2 available, best #2, no table)
```

```
Advertised to update-groups:
```

```
4 5
```

```
Refresh Epoch 2
```

```
24
```

```
10.5.7.7 (via default) from 10.5.7.7 (24.0.0.7)
```

```
Origin incomplete, localpref 100, valid, external
```

```
Extended Community: RT:24:3 0x8800:32768:0 0x8801:3:282
```

```
0x8802:65281:2560 0x8803:1:1500 0x8806:0:168627469
```

```
mpls labels in/out 5032/7033
```

```
rx pathid: 0, tx pathid: 0
```

```
Refresh Epoch 3
```

```
24
```

```
10.5.6.6 (via default) from 10.5.6.6 (24.0.0.6)
```

```
Origin incomplete, localpref 100, valid, external, best
```

```
Extended Community: RT:24:3 0x8800:32768:0 0x8801:3:282
```

```
0x8802:65281:2560 0x8803:1:1500 0x8806:0:168627469
```

```
mpls labels in/out 5032/6030
```

```
rx pathid: 0, tx pathid: 0x0
```

No additional transport label is needed because the route to the next-hop is connected. Oddly, the route is a /32; this is the result of the "mpls bgp forwarding" command that was automatically added to the configuration when the eBGP VPN neighbor came up. The logic is that routers must have a /32 route to the BGP next-hop. Some IOS platforms/versions will still forward traffic when VPN next-hops are not host routes, but XR never will. This is revealed later.

```
R5#show ip route 10.5.6.6
```

```
Routing entry for 10.5.6.6/32
```

```
Known via "connected", distance 0, metric 0 (connected, via interface)
```

```
Routing Descriptor Blocks:
```

```
* directly connected, via GigabitEthernet2.556
```

```
Route metric is 0, traffic share count is 1
```

When CSR6 receives packets with label 6030, it swaps 6030 for 94006. This is XRv4's VPN label for the prefix 10.13.13.13/32.

```
R6#show bgp vpnv4 unicast rd 24:3 10.13.13.13/32
```

```
BGP routing table entry for 24:3:10.13.13.13/32, version 3082
```

```
Paths: (1 available, best #1, table EIGRP)
```

```
Advertised to update-groups:
```

```
6
```

```
Refresh Epoch 12
```

```
Local
```

```
24.0.0.14 (metric 10) (via default) from 24.0.0.2 (24.0.0.2)
```

```
Origin incomplete, metric 10752, localpref 100, valid, internal, best
```

```
Extended Community: RT:24:3 Cost:pre-bestpath:128:10752 0x8800:32768:0
```

```
0x8801:3:282 0x8802:65281:2560 0x8803:1:1500 0x8806:0:168627469
```

```
Originator: 24.0.0.14, Cluster list: 24.0.0.2
```

```
mpls labels in/out 6030/94006
```

```
rx pathid: 0, tx pathid: 0x0
```

CSR6 may need to add another label for transport; in this case, it does not, since CSR6 is both the ASBR and penultimate hop towards XRv4. The outgoing label stack to XRv4 is 94006.

```
R6#show ip route 24.0.0.14
```

```
Routing entry for 24.0.0.14/32
```

```
Known via "isis", distance 115, metric 10, type level-2
```

```
Redistributing via isis 24
```

```
Last update from 24.6.14.14 on GigabitEthernet2.564, 03:21:33 ago
```

```
Routing Descriptor Blocks:
```

```
* 24.6.14.14, from 24.0.0.14, 03:21:33 ago, via GigabitEthernet2.564
```

```
Route metric is 10, traffic share count is 1
```

```
R6#show mpls ldp bindings 24.0.0.14 32 neighbor 24.0.0.14
```

```
lib entry: 24.0.0.14/32, rev 19
```

```
remote binding: lsr: 24.0.0.14:0, label: imp-null
```

XRv4 receives packets labeled 94006, removes all labels, and forwards the traffic to the customer. The LSP appears to be operational.

```
RP/0/0/CPU0:XRv4#show mpls forwarding vrf EIGRP prefix 10.13.13.13/32
```

Local	Outgoing	Prefix	Outgoing	Next Hop	Bytes

Label	Label	or ID	Interface	Switched
94006	Unlabelled	10.13.13.13/32 [V]	Gi0/0/0/0.534	10.13.14.13
				5882

A quick ping test shows that bidirectional connectivity does not exist. Debugging on XRv3 reveals that traffic is at least working in one direction, which is the LSP we just verified. XRv3 receives the echo request and sends a reply back to 10.3.3.3, but the reply is not making it back.

```
R3#ping 10.13.13.13 so 10.3.3.3
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.13.13.13, timeout is 2 seconds:
Packet sent with a source address of 10.3.3.3
.....
Success rate is 0 percent (0/5)
```

```
RP/0/0/CPU0:XRv3#debug icmp ipv4
ipv4_io[264]: IPv4 ICMP: GigabitEthernet0/0/0/0.534: Received echo request
from 10.3.3.3
ipv4_io[264]: IPv4 ICMP: GigabitEthernet0/0/0/0.534: Sending echo reply to
10.3.3.3
```

We will trace the LSP from XRv3 to CSR3, which could potentially follow a very different path. XRv3 has an EIGRP internal route to CSR3 as expected. The next-hop is via XRv4, which is the ingress LSR. Since the routes have been exchanged end-to-end, it is likely a data-plane issue.

```
RP/0/0/CPU0:XRv3#show route 10.3.3.3
Routing entry for 10.3.3.3/32
  Known via "eigrp 3", distance 90, metric 16000, type internal
  Routing Descriptor Blocks
    10.13.14.14, from 10.13.14.14, via GigabitEthernet0/0/0/0.534
      Route metric is 16000
  No advertising protos.
```

XRv4 has a VPN route for 10.3.3.3/32 which is via 10.5.6.5 using label 5020, which was allocated by CSR5. CSR5 is the remote ASBR, not the local one; since CSR6 did not change the VPN next-hop, it did not allocate a new label for this prefix. This makes sense as this behavior is observed in intra-AS VPNs all the time. It only makes sense for BGP to allocate a label for a prefix, and perform a label swap, if the BGP next-hop changes.

```
RP/0/0/CPU0:XRv4#show bgp vpnv4 unicast vrf EIGRP 10.3.3.3/32 | begin 13$
13
  10.5.6.5 (metric 10) from 24.0.0.2 (24.0.0.6)
    Received Label 5020
      Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, imported
      Received Path ID 0, Local Path ID 1, version 4419
```



```

Extended community: EIGRP route-info:0x8000:0 EIGRP AD:3:288 EIGRP
RHB:255:1:2560 EIGRP LM:0x0:1:1500 EIGRP VRR:0x0:3.12.3.10 RT:13:3
Originator: 24.0.0.6, Cluster list: 24.0.0.2
Connector: type: 1, Value:13:3:13.0.0.12
Source VRF: default, Source Route Distinguisher: 13:3

```

When we perform a route lookup for 10.5.6.5, we find a matching /24. However, there are no labels allocated for this prefix. This is because all of the routers, due to my personal configuration habits, are only allocating labels for host routes.

```

RP/0/0/CPU0:XRv4#show route 10.5.6.5
Routing entry for 10.5.6.0/24
  Known via "isis 24", distance 115, metric 10, type level-2
  Routing Descriptor Blocks
    24.6.14.6, from 24.0.0.6, via GigabitEthernet0/0/0/0.564
      Route metric is 10
  No advertising protos.

```

```

RP/0/0/CPU0:XRv4#show mpls ldp bindings 10.5.6.0/24
[no output]

```

We can see the error in the FIB as well. Only a single label is imposed and the FIB marks this as an “unresolved” entry. There is no possible way XRv4 will be able to forward traffic along this LSP.

```

RP/0/0/CPU0:XRv4#show cef vrf EIGRP 10.3.3.3
10.3.3.3/32, version 8379, internal 0x5000001 0x0 (ptr 0xa142d974) [1], 0x0
(0x0), 0x208 (0xa156d488)
  Prefix Len 32, traffic index 0, precedence n/a, priority 3
  via 10.5.6.5, 0 dependencies, recursive [flags 0x6000]
  path-idx 0 NHID 0x0 [0xa0f67254 0x0]
  recursion-via-/32
  next hop VRF - 'default', table - 0xe0000000
  unresolved
  labels imposed {5020}

```

I quickly disable the host-route label filters on all AS 24 routers. I could have been more elegant and used a prefix-list, but for brevity I simply allocate labels for all prefixes. The configuration removal is not shown, but instead I show that a valid LDP label has been bound to prefix 10.5.6.0/24 by all LDP peers. Since CSR6 is connected, it allocated implicit null, but the other routers allocate non-null labels as they are IGP-learned.

```

RP/0/0/CPU0:XRv4#show mpls ldp bindings 10.5.6.0/24
10.5.6.0/24, rev 38
  Local binding: label: 94003
  Remote bindings: (3 peers)
    Peer          Label

```

-----	-----
24.0.0.2:0	2100
24.0.0.6:0	ImpNull
24.0.0.7:0	7082

Despite the label resolution, the FIB remains dysfunctional. The single label stack is correct but CEF is still unable to forward packets. At this point, one would never know how to solve this problem without referencing Cisco's documentation on IOS XR. It clearly states that there must be a /32 route to the BGP next-hop for VPN routes. This is true whether the BGP route is iBGP or eBGP learned. The output hints at this issue since XRv4 claims it is trying to perform "recursion-via-/32" but the longest-match route to 10.5.6.5 is a /24.

```
RP/0/0/CPU0:XRv4#show cef vrf EIGRP 10.3.3.3
10.3.3.3/32, version 8379, internal 0x5000001 0x0 (ptr 0xa142d974) [1], 0x0
(0x0), 0x208 (0xa156d488)
Prefix Len 32, traffic index 0, precedence n/a, priority 3
via 10.5.6.5, 0 dependencies, recursive [flags 0x6000]
path-idx 0 NHID 0x0 [0xa0f67254 0x0]
recursion-via-/32
next hop VRF - 'default', table - 0xe0000000
unresolved
labels imposed {5020}
```

Solving this problem is a bit awkward. Next-hop-self would be the most obvious solution, but we already used that successfully in AS 13. Recall that XE routers have installed a connected /32 on their transit links for which there is an eBGP VPN peer. Both CSR6 and CSR7 have these peer host routes (somewhat similar to the result of PPP neighbor routes) on each transit link.

```
R6#show ip route connected | include ^C.*/32
C      10.5.6.5/32 is directly connected, GigabitEthernet2.556
C      10.6.11.11/32 is directly connected, GigabitEthernet2.561
C      24.0.0.6/32 is directly connected, Loopback0

R7#show ip route connected | include ^C.*/32
C      10.5.7.5/32 is directly connected, GigabitEthernet2.557
C      24.0.0.7/32 is directly connected, Loopback0
```

Rather than advertise the transit /24, CSR6 and CSR7 could advertise the peer /32 instead. This would satisfy the requirement that the BGP next-hop is a host route. On CSR6, this means removing the passive-interface configuration and adding new route-maps/prefix-lists for redistribution. On CSR7, it means simply adjusting the existing prefix-list. The other benefit is that we can re-add our label filter to only allocate labels for host routes (which is a good practice). The configuration for the label filtering is not shown, but it is enabled again.

! CSR6

```

ip prefix-list PL_R5R6 seq 5 permit 10.5.6.5/32
ip prefix-list PL_R6XRv1 seq 5 permit 10.6.11.11/32

route-map RM_CONN_TO_ISIS permit 10
  match ip address prefix-list PL_R6XRv1

route-map RM_CONN_TO_ISIS permit 20
  match ip address prefix-list PL_R5R6

router isis 24
  no passive-interface GigabitEthernet2.556
  no passive-interface GigabitEthernet2.561
  redistribute connected route-map RM_CONN_TO_ISIS

! CSR7
no ip prefix-list PL_R5R7 seq 5 permit 10.5.7.0/24
ip prefix-list PL_R5R7 seq 5 permit 10.5.7.5/32

```

We can verify this by checking the IS-IS LSPs as we did before to ensure the host routes are being advertised properly. Now, they have a length of 32 which satisfies XR's forwarding constraints.

```

R6#show isis database detail | section R6.00-00
R6.00-00          0x000000DB  0xFCB5          1176          0/0/0
[snip]
  IP Address:    24.0.0.6
  Metric: 0      IP 24.0.0.6/32
  Metric: 0      IP 10.5.6.5/32
  Metric: 0      IP 10.6.11.11/32
  IPv6 Address:  ::24:0:0:6
  Metric: 0      IPv6 (MT-IPv6) ::24:0:0:6/128

R7#show isis database detail | section R7.00-00
R7.00-00          * 0x000000D8  0x7019          1182          0/0/0
[snip]
  IP Address:    24.0.0.7
  Metric: 0      IP 24.0.0.7/32
  Metric: 0      IP 10.5.7.5/32
  IPv6 Address:  ::24:0:0:7
  Metric: 0      IPv6 (MT-IPv6) ::24:0:0:7/128

```

Interestingly, the label stack changes. Despite this being a “connected” route on CSR6, it is not a “local” route. As such, CSR6 cannot allocate a null-label for it. Recall that XRv4's VPN label for 10.3.3.3/32 was allocated by CSR5; we cannot expose this to CSR6 or else traffic will be dropped. CSR6 is not performing a swap of the VPN label since it did not change the BGP next-hop, so the top-most label allows CSR6 to pass the VPN label to CSR5 intact. If a null-label were allocated for this preferred, CSR6 would be indicating that it wants to see the next label in the stack, which would break the LSP.

```
RP/0/0/CPU0:XRv4#show mpls ldp bindings 10.5.6.5/32
10.5.6.5/32, rev 50
```

```
Local binding: label: 94018
```

```
Remote bindings: (3 peers)
```

Peer	Label
24.0.0.2:0	2103
24.0.0.6:0	6035
24.0.0.7:0	7087

Most importantly, XRv4's FIB entry for this VPN route is now fully operable. We see two labels in the stack and a valid next-hop. The entry is no longer flagged as "unresolved". XRv4 now passes traffic to CSR6 using label stack {6035 5020}.

```
RP/0/0/CPU0:XRv4#show cef vrf EIGRP 10.3.3.3
10.3.3.3/32, version 8533, internal 0x5000001 0x0 (ptr 0xa142ef74) [1], 0x0
(0x0), 0x208 (0xa156d3e8)
```

```
Prefix Len 32, traffic index 0, precedence n/a, priority 3
```

```
via 10.5.6.5, 5 dependencies, recursive [flags 0x6000]
```

```
path-idx 0 NHID 0x0 [0xa15d5f74 0x0]
```

```
recursion-via-/32
```

```
next hop VRF - 'default', table - 0xe0000000
```

```
next hop 10.5.6.5 via 94018/0/21
```

```
next hop 24.6.14.6/32 Gi0/0/0/0.564 labels imposed {6035 5020}
```

When CSR6 receives this flow, it pops label 6035 to reveal label 5020 to CSR5. There is no BGP VPN label activity on CSR6 whatsoever.

```
R6#show mpls forwarding-table labels 6035
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
6035	Pop Label	10.5.6.5/32	2028	Gi2.556	10.5.6.5

CSR5 receives this label and swaps it to XRv2's local label for the prefix, which is 92002. The BGP next-hop has changed to XRv2 which is why the label swap must occur.

```
R5#show bgp vpnv4 unicast rd 13:3 10.3.3.3/32 | begin Local
```

```
[snip]
```

```
Local
```

```
13.0.0.12 (metric 3) (via default) from 13.0.0.12 (13.0.0.12)
```

```
Origin incomplete, metric 10880, localpref 100, valid, internal, best
```

```
Extended Community: RT:13:3 Cost:pre-bestpath:128:10880 0x8800:32768:0
```

```
0x8801:3:288 0x8802:65281:2560 0x8803:1:1500 0x8806:0:167971843
```

```
Connector Attribute: count=1
```

```
type 1 len 12 value 13:3:13.0.0.12
```

```
mpls labels in/out 5020/92002
```

```
rx pathid: 0, tx pathid: 0x0
```

CSR5 also adds a transport label to get traffic to XRv2. The route to 13.0.0.12/32 is IGP-learned via CSR8, so the appropriate LDP label is used. The label stack becomes {8000 92002}.

```
R5#show ip route 13.0.0.12
Routing entry for 13.0.0.12/32
  Known via "ospf 13", distance 110, metric 3, type intra area
  Last update from 13.5.8.8 on GigabitEthernet2.558, 23:00:03 ago
  Routing Descriptor Blocks:
  * 13.5.8.8, from 13.0.0.12, 23:00:03 ago, via GigabitEthernet2.558
    Route metric is 3, traffic share count is 1
```

```
R5#show mpls ldp bindings 13.0.0.12 32 neighbor 13.0.0.8
  lib entry: 13.0.0.12/32, rev 4
    remote binding: lsr: 13.0.0.8:0, label: 8000
```

If you casually look at the FIB to determine this, only the label swap is shown. Use the “detail” option to see the follow-on push operation.

```
R5#show mpls forwarding-table labels 5020 detail
Local      Outgoing    Prefix          Bytes Label    Outgoing    Next Hop
Label      Label       or Tunnel Id   Switched       interface
5020       92002      13:3:10.3.3.3/32 0              Gi2.558      13.5.8.8
MAC/Encaps=18/26, MRU=1496, Label Stack{8000 92002}
005056A9FB1C005056A9DC6381000DE68847 01F4000016762000
No output feature configured
```

CSR8 performs PHP to expose label 92002 to XRv2. XRv2 removes all labels and forwards the packet to CSR3. These two operations are basic L3VPN as the inter-AS portion concludes at CSR5.

```
R8#show mpls forwarding-table labels 8000
Local      Outgoing    Prefix          Bytes Label    Outgoing    Next Hop
Label      Label       or Tunnel Id   Switched       interface
8000       Pop Label   13.0.0.12/32   5328608       Gi2.582     13.8.12.12

RP/0/0/CPU0:XRv2#show mpls forwarding vrf EIGRP prefix 10.3.3.3/32
Local  Outgoing    Prefix          Outgoing    Next Hop        Bytes
Label  Label       or ID           Interface    Interface       Switched
-----
92002  Unlabelled  10.3.3.3/32 [V]  Gi0/0/0/0.532 10.3.12.3      204920
```

The LSP is now operational. Using traceroute from CSR3, we confirm the label stack we verified first. Technically, this is considered 3 separate LSPs since the VPN label changes 3 times. The first LSP connects XRv2 and CSR5, the second connects CSR5 and CSR6 between ASes, and the third connects CSR6 to XRv4. The three VPN labels are highlighted; these are BGP-swap actions.

```

R3#traceroute 10.13.13.13 source 10.3.3.3
Type escape sequence to abort.
Tracing the route to 10.13.13.13
VRF info: (vrf in name/id, vrf out name/id)
 1 10.3.12.12 4 msec 2 msec 2 msec
 2 13.8.12.8 [MPLS: Labels 8002/5032 Exp 0] 11 msec 8 msec 7 msec
 3 13.5.8.5 [MPLS: Label 5032 Exp 0] 26 msec 30 msec 31 msec
 4 10.5.6.6 [MPLS: Label 6030 Exp 0] 30 msec 31 msec 32 msec
 5 24.6.14.14 [MPLS: Label 94006 Exp 0] 18 msec 21 msec 21 msec
 6 10.13.14.13 20 msec 15 msec 15 msec

```

The return path consists of only two LSPs since the BGP label was swapped once, not twice. Since CSR6, as the egress ASBR, did not change the BGP next-hop, it does not swap the VPN label. Thus, the two LSPs connect XRv4 to CSR5, and CSR5 to XRv2. Notwithstanding MPLS-TR, all LSPs are unidirectional and can be non-congruent. This means each AS can treat the eBGP next-hops according to local policies, provided reachability is maintained.

```

RP/0/0/CPU0:XRv3#traceroute 10.3.3.3 source 10.13.13.13
Type escape sequence to abort.
Tracing the route to 10.3.3.3

 1 10.13.14.14 0 msec 0 msec 0 msec
 2 24.6.14.6 [MPLS: Labels 6035/5020 Exp 0] 0 msec 0 msec 0 msec
 3 10.5.6.5 [MPLS: Label 5020 Exp 0] 0 msec 0 msec 0 msec
 4 13.5.8.8 [MPLS: Labels 8000/92002 Exp 0] 0 msec 0 msec 9 msec
 5 13.8.12.12 [MPLS: Label 92002 Exp 0] 0 msec 0 msec 0 msec
 6 10.3.12.3 0 msec 0 msec 0 msec

```

Despite this pair of LSPs being functional, there is still an issue lurking in the network. To demonstrate it, we will use VPNv6 prefixes exchanged between ASes. Currently, XRv2 prefers CSR5 over XRv1 as the egress ASBR towards ::10:3:3:3/128 since it has a lower BGP RID.

```

RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast rd 24:3 ::10:13:13:13/128 | begin 24,
24, (Received from a RR-client)
 13.0.0.5 (metric 3) from 13.0.0.5 (13.0.0.5)
    Received Label 5024
    Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, not-in-vrf
    Received Path ID 0, Local Path ID 1, version 339
    Extended community: EIGRP route-info:0x8000:0 EIGRP AD:3:282 EIGRP
RHB:255:1:2560 EIGRP LM:0x0:1:1500 EIGRP VRR:0x0:13.13.13.10 RT:24:3
    Path #2: Received by speaker 0
    Not advertised to any peer
24, (Received from a RR-client)
 13.0.0.11 (metric 3) from 13.0.0.11 (13.0.0.11)
    Received Label 91035

```

```

    Origin incomplete, localpref 100, valid, internal, import-candidate,
not-in-vrf
    Received Path ID 0, Local Path ID 0, version 0
    Extended community: EIGRP route-info:0x8000:0 EIGRP AD:3:282 EIGRP
RHB:255:1:2560 EIGRP LM:0x0:1:1500 EIGRP VRR:0x0:13.13.13.10 RT:24:3

```

We can adjust this in many ways. I will use inbound local-preference on XRv1 to increase its preference as an egress ASBR for that VPNv6 prefix only. I use a parameterized RPL in case I need to reuse it later.

```

! XRv1
prefix-set PS_XRV3_V6
::10:13:13:13/128
end-set

route-policy RPL_SET_LOCAL_PREF($PS, $LPREF)
  if destination in $PS then
    set local-preference $LPREF
  else
    pass
  endif
end-policy

router bgp 13
neighbor 10.6.11.6
address-family vpnv6 unicast
  route-policy RPL_SET_LOCAL_PREF(PS_XRV3_V6, 200) in

```

Now, the RR only sees one route because CSR5 has accepted this local-preference 200 prefix as best. Since XRv2 is also the PE, it cares about the received label which is 91035. This is the first label added to the stack.

```

RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast rd 24:3 ::10:13:13:13/128 | begin 24,
24, (Received from a RR-client)
13.0.0.11 (metric 3) from 13.0.0.11 (13.0.0.11)
  Received Label 91035
    Origin incomplete, localpref 200, valid, internal, best, group-best,
import-candidate, not-in-vrf
    Received Path ID 0, Local Path ID 1, version 375
    Extended community: EIGRP route-info:0x8000:0 EIGRP AD:3:282 EIGRP
RHB:255:1:2560 EIGRP LM:0x0:1:1500 EIGRP VRR:0x0:13.13.13.10 RT:24:3

```

A transport label is also added since XRv2 routes via CSr8 to reach XRv1. The route is IGP-learned so CSR8's LDP label of 8001 is pushed atop the stack. The label stack is now {8001 91035}.

```

RP/0/0/CPU0:XRv2#show route 13.0.0.11
Routing entry for 13.0.0.11/32
  Known via "ospf 13", distance 110, metric 3, type intra area

```

Routing Descriptor Blocks

13.8.12.8, from 13.0.0.11, via GigabitEthernet0/0/0/0.582

Route metric is 3

No advertising protos.

```
RP/0/0/CPU0:XRv2#show mpls ldp bindings 13.0.0.11/32 neighbor 13.0.0.8
13.0.0.11/32, rev 14
```

Local binding: label: 92006

Remote bindings: (2 peers)

Peer	Label
13.0.0.8:0	8001

CSR8 is an ordinary P-router and performs PHP to expose label 91035 to XRv1.

```
R8#show mpls forwarding-table labels 8001
```

Local	Outgoing	Prefix	Bytes	Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched		interface	
8001	Pop Label	13.0.0.11/32	1667433		Gi2.581	13.8.11.11

XRv1's LFIB has a seemingly valid entry for this VPN prefix with a next-hop of 10.5.6.5. We see 0 bytes being label switched, which is an indication of a potential issue. The FIB does not mark the entry as unresolved; in fact, it shows as a /32 local adjacency since IPv4 ARP resolution occurred for this IP address. The problem with the FIB entry is that it makes no mention of MPLS labels or MPLS forwarding.

```
RP/0/0/CPU0:Xshow mpls forwarding labels 91035
```

Local	Outgoing	Prefix	Outgoing	Next Hop	Bytes
Label	Label	or ID	Interface		Switched
91035	6014	24:3:::10:13:13:13/128	\	10.6.11.6	0

```
RP/0/0/CPU0:XRv1#show cef 10.6.11.6
```

10.6.11.6/32, version 0, internal 0x1020001 0x0 (ptr 0xa14487f4) [1], 0x0 (0xa1413ab8), 0x0 (0x0)

local adjacency 10.6.11.6

Prefix Len 32, traffic index 0, Adjacency-prefix, precedence n/a, priority 15

via 10.6.11.6, GigabitEthernet0/0/0/0.561, 3 dependencies, weight 0, class 0 [flags 0x0]

path-idx 0 NHID 0x0 [0xa10853a0 0x0]

next hop 10.6.11.6

local adjacency

However, the FIB has derived this /32 based on ARP, not based on the RIB. The RIB only shows a /24 for this prefix. Again, there is not a good way to troubleshoot this other than to know that XR requires a /32 route to the BGP next-hop, period.


```
RP/0/0/CPU0:XRv1#show route 10.6.11.6
Routing entry for 10.6.11.0/24
  Known via "connected", distance 0, metric 0 (connected)
  Routing Descriptor Blocks
    directly connected, via GigabitEthernet0/0/0/0.561
      Route metric is 0
  No advertising protos.
```

To repair this, we can configure a static /32 to the BGP peer. There is no rule that says the host route must be appear as “connected” in the RIB, but it must exist. Since this is a host-route, specifying a next-hop makes no sense, so we specify the outgoing interface. A “static” route will meet XR’s forwarding requirement.

```
! XRv1
router static
  address-family ipv4 unicast
    10.6.11.6/32 GigabitEthernet0/0/0/0.561
```

Despite us fixing the problem, XR gives us a warning message about using this technique. When the prefix length is shorter than /32, this will cause the router proxy-ARP for all destinations matching that network, so the log message is well-founded and sensible. It is interesting to see it displayed when it is actually fixing an XR limitation in the first place.

```
! XRv11
ipv4_static[1040]: %ROUTING-IP_STATIC-4-CONFIG_NEXTHOP_ETHER_INTERFACE :
Route for 10.6.11.6 is configured via ethernet interface without nexthop,
Please check if this is intended
```

Now, the RIB has a /32 and the FIB reports some MPLS label activity. This is indicative of a working system. Also notice that because this is a host route, LDP allocates a label for it. The label is useless since XRv1 changes the BGP next-hop, but if it didn’t, we could also use the redistribution technique used in AS 24 provided an LDP label was allocated for the transit link (and it is).

```
RP/0/0/CPU0:XRv1#show route 10.6.11.6
Routing entry for 10.6.11.6/32
  Known via "static", distance 1, metric 0 (connected)
  Routing Descriptor Blocks
    directly connected, via GigabitEthernet0/0/0/0.561
      Route metric is 0
  No advertising protos.
```

```
RP/0/0/CPU0:XRv1#show cef 10.6.11.6
10.6.11.6/32, version 554, attached, internal 0x1020041 0x0 (ptr 0xa14487f4)
[1], 0x0 (0xa1413ab8), 0xa20 (0xa156d3c0)
local adjacency 10.6.11.6
```

```

Prefix Len 32, traffic index 0, Adjacency-prefix, precedence n/a, priority
15
  via GigabitEthernet0/0/0/0.561, 3 dependencies, weight 0, class 0 [flags
0x8]
    path-idx 0 NHID 0x0 [0xa10853a0 0xa10854f0]
    local adjacency
      local label 91009      labels imposed {ImplNull}

```

Without tracing the entire LSP, we perform a ping test from XRv3. After this is done, we can see 520 bytes of traffic matching this LFIB entry which we didn't see initially. This accounts for 5 104-byte packets, which includes the VPN label for inter-AS operations.

```

RP/0/0/CPU0:XRv3#ping ::10:3:3:3 source ::10:13:13:13
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to ::10:3:3:3, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 19/29/69 ms

```

```

RP/0/0/CPU0:XRv1#show mpls forwarding labels 91035

```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
91035	6014	24:3::10:13:13:13/128	Gi0/0/0/0.561	10.6.11.6	520

Now that the entire transport network has been correctly configured, we will establish the inter-AS sham-link using option B. The configuration is identical to that used in option A since the sham-link endpoints have not changed; VPNv4/v6 eBGP peers are already exchanged extended communities as well. We also know that the OSPF special communities are transitive so building the sham-links should be no issue. In fact, with the network properly verified, the sham-links just “come up”.

```

R8#show ospfv3 vrf OSPF sham-links | include ^Sham
Sham Link OSPFv3_SL0 to address FD00::2 is up
Sham Link OSPFv3_SL1 to address FD00::2 is up

```

Checking CSR8, we can see the local and remote sham-link endpoints inside of VRF OSPF as expected. Once these two addresses can exchange targeted OSPF hellos, the sham-link can form.

```

R8#show bgp vpnv6 unicast vrf OSPF | include FD00::
*>i FD00::2/128      ::FFFF:13.0.0.5      0      100      0 24 i
*>  FD00::8/128      ::                    0      32768 i

```

To show the transitivity of the OSPF extended communities, we check CSR5 for the route to CSR9's loopback. From both CSR6 and CSR7, it learns the route with the communities intact, which allows OSPF to treat the MPLS network as an area 0 link.

```

R5#show bgp vpnv6 unicast rd 24:2 ::10:9:9:9/128 | begin 24$
[snip]
24
::FFFF:10.5.7.7 (via default) from 10.5.7.7 (24.0.0.7)
  Origin incomplete, localpref 100, valid, external
  Extended Community: RT:24:2 OSPF ROUTER ID:10.2.9.2:0
    OSPF RT:0.0.0.0:2:0
  mpls labels in/out 5028/7052
  rx pathid: 0, tx pathid: 0
Refresh Epoch 3
24
::FFFF:10.5.6.6 (via default) from 10.5.6.6 (24.0.0.6)
  Origin incomplete, localpref 100, valid, external, best
  Extended Community: RT:24:2 OSPF ROUTER ID:10.2.9.2:0
    OSPF RT:0.0.0.0:2:0
  mpls labels in/out 5028/6011
  rx pathid: 0, tx pathid: 0x0

```

To test its operation, I perform a quick traceroute from CSR4 to CSR9 and CSR9 to CSR10. The first output shows the OSPF-to-OSPF communication over MPLS, which is preferred over the backdoor. Notice that 3 separate LSPs (3 VPN labels) are required since the AS 13 ASBRs use iBGP next-hop-self. The second shows OSPF-to-central-services over the inter-AS boundary, which is the result out of RT policy. Only 2 LSPs (2 VPN labels) are required since AS 24 ASBRs redistribute the BGP next-hops into IGP.

```

R4#traceroute ipv6
Target IPv6 address: ::10:9:9:9
Source address: ::10:4:4:4
[snip]
Tracing the route to ::10:9:9:9
 1 FD00:10:4:8::8 11 msec 4 msec 4 msec
 2 ::FFFF:13.5.8.5 [MPLS: Label 5028 Exp 0] 9 msec 7 msec 10 msec
 3 ::FFFF:10.5.6.6 [MPLS: Label 6011 Exp 0] 18 msec 22 msec 23 msec
 4 2024:24:6:14::14 [MPLS: Labels 94009/2010 Exp 0] 22 msec 29 msec 23 msec
 5 FD00:10:2:9::2 [MPLS: Label 2010 Exp 0] 22 msec 23 msec 22 msec
 6 FD00:10:2:9::9 23 msec 15 msec 16 msec

```

```

R9#traceroute ipv6
Target IPv6 address: ::110:0:0:2
Source address: ::10:9:9:9
[snip]
Tracing the route to ::110:0:0:2
 1 FD00:10:2:9::2 4 msec 3 msec 4 msec
 2 2024:24:2:14::14 [MPLS: Labels 94018/5012 Exp 0] 7 msec 8 msec 15 msec
 3 ::FFFF:24.6.14.6 [MPLS: Labels 6035/5012 Exp 0] 35 msec 33 msec 34 msec
 4 ::FFFF:10.5.6.5 [MPLS: Label 5012 Exp 0] 34 msec 33 msec 32 msec
 5 FD00:10:8:10::8 [MPLS: Label 8012 Exp 0] 23 msec 21 msec 21 msec

```

```
6 FD00:10:8:10::10 22 msec 15 msec 15 msec
```

One final note about this design is that there is no exchange of IGP networks between ASes. Similar to option A, only VPN routes are exchanged. Even though the VPNv4/v6 peers exist in the global table, the IGP routes are not leaked. This keeps network integrity between the ASes and is an added benefit of option B.

8.4.2.2 L2VPN

L2VPN service over option B is very complex. Unlike option A, we cannot simply terminate the L2VPN on the ASBRs and pipe the frames across a transit link. The expectation is that traffic is label-switched for the entire path. Unlike option C (discussed later), we cannot directly peer the PEs since IGP routes are not leaked across AS boundaries. The solution involves multi-segment PWs (MSPW). This topic is covered in detail in a dedicated section, but the idea is to create many PWs that are stitched together. Similar to having or 2 or 3 different VPN labels for option B L3VPN, this design requires 3 stitched PWs for inter-AS over option B. Statically configuring these PWs is shown in the dedicated section elsewhere in this book; for variety, this test will use BGP auto-discovery between AS boundaries. Before we configure any VPLS instances, we need to build the BGP infrastructure. There are additional limitations on L2VPN: specifically, we must use BGP next-hop-self and we must retain all routes on the ASBR. The alternatives demonstrated in L3VPN, such as RR configuration and local VRF configuration, are not applicable here. Technically, the RR approach could work but Cisco does not recommend it. CSR2 and CSR6 peer directly in AS 24 while XRv2 is used as an RR in AS 13. On CSR5 and CSR6 are L2VPN-capable ASBRs; this is a limitation of the feature as only one inter-AS link can exist. We also must configure an eBGP L2VPN VPLS session between CSR5 and CSR6.

```
! CSR2
router bgp 24
  address-family l2vpn vpls
    neighbor 24.0.0.6 activate

! CSR6
router bgp 24
  address-family l2vpn vpls
    no bgp default route-target filter
    neighbor 10.5.6.5 activate
    neighbor 24.0.0.2 activate
    neighbor 24.0.0.2 next-hop-self
```

The configuration in AS 13 is very similar. The prefix-length size increase is an XR interoperability option and is described in the dedicated L2VPN section. So far, the configuration is very similar to L3VPN except using a different AFI. The same RT retention and next-hop-self design goals are in effect as those are fundamental tenants of option B.

```
! CSR8
router bgp 13
```

```

address-family l2vpn vpls
  neighbor 13.0.0.12 activate
  neighbor 13.0.0.12 prefix-length-size 2

! CSR5
router bgp 13
  address-family l2vpn vpls
    no bgp default route-target filter
    neighbor 10.5.6.6 activate
    neighbor 13.0.0.12 activate
    neighbor 13.0.0.12 prefix-length-size 2
    neighbor 13.0.0.12 next-hop-self

! XRv2
router bgp 13
  af-group L2VPN address-family l2vpn vpls-vpws
    route-reflector-client
    Signalling bgp disable

neighbor 13.0.0.5
  address-family l2vpn vpls-vpws
  use af-group L2VPN

neighbor 13.0.0.8
  address-family l2vpn vpls-vpws
  use af-group L2VPN

```

For brevity, I check the BGP peers XRv2 and CSR6. Assuming these routers see all their peers, we can be confident all routers have L2VPN VPLS BGP configured properly.

```

RP/0/0/CPU0:XRv2#show bgp l2vpn vpls summary | begin ^Neighbor
Neighbor      Spk    AS  MsgRcvd  MsgSent    TblVer  InQ  OutQ  Up/Down  St/PfxRcd
13.0.0.5      0     13   18326    17397      25     0    0    1d01h    0
13.0.0.8      0     13   18100    17389      25     0    0    1d01h    0

R6#show bgp l2vpn vpls all summary | begin ^Neighbor
Neighbor      V      AS  MsgRcvd  MsgSent    TblVer  InQ  OutQ  Up/Down  State/PfxRcd
10.5.6.5      4     13    72       74         56     0    0  00:27:51    0
24.0.0.2      4     24   363      320         56     0    0  00:43:48    0

```

Next, we will configure the L2VPNs and bind them to the bridge-domain. These tie the VFI to the ACs. The VPN configurations are incomplete but we can at least verify BGP advertisement by configuring a VFI. I intentionally configure different VPN IDs because these might be intra-AS VPLS instances as well. We can pretend there are other clients within each AS that all agree on the VPN ID.

```

! CSR8
l2vpn vfi context VPLS
  vpn id 13
  autodiscovery bgp signaling ldp template TMP_VPLS

```

```

! CSR2
l2vpn vfi context VPLS
  vpn id 24
  autodiscovery bgp signaling ldp template TMP_VPLS

```

Additionally, in order for this design to work, we must enable PW routing. This allows PWs to be stitched together. The TPE tie-breaker is discussed more later.

```

! All ASBRs and PEs
l2vpn
  logging pseudowire status
  pseudowire routing
  terminating-pe tie-breaker

```

Since we only expect to see 2 routes, we look at all the details for all routes across all RDs. CSR6 has both routes which is a good sign that the route exchange is occurring. We can immediately see a mismatch in the route-target values and L2VPN attachment group ID (AGI). Both of them are automatically derived from the BGP ASN and VPN IN, which explains the values RT:13:13 and RT:24:24. The ASBRs are able to retain these prefixes because, as option B requires, the default RT filter has been disabled.

```

R6#show bgp l2vpn vpls all detail
BGP routing table entry for 13:13:13.0.0.8/96, version 58
  Paths: (1 available, best #1, table L2VPN-VPLS-BGP-Table)
  Advertised to update-groups:
    5
  Refresh Epoch 2
  13
    10.5.6.5 from 10.5.6.5 (13.0.0.5)
      Origin incomplete, localpref 100, valid, external, best, AGI version(0)
      Extended Community: RT:13:13 L2VPN AGI:13:13
      mpls labels in/out exp-null/exp-null
      rx pathid: 0, tx pathid: 0x0
BGP routing table entry for 24:24:24.0.0.2/96, version 59
  Paths: (1 available, best #1, table L2VPN-VPLS-BGP-Table)
  Advertised to update-groups:
    4
  Refresh Epoch 5
  Local
    24.0.0.2 (metric 20) from 24.0.0.2 (24.0.0.2)
      Origin incomplete, metric 0, localpref 100, valid, internal, best, AGI
version(0)
      Extended Community: RT:24:24 L2VPN AGI:24:24
      mpls labels in/out exp-null/exp-null
      rx pathid: 0, tx pathid: 0x0

```

To clean this up, I will use a different set of RTs to achieve inter-AS connectivity. We can leave “auto-route-target” enabled because I am pretending other members of this VPN instance exist within the VRF. Thus, I will add additional RT import statements to each VFI instance. To minimize changes, I configure CSR2 to import RT:13:13 and CSR8 to import RT:24:24. No new RTs need to be defined in this case and no new RTs are attached to the existing routes.

```
! CSR8
l2vpn vfi context VPLS
  autodiscovery bgp signaling ldp template TMP_VPLS
  route-target import 24:24

! CSR2
l2vpn vfi context VPLS
  autodiscovery bgp signaling ldp template TMP_VPLS
  route-target import 13:13

R2#show l2vpn vfi name VPLS
Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No
VFI name: VPLS, state: up, type: multipoint, signaling: LDP
VPN ID: 24, VPLS-ID: 24:24
RD: 24:24, RT: 24:24, import 13:13
Bridge-Domain 3 attachment circuits:
Pseudo-port interface: pseudowire100009
Interface          Peer Address      VC ID      Discovered Router ID  S
```

```
R8#show l2vpn vfi name VPLS
Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No
VFI name: VPLS, state: up, type: multipoint, signaling: LDP
VPN ID: 13, VPLS-ID: 13:13
RD: 13:13, RT: 13:13, import 24:24
Bridge-Domain 3 attachment circuits:
Pseudo-port interface: pseudowire100014
Interface          Peer Address      VC ID      Discovered Router ID  S
```

Despite these changes, the L2VPN routes are still being rejected. The debug output tells us the extended community is not supported, and normally this indicates an RT issue. However, the AGI is also an extended community. This is specifically used to determine if two VFIs are in the same VPLS instance. The BGP output above clearly shows different values.

```
R2#debug bgp l2vpn vpls updates in
BGP updates debugging is on (inbound) for address family: L2VPN Vpls

BGP(9): 24.0.0.6 rcvd 13:13:13.0.0.8/96 -- DENIED due to: extended community
not supported;
```

We adjust this VPLS ID on both CSR2 and CSR8 so that they are in the same VPN again. Other intra-AS clients would also have to agree on this value in order for VPLS to operate as well.

```

! CSR2 and CSR8
l2vpn vfi context VPLS
  autodiscovery bgp signaling ldp template TMP_VPLS
  vpls-id 13:24

```

Checking CSR2, we can see that the remote L2VPN route from AS 13 was accepted. The route-targets still differ, but since those are being imported on opposing VFIs, it doesn't matter.

```

R2#show bgp l2vpn vpls all detail
BGP routing table entry for 13:13:13.0.0.8/96, version 64
  Paths: (1 available, best #1, table L2VPN-VPLS-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 10
  13
    24.0.0.6 (metric 20) from 24.0.0.6 (24.0.0.6)
      Origin incomplete, metric 0, localpref 100, valid, internal, best, AGI
version(301989894)
      Extended Community: RT:13:13 L2VPN AGI:13:24
      mpls labels in/out exp-null/exp-null
      rx pathid: 0, tx pathid: 0x0
BGP routing table entry for 24:24:24.0.0.2/96, version 63
  Paths: (1 available, best #1, table L2VPN-VPLS-BGP-Table)
  Advertised to update-groups:
    3
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (24.0.0.2)
      Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best, AGI version(0)
      Extended Community: RT:24:24 L2VPN AGI:13:24
      mpls labels in/out exp-null/exp-null
      rx pathid: 0, tx pathid: 0x0

```

Checking CSR2, we can see that it sees 13.0.0.2 as the remote PW endpoints. Of course, we know these two IP addresses don't have reachability to one another. Checking the details of the PW, we can see that the BGP next-hop is identified as 24.0.0.6. This means that CSR2 will try to build the PW to this endpoint knowing that the VPLS is inter-AS.

```

R2#show l2vpn atom vc

```

Interface	Peer ID	VC ID	Service		Status
			Type	Name	
pw100010	13.0.0.8	24	vfi	VPLS	DOWN

```

R2#show l2vpn atom vc detail
pseudowire100010 is up, VC status is down PW type: Ethernet

```



```
Create time: 00:04:36, last status change time: 00:01:50
  Last label FSM state change time: 00:01:50
Destination address: 13.0.0.8 VC ID: 24
Next hop PE address: 24.0.0.6
[snip]
```

One of the challenges of dealing with MSPW is that if one of the circuit legs is down, the entire MSPW is down. Examining the details on CSR6, we discover LDP trying to establish a session across the ASBR link. This makes sense as the VPLS is LDP-signaled, yet LDP is not enabled on the transit link.

```
R6#show l2vpn atom vc destination 13.0.0.8 detail
pseudowire100020 is up, VC status is down PW type: Ethernet
  Create time: 00:04:58, last status change time: 4d02h
  Last label FSM state change time: 00:04:58
Destination address: 13.0.0.8 VC ID: 1001
Next hop PE address: 10.5.6.5
  Output interface: none, imposed label stack {}
  Preferred path: not configured
  Default path: no route
  No adjacency
Member of xconnect service mpls 24.0.0.2:1001
  Associated member pw100021 is up, status is down
  Interworking type is Like2Like
  Service id: 0x8a00000a
Signaling protocol: LDP, peer unknown
  Targeted Hello: 10.5.6.6(from BGP) -> 10.5.6.5, LDP is DOWN, no binding
[snip]
```

The TPE command we entered earlier is a requirement for auto-discovered VPLS over option B. A terminating PE is one of the remote PEs, not an ASBR, that terminates an MSPW. By default, all PEs are in the active state in terms of tLDP sessions. One of them MUST be in the passive state in order for this feature to work. The device with the numerically higher L2VPN RID (based on the highest loopback by default) is the active device. Before continuing, we will enable passive reception of tLDP sessions on all devices except the active TPE. Looking at the XC details below, we see that CSR2 has a higher L2VPN RID and CSR8 is marked as passive.

```
R8#show xconnect rib detail
Local Router ID: 13.0.0.8
VPLS-ID: 13:24, Target ID: 24.0.0.2
  iBGP Peer
  Next-Hop: 13.0.0.5
  Hello-Source: 13.0.0.8
  Route-Target: 24:24
  Incoming RD: 24:24
  Forwarder: VFI VPLS
  Provisioned: No
  Passive: Yes
```

NLRI handle: BD000005

```
R2#show xconnect rib detail
```

```
Local Router ID: 24.0.0.2
```

```
VPLS-ID: 13:24, Target ID: 13.0.0.8
```

```
iBGP Peer
```

```
Next-Hop: 24.0.0.6
```

```
Hello-Source: 24.0.0.2
```

```
Route-Target: 13:13
```

```
Incoming RD: 13:13
```

```
Forwarder: VFI VPLS
```

```
Provisioned: Yes
```

```
NLRI handle: 38000004
```

As a result of this, we configure CSR8, CSR5, and CSR6 to accept tLDP sessions. We could also configure CSR2, but it technically is not required.

```
! CSR8, CSR5, and CSR6
mpls ldp discovery targeted-hello accept
```

Now, we can see LDP trying to establish a session over the transit link with targeted hellos, but with LDP disabled, a peer can never form. CSR6 can communicate directionally with CSR2, which shows good intra-AS communications. CSR5 claims it has no route to CSR6's loopback, which is its LDP ID. This is true, especially with LDP being totally disabled on the link. Notice that CSR6 is active for this session and CSR5 is passive; the session was initiated from CSR2 and the active-ness of a session is transitive from one TPE to the other.

```
R6#show mpls ldp discovery | begin Target
```

```
Targeted Hellos:
```

```
10.5.6.6 -> 10.5.6.5 (ldp): active, xmit
```

```
24.0.0.6 -> 24.0.0.2 (ldp): active/passive, xmit/recv
```

```
LDP Id: 24.0.0.2:0
```

```
R5#show mpls ldp discovery | begin Target
```

```
Targeted Hellos:
```

```
10.5.6.5 -> 10.5.6.6 (ldp): passive, xmit/recv
```

```
LDP Id: 24.0.0.6:0; no route
```

First, let's enable LDP on the link and see if it makes a difference. So far, it doesn't. We would still have the problem of CSR5 not having a route to CSR6's loopback, and vice versa.

```
! CSR5 and CSR6
interface GigabitEthernet2.556
mpls ip
```

The most obvious fix for this problem is some patchwork with static routes, or maybe IGP/BGP. This goes against the spirit of option B, so I will use a more elegant solution. We can configure LDP to use a different TCP transport address on a per-interface basis which is very useful in situations like this. I configure both CSR5 and CSR6 to use their connected interface addresses for the TCP session. This would apply to any neighbors discovered on that interface.

```
! CSR5 and CSR6
interface GigabitEthernet2.556
 mpls ldp discovery transport-address interface
```

The PW immediately comes up. While CSR5 still has no route to CSR6's loopback, the LDP neighbor forms as expected. The active/passive monikers seem to disappear as the MSPW is formed since the active formation of the MSPW occurs only during setup.

```
R5#show mpls ldp discovery | begin Targeted
Targeted Hellos:
  10.5.6.5 -> 10.5.6.6 (ldp): active/passive, xmit/recv
    LDP Id: 24.0.0.6:0; no route
  13.0.0.5 -> 13.0.0.8 (ldp): active/passive, xmit/recv
    LDP Id: 13.0.0.8:0
```

Checking the PW details on CSR2, we can see the PW is operational. The two-label stack shows a PW label (tLDP) and a transport label (LDP) in use. The targeted hellos are directed at the next-hop PE, not the target, as this MSPW is smart enough to know that there isn't reachability between the endpoints. When the BGP next-hop changes, rather than swap a label as in L3VPN, a PW is terminated. This ensures that inter-AS connectivity can work when BGP adjusts next-hops.

```
R2#show l2vpn atom vc vcid 24 detail
pseudowire100010 is up, VC status is up PW type: Ethernet
  Create time: 00:28:48, last status change time: 00:04:13
  Last label FSM state change time: 00:04:13
  Destination address: 13.0.0.8 VC ID: 24
  Next hop PE address: 24.0.0.6
    Output interface: Gi2.524, imposed label stack {94008 6073}
    Preferred path: not configured
    Default path: active
    Next hop: 24.2.14.14
  Member of vfi service VPLS
    Bridge-Domain id: 3
    Service id: 0xbf000003
  Signaling protocol: LDP, peer 24.0.0.6:0 up
  Targeted Hello: 24.0.0.2 (LDP Id) -> 24.0.0.6, LDP is UP
  [snip]
```

MPLS OAM is very useful to verify the integrity of the MSPW. Rather than trace intra-AS LSPs again, we will use this tool. Since we know there are 3 segments, we can use traceroute to reveal all of the PW

labels in use across the different PWs. We cannot use ordinary MPLS or IP-based traceroutes since we don't have connectivity between AS loopbacks. The customers are in a L2VPN and are totally unaware of any MPLS presence. OAM also reveals the PE/ASBR devices along the path that are making label adjustments. CSR2 uses label 6073 along the PW to CSR6. CSR6 swaps it for label 5040 towards CSR5, and CSR5 swaps it for label 8015 towards CSR8.

```
R2#traceroute mpls pseudowire 13.0.0.8 24 segment 3
Tracing MS-PW segments within range [1-3] peer address 13.0.0.8 and timeout 2
seconds
[snip]
```

```
Type escape sequence to abort.
L 1 24.6.14.6 9 ms [Labels: 6073 Exp: 0]
    local 24.0.0.2 remote 13.0.0.8 vc id 24

L 2 10.5.6.5 7 ms [Labels: 5040 Exp: 0]
    local 24.0.0.6 remote 13.0.0.5 vc id 1001

! 3 13.5.8.8 6 ms [Labels: 8015 Exp: 0]
    local 13.0.0.5 remote 13.0.0.8 vc id 1001
```

MPLS ping gives us useful results as well. The capital 'L' means that the traffic would have used a labeled path, but expired in transit. It is rare to see this in a ping message, but if we don't specify the proper number of segments in the MSPW, we will see this. This can be valuable for testing a certain number of segments in the MSPW if necessary.

```
R2#ping mpls pseudowire 13.0.0.8 24 segment 1
Sending 5, 72-byte MPLS Echos to 13.0.0.8,
    timeout is 2 seconds, send interval is 0 msec:
[snip]
Type escape sequence to abort.
LLLLL
Success rate is 0 percent (0/5)
Total Time Elapsed 50 ms
```

```
R2#ping mpls pseudowire 13.0.0.8 24 segment 2
Sending 5, 72-byte MPLS Echos to 13.0.0.8,
    timeout is 2 seconds, send interval is 0 msec:
[snip]
Type escape sequence to abort.
LLLLL
Success rate is 0 percent (0/5)
Total Time Elapsed 42 ms
```

```
R2#ping mpls pseudowire 13.0.0.8 24 segment 3
%Total number of MS-PW segments is less than segment number; Adjusting the
segment number to 3
```

```
Sending 5, 72-byte MPLS Echos to 13.0.0.8,
    timeout is 2 seconds, send interval is 0 msec:
[snip]
```

Type escape sequence to abort.

!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 5/8/17 ms
Total Time Elapsed 41 ms

The ultimate test is ensuring the CE devices can communicate over the inter-AS VPLS. Ping and traceroute reveal that connectivity is functional and that the CEs are one hop away at layer 3.

```
R3#ping vrf VPLS 10.0.0.1
```

Type escape sequence to abort.

```
Sending 5, 100-byte ICMP Echos to 10.0.0.1, timeout is 2 seconds:
```

!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 10/10/12 ms

```
R3#traceroute vrf VPLS 10.0.0.1
```

Type escape sequence to abort.

Tracing the route to 10.0.0.1

VRF info: (vrf in name/id, vrf out name/id)

```
 1 10.0.0.1 8 msec 10 msec 9 msec
```

Some additional verifications on the ASBRs are valuable. The L2VPN RIB is a nicely organized view of all PWs in the network. We can see each of the L2VPN routes along with their targets, next-hops, and other details. The output is similar on CSR5 and CSR6 as they each have one iBGP and one eBGP route. The eBGP route is targeted for the TPE loopback but has a next-hop that is the ASBR peer, as expected.

```
R6#show l2vpn rib
```

Local Router ID: 24.0.0.6

+-- Origin of entry (i=iBGP/e=eBGP)

| +- Imported without a matching route target (Yes/No)?

| | +- Provisioned (Yes/No)?

| | | +- Stale entry (Yes/No)?

| | | |

v v v v

O	I	P	S	VPLS-ID	Target ID	Next-Hop	Route-Target
e	N	N	N	13:24	13.0.0.8	10.5.6.5	13:13
i	N	Y	N	13:24	24.0.0.2	24.0.0.2	24:24

```
R5#show l2vpn rib
```

Local Router ID: 13.0.0.5

+-- Origin of entry (i=iBGP/e=eBGP)

| +- Imported without a matching route target (Yes/No)?

| | +- Provisioned (Yes/No)?

| | | +- Stale entry (Yes/No)?

v v v v				
O I P S	VPLS-ID	Target ID	Next-Hop	Route-Target
-+--+--+-----+-----+-----+-----+-----				
i N Y N	13:24	13.0.0.8	13.0.0.8	13:13
e N N N	13:24	24.0.0.2	10.5.6.6	24:24

As a best practice, I would recommend some inter-AS LDP “cleanup” activities on CSR5 and CSR6. Since they are running LDP with one another, they are exchanging all of their LDP-allocated labels. On CSR6, we can see these labels. Since CSR6 has no routes to any of these networks, the labels are useless and waste memory.

```
R6#show mpls ldp bindings neighbor 13.0.0.5
lib entry: 10.5.6.6/32, rev 62
    remote binding: lsr: 13.0.0.5:0, label: 5022
lib entry: 10.5.7.7/32, rev 63
    remote binding: lsr: 13.0.0.5:0, label: 5039
lib entry: 13.0.0.5/32, rev 58
    remote binding: lsr: 13.0.0.5:0, label: imp-null
lib entry: 13.0.0.8/32, rev 61
    remote binding: lsr: 13.0.0.5:0, label: 5002
lib entry: 13.0.0.11/32, rev 60
    remote binding: lsr: 13.0.0.5:0, label: 5001
lib entry: 13.0.0.12/32, rev 59
    remote binding: lsr: 13.0.0.5:0, label: 5000
```

To fix it, we perform outbound LDP label filtering. This is covered in detail in the LDP section. The XE configuration for this is somewhat involved, but the logic of the snippets below indicate that labels for all prefixes can be advertised to any internal peer; I use ACLs matching 13.0.0.0/24 and 24.0.0.0/24 to signify “internal”. Otherwise, no other peers can have any labels for any IP prefixes. This does not affect tLDP label advertisement for PWs as the inter-AS VPLS is still operational.

```
! CSR5
no mpls ldp advertise-labels
mpls ldp advertise-labels for ACL_ANY to ACL_INTERNAL_PEERS
ip access-list standard ACL_ANY
    permit any
ip access-list standard ACL_INTERNAL_PEERS
    permit 13.0.0.0 0.0.0.255
```

```
! CSR6
no mpls ldp advertise-labels
mpls ldp advertise-labels for ACL_ANY to ACL_INTERNAL_PEERS
ip access-list standard ACL_ANY
    permit any
ip access-list standard ACL_INTERNAL_PEERS
    permit 24.0.0.0 0.0.0.255
```

When we check for remote label bindings on each ASBR from the other ASBR, we see no output. This is the expected result.

```
R6#show mpls ldp bindings neighbor 13.0.0.5  
[no output]
```

```
R5#show mpls ldp bindings neighbor 24.0.0.6  
[no output]
```

8.4.2.3 mVPN – GRE (Profile 0)

When using option B, MVPN using GRE is supported between ASes as well. This presents many unique challenges since the only route exchanges that occur are VPN based. For example, this could include VPNv4, VPNv6, L2VPN, etc. Ordinary IPv4/v6 unicast routes are not exchanged which can be considered a benefit of option B as addresses can overlap and remain uncoordinated between providers. Only VPN details, such as RD, RT, VPLS-ID, etc must be coordinated between ASes. We will use VRF EIGRP to test this feature between ASes. I will use PIM-SSM for the default MDT along with BGP IPv4 MDT for signaling. Beginning with AS 24, I configure these basic parameters.

```
! CSR2  
vrf definition EIGRP  
  address-family ipv4  
    mdt default 232.13.24.255  
  address-family ipv6  
    mdt default 232.13.24.255  
  
router bgp 24  
  address-family ipv4 mdt  
  neighbor 24.0.0.14 activate  
  
! XRv4  
multicast-routing  
vrf EIGRP  
  address-family ipv4  
    mdt default ipv4 232.13.24.255  
  address-family ipv6  
    mdt default ipv4 232.13.24.255  
  
router bgp 24  
  address-family ipv4 mdt  
  neighbor 24.0.0.2  
  address-family ipv4 mdt
```

With two PEs in that AS, we should see the default MDT form between XRv4 and CSR2. Checking CSR2, it has the MDT route from XRv4. This carries the default MDT multicast group and the MDT source. Effectively, this is the P(S,G) information needed to build the MDT towards a peer. We can ensure the

default MDTs match by checking the details for both routes on CSR2. There is no concept of RTs for these routes which is why exchanging extended communities is not required for the IPv4 MDT AFI. The P-sources and P-groups are highlighted below.

```
R2#show bgp ipv4 mdt vrf EIGRP detail
BGP routing table entry for 24:3:24.0.0.2/32          version 2
Paths: (1 available, best #1, table IPv4-MDT-BGP-Table)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (24.0.0.2)
      Origin incomplete, localpref 100, valid, sourced, local, best,
      MDT group address: 232.13.24.255
      rx pathid: 0, tx pathid: 0x0

BGP routing table entry for 24:3:24.0.0.14/32        version 3
Paths: (1 available, best #1, table IPv4-MDT-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    24.0.0.14 from 24.0.0.14 (24.0.0.14)
      Origin IGP, localpref 100, valid, internal, best,
      MDT group address: 232.13.24.255
      rx pathid: 0, tx pathid: 0x0
```

Checking the P(S,G) information on CSR2 and XRv4, we can see the SPTs built between the peers. Since the routers are directly connected, the tree exists only on one link for now. CSR2 marks this with the big 'Z' flag to signify a multicast tunnel.

```
R2#show ip mroute 232.13.24.255 24.0.0.14 | begin \
(24.0.0.14, 232.13.24.255), 00:02:48/00:00:11, flags: sTIZ
Incoming interface: GigabitEthernet2.524, RPF nbr 24.2.14.14
Outgoing interface list:
  MVRF EIGRP, Forward/Sparse, 00:02:48/00:00:11

RP/0/0/CPU0:XRv4#show pim topology 232.13.24.255 24.0.0.2 | begin 232
(24.0.0.2,232.13.24.255)SPT SSM Up: 00:03:08
JP: Join(00:00:39) RPF: GigabitEthernet0/0/0/0.524,24.2.14.2 Flags:
Loopback0          00:03:08  fwd LI LH
```

We can see a VRF-aware PIM neighbor inside the VPN, indicating that the default MDT is working.

```
R2#show ip pim vrf EIGRP neighbor | begin ^Neighbor
Neighbor          Interface          Uptime/Expires    Ver    DR
Address
10.1.2.1          GigabitEthernet2.512  1d18h/00:01:41   v2     1 / S P G
```



```
24.0.0.14 Tunnel15 00:04:10/00:01:19 v2 1 / DR P G
```

```
RP/0/0/CPU0:XRv4#show pim vrf EIGRP neighbor | begin ^Neighbor
Neighbor Address      Interface      Uptime      Expires      DR pri      Flags
10.13.14.13           GigabitEthernet0/0/0/0.534 1d21h      00:01:36 1          B P
10.13.14.14*          GigabitEthernet0/0/0/0.534 2d12h      00:01:27 1 (DR) B P E
24.0.0.2              mdtEIGRP      00:04:33   00:01:36 1          P
24.0.0.14*            mdtEIGRP      00:06:45   00:01:26 1 (DR) P
```

Because we want to extend this MDT across AS boundaries, we must run IPv4 MDT with the ASBRs as well. CSR2 will be an RR for this AFI and will peer with CSR6 and CSR7 as well.

```
! CSR2
router bgp 24
 address-family ipv4 mdt
  neighbor 24.0.0.6 activate
  neighbor 24.0.0.6 route-reflector-client
  neighbor 24.0.0.7 activate
  neighbor 24.0.0.7 route-reflector-client
  neighbor 24.0.0.14 route-reflector-client
```

```
! CSR6 and CSR7
router bgp 24
 address-family ipv4 mdt
  neighbor 24.0.0.2 activate
```

On CSR2, we quickly verify all of the sessions come up. We also verify that CSR6 and CSR7 learn the MDT routes from CSR2. CSR6 and CSR7 are not sending any new routes into the network, so we expected to see 0 prefixes received. There is no concept of RT retention for this AFI, so the option B ASBRs do not need to worry about filtering the routes not used locally.

```
R2#show bgp ipv4 mdt all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent      TblVer  InQ  OutQ  Up/Down  State/PfxRcd
24.0.0.6      4      24      24      216         5     0     0 00:00:56      0
24.0.0.7      4      24      21      221         5     0     0 00:00:56      0
24.0.0.14     4      24     156     199         5     0     0 00:00:40      1
```

```
R6#show bgp ipv4 mdt all | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 24:3 (default for vrf EIGRP)
*>i 24.0.0.2/32      24.0.0.2          0     100      0 ?
*>i 24.0.0.14/32     24.0.0.14         100    0 i
```

```
R7#show bgp ipv4 mdt all | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 24:3 (default for vrf EIGRP)
*>i 24.0.0.2/32      24.0.0.2          0     100      0 ?
*>i 24.0.0.14/32     24.0.0.14         100    0 i
```

Before configuring the eBGP IPv4 MDT connections to AS 13, I will configure the intra-AS parameters inside AS 13. Since there is only 1 PE, there won't be any MDT construction to verify. We will build the BGP sessions and ensure the MDT route from CSR8 is present on XRv1 and CSR5 (ASBRs). Note that there is no reason for XRv2 to negotiate this AFI with CSR8; VRF OSPF will not be using PIM/GRE for MVPN. It is required that the default MDT group match between ASes, so this must be coordinated between providers.

```
! XRv2
router bgp 13
  address-family ipv4 mdt

  af-group MDT address-family ipv4 mdt
  route-reflector-client

neighbor 13.0.0.5
  address-family ipv4 mdt
  use af-group MDT

neighbor 13.0.0.11
  address-family ipv4 mdt
  use af-group MDT

multicast-routing
vrf EIGRP
  address-family ipv4
    mdt default ipv4 232.13.24.255
  address-family ipv6
    mdt default ipv4 232.13.24.255

! XRv1
router bgp 13
  address-family ipv4 mdt
  neighbor 13.0.0.12
  address-family ipv4 mdt

! CSR5
Router bgp 13
  address-family ipv4 mdt
  neighbor 13.0.0.12 activate
```

Checking XRv2, we can see the IPv4 MDT peers are up; no routes are received from the ASBRs as expected. Both XRv1 and CSR5 have XRv2's MDT route. I show the summary on CSR5 and the details on XRv1 to ensure the P(S,G) information is correct. This verifies all of the BGP auto-discovery signaling.

```
RP/0/0/CPU0:XRv2#show bgp ipv4 mdt summary | begin ^Neighbor
Neighbor      Spk    AS MsgRcvd MsgSent  TblVer  InQ  OutQ  Up/Down  St/PfxRcd
```

13.0.0.5	0	13	24593	23511	2	0	0	00:00:15	0
13.0.0.11	0	13	23250	23470	2	0	0	00:00:14	0

```
R5#show bgp ipv4 mdt all | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 13:3
*>i 13.0.0.12/32    13.0.0.12      100      0 i
```

```
RP/0/0/CPU0:XRv1#show bgp ipv4 mdt rd 13:3 13.0.0.12
BGP routing table entry for 13.0.0.12/96, Route Distinguisher: 13:3
[snip]
Local
  13.0.0.12 (metric 3) from 13.0.0.12 (13.0.0.12)
    Origin IGP, localpref 100, valid, internal, best, group-best
    Received Path ID 0, Local Path ID 1, version 2
    MDT group address: 232.13.24.255
```

Next, we configure the eBGP IPv4 MDT peers. Unlike MPLS L3VPN and L2VPN, there is no reliance on MPLS here, so any kind of label operations need not occur on the transit links. The configuration is very basic so I only show XRv1 and CSR6 for brevity. Since all of these eBGP neighbors were already defined, we only need to activate the new AFI rather than redefine the general session parameters.

```
! XRv1
router bgp 13
  neighbor 10.6.11.6
  address-family ipv4 mdt
    route-policy RPL_PASS in
    route-policy RPL_PASS out

! CSR6
router bgp 24
  address-family ipv4 mdt
  neighbor 10.5.6.5 activate
  neighbor 10.6.11.11 activate
```

Checking CSR6 and CSR5 for neighbors, we can see that all peers are up. CSR6 receives 1 prefix from CSR5 and XRv1 which represents XRv2's MDT route. CSR5 learns 2 prefixes from CSR6 and CSR7 which represent MDT routes from CSR2 and XRv4. So far, everything looks good.

```
R6#show bgp ipv4 mdt all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State/PfxRcd
10.5.6.5      4      13      57      48      4      0   0 00:01:13      1
10.6.11.11    4      13      14      34      4      0   0 00:00:23      1
24.0.0.2      4      24     466     196      4      0   0 00:16:58      2

R5#show bgp ipv4 mdt all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State/PfxRcd
10.5.6.6      4      24      49      58      4      0   0 00:02:02      2
10.5.7.7      4      24      47      43      4      0   0 00:01:50      2
```

```
13.0.0.12      4      13      228      136      4      0      0 00:08:14      1
```

Next, we will verify that the RRs in each AS have received these MDT routes. CSR2 shows two copies of the route from CSR6 and CSR7. The routes appear valid since the transit link host routes were redistributed into IS-IS earlier.

```
R2#show bgp ipv4 mdt rd 13:3 13.0.0.12
BGP routing table entry for 13:3:13.0.0.12/32          version 6
Paths: (2 available, best #2, table IPv4-MDT-BGP-Table)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  13, (Received from a RR-client)
    10.5.7.5 from 24.0.0.7 (24.0.0.7)
      Origin IGP, metric 0, localpref 100, valid, internal,
      MDT group address: 232.13.24.255

      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  13, (Received from a RR-client)
    10.5.6.5 from 24.0.0.6 (24.0.0.6)
      Origin IGP, metric 0, localpref 100, valid, internal, best,
      MDT group address: 232.13.24.255
      rx pathid: 0, tx pathid: 0x0
```

XRv2 cannot install these BGP routes since the next-hop is inaccessible. For L3VPN tests, AS 13 ASBRs used next-hop-self rather than advertise the transit links. With inter-AS PIM/GRE using option B, we can use either method.

```
RP/0/0/CPU0:XRv2#show bgp ipv4 mdt rd 24:3 24.0.0.2
BGP routing table entry for 24.0.0.2/96, Route Distinguisher: 24:3
Versions:
  Process          bRIB/RIB  SendTblVer
  Speaker          0          0
Paths: (2 available, no best path)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  24, (Received from a RR-client)
    10.5.6.6 (inaccessible) from 13.0.0.5 (13.0.0.5)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Received Path ID 0, Local Path ID 0, version 0
      MDT group address: 232.13.24.255

  Path #2: Received by speaker 0
  Not advertised to any peer
  24, (Received from a RR-client)
    10.6.11.6 (inaccessible) from 13.0.0.11 (13.0.0.11)
```

```
Origin incomplete, localpref 100, valid, internal
Received Path ID 0, Local Path ID 0, version 0
MDT group address: 232.13.24.255
```

Correcting this problem is simple; we apply next-hop-self on the ASBRs towards the RR and the next-hops become accessible again. Now, iBGP speakers inside of each AS can process the BGP routes.

```
! XRv1
router bgp 13
 neighbor 13.0.0.12
  address-family ipv4 mdt
  next-hop-self
```

```
! CSR5
router bgp 13
 address-family ipv4 mdt
 neighbor 13.0.0.12 next-hop-self
```

```
RP/0/0/CPU0:XRv2#show bgp ipv4 mdt rd 24:3 24.0.0.2 | begin 24,
24, (Received from a RR-client)
 13.0.0.5 (metric 3) from 13.0.0.5 (13.0.0.5)
  Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best
  Received Path ID 0, Local Path ID 1, version 5
  MDT group address: 232.13.24.255
```

```
Path #2: Received by speaker 0
Not advertised to any peer
24, (Received from a RR-client)
 13.0.0.11 (metric 3) from 13.0.0.11 (13.0.0.11)
  Origin incomplete, localpref 100, valid, internal
  Received Path ID 0, Local Path ID 0, version 0
  MDT group address: 232.13.24.255
```

With the routes properly advertised and installed, ideally the inter-AS MDT would be built. A quick check of all three PEs shows that something is wrong. Every single PE for every single P(S,G) has the same issue; there is no route back to the P-source. This makes sense since these loopbacks were never exchanged across AS boundaries. This is somewhat similar to the L2VPN problem we solved with MSPW as building an end-to-end PW is not possible.

```
RP/0/0/CPU0:XRv4#show pim topology 232.13.24.255 13.0.0.12 | begin 232
(13.0.0.12,232.13.24.255)SPT SSM Up: 00:20:01
JP: Join(00:00:46) RPF: Null1,0.0.0.0 Flags:
  Loopback0 00:20:01 fwd LI LH
```

```
R2#show ip mroute 232.13.24.255 13.0.0.12 | begin \
(13.0.0.12, 232.13.24.255), 00:20:36/stopped, flags: sTIZ
```

```
Incoming interface: Null, RPF nbr 0.0.0.0
Outgoing interface list:
  MVRP EIGRP, Forward/Sparse, 00:20:36/00:00:23
```

```
RP/0/0/CPU0:XRv2#show pim topology 232.13.24.255 24.0.0.14 | begin 232
(24.0.0.14,232.13.24.255)SPT SSM Up: 00:12:12
JP: Join(00:00:41) RPF: Null,0.0.0.0 Flags:
  Loopback0                                00:12:12  fwd LI LH
```

```
RP/0/0/CPU0:XRv2#show pim topology 232.13.24.255 24.0.0.2 | begin 232
(24.0.0.2,232.13.24.255)SPT SSM Up: 00:12:14
JP: Join(00:00:39) RPF: Null,0.0.0.0 Flags:
  Loopback0                                00:12:14  fwd LI LH
```

The fact that each router is actively trying to build the MDTs is a good sign that BGP is configured properly. We now must adjust PIM so that it can build trees between ASes by somehow fixing RPF. The normal RPF fix-up techniques, such as static multicast routes or BGP IPv4/v6 multicast AFI, are not appropriate here. A specific feature known as PIM proxy vector was invented specifically to solve this problem. This is configured on the PEs which encode the BGP next-hop inside of the PIM joins; this allows routers to compute RPF towards the vector address rather than the root address. To demonstrate the basic functionality, I enable this on CSR2. I also include the RD which is required for using this feature with MPLS VPNs.

```
! CSR2
ip multicast vrf EIGRP rpf proxy rd vector
```

To verify that the vector has been originated, we can check the MRIB for proxy entries. The P(S,G) for the inter-AS PE is shown below. The RD of 13:3 is encoded along with the BGP next-hop of the best route. The assigner is local to CSR2.

```
R2#show ip mroute proxy
(13.0.0.12, 232.13.24.255)
Proxy          Assigner      Origin      Uptime/Expire
13:3/10.5.6.5  0.0.0.0      BGP MDT     00:03:10/stopped
```

The ability to communicate this new PIM TLV is a special PIM capability negotiated during neighbor formation. The 'P' flag in the PIM neighbors shows which peers can support it. At a glance, it appears all neighbors can from CSR2's perspective. We have used this PIM show command many times but never paid attention to the 'P' flag until now.

```
R2#show ip pim neighbor | begin ^Neigh
Neighbor      Interface      Uptime/Expires  Ver  DR
Address
24.2.14.14    GigabitEthernet2.524  2d13h/00:01:44  v2   1 / DR P G
24.2.7.7      GigabitEthernet2.527  2d13h/00:01:25  v2   1 / DR S P G
```

Looking at the MRIB, we can see that the proxy information is revealed here per (S,G). The big 'V' flag means both the PIM vector and RD are encoded in the PIM joins. When the vector is present, PIM routers will use that for RPF rather than the root of the tree. All routers in AS 24 should have a route to 10.5.6.5, but none of them have a route to 13.0.0.12. Now, CSR2 appears to have a valid P(S,G) entry for this group.

```
R2#show ip mroute 232.13.24.255 13.0.0.12 | begin \  
(13.0.0.12, 232.13.24.255), 01:12:57/stopped, flags: STIZV  
  Incoming interface: GigabitEthernet2.524, RPF nbr 24.2.14.14, vector  
10.5.6.5  
  Outgoing interface list:  
    MVRP EIGRP, Forward/Sparse, 01:12:57/00:02:43  
  
R2#show ip rpf 10.5.6.5  
RPF information for ? (10.5.6.5)  
  RPF interface: GigabitEthernet2.524  
  RPF neighbor: ? (24.2.14.14)  
  RPF route/mask: 10.5.6.5/32  
  RPF type: unicast (isis 24)  
  Doing distance-preferred lookups across tables  
  RPF topology: ipv4 multicast base, originated from ipv4 unicast base
```

```
R2#show ip rpf 13.0.0.12  
failed, no route exists
```

With debugging enabled, we can see CSR2 originate the P(S,G) join towards XRv4 which is in the reverse path towards 10.5.6.5. The RD and vector are both included in the PIM join.

```
R2#debug ip pim 232.13.24.255  
PIM debugging is on  
  
PIM(0): Insert (13.0.0.12,232.13.24.255) join in nbr 24.2.14.14's queue  
PIM(0): Building Join/Prune packet for nbr 24.2.14.14  
PIM(0): Adding v2 (13.0.0.12/32, 232.13.24.255), S-bit Join MDT proxy  
13:3/10.5.6.5  
PIM(0): Send v2 join/prune to 24.2.14.14 (GigabitEthernet2.524)
```

When XRv4 receives the join, it claims the proxy vector is disabled towards CSR2 (and reveals a spelling error).

```
RP/0/0/CPU0:XRv4#debug pim protocol join-prune  
pim[1160]: [13] VRF : default Received J/P on Gi0/0/0/0.524 from 24.2.14.2  
target: 24.2.14.14 (to us) containing 1 group size:46  
pim[1160]: [13] J/P group 232.13.24.255 found grange in vrf default  
pim[1160]: [13] VRF : default J/P Group 232.13.24.255 includes 1 joins  
pim[1160]: [13] VRF : default J/P recvieved when proxy is disabled to  
13.0.0.12
```

```
pim[1160]: [13] VRF : default J/P Group 232.13.24.255 includes 0 prunes
```

We can enable the feature under XRv4, taking note that we enable it under the global AFI rather than the EIGRP VPN. We are trying to get XRv4 to at least understand the P(S,G) from CSR2 first.

```
! XRv4
router pim
  address-family ipv4
    rpf-vector
```

With debugging still enabled, we are now presented with another error. Although cryptic, this message effectively says that XR does not understand the PIM vector + RD join message. XR makes no attempt to read the vector address by itself, even if it cannot understand the RD. A component of the PIM join for (13.0.0.12, 232.13.24.255) from CSR2 cannot be understood. As a result, XR totally ignores the PIM join.

```
RP/0/0/CPU0:XRv4#debug pim protocol join-prune
pim[1160]: [13] VRF : default Received J/P on Gi0/0/0/0.524 from 24.2.14.2
target: 24.2.14.14 (to us) containing 1 group size:46
pim[1160]: [13] J/P group 232.13.24.255 found grange in vrf default
pim[1160]: [13] VRF : default J/P Group 232.13.24.255 includes 1 joins
pim[1160]: [13] VRF : default J/P with unknown proxy type 2 forwarding...
pim[1160]: [13] VRF : default, RECV J/P entry: Join, root: 13.0.0.12 proxy
0.0.0.0, grp: 232.13.24.255, tgt: 24.2.14.14, flags: S , on intf
Gi0/0/0/0.524, sender: 24.2.14.2
```

Until testing this feature, I was not aware that XR has no support for the PIM vector with RD. It only supports the PIM vector by itself, which means XR cannot be used in an option B environment for inter-AS MVPN. Not supporting RD means that inter-AS multicast support is still possible but not within the scope of MPLS VPNs. XRv4 has no RPF interface for this P(S,G) and cannot build the tree.

```
RP/0/0/CPU0:XRv4#show pim topology 232.13.24.255 13.0.0.12 | begin 232
(13.0.0.12,232.13.24.255)SPT SSM Up: 01:26:49 Vector: 0.0.0.0
JP: Join(00:00:38) RPF: Null,0.0.0.0 proxy-disabled, Flags:
  Loopback0 00:18:25 fwd LI LH
  GigabitEthernet0/0/0/0.524 00:05:07 fwd Join(00:03:16)
```

In an attempt to demonstrate the operation of the PIM vector with RD, I will make an RPF adjustment on CSR2. Using a static multicast route, I will assign CSR7 as the RPF neighbor for 10.0.0.0/8 which will cover the transit interfaces. I verify that the multicast route is properly installed and active.

```
! CSR2
ip mroute 10.0.0.0 255.0.0.0 24.2.7.7
```

```
R2#show ip static route multicast | begin Static
Static multicast local RIB for multicast
MC 10.0.0.0/8 [1/0] via 24.2.7.7 [A]
```


Debugging on CSR2, we can see the PIM join with vector + RD is now being sent to CSR7, who understands the message. We can also confirm this by checking the MRIB. We can see the vector is still applied, but the RPF neighbor has been adjusted via an “Mroute”. The only reason this is a decent solution is because we can easily bypass the only XR router in the AS. If there were other XR routers, the static multicast routes would be needed anywhere the RPF interfaces would transit XR routers. This is extremely sloppy and even using BGP IPv4 multicast is a poor option since we are effectively bypassing entire sets of routers.

```
R2#debug ip pim 232.13.24.255
PIM debugging is on
PIM(0): Insert (13.0.0.12,232.13.24.255) join in nbr 24.2.7.7's queue
PIM(0): Building Join/Prune packet for nbr 24.2.7.7
PIM(0): Adding v2 (13.0.0.12/32, 232.13.24.255), S-bit Join MDT proxy
13:3/10.5.6.5
PIM(0): Send v2 join/prune to 24.2.7.7 (GigabitEthernet2.527)
```

```
R2#show ip mroute 232.13.24.255 13.0.0.12 | begin \
(13.0.0.12, 232.13.24.255), 01:31:13/stopped, flags: sTIZV
Incoming interface: GigabitEthernet2.527, RPF nbr 24.2.7.7, Mroute, vector
10.5.6.5
Outgoing interface list:
MVRFP EIGRP, Forward/Sparse, 01:31:13/00:02:27
```

Checking CSR7, we can see the PIM vector from CSR2. Rather than being locally originated and assigned by BGP, this vector was PIM-learned from CSR2. CSR7’s RPF route for 10.5.6.5 will be IGP-learned from CSR6, so no RPF fixup is needed.

```
R7#show ip mroute proxy
(13.0.0.12, 232.13.24.255)


| Proxy         | Assigner | Origin | Uptime/Expire     |
|---------------|----------|--------|-------------------|
| 13:3/10.5.6.5 | 24.2.7.2 | PIM    | 00:05:26/00:02:16 |


```

```
R7#show ip mroute 232.13.24.255 13.0.0.12 | begin \
(13.0.0.12, 232.13.24.255), 00:06:14/00:02:57, flags: sTV
Incoming interface: GigabitEthernet2.567, RPF nbr 24.6.7.6, vector 10.5.6.5
Outgoing interface list:
GigabitEthernet2.527, Forward/Sparse, 00:06:14/00:02:57
```

CSR7 passes the PIM join to CSR6 who has a similar set of outputs. The incoming interface is via CSR5 due to being the oldest eBGP route.

```
R6#show ip mroute proxy
(13.0.0.12, 232.13.24.255)


| Proxy         | Assigner | Origin | Uptime/Expire     |
|---------------|----------|--------|-------------------|
| 13:3/10.5.6.5 | 24.6.7.7 | PIM    | 00:07:51/00:02:02 |


```

```

R6#show ip mroute 232.13.24.255 13.0.0.12 | begin \
(13.0.0.12, 232.13.24.255), 00:08:06/00:03:16, flags: sTV
Incoming interface: GigabitEthernet2.556, RPF nbr 10.5.6.5, vector 10.5.6.5
Outgoing interface list:
GigabitEthernet2.567, Forward/Sparse, 00:08:06/00:03:16

R6#show bgp ipv4 mdt rd 13:3 13.0.0.12
BGP routing table entry for 13:3:13.0.0.12/32          version 4
Paths: (2 available, best #2, table IPv4-MDT-BGP-Table)
Advertised to update-groups:
  1          2
Refresh Epoch 1
13
10.6.11.11 from 10.6.11.11 (13.0.0.11)
  Origin IGP, localpref 100, valid, external,
  MDT group address: 232.13.24.255
  rx pathid: 0, tx pathid: 0
Refresh Epoch 1
13
10.5.6.5 from 10.5.6.5 (13.0.0.5)
  Origin IGP, localpref 100, valid, external, best,
  MDT group address: 232.13.24.255
  rx pathid: 0, tx pathid: 0x0

```

We are fortunate that CSR5 was selected over XRv1 since we know XRv1 cannot understand the PIM vector + RD. Checking CSR5 quickly, we can see the PIM vector was learned. The actual vector gets removed at this point as it no longer serves its purpose; everyone in AS 13 has a route to 13.0.0.12. CSR5 makes no mention of the vector when the PIM is sent to CSR8. I prove this with show and debug commands as the PIM vector + RD is received but not forwarded.

```

R5#show ip mroute proxy
(13.0.0.12, 232.13.24.255)
Proxy          Assigner          Origin    Uptime/Expire
13:3/local     10.5.6.6          PIM       00:11:49/00:02:02

```

```

R5#show ip mroute 232.13.24.255 13.0.0.12 | begin \
(13.0.0.12, 232.13.24.255), 00:14:48/00:03:28, flags: sT
Incoming interface: GigabitEthernet2.558, RPF nbr 13.5.8.8
Outgoing interface list:
GigabitEthernet2.556, Forward/Sparse, 00:14:48/00:03:28

```

```

R5#debug ip pim 232.13.24.255
PIM debugging is on

```

```

PIM(0): Received v2 Join/Prune on GigabitEthernet2.556 from 10.5.6.6, to us
PIM(0): Join-list: (13.0.0.12/32, 232.13.24.255), S-bit set, RD/V
13:3/10.5.6.5

```

PIM(0): Update GigabitEthernet2.556/10.5.6.6 to (13.0.0.12, 232.13.24.255), Forward state, by PIM SG Join

PIM(0): Insert (13.0.0.12,232.13.24.255) join in nbr 13.5.8.8's queue

PIM(0): Building Join/Prune packet for nbr 13.5.8.8

PIM(0): Adding v2 (13.0.0.12/32, 232.13.24.255), S-bit Join

PIM(0): Send v2 join/prune to 13.5.8.8 (GigabitEthernet2.558)

To demonstrate what happens when XRv1 is the best ingress point into AS 13, I will clear CSR6's BGP session to CSR5. CSR6 now selects XRv1 as the best route as it is the oldest.

```
R6#clear bgp ipv4 mdt 10.5.6.5
```

```
R6#show bgp ipv4 mdt rd 13:3 13.0.0.12
```

```
BGP routing table entry for 13:3:13.0.0.12/32          version 9
```

```
Paths: (2 available, best #2, table IPv4-MDT-BGP-Table)
```

```
  Advertised to update-groups:
```

```
    1          2
```

```
Refresh Epoch 2
```

```
13
```

```
  10.5.6.5 from 10.5.6.5 (13.0.0.5)
```

```
    Origin IGP, localpref 100, valid, external,
```

```
    MDT group address: 232.13.24.255
```

```
    rx pathid: 0, tx pathid: 0
```

```
Refresh Epoch 1
```

```
13
```

```
  10.6.11.11 from 10.6.11.11 (13.0.0.11)
```

```
    Origin IGP, localpref 100, valid, external, best,
```

```
    MDT group address: 232.13.24.255
```

```
    rx pathid: 0, tx pathid: 0x0
```

When CSR2 receives it, it still prefers routes from CSR6 due to having a lower BGP RID than CSR7, except CSR6's advertised best path now has a next-hop of 10.6.11.11.

```
R2#show bgp ipv4 mdt rd 13:3 13.0.0.12
```

```
BGP routing table entry for 13:3:13.0.0.12/32          version 11
```

```
Paths: (2 available, best #2, table IPv4-MDT-BGP-Table)
```

```
  Advertised to update-groups:
```

```
    2
```

```
Refresh Epoch 1
```

```
13, (Received from a RR-client)
```

```
  10.5.7.5 from 24.0.0.7 (24.0.0.7)
```

```
    Origin IGP, metric 0, localpref 100, valid, internal,
```

```
    MDT group address: 232.13.24.255
```

```
    rx pathid: 0, tx pathid: 0
```

```
Refresh Epoch 1
```

```
13, (Received from a RR-client)
```

```
  10.6.11.11 from 24.0.0.6 (24.0.0.6)
```

```
Origin IGP, metric 0, localpref 100, valid, internal, best,  
MDT group address: 232.13.24.255  
rx pathid: 0, tx pathid: 0x0
```

This is acceptable since CSR2 has an RPF fixup for all of 10.0.0.0/8, so the RPF interface is still towards CSR7. CSR7 receives the join from CSR2 and passes it to CSR6. CSR6 then passes it to XRv1 along with the PIM vector + RD.

```
R2#show ip mroute 232.13.24.255 13.0.0.12 | begin \  
(13.0.0.12, 232.13.24.255), 01:49:46/00:02:30, flags: sTIZV  
Incoming interface: GigabitEthernet2.527, RPF nbr 24.2.7.7, Mroute, vector  
10.6.11.11  
Outgoing interface list:  
MVRP EIGRP, Forward/Sparse, 01:49:46/00:02:30  
  
R6#show ip mroute 232.13.24.255 13.0.0.12 | begin \  
(13.0.0.12, 232.13.24.255), 00:22:11/00:03:00, flags: sTV  
Incoming interface: GigabitEthernet2.561, RPF nbr 10.6.11.11, vector  
10.6.11.11  
Outgoing interface list:  
GigabitEthernet2.567, Forward/Sparse, 00:22:11/00:03:00
```

Debugging PIM on XRv1, we can see the PIM join received from CSR6 but is rejected. Even though XRv1 has a perfectly valid RPF to 13.0.0.2, the reception of a join with an unknown TLV is grounds for ignoring it entirely. It builds P(S,G) state but the RPF remains null.

```
RP/0/0/CPU0:XRv1#debug pim protocol join-prune  
  
pim[1160]: [13] VRF : default (13.0.0.12,232.13.24.255) J/P processing  
pim[1160]: [13] VRF : default(13.0.0.12,232.13.24.255) No RPF neighbor to  
send J/P  
pim[1160]: [13] VRF : default Received J/P on Gi0/0/0/0.561 from 10.6.11.6  
target: 10.6.11.11 (to us) containing 1 group size:46  
pim[1160]: [13] J/P group 232.13.24.255 found grange in vrf default  
pim[1160]: [13] VRF : default J/P Group 232.13.24.255 includes 1 joins  
pim[1160]: [13] VRF : default J/P with unknown proxy type 2 forwarding...  
pim[1160]: [13] VRF : default, RECV J/P entry: Join, root: 13.0.0.12 proxy  
0.0.0.0, grp: 232.13.24.255, tgt: 10.6.11.11, flags: S , on intf  
Gi0/0/0/0.561, sender: 10.6.11.6  
pim[1160]: [13] VRF : default J/P Group 232.13.24.255 includes 0 prunes
```

We can confirm the P(S,G) creation and valid RPF using ordinary PIM show commands.

```
RP/0/0/CPU0:XRv1#show pim topology 232.13.24.255 | begin 232  
(13.0.0.12,232.13.24.255)SPT SSM Up: 00:05:28 Vector: 0.0.0.0  
JP: Join(00:00:24) RPF: Null,0.0.0.0 proxy-disabled, Flags:  
GigabitEthernet0/0/0/0.561 00:05:28 fwd Join(00:03:06)
```

```
RP/0/0/CPU0:XRv1#show pim rpf 13.0.0.12
```

```
Table: IPv4-Unicast-default
```

```
* 13.0.0.12/32 [110/3]
```

```
via GigabitEthernet0/0/0/0.581 with rpf neighbor 13.8.11.8
```

As expected, XR cannot support this architecture. Rather than leave CSR6's bestpath decision to chance, I will configure MED outbound on XRv1 so that CSR5 is always the preferred ingress point. This way, when routers reboot or BGP sessions are cleared, inter-AS MVPN can still work by bypassing XRv1.

```
! XRv1
route-policy RPL_MDT_MED_OUT($MED)
  set med $MED
end-policy

router bgp 13
  neighbor 10.6.11.6
  address-family ipv4 mdt
  route-policy RPL_MDT_MED_OUT(1111) out
```

We check CSR6 to ensure the MED was set correctly and that CSR6 selects CSR5 as the best-path.

```
R6#show bgp ipv4 mdt rd 13:3 13.0.0.12
BGP routing table entry for 13:3:13.0.0.12/32          version 10
Paths: (2 available, best #1, table IPv4-MDT-BGP-Table)
  Advertised to update-groups:
    1          2
Refresh Epoch 2
13
  10.5.6.5 from 10.5.6.5 (13.0.0.5)
    Origin IGP, localpref 100, valid, external, best,
    MDT group address: 232.13.24.255
    rx pathid: 0, tx pathid: 0x0
Refresh Epoch 1
13
  10.6.11.11 from 10.6.11.11 (13.0.0.11)
    Origin IGP, metric 1111, localpref 100, valid, external,
    MDT group address: 232.13.24.255
    rx pathid: 0, tx pathid: 0
```

A quick check on CSR2 shows this as well, which means the PIM join with vector + RD should be going through CSR5 again.

```
R2#show bgp ipv4 mdt rd 13:3 13.0.0.12
BGP routing table entry for 13:3:13.0.0.12/32          version 12
Paths: (2 available, best #2, table IPv4-MDT-BGP-Table)
  Advertised to update-groups:
```

```

2
Refresh Epoch 1
13, (Received from a RR-client)
  10.5.7.5 from 24.0.0.7 (24.0.0.7)
    Origin IGP, metric 0, localpref 100, valid, internal,
    MDT group address: 232.13.24.255
    rx pathid: 0, tx pathid: 0
Refresh Epoch 1
13, (Received from a RR-client)
  10.5.6.5 from 24.0.0.6 (24.0.0.6)
    Origin IGP, metric 0, localpref 100, valid, internal, best,
    MDT group address: 232.13.24.255
    rx pathid: 0, tx pathid: 0x0

```

We will pick up our verification on CSR5 where we left off. CSR5 passes the join to CSR8 using the normal RPF rules (no need for PIM vector) and CSR8 passes the join to XRv2.

```

R5#show ip mroute 232.13.24.255 13.0.0.12 | begin \
(13.0.0.12, 232.13.24.255), 00:02:14/00:03:06, flags: sT
  Incoming interface: GigabitEthernet2.558, RPF nbr 13.5.8.8
  Outgoing interface list:
    GigabitEthernet2.556, Forward/Sparse, 00:02:14/00:03:06

```

```

R8#show ip mroute 232.13.24.255 13.0.0.12 | begin \
(13.0.0.12, 232.13.24.255), 00:02:43/00:02:44, flags: sT
  Incoming interface: GigabitEthernet2.582, RPF nbr 13.8.12.12
  Outgoing interface list:
    GigabitEthernet2.558, Forward/Sparse, 00:02:43/00:02:44

```

Since there is no need for PIM vector or RD for this right-to-left MDT, XRv2 simply adds CSR8 to its OIL. This means that multicast signaling within the VPN can technically flow from left-to-right as XRv2 is capable of forwarding traffic down the MDT. The design is dysfunctional but it's better than having no connectivity.

```

RP/0/0/CPU0:XRv2#show pim topology 232.13.24.255 13.0.0.12 | begin 232
(13.0.0.12,232.13.24.255)SPT SSM Up: 02:09:58
JP: Join(never) RPF: Loopback0,13.0.0.12* Flags:
  Loopback0                02:09:58  fwd LI LH
  GigabitEthernet0/0/0/0.582 00:04:03  fwd Join(00:03:24)

```

To prove this, we can check the PIM neighbors on XRv2 and CSR2. CSR2 sees XRv2 as a VPN PIM neighbor but not vice versa. This is because XRv2's PIM hellos are traversing the inter-AS MDT and being received by CSR2. The opposite is not true since CSR2 cannot join XRv2's SPT.

```

RP/0/0/CPU0:XRv2#show pim vrf EIGRP neighbor | begin ^Neigh
Neighbor Address      Interface                Uptime    Expires  DR pri  Flags
10.3.12.3             GigabitEthernet0/0/0/0.532 2d14h     00:01:41 1      P

```

```

10.3.12.12*      GigabitEthernet0/0/0/0.532 2d14h      00:01:28 1 (DR) B P E
13.0.0.12*      mdtEIGRP                        02:12:58 00:01:19 1 (DR) P

```

```

R2#show ip pim vrf EIGRP neighbor | begin ^Neigh
Neighbor          Interface          Uptime/Expires      Ver    DR
Address
10.1.2.1          GigabitEthernet2.512    1d21h/00:01:43     v2     1 / S P G
13.0.0.12         Tunnel5                 00:07:09/00:01:36  v2     1 / P G
24.0.0.14         Tunnel5                 00:32:16/00:01:34  v2     1 / DR G

```

Since XRv4 is incapable of originating the PIM vector + RD, we can use a static multicast route for the remote SPT root. This is worse than what we did on CSR2, which was simply to change the RPF to the transit links; we still used PIM vector + RD. On XRv4, we are reducing the scalability of option B since XRv4 needs to identify the remote loopbacks specifically. Other than using BGP IPv4 multicast for RPF fixup, this is the most logical way to fix the problem. For variety, I will use CSR7 as the RPF interface. CSR7 will be the replication point for intra-AS multicast as the links to XRv4 and CSR2 will be downstream interfaces in the OIL. I also remove the PIM configuration that enables RPF vector as it is just clutter.

```

! XRv4
no router pim

router static
 address-family ipv4 multicast
  13.0.0.0/8 24.7.14.7

```

XRv4 can now issue a PIM join towards 13.0.0.2 using CSR7. There is no PIM proxy involved on XRv4 and CSR7 treats this like a normal PIM join. It adds another interface to the OIL, and the rest of the MDT upstream remains unchanged. CSR6 is still CSR7's RPF neighbor and is the ASBR from which the EIGRP VPN multicast will arrive.

```

RP/0/0/CPU0:XRv4#show pim topology 232.13.24.255 13.0.0.12 | begin 232
(13.0.0.12,232.13.24.255)SPT SSM Up: 02:12:02
JP: Join(00:00:25) RPF: GigabitEthernet0/0/0/0.574,24.7.14.7 Flags:
  Loopback0                00:44:11 fwd LI LH

```

```

R7#show ip mroute 232.13.24.255 13.0.0.12 | begin \
(13.0.0.12, 232.13.24.255), 00:40:14/00:03:20, flags: sTV
Incoming interface: GigabitEthernet2.567, RPF nbr 24.6.7.6, vector 10.5.6.5
Outgoing interface list:
  GigabitEthernet2.574, Forward/Sparse, 00:00:21/00:03:08
  GigabitEthernet2.527, Forward/Sparse, 00:40:14/00:03:20

```

As a result of this, XRv4 now sees PIM hellos from XRv2. The network is still broken but all PEs inside AS 24 have now joined XRv2's MDT across the AS boundary.

```
RP/0/0/CPU0:XRv4#show pim vrf EIGRP neighbor | begin ^Neighbor
Neighbor Address      Interface      Uptime      Expires    DR pri    Flags
10.13.14.13           GigabitEthernet0/0/0/0.534 2d00h      00:01:30 1        B P
10.13.14.14*          GigabitEthernet0/0/0/0.534 2d14h      00:01:20 1 (DR) B E
13.0.0.12             mdtEIGRP      00:01:33   00:01:41 1        P
24.0.0.2              mdtEIGRP      00:43:31   00:01:25 1        P
24.0.0.14*           mdtEIGRP      00:43:48   00:01:17 1 (DR)
```

We see an angry syslog message on XRv4 generated by LPTS. Packets are being dropped but no details are revealed. The signaling appears to be functional so this may not be an issue. We will investigate this briefly to see if this will totally break the MVPN traffic or not.

```
! XRv4
%OS-LPTS-3-BAD_LISTENER_TAG : 'bad listener tag detected on the packet,
dropping packet'
```

With debugging, I try to find the reason for the drops. With an entry absent from the iFIB, this might be the reason why LPTS is dropping packets. The VRF ID maps to the default VRF, which means this is probably MDT related.

```
RP/0/0/CPU0:XRv4#debug lpts packet fast-path drops
RP/0/0/CPU0:XRv4#debug lpts packet slow-path drops

netio[309]: lpts decaps [0xb0c0eb14/124 md ?L3] IFIB lookup failed, dropping
VRF 0x60000000

RP/0/0/CPU0:XRv4#show lpts vrf
VRF-ID      VRF-NAME
0x00000001  *
0x60000000  default
0x60000001  **nVSatellite
0x60000002  EIGRP
```

Debugging LPTS packets on the slow-path, we can see this in an IPv6 PIM message. The output is verbose so be sure to send this to the log buffer. The packet is sourced from XRv2's MDT source towards the all-PIM-routers IPv6 multicast group of FF02::D. This occurs within the MDT, so XRv4 should have an iFIB entry to permit this traffic.

```
RP/0/0/CPU0:XRv4#debug lpts packet slow-path
netio[309]: lpts sub ifib/pifib [0xb0c0eb14/82 md VRF 0x60000000 IP6
::ffff:13.0.0.12 -> ff02::d p10] lookup successful IFH/VRF: 0x00001480 opcode
DELIVER, flow_type PIM-mcast-known, local flag 0, listener tag IPv6_STACK,
deliver
netio[309]: lpts pifib [0xb0c0eb14/82 md VRF 0x60000000 IP6 ::ffff:13.0.0.12
-> ff02::d p10] to local IPv6_STACK
```



```
netio[309]: lpts decaps [0xb0c0eb14/82 md VRF 0x60000000 IP6 ::ffff:13.0.0.12
-> ff02::d p10] to local stack (listener tag = IPv6_STACK)
netio[309]: %OS-LPTS-3-BAD_LISTENER_TAG : 'bad listener tag detected on the
packet, dropping packet'
netio[309]: lpts decaps [0xb0c0eb14/124 md ?L3] IFIB lookup failed, dropping
VRF 0x60000000
```

Checking the iFIB, we can clear see that PIM traffic destined to FF02::D is allowed from any source inside the MDT.

```
RP/0/0/CPU0:XRv4#show lpts ifib type raw6 brief | include PIM
RAWIP6    default  PIM    md          0/0/CPU0    ff02::d any
RAWIP6    *        PIM    Gi0/0/0/0.534 0/0/CPU0    ff02::d any
RAWIP6    default  PIM    any          0/0/CPU0    any any
RAWIP6    EIGRP    PIM    any          0/0/CPU0    any any
```

XRv4 also has a PIMv6 neighbor with XRv2 inside the VPN, which means the IPv6 PIM signaling is working properly as expected. I will assume this log message is the result of a lack of XRv support since everything appears functional.

```
RP/0/0/CPU0:XRv4#show pim vrf EIGRP ipv6 neighbor | begin ^mdt
mdtEIGRP
Neighbor Address                Uptime    Expires    DR pri    DR    Flags
::ffff:13.0.0.12                01:20:47  00:01:43  1          P
::ffff:24.0.0.2                 02:02:58  00:01:20  1
::ffff:24.0.0.14*               02:02:58  00:01:32  1          (DR)  P
```

XRv2 is still unable to join the trees of XRv2 and CSR2 as it has no route to those MDT endpoints. XR's inability to support PIM vector + RD means that XRv2 will need to perform RPF lookups for these endpoints. The same is true for core routers like CSR8 since there is no PIM vector coming from the PE anymore. To solve this semi-dynamically, I use BGP IPv4 multicast. I enable it between all neighbors inside AS 24 to start. No routes have been advertised into this AFI yet. This is definitely not within the spirit of option B but is a valid, dynamic workaround.

```
! XRv2
router bgp 13
 address-family ipv4 multicast

 af-group MCAST_V4 address-family ipv4 multicast
   route-reflector-client

 neighbor 13.0.0.5
   address-family ipv4 multicast
   use af-group MCAST_V4

 neighbor 13.0.0.8
```

```

address-family ipv4 multicast
  use af-group MCAST_V4

neighbor 13.0.0.11
  address-family ipv4 multicast
  use af-group MCAST_V4

! CSR5 and CSR8
router bgp 13
  address-family ipv4 multicast
  neighbor 13.0.0.12 activate

! XRv1
router bgp 13
  address-family ipv4 multicast
  neighbor 13.0.0.12
  address-family ipv4 multicast

```

Checking the RR, we can see that the AFI was successfully negotiated with all peers. No routes have been exchanged yet, as expected.

```

RP/0/0/CPU0:XRv2#show bgp ipv4 multicast summary | begin ^Neigh
Neighbor      Spk    AS  MsgRcvd  MsgSent    TblVer  InQ  OutQ  Up/Down  St/PfxRcd
13.0.0.5       0    13   26114    25087        2     0     0  00:00:48      0
13.0.0.8       0    13   25827    25086        2     0     0  00:00:39      0
13.0.0.11      0    13   24767    25034        2     0     0  00:00:26      0

```

I will use CSR5 as the ingress ASBR and CSR7 as the egress ASBR for this tree. As such, CSR5 will configure a static multicast route for 24.0.0.0/8 and advertise it into multicast BGP. This also implies we need to adjust the next-hop on CSR5 as the route is advertised towards XRv2 since AS 13 does not have reachability to the transit links.

```

! CSR5
ip mroute 24.0.0.0 255.0.0.0 10.5.7.7
router bgp 13
  address-family ipv4 multicast
  network 24.0.0.0
  neighbor 13.0.0.12 next-hop-self

```

```

R5#show ip route multicast static | begin Gate
Gateway of last resort is not set
S    24.0.0.0/8 [1/0] via 10.5.7.7

```

```

R5#show bgp ipv4 multicast | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
*>  24.0.0.0      10.5.7.7          0         32768 i

```

Quickly checking RPF on CSR8 and XRv2, we can see they now have a valid lookup for 24.0.0.2 and 24.0.0.14 inside AS 24. I use XRv14 as an example.

```
RP/0/0/CPU0:XRv2#show pim rpf 24.0.0.14
Table: IPv4-Multicast-default
* 24.0.0.14/32 [200/3]
    via GigabitEthernet0/0/0/0.582 with rpf neighbor 13.8.12.8

R8#show ip rpf 24.0.0.14
RPF information for ? (24.0.0.14)
  RPF interface: GigabitEthernet2.558
  RPF neighbor: ? (13.5.8.5)
  RPF route/mask: 24.0.0.0/8
  RPF type: multicast (bgp 13)
  Doing distance-preferred lookups across tables
  RPF topology: ipv4 multicast base, originated from ipv4 unicast base
```

Next, we trace the P(S,G) tree. XRv2 originates the P(S,G) join towards CSR8, who forwards it to CSR5. CSR8 clearly shows this as a multicast BGP RPF lookup. CSR7 shows it as a static multicast route as it was the originator of the BGP route for the rest of the AS. If PIM vector + RD is not supported, this would be the next best option, with intra-AS static multicast routes being the least preferred method.

```
P/0/0/CPU0:XRv2#show pim topology 232.13.24.255 24.0.0.14 | begin 232
(24.0.0.14,232.13.24.255)SPT SSM Up: 00:07:28
JP: Join(00:00:46) RPF: GigabitEthernet0/0/0/0.582,13.8.12.8 Flags:
  Loopback0                                00:07:28 fwd LI LH
```

```
R8#show ip mroute 232.13.24.255 24.0.0.14 | begin \
(24.0.0.14, 232.13.24.255), 00:02:52/00:02:36, flags: sT
  Incoming interface: GigabitEthernet2.558, RPF nbr 13.5.8.5, Mbgp
  Outgoing interface list:
    GigabitEthernet2.582, Forward/Sparse, 00:02:52/00:02:36
```

```
R5#show ip mroute 232.13.24.255 24.0.0.14 | begin \
(24.0.0.14, 232.13.24.255), 00:03:26/00:03:01, flags: sT
  Incoming interface: GigabitEthernet2.557, RPF nbr 10.5.7.7, Mroute
  Outgoing interface list:
    GigabitEthernet2.558, Forward/Sparse, 00:03:26/00:03:01
```

Checking CSR7 for both the CSR2 and XRv2 joins, we can see both. IGP dictates that both route via XRv4.

```
R7#show ip mroute 232.13.24.255 24.0.0.14 | begin \
(24.0.0.14, 232.13.24.255), 00:04:21/00:03:03, flags: sT
  Incoming interface: GigabitEthernet2.574, RPF nbr 24.7.14.14
  Outgoing interface list:
    GigabitEthernet2.557, Forward/Sparse, 00:04:21/00:03:03
```

```
R7#show ip mroute 232.13.24.255 24.0.0.2 | begin \
(24.0.0.2, 232.13.24.255), 00:04:31/00:02:54, flags: sT
Incoming interface: GigabitEthernet2.574, RPF nbr 24.7.14.14
Outgoing interface list:
GigabitEthernet2.557, Forward/Sparse, 00:04:31/00:02:54
```

Checking XRv4, it is the root of one tree, and a transit router for the other. CSR2 is the root of the second tree, so the MDT appears to be fully signaled now. Some of this multicast state already existed since CSR2 and XRv4 were already in the MDT together, but we verify it again for completeness.

```
RP/0/0/CPU0:XRv4#show pim topology 232.13.24.255 24.0.0.14 | begin 232
(24.0.0.14,232.13.24.255)SPT SSM Up: 02:18:48
JP: Join(00:00:47) RPF: Loopback0,24.0.0.14* Flags:
Loopback0 02:18:48 fwd LI LH
GigabitEthernet0/0/0/0.524 02:18:48 fwd Join(00:03:18)
GigabitEthernet0/0/0/0.574 00:05:32 fwd Join(00:02:52)
```

```
RP/0/0/CPU0:XRv4#show pim topology 232.13.24.255 24.0.0.2 | begin 232
(24.0.0.2,232.13.24.255)SPT SSM Up: 02:18:51
JP: Join(00:00:44) RPF: GigabitEthernet0/0/0/0.524,24.2.14.2 Flags:
Loopback0 02:18:51 fwd LI LH
GigabitEthernet0/0/0/0.574 00:05:35 fwd Join(00:02:48)
```

```
R2#show ip mroute 232.13.24.255 24.0.0.2 | begin \
(24.0.0.2, 232.13.24.255), 04:11:32/00:03:15, flags: sT
Incoming interface: Loopback0, RPF nbr 0.0.0.0
Outgoing interface list:
GigabitEthernet2.524, Forward/Sparse, 01:38:12/00:03:15
```

Last, we ensure that XRv2 can now see PIM hellos from XRv4 and CSR2, which allows the neighbors to form. This proves that the inter-AS MVPN signaling is working as expected.

```
RP/0/0/CPU0:XRv2#show pim vrf EIGRP neighbor | begin ^Neigh
Neighbor Address Interface Uptime Expires DR pri Flags
10.3.12.3 GigabitEthernet0/0/0/0.532 2d16h 00:01:30 1 P
10.3.12.12* GigabitEthernet0/0/0/0.532 2d16h 00:01:18 1 (DR) B P E
13.0.0.12* mdT EIGRP 03:56:28 00:01:20 1 P
24.0.0.2 mdT EIGRP 00:07:32 00:01:35 1 P
24.0.0.14 mdT EIGRP 00:07:29 00:01:16 1 (DR)
```

To test it, we can re-use the ASM group configured on XRv3 from earlier sections. XRv3 is joining 225.13.13.13 on its loopback and sending the C(*,G) join towards the RP, which is CSR3. The fact that XRv3 is learning the RP is a good indication that the default MDT is operational.

```
RP/0/0/CPU0:XRv3#show pim rp mapping
PIM Group-to-RP Mappings
Group(s) 224.0.0.0/4
```

RP 10.3.3.3 (?), v2

Info source: 10.1.13.1 (?), elected via bsr, priority 0, holdtime 150
Uptime: 01:53:08, expires: 00:01:46

RP/0/0/CPU0:XRv3#show igmp group 225.13.13.13

IGMP Connected Group Membership

Group Address	Interface	Uptime	Expires	Last Reporter
225.13.13.13	Loopback0	1d05h	never	10.13.13.13

XRv3, CSR1, and CSR2 all have this C(*,G) entry. CSR2 indicates that traffic is received from the PMSI, specifically from XRv2.

RP/0/0/CPU0:XRv3#show pim topology 225.13.13.13 | begin 225

(*,225.13.13.13) SM Up: 1d05h RP: 10.3.3.3

JP: Join(00:00:46) RPF: GigabitEthernet0/0/0/0.513,10.1.13.1 Flags: LH
Loopback0 1d05h fwd LI II LH

R1#show ip mroute 225.13.13.13 | begin \

(*, 225.13.13.13), 01:54:36/00:02:49, RP 10.3.3.3, flags: S

Incoming interface: GigabitEthernet2.512, RPF nbr 10.1.2.2

Outgoing interface list:

GigabitEthernet2.513, Forward/Sparse, 01:54:36/00:02:49

R2#show ip mroute vrf EIGRP 225.13.13.13 | begin \

(*, 225.13.13.13), 01:54:48/00:02:49, RP 10.3.3.3, flags: S

Incoming interface: Tunnel5, RPF nbr 13.0.0.12

Outgoing interface list:

GigabitEthernet2.512, Forward/Sparse, 00:18:23/00:02:49

XRv2 sends the C(*,G) join to CSR3, who is the root of the shared tree. This completes the C(*,G) signaling.

RP/0/0/CPU0:XRv2#show pim vrf EIGRP topology 225.13.13.13 | begin 225

(*,225.13.13.13) SM Up: 00:13:14 RP: 10.3.3.3

JP: Join(00:00:32) RPF: GigabitEthernet0/0/0/0.532,10.3.12.3 Flags:
mdtEIGRP 00:13:14 fwd Join(00:03:02)

R3#show ip mroute 225.13.13.13 | begin \

(*, 225.13.13.13), 00:14:14/00:03:12, RP 10.3.3.3, flags: S

Incoming interface: Null, RPF nbr 0.0.0.0

Outgoing interface list:

GigabitEthernet2.532, Forward/Sparse, 00:14:14/00:03:12

CSR3 is also the source for the group. We will not detail the entire c-mcast signaling process, to include PIM registration and SPT switchover. Instead, we will focus on the P(S,G) which has already been signaled and will not change as no data MDTs are configured.

```

R3#ping ip
Target IP address: 225.13.13.13
Repeat count [1]: 1000000
Datagram size [100]:
Timeout in seconds [2]: 1
Extended commands [n]: y
Interface [All]: loopback0
Time to live [255]:
Source address or interface: loopback0

```

CSR3 sends its first packet down the shared tree across the MDT, and XRv3 immediately switches to the SPT. For brevity, we will not verify packet counters at every single hop. Instead, we can look at XRv3's C(S,G) state. It now has a C(*,G) and C(S,G) for the group in question. The traffic is being processed locally so the OIL is empty.

```

RP/0/0/CPU0:XRv3#show pim topology 225.13.13.13 | begin 225
(*,225.13.13.13) SM Up: 1d05h RP: 10.3.3.3
JP: Join(00:00:05) RPF: GigabitEthernet0/0/0/0.513,10.1.13.1 Flags: LH
    Loopback0                1d05h      fwd LI II LH

(10.3.3.3,225.13.13.13) SPT SM Up: 00:01:43
JP: Join(00:00:05) RPF: GigabitEthernet0/0/0/0.513,10.1.13.1 Flags:
KAT(00:01:47) RA
    No interfaces in immediate olist

```

Checking the packet counters on XRv3, we can see one packet along the shared tree and several more along the SPT. This is because XRv3 immediately joined the SPT, which doesn't introduce any efficiencies in this topology.

```

RP/0/0/CPU0:XRv3#show mfib route 225.13.13.13 * | begin 225
(*,225.13.13.13),    Flags:  C
    Up: 1d05h
    Last Used: 00:02:54
    SW Forwarding Counts: 1/1/100
    SW Replication Counts: 1/0/0
    SW Failure Counts: 0/0/0/0/0
    Loopback0 Flags:  IC NS EG, Up:1d05h
    GigabitEthernet0/0/0/0.513 Flags:  A NS, Up:02:02:52

RP/0/0/CPU0:XRv3#show mfib route 225.13.13.13 10.3.3.3 | begin 225
(10.3.3.3,225.13.13.13),    Flags:
    Up: 00:02:35
    Last Used: 00:00:00
    SW Forwarding Counts: 140/140/14000
    SW Replication Counts: 140/0/0
    SW Failure Counts: 0/0/0/0/0
    Loopback0 Flags:  IC NS EG, Up:00:02:35

```

GigabitEthernet0/0/0/0.513 Flags: A, Up:00:02:35

There is one key component we have overlooked during this data transfer. When MVPN is enabled for a VRF, BGP will add a “connector attribute” to each VPN route. This allows the customer multicast traffic to pass RPF. For example, CSR2 is the egress MVPN router that decapsulates traffic along the MDT and forwards it to the CE. CSR2 uses the BGP route for RPF shown below. **The RPF rule of MDT states that the BGP next-hop MUST equal the MDT endpoint.** In this case, 10.5.6.5 is not the same as 13.0.0.12, so RPF would normally fail. Since this VPN route was originated by XRv2, there must be some mechanism to carry the original PE address as the BGP next-hop (and MPLS label) changes at the ASBR. The connector attribute is transitive for this reason and serves to carry the originating PE address.

```
R2#show bgp vpnv4 unicast vrf EIGRP 10.3.3.3/32
BGP routing table entry for 24:3:10.3.3.3/32, version 5646
Paths: (1 available, best #1, table EIGRP)
  Not advertised to any peer
  Refresh Epoch 1
  13, (Received from a RR-client), imported path from 13:3:10.3.3.3/32
(global)
  10.5.6.5 (metric 20) (via default) from 24.0.0.6 (24.0.0.6)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    Extended Community: RT:13:3 0x8800:32768:0 0x8801:3:288
      0x8802:65281:2560 0x8803:1:1500 0x8806:0:167971843
    Connector Attribute: count=1
      type 1 len 12 value 13:3:13.0.0.12
    mpls labels in/out nlabel/5072
    rx pathid: 0, tx pathid: 0x0
```

When this attribute is present, it is used instead of the BGP next-hop for the RPF check. Fortunately, this behavior is easy and automatic and requires no configuration. If the connector attribute was somehow stripped in transit, RPF would fail, and customer multicast traffic would be dropped at the egress PE. Note that this attribute is meaningless within an AS (assuming the next-hop does not change) since the BGP next-hop equals the originating PE anyway.

```
R2#show ip rpf vrf EIGRP 10.3.3.3
RPF information for ? (10.3.3.3)
  RPF interface: Tunnel5
  RPF neighbor: ? (13.0.0.12)
  RPF route/mask: 10.3.3.3/32
  RPF type: unicast (bgp 24)
  Doing distance-preferred lookups across tables
  BGP originator: 13.0.0.12
  RPF topology: ipv4 multicast base, originated from ipv4 unicast base
```

This concludes the PIM/GRE inter-AS option B lab. In summary, always plan to live without PIM vector + RD in this design if XR is in the multicast shortest path.

8.4.2.4 MVPN – mLDP (Profile 17)

To test mLDP between ASes, I will use profile 17. This uses BGP for auto-discovery, default MDTs using P2MP trees, and PIM for customer multicast signaling. First, we will prepare VRF OSPF on CSR2 and CSR8 for mLDP profile 17, ignoring the inter-AS requirement for now (which means the configuration is incomplete). As with all mLDP trees, the VPN ID must match as this is part of what is carried in the opaque field.

```
! CSR2 and CSR8
vrf definition OSPF
  vpn id 1300:2400
  address-family ipv4
    mdt preference mldp
    mdt auto-discovery mldp
    mdt default mpls mldp p2mp
  address-family ipv6
    mdt preference mldp
    mdt auto-discovery mldp
    mdt default mpls mldp p2mp
```

Since this profile relies on BGP MVPN AFI, we must configure those as well. The configuration is very simple but somewhat involved as we will configure it between PEs, RRs, and ASBRs. In this network, only XRv4 doesn't need to run these AFIs. We begin with AS 24.

```
! CSR2
router bgp 24
  address-family ipv4 mvpn
    neighbor 24.0.0.6 activate
    neighbor 24.0.0.6 send-community extended
    neighbor 24.0.0.6 route-reflector-client
    neighbor 24.0.0.7 activate
    neighbor 24.0.0.7 send-community extended
    neighbor 24.0.0.7 route-reflector-client
  address-family ipv6 mvpn
    neighbor 24.0.0.6 activate
    neighbor 24.0.0.6 send-community extended
    neighbor 24.0.0.6 route-reflector-client
    neighbor 24.0.0.7 activate
    neighbor 24.0.0.7 send-community extended
    neighbor 24.0.0.7 route-reflector-client
```

```
! CSR6 and CSR7
router bgp 24
  address-family ipv4 mvpn
    neighbor 24.0.0.2 activate
    neighbor 24.0.0.2 send-community extended
  address-family ipv6 mvpn
    neighbor 24.0.0.2 activate
```



```
neighbor 24.0.0.2 send-community extended
```

Once these are configured, we quickly verify the AFIs are properly negotiated with both ASBRs for IPv4 and IPv6. No routes are being received from the ASBRs yet.

```
R2#show bgp ipv4 mvpn all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent   TblVer   InQ OutQ Up/Down   State/PfxRcd
24.0.0.6      4      24      27     68       1    0    0 00:00:57      0
24.0.0.7      4      24      21     47       1    0    0 00:00:36      0
```

```
R2#show bgp ipv6 mvpn all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent   TblVer   InQ OutQ Up/Down   State/PfxRcd
24.0.0.6      4      24      28     69       1    0    0 00:01:02      0
24.0.0.7      4      24      22     48       1    0    0 00:00:40      0
```

CSR2 is currently originating a single type-1 Intra-AS I-PMSI route. This is used to build the default MDTs with other PEs in a given MVPN instance. One is created for both IPv4 and IPv6.

```
R2#show bgp ipv4 mvpn all | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 24:2 (default for vrf OSPF)
*>  [1] [24:2] [24.0.0.2]/12
      0.0.0.0      32768 ?
```

```
R2#show bgp ipv6 mvpn all | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 24:2 (default for vrf OSPF)
*>  [1] [24:2] [24.0.0.2]/12
      ::      32768 ?
```

We can confirm that CSR6 and CSR7 learn these prefixes. Since CSR6 has VRF OSPF configured locally, we can reference the route via the VRF table to view the details. These routes have RTs just like VPNv4 routes, so the option B consideration of ASBR RT retention matter for MVPN AFIs. Notice that the community of no-export is also set, which effectively makes this an intra-AS or intra-confederation AD route. PMSI tunnel type 2 represents mLDP P2MP and the tunnel root is 24.0.0.2, which is carried in the tunnel parameters.

```
R6#show bgp ipv4 mvpn vrf OSPF route-type 1 24.0.0.2
BGP routing table entry for [1][24:2][24.0.0.2]/12, version 4
Paths: (1 available, best #1, table MVPNv4-BGP-Table, not advertised to EBGp
peer)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    24.0.0.2 (metric 20) from 24.0.0.2 (24.0.0.2)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Community: no-export
```

```
Extended Community: RT:24:2
PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 17, label: exp-null,
tunnel parameters: 0600 0104 1800 0002 0007 0100 0400 0200 00
rx pathid: 0, tx pathid: 0x0
```

For variety, we check CSR7 for the IPv6 MVPN I-PMSI route. It does not exist; CSR7 must retain the routes somehow. This was not a consideration with IPv4 MDT since the MDT address is what really determines MVPN membership in that design.

```
R7#show bgp ipv6 mvpn rd 24:2 route-type 1 24.0.0.2
% Network not in table
```

Since CSR7 configured CSR2 as an RR-client for other AFIs as a workaround, we will use that technique again for this AFI. We could have also configured the VRFs locally or instructed the ASBR to disable the default RT-filter.

```
! CSR7
router bgp 24
 address-family ipv4 mvpn
   neighbor 24.0.0.2 route-reflector-client
 address-family ipv6 mvpn
   neighbor 24.0.0.2 route-reflector-client
```

When the session comes back up, CSR7 has the proper MVPN routes. The IPv6 route has identical characteristics as the IPv4 route, but I display the details for completeness.

```
R7#show bgp ipv6 mvpn rd 24:2 route-type 1 24.0.0.2
BGP routing table entry for [1][24:2][24.0.0.2]/12, version 5
Paths: (1 available, best #1, table MVPNV6-BGP-Table, not advertised to EBG
peer)
  Not advertised to any peer
  Refresh Epoch 2
  Local, (Received from a RR-client)
    24.0.0.2 (metric 20) from 24.0.0.2 (24.0.0.2)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Community: no-export
      Extended Community: RT:24:2
      PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 17, label: exp-null,
      tunnel parameters: 0600 0104 1800 0002 0007 0100 0400 0300 00
      rx pathid: 0, tx pathid: 0x0
```

Next, we will configure AS 13 similarly. Both ASBRs will retain all RTs rather than use the workarounds employed in AS 24. XRv2 negotiates this capability with all routes in AS 13. This is configuration intensive but very simple.

```
! XRv2
```

```

router bgp 13
  address-family ipv4 mvpn
  address-family ipv6 mvpn

  af-group MVPNV4 address-family ipv4 mvpn
    route-reflector-client
  af-group MVPNV6 address-family ipv6 mvpn
    route-reflector-client

  neighbor 13.0.0.5
    address-family ipv4 mvpn
      use af-group MVPNV4
    address-family ipv6 mvpn
      use af-group MVPNV6

  neighbor 13.0.0.8
    address-family ipv4 mvpn
      use af-group MVPNV4
    address-family ipv6 mvpn
      use af-group MVPNV6

  neighbor 13.0.0.11
    address-family ipv4 mvpn
      use af-group MVPNV4
    address-family ipv6 mvpn
      use af-group MVPNV6

! CSR8
router bgp 13
  address-family ipv4 mvpn
    neighbor 13.0.0.12 activate
  address-family ipv6 mvpn
    neighbor 13.0.0.12 activate

! CSR5
router bgp 13
  address-family ipv4 mvpn
    no bgp default route-target filter
    neighbor 13.0.0.12 activate
  address-family ipv6 mvpn
    no bgp default route-target filter
    neighbor 13.0.0.12 activate

! XRv1
router bgp 13
  address-family ipv4 mvpn
    retain route-target all

  address-family ipv6 mvpn

```

```
retain route-target all

neighbor 13.0.0.12
address-family ipv4 mvpn
address-family ipv6 mvpn
```

Quickly checking XRv2, we can see that all sessions are up. XRv2 learns 2 routes from CSR8, which is odd as we would expect it to only learn one.

```
RP/0/0/CPU0:XRv2#show bgp ipv4 mvpn summary | begin ^Neigh
Neighbor      Spk    AS  MsgRcvd  MsgSent    TblVer  InQ  OutQ  Up/Down    St/PfxRcd
13.0.0.5       0    13   26773    25781       3     0     0  00:04:16      0
13.0.0.8       0    13   26388    25790       3     0     0  00:04:43      2
13.0.0.11      0    13   25392    25736       3     0     0  00:02:52      0

RP/0/0/CPU0:XRv2#show bgp ipv6 mvpn summary | begin ^Neigh
Neighbor      Spk    AS  MsgRcvd  MsgSent    TblVer  InQ  OutQ  Up/Down    St/PfxRcd
13.0.0.5       0    13   26773    25781       3     0     0  00:04:19      0
13.0.0.8       0    13   26389    25790       3     0     0  00:04:46      2
13.0.0.11      0    13   25392    25736       3     0     0  00:02:55      0
```

The reason for learning two I-PMSI routes is because of the central-services VPN. VRF OSPF imports VRF BGP's exported RT, which means that it will create an I-PMSI route with VRF BGP's RD. This is harmless.

```
RP/0/0/CPU0:XRv2#show bgp ipv4 mvpn | begin Network
Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 13:1
*>i[1][13.0.0.8]/40  13.0.0.8      0      100      0 ?
Route Distinguisher: 13:2
*>i[1][13.0.0.8]/40  13.0.0.8      0      100      0 ?
```

Both XRv1 and CSR5 successfully learn these routes, thanks to the RT filter being disabled. The key details, such as the BGP next-hop, communities, tunnel type (mLDP P2MP) and tunnel root (13.0.0.8) are all highlighted. These messages look almost identical to the ones in AS 24 except with different IP addressing.

```
RP/0/0/CPU0:XRv1#show bgp ipv4 mvpn rd 13:2 [1][13.0.0.8]/40
BGP routing table entry for [1][13.0.0.8]/40, Route Distinguisher: 13:2
Versions:
  Process          bRIB/RIB    SendTblVer
  Speaker           3           3
Paths: (1 available, best #1, not advertised to EBGp peer)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  Local
    13.0.0.8 (metric 2) from 13.0.0.12 (13.0.0.8)
```

```

Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, not-in-vrf
Received Path ID 0, Local Path ID 1, version 3
Community: no-export
Extended community: RT:13:2
Originator: 13.0.0.8, Cluster list: 13.0.0.12
PMSI: flags 0x00, type 2, label 0, ID
0x060001040d000008000701000400020000

```

```

R5#show bgp ipv6 mvpn rd 13:2 route-type 1 13.0.0.8
BGP routing table entry for [1][13:2][13.0.0.8]/12, version 5
Paths: (1 available, best #1, table MVPNV6-BGP-Table, not advertised to EBGp
peer)
Not advertised to any peer
Refresh Epoch 1
Local
13.0.0.8 (metric 2) from 13.0.0.12 (13.0.0.12)
Origin incomplete, metric 0, localpref 100, valid, internal, best
Community: no-export
Extended Community: RT:13:2
Originator: 13.0.0.8, Cluster list: 13.0.0.12
PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 17, label: exp-null,
tunnel parameters: 0600 0104 0D00 0008 0007 0100 0400 0400 00
rx pathid: 0, tx pathid: 0x0

```

Next, we will configure the inter-AS peers. For brevity, I only show the configuration on XRv1 and CSR6.

```

! XRv1
router bgp 13
neighbor 10.6.11.6
address-family ipv4 mvpn
route-policy RPL_PASS in
route-policy RPL_PASS out
address-family ipv6 mvpn
route-policy RPL_PASS in
route-policy RPL_PASS out

! CSR6
address-family ipv4 mvpn
neighbor 10.5.6.5 activate
neighbor 10.6.11.11 activate
address-family ipv6 mvpn
neighbor 10.5.6.5 activate
neighbor 10.6.11.11 activate

```

Once all the peers are configured, we perform a quick verification on CSR5 and CSR6 to ensure all peers come up. The problem at this point is clear; no MVPN routes are being exchanged between ASes. The type-1 I-PMSI routes are intra-AS only which is why BGP automatically applies to no-export community.

We saw this earlier on both IPv4 and IPv6 MVPN routes, and this explains why there is no eBGP MVPN route exchange.

```
R6#show bgp ipv4 mvpn all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State/PfxRcd
10.5.6.5      4      13      53     49       4    0   0 00:00:38      0
10.6.11.11    4      13      19     81       4    0   0 00:01:13      0
24.0.0.2      4      24     387    290       4    0   0 00:28:59      1
```

```
R6#show bgp ipv6 mvpn all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State/PfxRcd
10.5.6.5      4      13      53     50       4    0   0 00:00:59      0
10.6.11.11    4      13      19     82       4    0   0 00:01:35      0
24.0.0.2      4      24     389    292       4    0   0 00:29:20      1
```

```
R5#show bgp ipv4 mvpn all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State/PfxRcd
10.5.6.6      4      24      50     53       5    0   0 00:00:49      0
10.5.7.7      4      24      36     46       5    0   0 00:00:23      0
13.0.0.12     4      13     230    160       5    0   0 00:12:54      2
```

```
R5#show bgp ipv6 mvpn all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State/PfxRcd
10.5.6.6      4      24      50     53       5    0   0 00:01:10      0
10.5.7.7      4      24      47     46       5    0   0 00:00:45      0
13.0.0.12     4      13     232    162       5    0   0 00:13:16      2
```

To solve this, we can add the optional “inter-as” modifier to the BGP AD configuration under the VRF. From the PEs perspective, this actually does not create a different I-PMSI route; it simply removes the “no-export” community from the type-1 route. Checking CSR2’s local route, the lack of the “no-export” community is the only difference. Only the RT remains.

```
! CSR2 and CSR8
vrf definition OSPF
 address-family ipv4
  mdt auto-discovery mldp inter-as
 address-family ipv6
  mdt auto-discovery mldp inter-as
```

```
R2#show bgp ipv4 mvpn rd 24:2 route-type 1 24.0.0.2
BGP routing table entry for [1][24:2][24.0.0.2]/12, version 11
Paths: (1 available, best #1, table MVPNv4-BGP-Table)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (24.0.0.2)
      Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
  best
    Extended Community: RT:24:2
```

```

PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 17, label: exp-null,
tunnel parameters: 0600 0104 1800 0002 0007 0100 0400 0200 00
rx pathid: 0, tx pathid: 0x0

```

CSR2 also has the remote I-PMSI route from CSR8 with RD 13:2. This has been successfully imported into VRF OSPF. The fact that it received a copy from both CSR6 and CSR7 indicates that both ASBRs are passing routes properly. Notice their neither CSR6 nor CSR7 changed the eBGP next-hop when advertising it to CSR (iBGP peer), which is expected.

```

R2#show bgp ipv4 mvpn rd 13:2 route-type 1 13.0.0.8
BGP routing table entry for [1][13:2][13.0.0.8]/12, version 18
Paths: (2 available, best #1, table MVPNV4-BGP-Table)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  13, (Received from a RR-client)
    10.5.6.5 (metric 20) from 24.0.0.6 (24.0.0.6)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:13:2
      PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 17, label: exp-null,
      tunnel parameters: 0600 0104 0D00 0008 0007 0100 0400 0200 00
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  13, (Received from a RR-client)
    10.5.7.5 (metric 20) from 24.0.0.7 (24.0.0.7)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Extended Community: RT:13:2
      PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 17, label: exp-null,
      tunnel parameters: 0600 0104 0D00 0008 0007 0100 0400 0200 00
      rx pathid: 0, tx pathid: 0

```

CSR8 does not have CSR2's I-PMSI routes. Checking XRv2, we can see this is the result (again) of failing to set next-hop-self on the ASBRs. Neither path is the bestpath as a result.

```

R8#show bgp ipv4 mvpn rd 24:2
[no output]

```

```

RP/0/0/CPU0:XRv2#show bgp ipv4 mvpn rd 24:2 | begin Network
      Network                Next Hop                Metric LocPrf Weight Path
Route Distinguisher: 24:2
* i [1][24.0.0.2]/40        10.5.6.6                0      100      0 24 ?
* i                          10.6.11.6                100     100      0 24 ?

```

After making the adjustment on XRv1 and CSR5, CSR8 learns one copy of the route (the bestpath from the RR).

```

! CSR5

```

```

address-family ipv4 mvpn
  neighbor 13.0.0.12 next-hop-self
address-family ipv6 mvpn
  neighbor 13.0.0.12 next-hop-self

```

```

! XRv1
router bgp 13
  neighbor 13.0.0.12
    address-family ipv4 mvpn
      next-hop-self
    address-family ipv6 mvpn
      next-hop-self

```

```

R8#show bgp ipv4 mvpn rd 24:2 | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 24:2
*>i [1][24:2][24.0.0.2]/12
      13.0.0.5          0          100          0 24 ?

```

When we check the mLDP database, we see several dynamically-discovered P2MP trees. I will look at the first tree rooted in CSR8. CSR2 should be a leaf in this tree, and the upstream client should point towards 13.0.0.8.

```

R2#show mpls mldp database summary

```

LSM ID Cnt.	Type	Root	Decoded Opaque Value	Client
B	P2MP	13.0.0.8	[gid 131072 (0x00020000)]	1
5	P2MP	24.0.0.2	[gid 131072 (0x00020000)]	1
6	P2MP	24.0.0.2	[gid 196608 (0x00030000)]	1
C	P2MP	13.0.0.8	[gid 262144 (0x00040000)]	1

However, we see no output for the upstream client. CSR2 has no idea how to reach 13.0.0.8, the root of the tree, and this cannot send label mapping messages towards it. The same is true for CSR8 trying to reach 24.0.0.2.

```

R2#show mpls mldp database id B
LSM ID : B   Type: P2MP   Uptime : 00:08:26
  FEC Root      : 13.0.0.8
  Opaque decoded : [gid 131072 (0x00020000)]
  Opaque length  : 4 bytes
  Opaque value   : 01 0004 00020000
  Upstream client(s) :
    None
    Expires      : N/A          Path Set ID   : B
  Replication client(s):
    MDT (VRF OSPF)
    Uptime       : 00:08:26     Path Set ID   : None

```


Interface : Lspvif1

```
R8#show ip cef 24.0.0.2
0.0.0.0/0
no route
```

```
R2#show ip cef 13.0.0.8
0.0.0.0/0
no route
```

A temporary static route shows what the “proper” output would look like. A downstream label is allocated on CSR2 and advertised to CSR7 so that LSM can be received along this P2MP tree. The static route is immediately removed after this output is displayed.

```
! CSR2
ip route 13.0.0.8 255.255.255.255 24.2.7.7

R2#show mpls mldp database id B
LSM ID : B    Type: P2MP    Uptime : 00:09:37
FEC Root      : 13.0.0.8
Opaque decoded : [gid 131072 (0x00020000)]
Opaque length  : 4 bytes
Opaque value   : 01 0004 00020000
Upstream client(s) :
  24.0.0.7:0    [Active]
    Expires      : Never          Path Set ID   : B
    Out Label (U) : None          Interface    : GigabitEthernet2.527*
    Local Label (D) : 2111        Next Hop      : 24.2.7.7
Replication client(s):
  MDT (VRF OSPF)
    Uptime       : 00:09:37      Path Set ID   : None
    Interface    : Lspvif1
```

Inter-AS option B with non-segmented (that is, end-to-end) mLDP trees is not supported on XE at this time. The “proper” way to do this configuration would be to use segmented trees with the ASBRs as the stitching points, much like L2VPN. On real XR platforms (not XRV), this is supported and is the point at which the type-2 inter-AS I-PMSI routes are created. Even though the type-1 I-PMSI routes were exchanged, the core trees cannot be built. The routers need real unicast routes (not just RPF fixup) towards the FEC root per tree. Of course, we know this is very bad design for option B. Introducing loopback leaking this late in the option B test will invalidate many of the other tasks we have accomplished. However, we will complete this style of design when testing option C since leaking PE loopbacks is required for all MPLS services. Cisco recommends using profile 0 as we demonstrated earlier, along with the PIM vector + RD, to support inter-AS MVPN on XE routers with option B. In the event the feature is ever supported, one can continue this lab based on the partial configuration.

8.4.2.5 MPLS TE

This section details how to configure inter-AS TE with option B. Somewhat like the failed MVPN mLDPLab for option B, MPLS TE is a little unwieldy because it assumes the ASes are aware of one another's loopbacks. It does technically "work" with option B, but from a design perspective, it makes more sense in option C. TE LSPs can be signaled but their utility is very limited. The operation of inter-AS TE is almost identical to inter-area/inter-level TE examined in the Unified MPLS section. It is covered in detail here as well, but essentially, loose hop path expansion is used on each ASBR to stitch a patch from head to tail.

The configuration is very simple. First, we must enable MPLS TE on the transit interfaces on all routers. On the XE routers, we identify the remote ASBR peer as a "passive" neighbor. This feature is not supported on XR and we will use an alternative approach discussed later.

```
! CSR6
interface GigabitEthernet2.556
  mpls traffic-eng tunnels
  mpls traffic-eng passive-interface nbr-te-id 13.0.0.5 nbr-if-addr 10.5.6.5
  ip rsvp bandwidth 200000

interface GigabitEthernet2.561
  mpls traffic-eng tunnels
  mpls traffic-eng passive-interface nbr-te-id 13.0.0.11 nbr-if-addr
10.6.11.11
  ip rsvp bandwidth 200000

! XRv1
rsvp
  interface GigabitEthernet0/0/0/0.561
    bandwidth 200000

mpls traffic-eng
  interface GigabitEthernet0/0/0/0.561

! CSR5
interface GigabitEthernet2.556
  mpls traffic-eng tunnels
  mpls traffic-eng passive-interface nbr-te-id 24.0.0.6 nbr-if-addr 10.5.6.6
  ip rsvp bandwidth 200000

interface GigabitEthernet2.557
  mpls traffic-eng tunnels
  mpls traffic-eng passive-interface nbr-te-id 24.0.0.7 nbr-if-addr 10.5.7.7
  ip rsvp bandwidth 200000

! CSR7
interface GigabitEthernet2.557
  mpls traffic-eng tunnels
  mpls traffic-eng passive-interface nbr-te-id 13.0.0.5 nbr-if-addr 10.5.7.5
  ip rsvp bandwidth 200000
```

Once this is configured, we can verify the TED in each AS. Beginning with AS 24, we will use the same cursory check seen in the initial option B verification. There are three new links in the TED that are a result of the MPLS TE passive-interfaces configured on the ASBRs. The links are highlighted below; there is no internal neighbor node ID for these peers, so the value is sent to $2^{32} - 1$ as a way of signaling a null value. The IS-IS system ID within the NET is the peer TE ID encoded into the first 4 bytes of the system ID. Since these are valid links in the graph, PCALC can consider them for TE LSPs.

```
R2#show mpls traffic-eng topology brief | include IGP Id
IGP Id: 0000.0000.0002.00, MPLS TE Id:24.0.0.2 Router Node (isis level-2)
    link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0007.00, nbr_node_id:2, gen:18
    link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0014.00, nbr_node_id:4, gen:18
IGP Id: 0000.0000.0006.00, MPLS TE Id:24.0.0.6 Router Node (isis level-2)
    link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0014.00, nbr_node_id:4, gen:27
    link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0007.00, nbr_node_id:2, gen:27
    link[2]: Point-to-Point, Nbr IGP Id: 0D00.0005.0000.00, nbr_node_id:4294967295,
gen:27
    link[3]: Point-to-Point, Nbr IGP Id: 0D00.000B.0000.00, nbr_node_id:4294967295,
gen:27
IGP Id: 0000.0000.0007.00, MPLS TE Id:24.0.0.7 Router Node (isis level-2)
    link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0002.00, nbr_node_id:1, gen:28
    link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0006.00, nbr_node_id:3, gen:28
    link[2]: Point-to-Point, Nbr IGP Id: 0D00.0005.0000.00, nbr_node_id:4294967295,
gen:28
    link[3]: Point-to-Point, Nbr IGP Id: 0000.0000.0014.00, nbr_node_id:4, gen:28
IGP Id: 0000.0000.0014.00, MPLS TE Id:24.0.0.14 Router Node (isis level-2)
    link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0002.00, nbr_node_id:1, gen:24
    link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0006.00, nbr_node_id:3, gen:24
    link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0007.00, nbr_node_id:2, gen:24
```

Inside AS 13, we now see 16 opaque-area LSAs versus 14 from before. Ideally, we would have seen 17; this is where XR fails to support inter-AS TE. The transit link between XRv1 and CSR6 is not visible to AS 13 as a result of this XR limitation.

```
R8#show ip ospf 13 0 database database-summary
```

```
OSPF Router with ID (13.0.0.8) (Process ID 13)
```

```
Area 0 database summary
```

LSA Type	Count	Delete	Maxage
Router	4	0	0
Network	0	0	0
Summary Net	0	0	0
Summary ASBR	0	0	0
Type-7 Ext	0	0	0
Prefixes redistributed in Type-7 0			
Opaque Link	0	0	0
Opaque Area	16	0	0
Subtotal	20	0	0

We can see the two inter-AS links on CSR5, along with the statically-configured neighbor IGP IDs.

```
R8#show mpls traffic-eng topology brief | include IGP Id
IGP Id: 13.0.0.5, MPLS TE Id:13.0.0.5 Router Node (ospf 13 area 0)
  link[0]: Point-to-Point, Nbr IGP Id: 13.0.0.11, nbr_node_id:3, gen:12
  link[1]: Point-to-Point, Nbr IGP Id: 13.0.0.8, nbr_node_id:1, gen:12
  link[2]: Point-to-Point, Nbr IGP Id: 24.0.0.6, nbr_node_id:4294967295, gen:12
  link[3]: Point-to-Point, Nbr IGP Id: 24.0.0.7, nbr_node_id:4294967295, gen:12
IGP Id: 13.0.0.8, MPLS TE Id:13.0.0.8 Router Node (ospf 13 area 0)
  link[0]: Point-to-Point, Nbr IGP Id: 13.0.0.5, nbr_node_id:2, gen:10
  link[1]: Point-to-Point, Nbr IGP Id: 13.0.0.12, nbr_node_id:4, gen:10
  link[2]: Point-to-Point, Nbr IGP Id: 13.0.0.11, nbr_node_id:3, gen:10
IGP Id: 13.0.0.11, MPLS TE Id:13.0.0.11 Router Node (ospf 13 area 0)
  link[0]: Point-to-Point, Nbr IGP Id: 13.0.0.12, nbr_node_id:4, gen:6
  link[1]: Point-to-Point, Nbr IGP Id: 13.0.0.5, nbr_node_id:2, gen:6
  link[2]: Point-to-Point, Nbr IGP Id: 13.0.0.8, nbr_node_id:1, gen:6
IGP Id: 13.0.0.12, MPLS TE Id:13.0.0.12 Router Node (ospf 13 area 0)
  link[0]: Point-to-Point, Nbr IGP Id: 13.0.0.11, nbr_node_id:3, gen:5
  link[1]: Point-to-Point, Nbr IGP Id: 13.0.0.8, nbr_node_id:1, gen:5
```

Just like with inter-area TE, we can accomplish inter-AS TE with two different mentalities. The first is tunnel stitching, which similar to the L2VPN MSPW, would result in intra-AS TE tunnels that terminate on the ASBRs. We would optionally configure a one-hop tunnel between ASBRs but that doesn't make much sense. To demonstrate this capability, I will create a TE tunnel from CSR2 to CSR6 over the high-cost path via CSR7. This will fully replace the LDP label normally imposed by CSR2 for transport across AS 24. The reason the tunnel should go to CSR6 and not CSR7 is because CSR2's VPN routes prefer CSR6 as the egress ASBR. CSR2 selected CSR6 due to having a lower BGP RID, so tunneling traffic to CSR7 would not be effective. We will test connectivity to a central services route.

```
R2#show bgp vpnv4 unicast vrf OSPF 110.0.0.0
BGP routing table entry for 24:2:110.0.0.0/32, version 6424
Paths: (1 available, best #1, table OSPF)
  Not advertised to any peer
  Refresh Epoch 1
  13 100, (Received from a RR-client), imported path from 13:1:110.0.0.0/32
(global)
  10.5.6.5 (metric 20) (via default) from 24.0.0.6 (24.0.0.6)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    Extended Community: RT:13:1
    mpls labels in/out nolabel/5008
    rx pathid: 0, tx pathid: 0x0
```

The tunnel configuration is very straightforward using explicit-paths and autoroute. Remember that this will place all IP traffic towards 24.0.0.6/32 into the tunnel, which may include L2VPN traffic.

```
! CSR2
ip explicit-path name EP_2_7_14_6 enable
```

```
next-address 24.0.0.7
next-address 24.0.0.14
next-address 24.0.0.6
```

```
interface Tunnel200
description INTRA-AS TO CSR6
ip unnumbered Loopback0
tunnel mode mpls traffic-eng
tunnel destination 24.0.0.6
tunnel mpls traffic-eng autoroute announce
tunnel mpls traffic-eng path-option 10 explicit name EP_2_7_14_6
```

We quickly check the status of the tunnel to ensure it is up, then use MPLS traceroute to check the data plane. For additional details on MPLS TE, check the dedicated chapter. These TE examples will focus only on the inter-AS components.

```
R2#show mpls traffic-eng tunnels tunnel 200 brief | begin TUNNEL
TUNNEL NAME          DESTINATION          UP IF    DOWN IF    STATE/PROT
INTRA-AS TO CSR6     24.0.0.6             -        Gi2.527    up/up
```

```
R2#traceroute mpls traffic-eng tunnel 200
Tracing MPLS TE Label Switched Path on Tunnel200, timeout is 2 seconds
[snip]
Type escape sequence to abort.
 0 24.2.7.2 MRU 1500 [Labels: 7041 Exp: 0]
L 1 24.2.7.7 MRU 1500 [Labels: 94003 Exp: 0] 8 ms
L 2 24.7.14.14 MRU 1500 [Labels: implicit-null Exp: 0] 2 ms
! 3 24.6.14.6 4 ms
```

As soon as we build this tunnel, VPN connectivity inside both VPNs is broken. Both VPNs are ultimately depending on CSR6 as their egress point, but the tunnel is “unusable”.

```
R2#show ip cef vrf EIGRP 10.3.3.3
10.3.3.3/32
  nexthop 24.0.0.6 Tunnel200 unusable: no label
```

```
R2#show ip cef vrf OSPF 10.4.4.4
10.4.4.4/32
  nexthop 24.0.0.6 Tunnel200 unusable: no label
```

The output is a bit misleading because clearly the tunnel has an associated RSVP label; we proved it with MPLS OAM. If we follow the route recursion more closely, we can reveal the error via a manual tracing procedure. Inside the EIGRP VPN as an example, the VPN next-hop is XRv1’s transit link interface towards CSR6. This is because CSR6 did not adjust the next-hop and instead redistributed the host route 10.6.11.11/32 into IGP.

```

R2#show bgp vpnv4 unicast vrf EIGRP 10.3.3.3
BGP routing table entry for 24:3:10.3.3.3/32, version 6288
Paths: (1 available, best #1, table EIGRP)
  Not advertised to any peer
  Refresh Epoch 1
  13, (Received from a RR-client), imported path from 13:3:10.3.3.3/32
(global)
  10.6.11.11 (metric 20) (via default) from 24.0.0.6 (24.0.0.6)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    Extended Community: RT:13:3 0x8800:32768:0 0x8801:3:288
      0x8802:65281:2560 0x8803:1:1500 0x8806:0:167971843
    Connector Attribute: count=1
      type 1 len 12 value 13:3:13.0.0.12
    mpls labels in/out nlabel/91010
    rx pathid: 0, tx pathid: 0x0

```

The route is learned via a TE tunnel, so one might think the RSVP label can be imposed now. This is false and is explained below.

```

R2#show ip route 10.6.11.11
Routing entry for 10.6.11.11/32
  Known via "isis", distance 115, metric 20, type level-2
  Redistributing via isis 24
  Last update from 24.0.0.6 on Tunnel200, 00:06:31 ago
  Routing Descriptor Blocks:
    * 24.0.0.6, from 24.0.0.6, 00:06:31 ago, via Tunnel200
      Route metric is 20, traffic share count is 1

```

When we first configured CSR6 to redistribute those transit links, we noticed that CSR6 allocated non-null labels for them. Despite being connected host-routes, they are not local routes, so LDP treats them as if they were IGP learned. As such, the BGP next-hop is not the same LSR as the tunnel destination. To solve this, we require a third label between the transport RSVP label and the BGP VPN label. This is normally achieved with tLDP running across the TE tunnel. If CSR2 can learn CSR6's local label for 10.6.11.11/32, it can push that label second in the stack, which allows CSR6 to switch the packet to XRv1. CSR6 cannot swap the BGP label since it did not allocate it (did not change the BGP next-hop). Instead, this third label allows us the tunnel the VPN label outside of the AS so that XRv1 can perform the swap. CSR6 must accept LDP targeted sessions (it already is to support VPLS but I show the configuration again) and CSR2 must enable tLDP on the tunnel.

```

! CSR6
mpls ldp discovery targeted-hello accept

! CSR2
interface Tunnel200
  mpls ip

```

When the tLDP session forms, CSR2 can learn CSR6's label for 10.6.11.11/32 and push it onto the label stack above the VPN label. The TE label is added last and is not shown in the CEF output. The tunnel is now usable and unicast connectivity should be restored.

```
R2#show mpls ldp bindings 10.6.11.11 32 neighbor 24.0.0.6
  lib entry: 10.6.11.11/32, rev 43
    remote binding: lsr: 24.0.0.6:0, label: 6005
```

```
R2#show ip cef vrf EIGRP 10.3.3.3
10.3.3.3/32
  nexthop 24.0.0.6 Tunnel200 label 6005 91010
```

```
R2#show ip cef vrf OSPF 10.4.4.4
10.4.4.4/32
  nexthop 24.0.0.6 Tunnel200 label 6005 91007
```

A traceroute inside VRF EIGRP proves this; we can see 3 labels in the stack while the packet transits CSR7 and XRv4. We can see that label 91010, the VPN label for 10.3.3.3/32, is tunneled all the way to XRv1.

```
R1#traceroute 10.3.3.3 source 10.1.1.1
Type escape sequence to abort.
Tracing the route to 10.3.3.3
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.2.2 6 msec 4 msec 4 msec
 2 24.2.7.7 [MPLS: Labels 7041/6005/91010 Exp 0] 12 msec 12 msec 11 msec
 3 24.7.14.14 [MPLS: Labels 94003/6005/91010 Exp 0] 21 msec 31 msec 32 msec
 4 24.6.14.6 [MPLS: Labels 6005/91010 Exp 0] 31 msec 31 msec 31 msec
 5 10.6.11.11 [MPLS: Label 91010 Exp 0] 30 msec 31 msec 37 msec
 6 13.8.11.8 [MPLS: Labels 8000/92002 Exp 0] 31 msec 31 msec 39 msec
 7 13.8.12.12 [MPLS: Label 92002 Exp 0] 45 msec 10 msec 10 msec
 8 10.3.12.3 16 msec 12 msec 11 msec
```

Upon receipt, XRv1 may want to put this traffic into a TE tunnel as well. Below is a basic TE tunnel to stitch traffic from the ASBR to the egress PE.

```
! XRv1
explicit-path name EP_11_5_8_12
  index 10 next-address strict ipv4 unicast 13.0.0.5
  index 20 next-address strict ipv4 unicast 13.0.0.8
  index 30 next-address strict ipv4 unicast 13.0.0.12

interface tunnel-te300
  description INTRA AS TO XRv2
  ipv4 unnumbered Loopback0
  autoroute announce
  destination 13.0.0.12
  path-option 10 explicit name EP_11_5_8_12
```

Unlike AS 24, XRv1 does not have to worry about running tLDP over this TE tunnel. Since the final destination is the tunnel target (that is to say, the LDP label would have been a null-label anyway), a third label is not required. We verify that the tunnel comes up on XRv1, then verify the data plane with MPLS traceroute.

```
RP/0/0/CPU0:XRv1#show mpls traffic-eng tunnels brief
              TUNNEL NAME          DESTINATION      STATUS  STATE
              tunnel-te200         13.0.0.12        up    up
Displayed 1 (of 1) heads, 0 (of 0) midpoints, 0 (of 0) tails
Displayed 1 up, 0 down, 0 recovering, 0 recovered heads

RP/0/0/CPU0:XRv1#traceroute mpls traffic-eng tunnel-te 200
Tracing MPLS TE Label Switched Path on tunnel-te200, timeout is 2 seconds
[snip]
Type escape sequence to abort.

 0 13.5.11.11 MRU 1500 [Labels: 5044 Exp: 0]
L 1 13.5.11.5 MRU 1500 [Labels: 8017 Exp: 0] 0 ms
L 2 13.5.8.8 MRU 1500 [Labels: implicit-null Exp: 0] 0 ms
! 3 13.8.12.12 1 ms
```

Traceroute from inside VRF EIGRP also shows this tunnel as functional. We can see the last few transport labels are 5044 and 8017, which describe the TE path via CSR5 and CSR8, respectively.

```
R1#traceroute 10.3.3.3 source 10.1.1.1
Type escape sequence to abort.
Tracing the route to 10.3.3.3
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.2.2 7 msec 4 msec 3 msec
 2 24.2.7.7 [MPLS: Labels 7041/6005/91010 Exp 0] 13 msec 12 msec 10 msec
 3 24.7.14.14 [MPLS: Labels 94003/6005/91010 Exp 0] 22 msec 37 msec 32 msec
 4 24.6.14.6 [MPLS: Labels 6005/91010 Exp 0] 75 msec 26 msec 19 msec
 5 10.6.11.11 [MPLS: Label 91010 Exp 0] 19 msec 21 msec 31 msec
 6 13.5.11.5 [MPLS: Labels 5044/92002 Exp 0] 31 msec 32 msec 31 msec
 7 13.5.8.8 [MPLS: Labels 8017/92002 Exp 0] 22 msec 51 msec 12 msec
 8 13.8.12.12 [MPLS: Label 92002 Exp 0] 19 msec 22 msec 20 msec
 9 10.3.12.3 20 msec 12 msec 12 msec
```

This kind of inter-AS TE approach is very simple but does not take advantage of the new inter-AS links. It's similar to option A with the added complexity of a third MPLS label in certain instances. With option B, actually having PE-PE tunnels across AS boundaries does not make sense. Below, I demonstrate such an example. Using loose-hop path expansion, we can specify the ASBRs along the path. This information is written to the RSVP PATH ERO with a special flag so the loose-hops know to "expand" the ERO to replace a single loose hop with several strict hops. We must use some form of static route here as

autoroute announce and forwarding adjacency are unsupported; this seems to violate the option B design as it requires routers to know about one other's TE IDs.

```
! CSR2
ip explicit-path name EP_LOOSE_2_7_5_11_12 enable
  next-address loose 24.0.0.7
  next-address loose 13.0.0.5
  next-address loose 13.0.0.12

interface Tunnel201
  description INTER-AS TO XRV2
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 13.0.0.12
  tunnel mpls traffic-eng autoroute destination
  tunnel mpls traffic-eng path-option 10 explicit name EP_LOOSE_2_7_5_11_12
```

To understand the process, we will enable PCALC debugging on CSR2, CSR7, and CSR5. This will reveal what the PCALC algorithm is doing behind the scenes to build the loose path.

```
! CSR2, CSR7, and CSR5
debug mpls traffic-eng path lookup
```

CSR2 begins the PCALC process by feeding the explicit-path into the PCALC algorithm. The hops are all identified as loose hops. CSR2 immediately realizes it does not have TE ID 13.0.0.12 in its TED; because of loose-hop expansion, CSR2 assumes that if it can compute the path to CSR7, then CSR7 can continue to compute partial paths towards the final destination. This is not typical for RSVP-TE since normally only the headend executes CPF and involves PCALC while the middle/tail routers only interact via the signaling protocol, RSVP. CSR2's best dynamic path to CSR7 is via XRv4, so the first loose-hop of 24.0.0.7 is expanded into a series of strict hops.

```
! CSR2
TE-PCALC-API: 24.0.0.2_2->13.0.0.12_201 {7}: P2P LSP Path Lookup called
TE-PCALC: 24.0.0.2_2->13.0.0.12_201 {7}: Path Request Info
  Flags: IP_EXPLICIT_PATH METRIC_TE
  IP explicit-path: Supplied
    24.0.0.7 Loose
    13.0.0.5 Loose
    13.0.0.12 Loose
  bw 0, min_bw 0, metric: 0
  setup_pri 7, hold_pri 7
  affinity_bits 0x0, affinity_mask 0xFFFF
TE-PCALC-PATH: 24.0.0.2_2->13.0.0.12_201 {7}: Area (isis level-2) Path
Lookup begin
TE-PCALC-PATH: Area (isis level-2): Dest ip addr 13.0.0.12 not found
TE-PCALC-PATH: lsr_exists:first Loose Hop is to addr 24.0.0.7
```

```

TE-PCALC-PATH:Path from 0000.0000.0002.00 -> 0000.0000.0007.00:
  24.7.14.14->24.7.14.7 (admin_weight=20):
  24.2.14.2->24.2.14.14 (admin_weight=10):
  num_hops 3, accumulated_aw 20, min_bw 200000
TE-PCALC-PATH: 24.0.0.2_2->13.0.0.12_201 {7}: Freeing rrr_path_setup_t
TE-PCALC-PATH: 24.0.0.2_2->13.0.0.12_201 {7}: Free all paths in path tree
TE-PCALC: Verify Path Lookup: 24.0.0.2_2->13.0.0.12_201 {7}: (protocol nil
area nil)
  Flags: METRIC_TE
  Last Strict Router: 24.0.0.7
  sub-lsp weight:0 (Total LSP weight:20)
  Hop List:
    24.2.14.14
    24.7.14.7
    24.0.0.7
    13.0.0.5 Loose
    13.0.0.12 Loose
TE-PCALC-VERIFY: VERIFY to 24.0.0.7 BEGIN:
TE-PCALC-VERIFY: Verify:
TE-PCALC-VERIFY: 0000.0000.0002.00, 24.0.0.2 points to
TE-PCALC-VERIFY: 0000.0000.0014.00, 24.2.14.14
TE-PCALC-VERIFY: Verify:
TE-PCALC-VERIFY: 0000.0000.0014.00, 24.2.14.14 points to
TE-PCALC-VERIFY: 0000.0000.0007.00, 24.7.14.7
TE-PCALC-VERIFY: VERIFY to 24.0.0.7 PASSED
TE-PCALC-PATH: 24.0.0.2_2->13.0.0.12_201 {7}: Area (isis level-2) Path
Lookup end: path found
TE-PCALC-API: 24.0.0.2_2->13.0.0.12_201 {7}: P2P LSP Path Lookup result:
success

```

The ERO that CSR2 sends to XRv4 contains this refinement computed above. From XRv4's perspective, it doesn't have to do any loose hop expansion since it is just following a strict path to CSR7.

```

R2#show ip rsvp sender detail filter session-type 7 destination 13.0.0.12 |
section outgoing
  ERO: (outgoing)
    24.2.14.14 (Strict IPv4 Prefix, 8 bytes, /32)
    24.7.14.7 (Strict IPv4 Prefix, 8 bytes, /32)
    24.0.0.7 (Strict IPv4 Prefix, 8 bytes, /32)
    13.0.0.5 (Loose IPv4 Prefix, 8 bytes, /32)
    13.0.0.12 (Loose IPv4 Prefix, 8 bytes, /32)

```

CSR7's debug output isn't very useful, likewise due to the next loose-hop being inter-AS. CSR7 invokes the LSP expand algorithm based on the loose hop of 13.0.0.5 inside the PATH ERO from XRv4. Also note that nodes CSR2 and XRv4 are "exclude nodes"; this guarantees that, while computing the loose path, CSR7 does not consider those nodes in the path to create a loop. The reason it knows about these nodes is the RSVP PATH RRO, which tracks the hops in the forward direction (seen later).

```

! CSR7
TE-PCALC-API: 24.0.0.2_2->13.0.0.5_201 {7}: LSP Path Expand called
TE-PCALC: 24.0.0.2_2->13.0.0.5_201 {7}: Path Request Info
  Flags:  END_SWCAP_UNKNOWN
  IP explicit-path: None (dynamic)
  bw 0, min_bw 0, metric: 0
  setup_pri 7, hold_pri 7
  affinity_bits 0x0, affinity_mask 0x0
TE-PCALC-PATH: 24.0.0.2_2->13.0.0.5_201 {7}: rrr_pcalc_lsr_expand: Exclude
node: 24.0.0.14 (intf: 24.7.14.14)
TE-PCALC-PATH: 24.0.0.2_2->13.0.0.5_201 {7}: rrr_pcalc_lsr_expand: Exclude
node: 24.0.0.2 (intf: 24.2.14.2)
TE-PCALC-PATH: 24.0.0.2_2->13.0.0.5_201 {7}: Area (isis level-2) Path Lookup
begin
TE-PCALC-PATH: expand_lsr: Dst addr 13.0.0.5 not found in area (isis level-
2)
TE-PCALC-PATH: 24.0.0.7_2->13.0.0.5_201 {7}: Area (isis level-2) Path Lookup
end: path not found
13.0.0.5 Can't Expand at this time
TE-PCALC-API: 24.0.0.7_2->13.0.0.5_201 {7}: LSP Path Expand result: failed
TE-PCALC-PATH: 24.0.0.7_2->13.0.0.5_201 {7}: Freeing rrr_path_setup_t

```

The debug output makes it look like the process failed, but it didn't. Instead, CSR7 just failed to expand the loose hop, so it removes its own addresses from the ERO and sends the PATH message onward to CSR5 without expanding 13.0.0.5.

```

R7#show ip rsvp sender detail filter session-type 7 destination 13.0.0.12 |
section ERO
  ERO: (incoming)
    24.7.14.7 (Strict IPv4 Prefix, 8 bytes, /32)
    24.0.0.7 (Strict IPv4 Prefix, 8 bytes, /32)
    13.0.0.5 (Loose IPv4 Prefix, 8 bytes, /32)
    13.0.0.12 (Loose IPv4 Prefix, 8 bytes, /32)
  ERO: (outgoing)
    10.5.7.5 (Strict IPv4 Prefix, 8 bytes, /32)
    13.0.0.5 (Loose IPv4 Prefix, 8 bytes, /32)
    13.0.0.12 (Loose IPv4 Prefix, 8 bytes, /32)

```

To show the PATH RRO, we look at the same detailed output with a different filter. Using this, we can see all the hops in the path to prevent loose-hop expansion loops. It also details any FRR within the network.

```

R7#show ip rsvp sender detail filter session-type 7 destination 13.0.0.12 |
section RRO
  RRO:
    24.7.14.14/32, Flags:0x0 (No Local Protection)

```

24.2.14.2/32, Flags:0x0 (No Local Protection)

When CSR5 receives the PATH message, it needs to expand the path to XRv2. It ignores the hop to 13.0.0.5 since that is its local TE ID and begins processing the next-hop in the ERO. Several error messages are displayed since CSR5 has no idea how to interpret the RSVP PATH RRO; the three highlighted IP addresses aren't in the TED (passive interfaces don't count) and so the router warns us that it cannot guarantee a loop free path as a result. CSR5 expands the loose path to XRv2 to include CSR8 as a strict hop.

```
! CSR5
TE-PCALC-API: 24.0.0.2_2->13.0.0.12_201 {7}: LSP Path Expand called
TE-PCALC: 24.0.0.2_2->13.0.0.12_201 {7}: Path Request Info
  Flags:  END_SWCAP_UNKNOWN
  IP explicit-path: None (dynamic)
  bw 0, min_bw 0, metric: 0
  setup_pri 7, hold_pri 7
  affinity_bits 0x0, affinity_mask 0x0
TE-PCALC-PATH: 24.0.0.2_2->13.0.0.12_201 {7}: rrr_pcalc_lsr_expand: Can't
get router ID addr for 10.5.7.7
TE-PCALC-PATH: 24.0.0.2_2->13.0.0.12_201 {7}: rrr_pcalc_lsr_expand: Can't
get router ID addr for 24.7.14.14
TE-PCALC-PATH: 24.0.0.2_2->13.0.0.12_201 {7}: rrr_pcalc_lsr_expand: Can't
get router ID addr for 24.2.14.2
TE-PCALC-PATH: 24.0.0.2_2->13.0.0.12_201 {7}: Area (ospf 13 area 0) Path
Lookup begin
TE-PCALC-PATH: exclude_path: system_id 0-0-0-0-0-0-0 not known!
TE-PCALC-PATH: exclude_path: system_id 0-0-0-0-0-0-0 not known!
TE-PCALC-PATH: exclude_path: system_id 0-0-0-0-0-0-0 not known!
TE-PCALC-PATH: Path from 13.0.0.5 -> 13.0.0.12:
  13.8.12.8->13.8.12.12 (admin_weight=2):
  13.5.8.5->13.5.8.8 (admin_weight=1):
  num_hops 3, accumulated_aw 2, min_bw 200000
TE-PCALC-PATH: 13.0.0.5_2->13.0.0.12_201 {7}: Area (ospf 13 area 0) Path
Lookup end: path found
13.0.0.12 expands to:
13.5.8.8
13.8.12.12
13.0.0.12
TE-PCALC-API: 13.0.0.5_2->13.0.0.12_201 {7}: LSP Path Expand result: success
TE-PCALC-PATH: 13.0.0.5_2->13.0.0.12_201 {7}: Freeing rrr_path_setup_t
```

We can see the incoming ERO with 2 loose hops and the outgoing ERO with zero loose hops. At this point, the remaining signaling is all intra-AS and very simple.

```
R5#show ip rsvp sender detail filter session-type 7 destination 13.0.0.12 |
section ERO
  ERO: (incoming)
```

```

10.5.7.5 (Strict IPv4 Prefix, 8 bytes, /32)
13.0.0.5 (Loose IPv4 Prefix, 8 bytes, /32)
13.0.0.12 (Loose IPv4 Prefix, 8 bytes, /32)
ERO: (outgoing)
13.5.8.8 (Strict IPv4 Prefix, 8 bytes, /32)
13.8.12.12 (Strict IPv4 Prefix, 8 bytes, /32)
13.0.0.12 (Strict IPv4 Prefix, 8 bytes, /32)

```

We cannot use traceroute to verify the LSP since routers in AS 13 has no reachability back to CSR2, so the traceroute replies will not return. This is part of the reason why inter-AS option B is awkward. However, we can prove that the traceroute probes are reaching XRv2 using some debug. OAM LSP verification (LSPV) reports packets from RID 13.0.0.2, which means the unidirectional LSP is functional. XRv2 reports there is no reverse LSP, so there is no ability to reply.

```

R2#traceroute mpls traffic-eng tunnel 201 source 24.0.0.2
Tracing MPLS TE Label Switched Path on Tunnel201, timeout is 2 seconds
[snip]
Type escape sequence to abort.
 0 24.2.14.2 MRU 1500 [Labels: 94005 Exp: 0]
L 1 24.2.14.14 MRU 1500 [Labels: 7014 Exp: 0] 9 ms
L 2 24.7.14.7 MRU 1500 [Labels: 5080 Exp: 0] 4 ms
. 3 *
. 4 *
. 5 *

```

```

RP/0/0/CPU0:XRv2#debug mpls traffic-eng oam
DBG-OAM_EVT[1]: mpls_te_s2l_fill_lsp_ping_info:212: LSPV-S2L: RID 13.0.0.12,
nhRID 0.0.0.0, nhIFh 0x0, nhIF adr 0.0.0.0, out lbl 1048577, FRR not active,
MP lbl 1048577, RW siblings 1
DBG-OAM_EVT[1]: mpls_te_lspv_fill_p2p_mid_tail_prop_using_rev_lsp:292: No
rev_lsp

```

When we traceroute inside the VPN, the traffic flows through. Looking carefully at the first few labels of the TE tunnel above, we can clearly see that VPN traffic isn't even going inside the TE tunnel at all. For example, XRv4 is the next-hop after CSR2, but it uses label 94017 versus label 94005.

```

R1#traceroute 10.3.3.3 source 10.1.1.1
Type escape sequence to abort.
Tracing the route to 10.3.3.3
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.2.2 5 msec 4 msec 4 msec
 2 24.2.14.14 [MPLS: Labels 94017/91010 Exp 0] 10 msec 8 msec 9 msec
 3 24.6.14.6 [MPLS: Labels 6005/91010 Exp 0] 23 msec 31 msec 32 msec
 4 10.6.11.11 [MPLS: Label 91010 Exp 0] 29 msec 32 msec 29 msec
 5 13.8.11.8 [MPLS: Labels 8000/92002 Exp 0] 32 msec 31 msec 31 msec
 6 13.8.12.12 [MPLS: Label 92002 Exp 0] 20 msec 19 msec 21 msec
 7 10.3.12.3 19 msec 12 msec 11 msec

```

Following the route recursion, this makes perfect sense. CSR2 is going to push a BGP label allocated by XRv1 (10.6.11.11) and tunnel traffic towards the ASBR, CSR6. Having an end-to-end tunnel does nothing for us because the BGP next-hop isn't 13.0.0.12. It doesn't matter that CSR2 tries to route VPN traffic to XRv2 directly over the TE tunnel because the BGP topology doesn't enable this. The VPN traffic must pass through the ASBR that advertised the best-path or else the label swapping cannot occur properly.

```
R2#show bgp vpnv4 unicast vrf EIGRP 10.3.3.3/32
BGP routing table entry for 24:3:10.3.3.3/32, version 6288
Paths: (1 available, best #1, table EIGRP)
  Not advertised to any peer
  Refresh Epoch 1
  13, (Received from a RR-client), imported path from 13:3:10.3.3.3/32
(global)
  10.6.11.11 (metric 20) (via default) from 24.0.0.6 (24.0.0.6)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    Extended Community: RT:13:3 0x8800:32768:0 0x8801:3:288
      0x8802:65281:2560 0x8803:1:1500 0x8806:0:167971843
    Connector Attribute: count=1
      type 1 len 12 value 13:3:13.0.0.12
    mpls labels in/out nlabel/91010
    rx pathid: 0, tx pathid: 0x0
```

```
R2#show ip route 10.6.11.11
Routing entry for 10.6.11.11/32
  Known via "isis", distance 115, metric 20, type level-2
  Redistributing via isis 24
  Last update from 24.2.14.14 on GigabitEthernet2.524, 00:34:33 ago
  Routing Descriptor Blocks:
  * 24.2.14.14, from 24.0.0.6, 00:34:33 ago, via GigabitEthernet2.524
    Route metric is 20, traffic share count is 1
```

If we try to forcefully move traffic into the tunnel, everything breaks. I add a temporary bogus static route below to prove it. Suddenly the route recursion looks correct at a glance. This is why it is important to track the actual label values and not just count the number of labels in the stack.

```
R2#show ip route 10.6.11.11
Routing entry for 10.6.11.11/32
  Known via "static", distance 1, metric 0 (connected)
  Routing Descriptor Blocks:
  * directly connected, via Tunnel201
    Route metric is 0, traffic share count is 1
```

```
R2#show ip cef vrf EIGRP 10.3.3.3
10.3.3.3/32
  nexthop 10.6.11.11 Tunnel201 label 91010
```

Using ping and traceroute, we can clearly see VPN connectivity is broken. VPN label 91010 is still being added to the stack but XRv1 has been totally bypassed. This VPN label will be exposed to XRv2 and the traffic will be dropped as a result.

```
R1#ping 10.3.3.3 source 10.1.1.1
Type escape sequence to abort.
Sending5, 100-byte ICMP Echos to 10.3.3.3, timeout is 2 seconds:
Packet sent with a source address of 10.1.1.1
.....
Success rate is 0 percent (0/5)
```

```
R1#traceroute 10.3.3.3 source 10.1.1.1
Type escape sequence to abort.
Tracing the route to 10.3.3.3
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.2.2 6 msec 3 msec 4 msec
 2 * *
```

Using EPC outbound on CSR8 towards XRv2, we can see this failure. CSR8 performs PHP along the TE LSP as it should, exposing the incorrect VPN label to XRv2. This proves that the TE tunnel and static route on CSR2 are functioning properly, but the lack of synchronization with the BGP topology means option B is broken. Label 0x16382 is 91010 in decimal, and XRv2 drops this traffic as there is no corresponding LFIB entry. This label should have been exposed to XRv1.

```
R8#show mpls traffic-eng tunnels role middle | include Label
InLabel : GigabitEthernet2.558, 8009
OutLabel : GigabitEthernet2.582, implicit-null

R8#show monitor capture CAP buffer detailed
 4 122 1.794017 00:50:56:A9:FB:1C -> 00:50:56:A9:0E:6F MPLS unicast
0000: 005056A9 0E6F0050 56A9FB1C 81000DFE .PV..o.PV.....
0010: 88471638 21FA4500 00640017 0000FE01 .G.8!.E..d.....
0020: A47A0A01 01010A03 03030800 CE080006 .z.....
0030: 00020000 000020D8 8F61ABCD ABCDABCD ..... ..a.....

RP/0/0/CPU0:XRv2#show mpls forwarding labels 91010
[no output]
```

In summary, inter-AS TE does technically work (PCALC completes and the LSP can be signaled) over an option B network. It is not very useful because the ASBRs will adjust the BGP next-hops at least once, so tunneling traffic across ASes in a single LSP is not compatible with the option B design. This feature would make more sense for UMPLS or inter-AS option C architectures where MPLS service next-hops (L3VPN, L2VPN, etc) are unchanged. For MPLS-TE in an option B environment, I would recommend the tunnel stitching method as used for option A.

8.4.2.6 Confederation variation

Confederations with option B are a little more interesting than with option A. With option A, we only had to adjust the BGP configurations slightly: confederation ASN/peer specification, next-hop processing on ASBRs, and filter removal. We will have to do these things for option B, but the configuration is more involved on XR as new commands are introduced for intraconfederation (inter-subAS) MPLS forwarding. First, I begin by changing the BGP ASNs as was done with option A. This configuration will be identical for all confederation variations. This also includes adjusting CSR10, a CE router, to peer with AS 42518 rather than AS 24.

```
! CSR2 and XRv4
router bgp 24
  bgp confederation identifier 42518

! CSR6 and CSR7
router bgp 24
  bgp confederation identifier 42518
  bgp confederation peers 13

! CSR8 and XRv2
router bgp 13
  bgp confederation identifier 42518

! CSR5 and XRv1
router bgp 13
  bgp confederation identifier 42518
  bgp confederation peers 24

! CSR10
router bgp 100
  neighbor 10.8.10.8 remote-as 42518
  neighbor FD00:10:8:10::8 remote-as 42518
```

For brevity, I check CSR5 and CSR6 for their VPNv4/v6 sessions between ASes. This check ensures that the confed-internal connections to the RRs are operational. It also validates the confed-external connections between all sets of ASBRs. Quickly scanning the last column, we can see the peers are still up. This verification is easier than with option A since there were per-VRF peers across the intraconfederation (inter-subAS) boundaries.

```
R5#show bgp vpnv4 unicast all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
10.5.6.6       4      24    163    148     181   0    0 00:12:25      8
10.5.7.7       4      24    122    154     181   0    0 00:12:23      8
13.0.0.12      4      13    221    142     181   0    0 00:11:30      9

R5#show bgp vpnv6 unicast all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
10.5.6.6       4      24    163    148     550   0    0 00:12:31     12
10.5.7.7       4      24    123    154     550   0    0 00:12:29     12
13.0.0.12      4      13    222    143     550   0    0 00:11:36      9
```



```
R6#show bgp vpnv4 unicast all summary | begin ^Neigh
```

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
10.5.6.5	4	13	148	164	533	0	0	00:12:43	9
10.6.11.11	4	13	64	135	533	0	0	00:11:39	9
24.0.0.2	4	24	488	171	533	0	0	00:12:50	8

```
R6#show bgp vpnv6 unicast all summary | begin ^Neigh
```

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
10.5.6.5	4	13	148	164	945	0	0	00:12:43	9
10.6.11.11	4	13	64	135	945	0	0	00:11:39	6
24.0.0.2	4	24	488	171	945	0	0	00:12:50	12

We can remove most of the EBGp-oriented filters from XRv1 as well. Some of the RPLs are used to adjust BGP best-path selection, so those are left in place and documented below (commented out). This step is unnecessary but helps clean up the configuration.

```
! XRv2
router bgp 13
 neighbor 10.6.11.6
  address-family vpnv4 unicast
    no route-policy RPL_PASS in
    no route-policy RPL_PASS out

  address-family vpnv6 unicast
    ! route-policy RPL_SET_LOCAL_PREF(PS_XRV3_V6, 200) in
    no route-policy RPL_PASS out

  address-family ipv4 mdt
    no route-policy RPL_PASS in
    ! route-policy RPL_MDT_MED_OUT(1111) out

  address-family ipv4 mvpn
    no route-policy RPL_PASS in
    no route-policy RPL_PASS out

  address-family ipv6 mvpn
    no route-policy RPL_PASS in
    no route-policy RPL_PASS out
```

To review from earlier, CSR5 and XRv1 are configured to retain all RTs (the preferred approach for option B ASBRs), CSR7 is a route-reflector, and CSR6 imports the routes locally into VRFs. Checking CSR5, we can see routes received from CSR6 and CSR7 with RD 24:2 (represents OSPF VPN). These are confed-external as expected, and like option A, the next-hops are inaccessible. Like option A, these sub-AS loopbacks are not supposed to leak between AS boundaries, so next-hop-self is the best option to resolve this.

```
R5#show bgp vpnv4 unicast rd 24:2 10.9.9.9/32
```

```

BGP routing table entry for 24:2:10.9.9.9/32, version 170
Paths: (2 available, no best path)
  Not advertised to any peer
  Refresh Epoch 1
  (24)
    24.0.0.2 (inaccessible) (via default) from 10.5.7.7 (24.0.0.7)
      Origin incomplete, metric 1, localpref 100, valid, confed-external
      Extended Community: RT:24:2 OSPF ROUTER ID:10.2.9.2:0
      OSPF RT:0.0.0.0:2:0
      mpls labels in/out 5047/2015
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  (24)
    24.0.0.2 (inaccessible) (via default) from 10.5.6.6 (24.0.0.6)
      Origin incomplete, metric 1, localpref 100, valid, confed-external
      Extended Community: RT:24:2 OSPF ROUTER ID:10.2.9.2:0
      OSPF RT:0.0.0.0:2:0
      mpls labels in/out 5047/2015
      rx pathid: 0, tx pathid: 0

```

The next-hop-self configuration is long only because so many AFIs are negotiated between the ASBRs, such as VPNv4/v6, IPv4 MDT, MVPNV4/v6, L2VPN VPLS, etc. For brevity, I limit the documentation to CSR6 and XRv1.

```

! CSR6
router bgp 24
  address-family ipv4 mvpn
    neighbor 10.5.6.5 next-hop-self
    neighbor 10.6.11.11 next-hop-self

  address-family vpnv4
    neighbor 10.5.6.5 next-hop-self
    neighbor 10.6.11.11 next-hop-self

  address-family ipv4 mdt
    neighbor 10.5.6.5 next-hop-self
    neighbor 10.6.11.11 next-hop-self

  address-family ipv6 mvpn
    neighbor 10.5.6.5 next-hop-self
    neighbor 10.6.11.11 next-hop-self

  address-family vpnv6
    neighbor 10.5.6.5 next-hop-self
    neighbor 10.6.11.11 next-hop-self

  address-family l2vpn vpls
    neighbor 10.5.6.5 next-hop-self

```

```

! XRv1
router bgp 13
  neighbor 10.6.11.6
    address-family vpnv4 unicast
      next-hop-self

  address-family vpnv6 unicast
    next-hop-self

  address-family ipv4 mdt
    next-hop-self

  address-family ipv4 mvpn
    next-hop-self

  address-family ipv6 mvpn
    next-hop-self

```

Similar to option A, applying this configuration seems to “fix” everything. This is mostly because it restores the traditional option B eBGP behavior at the AS boundaries. As a very fast verification, we can see the OSPF sham-links are up, and XRv3 learns the C-PIM RP information. This quickly tests us, with some degree of confidence, that unicast and multicast connectivity are operable between sub-ASes.

```

R8#show ospfv3 vrf OSPF sham-links | include ^Sham
Sham Link OSPFv3_SL0 to address FD00::2 is up
Sham Link OSPFv3_SL1 to address FD00::2 is up

RP/0/0/CPU0:XRv3#show pim rp mapping
PIM Group-to-RP Mappings
Group(s) 224.0.0.0/4
  RP 10.3.3.3 (?), v2
    Info source: 10.1.13.1 (?), elected via bsr, priority 0, holdtime 150
    Uptime: 00:03:59, expires: 00:01:32

```

I quickly trace the LSP from CSR9 to CSR10 inside of the central services VPN using IPv6. CSR9 has an external route via CSR2. This is an indication that the sham-link is working since CSR9 is going to prefer the shortest path to the originating ASBR, which is 10.4.8.8 (CSR8). The shortest path to CSR8 is via the MPLS network and the path is intra-area. Just because the shared services route is external does not mean the sham-link cannot influence forwarding towards it.

```

R9#show ipv6 route ::110:0:0:2
Routing entry for ::110:0:0:2/128
  Known via "ospf 2", distance 110, metric 1, type extern 2
  Route count is 1/1, share count 0
  Routing paths:
    FE80::2, GigabitEthernet2.529

```

Last updated 00:07:28 ago

```
R9#show ospfv3 2 database external ::110:0:0:2/128
```

```
OSPFv3 2 address-family ipv6 (router-id 10.4.9.9)
```

```
Type-5 AS External Link States
```

```
LS age: 1802
LS Type: AS External Link
Link State ID: 6
Advertising Router: 10.4.8.8
LS Seq Number: 80000003
Checksum: 0x45F8
Length: 44
Prefix Address: ::110:0:0:2
Prefix Length: 128, Options: DN
Metric Type: 2 (Larger than any link state path)
Metric: 1
```

```
R9#show ospfv3 2 ipv6 border-routers
```

```
OSPFv3 2 address-family ipv6 (router-id 10.4.9.9)
```

```
Codes: i - Intra-area route, I - Inter-area route
```

```
i 10.2.9.2 [1] via FE80::2, GigabitEthernet2.529, ABR/ASBR, Area 0, SPF 161
i 10.4.8.8 [2] via FE80::2, GigabitEthernet2.529, ABR/ASBR, Area 0, SPF 161
```

CSR2's VPNv4 route originates from AS 100 and transits sub-AS 13. RT:13:1 is imported locally by both VRF EIGRP and OSPF, and label 5030 is used to forward traffic to CSR5. This implies that CSR2 has an IGP route to CSR5, also implying that CSR6 did not set "next-hop-self" for this AFI. Label 94032 is used since the IGP route points through XRv4, requiring XRv4's LDP label be imposed. The label stack becomes {94032 5030}.

```
R2#show bgp vpnv6 unicast vrf OSPF ::110:0:0:2/128
BGP routing table entry for [24:2>::110:0:0:2/128, version 791
Paths: (1 available, best #1, table OSPF)
  Not advertised to any peer
  Refresh Epoch 2
  (13) 100, (Received from a RR-client), imported path from
[13:1>::110:0:0:2/128 (global)
  ::FFFF:10.5.6.5 (metric 20) (via default) from 24.0.0.6 (24.0.0.6)
  Origin incomplete, metric 0, localpref 100, valid, confed-internal,
best
  Extended Community: RT:13:1
  mpls labels in/out nolabel/5030
  rx pathid: 0, tx pathid: 0x0
```

```
R2#show ip route 10.5.6.5
Routing entry for 10.5.6.5/32
  Known via "isis", distance 115, metric 20, type level-2
  Redistributing via isis 24
  Last update from 24.2.14.14 on GigabitEthernet2.524, 00:47:38 ago
  Routing Descriptor Blocks:
  * 24.2.14.14, from 24.0.0.6, 00:47:38 ago, via GigabitEthernet2.524
    Route metric is 20, traffic share count is 1
```

```
R2#show mpls ldp bindings 10.5.6.5 32 neighbor 24.0.0.14
  lib entry: 10.5.6.5/32, rev 15
    remote binding: lsr: 24.0.0.14:0, label: 94032
```

XRv4 and CSR6 are P routers along this LSP, performing label swap/pop operations as shown below. Although CSR6 is an ASBR, it must tunnel the VPNv4-labeled traffic to CSR5 as it did not set a new VPN next-hop (and thus did not allocate a new VPN label).

```
RP/0/0/CPU0:XRv4#show mpls forwarding labels 94032
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94032	6069	10.5.6.5/32	Gi0/0/0/0.564	24.6.14.6	12944

```
R6#show mpls forwarding-table labels 6069
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
6069	Pop Label	10.5.6.5/32	14224	Gi2.556	10.5.6.5

CSR5 performs the function of a classic option B ASBR by swapping the VPN label for the original VPN label of 8004, allocated by CSR8. Since CSR5 did change the next-hop (result of next-hop-self during this confederation test), the VPN label must be swapped. CSR8 receives traffic with VPN label 8004, removes all labels, and delivers the traffic to CSR10 inside the VPN.

```
R5#show mpls forwarding-table labels 5030
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
5030	8004	[13:1]::110:0:0:2/128	\	Gi2.558	13.5.8.8

```
R8#show mpls forwarding-table labels 8004 detail
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
8004	No Label	::110:0:0:2/128[V]	\	Gi2.580	FE80::10

MAC/Encaps=18/18, MRU=1504, Label Stack{}

005056A9F961005056A9FB1C81000DFC86DD

VPN route: BGP

No output feature configured

Traceroute on CSR9 confirms this operation and the label stack. So far, this is identical behavior to what was observed in option B.

```
R9#traceroute ipv6
```

```
Target IPv6 address: ::110:0:0:2
```

```
Source address: ::10:9:9:9
```

```
[snip]
```

```
1 FD00:10:2:9::2 5 msec 4 msec 4 msec
2 2024:24:2:14::14 [MPLS: Labels 94032/5030 Exp 0] 11 msec 8 msec 15 msec
3 ::FFFF:24.6.14.6 [MPLS: Labels 6069/5030 Exp 0] 35 msec 34 msec 35 msec
4 ::FFFF:10.5.6.5 [MPLS: Label 5030 Exp 0] 30 msec 35 msec 33 msec
5 FD00:10:8:10::8 [MPLS: Label 8004 Exp 0] 23 msec 21 msec 22 msec
6 FD00:10:8:10::10 23 msec 15 msec 16 msec
```

Next, I will simulate CSR5 going down for maintenance. The router remains online, but its intraconfederation links are inoperable. The only connection between sub-ASes is between XRv1 and CSR6. CSR6 now learns the central services routes from XRv1 only. Everything appears correct. CSR6 does not adjust the next-hop and so does not allocate a local label, nor perform a VPNv6 label swap. The route is confed-external and originates from AS 100, transiting through sub-AS 13.

```
R6#show bgp vpnv6 unicast vrf OSPF ::110:0:0:2/128
```

```
BGP routing table entry for [24:2>::110:0:0:2/128, version 1058
```

```
Paths: (1 available, best #1, table OSPF)
```

```
Not advertised to any peer
```

```
Refresh Epoch 1
```

```
(13) 100, imported path from [13:1>::110:0:0:2/128 (global)
```

```
::FFFF:10.6.11.11 (via default) from 10.6.11.11 (13.0.0.11)
```

```
Origin incomplete, metric 0, localpref 100, valid, confed-external,
best
```

```
Extended Community: RT:13:1
```

```
mpls labels in/out nlabel/91009
```

```
rx pathid: 0, tx pathid: 0x0
```

CSR2 correct imposes a transport label from XRv4 towards 10.6.11.11 (VPN next-hop) along with the VPN label of 91009 above.

```
R2#show ipv6 cef vrf OSPF ::110:0:0:2
```

```
::110:0:0:2/128
```

```
nexthop 24.2.14.14 GigabitEthernet2.524 label 94034 91009
```

XRv1 appears to perform the correct VPN label swapping as well, just as CSR5 did.

```
RP/0/0/CPU0:XRv1#show mpls forwarding labels 91009
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
91009	8004	13:1:::110:0:0:2/128	\	13.0.0.8	2220

Traceroute on CSR9 indicates proper connectivity from CSR9 to CSR10.

```
R9#traceroute ipv6
Target IPv6 address: ::110:0:0:2
Source address: ::10:9:9:9
[snip]

 1 FD00:10:2:9::2 4 msec 4 msec 4 msec
 2 2024:24:2:14::14 [MPLS: Labels 94034/91009 Exp 0] 7 msec 7 msec 10 msec
 3 ::FFFF:24.6.14.6 [MPLS: Labels 6074/91009 Exp 0] 33 msec 33 msec 34 msec
 4 FD00:10:6:11::11 [MPLS: Label 91009 Exp 0] 34 msec 33 msec 33 msec
 5 FD00:10:8:10::8 [MPLS: Label 8004 Exp 0] 16 msec 17 msec 16 msec
 6 FD00:10:8:10::10 29 msec 15 msec 14 msec
```

Using traceroute in the opposite direction, we notice a very undesirable effect. Traffic is preferring the backdoor link over the MPLS network. This is a result of the weight being higher for locally-originated routes. CSR8 should not be preferring the backdoor link to CSR9 anyway; this is an indication that the sham-links have failed. This issue is not specific to option B confederations, but we did not examine this exact case in the original option B L3VPN section.

```
R10#traceroute ipv6
Target IPv6 address: ::10:9:9:9
Source address: ::110:0:0:2
[snip]

 1 FD00:10:8:10::8 3 msec 3 msec 3 msec
 2 FD00:10:4:8::4 4 msec 4 msec 3 msec
 3 FD00:10:4:9::9 7 msec 8 msec 22 msec
```

```
R8#show bgp vpnv6 unicast vrf BGP ::10:9:9:9/128
BGP routing table entry for [13:1]::10:9:9:9/128, version 203
Paths: (2 available, best #2, table BGP)
  Advertised to update-groups:
    4
  Refresh Epoch 1
  (24), imported path from [24:2]::10:9:9:9/128 (global)
    ::FFFF:13.0.0.11 (metric 2) (via default) from 13.0.0.12 (13.0.0.12)
      Origin incomplete, metric 1, localpref 100, valid, confed-internal
      Extended Community: RT:24:2 OSPF ROUTER ID:10.2.9.2:0
      OSPF RT:0.0.0.0:2:0
      Originator: 13.0.0.11, Cluster list: 13.0.0.12
```

```

mpls labels in/out nolabel/91021
rx pathid: 0, tx pathid: 0
Refresh Epoch 1
Local, imported path from [13:2]::10:9:9:9/128 (OSPF)
FE80::4 (FE80::4) (via vrf OSPF) (via OSPF) from 0.0.0.0 (13.0.0.8)
Origin incomplete, metric 501, localpref 100, weight 32768, valid,
external, best
Extended Community: RT:13:2 OSPF ROUTER ID:10.4.8.8:0
OSPF RT:0.0.0.0:2:0
rx pathid: 0, tx pathid: 0x0

```

Tracing the LSP between sham-link endpoints (necessary for the targeted OSPF hello exchange), we see that CSR8 has a valid VPNv6 route with associated label from XRv1. The route to the BGP next-hop is IGP, so the LDP label is used, which is implicit-null.

```

R8#show bgp vpnv6 unicast vrf OSPF fd00::2/128
BGP routing table entry for [13:2]FD00::2/128, version 204
Paths: (1 available, best #1, table OSPF)
Not advertised to any peer
Refresh Epoch 1
(24), imported path from [24:2]FD00::2/128 (global)
::FFFF:13.0.0.11 (metric 2) (via default) from 13.0.0.12 (13.0.0.12)
Origin IGP, metric 0, localpref 100, valid, confed-internal, best
Extended Community: RT:24:2
Originator: 13.0.0.11, Cluster list: 13.0.0.12
mpls labels in/out nolabel/91022
rx pathid: 0, tx pathid: 0x0

```

```

R8#show ip route 13.0.0.11
Routing entry for 13.0.0.11/32
Known via "ospf 13", distance 110, metric 2, type intra area
Last update from 13.8.11.11 on GigabitEthernet2.581, 01:19:36 ago
Routing Descriptor Blocks:
* 13.8.11.11, from 13.0.0.11, 01:19:36 ago, via GigabitEthernet2.581
Route metric is 2, traffic share count is 1

```

```

R8#show mpls ldp bindings 13.0.0.11 32 neighbor 13.0.0.11
lib entry: 13.0.0.11/32, rev 6
remote binding: lsr: 13.0.0.11:0, label: imp-null

```

XRv1 swaps the VPN label to 6059, which is what CSR6 should be expecting.

```

RP/0/0/CPU0:XRv1#show mpls forwarding labels 91022

```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
91022	6059	24:2:fd00::2/128	Gi0/0/0/0.561	10.6.11.6	23808

CSR6 swaps the VPN label of 6059 to 2020, and pushes a new LDP label of 94009 to tunnel the VPN traffic through XRv4.

```
R6#show bgp vpnv6 unicast vrf OSPF fd00::2/128
BGP routing table entry for [24:2]FD00::2/128, version 987
Paths: (1 available, best #1, table OSPF)
  Advertised to update-groups:
    14
  Refresh Epoch 1
  Local
    ::FFFF:24.0.0.2 (metric 20) (via default) from 24.0.0.2 (24.0.0.2)
    Origin IGP, metric 0, localpref 100, valid, confed-internal, best
    Extended Community: RT:24:2
    mpls labels in/out 6059/2020
    rx pathid: 0, tx pathid: 0x0
```

```
R6#show ip route 24.0.0.2
Routing entry for 24.0.0.2/32
  Known via "isis", distance 115, metric 20, type level-2
  Redistributing via isis 24
  Last update from 24.6.14.14 on GigabitEthernet2.564, 02:01:40 ago
  Routing Descriptor Blocks:
  * 24.6.14.14, from 24.0.0.2, 02:01:40 ago, via GigabitEthernet2.564
    Route metric is 20, traffic share count is 1
```

```
R6#show mpls ldp bindings 24.0.0.2 32 neighbor 24.0.0.14
lib entry: 24.0.0.2/32, rev 21
remote binding: lsr: 24.0.0.14:0, label: 94009
```

XRv4 pops the LDP label to expose label 2020 to CSR2. Since this is the local label for a connected route, CSR2 pops the VPN label and performs a VRF-aware routing lookup on the packet. The traffic is delivered to loopback 2 as expected.

```
RP/0/0/CPU0:XRv4#show mpls forwarding labels 94009
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94009	Pop	24.0.0.2/32	Gi0/0/0/0.524	24.2.14.2	2663946

```
R2#show mpls forwarding-table labels 2020
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Outgoing interface	Next Hop
2020	Pop Label	FD00::2/128[V]	0	aggregate/OSPF	

```
R2#show ipv6 route vrf OSPF fd00::2
Routing entry for FD00::2/128
  Known via "connected", distance 0, metric 0, type receive, connected
  Route count is 1/1, share count 0
```

Routing paths:

receive via Loopback2

Last updated 17:30:28 ago

Debugging OSPFv3 packets on CSR2 for IPv4 and IPv6, we can clearly see CSR2 receiving packets along both sham-links. This implies that connectivity from CSR8 to CSR2 is functioning correctly.

```
R2#debug ospfv3 vrf OSPF ipv6 packet
OSPFv3 packet debugging is on for process 2, IPv6, vrf OSPF
```

```
OSPFv3-2-IPv6-OSPF PAK    SL1: Sham link packet: interface VRF ID 0, packet
VRF ID 2
OSPFv3-2-IPv6-OSPF PAK   : SL1: IN: FD00::8->FD00::2: ver:3 type:1 len:36
rid:10.4.8.8 area:0.0.0.0 chksum:EECE inst:0
```

```
R2#debug ospfv3 vrf OSPF ipv4 packet
OSPFv3 packet debugging is on for process 2, IPv4, vrf OSPF
```

```
OSPFv3-2-IPv4-OSPF PAK    SL0: Sham link packet: interface VRF ID 0, packet
VRF ID 2
OSPFv3-2-IPv4-OSPF PAK   : SL0: IN: FD00::8->FD00::2: ver:3 type:1 len:36
rid:10.4.8.8 area:0.0.0.0 chksum:ADD0 inst:64
```

The output does not indicate any attempt to send packets out of the sham-links towards FD00::8/128. CSR2 does not have a VPN route to this destination in its OSPF VPN table, or in any table at all. The same is true for CSR6, which likely indicates some kind of routing problem inside sub-AS 13.

```
R2#show bgp vpnv6 unicast vrf OSPF fd00::8/128
% Network not in table
```

```
R2#show bgp vpnv6 unicast all fd00::8/128
% Network not in table
```

```
R6#show bgp vpnv6 unicast all FD00::8/128
% Network not in table
```

XRv1 doesn't even have it. XRv2 does, and reports that it advertises it to XRv1 as well.

```
RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast rd 13:2 fd00::8/128 brief | begin
Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 13:2					
*>ifd00::8/128	13.0.0.8	0	100	0	i

```
RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast rd 13:2 advertised neighbor 13.0.0.11
summary
Network                Next Hop                From                AS Path
```

```

Route Distinguisher: 13:2
::10:4:4:4/128      13.0.0.8      13.0.0.8      ?
::10:9:9:9/128      13.0.0.8      13.0.0.8      ?
fd00::8/128         13.0.0.8      13.0.0.8      i

```

This issue actually has nothing to do with confederations, but is good practice for solving a “non-issue”. In the option B section, we configured XRv1 not to be a candidate ASBR for the OSPF VPN using an RPL to selectively retain RTs. This behavior is the result of design decisions we made earlier for demonstration purposes. For this test, I will remove this configuration by retaining all RTs on XRv1. The sham-links immediately come up and proper connectivity is restored.

```

! XRv1
router bgp 13
  address-family vpnv6 unicast
    retain route-target all

R10#traceroute ipv6
Target IPv6 address: ::10:9:9:9
Source address: ::110:0:0:2
[snip]

```

```

 1 FD00:10:8:10::8 3 msec 3 msec 3 msec
 2 2013:13:8:11::11 [MPLS: Label 91021 Exp 0] 8 msec 7 msec 10 msec
 3 ::FFFF:10.6.11.6 [MPLS: Label 6060 Exp 0] 14 msec 18 msec 22 msec
 4 2024:24:6:14::14 [MPLS: Labels 94009/2019 Exp 0] 22 msec 23 msec 23 msec
 5 FD00:10:2:9::2 [MPLS: Label 2019 Exp 0] 27 msec 22 msec 23 msec
 6 FD00:10:2:9::9 23 msec 14 msec 15 msec

```

As a side note, Cisco clearly states that some additional commands are necessary when using inter-AS option B with confederations under certain conditions. Specifically, the documentation states that the “mpls activate” stanza, complete with a list of transit links, must be configured under BGP. **This is only required when the BGP next-hop is learned through IGP or static; that is, if the next-hop is not connected.** I suspect that this would be required for ordinary eBGP peers as well given the multi-hop condition and is not specific to BGP confederations. I quickly configure XRv1 to peer with a new loopback on CSR6. CSR6 still peers with XRv1’s directly connected interface, but sources it’s BGP session from the new loopback. XRv1 no longer needs a host-route to CSR6’s connected interface, but does need one to the loopback for the BGP peer to form. We can also assume that it is needed for MPLS forwarding to work. You can use eBGP-multihop or ignore the connected check. Both of these qualify as “multi-hop” in terms of Cisco’s documentation of this feature.

```

! CSR6
interface Loopback611
  description EBGp MHOP TEST
  ip address 10.6.110.110 255.255.255.255

router bgp 24

```

```

neighbor 10.6.11.11 update-source Loopback611

! XRv1
router static
address-family ipv4 unicast
no 10.6.11.6/32 GigabitEthernet0/0/0/0.561
10.6.110.110/32 GigabitEthernet0/0/0/0.561 10.6.11.6

router bgp 13
no neighbor 10.6.11.6
neighbor 10.6.110.110
remote-as 24
ignore-connected-check
address-family vpnv4 unicast
next-hop-self

address-family vpnv6 unicast
route-policy RPL_SET_LOCAL_PREF(PS_XRV3_V6, 200) in
next-hop-self

address-family ipv4 mdt
route-policy RPL_MDT_MED_OUT(1111) out
next-hop-self

address-family ipv4 mvpn
next-hop-self

address-family ipv6 mvpn
next-hop-self

```

XRv1 successfully learns the remote sham-link endpoint in AS 24, which is FD00::2/128. Traffic arriving to XRv1 for this VPN route would have label 91022 which XRv1 should swap to label 6059.

```

RP/0/0/CPU0:XRv1#show bgp vpnv6 unicast rd 24:2 FD00::2/128
BGP routing table entry for fd00::2/128, Route Distinguisher: 24:2
Versions:
  Process          bRIB/RIB   SendTblVer
  Speaker          576       576
  Local Label: 91022
Paths: (1 available, best #1)
  Advertised to peers (in unique update groups):
    13.0.0.12
  Path #1: Received by speaker 0
  Advertised to peers (in unique update groups):
    13.0.0.12
  (24)
    10.6.110.110 from 10.6.110.110 (24.0.0.6)
      Received Label 6059

```

```

Origin IGP, metric 0, localpref 100, valid, confed-external, best,
group-best, import-candidate, not-in-vrf
Received Path ID 0, Local Path ID 1, version 576
Extended community: RT:24:2

```

XRv1 has a /32 route to the BGP next-hop, which meets the MPLS forwarding requirement in XR. Since the route is learned via static, it also expects an LDP label for this destination. A local label is allocated, but a remote label remains unbound. There is no LDP neighbor with 10.6.11.6 at all.

```

RP/0/0/CPU0:XRv1#show route 10.6.110.110
Routing entry for 10.6.110.110/32
  Known via "static", distance 1, metric 0
  Routing Descriptor Blocks
    10.6.11.6, via GigabitEthernet0/0/0/0.561
    Route metric is 0
  No advertising protos.

RP/0/0/CPU0:XRv1#show mpls ldp bindings 10.6.110.110/32
10.6.110.110/32, rev 17
  Local binding: label: 91037
  No remote bindings

```

Despite this lack of a new label binding, the LFIB does not appear to indicate an obvious fault. The label swap looks like it should be working, but we see 0 bytes switched. The sham-links are actively trying to form, so this number should be increasing if XRv1 was functioning properly. XRv1, at this point, is not able to send traffic as there is no transport label binding.

```

RP/0/0/CPU0:XRv1#show mpls forwarding labels 91022

```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
91022	6059	24:2:fd00::2/128		10.6.110.110	0

CSR6 doesn't have this problem as its architecture is different. It is not adjusting the VPNv4 next-hop and it redistributes the connected host-route into IGP. As such, it can tunnel traffic to XRv1 without issue. Traffic with VPN label 91017 is tunneled to XRv1 through CSR6.

```

R6#show bgp vpnv6 unicast rd 13:2 FD00::8/128
BGP routing table entry for [13:2]FD00::8/128, version 1148
Paths: (1 available, best #1, no table)
  Advertised to update-groups:
    10
  Refresh Epoch 1
  (13)
    ::FFFF:10.6.11.11 (via default) from 10.6.11.11 (13.0.0.11)
    Origin IGP, metric 0, localpref 100, valid, confed-external, best

```

```
Extended Community: RT:13:2
mpls labels in/out nlabel/91017
rx pathid: 0, tx pathid: 0x0
```

```
R6#show mpls forwarding-table 10.6.11.11 32
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Outgoing interface	Next Hop
6074	Pop Label	10.6.11.11/32	28546	Gi2.561	10.6.11.11

XRv1 is able to receive these packets. It then performs the appropriate VPN label swap and transport label push operations as necessary. The byte counters increase, so we theorize that CSR2 can send traffic to CSR8, but not vice versa.

```
RP/0/0/CPU0:XRv1#show mpls forwarding labels 91017
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
91017	8016	13:2:fd00::8/128		13.0.0.8	24064

OSPFv3 debugging on CSR2 and CSR8 reveal this is true. CSR2 only shows LAN-side OSPFv3 exchanges with no sham-link activity. This indicates a problem with XRv1 forwarding MPLS traffic towards CSR6 as a result of the multi-hop BGP peer.

```
R8#debug ospfv3 vrf OSPF packet
```

```
OSPFv3 packet debugging is on for process 2, IPv4, vrf OSPF
OSPFv3 packet debugging is on for process 2, IPv6, vrf OSPF
```

```
OSPFv3-2-IPv6-OSPF PAK    SL1: Sham link packet: interface VRF ID 0, packet
VRF ID 2
```

```
OSPFv3-2-IPv6-OSPF PAK   : SL1: IN: FD00::2->FD00::8: ver:3 type:1 len:36
rid:10.2.9.2 area:0.0.0.0 chksum:EDD7 inst:0
```

```
OSPFv3-2-IPv4-OSPF PAK    SL0: Sham link packet: interface VRF ID 0, packet
VRF ID 2
```

```
OSPFv3-2-IPv4-OSPF PAK   : SL0: IN: FD00::2->FD00::8: ver:3 type:1 len:36
rid:10.2.9.2 area:0.0.0.0 chksum:ACD9 inst:64
```

```
R2#debug ospfv3 vrf OSPF packet
```

```
OSPFv3 packet debugging is on for process 2, IPv4, vrf OSPF
OSPFv3 packet debugging is on for process 2, IPv6, vrf OSPF
```

```
OSPFv3-2-IPv4-OSPF PAK   : Gi2.529: IN: FE80::9->FF02::5: ver:3 type:1 len:40
rid:10.4.9.9 area:0.0.0.0 chksum:9653 inst:64
```

```
OSPFv3-2-IPv6-OSPF PAK   : Gi2.529: OUT: FE80::2->FF02::5: ver:3 type:1 len:40
rid:10.2.9.2 area:0.0.0.0 chksum:D765 inst:0
```

```
OSPFv3-2-IPv4-OSPF PAK   : Gi2.529: OUT: FE80::2->FF02::5: ver:3 type:1 len:40
rid:10.2.9.2 area:0.0.0.0 chksum:9666 inst:64
```

```
OSPFv3-2-IPv6-OSPF PAK   : Gi2.529: IN: FE80::9->FF02::5: ver:3 type:1 len:40
rid:10.4.9.9 area:0.0.0.0 chksum:D752 inst:0und all
```

The most obvious solution to this problem would be to somehow inform XRv1 that it can use a null label in the stack as transport to reach 10.6.110.110/32. We know that it is connected to CSR6 so no long-range transport is necessary. Although sloppy, we can enable LDP between the peers to accomplish this. I use some intelligent outbound LDP label filtering on both XRv1 and CSR6. On CSR6, the sequence of filters does matter, so the more specific filter towards XRv1 is placed first. This allows CSR6 to only advertise the implicit-null label for 10.6.110.110/32 towards XRv1, while the internal peers can get all labels. XRv1 advertises no labels at all to CSR6, effectively making it a receive-only peer.

```
! CSR6
interface GigabitEthernet2.561
  mpls ip
  mpls ldp discovery transport-address interface

no mpls ldp advertise-labels
mpls ldp advertise-labels for ACL_BGP_LOOP to ACL_XRV1
mpls ldp advertise-labels for ACL_ANY to ACL_INTERNAL_PEERS

ip access-list standard ACL_ANY
  permit any
ip access-list standard ACL_BGP_LOOP
  permit 10.6.110.110
ip access-list standard ACL_INTERNAL_PEERS
  permit 24.0.0.0 0.0.0.255
ip access-list standard ACL_XRV1
  permit 13.0.0.11

! XRv1
ipv4 access-list ACL_DENY
  10 deny ipv4 any any

mpls ldp
  address-family ipv4
  label
  local
  advertise
  to 24.0.0.6:0 for ACL_DENY

interface GigabitEthernet0/0/0.561
  address-family ipv4
  discovery transport-address interface
```

We check XRv1 to see that the peer is up and exactly 1 IPv4 label was received. The label is imp-null and is bound to prefix 10.6.110.110/32. CSR6 receives no labels from XRv1 at all, as expected.

```
RP/0/0/CPU0:XRv1#show mpls ldp neighbor brief
```

Peer	GR	NSR	Up Time	Discovery	Addresses	Labels

				ipv4	ipv6	ipv4	ipv6	ipv4	ipv6
13.0.0.12:0	N	N	14:33:17	1	0	3	0	4	0
13.0.0.5:0	N	N	14:33:17	1	0	5	0	4	0
13.0.0.8:0	N	N	02:43:39	1	0	4	0	5	0
24.0.0.6:0	N	N	00:04:56	1	0	6	0	1	0

```
RP/0/0/CPU0:XRv1#show mpls ldp bindings neighbor 24.0.0.6
10.6.110.110/32, rev 17
```

```
Local binding: label: 91037
```

```
Remote bindings: (1 peers)
```

Peer	Label
24.0.0.6:0	ImpNull

```
R6#show mpls ldp bindings neighbor 13.0.0.11
[no output]
```

Now, we can verify XRv1's LFIB entry along the sham-link transit path to see bytes being switched. This fixes the MPLS forwarding problem that confederations (or any multi-hop eBGP session) may create on the transit links. Since the next-hop is recursive (not connected), it is acceptable to not have an outgoing interface in the LFIB.

```
RP/0/0/CPU0:XRv1#show mpls forwarding labels 91022
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
91022	6059	24:2:fd00::2/128		10.6.110.110	3188

We quickly check the sham-links to ensure they are up, then use traceroute within the VPN to verify it. This proves that XRv1 is now able to swap VPN labels between the sub-ASes.

```
R8#show ospfv3 vrf OSPF sham-links | include ^Sham
Sham Link OSPFv3_SL0 to address FD00::2 is up
Sham Link OSPFv3_SL1 to address FD00::2 is up
```

```
R4#traceroute 10.9.9.9 source 10.4.4.4
Type escape sequence to abort.
Tracing the route to 10.9.9.9
VRF info: (vrf in name/id, vrf out name/id)
 1 10.4.8.8 5 msec 4 msec 4 msec
 2 13.8.11.11 [MPLS: Label 91020 Exp 0] 10 msec 9 msec 9 msec
 3 10.6.11.6 [MPLS: Label 6046 Exp 0] 24 msec 30 msec 29 msec
 4 24.6.14.14 [MPLS: Labels 94009/2030 Exp 0] 32 msec 136 msec 21 msec
 5 10.2.9.2 [MPLS: Label 2030 Exp 0] 21 msec 21 msec 21 msec
 6 10.2.9.9 20 msec 11 msec 13 msec
```


The LDP solution was very configuration-intensive and is generally a bad practice. Instead, Cisco built the “mpls activate” XR command stanza discussed briefly earlier. This instructs BGP to perform a null label (implicit-null) rewrite on traffic going towards a multi-hop external or confed-external peer, rather than rely on LDP. To test this, I disable LDP on CSR6’s interface (not shown) but leave the existing LDP filters in place for reference. The LDP neighbor goes down on XRv1, and forwarding is broken again. Traceroute inside of a customer VPN confirms this. Once the sham-link fails, the OSPF VPN will have a backdoor link available, but since the sham-link is a demand circuit, it may never fail without some other OSPF/BGP reconvergence event as a catalyst. Since the VPNv4/v6 control-plane is working, this can be a dangerous situation as traffic is blackholed and the backdoor link cannot be utilized.

```
RP/0/0/CPU0:XRv1#show mpls ldp bindings neighbor 24.0.0.6
[no output]
```

```
R4#traceroute 10.9.9.9 source 10.4.4.4
Type escape sequence to abort.
Tracing the route to 10.9.9.9
VRF info: (vrf in name/id, vrf out name/id)
 1 10.4.8.8 6 msec 4 msec 3 msec
 2 * * *
```

We can see the result of having this broken by checking the global FIB. There is no label for this prefix, so all labels are removed from the stack. This is the fundamental problem seen earlier, now reintroduced as a result of disabling LDP.

```
RP/0/0/CPU0:XRv1#show cef 10.6.110.110
10.6.110.110/32, version 3991, internal 0x1000001 0x0 (ptr 0xa14476f4) [1],
0x0 (0xa14139bc), 0xa20 (0xa156d708)
local adjacency 10.6.11.6
Prefix Len 32, traffic index 0, precedence n/a, priority 4
via 10.6.11.6, GigabitEthernet0/0/0/0.561, 5 dependencies, weight 0, class
0 [flags 0x0]
path-idx 0 NHID 0x0 [0xa1085100 0x0]
next hop 10.6.11.6
local adjacency
local label 91037 labels imposed {None}
```

On XRv1, we simply enable the feature under BGP to specify the null label rewrite. This is a purpose-built command introduced specifically for this purpose, which makes it more wieldy than the LDP solution. It allows BGP to assume implicit-null for the multi-hop peer without relying on LDP at all. This is a way to bypass the typical MPLS label imposition process. Although BGP technically has no concept of “interfaces” per se, this feature simply allows BGP to shortcut the label stacking process intelligently.

```
! XRv1
router bgp 13
mpls activate
```

```
interface GigabitEthernet0/0/0/0.561
```

Checking the FIB, we can see implicit-null is now bound to this prefix, despite it not being LDP learned. Connectivity has been restored across the VPNs now that XRv1 can label-switch traffic between sub-ASes again.

```
RP/0/0/CPU0:XRv1#show cef 10.6.110.110
10.6.110.110/32, version 3997, internal 0x1000001 0x0 (ptr 0xa14476f4) [1],
0x0 (0xa14139bc), 0xa20 (0xa156dac8)
local adjacency 10.6.11.6
  Prefix Len 32, traffic index 0, precedence n/a, priority 4
  via 10.6.11.6, GigabitEthernet0/0/0/0.561, 5 dependencies, weight 0, class
0 [flags 0x0]
  path-idx 0 NHID 0x0 [0xa1085100 0xa10853a0]
  next hop 10.6.11.6
  local adjacency
    local label 91037      labels imposed {ImplNull}
```

```
R4#traceroute 10.9.9.9 source 10.4.4.4
Type escape sequence to abort.
Tracing the route to 10.9.9.9
VRF info: (vrf in name/id, vrf out name/id)
 1 10.4.8.8 6 msec 4 msec 4 msec
 2 13.8.11.11 [MPLS: Label 91020 Exp 0] 51 msec 9 msec 7 msec
 3 10.6.11.6 [MPLS: Label 6046 Exp 0] 25 msec 49 msec 29 msec
 4 24.6.14.14 [MPLS: Labels 94009/2030 Exp 0] 12 msec 32 msec 30 msec
 5 10.2.9.2 [MPLS: Label 2030 Exp 0] 16 msec 16 msec 15 msec
 6 10.2.9.9 20 msec 10 msec 9 msec
```

From CSR6's perspective, nothing terribly interesting has happened. We can test XE as well to ensure that, when recursively looking up remote BGP next-hops, confederations are still supported. I shut down XRv1's BGP session to CSR6 to force traffic across CSR5. CSR5 is reconfigured to peer with 10.6.110.110 much like XRv1. CSR6 just changes the update-source to loopback611.

```
! CSR6
router bgp 24
  neighbor 10.5.6.5 update-source Loopback611

! CSR5
router bgp 13
  no neighbor 10.5.6.6
  neighbor 10.6.110.110 remote-as 24
  neighbor 10.6.110.110 disable-connected-check

address-family ipv4 mvpn
  neighbor 10.6.110.110 activate
  neighbor 10.6.110.110 next-hop-self
```

```
address-family vpnv4
  neighbor 10.6.110.110 activate
  neighbor 10.6.110.110 next-hop-self
```

```
address-family ipv4 mdt
  neighbor 10.6.110.110 activate
  neighbor 10.6.110.110 next-hop-self
```

```
address-family ipv6 mvpn
  neighbor 10.6.110.110 activate
  neighbor 10.6.110.110 next-hop-self
```

```
address-family vpnv6
  neighbor 10.6.110.110 activate
  neighbor 10.6.110.110 next-hop-self
```

```
address-family l2vpn vpls
  neighbor 10.6.110.110 activate
  neighbor 10.6.110.110 next-hop-self
```

```
ip route 10.6.110.110 255.255.255.255 GigabitEthernet2.556 10.5.6.6
```

Checking CSR5's VPNv6 route to CSR2 for the sham-link, we can see it is reachable via CSR6's new loopback address. CSR5 has a static route for this peer just like Xrv1 did, which would suggest that CSR5 would have the same label binding problems.

```
R5#show bgp vpnv6 unicast rd 24:2 FD00::2/128
BGP routing table entry for [24:2]FD00::2/128, version 676
Paths: (1 available, best #1, no table)
  Advertised to update-groups:
    11
  Refresh Epoch 1
  (24)
    ::FFFF:10.6.110.110 (via default) from 10.6.110.110 (24.0.0.6)
      Origin IGP, metric 0, localpref 100, valid, confed-external, best
      Extended Community: RT:24:2
      mpls labels in/out 5038/6059
      rx pathid: 0, tx pathid: 0x0
```

```
R5#show ip route 10.6.110.110
Routing entry for 10.6.110.110/32
  Known via "static", distance 1, metric 0
  Routing Descriptor Blocks:
  * 10.5.6.6, via GigabitEthernet2.556
    Route metric is 0, traffic share count is 1
```

Checking the LFIB, we can see that label swap occurring with some byte counts. The next-hop automatically recurses to 10.5.6.6, which is different than XR. As long as the interface is configured for MPLS BGP forwarding, XE does not require any special configuration to support this architecture. A quick traceroute proves that the label switching is functional.

```
R5#show mpls forwarding-table labels 5038
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Label	Outgoing interface	Next Hop
5038	6059	[24:2]FD00::2/128	2044	\	Gi2.556	10.5.6.6

```
R9#traceroute 10.4.4.4 source 10.9.9.9
Type escape sequence to abort.
Tracing the route to 10.4.4.4
VRF info: (vrf in name/id, vrf out name/id)
 1 10.2.9.2 5 msec 4 msec 3 msec
 2 24.2.14.14 [MPLS: Labels 94005/5013 Exp 0] 9 msec 9 msec 9 msec
 3 24.6.14.6 [MPLS: Labels 6048/5013 Exp 0] 24 msec 31 msec 31 msec
 4 10.5.6.5 [MPLS: Label 5013 Exp 0] 30 msec 37 msec 30 msec
 5 10.4.8.8 [MPLS: Label 8015 Exp 0] 21 msec 20 msec 20 msec
 6 10.4.8.4 20 msec 11 msec 11 msec
```

If we temporarily remove “mpls bgp forwarding” from CSR5’s link to CSR6, forwarding does not work. This command is loosely equivalent to XR’s “mpls activate” except that XE requires it for options B and C for single and multi-hop BGP peers. XR only requires “mpls activate” for multi-hop peers. In any event, VPN traffic fails, and CSR5’s LFIB drops traffic since the VPN next-hop 10.6.110.110 is not reachable via an MPLS-enabled interface. Note: LDP is enabled on this transit link to support VPLS and has no effect on the BGP forwarding.

```
R9#traceroute 10.4.4.4 source 10.9.9.9
Type escape sequence to abort.
Tracing the route to 10.4.4.4
VRF info: (vrf in name/id, vrf out name/id)
 1 10.2.9.2 3 msec 3 msec 3 msec
 2 * * *
```

```
R5#show mpls forwarding-table labels 5038
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Label	Outgoing interface	Next Hop
5038	6059	[24:2]FD00::2/128	2236	\	drop	

Before continuing, I restore all BGP sessions in the network. A quick check on CSR6 and CSR5 will show that all sessions have been restored. I scan the right-most column for any positive integer, indicating that prefixes are being received from each peer.

```
R5#show bgp vpnv4 unicast all summary | begin ^Neigh
```

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
10.5.7.7	4	24	55	46	326	0	0	00:00:57	7
10.6.110.110	4	24	76	93	326	0	0	00:38:39	7
13.0.0.12	4	13	1750	1634	326	0	0	03:49:02	8

```
R5#show bgp vpnv6 unicast all summary | begin ^Neigh
```

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
10.5.7.7	4	24	55	46	695	0	0	00:00:57	6
10.6.110.110	4	24	76	93	695	0	0	00:38:39	6
13.0.0.12	4	13	1750	1634	695	0	0	03:49:02	9

```
R6#show bgp vpnv4 unicast all summary | begin ^Neigh
```

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
10.5.6.5	4	13	93	76	783	0	0	00:38:36	8
10.6.11.11	4	13	20	37	783	0	0	00:00:33	8
24.0.0.2	4	24	2184	1688	783	0	0	03:50:00	7

```
R6#show bgp vpnv6 unicast all summary | begin ^Neigh
```

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
10.5.6.5	4	13	93	76	1262	0	0	00:38:36	8
10.6.11.11	4	13	20	37	1262	0	0	00:00:33	8
24.0.0.2	4	24	2184	1688	1262	0	0	03:50:00	6

MVPN works the same as it does in option B. Because mLDP is not supported across XE-based ASBRs, we will focus on PIM/GRE inside the EIGRP VPN. The default MDT uses P-group 232.13.24.255 and is exchanged across the intraconfederation links. CSR2, for example, receives the intra-AS route from XRv4 and two intraconfederation routes from CSR6 and CSR7. The next-hops for the intraconfederation routes are CSR5's connected interfaces, which is acceptable as CSR2 has IGP routes to these host addresses.

```
R2#show bgp ipv4 mdt all | begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 13:3					
* i 13.0.0.12/32	10.5.7.5	0	100	0	(13) i
*>i	10.5.6.5	0	100	0	(13) i
Route Distinguisher: 24:3 (default for vrf EIGRP)					
*> 24.0.0.2/32	0.0.0.0			0	?
*>i 24.0.0.14/32	24.0.0.14		100	0	i

Looking at the routes from the ASBRs, the most significant component of the MDT update is the default group address. This address must match within a confederation for the MVPN to form.

```
R2#show bgp ipv4 mdt all 13.0.0.12
```

```
BGP routing table entry for 13:3:13.0.0.12/32 version 16
```

```
Paths: (2 available, best #2, table IPv4-MDT-BGP-Table)
```

```
Advertised to update-groups:
```

```
1
```

```
Refresh Epoch 1
```

```

(13), (Received from a RR-client)
  10.5.7.5 from 24.0.0.7 (24.0.0.7)
    Origin IGP, metric 0, localpref 100, valid, confed-internal,
    MDT group address: 232.13.24.255
    rx pathid: 0, tx pathid: 0
Refresh Epoch 1
(13), (Received from a RR-client)
  10.5.6.5 from 24.0.0.6 (24.0.0.6)
    Origin IGP, metric 0, localpref 100, valid, confed-internal, best,
    MDT group address: 232.13.24.255
    rx pathid: 0, tx pathid: 0x0

```

CSR2 sees both XRv2 and XRv4 as neighbors, which shows that it can receive traffic along the MDT. A more thorough verification would involve checking the other neighbors as well, but the architecture is identical to option B.

```

R2#show ip pim vrf EIGRP neighbor | begin ^Neigh
Neighbor          Interface          Uptime/Expires    Ver    DR
Address
10.1.2.1          GigabitEthernet2.512    20:06:07/00:01:34 v2      1 / S P G
13.0.0.12         Tunnel17                 00:54:05/00:01:35 v2      1 / P G
24.0.0.14         Tunnel17                 04:02:38/00:01:42 v2      1 / DR G

```

The BGP connector attribute is used to ensure that traffic arriving from the PMSI comes from the originator of the VPN route. In this case, that is 13.0.0.12, which matches the PIM neighbor above. RPF should pass for traffic arriving at CSR2. When the BGP connector attribute is present, it is evaluated in place of the BGP next-hop. Since 10.5.6.5 is not a PIM neighbor nor the MDT source of the traffic, the connector attribute helps forms the VPN end-to-end across an option B deployment.

```

R2#show bgp vpnv4 unicast vrf EIGRP 10.3.3.3/32
BGP routing table entry for 24:3:10.3.3.3/32, version 1388
Paths: (1 available, best #1, table EIGRP)
  Not advertised to any peer
  Refresh Epoch 2
  (13), (Received from a RR-client), imported path from 13:3:10.3.3.3/32
(global)
  10.5.6.5 (metric 20) (via default) from 24.0.0.6 (24.0.0.6)
    Origin incomplete, metric 10880, localpref 100, valid, confed-internal,
best
  Extended Community: RT:13:3 Cost:pre-bestpath:128:10880 0x8800:32768:0
    0x8801:3:288 0x8802:65281:2560 0x8803:1:1500 0x8806:0:167971843
  Connector Attribute: count=1
    type 1 len 12 value 13:3:13.0.0.12
  mpls labels in/out nolabel/5061
  rx pathid: 0, tx pathid: 0x0

```

The PIM vector is still in play so that any core routers in AS 24 can successfully perform RPF lookups for the P-source 13.0.0.12. The P(S,G) join moves from CSR2 > CSR7 > CSR6 as shown below by way of the PIM vector.

```
R2#show ip mroute proxy
(13.0.0.12, 232.13.24.255)
  Proxy          Assigner      Origin      Uptime/Expire
  13:3/10.5.6.5   0.0.0.0      BGP MDT     00:58:53/stopped

R7#show ip mroute proxy
(13.0.0.12, 232.13.24.255)
  Proxy          Assigner      Origin      Uptime/Expire
  13:3/10.5.6.5   24.2.7.2     PIM         01:01:58/00:02:11

R6#show ip mroute proxy
(13.0.0.12, 232.13.24.255)
  Proxy          Assigner      Origin      Uptime/Expire
  13:3/10.5.6.5   24.6.7.7     PIM         00:59:05/00:02:44
```

CSR7 cannot perform an RPF lookup for 13.0.0.12, so it uses the PIM vector instead. This is originated by the PE, CSR2, via BGP MDT. The PIM vector is the BGP next-hop which all routers in the AS should be able to reach. All of this information is the same as in option B.

```
R7#show ip mroute 232.13.24.255 13.0.0.12 | begin \
(13.0.0.12, 232.13.24.255), 03:39:30/00:02:44, flags: sTV
  Incoming interface: GigabitEthernet2.567, RPF nbr 24.6.7.6, vector 10.5.6.5
  Outgoing interface list:
    GigabitEthernet2.574, Forward/Sparse, 00:59:57/00:02:33
    GigabitEthernet2.527, Forward/Sparse, 00:59:57/00:02:44

R7#show ip rpf 13.0.0.12
  failed, no route exists

R7#show ip rpf 10.5.6.5
RPF information for ? (10.5.6.5)
  RPF interface: GigabitEthernet2.567
  RPF neighbor: ? (24.6.7.6)
  RPF route/mask: 10.5.6.5/32
  RPF type: unicast (isis 24)
  Doing distance-preferred lookups across tables
  RPF topology: ipv4 multicast base, originated from ipv4 unicast base
```

XRv3 is able to learn the C-PIM RP information, which is an indication that the intraconfederation MDT is functional.

```
RP/0/0/CPU0:XRv3#show pim rp mapping
PIM Group-to-RP Mappings
```

```
Group(s) 224.0.0.0/4
  RP 10.3.3.3 (?), v2
    Info source: 10.1.13.1 (?), elected via bsr, priority 0, holdtime 150
    Uptime: 00:05:08, expires: 00:02:25
```

As a final check, CSR3 begins sending pings to the C-group of 225.13.13.13 which is joined on XRv3. XRv3 receives packets from CSR1 which is an indication that the MVPN is functioning properly. The mechanics of this MVPN design are identical whether a confederation is used or not.

```
R3#ping ip
Target IP address: 225.13.13.13
Repeat count [1]: 10000
Datagram size [100]:
Timeout in seconds [2]: 1
Extended commands [n]: y
Interface [All]: loopback0
Time to live [255]:
Source address or interface: loopback0
[snip]
```

```
RP/0/0/CPU0:XRv3#show mfib route 225.13.13.13 10.3.3.3 | begin 225
(10.3.3.3,225.13.13.13),    Flags:
  Up: 00:00:51
  Last Used: 00:00:00
  SW Forwarding Counts: 51/51/5100
  SW Replication Counts: 51/0/0
  SW Failure Counts: 0/0/0/0/0
  Loopback0 Flags:  IC NS EG, Up:00:00:51
  GigabitEthernet0/0/0/0.513 Flags:  A, Up:00:00:51
```

Additional Reading – Reference configurations "inter-as-mpls-b-confed"

8.4.3 Option C (ASBR eBGP + Label, RR VPNv4 eBGP)

Option C is a highly scalable but highly complex inter-AS VPN solution. The VPN advertisement logic is similar to CSC or UMPLS since the MPLS service label never changes. That is, there is no ASBR performing a VPNv4/v6 or PW label swap as seen in option B. Instead, BGP labeled-unicast is used between the ASBRs so that labels can be allocated for remote PE loopbacks, which implies PE loopbacks are leaked between ASes. This is considered highly insecure and requires very close coordination between providers, but in doing this, the MPLS services are truly end-to-end. BGP imposes a second label over the transit links so that the bottom-most label is never swapped. For maximum scalability, the RRs in each AS are peered so the PEs need not run inter-AS eBGP sessions. As a result, the total number of new BGP sessions to configure is the sum of the RRs and ASBRs, which is a small number compared to options A and B. Continuing from option B, our ASBRs have very complex BGP configurations since they are supporting every “service” AFI we tested between ASes.

8.4.3.1 L3VPN

The first thing to do in this configuration is to establish the basic VPNv4/v6 peers within an AS between the PEs and RRs. Technically there is no need for the RRs to identify their single VPNv4/v6 peer as an RR-client, but I will do it as a best practice since it becomes significant if there are more PEs later. We will begin with AS 24; I do not show the basic peer-session parameters again.

```
! CSR2
router bgp 24
  address-family vpnv4
    neighbor 24.0.0.14 activate
    neighbor 24.0.0.14 route-reflector-client
  address-family vpnv6
    neighbor 24.0.0.14 activate
    neighbor 24.0.0.14 route-reflector-client

! XRv4
router bgp 24
  address-family vpnv4 unicast
  address-family vpnv6 unicast

  neighbor 24.0.0.2
    address-family vpnv4 unicast
    address-family vpnv6 unicast
```

From CSR2, we verify the session comes up. In the basic inter-AS verification, we also configured all of the PE-CE interactions, to include redistribution. We do not need to verify this again as it is basic and unrelated to option C.

```
R2#show bgp vpnv4 unicast all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
24.0.0.14     4      24      32      41      6563   0    0 00:02:48      3

R2#show bgp vpnv6 unicast all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
24.0.0.14     4      24      33      42      6065   0    0 00:02:56      4
```

A quick traceroute from CSR1 to XRv3 shows that the L3VPN within AS 24 is functional. The LSP is very short but this verifies the core MPLS components like VPNv4, CEF, LDP, and redistribution.

```
R1#traceroute 10.13.13.13 source 10.1.1.1
Type escape sequence to abort.
Tracing the route to 10.13.13.13
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.2.2 4 msec 4 msec 4 msec
 2 24.2.14.14 [MPLS: Label 94006 Exp 0] 5 msec 3 msec 4 msec
```

```
3 10.13.14.13 4 msec 12 msec 13 msec
```

Next, I will configure the same features in AS 13. XRv2 is a PE for the EIGRP VPN and also the RR for the AS.

```
! XRv2
router bgp 13
  bgp cluster-id 13.0.0.12
  address-family vpnv4 unicast
  address-family vpnv6 unicast

  af-group VPNV4 address-family vpnv4 unicast
    route-reflector-client
  af-group VPNV6 address-family vpnv6 unicast
    route-reflector-client

neighbor 13.0.0.8
  address-family vpnv4 unicast
    use af-group VPNV4
  address-family vpnv6 unicast
    use af-group VPNV6

! CSR8
router bgp 13
  address-family vpnv4
    neighbor 13.0.0.12 activate
  address-family vpnv6
    neighbor 13.0.0.12 activate
```

We verify that all of the BGP AFIs were successfully negotiated on XRv2.

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast summary | begin ^Neigh
Neighbor      Spk    AS  MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
13.0.0.8      0     13   33429   32516    1620    0    0  00:05:08      7

RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast summary | begin ^Neigh
Neighbor      Spk    AS  MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
13.0.0.8      0     13   33429   32516    1695    0    0  00:05:11      7
```

Next, I configure the RT policies on VRF EIGRP and VRF BGP so that VRF BGP is the central services VPN. We did this earlier for all other labs so this is nothing new. I only do this in order to quickly verify the local L3VPN connectivity within AS 13 before continuing.

```
! CSR8
vrf definition BGP
  address-family ipv4
    route-target export 13:1
    route-target import 13:3
```

```

address-family ipv6
  route-target export 13:1
  route-target import 13:3

! Xrv2
vrf EIGRP
  address-family ipv4 unicast
    import route-target
      13:1
    export route-target
      13:3

  address-family ipv6 unicast
    import route-target
      13:1
    export route-target
      13:3

```

Using traceroute from CSR2 to CSR10, we can see that the intra-AS L3VPN is operational.

```

R3#traceroute 110.0.0.0 source 10.3.3.3
Type escape sequence to abort.
Tracing the route to 110.0.0.0
VRF info: (vrf in name/id, vrf out name/id)
 1 10.3.12.12 3 msec 3 msec 1 msec
 2 10.8.10.8 [MPLS: Label 8003 Exp 0] 5 msec 3 msec 3 msec
 3 10.8.10.10 4 msec 4 msec 4 msec

```

Next, we will prepare the ASBRs by configuring their transit links. I include the inter-AS TE configurations as well since these commands were discussed in the option B section. Like option B, there is a single transit link in the global table, so the configuration is straightforward. PIM is enabled with BSR-border to enable GRE-encapsulated multicast between both ASes without the RP information leaking across. There is nothing new about these interface configurations but I show them for reference.

```

! CSR6
interface GigabitEthernet2.556
  encapsulation dot1Q 3556
  ip address 10.5.6.6 255.255.255.0
  ip pim bsr-border
  ip pim sparse-mode
  ipv6 address FE80::6 link-local
  ipv6 address FD00:10:5:6::6/64
  mpls traffic-eng tunnels
  mpls traffic-eng passive-interface nbr-te-id 13.0.0.5 nbr-if-addr 10.5.6.5
  ip rsvp bandwidth 200000

interface GigabitEthernet2.561
  encapsulation dot1Q 3561

```

```

ip address 10.6.11.6 255.255.255.0
ip pim bsr-border
ip pim sparse-mode
ipv6 address FE80::6 link-local
ipv6 address FD00:10:6:11::6/64

! XRv1
interface GigabitEthernet0/0/0/0.561
  ipv4 address 10.6.11.11 255.255.255.0
  ipv6 address fe80::11 link-local
  ipv6 address fd00:10:6:11::11/64
  encapsulation dot1q 3561

router pim
  address-family ipv4
    interface GigabitEthernet0/0/0/0.561
      bsr-border

mpls traffic-eng
  interface GigabitEthernet0/0/0/0.561

! CSR5
interface GigabitEthernet2.556
  encapsulation dot1Q 3556
  ip address 10.5.6.5 255.255.255.0
  ip pim bsr-border
  ip pim sparse-mode
  ipv6 address FE80::5 link-local
  ipv6 address FD00:10:5:6::5/64
  mpls traffic-eng tunnels
  mpls traffic-eng passive-interface nbr-te-id 24.0.0.6 nbr-if-addr 10.5.6.6
  ip rsvp bandwidth 200000

interface GigabitEthernet2.557
  encapsulation dot1Q 3557
  ip address 10.5.7.5 255.255.255.0
  ip pim bsr-border
  ip pim sparse-mode
  ipv6 address FE80::5 link-local
  ipv6 address FD00:10:5:7::5/64
  mpls traffic-eng tunnels
  mpls traffic-eng passive-interface nbr-te-id 24.0.0.7 nbr-if-addr 10.5.7.7
  ip rsvp bandwidth 200000

! CSR7
interface GigabitEthernet2.557
  encapsulation dot1Q 3557
  ip address 10.5.7.7 255.255.255.0
  ip pim bsr-border

```

```

ip pim sparse-mode
ipv6 address FE80::7 link-local
ipv6 address FD00:10:5:7::7/64
mpls traffic-eng tunnels
mpls traffic-eng passive-interface nbr-te-id 13.0.0.5 nbr-if-addr 10.5.7.5
ip rsvp bandwidth 200000

```

Next, we will configure the eBGP labeled-unicast sessions. BGP should be configured to allocate labels only for internal loopback addresses in each AS as a best practice. This is valuable when these routers are also exchanging Internet routing tables and allocating labels for them is a waste of resources. In this test, I ensure that only these loopbacks can be exchanged at all, which ensures that a redistribution error in the local AS does not advertise additional prefixes to the eBGP peer. In XE, I use a route-map to identify which routes can be advertised with labels bound. XR decouples these functions by specifying an RPL for label allocation and for prefix advertisement. Using parameterization, I can re-use the RPL. Only CSR6 and XRv1 configurations are shown for brevity. Note that only PE loopbacks need to be advertised, but I build more generic RPLs so that adding PEs later doesn't require the BGP filters to change.

```

! CSR6
ip prefix-list PL_LOCAL_LOOPBACKS seq 5 permit 24.0.0.0/24 ge 32

route-map RM_IPV4_LUCAST_EBGP permit 10
  match ip address prefix-list PL_LOCAL_LOOPBACKS
  set mpls-label

router bgp 24
  no bgp default ipv4-unicast
  neighbor 10.5.6.5 remote-as 13
  neighbor 10.6.11.11 remote-as 13
  address-family ipv4
    neighbor 10.5.6.5 activate
    neighbor 10.5.6.5 route-map RM_IPV4_LUCAST_EBGP out
    neighbor 10.5.6.5 send-label
    neighbor 10.6.11.11 activate
    neighbor 10.6.11.11 route-map RM_IPV4_LUCAST_EBGP out
    neighbor 10.6.11.11 send-label

! XRv1
prefix-set PS_LOCAL_LOOPBACKS
  13.0.0.0/24 ge 32
end-set

route-policy RPL_IF_DEST_PASS($PS)
  if destination in $PS then
    pass
  endif
end-policy

```

```

router bgp 13
  address-family ipv4 unicast
    allocate-label route-policy RPL_IF_DEST_PASS (PS_LOCAL_LOOPBACKS)

  neighbor 10.6.11.6
    remote-as 24
    address-family ipv4 labeled-unicast
      route-policy RPL_PASS in
      route-policy RPL_IF_DEST_PASS (PS_LOCAL_LOOPBACKS) out

```

Once the configuration is complete, I will check the neighbor details to ensure the IPv4 MPLS label capability is negotiated. In XR, there is a specific AFI for it, which simplifies the logic in the parser. In XE, we need to specifically check for this capability exchange rather than look at the IPv4 unicast summary. Since the capabilities is exchanged bidirectionally with all peers, we can assume the configuration is correct so far.

```

R6#show bgp ipv4 unicast neighbors | include ^BGP|MPLS_Label
BGP neighbor is 10.5.6.5, remote AS 13, external link
  ipv4 MPLS Label capability: advertised and received
BGP neighbor is 10.6.11.11, remote AS 13, external link
  ipv4 MPLS Label capability: advertised and received

```

```

R5#show bgp ipv4 unicast neighbors | include ^BGP|MPLS_Label
BGP neighbor is 10.5.6.6, remote AS 24, external link
  ipv4 MPLS Label capability: advertised and received
BGP neighbor is 10.5.7.7, remote AS 24, external link
  ipv4 MPLS Label capability: advertised and received

```

Next, we need to exchange loopbacks across AS boundaries. The first option is to redistribute them into BGP, and then in the remote AS, redistribute them back into IGP. We will do this in AS 24. CSR7 will use network statements as this is a very simple way to advertise specific PE loopbacks without worrying about redistributing too many routes. The downside is that when new PEs are added in the future, the ASBR must be updated to advertise the new PE loopbacks. We can verify the network statements work by checking to see if BGP allocates local labels on CSR7. We see labels 7000 and 7002, which is appropriate.

```

! CSR7
router bgp 24
  address-family ipv4
    network 24.0.0.2 mask 255.255.255.255
    network 24.0.0.14 mask 255.255.255.255

```

```

R7#show bgp ipv4 unicast labels
  Network          Next Hop          In label/Out label
  24.0.0.2/32      24.7.14.14      7000/nolabel

```

```
24.0.0.14/32      24.7.14.14      7002/nolabel
```

Quickly checking CSR5, we can see these routes are learned from 10.5.7.7 (CSR7) and carry the proper outgoing labels.

```
R5#show bgp ipv4 unicast labels
  Network          Next Hop      In label/Out label
  24.0.0.2/32       10.5.7.7       nolabel/7000
  24.0.0.14/32      10.5.7.7       nolabel/7002
```

An alternative approach is to use redistribution. CSR6 will redistribute PE loopbacks from IS-IS into BGP using an intelligent filter. We could re-use the route-map/prefix-list from earlier, but that will redistribute the core and ASBR loopbacks as well as the prefix-list is not host-specific. Being specific with the IGP-to-BGP redistribution will minimize the security risks of leaking loopbacks by specifically targeting only the PE routers. To be fancy, I use IS-IS route-tags to make this a semi-dynamic process, which means we won't have to change the BGP configuration on CSR6 when new PEs are added, provided we tag the loopback properly. First, we need to add the route-tags to CSR2 and XRv4 loopback interfaces and verify success.

```
! CSR2
interface Loopback0
  isis tag 24

! XRv4
router isis 24
  interface Loopback0
    address-family ipv4 unicast
    tag 24
```

Checking CSR6, we can see the route-tags locally. These are carried inside the IS-IS LSPs.

```
R6#show ip route 24.0.0.2
Routing entry for 24.0.0.2/32
  Known via "isis", distance 115, metric 20
  Tag 24, type level-2
  Redistributing via isis 24
  Last update from 24.6.14.14 on GigabitEthernet2.564, 00:05:18 ago
  Routing Descriptor Blocks:
  * 24.6.14.14, from 24.0.0.2, 00:05:18 ago, via GigabitEthernet2.564
    Route metric is 20, traffic share count is 1
    Route tag 24
```

```
R6#show ip route 24.0.0.14
Routing entry for 24.0.0.14/32
  Known via "isis", distance 115, metric 10
  Tag 24, type level-2
```

```

Redistributing via isis 24
Last update from 24.6.14.14 on GigabitEthernet2.564, 00:04:57 ago
Routing Descriptor Blocks:
* 24.6.14.14, from 24.0.0.14, 00:04:57 ago, via GigabitEthernet2.564
  Route metric is 10, traffic share count is 1
  Route tag 24

```

CSR6's configuration becomes simple at this point as we create a route-map to match tag 24 for redistribution from IS-IS into BGP. The outbound BGP filter affords us some additional protection by only advertising (and allocating labels for) routes that match the host-route filter.

```

! CSR6
route-map RM_ISIS_TO_BGP permit 10
  match tag 24

router bgp 24
  address-family ipv4
    redistribute isis 24 level-2 route-map RM_ISIS_TO_BGP

```

We confirm it by checking CSR6 for local label allocation. We see labels 6004 and 6003 which are valid local labels for CSR6.

```

R6#show bgp ipv4 unicast labels

```

Network	Next Hop	In label/Out label
24.0.0.2/32	24.6.14.14	6004/nolabel
24.0.0.14/32	24.6.14.14	6003/nolabel

When we check CSR5, we can see these labels are distributed properly. CSR5 learns the same pair of loopbacks from CSR7 and CSR6 as expected.

```

R5#show bgp ipv4 unicast labels

```

Network	Next Hop	In label/Out label
24.0.0.2/32	10.5.6.6	nolabel/6004
	10.5.7.7	nolabel/7000
24.0.0.14/32	10.5.6.6	nolabel/6003
	10.5.7.7	nolabel/7002

XRv1 also learns these labels, but it also allocates local labels for them. We noticed that CSR5 made no effort to do this and there is "nolabel" assigned. The reason is because XR will always allocate a local label for a prefix if it receives a label from a peer, but XE does not. The logic of XE is that because ALL of the IPv4 labeled-unicast peers on CSR5 are only configured to allocate labels for routes matching 13.0.0.0/24 ge 32, there is no possibility of BGP being able to label switch traffic to these destinations. This is specific to BGP; just because BGP is not aware of a local/remote label doesn't mean RSVP, LDP, or other mechanisms cannot fulfill that role. We will see these LSP connections later when we evaluate the LFIB.


```
RP/0/0/CPU0:XRv1#show bgp ipv4 labeled-unicast labels | begin Network
  Network          Next Hop          Rcvd Label      Local Label
*> 24.0.0.2/32      10.6.11.6          6004            91004
*> 24.0.0.14/32    10.6.11.6          6003            91005
```

Next, we need to configure AS 13 to advertise its PE loopbacks to AS 24. The IGP-to-BGP option is one option and is certainly valid, but we can alternatively run IPv4 labeled-unicast internally within an AS to avoid the redistribution entirely. On all routers in AS 13, I enable IPv4 labeled-unicast. The local PEs can simply advertise their connected loopbacks directly into BGP without the ASBRs having to redistribute it from IGP. The benefit of this approach is protecting the P (core) routers as they will not have to learn the remote PE loopbacks via IGP. In these small networks, this benefit does not really materialize, but with many core routers and potentially many remote PE loopbacks, this can be significant. This requires more BGP configuration on the PEs and ASBRs, as well as an additional MPLS label in the data plane, which are small drawbacks. Labels for this AFI are also allocated for local loopbacks using identical configurations seen on XRv1.

```
! XRv2
prefix-set PS_LOCAL_LOOPBACKS
  13.0.0.0/24 ge 32
end-set

route-policy RPL_IF_DEST_PASS($PS)
  if destination in $PS then
    pass
  endif
end-policy

router bgp 13
  address-family ipv4 unicast
    network 13.0.0.12/32
    allocate-label route-policy RPL_IF_DEST_PASS(PS_LOCAL_LOOPBACKS)

  af-group IPV4_LUCAST address-family ipv4 labeled-unicast
    route-reflector-client

  neighbor 13.0.0.5
    use session-group IBGP
    address-family ipv4 labeled-unicast
    use af-group IPV4_LUCAST

  neighbor 13.0.0.8
    address-family ipv4 labeled-unicast
    use af-group IPV4_LUCAST

  neighbor 13.0.0.11
    use session-group IBGP
```

```

address-family ipv4 labeled-unicast
  use af-group IPV4_LUCAST

! XRv1
router bgp 13
  neighbor 13.0.0.12
    remote-as 13
    timers 10 40
    password encrypted 11203B22274358
    update-source Loopback0
  address-family ipv4 labeled-unicast

! CSR5
router bgp 13
  neighbor 13.0.0.12 remote-as 13
  neighbor 13.0.0.12 password IBGP13
  neighbor 13.0.0.12 update-source Loopback0
  neighbor 13.0.0.12 timers 10 40
  address-family ipv4
    neighbor 13.0.0.12 activate
    neighbor 13.0.0.12 send-label

! CSR8
router bgp 13
  neighbor 13.0.0.12 remote-as 13
  neighbor 13.0.0.12 password IBGP13
  neighbor 13.0.0.12 update-source Loopback0
  neighbor 13.0.0.12 timers 10 40
  address-family ipv4
    network 13.0.0.8 mask 255.255.255.255
    neighbor 13.0.0.12 activate
    neighbor 13.0.0.12 send-label

```

First we check the BGP sessions for this AFI on XRv2. We can see all 3 sessions come up. We can also make sense out of the route counters. CSR8 and XRv2 both advertise their PE loopbacks into BGP via the network statement so that the ASBRs don't need to. This is why there is 1 route (13.0.0.8/32) from CSR8. XRv1 and CSR5 are both advertising the 2 routes learned from AS 24.

```

RP/0/0/CPU0:XRv2#show bgp ipv4 labeled-unicast summary | begin ^Neighbor
Neighbor      Spk    AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
13.0.0.5      0     13      23       18         3    0    0 00:02:38      2
13.0.0.8      0     13   33928   32975         3    0    0 00:00:59      1
13.0.0.11     0     13      22       19         3    0    0 00:02:55      2

```

When we check the route information, XRv2 cannot select a best-path for any of the routes from AS 24. This is because the ASBRs did not adjust the next-hop when sending traffic to XRv2. When we tested option B, AS 13 ASBRs used next-hop-self while AS 24 ASBRs advertised the transit links into IGP. Both were valid techniques, and remain so for option C. This time, AS 13 ASBRs will advertise the transit link

host addresses into IGP. Remember that XR routers require a /32 route to the BGP next-hop for VPN routes, so we cannot advertise the transit /24.

```
RP/0/0/CPU0:XRv2#show bgp ipv4 labeled-unicast | begin Network
  Network                Next Hop                Metric LocPrf Weight Path
* >i13.0.0.8/32          13.0.0.8                  0      100      0 i
* i24.0.0.2/32          10.5.7.7                  20      100      0 24 i
* i                      10.6.11.6                 20      100      0 24 ?
* i24.0.0.14/32         10.5.7.7                  10      100      0 24 i
* i                      10.6.11.6                 10      100      0 24 ?
```

The configuration on CSR5 is simple as there are auto-generated /32 connected routes added to the RIB when eBGP sessions relying on label exchange (IPv4/v6 labeled unicast, VPNv4/v6 unicast, etc) form across AS boundaries. A quick check of the MPLS interfaces proves it as these links are not LDP enabled but are BGP enabled.

```
R5#show mpls interfaces
Interface                IP                Tunnel    BGP  Static Operational
GigabitEthernet2.551     Yes (ldp)         Yes       No   No        Yes
GigabitEthernet2.556     No                Yes       Yes  No        Yes
GigabitEthernet2.557     No                Yes       Yes  No        Yes
GigabitEthernet2.558     Yes (ldp)         Yes       No   No        Yes
```

Checking the RIB, we can see the host routes for the eBGP peer. This is what must be redistributed into IGP.

```
R5#show ip route 10.5.6.6
Routing entry for 10.5.6.6/32
  Known via "connected", distance 0, metric 0 (connected, via interface)
  Routing Descriptor Blocks:
  * directly connected, via GigabitEthernet2.556
    Route metric is 0, traffic share count is 1
```

```
R5#show ip route 10.5.7.7
Routing entry for 10.5.7.7/32
  Known via "connected", distance 0, metric 0 (connected, via interface)
  Routing Descriptor Blocks:
  * directly connected, via GigabitEthernet2.557
    Route metric is 0, traffic share count is 1
```

To redistribute them, I create the most specific all-encompassing prefix-list possible (for additional practice). Quickly checking the OSPF LSDB, we can see only these two host routes were redistributed.

```
! CSR5
ip prefix-list PL_TRANSIT_HOST_ROUTES seq 5 permit 10.5.6.0/23 ge 32
route-map RM_CONN_TO_OSPF permit 10
  match ip address prefix-list PL_TRANSIT_HOST_ROUTES
```

```
router ospf 13
 redistribute connected subnets route-map RM_CONN_TO_OSPF
```

```
R5#show ip ospf database | begin -5
```

Type-5 AS External Link States

Link ID	ADV Router	Age	Seq#	Checksum	Tag
10.5.6.6	13.0.0.5	25	0x80000001	0x00175C	0
10.5.7.7	13.0.0.5	25	0x80000001	0x00026F	0

Because these are host routes, the LDP label allocation filter still allocates local labels for these. As seen in the option B design, CSR5 does not allocate null labels for these prefixes despite them being “connected”. They are not local routes, and because CSR5 has not adjusted the BGP next-hop, it must tunnel inter-AS traffic to CSR6 or CSR7 where the BGP label swap can occur. Revealing the BGP labeled-unicast label to CSR5, if allocated by CSR6 or CSR7, will result in the packet being dropped.

```
R5#show mpls ldp bindings 10.5.6.6 32 local
```

```
lib entry: 10.5.6.6/32, rev 30
```

```
local binding: label: 5048
```

```
R5#show mpls ldp bindings 10.5.7.7 32 local
```

```
lib entry: 10.5.7.7/32, rev 32
```

```
local binding: label: 5015
```

On XRv1, no such connected host route exists. XR only sees a /24 for the remote eBGP peer. While this is good enough for a BGP session to form, it will not suffice for MPLS forwarding.

```
RP/0/0/CPU0:XRv1#show route 10.6.11.6
```

Routing entry for 10.6.11.0/24

```
Known via "connected", distance 0, metric 0 (connected)
```

```
Routing Descriptor Blocks
```

```
directly connected, via GigabitEthernet0/0/0/0.561
```

```
Route metric is 0
```

```
No advertising protos.
```

The common fix for this problem is to add a static route host to the eBGP peer. This is the same configuration we used in the option B configuration, and we apply it again. Once we add this route, the requirement for the BGP next-hop to be a /32 host route is met. I add a route-tag of 13 for reasons discussed next.

```
! XRv1
```

```
router static
```

```
address-family ipv4 unicast
```

```
10.6.11.6/32 GigabitEthernet0/0/0/0.561 tag 13
```

```
RP/0/0/CPU0:XRv1#show route 10.6.11.6
Routing entry for 10.6.11.6/32
  Known via "static", distance 1, metric 0 (connected)
  Routing Descriptor Blocks
    directly connected, via GigabitEthernet0/0/0/0.561
      Route metric is 0
  No advertising protos.
```

Like CSR5, XRv1 must redistribute this host route into IGP and allocate a local label for it via LDP. Rather than redistribute from connected, XRv1 will redistribute from static. The logic is identical except the route-source changes. Because the static route can be tagged, I use this technique rather than a prefix-based matching configuration for a more generalized solution.

```
! XRv1
route-policy RPL_IF_TAG_PASS($TAG)
  if tag is $TAG then
    pass
  endif
end-policy

router ospf 13
  redistribute static route-policy RPL_IF_TAG_PASS(13)
```

Checking the OSPF LSDB on XRv1, we can see 3 total external LSAs. Two were originated from CSR5 which we verified earlier, and one new one is originated from XRv1. We can see the obvious age difference and route tag on the transit link to CSR6 as redistributed by XRv1. This is because CSR5 was configured a few minutes before XRv1.

```
RP/0/0/CPU0:XRv1#show ospf database | begin -5
Type-5 AS External Link States
```

Link ID	ADV Router	Age	Seq#	Checksum	Tag
10.5.6.6	13.0.0.5	688	0x80000001	0x00175c	0
10.5.7.7	13.0.0.5	688	0x80000001	0x00026f	0
10.6.11.6	13.0.0.11	4	0x80000001	0x009abf	13

Just like CSR5, XRv1 allocates a non-null local label in LDP for this prefix. This allows the BGP label to be tunneled to CSR6 so it can be swapped there; since XRv1 does not change the BGP next-hop, it will not swap the BGP label.

```
RP/0/0/CPU0:XRv1#show mpls ldp bindings 10.6.11.6/32 local
10.6.11.6/32, rev 51
  Local binding: label: 91008
```

When we check XRv2, we see that the remote loopbacks to CSR2 and XRv4 are reachable now. A new issue has arisen as XRv2 only sees CSR5 as an egress point with CSR7 as a next-hop. XRv1 is no longer advertising these routes to XRv2.

```
RP/0/0/CPU0:XRv2#show bgp ipv4 labeled-unicast | begin Network
  Network                Next Hop                Metric LocPrf Weight Path
*>i13.0.0.8/32           13.0.0.8                0      100      0 i
*>i24.0.0.2/32           10.5.7.7                20      100      0 24 i
*>i24.0.0.14/32          10.5.7.7                10      100      0 24 i
```

```
RP/0/0/CPU0:XRv2#show bgp ipv4 labeled-unicast summary | begin ^Neigh
Neighbor      Spk    AS  MsgRcvd  MsgSent    TblVer  InQ  OutQ  Up/Down  St/PfxRcd
13.0.0.5       0     13     211     200         9    0    0 00:32:30      2
13.0.0.8       0     13    34125    33157         9    0    0 00:30:50      1
13.0.0.11      0     13     203     201         9    0    0 00:32:47      0
```

Upon further inspection, we see this is normal BGP behavior. Since CSR6 redistributed these loopbacks from IGP, the origin is set to “incomplete”. CSR7 used the network statement, which sets the origin to “IGP”. On CSR5, the routes from CSR7 are always preferred as a result. AS 24 (probably) unintentionally influenced AS 13’s path selection process using BGP attributes.

```
R5#show bgp ipv4 unicast 24.0.0.2/32
BGP routing table entry for 24.0.0.2/32, version 2
Paths: (2 available, best #2, table default)
  Advertised to update-groups:
    5
  Refresh Epoch 2
  24
    10.5.6.6 from 10.5.6.6 (24.0.0.6)
      Origin incomplete, metric 20, localpref 100, valid, external
      mpls labels in/out nlabel/6004
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  24
    10.5.7.7 from 10.5.7.7 (24.0.0.7)
      Origin IGP, metric 20, localpref 100, valid, external, best
      mpls labels in/out nlabel/7000
      rx pathid: 0, tx pathid: 0x0
```

The RR reflects this path down to XRv1 who comes to a similar conclusion. Since it’s only eBGP peer is CSR6, it only learned the origin “incomplete” routes from AS 13 and now prefers the iBGP path via CSR5. This might be desirable, but in our case, it is not. I would prefer to have more explicit control over the ASBR links.

```
RP/0/0/CPU0:XRv1#show bgp ipv4 labeled-unicast 24.0.0.2/32 | begin 24$
  24
    10.6.11.6 from 10.6.11.6 (24.0.0.6)
```

```

Received Label 6004
Origin incomplete, metric 20, localpref 100, valid, external
Received Path ID 0, Local Path ID 0, version 0
Origin-AS validity: not-found
Path #2: Received by speaker 0
Not advertised to any peer
24
10.5.7.7 (metric 20) from 13.0.0.12 (13.0.0.5)
Received Label 7000
Origin IGP, metric 20, localpref 100, valid, internal, best, group-best
Received Path ID 0, Local Path ID 1, version 14
Originator: 13.0.0.5, Cluster list: 13.0.0.12

```

There are many solutions to this problem, but the simplest one is to configure CSR6 to set the origin to IGP in the route-map when ISIS to BGP redistribution occurs.

```

! CSR6
route-map RM_ISIS_TO_BGP permit 10
set origin igp

```

XRv1 now prefers the eBGP path since the origins tie. Because this is XRv1's best path, it can be advertised to the RR. Now XRv2 sees 2 routes from both CSR5 and XRv1 as intended. In real life, this would probably never happen, since AS 24 would have a common PE loopback leaking strategy on all ASBRs rather than use a combination of redistribution and BGP network statements.

```

RP/0/0/CPU0:XRv1#show bgp ipv4 labeled-unicast 24.0.0.2/32 | begin 24$
24
10.6.11.6 from 10.6.11.6 (24.0.0.6)
Received Label 6004
Origin IGP, metric 20, localpref 100, valid, external, best, group-best
Received Path ID 0, Local Path ID 1, version 16
Origin-AS validity: not-found
Path #2: Received by speaker 0
Not advertised to any peer
24
10.5.7.7 (metric 20) from 13.0.0.12 (13.0.0.5)
Received Label 7000
Origin IGP, metric 20, localpref 100, valid, internal
Received Path ID 0, Local Path ID 0, version 0
Originator: 13.0.0.5, Cluster list: 13.0.0.12

```

```

RP/0/0/CPU0:XRv2#show bgp ipv4 labeled-unicast summary | begin ^Neigh
Neighbor      Spk   AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
13.0.0.5      0    13    263      249        9    0    0 00:40:43      2
13.0.0.8      0    13   34176    33206        9    0    0 00:39:03      1
13.0.0.11     0    13    254      250        9    0    0 00:41:00      2

```

XRv2 selects the routes from CSR5 as best over those from XRv1 as it has a lower BGP RID. This will be true for all prefixes by default, which means there is no path diversity for the inter-AS flows.

```
RP/0/0/CPU0:XRv2#show bgp ipv4 labeled-unicast 24.0.0.2/32 | begin 24,
24, (Received from a RR-client)
  10.5.7.7 (metric 20) from 13.0.0.5 (13.0.0.5)
    Received Label 7000
    Origin IGP, metric 20, localpref 100, valid, internal, best, group-best
    Received Path ID 0, Local Path ID 1, version 8
  Path #2: Received by speaker 0
  Not advertised to any peer
24, (Received from a RR-client)
  10.6.11.6 (metric 20) from 13.0.0.11 (13.0.0.11)
    Received Label 6004
    Origin IGP, metric 20, localpref 100, valid, internal
    Received Path ID 0, Local Path ID 0, version 0
```

Rather than always rely on tie-breakers (which isn't interesting), I will configure XRv2 to prefer the path to CSR2 via XRv1. The path to XRv4 via CSR5 will remain best. I can use weight on XRv2 to accomplish this; as an RR, if the best-path is chosen locally, it is advertised to all other peers. Since XRv2 is also a PE, it will affect the EIGRP VPN traffic between CSR3 and CSR1, and it will also influence CSR8's routing. CSR8 only gets the best routes anyway, so we don't need to adjust any path attributes beyond the local weight value. I continue to use parameterized RPLs since they are very powerful.

```
! XRv2
prefix-set PS_REMOTE_CSR2
  24.0.0.2/32
end-set

route-policy RPL_SET_WEIGHT($PS, $WT)
  if destination in $PS then
    set weight $WT
  else
    pass
  endif
end-policy

router bgp 13
  neighbor 13.0.0.11
  address-family ipv4 labeled-unicast
    route-policy RPL_SET_WEIGHT(PS_REMOTE_CSR2, 2222) in
```

We can check the BGP tables on XRv2 and CSR8 to verify the routing changes. XRv2 selects the route via XRv1 as best due to the increased weight. This is the only route advertised to other iBGP peers (add-path not enabled) so CSR8 has no choice but to also select it as the best path. CSR8 is not aware of the weight modification.


```
RP/0/0/CPU0:XRv2#show bgp ipv4 labeled-unicast | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
*>i13.0.0.8/32      13.0.0.8          0      100      0 i
* i24.0.0.2/32      10.5.7.7          20      100      0 24 i
*>i 10.6.11.6/32    10.6.11.6          20      100     2222 24 i
*>i24.0.0.14/32     10.5.7.7          10      100      0 24 i
* i 10.6.11.6/32    10.6.11.6          10      100      0 24 i
```

```
R8#show bgp ipv4 unicast | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
*> 13.0.0.8/32      0.0.0.0          0          32768 i
*>i 24.0.0.2/32      10.6.11.6          20      100      0 24 i
*>i 24.0.0.14/32     10.5.7.7          10      100      0 24 i
```

Checking the BGP route labels, we can see the local labels allocated by CSR6 and CSR7 are still present with these prefixes. This is important because during recursive lookups of VPN routes, when the VPN next-hop is a BGP labeled-unicast route, these out-labels will be pushed atop the label stack.

```
R8#show bgp ipv4 unicast labels
      Network      Next Hop      In label/Out label
13.0.0.8/32        0.0.0.0        imp-null/nolabel
24.0.0.2/32        10.6.11.6      nolabel/6004
24.0.0.14/32       10.5.7.7       nolabel/7002
```

Now that AS 13 has received all remote PE loopbacks with corresponding labels from AS 24, the opposite must occur. CSR8 and XRv2 are already advertising their loopbacks into BGP, so CSR6 and CSR7 should be learning these prefixes now. CSR6 learns both pairs of prefixes/labels from both AS 13 ASBRs, and CSR7 learns both prefixes/labels from CSR5 only. This is the correct result.

```
R6#show bgp ipv4 unicast labels
      Network      Next Hop      In label/Out label
13.0.0.8/32        10.6.11.11     nolabel/91001
                  10.5.6.5      nolabel/5003
13.0.0.12/32       10.6.11.11     nolabel/91002
                  10.5.6.5      nolabel/5001
24.0.0.2/32        24.6.14.14     6004/nolabel
24.0.0.14/32       24.6.14.14     6003/nolabel
```

```
R7#show bgp ipv4 unicast labels
      Network      Next Hop      In label/Out label
13.0.0.8/32        10.5.7.5        nolabel/5003
13.0.0.12/32       10.5.7.5        nolabel/5001
24.0.0.2/32        24.7.14.14      7000/nolabel
24.0.0.14/32       24.7.14.14      7002/nolabel
```

AS 24 is not going to run BGP labeled-unicast internally. Rather, it will redistribute these eBGP routes into IGP. LDP will be responsible for allocating local labels for them, and because they are host-routes, the existing LDP label allocation filter is valid. We will use two different redistribution strategies. On CSR7, the route-map will redistribute any BGP-labeled prefix into ISIS. This technique assumes the peer AS will only advertise BGP labels for remote PE loopbacks, so it is less secure but very simple/dynamic. The security could be increased using an inbound BGP filter as well, which might be a good compromise approach to achieve both security and simplicity. I set the metric to 10 only because the default is 0; a value of 10 allows other ASBRs to set higher or lower metrics, adding flexibility.

```
! CSR7
route-map RM_BGP_TO_ISIS permit 10
  match mpls-label
  set metric 10

router isis 24
  redistribute bgp 24 route-map RM_BGP_TO_ISIS
```

We can check CSR7's local ISIS LSP to see these prefixes successfully redistributed. We can see that the metric value of 10 has been applied properly.

```
R7#show isis database detail R7.00-00 | include 13\.
Metric: 10          IP 13.0.0.8/32
Metric: 10          IP 13.0.0.12/32
```

We also check the local LDP labels for these prefixes to ensure CSR7 allocated non-null labels. CSR6 and CSR7 will be responsible for connecting the LDP and BGP LSPs as we will see later.

```
R7#show mpls ldp bindings 13.0.0.8 32 local
lib entry: 13.0.0.8/32, rev 62
  local binding: label: 7007
```

```
R7#show mpls ldp bindings 13.0.0.12 32 local
lib entry: 13.0.0.12/32, rev 64
  local binding: label: 7015
```

Before we perform the redistribution on CSR6, I will prefer XRv1 as the next-hop router for all traffic leaving AS 24. Constantly relying on tie-breakers (eBGP oldest route in this case) is boring and nondeterministic. Since CSR6 has no other labeled-unicast peers, I can use the weight attribute again. To be a minimalist, I will configure it for the entire peer, not a per-prefix basis.

```
! CSR6
address-family ipv4
  neighbor 10.6.11.11 weight 6666

R6#show bgp ipv4 unicast | begin Network
```

	Network	Next Hop	Metric	LocPrf	Weight	Path
*>	13.0.0.8/32	10.6.11.11			6666	13 i
*		10.5.6.5			0	13 i
*>	13.0.0.12/32	10.6.11.11			6666	13 i
*		10.5.6.5			0	13 i
*>	24.0.0.2/32	24.6.14.14	20		32768	i
*>	24.0.0.14/32	24.6.14.14	10		32768	i

On CSR6, I will take a more secure approach by matching only labeled prefixes from specific prefix ranges. In doing this, I can achieve some traffic engineering for traffic leaving AS 24. Traffic leaving AS 24 towards CSR8 will prefer CSR6 as an egress point while traffic towards XRv2 will prefer CSR7. These preferences are achieved using IGP metric values. I use fancy ACLs to match the 3rd least significant bit in the network address. For 13.0.0.8, this bit is cleared (8 = 1000) and for 13.0.0.12, this bit is set (12 = 1100). This odd-ball filtering has little real-life utility but is a valid way to match prefixes in XE.

```
! CSR6
ip access-list standard ACL_REMOTE_LOOPBACKS_0
 permit 13.0.0.0 0.0.0.251
ip access-list standard ACL_REMOTE_LOOPBACKS_1
 permit 13.0.0.4 0.0.0.251

route-map RM_BGP_TO_ISIS permit 10
 match ip address ACL_REMOTE_LOOPBACKS_0
 match mpls-label
 set metric 5
route-map RM_BGP_TO_ISIS permit 20
 match ip address ACL_REMOTE_LOOPBACKS_1
 match mpls-label
 set metric 15

router isis 24
 redistribute bgp 24 route-map RM_BGP_TO_ISIS
```

Like CSR7, we check to ensure the routes were redistributed into ISIS and have the correct metrics. We also check the LDP LIB to ensure non-null local labels were allocated for both prefixes.

```
R6#show isis database detail R6.00-00 | include 13\.
Metric: 5          IP 13.0.0.8/32
Metric: 15         IP 13.0.0.12/32

R6#show mpls ldp bindings 13.0.0.8 32 local
lib entry: 13.0.0.8/32, rev 54
local binding: label: 6075

R6#show mpls ldp bindings 13.0.0.12 32 local
lib entry: 13.0.0.12/32, rev 55
local binding: label: 6066
```

A good test of our traffic engineering policy is to check the FIB on other AS 24 routers. CSR2 always traverses XRv4 since the IGP cost towards CSR7 is high. XRv4 allocates local labels for these prefixes as expected.

```
R2#show ip cef 13.0.0.8
13.0.0.8/32
  nexthop 24.2.14.14 GigabitEthernet2.524 label 94005
```

```
R2#show ip cef 13.0.0.12
13.0.0.12/32
  nexthop 24.2.14.14 GigabitEthernet2.524 label 94003
```

On XRv4, we see that traffic to CSR8 routes via CSR6 while traffic to XRv2 routes via CSR7. The outbound labels used by XRv4 are the matching local labels allocated by CSR6 and CSR7 we verified earlier.

```
RP/0/0/CPU0:XRv4#show cef 13.0.0.8/32
13.0.0.8/32, version 821, internal 0x1000001 0x0 (ptr 0xa142e1f4) [1], 0x0
(0xa13f9758), 0xa28 (0xa156d1b8)
local adjacency 24.6.14.6
  Prefix Len 32, traffic index 0, precedence n/a, priority 3
  via 24.6.14.6, GigabitEthernet0/0/0/0.564, 5 dependencies, weight 0, class
0 [flags 0x0]
  path-idx 0 NHID 0x0 [0xa1085154 0x0]
  next hop 24.6.14.6
  local adjacency
  local label 94005 labels imposed {6075}
```

```
RP/0/0/CPU0:XRv4#show cef 13.0.0.12/32
13.0.0.12/32, version 818, internal 0x1000001 0x0 (ptr 0xa142ef74) [1], 0x0
(0xa13f9908), 0xa28 (0xa156d2a8)
local adjacency 24.7.14.7
  Prefix Len 32, traffic index 0, precedence n/a, priority 3
  via 24.7.14.7, GigabitEthernet0/0/0/0.574, 5 dependencies, weight 0, class
0 [flags 0x0]
  path-idx 0 NHID 0x0 [0xa10852f8 0x0]
  next hop 24.7.14.7
  local adjacency
  local label 94003 labels imposed {7015}
```

Now that all PEs have learned about all other PEs between ASes, we should have PE-to-PE reachability. Since we intend on transporting MPLS services over these links, we must also ensure that the path is always MPLS-encapsulated. We will manually trace the LSP from CSR2 to CSR8, and then in the reverse direction. Since the ASes use different remote loopback leaking strategies, the route recursion (and resulting label stacks) will differ. We will begin at CSR2. CSR2 has an IGP route to 13.0.0.8/32 via XRv4

along with a corresponding LDP label. The label stack becomes 94005; the lack of BGP labeled-unicast internal to AS 24 means that there are no recursive BGP lookups in the global table.

```
R2#show ip route 13.0.0.8
Routing entry for 13.0.0.8/32
  Known via "isis", distance 115, metric 25, type level-2
  Redistributing via isis 24
  Last update from 24.2.14.14 on GigabitEthernet2.524, 00:18:46 ago
  Routing Descriptor Blocks:
  * 24.2.14.14, from 24.0.0.6, 00:18:46 ago, via GigabitEthernet2.524
    Route metric is 25, traffic share count is 1
```

```
R2#show mpls ldp bindings 13.0.0.8 32 neighbor 24.0.0.14
  lib entry: 13.0.0.8/32, rev 57
  remote binding: lsr: 24.0.0.14:0, label: 94005
```

XRv4 is a P router along this LSP and performs a basic label swap between two LDP labels. If XRv4 wasn't a PE at all and only a P router for all LSPs, this would nicely illustrate the drawback of not using BGP within the AS. Now, all the P routers must learn all of the remote PE loopbacks as IGP routes and bind LDP labels for them, as XRv4 has done. Nonetheless, this is a valid technique and does work.

```
RP/0/0/CPU0:XRv4#show mpls forwarding labels 94005
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94005	6075	13.0.0.8/32	Gi0/0/0/0.564	24.6.14.6	3498

```
RP/0/0/CPU0:XRv4#show route 13.0.0.8/32
Routing entry for 13.0.0.8/32
  Known via "isis 24", distance 115, metric 15, type level-2
  Routing Descriptor Blocks
    24.6.14.6, from 24.0.0.6, via GigabitEthernet0/0/0/0.564
    Route metric is 15
  No advertising protos.
```

```
RP/0/0/CPU0:XRv4#show mpls ldp bindings 13.0.0.8/32 neighbor 24.0.0.6
13.0.0.8/32, rev 63
  Local binding: label: 94005
  Remote bindings: (3 peers)
    Peer          Label
    -----
    24.0.0.6:0    6075
```

CSR6 receives packets with label 6075 and performs a label swap, but not to another LDP label. Since the route to 13.0.0.8/32 is a BGP route, the BGP label of 91001 must be used. This was allocated by XRv1. The LFIB is smart enough to connect the LDP and BGP LSPs seamlessly.

```

R6#show mpls forwarding-table labels 6075
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id Switched      interface
6075       91001      13.0.0.8/32 3846         Gi2.561     10.6.11.11

R6#show ip route 13.0.0.8
Routing entry for 13.0.0.8/32
  Known via "bgp 24", distance 20, metric 0
  Tag 13, type external
  Redistributing via isis 24
  Advertised by isis 24 metric-type internal level-2 route-map RM_BGP_TO_ISIS
  Last update from 10.6.11.11 00:30:35 ago
  Routing Descriptor Blocks:
  * 10.6.11.11, from 10.6.11.11, 00:30:35 ago
    Route metric is 0, traffic share count is 1
    AS Hops 1
    Route tag 13
    MPLS label: 91001

```

If we looked only at the BGP route, we might be concerned since it appears that no local label has been assigned. From the perspective of BGP, this is true, but BGP does not represent the only mechanism by which LSPs can be built. Always check the LFIB for the correct forwarding information.

```

R6#show bgp ipv4 unicast 13.0.0.8/32 bestpath
BGP routing table entry for 13.0.0.8/32, version 13
Paths: (2 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  13
    10.6.11.11 from 10.6.11.11 (13.0.0.11)
      Origin IGP, localpref 100, weight 6666, valid, external, best
      mpls labels in/out nolabel/91001
      rx pathid: 0, tx pathid: 0x0

```

When XRv1 receives packets with label 91001, it removes the topmost label and forwards the packet towards CSR8. The route is learned via IGP, so the LDP label of implicit-null is used.

```

RP/0/0/CPU0:XRv1#show mpls forwarding labels 91001
Local  Outgoing  Prefix      Outgoing  Next Hop      Bytes
Label  Label      or ID       Interface  Next Hop      Switched
-----
91001  Pop         13.0.0.8/32 Gi0/0/0/0.581 13.8.11.8     148542

RP/0/0/CPU0:XRv1#show route 13.0.0.8/32
Routing entry for 13.0.0.8/32
  Known via "ospf 13", distance 110, metric 2, type intra area
  Routing Descriptor Blocks
    13.8.11.8, from 13.0.0.8, via GigabitEthernet0/0/0/0.581

```

Route metric is 2
No advertising protos.

```
RP/0/0/CPU0:XRv1#show mpls ldp bindings 13.0.0.8/32 neighbor 13.0.0.8
13.0.0.8/32, rev 16
  Local binding: label: 91001
  Remote bindings: (3 peers)
    Peer          Label
    -----
    13.0.0.8:0    ImpNull
```

Be careful not to immediately assume BGP is performing this label swap. Looking at the BGP route, we can see a received label of 3 (implicit-null). This is because 13.0.0.8/32 is directly connected to CSR8, but this is not the implicit-null that is used. The route would have to be BGP-learned in order for this label to be consulted. Although harmless, this is an important detail; always be cognizant of the route source as it determines which label must be used.

```
RP/0/0/CPU0:XRv1#show bgp ipv4 labeled-unicast 13.0.0.8/32 | begin Local$
Local
  13.0.0.8 (metric 2) from 13.0.0.12 (13.0.0.8)
    Received Label 3
    Origin IGP, metric 0, localpref 100, valid, internal, best, group-best
    Received Path ID 0, Local Path ID 1, version 21
    Originator: 13.0.0.8, Cluster list: 13.0.0.12
```

If there were more MPLS labels in the stack (such as VPNv4, etc) then this VPN label would be exposed to CSR8 correctly. In this case, the IP packet would be exposed to CSR8 which is correct for global traffic, like the eBGP session to be configured soon. Using traceroute from CSR2, we can verify the label stack.

```
R2#traceroute 13.0.0.8 source 24.0.0.2
Type escape sequence to abort.
Tracing the route to 13.0.0.8
VRF info: (vrf in name/id, vrf out name/id)
 1 24.2.14.14 [MPLS: Label 94005 Exp 0] 6 msec 6 msec 6 msec
 2 24.6.14.6 [MPLS: Label 6075 Exp 0] 27 msec 30 msec 30 msec
 3 10.6.11.11 [MPLS: Label 91001 Exp 0] 22 msec 18 msec 21 msec
 4 13.8.11.8 25 msec 12 msec 11 msec
```

Next, we will trace the LSP from CSR8 to CSR2. CSR8 has a BGP route to 24.0.0.2/32, which is different than CSR2's IGP route to 13.0.0.8/32. This means CSR8 must push a BGP label onto the stack, and in this case, label 6004 is used. This means that CSR6 is the BGP next-hop as it allocated a local label for this prefix. This makes sense because XRv1 did not set the next-hop to itself, nor was it configured to allocate local labels for remote loopbacks.

```
R8#show ip route 24.0.0.2
Routing entry for 24.0.0.2/32
```

```
Known via "bgp 13", distance 200, metric 20
Tag 24, type internal
Last update from 10.6.11.6 00:56:58 ago
Routing Descriptor Blocks:
* 10.6.11.6, from 13.0.0.12, 00:56:58 ago
  Route metric is 20, traffic share count is 1
  AS Hops 1
  Route tag 24
  MPLS label: 6004
```

CSR8 now needs to lookup the route to the BGP next-hop, which is 10.6.11.6. XRv1 has a static route for this prefix which was redistributed into OSPF, so CSR8 will have an LDP label from XRv1 to describe the path to this prefix. The LDP label is 91008, making the label stack {91008 6004}.

```
R8#show ip route 10.6.11.6
Routing entry for 10.6.11.6/32
  Known via "ospf 13", distance 110, metric 20
  Tag 13, type extern 2, forward metric 1
  Last update from 13.8.11.11 on GigabitEthernet2.581, 01:30:51 ago
  Routing Descriptor Blocks:
  * 13.8.11.11, from 13.0.0.11, 01:30:51 ago, via GigabitEthernet2.581
    Route metric is 20, traffic share count is 1
    Route tag 13
```

```
R8#show mpls ldp bindings 10.6.11.6 32 neighbor 13.0.0.11
lib entry: 10.6.11.6/32, rev 21
remote binding: lsr: 13.0.0.11:0, label: 91008
```

When XRv1 receives this packet, it removes label 91008 and sends the packet towards CSR6. BGP is not involved in this operation at all. Since the route was locally configured/originated, XRv1 removes the label without having received an implicit-null. The label stack becomes 6004, which exposes the BGP label to CSR6 appropriately.

```
RP/0/0/CPU0:XRv1#show mpls forwarding labels 91008
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
91008	Pop	10.6.11.6/32	Gi0/0/0/0.561	10.6.11.6	18738

```
RP/0/0/CPU0:XRv1#show route 10.6.11.6/32
Routing entry for 10.6.11.6/32
  Known via "static", distance 1, metric 0 (connected)
  Tag 13
  Routing Descriptor Blocks
    directly connected, via GigabitEthernet0/0/0/0.561
    Route metric is 0
  No advertising protos.
```


CSR6 swaps label 6004 for 94009 and forwards the packet to XRv4. The route is learned via IGP which means an outgoing LDP label is used. This is how CSR6 connects the BGP and LDP LSPs.

```
R6#show mpls forwarding-table labels 6004
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Outgoing interface	Next Hop
6004	94009	24.0.0.2/32	8794	Gi2.564	24.6.14.14

```
R6#show ip route 24.0.0.2
```

```
Routing entry for 24.0.0.2/32
```

```
Known via "isis", distance 115, metric 20
```

```
Tag 24, type level-2
```

```
Redistributing via isis 24, bgp 24
```

```
Advertised by bgp 24 level-2 route-map RM_ISIS_TO_BGP
```

```
Last update from 24.6.14.14 on GigabitEthernet2.564, 02:25:35 ago
```

```
Routing Descriptor Blocks:
```

```
* 24.6.14.14, from 24.0.0.2, 02:25:35 ago, via GigabitEthernet2.564
```

```
Route metric is 20, traffic share count is 1
```

```
Route tag 24
```

```
R6#show mpls ldp bindings 24.0.0.2 32 neighbor 24.0.0.14
```

```
lib entry: 24.0.0.2/32, rev 43
```

```
remote binding: lsr: 24.0.0.14:0, label: 94009
```

As always, be careful not to only check BGP. The output makes it appear like there is no outgoing label, which is true from a BGP standpoint. The LFIB connects the LSPs together even if BGP is unaware of it.

```
R6#show bgp ipv4 unicast 24.0.0.2/32
```

```
BGP routing table entry for 24.0.0.2/32, version 7
```

```
Paths: (1 available, best #1, table default)
```

```
Advertised to update-groups:
```

```
2
```

```
Refresh Epoch 1
```

```
Local
```

```
24.6.14.14 from 0.0.0.0 (24.0.0.6)
```

```
Origin IGP, metric 20, localpref 100, weight 32768, valid, sourced,  
best
```

```
mpls labels in/out 6004/nolabel
```

```
rx pathid: 0, tx pathid: 0x0
```

XRv4 pops labels 94009, which reveals either the IP packet or MPLS service label to CSR2. This is the correct behavior.

```
RP/0/0/CPU0:XRv4#show mpls forwarding labels 94009
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched

```
-----  
94009 Pop 24.0.0.2/32 Gi0/0/0/0.524 24.2.14.2 17343839
```

We quickly verify this with a traceroute. This LSP tracing is critical because it does not make sense to continue option C configurations until end-to-end LSPs are established between PEs. Otherwise, none of the MPLS services will operate properly.

```
R8#traceroute 24.0.0.2 source 13.0.0.8  
Type escape sequence to abort.  
Tracing the route to 24.0.0.2  
VRF info: (vrf in name/id, vrf out name/id)  
 1 13.8.11.11 [MPLS: Labels 91008/6004 Exp 0] 7 msec 6 msec 6 msec  
 2 10.6.11.6 [MPLS: Label 6004 Exp 0] 29 msec 31 msec 30 msec  
 3 24.6.14.14 [MPLS: Label 94009 Exp 0] 20 msec 20 msec 20 msec  
 4 24.2.14.2 20 msec 11 msec 11 msec
```

One of my personal favorite characteristics of option C is the simplified BGP configuration on the ASBRs. We do not need to extend every MPLS service to them (option B) nor do we have to configure a per-VPN eBGP session across AS boundaries (option A). The ASBR configuration is complete and now we can focus on MPLS service delivery. The next step is to configure the service-specific BGP sessions between the RRs in different ASes. Technically, although RR loopbacks must be leaked across AS boundaries to achieve IP reachability, there is no requirement for those prefixes to be labeled. In our setup, the RRs are also PEs, so we do not need to implement any additional filtering. The benefit of having the RR loopbacks be labeled is it makes the configuration more consistent and allows the RR-to-RR BGP session to be protected by TE-FRR. Below is the basic configuration of VPNv4/v6 between XRv2 and CSR2. This must be a multi-hop eBGP session as the routers are in different ASes and are therefore many hops apart. This is also the first time XRv2 has an eBGP peer, so we will define a basic pass-any RPL for now. We verify the connection is functional on XRv2 for both AFIs before continuing.

```
! CSR2  
router bgp 24  
 neighbor 13.0.0.12 remote-as 13  
 neighbor 13.0.0.12 ebgp-multihop 8  
 neighbor 13.0.0.12 update-source Loopback0  
 address-family vpnv4  
   neighbor 13.0.0.12 activate  
 address-family vpnv6  
   neighbor 13.0.0.12 activate  
  
! XRv2  
route-policy RPL_PASS  
  pass  
end-policy  
  
router bgp 13  
 neighbor 24.0.0.2
```

```

remote-as 24
update-source Loopback0
ebgp-multihop 8
address-family vpnv4 unicast
  route-policy RPL_PASS in
  route-policy RPL_PASS out
address-family vpnv6 unicast
  route-policy RPL_PASS in
  route-policy RPL_PASS out

```

Since the RR's already have all of the intra-AS VPN routes, they immediately advertise their best-paths to their eBGP neighbors. By default, they set the next-hop to their update-sources which is expected for eBGP. We will see how this can become problematic later.

```

RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast summary | begin ^Neigh
Neighbor      Spk    AS  MsgRcvd  MsgSent  TblVer  InQ  OutQ  Up/Down  St/PfxRcd
13.0.0.8      0     13   34826    33829    1708    0    0  01:20:37      6
24.0.0.2      0     24     46     22    1708    0    0  00:01:57      7

```

```

RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast summary | begin ^Neigh
Neighbor      Spk    AS  MsgRcvd  MsgSent  TblVer  InQ  OutQ  Up/Down  St/PfxRcd
13.0.0.8      0     13   34826    33830    1783    0    0  01:20:41      6
24.0.0.2      0     24     46     22    1783    0    0  00:02:00      7

```

Looking at the VRF OSPF routes on both RRs, we can see the BGP next-hop modifications.

```

RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast rd 24:2 | begin Network
Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 24:2
*> 10.2.9.0/24  24.0.0.2      0          0 24 ?
*> 10.9.9.9/32  24.0.0.2      1          0 24 ?

```

```

R2#show bgp vpnv4 unicast rd 13:2 | begin Network
Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 13:2
*> 10.4.4.4/32  13.0.0.12     0          0 13 ?
*> 10.4.8.0/24  13.0.0.12     0          0 13 ?

```

Now that the RRs within each AS have routes from one another, we must adjust the VRF route target import/export policies. One of the main drawbacks of options B and C is that they generally require the providers to agree on these policies since RTs are directly exchanged. We could easily accomplish this by simply importing the RTs exported by the peer AS, but we tested that with option B already; it is known to work. There is an alternative solution known as “route target rewriting” which is valid for both options B and C. The logic is simple; match some RT value being advertised to a peer (outbound filter in BGP), remove that RT, and then add a new one that is already being imported by the peer. This will allow an AS to keep the RT's inside the AS and adjust their outbound values to be set to the imported RTs in the remote AS. Before we configure this feature, we will ensure all of the VRFs are

importing/exporting their local RTs only. Under no circumstances should a local VRF be importing any remote RTs. This restriction will make inter-AS central services very interesting as well.

```
! CSR8
vrf definition BGP
  address-family ipv4
    route-target export 13:1
    route-target import 13:3
    route-target import 13:2
  address-family ipv6
    route-target export 13:1
    route-target import 13:3
    route-target import 13:2

vrf definition OSPF
  address-family ipv4
    route-target export 13:2
    route-target import 13:1
    route-target import 13:2
  address-family ipv6
    route-target export 13:2
    route-target import 13:1
    route-target import 13:2

! XRv2
vrf EIGRP
  address-family ipv4 unicast
    import route-target
      13:1
      13:3
    export route-target
      13:3

  address-family ipv6 unicast
    import route-target
      13:1
      13:3
    export route-target
      13:3

! CSR2
vrf definition EIGRP
  address-family ipv4
    route-target export 24:3
    route-target import 24:3
  address-family ipv6
    route-target export 24:3
    route-target import 24:3
```

```

vrf definition OSPF
  address-family ipv4
    route-target export 24:2
    route-target import 24:2
  address-family ipv6
    route-target export 24:2
    route-target import 24:2

! XRv4
vrf EIGRP
  address-family ipv4 unicast
    import route-target
      24:3
    export route-target
      24:3

  address-family ipv6 unicast
    import route-target
      24:3
    export route-target
      24:3

```

You may notice that some of these RT policies don't even make sense. For example, there are no other EIGRP VPN PEs in AS 13, so there is no reason to import RT:13:3. The same is true for CSR2 importing RT:24:2 inside the OSPF VPN. However, if we assume that those other PEs did exist for each VPN within each AS, these RT policies might be logical. We will examine 10.9.9.9/32 inside VRF OSPF on XRv2 as an example. CSR2 sets the RT to be 24:2 which, according to the new policy, is not imported anywhere in AS 13. As such, OSPF will not import the route into VRF OSPF; in fact, the route isn't even retained in the RD table as there is no reason to keep it; if no VRF imports the RT locally, the VPN route is rejected.

```

RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast rd 24:2 10.9.9.9/32 | begin 24$
24
  24.0.0.2 (metric 20) from 24.0.0.2 (24.0.0.2)
    Received Label 2009
    Origin incomplete, metric 1, localpref 100, valid, external, best,
group-best, import-candidate, not-in-vrf
    Received Path ID 0, Local Path ID 1, version 1696
    Extended community: OSPF router-id:10.2.9.2 OSPF route-type:0:2:0x0
RT:24:2

```

```

R8#show bgp vpnv4 unicast vrf OSPF 10.9.9.9/32
% Network not in table

```

```

R8#show bgp vpnv4 unicast rd 24:2 10.9.9.9/32
% Network not in table

```

We will focus on CSR2 first. We must build extended community lists to match the RTs in question. The standard list gives some basic community options like RT, SoO, etc. We can also use a regex-based expanded list to make the RT look like whatever we want. I use a regex with some bogus RTs just to demonstrate; the second list would match 24:3, 24:8, or 24:9. Also notice that the standard list builds the same string as the extended one, but is easier to configure. The expanded list can match any extended community provided the administrator knows the exact textual string.

```
! CSR2
ip extcommunity-list standard EXTCOML_RT_24_2 permit rt 24:2
ip extcommunity-list expanded EXTCOML_RT_24_3 permit RT:24:[389]

R2#show ip extcommunity-list
Standard extended community-list EXTCOML_RT_24_2
  10 permit RT:24:2
Expanded extended community-list EXTCOML_RT_24_3
  10 permit RT:24:[389]
```

Next, we will wrap these in a route-map. Each permit clause will match a different list defined above. When a match occurs, we must remove the matching communities using the same list defined above, then append the new RT. The “additive” keyword is important otherwise unrelated extended communities, such as the custom EIGRP/OSPF ones, will be overwritten. We apply the route-map outbound to the eBGP VPNv4/v6 peers. Since there are no IPv4/v6 specific matches in these clauses, the filter is generic for all AFIs that use the RT extended community.

```
! CSR2
route-map RM_RT_REWRITE permit 10
  match extcommunity EXTCOML_RT_24_2
  set extcomm-list EXTCOML_RT_24_2 delete
  set extcommunity rt 13:2 additive

route-map RM_RT_REWRITE permit 20
  match extcommunity EXTCOML_RT_24_3
  set extcomm-list EXTCOML_RT_24_3 delete
  set extcommunity rt 13:3 additive

router bgp 24
  address-family vpnv4
    neighbor 13.0.0.12 route-map RM_RT_REWRITE out
  address-family vpnv6
    neighbor 13.0.0.12 route-map RM_RT_REWRITE out
```

After a soft-out on both AFIs (not shown), XRv2 will learn the VPN routes with updated RT values. On XRv2, we check the CSR9 and CSR1 loopbacks to confirm the proper RTs are carried. We also see all other extended communities still intact as a result of the “additive” keyword on CSR2. If we failed to delete the old RTs, that would have technically also worked as the route would have carried the RT for

both ASes. That would be more of a “RT augment” versus a “RT rewrite” design and I would consider that sloppy unless there was a compelling reason for it.

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast rd 24:2 10.9.9.9/32 | begin 24$
24
  24.0.0.2 (metric 20) from 24.0.0.2 (24.0.0.2)
    Received Label 2009
    Origin incomplete, metric 1, localpref 100, valid, external, best,
group-best, import-candidate, not-in-vrf
    Received Path ID 0, Local Path ID 1, version 1730
    Extended community: OSPF router-id:10.2.9.2 OSPF route-type:0:2:0x0
RT:13:2
```

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast rd 24:3 10.1.1.1/32 | begin 24$
24
  24.0.0.2 (metric 20) from 24.0.0.2 (24.0.0.2)
    Received Label 2012
    Origin incomplete, metric 10880, localpref 100, valid, external, best,
group-best, import-candidate, not-in-vrf
    Received Path ID 0, Local Path ID 1, version 1738
    Extended community: EIGRP route-info:0x8000:0 EIGRP AD:3:288 EIGRP
RHB:255:1:2560 EIGRP LM:0xff:1:1500 EIGRP VRR:0x0:1.1.1.10 RT:13:3
```

We can take it a step further by checking for these VPN routes inside of the VRFs. XRv2 sees CSR1 inside VRF EIGRP and CSR8 sees CSR9 inside VRF OSPF. This is a good indication that the configuration worked.

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf EIGRP 10.1.1.1/32 | begin 24$
24
  24.0.0.2 (metric 20) from 24.0.0.2 (24.0.0.2)
    Received Label 2012
    Origin incomplete, metric 10880, localpref 100, valid, external, best,
group-best, import-candidate, imported
    Received Path ID 0, Local Path ID 1, version 1740
    Extended community: EIGRP route-info:0x8000:0 EIGRP AD:3:288 EIGRP
RHB:255:1:2560 EIGRP LM:0xff:1:1500 EIGRP VRR:0x0:1.1.1.10 RT:13:3
    Source VRF: default, Source Route Distinguisher: 24:3
```

```
R8#show bgp vpnv4 unicast vrf OSPF 10.9.9.9/32
BGP routing table entry for 13:2:10.9.9.9/32, version 354
Paths: (1 available, best #1, table OSPF)
  Not advertised to any peer
  Refresh Epoch 1
  24, imported path from 24:2:10.9.9.9/32 (global)
    24.0.0.2 (metric 20) (via default) from 13.0.0.12 (13.0.0.12)
      Origin incomplete, metric 1, localpref 100, valid, internal, best
      Extended Community: RT:13:2 OSPF ROUTER ID:10.2.9.2:0
      OSPF RT:0.0.0.0:2:0
      mpls labels in/out nolabel/2009
```

```
rx pathid: 0, tx pathid: 0x0
```

Next, we need to do the same thing in the opposite direction by focusing on XRv2. AS 13 was able to import VPN routes from AS 24 since CSR2 adjusted the RTs to comply with AS 13 policies. The configuration on XR is very similar to XE. I use in-line sets with RPL for brevity on XRv2; this is less flexible than parameterization but still works.

```
! XRv2
route-policy RPL_RT_REWRITE
  if extcommunity rt matches-every (13:2) then
    delete extcommunity rt in (13:2)
    set extcommunity rt (24:2) additive
  elseif extcommunity rt matches-every (13:3) then
    delete extcommunity rt in (13:3)
    set extcommunity rt (24:3) additive
  endif
end-policy

router bgp 13
 neighbor 24.0.0.2
  address-family vpnv4 unicast
    route-policy RPL_RT_REWRITE out
  address-family vpnv6 unicast
    route-policy RPL_RT_REWRITE out
```

We quickly check CSR2 for the presence of remote AS VPN routes in both the OSPF and EIGRP VPNs.

```
R2#show bgp vpnv4 unicast vrf OSPF 10.4.4.4/32
BGP routing table entry for 24:2:10.4.4.4/32, version 7555
Paths: (1 available, best #1, table OSPF)
Flag: 0x820
  Not advertised to any peer
  Refresh Epoch 1
  13, imported path from 13:2:10.4.4.4/32 (global)
    13.0.0.12 (metric 30) (via default) from 13.0.0.12 (13.0.0.12)
      Origin incomplete, localpref 100, valid, external, best
      Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
        OSPF RT:0.0.0.0:2:0
      mpls labels in/out nolabel/92014
      rx pathid: 0, tx pathid: 0x0
```

```
R2#show bgp vpnv4 unicast vrf EIGRP 10.3.3.3/32
BGP routing table entry for 24:3:10.3.3.3/32, version 7557
Paths: (1 available, best #1, table EIGRP)
  Not advertised to any peer
  Refresh Epoch 1
  13, imported path from 13:3:10.3.3.3/32 (global)
    13.0.0.12 (metric 30) (via default) from 13.0.0.12 (13.0.0.12)
```



```

Origin incomplete, metric 10880, localpref 100, valid, external, best
Extended Community: RT:24:3 0x8800:32768:0 0x8801:3:288
0x8802:65281:2560 0x8803:1:1500 0x8806:0:167971843
Connector Attribute: count=1
type 1 len 12 value 13:3:13.0.0.12
mpls labels in/out nlabel/92002
rx pathid: 0, tx pathid: 0x0

```

For the OSPF and EIGRP VPNs, the L3VPN control plane should be fully functional. However, this solution has not solved the central services problem for AS 24. CSR2 and XRv4 have no idea that these Internet routes are even available due to the RT policies.

```

R2#show bgp vpnv4 unicast vrf EIGRP 110.0.0.0/32
% Network not in table

```

```

R2#show bgp vpnv4 unicast vrf OSPF 110.0.0.0/32
% Network not in table

```

```

RP/0/0/CPU0:XRv4#show bgp vpnv4 unicast vrf EIGRP 110.0.0.0/32
% Network not in table

```

Just like with everything in networking, there are many solutions to this problem. Offhand, here are four potential solutions:

1. Export RTs 13:2 and 13:3 from CSR8's VRF BGP in addition to RT:13:1. These new RTs would get rewritten automatically by XRv2 to 24:2 and 24:3. This would require a minor RPL change on XRv2 as the "elseif" would need to be replaced with a regular "if" so both conditionals are evaluated independently from one another. This would allow both RTs to be rewritten.
2. Adjust the RPL on XRv2 so that all non-matched routes are still advertised, then import RT 13:1 on the AS 24 routers. This would violate the "policy" but is a valid solution.
3. Adjust the RPL on XRv2 so that all non-matched routes are still advertised, then export RTs 24:2 and 24:3 from CSR8's VRF BGP. Like #2, this also violates the RT "policy".
4. Adjust the RPL on XRv2 with a third match clause for RT 13:1. This RT would be removed and both RTs 24:2 and 24:3 would be added.

We will implement the fourth option. This effectively replaces one RT with a set of RTs, which is a valid operation. Of the four options, I find this to be the most effective, straightforward, and compliant with the "policy".

```

! XRv2
route-policy RPL_RT_REWRITE
  if extcommunity rt matches-every (13:2) then
    delete extcommunity rt in (13:2)
    set extcommunity rt (24:2) additive
  elseif extcommunity rt matches-every (13:3) then
    delete extcommunity rt in (13:3)

```

```

    set extcommunity rt (24:3) additive
elseif extcommunity rt matches-every (13:1) then
    delete extcommunity rt in (13:1)
    set extcommunity rt (24:2, 24:3) additive
endif
end-policy

```

After applying this policy, we check CSR2 for the central services routes inside both OSPF and EIGRP VPNs. The route has been imported to both and we can see both RTs 24:2 and 24:3 attached to the VPN route. We do not need to adjust RTs in the right-to-left direction since CSR2 is already rewriting the RTs exported by the AS 24 VRFs.

```

R2#show bgp vpnv4 unicast vrf EIGRP 110.0.0.0/32
BGP routing table entry for 24:3:110.0.0.0/32, version 7593
Paths: (1 available, best #1, table EIGRP)
Flag: 0x820
    Not advertised to any peer
    Refresh Epoch 1
    13 100, imported path from 13:1:110.0.0.0/32 (global)
        13.0.0.12 (metric 30) (via default) from 13.0.0.12 (13.0.0.12)
            Origin incomplete, localpref 100, valid, external, best
            Extended Community: RT:24:2 RT:24:3
            mpls labels in/out nolabel/92009
            rx pathid: 0, tx pathid: 0x0

```

```

R2#show bgp vpnv4 unicast vrf OSPF 110.0.0.0/32
BGP routing table entry for 24:2:110.0.0.0/32, version 7592
Paths: (1 available, best #1, table OSPF)
Flag: 0x820
    Not advertised to any peer
    Refresh Epoch 1
    13 100, imported path from 13:1:110.0.0.0/32 (global)
        13.0.0.12 (metric 30) (via default) from 13.0.0.12 (13.0.0.12)
            Origin incomplete, localpref 100, valid, external, best
            Extended Community: RT:24:2 RT:24:3
            mpls labels in/out nolabel/92009
            rx pathid: 0, tx pathid: 0x0

```

Now that the route exchange is complete, we can begin tracing LSPs. Since the RRs have adjusted the BGP next-hops, this means they will be performing MPLS service label swaps. It also puts them in the data plane which is highly undesirable. In the best case, this introduces a severe inefficiency, and in the worst case, it completely breaks connectivity. We will first examine the inefficiency case by examining the OSPF VPN from CSR9 to CSR4. The backdoor link is down as is the sham-link. We will manually trace the IPv6 LSP to see why this is a problem. Traffic entering CSR2 matches a VPNv6 route from XRv2 with label 92021. The route to 13.0.0.12, the VPN next-hop, is IGP-learned via XRv4. This means XRv4's LDP

label of 94003 is pushed also. The label stack becomes {92021 94003} and we confirm this by checking the IPv6 FIB.

```
R2#show bgp vpnv6 unicast vrf OSPF ::10:4:4:4/128
BGP routing table entry for [24:2>::10:4:4:4/128, version 7134
Paths: (1 available, best #1, table OSPF)
  Not advertised to any peer
  Refresh Epoch 1
  13, imported path from [13:2>::10:4:4:4/128 (global)
    ::FFFF:13.0.0.12 (metric 30) (via default) from 13.0.0.12 (13.0.0.12)
    Origin incomplete, localpref 100, valid, external, best
    Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
    OSPF RT:0.0.0.0:2:0
    mpls labels in/out nolaabel/92021
    rx pathid: 0, tx pathid: 0x0
```

```
R2#show ip route 13.0.0.12
Routing entry for 13.0.0.12/32
  Known via "isis", distance 115, metric 30, type level-2
  Redistributing via isis 24
  Last update from 24.2.14.14 on GigabitEthernet2.524, 02:42:01 ago
  Routing Descriptor Blocks:
    * 24.2.14.14, from 24.0.0.7, 02:42:01 ago, via GigabitEthernet2.524
      Route metric is 30, traffic share count is 1
```

```
R2#show mpls ldp bindings 13.0.0.12 32 neighbor 24.0.0.14
lib entry: 13.0.0.12/32, rev 58
remote binding: lsr: 24.0.0.14:0, label: 94003
```

```
R2#show ipv6 cef vrf OSPF ::10:4:4:4/128
::10:4:4:4/128
  nexthop 24.2.14.14 GigabitEthernet2.524 label 94003 92021
```

XRv4 is a normal P router and performs a label swap between two LDP labels. The packet is forwarded to CSR7. Notice that when we traced the LSP from CSR2 to CSR8 earlier, CSR6 was the egress ASBR, not CSR7. This is because CSR2 is trying to send traffic towards XRv2 this time, so CSR7 is the correct egress ASBR per our IGP metric adjustments. The label stack becomes {7015 92021}.

```
RP/0/0/CPU0:XRv4#show mpls forwarding labels 94003
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94003	7015	13.0.0.12/32	Gi0/0/0/0.574	24.7.14.7	42937

CSR7 also performs a label swap operation, but connects the LDP and BGP LSPs. Since the route to 13.0.0.12/32 is learned via BGP, CSR5's BGP label of 5001 is used. The label stack becomes {5001 92021}.

```
R7#show mpls forwarding-table labels 7015
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
7015	5001	13.0.0.12/32	49261	Gi2.557	10.5.7.5

```
R7#show ip route 13.0.0.12
```

```
Routing entry for 13.0.0.12/32
```

```
Known via "bgp 24", distance 20, metric 0
```

```
Tag 13, type external
```

```
Redistributing via isis 24
```

```
Advertised by isis 24 metric-type internal level-2 route-map RM_BGP_TO_ISIS
```

```
Last update from 10.5.7.5 02:56:47 ago
```

```
Routing Descriptor Blocks:
```

```
* 10.5.7.5, from 10.5.7.5, 02:56:47 ago
```

```
Route metric is 0, traffic share count is 1
```

```
AS Hops 1
```

```
Route tag 13
```

```
MPLS label: 5001
```

CSR5 swaps label 5001 for 8000. CSR5 connects a BGP LSP to an LDP LSP since the route to 13.0.0.12/32 is IGP learned. The next-hop is CSR8 which means CSR8's local label for 13.0.0.12/32 is used. The label stack becomes {8000 92021}.

```
R5#show mpls forwarding-table labels 5001
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
5001	8000	13.0.0.12/32	52165	Gi2.558	13.5.8.8

```
R5#show ip route 13.0.0.12
```

```
Routing entry for 13.0.0.12/32
```

```
Known via "ospf 13", distance 110, metric 3, type intra area
```

```
Last update from 13.5.8.8 on GigabitEthernet2.558, 05:28:25 ago
```

```
Routing Descriptor Blocks:
```

```
* 13.5.8.8, from 13.0.0.12, 05:28:25 ago, via GigabitEthernet2.558
```

```
Route metric is 3, traffic share count is 1
```

```
R5#show mpls ldp bindings 13.0.0.12 32 neighbor 13.0.0.8
```

```
lib entry: 13.0.0.12/32, rev 28
```

```
remote binding: lsr: 13.0.0.8:0, label: 8000
```

Here we can see the inefficiency. CSR8 should be the end of the LSP since it is the remote PE. However, upon receipt of packets labeled with 8000, it forwards traffic towards XRv2. Because XRv2 changes the VPN next-hop, it must be in the transit path since VPNv6 is responsible for swapping the label on XRv2. CSR8 performs PHP to expose label 92021 to XRv2.

```
R8#show mpls forwarding-table labels 8000
```

Local Label	Outgoing Label	Prefix	Bytes Label	Outgoing	Next Hop

Label	Label	or Tunnel Id	Switched	interface	
8000	Pop Label	13.0.0.12/32	3033679	Gi2.582	13.8.12.12

XRv2 doesn't even this VRF configured locally yet it is the end of the VPN LSP. We can see the prefix indexed by RD in the LFIB entry as XRv2 swaps label 92021 for 8014. Label 8014 is CSR8's original lab that, ideally, CSR2 would have used in the first place. The reason only one label is imposed is because CSR8 is one hop away. The normal route recursion still occurs as XRv2 may need to push transport labels if CSR8 were farther away. In this case, CSR8 signals implicit-null, so the label stack becomes 8014 as packets are sent back towards CSR8.

```
RP/0/0/CPU0:XRv2#show mpls forwarding labels 92021
```

Local	Outgoing	Prefix	Outgoing	Next Hop	Bytes
Label	Label	or ID	Interface		Switched
-----	-----	-----	-----	-----	-----
92021	8014	13:2:::10:4:4:4/128	\	13.0.0.8	3396

```
RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast rd 13:2 ::10:4:4:4/128 | begin Local,
Local, (Received from a RR-client)
13.0.0.8 (metric 2) from 13.0.0.8 (13.0.0.8)
Received Label 8014
Origin incomplete, metric 1, localpref 100, valid, internal, best,
group-best, import-candidate, not-in-vrf
Received Path ID 0, Local Path ID 1, version 1766
Extended community: OSPF router-id:10.4.8.8 OSPF route-type:0:2:0x0
RT:13:2
```

```
RP/0/0/CPU0:XRv2#show route 13.0.0.8/32
Routing entry for 13.0.0.8/32
Known via "ospf 13", distance 110, metric 2, type intra area
Routing Descriptor Blocks
13.8.12.8, from 13.0.0.8, via GigabitEthernet0/0/0/0.582
Route metric is 2
No advertising protos.
```

```
RP/0/0/CPU0:XRv2#show mpls ldp bindings 13.0.0.8/32 neighbor 13.0.0.8
13.0.0.8/32, rev 13
Local binding: label: 92005
Remote bindings: (2 peers)
Peer Label
-----
13.0.0.8:0 ImpNull
```

Moving back to CSR8, we confirm that packets labeled 8014 are mapped to VPN prefix ::10:4:4:4/128 and are delivered into the proper VPN.

```
R8#show mpls forwarding-table labels 8014 detail
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Label	Outgoing interface	Next Hop
8014	No Label	::10:4:4:4/128	[V] \	3126	Gi2.548	FE80::4

MAC/Encaps=18/18, MRU=1504, Label Stack{}

005056A92C57005056A9FB1C81000DDC86DD

VPN route: OSPF

No output feature configured

Using traceroute on CSR9, we can verify these LSPs. Technically there are two LSPs: one from CSR2 to XRv2, and one from XRv2 to CSR8. We can see CSR8 in the transit path twice, and we see the VPN label being swapped at XRv2.

```
R9#traceroute ipv6
Target IPv6 address: ::10:4:4:4
Source address: ::10:9:9:9
[snip]
Tracing the route to ::10:4:4:4
 1 FD00:10:2:9::2 4 msec 3 msec 4 msec
 2 2024:24:2:14::14 [MPLS: Labels 94003/92021 Exp 0] 17 msec 16 msec 53 msec
 3 ::FFFF:24.7.14.7 [MPLS: Labels 7015/92021 Exp 0] 51 msec 56 msec 51 msec
 4 ::FFFF:10.5.7.5 [MPLS: Labels 5001/92021 Exp 0] 50 msec 52 msec 52 msec
 5 ::FFFF:13.5.8.8 [MPLS: Labels 8000/92021 Exp 0] 50 msec 51 msec 51 msec
 6 2013:13:8:12::12 [MPLS: Label 92021 Exp 0] 39 msec 38 msec 39 msec
 7 FD00:10:4:8::8 [MPLS: Label 8014 Exp 0] 23 msec 22 msec 23 msec
 8 FD00:10:4:8::4 27 msec 20 msec 21 msec
```

The second and far worse issue with setting next-hop-self for these eBGP advertisements is that sometimes, entire FECs are broken. For example, CSR3 is currently unable to send traffic to CSR1 at all. We would not expect this since the inter-PE connections appear to be operational.

```
R3#ping 10.1.1.1 source 10.3.3.3
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.1.1.1, timeout is 2 seconds:
Packet sent with a source address of 10.3.3.3
.....
Success rate is 0 percent (0/5)
```

Let's trace the LSP. XRv2 has a VPNv4 route via remote label 2124. The VPN next-hop is a BGP route with label 6004. The BGP next-hop is an IGP route via CSR8 who allocates label 8019. The entire label stack should be {8019 6004 2124}.

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf EIGRP 10.13.13.13/32 | begin 24$
24
 24.0.0.2 (metric 20) from 24.0.0.2 (24.0.0.2)
    Received Label 2124
```

```

Origin incomplete, localpref 100, valid, external, best, group-best,
import-candidate, imported
Received Path ID 0, Local Path ID 1, version 1743
Extended community: EIGRP route-info:0x8000:0 EIGRP AD:3:282 EIGRP
RHB:255:1:2560 EIGRP LM:0x0:1:1500 EIGRP VRR:0x0:13.13.13.10 RT:13:3
Connector: type: 1, Value:24:3:24.0.0.14
Source VRF: default, Source Route Distinguisher: 24:3

```

```
RP/0/0/CPU0:XRv2#show route 24.0.0.2 detail
```

```
Routing entry for 24.0.0.2/32
```

```
Known via "bgp 13", distance 200, metric 20
```

```
Tag 24, type internal
```

```
Routing Descriptor Blocks
```

```
10.6.11.6, from 13.0.0.11
```

```
Route metric is 20
```

```
Label: 0x1774 (6004)
```

```
Tunnel ID: None
```

```
Extended communities count: 0
```

```
NHID:0x0 (Ref:0)
```

```
[snip]
```

```
RP/0/0/CPU0:XRv2#show route 10.6.11.6
```

```
Routing entry for 10.6.11.6/32
```

```
Known via "ospf 13", distance 110, metric 20
```

```
Tag 13, type extern 2
```

```
Routing Descriptor Blocks
```

```
13.8.12.8, from 13.0.0.11, via GigabitEthernet0/0/0/0.582
```

```
Route metric is 20
```

```
No advertising protos.
```

```
RP/0/0/CPU0:XRv2#show mpls ldp bindings 10.6.11.6/32 neighbor 13.0.0.8
```

```
10.6.11.6/32, rev 35
```

```
Local binding: label: 92011
```

```
Remote bindings: (2 peers)
```

Peer	Label
13.0.0.8:0	8019

Using traceroute, we see that XRv2 is making no attempt to impose any labels at all. There is IP connectivity because the eBGP session works, but no MPLS services are supported to CSR2. Having just traced the route recursion, this output does not make sense. CSR8 adds some labels but the first hop is totally unlabeled, which is unacceptable.

```
RP/0/0/CPU0:XRv2#traceroute 24.0.0.2 source 13.0.0.12
```

```
Type escape sequence to abort.
```

```
Tracing the route to 24.0.0.2
```

```

1 13.8.12.8 0 msec 0 msec 0 msec
2 13.8.11.11 [MPLS: Labels 91008/6004 Exp 0] 0 msec 0 msec 0 msec

```

```

3 10.6.11.6 [MPLS: Label 6004 Exp 0] 39 msec 9 msec 0 msec
4 24.6.14.14 [MPLS: Label 94009 Exp 0] 0 msec 0 msec 0 msec
5 24.2.14.2 0 msec 0 msec 39 msec

```

Checking the FIB, we see some interesting output. XRv2 claims that 24.0.0.2/32 is a local adjacency via CSR8. I am not sure how this is even possible, since if the adjacency was really local, it wouldn't have an IP next-hop unless that next-hop was equal to the queried prefix. CSR8 is, in fact, the "right direction", but this output has many issues.

```

RP/0/0/CPU0:XRv2#show cef 24.0.0.2
24.0.0.2/32, version 261, internal 0x1000001 0x0 (ptr 0xa142e3f4) [1], 0x0
(0xa13f96ec), 0xa20 (0xa15a1398)
local adjacency 13.8.12.8
Prefix Len 32, traffic index 0, precedence n/a, priority 4
via 13.8.12.8, GigabitEthernet0/0/0/0.582, 7 dependencies, weight 0, class
0 [flags 0x0]
path-idx 0 NHID 0x0 [0xa1085250 0xa1085154]
next hop 13.8.12.8
local adjacency
local label 92024 labels imposed {ImplNull}

```

For comparison, we look at the route to 24.0.0.14/32. This route looks fine as the route recursion unfolds and the label stack is revealed. Notice the presence of the "recursive" flag below as it is absent above.

```

RP/0/0/CPU0:XRv2#show cef 24.0.0.14
24.0.0.14/32, version 682, internal 0x5000001 0x0 (ptr 0xa142e474) [1], 0x0
(0xa13f9368), 0xa08 (0xa15a14b0)
Prefix Len 32, traffic index 0, precedence n/a, priority 4
via 10.5.7.7, 3 dependencies, recursive [flags 0x6000]
path-idx 0 NHID 0x0 [0xa160b9f4 0x0]
recursion-via-/32
next hop 10.5.7.7 via 92008/0/21
next hop 13.8.12.8/32 Gi0/0/0/0.582 labels imposed {8015 7002}

```

I personally think this is an XR limitation with respect to eBGP multi-hop peerings when used with inter-AS option C. I hypothesize that XR assumes that the eBGP peer is a local adjacency regardless of the route to that peer. As soon as we correct our design flaw by preserving the next-hop across the eBGP boundary, this problem disappears. XRv2 suddenly understands that it isn't actually connected to 24.0.0.2 and should impose labels when sending traffic towards that destination. This is a good time to correct the problem on both CSR2 and XRv2. The command below is really only useful for inter-AS option C and this is the perfect case to use it.

```

! XRv2
router bgp 13
neighbor 24.0.0.2

```



```

address-family vpnv4 unicast
  next-hop-unchanged
address-family vpnv6 unicast
  next-hop-unchanged

! CSR2
router bgp 24
  address-family vpnv4
    neighbor 13.0.0.12 next-hop-unchanged
  address-family vpnv6
    neighbor 13.0.0.12 next-hop-unchanged

```

As soon as we commit these changes, XRv2 writes the proper label stack to the FIB. This is a strange issue that I found particularly difficult to troubleshoot. Traceroute from XRv2 now shows a fully label switched path from XRv2 to CSR2.

```

RP/0/0/CPU0:XRv2#show cef 24.0.0.2
24.0.0.2/32, version 679, internal 0x1000001 0x0 (ptr 0xa142e3f4) [1], 0x0
(0xa13f96ec), 0xa08 (0xa15a14b0)
Prefix Len 32, traffic index 0, precedence n/a, priority 15
  via 10.6.11.6, 3 dependencies, recursive [flags 0x6000]
    path-idx 0 NHID 0x0 [0xa160b8f4 0x0]
    recursion-via-/32
    next hop 10.6.11.6 via 92011/0/21
    next hop 13.8.12.8/32 Gi0/0/0/0.582 labels imposed {8019 6004}

```

```

RP/0/0/CPU0:XRv2#traceroute 24.0.0.2 source 13.0.0.12
Type escape sequence to abort.
Tracing the route to 24.0.0.2
 1  13.8.12.8 [MPLS: Labels 8019/6004 Exp 0] 0 msec  0 msec  0 msec
 2  13.8.11.11 [MPLS: Labels 91008/6004 Exp 0] 0 msec  0 msec  39 msec
 3  10.6.11.6 [MPLS: Label 6004 Exp 0] 9 msec  0 msec  0 msec
 4  24.6.14.14 [MPLS: Label 94009 Exp 0] 0 msec  0 msec  0 msec
 5  24.2.14.2 9 msec  0 msec  0 msec

```

Despite this fix, CSR3 still cannot reach the remote EIGRP routers. It seems like there are other unrelated issues with XRv2. With ICMP debugging enabled, we can see XRv2 is sending network unreachables back. We know it cannot possibly be a transport problem based on our verification above, so next we will check the VRF-aware routes.

```

R3#debug ip icmp
ICMP packet debugging is on

R3#traceroute 10.1.1.1 source 10.3.3.3
Type escape sequence to abort.
Tracing the route to 10.1.1.1
VRF info: (vrf in name/id, vrf out name/id)

```

```

1 10.3.12.12 !N !N !N

! CSR3
ICMP: dst (10.3.3.3) net unreachable rcv from 10.3.12.12
ICMP: dst (10.3.3.3) net unreachable rcv from 10.3.12.12
ICMP: dst (10.3.3.3) net unreachable rcv from 10.3.12.12

```

Checking the VRF-aware CEF entries on XRv2, we see these are marked as “unresolved”. Normally this would occur when there is no route, or an invalid route, to the next-hop. It also may occur if the route to the BGP next-hop is not a /32, but both of those conditions do not apply here. We can clearly see that CEF is trying to perform the route recursion and identifies both 24.0.0.14 and 24.0.0.2 as /32 host routes.

```

RP/0/0/CPU0:XRv2#show cef vrf EIGRP 10.13.13.13/32
10.13.13.13/32, version 935, internal 0x5000001 0x0 (ptr 0xa142e2f4) [1], 0x0
(0x0), 0x208 (0xa15a1960)
Prefix Len 32, traffic index 0, precedence n/a, priority 3
via 24.0.0.14, 0 dependencies, recursive, bgp-ext [flags 0x6020]
path-idx 0 NHID 0x0 [0xa0f67254 0x0]
recursion-via-/32
next hop VRF - 'default', table - 0xe0000000
unresolved
labels imposed {94006}

RP/0/0/CPU0:XRv2#show cef vrf EIGRP 10.1.1.1/32
10.1.1.1/32, version 929, internal 0x5000001 0x0 (ptr 0xa142e974) [1], 0x0
(0x0), 0x208 (0xa15a1d48)
Prefix Len 32, traffic index 0, precedence n/a, priority 3
via 24.0.0.2, 0 dependencies, recursive, bgp-ext [flags 0x6020]
path-idx 0 NHID 0x0 [0xa0f67254 0x0]
recursion-via-/32
next hop VRF - 'default', table - 0xe0000000
unresolved
labels imposed {2076}

```

This was a very difficult problem to solve. We will start from the beginning of the recursion process by checking the VPNv4 route. We see that this recursive lookup actually did succeed. Using the route to 10.1.1.1/32 as an example, the VPN label of 2076 is programmed into the FIB. In the output above, we saw label 2076 being imposed properly. We can therefore assume the problem is not with VPNv4.

```

RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf EIGRP 10.1.1.1/32 | begin 24$
24
24.0.0.2 (metric 20) from 24.0.0.2 (24.0.0.2)
Received Label 2076
Origin incomplete, metric 10880, localpref 100, valid, external, best,
group-best, import-candidate, imported
Received Path ID 0, Local Path ID 1, version 32

```

```
Extended community: EIGRP route-info:0x8000:0 EIGRP AD:3:288 EIGRP
RHB:255:1:2560 EIGRP LM:0xff:1:1500 EIGRP VRR:0x0:1.1.1.10 RT:13:3
Source VRF: default, Source Route Distinguisher: 24:3
```

The second label should be the BGP label from one of the remote ASBRs towards 24.0.0.12. The remote label shown below is 7002, but there is no local label assigned. Although not exactly intuitive, XR requires a local label for sending traffic to a destination when MPLS VPN is in use. Clearly we did not require the local label with traceroute in the global table; we had no issues with that test. XE has no such requirement as CSR8 is functioning fine without it. We would never see this problem on XRv4 since it is not running BGP labeled-unicast at all.

```
RP/0/0/CPU0:XRv2#show route 24.0.0.14 detail
Routing entry for 24.0.0.14/32
  Known via "bgp 13", distance 200, metric 10
  Tag 24, type internal
  Routing Descriptor Blocks
    10.5.7.7, from 13.0.0.5
      Route metric is 10
      Label: 0x1b5a (7002)
      Tunnel ID: None
      Extended communities count: 0
      NHID:0x0 (Ref:0)
  Route version is 0xa (10)
  No local label
[snip]
```

```
R8#show mpls forwarding-table 24.0.0.2
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
None       6029      24.0.0.2/32     0            Gi2.581    13.8.11.11
```

This raises the question of how XRv1 is working correctly. Both XRv1 and XRv2 have the same label allocation policy which only allocates labels for local loopbacks within the "13.0.0.0/24 ge 32" range. Based on my experience and testing, XR will automatically allocate a label for BGP prefixes learned from eBGP peers regardless of the policy. Below, we can see that XRv1 allocates the local label 91004 which allows it to function as an LSR. This is not in compliance with the policy but happens anyway, probably because XR knows that it must do this in order for MPLS forwarding to work.

```
RP/0/0/CPU0:XRv1#show route 24.0.0.2 detail
Routing entry for 24.0.0.2/32
  Known via "bgp 13", distance 20, metric 20, [ei]-bgp, labeled unicast
  (3107)
  Tag 24, type external
  Routing Descriptor Blocks
    10.6.11.6, from 10.6.11.6, BGP external
      Route metric is 20
```

```

Label: 0x1774 (6004)
Tunnel ID: None
Extended communities count: 0
NHID:0x0 (Ref:0)
Route version is 0xe (14)
Local Label: 0x1637c (91004)
[snip]

```

To support the claim that eBGP and iBGP peers are treated differently with respect to label allocation, I will completely delete CSR6 as an eBGP peer so that XRv1 must learn the AS 24 routes via CSR7 (learned ultimately through CSR5 using iBGP). After this change, XRv1 no longer allocates labels for the AS 24 prefixes. This would break L3VPN route recursion if XRv1 were a PE. Without a local label for those prefixes, XRv1 cannot impose labels 7000 or 7002 at imposition for VPN services.

```

! XRv1
router bgp 13
no neighbor 10.6.11.6

```

```

RP/0/0/CPU0:XRv1#show bgp ipv4 labeled-unicast labels | begin Network
  Network          Next Hop      Rcvd Label    Local Label
*>i13.0.0.8/32      13.0.0.8        3             91001
*>i13.0.0.12/32     13.0.0.12       3             91002
*>i24.0.0.2/32      10.5.7.7        7000          nolabel
*>i24.0.0.14/32     10.5.7.7        7002          nolabel

```

Before continuing, I restore XRv1's original neighbor configuration to CSR6. The most obvious way to fix this is to expand the label allocation policy to encompass the remote loopbacks on all XR PEs within AS 13. Those who configure option C and simply say "allocate-label all" would never have seen this problem in the first place. Instead of modifying the RPL itself, I create a new prefix-set to better reflect the prefixes that require label allocation.

```

! XRv2
prefix-set PS_PE_LOOPBACKS
  13.0.0.0/24 ge 32,
  24.0.0.0/24 ge 32
end-set

no prefix-set PS_LOCAL_LOOPBACKS

router bgp 13
address-family ipv4 unicast
allocate-label route-policy RPL_IF_DEST_PASS(PS_PE_LOOPBACKS)

```

If we check the local labels for these BGP routes, now they exist. The CEF entries within the VPN tables are also resolved. We see a 3 label stack as expected, including the LDP, BGP, and VPN labels in

sequence. In summary, ensure your XR PEs running IPv4 labeled-unicast are configured to allocate local labels for remote and local loopbacks. This is not required on XR ASBRs, XE ASBRs, or XE PEs.

```
RP/0/0/CPU0:XRv2#show route 24.0.0.2 detail
Routing entry for 24.0.0.2/32
  Known via "bgp 13", distance 200, metric 20, [ei]-bgp
  Tag 24, type internal
  Routing Descriptor Blocks
    10.6.11.6, from 13.0.0.11
      Route metric is 20
      Label: 0x1774 (6004)
      Tunnel ID: None
      Extended communities count: 0
      NHID:0x0 (Ref:0)
  Route version is 0xa (10)
  Local Label: 0x16770 (92016)
[snip]

RP/0/0/CPU0:XRv2#show cef vrf EIGRP 10.1.1.1/32
10.1.1.1/32, version 11, internal 0x5000001 0x0 (ptr 0xa146a8f4) [1], 0x0
(0x0), 0x208 (0xa15534d8)
  Prefix Len 32, traffic index 0, precedence n/a, priority 3
  via 24.0.0.2, 5 dependencies, recursive, bgp-ext [flags 0x6020]
  path-idx 0 NHID 0x0 [0xa15bbff4 0x0]
  recursion-via-/32
  next hop VRF - 'default', table - 0xe0000000
  next hop 24.0.0.2 via 92016/0/21
  next hop 13.8.12.8/32 Gi0/0/0/0.582 labels imposed {8019 6004 2076}
```

If additional control is needed over these local labels which would normally not be allocated, we can statically define the actual label value once the RPL permits the allocation. Although not related to option C at all, I adjust the local label values so they are more visually apparent on XRv2. I use the format 925XX where XX is equal to the last octet.

```
! XRv2
mpls static
  address-family ipv4 unicast
    local-label 92502 allocate per-prefix 24.0.0.2/32
    local-label 92514 allocate per-prefix 24.0.0.14/32
```

This creates a syslog on XRv2 saying that there is a label discrepancy. BGP allocated the original values dynamically, and since we overrode them, we must clear this discrepancy.

```
! XRv2
%ROUTING-MPLS_STATIC-4-ERR_STATIC_LABEL_DISCREPANCY : The system detected 2
label discrepancies (static label could not be allocated due to conflict with
```

other applications). Please use 'clear mpls static local-label discrepancy' to fix this issue.

After issuing the command to clear the discrepancy, the problem is resolved. Checking the LFIB against these local labels, we can see them properly installed. These labels may never be used, but they are required to exist for MPLS to support L3VPN on XR.

```
RP/0/0/CPU0:XRv2#show mpls forwarding labels 92500 92599
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
92502	6004	24.0.0.2/32		10.6.11.6	0
92514	7002	24.0.0.14/32		10.5.7.7	0

For brevity, I will use traceroute to verify some key VPN connections. Both CSR1 and XRv4 can access central services resources. Per our traffic egress policy, traffic towards CSR8 egresses AS 24 via CSR6.

```
R1#traceroute 110.0.0.0 source 10.1.1.1
Type escape sequence to abort.
Tracing the route to 110.0.0.0
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.2.2 5 msec 4 msec 4 msec
 2 24.2.14.14 [MPLS: Labels 94003/8003 Exp 0] 25 msec 50 msec 93 msec
 3 24.6.14.6 [MPLS: Labels 6019/8003 Exp 0] 26 msec 33 msec 32 msec
 4 10.6.11.11 [MPLS: Labels 91001/8003 Exp 0] 21 msec 30 msec 73 msec
 5 10.8.10.8 [MPLS: Label 8003 Exp 0] 47 msec 32 msec 41 msec
 6 10.8.10.10 15 msec 18 msec 17 msec
```

```
RP/0/0/CPU0:XRv3#traceroute 110.0.0.3 source 10.13.13.13
Type escape sequence to abort.
Tracing the route to 110.0.0.3
 1 10.13.14.14 0 msec 0 msec 0 msec
 2 24.6.14.6 [MPLS: Labels 6019/8006 Exp 0] 9 msec 39 msec 9 msec
 3 10.6.11.11 [MPLS: Labels 91001/8006 Exp 0] 9 msec 9 msec 9 msec
 4 10.8.10.8 [MPLS: Label 8006 Exp 0] 9 msec 29 msec 29 msec
 5 10.8.10.10 19 msec 19 msec 9 msec
```

Alternatively, traffic towards XRv2 egresses via CSR7. These LSPs are functional as well. The fact that CSR1 can reach CSR3 is an indication that XRv2's L3VPN role as a PE has been configured correctly.

```
R1#traceroute 10.3.3.3 source 10.1.1.1
Type escape sequence to abort.
Tracing the route to 10.3.3.3
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.2.2 5 msec 4 msec 4 msec
 2 24.2.14.14 [MPLS: Labels 94005/92006 Exp 0] 468 msec 40 msec 17 msec
 3 24.7.14.7 [MPLS: Labels 7076/92006 Exp 0] 36 msec 21 msec 57 msec
```

```

4 10.5.7.5 [MPLS: Labels 5059/92006 Exp 0] 50 msec 32 msec 17 msec
5 13.5.8.8 [MPLS: Labels 8018/92006 Exp 0] 18 msec 21 msec 20 msec
6 13.8.12.12 [MPLS: Label 92006 Exp 0] 72 msec 38 msec 46 msec
7 10.3.12.3 201 msec 74 msec 60 msec

```

RP/0/0/CPU0:XRv3#traceroute 10.3.3.3 source 10.13.13.13

Type escape sequence to abort.

Tracing the route to 10.3.3.3

```

1 10.13.14.14 0 msec 0 msec 0 msec
2 24.7.14.7 [MPLS: Labels 7076/92006 Exp 0] 9 msec 19 msec 9 msec
3 10.5.7.5 [MPLS: Labels 5059/92006 Exp 0] 9 msec 19 msec 19 msec
4 13.5.8.8 [MPLS: Labels 8018/92006 Exp 0] 29 msec 19 msec 19 msec
5 13.8.12.12 [MPLS: Label 92006 Exp 0] 29 msec 19 msec 19 msec
6 10.3.12.3 29 msec 19 msec 29 msec

```

Earlier, we used BGP weight on XRv2 so that CSR8 would prefer different egress points from AS 13 as well. Traffic from CSR4 to CSR9 inside the OSPF VPN egresses via CSR6. Traffic from the central services router to XRv3 egresses via CSR7. I use IPv6 VPN routes for this test, but the LSP would be the same for IPv4 VPN routes as well.

R4#traceroute ipv6

Target IPv6 address: ::10:9:9:9

Source address: ::10:4:4:4

[snip]

Tracing the route to ::10:9:9:9

```

1 FD00:10:4:8::8 14 msec 4 msec 4 msec
2 2013:13:8:11::11 [MPLS: Labels 91008/6004/2010 Exp 0] 23 msec 16 msec 17 msec
3 ::FFFF:10.6.11.6 [MPLS: Labels 6004/2010 Exp 0] 47 msec 25 msec 28 msec
4 2024:24:6:14::14 [MPLS: Labels 94009/2010 Exp 0] 41 msec 46 msec 43 msec
5 FD00:10:2:9::2 [MPLS: Label 2010 Exp 0] 25 msec 25 msec 25 msec
6 FD00:10:2:9::9 25 msec 25 msec 25 msec

```

R10#traceroute ipv6

Target IPv6 address: ::10:13:13:13

Source address: ::110:0:0:1

[snip]

Tracing the route to ::10:13:13:13

```

1 FD00:10:8:10::8 11 msec 2 msec 2 msec
2 ::FFFF:13.5.8.5 [MPLS: Labels 5015/7002/94001 Exp 0] 103 msec 112 msec 178 msec
3 ::FFFF:10.5.7.7 [MPLS: Labels 7002/94001 Exp 0] 104 msec 114 msec 106 msec
4 2024:24:7:14::14 [MPLS: Label 94001 Exp 0] 157 msec 119 msec 98 msec
5 ::10:13:13:13 [AS 24] 156 msec 134 msec 169 msec

```

With the LSPs operational, we bring up the OSPFv3 sham-link between CSR8 and CSR2. The configuration has not changed at all from the first time it was configured with option A and reused again

with option B. We have tested this extensively already so we will be brief with this verification. As long as the sham-link endpoints are reachable (VPNv6 between PEs) then the sham-links should form. Two sham-links exist: one for IPv4 and one for IPv6 OSPFv3 AFIs. I also bring up the backdoor link between CSR4 and CSR9.

```
R8#show ospfv3 vrf OSPF sham-links | include ^Sham
Sham Link OSPFv3_SL0 to address FD00::2 is up
Sham Link OSPFv3_SL1 to address FD00::2 is up
```

CSR9 learns routes to CSR4's loopback via an intra-area path. This proves that the sham-link is working. Even if the backdoor link were down, we would still see these routes as intra-area as a result of the sham-links.

```
R9#show ip route 10.4.4.4
Routing entry for 10.4.4.4/32
  Known via "ospfv3 2", distance 110, metric 3, type intra area
  Last update from 10.2.9.2 on GigabitEthernet2.529, 00:00:55 ago
  Routing Descriptor Blocks:
    * 10.2.9.2, from 10.4.8.4, 00:00:55 ago, via GigabitEthernet2.529
      Route metric is 3, traffic share count is 1
```

```
R9#show ipv6 route ::10:4:4:4
Routing entry for ::10:4:4:4/128
  Known via "ospf 2", distance 110, metric 3, type intra area
  Route count is 1/1, share count 0
  Routing paths:
    FE80::2, GigabitEthernet2.529
    Last updated 00:00:59 ago
```

Earlier, we conducted the traceroute shown below from CSR9 to CSR4 using the IPv6 VPN routes. We saw CSR8 in the transit path twice due to XRV2 changing the VPNv6 next-hop. Now that the issue is resolved, traffic is sent directly to CSR8. I have temporarily broken CSR6's transit links (not shown) to ensure CSR7 can be a failover for traffic destined to CSR8, which it can. I did this as an additional test to show the value in having multiple inter-AS connections and using BGP attributes and IGP metrics to influence the traffic patterns.

```
R9#traceroute ipv6
Target IPv6 address: ::10:4:4:4
Source address: ::10:9:9:9
[snip]
Tracing the route to ::10:4:4:4
 1 FD00:10:2:9::2 4 msec 4 msec 4 msec
 2 2024:24:2:14::14 [MPLS: Labels 94003/8014 Exp 0] 50 msec 14 msec 7 msec
 3 ::FFFF:24.7.14.7 [MPLS: Labels 7075/8014 Exp 0] 28 msec 35 msec 51 msec
 4 ::FFFF:10.5.7.5 [MPLS: Labels 5003/8014 Exp 0] 23 msec 110 msec 10 msec
 5 FD00:10:4:8::8 [MPLS: Label 8014 Exp 0] 130 msec 75 msec 6 msec
```



```
6 FD00:10:4:8::4 10 msec 137 msec 111 msec
```

This concludes the option C L3VPN section. There are many ways to implement this design since it is very complex and involved. The biggest benefit is a simple ASBR configuration; the future labs, such as L2VPN, are very straightforward since the MPLS services are end-to-end.

8.4.3.2 L2VPN

MPLS L2VPN with option C is generally straightforward. Unlike option A, we do not need to terminate PWs on the ASBRs to remove all MPLS encapsulation. Also unlike option B, we do not need to create MSPWs via the ASBRs since the PEs have end-to-end reachability. For this test, I will use BGP auto-discovery with BGP signaling as opposed to LDP signaling shown in the past examples. I will still use disparate VPN IDs and route-targets so that we can focus on the inter-AS mechanics. The basic VFI and bridge-domain configurations are similar to options A and B and are not discussed in detail. The significant difference is that I do not apply the PW template to this VFI as BGP signaling does not appear to support the CW at all. I re-use the RT values of 24:3 and 13:3 for reasons described later; this will not interfere with the EIGRP VPN at all since it is an entirely different AFI.

```
! CSR2
l2vpn vfi context VPLS
  vpn id 200
  autodiscovery bgp signaling bgp
  ve id 2
    route-target export 24:3
    route-target import 24:3
  no auto-route-target

bridge-domain 3
  member GigabitEthernet2 service-instance 3
  member vfi VPLS

! CSR8
l2vpn vfi context VPLS
  vpn id 800
  autodiscovery bgp signaling bgp
  ve id 8
    route-target export 13:3
    route-target import 13:3
  no auto-route-target

bridge-domain 3
  member GigabitEthernet2 service-instance 3
  member vfi VPLS
```

Next, we will quickly verify that the VFIs were configured properly. Notice that the RT policies do not currently match, nor do the VPN IDs.

```
R2#show l2vpn vfi name VPLS
Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No
VFI name: VPLS, state: up, type: multipoint, signaling: BGP
VPN ID: 200, VE-ID: 2, VE-SIZE: 10
RD: 24:200, RT: 24:3
Bridge-Domain 3 attachment circuits:
Pseudo-port interface: pseudowire100033
Interface          Peer Address      VE-ID  Local Label  Remote Label      S
```

```
R8#show l2vpn vfi name VPLS
Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No
VFI name: VPLS, state: up, type: multipoint, signaling: BGP
VPN ID: 800, VE-ID: 8, VE-SIZE: 10
RD: 13:800, RT: 13:3
Bridge-Domain 3 attachment circuits:
Pseudo-port interface: pseudowire100006
Interface          Peer Address      VE-ID  Local Label  Remote Label      S
```

Next, we configure BGP. Unlike other AFIs that use extended-communities, the XE parser does not automatically add “send-community extended” to the BGP neighbor statements for the L2VPN VPLS AFI. For intra-AS this does not seem to matter, but XE won’t encode these communities when sending routes to external peers without the explicit command. Thus, I add the command to CSR2’s peer to XRv2, but not to CSR8 or XRv2 at all. XR is smart enough to send these communities to eBGP peers for AFIs that require it without being explicitly told to do so. I initially do not configure CSR8 as an RR client because, at a glance, the basic BGP advertisement rules would indicate it is not necessary.

```
! XRv2
router bgp 13
 address-family l2vpn vpls-vpws

 neighbor 13.0.0.8
   use session-group IBGP
   address-family l2vpn vpls-vpws

 neighbor 24.0.0.2
   address-family l2vpn vpls-vpws
   route-policy RPL_PASS in
   route-policy RPL_PASS out
   Signalling ldp disable

! CSR2
router bgp 24
 address-family l2vpn vpls
   neighbor 13.0.0.12 activate
   neighbor 13.0.0.12 send-community extended
   neighbor 13.0.0.12 suppress-signaling-protocol ldp

! CSR8
```

```

router bgp 24
  address-family l2vpn vpls
    neighbor 13.0.0.12 activate
    neighbor 13.0.0.12 suppress-signaling-protocol ldp

```

Checking CSR2 and XRv2, we can see the eBGP peer comes up. XRv2 learns routes from CSR2 but not vice versa. XRv2 also does not learn any VPLS routes from CSR8, either.

```

RP/0/0/CPU0:XRv2#show bgp l2vpn vpls summary | begin ^Neigh
Neighbor      Spk    AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
13.0.0.8      0     13    6717    6361       3     0     0  00:10:15      0
24.0.0.2      0     24   1399    1127       3     0     0  00:16:11      1

R2#show bgp l2vpn vpls all summary | begin ^Neigh
Neighbor      V      AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  State/PfxRcd
13.0.0.12     4      13     60     62         3     0     0  00:16:29      0

```

First, we will solve the intra-AS problem inside AS 13. XR's BGP debug for L2VPN AFI does not reveal the problem, but the issue is similar to that seen with option B. Since CSR8 is not an RR-client, XRv2 has no compelling reason to accept VPLS routes from this peer if it isn't consuming the information locally. In the option C design, these iBGP peers would all be RR-clients and we would never see this problem. If we configured CSR8 as an RR-client, despite it not having any measurable impact on the iBGP topology, the problem would be solved. This is what we did for VPNv4/v6 with no issues. Alternatively, we can simply instruct XRv2 to retain all RTs much like the option B ASBRs did. This is odd for an option C RR but is valid in this specific design as there are no other iBGP peers in AS 13 that are part of any VPLS instance. In an option C design, configuring CSR8 as an RR-client makes the most sense and is most realistic, but for variety, we will use the alternative method. Once we commit this change, XRv2 learns a VPLS route from CSR8.

```

! XRv2
router bgp 13
  address-family l2vpn vpls-vpws
    retain route-target all

```

```

RP/0/0/CPU0:XRv2#show bgp l2vpn vpls summary | begin ^Neigh
Neighbor      Spk    AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
13.0.0.8      0     13    6817    6457       5     0     0  00:08:36      1
24.0.0.2      0     24   1423    1151       5     0     0  00:26:21      1

```

Next, we need to figure out why CSR2 is not learning routes from XRv2. By enabling debugging on CSR2, we see this is a classic RT import/export problem. CSR2 advertises routes with RT:24:3 and imports the same RT. The route received from XRv2, originated by CSR8, has RT:13:3. No local VPLS instances are importing this RT, so the route is dropped.

```

R2#debug bgp l2vpn vpls updates in
BGP updates debugging is on (inbound) for address family: L2VPN Vpls

```

```

BGP(9): (base) 13.0.0.12 send UPDATE (format) 24:200:VEID-2:Blk-1:VBS-10:LB-
2026/136, next 24.0.0.2, metric 0, path Local, extended community RT:24:3
L2VPN L2:0x0:MTU-1500
BGP(9): 13.0.0.12 rcvd UPDATE w/ attr: nexthop 13.0.0.12, origin ?, merged
path 13, AS_PATH , extended community RT:13:3 L2VPN L2:0x0:MTU-1500
BGP(9): 13.0.0.12 rcvd 13:800:VEID-8:Blk-1:VBS-10:LB-8022/136 -- DENIED due
to: extended community not supported;

```

It follows that we would see the same issue on CSR8. We can quickly confirm with debugs for completeness.

```

R8#debug bgp l2vpn vpls updates in
BGP updates debugging is on (inbound) for address family: L2VPN Vpls

BGP(9): (base) 13.0.0.12 send UPDATE (format) 13:800:VEID-8:Blk-1:VBS-10:LB-
8022/136, next 13.0.0.8, metric 0, path Local, extended community RT:13:3
L2VPN L2:0x0:MTU-1500
BGP(9): 13.0.0.12 rcvd UPDATE w/ attr: nexthop 24.0.0.2, origin ?, localpref
100, metric 0, merged path 24, AS_PATH , extended community RT:24:3 L2VPN
L2:0x0:MTU-1500
BGP(9): 13.0.0.12 rcvd 24:200:VEID-2:Blk-1:VBS-10:LB-2026/136 -- DENIED due
to: extended community not supported;

```

Let's assume that the RT "policy" defined earlier is still in effect. That is to say, the ASes cannot adjust their RT policies to import anything outside of the local AS. We can solve this using RT rewrite again, and because I used the same RTs from the L3VPN section, we don't need to modify any route-maps or RPLs. The local AS exported RT will match the existing conditionals and be rewritten to the remote AS imported RT. As an aside, we also notice the next-hop of 13.0.0.12 above; this is incorrect and should be 13.0.0.8, so we must use next-hop-unchanged on XRv2 at a minimum.

```

! CSR2
router bgp 24
  address-family l2vpn vpls
    neighbor 13.0.0.12 route-map RM_RT_REWRITE out

! XRv2
router bgp 13
  neighbor 24.0.0.2
  address-family l2vpn vpls-vpws
    next-hop-unchanged
    route-policy RPL_RT_REWRITE out

```

The VPLS immediately forms. Notice that we did not have to adjust the "VPLS ID" or anything of the sort. The fact that the VPN IDs differ with BGP signaling only affects the auto-RD generation. LDP signaling

used the VPLS ID as the AGI field to identify nodes in the same VPLS instance. No such concept exists with BGP signaling.

```
R8#show l2vpn atom vc
```

Interface	Peer ID	VC ID	Service		Status
			Type	Name	
pw100007	2	800	vfi	VPLS	UP

```
R2#show l2vpn atom vc
```

Interface	Peer ID	VC ID	Service		Status
			Type	Name	
pw100034	8	200	vfi	VPLS	UP

Looking at the details, we see a significant problem that is easy to overlook. Somehow, one of the routers selected the incorrect label. CSR2 thinks its local label is 2023 but CSR8 claims the remote label along that PW is 2033. Forwarding will certainly not work in this case.

```
R2#show l2vpn vfi name VPLS
```

Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No

VFI name: VPLS, state: up, type: multipoint, signaling: BGP

VPN ID: 200, VE-ID: 2, VE-SIZE: 10

RD: 24:200, RT: 24:3

Bridge-Domain 3 attachment circuits:

Pseudo-port interface: pseudowire100033

Interface	Peer Address	VE-ID	Local Label	Remote Label	S
pseudowire100034	13.0.0.12	8	2023	8023	Y

```
R8#show l2vpn vfi name VPLS
```

Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No

VFI name: VPLS, state: up, type: multipoint, signaling: BGP

VPN ID: 800, VE-ID: 8, VE-SIZE: 10

RD: 13:800, RT: 13:3

Bridge-Domain 3 attachment circuits:

Pseudo-port interface: pseudowire100006

Interface	Peer Address	VE-ID	Local Label	Remote Label	S
pseudowire100007	24.0.0.2	2	8023	2033	Y

I have personally never seen this happen before. After several L2VPN XC reprovisions and BGP clears, I attempted to reboot. After the routers come back online, we check the VFI details again and see a similar issue. It seems that CSR2 has allocated two label ranges.

```
R2#show l2vpn vfi name VPLS | begin Interface
```

Interface	Peer Address	VE-ID	Local Label	Remote Label	S
pseudowire100002	13.0.0.12	8	2007	8001	Y

```
R8#show l2vpn vfi name VPLS | begin Interface
Interface      Peer Address    VE-ID  Local Label  Remote Label  S
pseudowire100002  24.0.0.2        2      8001        2037         Y
```

CSR2 is the culprit. It's local VPLS route clearly shows label base 2000, yet outbound debugging on CSR2 shows it advertising a route with label base 2030. CSR2 allocates a second label base and adjusts the outbound update to carry this new value.

```
R2#show bgp l2vpn vpls all ve-id 2 block-offset 1
BGP routing table entry for 24:200:VEID-2:Blk-1/136, version 4
Paths: (1 available, best #1, table L2VPN-VPLS-BGP-Table)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (24.0.0.2)
    Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
    AGI version(0), VE Block Size(10) Label Base(2000)
    Local Label Base(2030) Block ID (2)
    Extended Community: RT:24:3 L2VPN L2:0x0:MTU-1500
    mpls labels in/out 2030/2000
    rx pathid: 0, tx pathid: 0x0
```

! CSR2

```
R2#debug bgp l2vpn vpls updates out
```

BGP updates debugging is on (outbound) for address family: L2VPN Vpls

```
BGP(9): (base) 13.0.0.12 send UPDATE (format) 24:200:VEID-2:Blk-1:VBS-10:LB-
2030/136, next 24.0.0.2, metric 0, path Local, extended community RT:24:3
L2VPN L2:0x0:MTU-1500
```

Checking CSR2's LFIB, we can actually see two sets of labels have been allocated. It correctly computes that label 2007 should be forwarded to the EFP (not dropped), but because it signaled an additional label base to AS 13, CSR8 is using the incorrect label to send traffic into AS 24. When label 2037 is received, CSR2 will drop the traffic. This is the local label base (LLB) in action, shown above. I cannot find any clear documentation on what this is or why it is useful. After quickly skimming RFC 4761, this is no mention of "local label base". I would guess that the CSR2 would be smart enough to swap label 2037 for label 2007, then perform another lookup on 2007 to deliver the traffic to the correct EFP. This does not appear to happen, despite the MPLS in/out labels in the BGP prefix appearing to indicate so.

```
R2#show mpls forwarding-table labels 2000 - 2009
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id    Switched     interface
2000       No Label   lbl-blk-id(1:0) 0             drop
2001       No Label   lbl-blk-id(1:1) 0             drop
2002       No Label   lbl-blk-id(1:2) 0             drop
```

2003	No Label	lbl-blk-id(1:3)	0	drop	
2004	No Label	lbl-blk-id(1:4)	0	drop	
2005	No Label	lbl-blk-id(1:5)	0	drop	
2006	No Label	lbl-blk-id(1:6)	0	drop	
2007	No Label	lbl-blk-id(1:7)	0	none	point2point
2008	No Label	lbl-blk-id(1:8)	0	drop	
2009	No Label	lbl-blk-id(1:9)	0	drop	

```
R2#show mpls forwarding-table labels 2030 - 2039
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
2030	No Label	lbl-blk-id(2:0)	0	drop	
2031	No Label	lbl-blk-id(2:1)	0	drop	
2032	No Label	lbl-blk-id(2:2)	0	drop	
2033	No Label	lbl-blk-id(2:3)	0	drop	
2034	No Label	lbl-blk-id(2:4)	0	drop	
2035	No Label	lbl-blk-id(2:5)	0	drop	
2036	No Label	lbl-blk-id(2:6)	0	drop	
2037	No Label	lbl-blk-id(2:7)	0	drop	
2038	No Label	lbl-blk-id(2:8)	0	drop	
2039	No Label	lbl-blk-id(2:9)	0	drop	

Unable to make sense of this behavior on CSR2, I decided to configure a workaround. XRv4 will be the L2VPN VPLS RR for AS 24. It will only serve in this role for this one AFI; I theorize that if a VPLS PE is also running eBGP, then the router assumes it isn't an option C RR and makes label adjustments to support alternative architectures. XRv2 and XRv4 must establish a new BGP session to support this, and the L2VPN VPLS AFI is removed between CSR2 and XRv2. XRv4 configures CSR2 as an RR client rather than retain the RTs as XRv2 did. CSR2 will use the RT rewrite policy towards the RR so we don't need to redefine it on XRv4. This isn't a particularly realistic use of RT rewrite since the RR should be performing the rewrite towards an eBGP peer, but since there are no other iBGP VPLS participants in AS 24, this is a valid solution.

```
! XRv2
router bgp 13
  neighbor 24.0.0.2
    no address-family l2vpn vpls-vpws

  neighbor 24.0.0.14
    remote-as 24
    ebgp-multihop 8
    update-source Loopback0
    address-family l2vpn vpls-vpws
      route-policy RPL_PASS in
      route-policy RPL_RT_REWRITE out
    Signalling ldp disable
    next-hop-unchanged
```

```

! XRv4
route-policy RPL_PASS
  pass
end-policy

router bgp 24
  neighbor 24.0.0.2
    address-family l2vpn vpls-vpws
    route-reflector-client

  neighbor 13.0.0.12
    remote-as 13
    ebgp-multihop 8
    update-source Loopback0
    address-family l2vpn vpls-vpws
    route-policy RPL_PASS in
    route-policy RPL_PASS out
    Signalling ldp disable
    next-hop-unchanged

```

Suddenly, CSR2 stops adding the LLB to its VPLS updates and the PW starts working. The in/out labels are synchronized between both PEs since CSR2 is not using multiple label bases for the same PW. In real life, one would probably never run into this issue as the RR would never also be a PE. I wanted to illustrate that the VPN IDs and RDs can be totally different for inter-AS BGP-signaled VPLS. As long as the RTs match, the PWs will form.

```

R2#show bgp l2vpn vpls rd 24:200 ve-id 2 block-offset 1
BGP routing table entry for 24:200:VEID-2:Blk-1/136, version 5
Paths: (1 available, best #1, table L2VPN-VPLS-BGP-Table)
  Advertised to update-groups:
    8
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (24.0.0.2)
    Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
    AGI version(0), VE Block Size(15) Label Base(2040)
    Extended Community: RT:24:3 L2VPN L2:0x0:MTU-1500
    mpls labels in/out exp-null/2040
    rx pathid: 0, tx pathid: 0x0

```

```

R2#show l2vpn vfi name VPLS | begin Interface
Interface      Peer Address  VE-ID  Local Label  Remote Label  S
pseudowire100004  13.0.0.8      8      2047         8001         Y

```

```

R8#show l2vpn vfi name VPLS | begin Interface
Interface      Peer Address  VE-ID  Local Label  Remote Label  S
pseudowire100014  24.0.0.2      2      8001         2047         Y

```


When we check the PW details on CSR2 and CSR8, we can see the proper label stacks are build. CSR2 only has 2 labels since there is no BGP labeled-unicast running in AS 24. CSR8 requires three labels since the P-routers in AS 13 would not have reachability to CSR2, the PW endpoint. This allows MPLS to tunnel traffic across the core to the ASBRs that do have reachability to the remote loopbacks.

```
R2#show l2vpn atom vc service-name VPLS detail | include label_stack
Output interface: Gi2.524, imposed label stack {94003 8001}
```

```
R8#show l2vpn atom vc service-name VPLS detail | include label_stack
Output interface: Gi2.581, imposed label stack {91008 6029 2047}
```

We quickly test the PW using ping and traceroute from the CSR3.

```
R3#ping vrf VPLS 10.0.0.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.1, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 8/9/11 ms
```

```
R3#traceroute vrf VPLS 10.0.0.1
Type escape sequence to abort.
Tracing the route to 10.0.0.1
VRF info: (vrf in name/id, vrf out name/id)
 1 10.0.0.1 46 msec 8 msec 8 msec
```

Although no CE devices will be attached, I will test static ATOM (E-LINE using EVPL) between CSR2 and CSR8 as well. This will allow us to test inter-AS CFM. The configuration is almost identical on CSR2 and CSR8 with the only difference being the MEP IDs. I used the Y.1731 ITU carrier code (ICC) based MEG-ID just for variety. It consists of a 1 – 6 character ICC and a 1 – 12 character unique MEG-ID code (UMC). It can be up to 13 characters long. In my case, I use a 4 character ICC and a 9 character UMC; in real life these ICCs would be specific to each carrier with the MEG-ID changing per CFM instance. For demonstration and variety, I use the same one for inter-AS operations.

```
! CSR2
ethernet cfm ieee
ethernet cfm global
ethernet cfm logging
ethernet cfm domain C level 5
  service icc OPTC 123456789 evc EVC_999
    mep mpid 8
    continuity-check
    continuity-check static rmep

ethernet evc EVC_999
```

```

interface GigabitEthernet2
  service instance 999 ethernet EVC_999
  encapsulation dot1q 999
  cfm mep domain C mpid 2
  cos 3
  alarm notification all

! CSR8
ethernet cfm ieee
ethernet cfm global
ethernet cfm logging
ethernet cfm domain C level 5
  service icc OPTC 123456789 evc EVC_999
  mep mpid 2
  continuity-check
  continuity-check static rmep

ethernet evc EVC_999

interface GigabitEthernet2
  service instance 999 ethernet EVC_999
  encapsulation dot1q 999
  cfm mep domain C mpid 8
  cos 3
  alarm notification all

```

A quick check of the local MEPs shows they are configured correctly. Notice that the concatenated ICC/UMC is shown as a single string for each MEP.

```

R2#show ethernet cfm maintenance-points local domain C
Local MEPs:

```

MPID	Domain Name	Lvl	MacAddress	Type	CC
Ofld	Domain Id	Dir	Port	Id	
	MA Name		SrvcInst	Source	
	EVC name				

2	C	5	001e.1415.dbbf	BD-V	I
No	C	Up	Gi2	0	
	icc OPTC123456789		999	Static	
	EVC_999				

```

R8#show ethernet cfm maintenance-points local domain C
Local MEPs:

```

MPID	Domain Name	Lvl	MacAddress	Type	CC
Ofld	Domain Id	Dir	Port	Id	
	MA Name		SrvcInst	Source	
	EVC name				

```

-----
8      C                               5      001e.e64d.4dbf BD-V  I
No    C                               Up    Gi2          0
      icc OPTC123456789              999          Static
      EVC_999

```

Next, we must configure the PWs and bind them to local XC processes. The PW configuration is almost identical except with swapped IP addressing. The template was defined long ago and was discussed in the introductory section; it enables the control-word and sequencing. Both CSR2 and CSR8 have the same xconnect binding as they use the same EFP and PW interface enumerations.

```

! CSR2
interface pseudowire28
 source template type pseudowire TMP_VPLS
 encapsulation mpls
 neighbor 13.0.0.8 28

! CSR8
interface pseudowire28
 source template type pseudowire TMP_VPLS
 encapsulation mpls
 neighbor 24.0.0.2 28

! CSR2 and CSR8
l2vpn xconnect context EVPL_28
 member pseudowire28
 member GigabitEthernet2 service-instance 999

```

We confirm that the PW comes up before worrying about CFM. No issues here.

```

R2#show l2vpn atom vc service-name EVPL_28

```

Interface	Peer ID	VC ID	Type	Service Name	Status
pw28	13.0.0.8	28	p2p	EVPL_28	UP

```

R8#show l2vpn atom vc service-name EVPL_28

```

Interface	Peer ID	VC ID	Type	Service Name	Status
pw28	24.0.0.2	28	p2p	EVPL_28	UP

Next, we check for CFM RMEPs on each router. Since we statically identified the RMEPs were supposed to see, this adds a bit of security. When an expected RMEP is absent or an unexpected RMEP is preset, an alarm is raised. In our case, the network has converged properly as both CSR2 and CSR8 can see one another via CFM CCMs. The MA-ID (or MEG-ID) is identified by ICC as expected.

```
R8#show ethernet cfm maintenance-points remote domain C
```

MPID	Domain Name	MacAddress	IfSt	PtSt
Lvl	Domain ID	Ingress		
RDI	MA Name	Type Id	SrvcInst	
	EVC Name		Age	
	Local MEP Info			
8	C	001e.e64d.4dbf	Up	Up
5	C	Gi2: (13.0.0.8, 28)		
-	icc OPTC123456789	XCON N/A	999	
	EVC_999		1s	
	MPID: 2 Domain: C MA: icc OPTC123456789			

```
R8#show ethernet cfm maintenance-points remote domain C
```

MPID	Domain Name	MacAddress	IfSt	PtSt
Lvl	Domain ID	Ingress		
RDI	MA Name	Type Id	SrvcInst	
	EVC Name		Age	
	Local MEP Info			
2	C	001e.1415.dbbf	Up	Up
5	C	Gi2: (24.0.0.2, 28)		
-	icc OPTC123456789	XCON N/A	999	
	EVC_999		1s	
	MPID: 8 Domain: C MA: icc OPTC123456789			

As a final test, we use CFM loopback messages (LBM) to test connectivity. The inter-AS EVPL service is functioning properly and can be successfully managed with CFM.

```
R8#ping ethernet mpid 2 domain C service icc OPTC 123456789
Type escape sequence to abort.
Sending 5 Ethernet CFM loopback messages to 001e.1415.dbbf, timeout is 5
seconds:!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 6/18/48 ms
```

8.4.3.3 MVPN - GRE (Profile 0)

GRE-based MVPN complexity will vary based on how the remote PE loopbacks are leaked between ASes. Earlier, we saw two main methods: redistribute from BGP into IGP, or leave the loopbacks in BGP by running it everywhere and allocating labels for them. The first method is simpler and requires less configuration, but it exposes remote PE loopbacks to core routers. The second method requires more configuration and an additional MPLS label, but shields P router from remote PE loopbacks. AS 24 uses the IGP redistribution method while AS 13 uses BGP labeled-unicast.

We will use MVPN profile 0 to provide MVPN service to the EIGRP VPN. We begin by defining a default MDT for this VPN both IPv4 and IPv6 AFIs. This is relevant on all PEs, and like option B, the default MDT

group must match between ASes. We used SSM for simplicity for option B, so I will use ASM for option C. Each AS has its own independent RP.

```
! XRv2 and XRv4
multicast-routing
vrf EIGRP
  address-family ipv4
    mdt default ipv4 225.13.24.255
  address-family ipv6
    mdt default ipv4 225.13.24.255

! CSR2
vrf definition EIGRP
  address-family ipv4
    mdt default 225.13.24.255
  address-family ipv6
    mdt default 225.13.24.255
```

Before we configure the inter-AS MDT, we can ensure the intra-AS MDT has formed within AS 24. This means that the default MDT is functional within that AS. We can check the P(S,G) entries on each router to ensure the remote PE was discovered. Each of them have joined one another's SPT (XR has "SPT" flag and XE has 'T' flag). The XE 'Z' flag in XE indicates this is a multicast tunnel as well.

```
RP/0/0/CPU0:XRv4#show pim topology 225.13.24.255 24.0.0.2 | begin 2,2
(24.0.0.2,225.13.24.255) SPT SM Up: 00:02:32
JP: Join(00:00:14) RPF: GigabitEthernet0/0/0/0.524,24.2.14.2 Flags:
KAT(00:00:58) RA
  No interfaces in immediate olist
```

```
R2#show ip mroute 225.13.24.255 24.0.0.14 | begin \
(24.0.0.14, 225.13.24.255), 00:03:04/00:01:58, flags: TZ
  Incoming interface: GigabitEthernet2.524, RPF nbr 24.2.14.14
  Outgoing interface list:
    MVRF EIGRP, Forward/Sparse, 00:02:50/00:00:09
```

A final verification includes ensuring PIM neighbors have formed bidirectionally within the EIGRP VPN over the default MDT.

```
R2#show ip pim vrf EIGRP neighbor | begin ^Neigh
Neighbor      Interface      Uptime/Expires    Ver    DR
Address
10.1.2.1      GigabitEthernet2.512    02:07:17/00:01:15 v2      1 / S P G
24.0.0.14     Tunnel5           00:01:04/00:01:42 v2      1 / DR G
```

```
RP/0/0/CPU0:XRv4#show pim vrf EIGRP neighbor | begin ^Neigh
Neighbor Address    Interface      Uptime    Expires    DR pri    Flags
10.13.14.13      GigabitEthernet0/0/0/0.534 4d01h      00:01:15  1        B P
```

```

10.13.14.14*      GigabitEthernet0/0/0/0.534 4d15h      00:01:36 1 (DR) B E
24.0.0.2          mdtEIGRP                      00:01:19 00:01:28 1      P
24.0.0.14*        mdtEIGRP                      00:01:38 00:01:28 1 (DR)

```

Since we are not using SSM (and have not configured BGP IPv4 MDT or IPv4 MVPN AFI), the inter-AS MDT cannot form yet. XRv2 and CSR2 are disparate RPs with no means to exchange information about sources to one another. To merge the two PIM ASM domains, we will configure a basic MSDP session between CSR2 and XRv2 so that this information can be exchanged. For security, I apply outbound filters on each router so that only P(S,G) state sourced from local loopbacks destined for the specific default MDT group will be permitted. The maximum number of SAs (in this case, remote PEs) is set to 5 for additional security. This is an odd solution for an inter-AS MPLS VPN design but it perfectly valid.

```

! CSR2
ip msdp peer 13.0.0.12 connect-source Loopback0 remote-as 13
ip msdp sa-filter out 13.0.0.12 list ACL_MSDP_FILTER
ip msdp sa-limit 13.0.0.12 5
ip access-list extended ACL_MSDP_FILTER
 permit ip 24.0.0.0 0.0.0.15 host 225.13.24.255

```

```

! XRv2
ipv4 access-list ACL_MSDP_FILTER
 10 permit ipv4 13.0.0.0 0.0.0.15 host 225.13.24.255

```

```

router msdp
 peer 24.0.0.2
   connect-source Loopback0
   remote-as 24
   sa-filter out list ACL_MSDP_FILTER
 maximum external-sa 5

```

We confirm that the MSDP session comes up on both sides. We can also see that XRv2 received 2 SAs from CSR2 while CSR2 received 1 SA from XRv2. This makes sense as the number of external SAs sent by a router should match the number of local PEs in the AS.

```

RP/0/0/CPU0:XRv2#show msdp summary
Out of Resource Handling Enabled
Maximum External SA's Global : 20000
Current External Active SAs : 2

```

MSDP Peer Status Summary

Peer Address	AS	State	Uptime/ Downtime	Reset Count	Peer Name	Active SA Cnt	Cfg.Max Ext.SAs	TLV recv/sent
24.0.0.2	24	Up	00:00:17	0	?	2	5	2/2

```

R2#show ip msdp summary

```

MSDP Peer Status Summary

Peer Address	AS	State	Uptime/	Reset SA	Peer Name

			Downtime	Count	Count
13.0.0.12	13	Up	00:00:29	0	1 ?

We can confirm that the proper SAs were exchanged by checking the SA caches on each router. XE only shows the learned SAs, not the locally-originated ones. XR shows both; it also specifies with the “PI” flag that PIM is interested. This means there is some kind of P(*,G) or P(S,G) state that exists for this group that would make the router “care” about it.

```
R2#show ip msdp sa-cache 225.13.24.255
MSDP Source-Active Cache - 1 entries for 225.13.24.255
(13.0.0.12, 225.13.24.255), RP 13.0.0.12, BGP/AS 0, 00:02:23/00:05:27, Peer
13.0.0.12
```

```
RP/0/0/CPU0:XRv2#show msdp sa-cache 225.13.24.255
MSDP Flags:
E - set MRIB E flag , L - domain local source is active,
EA - externally active source, PI - PIM is interested in the group,
DE - SAs have been denied. Timers age/expiration,
Cache Entry:
(13.0.0.12, 225.13.24.255), RP 13.0.0.12, MBGP/AS 0, 00:02:54/local
  Learned from peer local, RPF peer local
  SAs recvd 0, Encapsulated data received: 0
  grp flags: PI, src flags: L
(24.0.0.2, 225.13.24.255), RP 24.0.0.2, MBGP/AS 24, 00:02:49/00:02:27
  Learned from peer 24.0.0.2, RPF peer 24.0.0.2
  SAs recvd 4, Encapsulated data received: 0
  grp flags: PI, src flags: E, EA, PI
(24.0.0.14, 225.13.24.255), RP 24.0.0.2, MBGP/AS 24, 00:02:49/00:02:27
  Learned from peer 24.0.0.2, RPF peer 24.0.0.2
  SAs recvd 4, Encapsulated data received: 0
  grp flags: PI, src flags: E, EA, PI
```

We verify the MRIB on CSR2. The existing intra-AS entries for the default MSDP now have the ‘A’ flag on them. This signifies they are candidates for MSDP advertisement as SAs when sources as discovered. The ‘M’ flag indicates that the source for this P(S,G) was installed via an MSDP SA, which is correct for the source of 13.0.0.12.

```
R2#show ip mroute 225.13.24.255 | begin \ (13
(13.0.0.12, 225.13.24.255), 00:06:44/00:02:06, flags: MTZ
  Incoming interface: GigabitEthernet2.524, RPF nbr 24.2.14.14
  Outgoing interface list:
    MVRF EIGRP, Forward/Sparse, 00:06:44/00:02:15
(24.0.0.2, 225.13.24.255), 00:19:48/00:01:38, flags: TA
  Incoming interface: Loopback0, RPF nbr 0.0.0.0
  Outgoing interface list:
    GigabitEthernet2.524, Forward/Sparse, 00:19:48/00:03:18
```

```
(24.0.0.14, 225.13.24.255), 00:20:01/00:02:44, flags: TAZ
Incoming interface: GigabitEthernet2.524, RPF nbr 24.2.14.14
Outgoing interface list:
MVRF EIGRP, Forward/Sparse, 00:19:48/00:01:11
```

XRv2 shows similar output. The 'E' flag indicates that these were installed by MSDP external SAs. It is equivalent to the 'M' flag in XE. Traffic is consumed locally so there are no OIL entries.

```
RP/0/0/CPU0:XRv2#show pim topology 225.13.24.255 | begin 0.2,2
(24.0.0.2,225.13.24.255)SPT SM Up: 00:09:47
JP: Join(00:00:02) RPF: GigabitEthernet0/0/0/0.582,13.8.12.8 Flags:
KAT(00:01:26) E RA
No interfaces in immediate olist

(24.0.0.14,225.13.24.255)SPT SM Up: 00:09:47
JP: Join(00:00:02) RPF: GigabitEthernet0/0/0/0.582,13.8.12.8 Flags:
KAT(00:01:18) E RA
No interfaces in immediate olist
```

Last, we verify that all 3 routers have formed PIM neighbors over the default MDT (emulated LAN). Each router should have 2 other neighbors. This confirms that the default MDT has formed correctly.

```
RP/0/0/CPU0:XRv2#show pim vrf EIGRP neighbor | begin ^Neigh
Neighbor Address    Interface          Uptime    Expires    DR pri    Flags
10.3.12.3           GigabitEthernet0/0/0/0.532 21:16:10  00:01:19  1         P
10.3.12.12*         GigabitEthernet0/0/0/0.532 21:16:15  00:01:24  1 (DR)    B P E
13.0.0.12*          mdteEIGRP          00:23:57  00:01:41  1         P
24.0.0.2             mdteEIGRP          00:11:10  00:01:28  1         P
24.0.0.14           mdteEIGRP          00:11:10  00:01:23  1 (DR)
```

```
RP/0/0/CPU0:XRv4#show pim vrf EIGRP neighbor | begin ^Neigh
Neighbor Address    Interface          Uptime    Expires    DR pri    Flags
10.13.14.13         GigabitEthernet0/0/0/0.534 4d01h     00:01:40  1         B P
10.13.14.14*        GigabitEthernet0/0/0/0.534 4d16h     00:01:31  1 (DR)    B E
13.0.0.12           mdteEIGRP          00:11:41  00:01:44  1         P
24.0.0.2             mdteEIGRP          00:24:59  00:01:30  1         P
24.0.0.14*          mdteEIGRP          00:25:18  00:01:25  1 (DR)
```

```
R2#show ip pim vrf EIGRP neighbor | begin ^Neigh
Neighbor            Interface          Uptime/Expires    Ver    DR
Address                                     Prio/Mode
10.1.2.1            GigabitEthernet2.512 02:31:23/00:01:16 v2      1 / S P G
13.0.0.12           Tunnel5           00:11:45/00:01:31 v2      1 / P G
24.0.0.14           Tunnel5           00:25:10/00:01:43 v2      1 / DR G
```


The configuration in AS 24 is complete. Since all routers know how to reach 13.0.0.12, the remote PE, there is no need for a PIM vector. That is to say, CSR2 and XRv2 don't need to inform any core routers (if there were some) about how to perform RPF for a P(S,G) join where S is 13.0.0.12. With option B, this was always necessary, since the core routers would never know this information. By chance, the configuration in AS 13 is also complete. Given the small topology, there are no dedicated P routers in AS 13, so the PIM vector is technically not needed in this very specific network. The default MDT would not even come up if the core routers in AS 13 did not have BGP routers towards the remote PE loopbacks in AS 24. The PEs could originate the PIM vector (not with RD) to solve this problem, which XR does support. The alternative would be to extend BGP labeled-unicast to any possible core routers, ruining the BGP-free core.

Since we have not yet tested any data MDTs, I configure these on XRv2 only. Since CSR3 is our multicast source, it is possible that not all remote PEs in AS 24 are interested in every flow generated by CSR3. Since XRv3's RPF interface towards CSR3 is fixed towards CSR1, we suspect that XRv4 will not join the data MDT once the traffic starts flowing. I use SSM for these groups so there is no involvement with the RPs or MSDP session. This design can be useful for automatic PE discovery without BGP modification, while still utilizing SSM for the high-bandwidth flows via the data MDTs.

```
! XRv2
multicast-routing
vrf EIGRP
  address-family ipv4
    mdt data 232.13.24.0/26 immediate-switch
  address-family ipv6
    mdt data 232.13.24.64/26 immediate-switch
```

We will initiate an IPv4 multicast flow from CSR3 to XRv3's local group of 225.13.13.13. this is the same test flow we used for the other inter-AS MVPN options for consistency.

```
R3#ping ip
Target IP address: 225.13.13.13
Repeat count [1]: 100000
Datagram size [100]:
Timeout in seconds [2]: 1
Extended commands [n]: y
Interface [All]: loopback0
Time to live [255]:
Source address or interface: loopback0
```

I do not discuss the customer PIM registration or SPT switchover process for brevity. XRv3 does eventually join the SPT, but what triggers the data MDT is the threshold (or "immediate-switch" in this case) configured on the ingress PE, which is XRv2. XRv2 sends a PIM TLV over the default MDT to other PEs that may want to join the data MDT. This "MDT join" allows remote PEs to issue P(S,G) joins towards

the ingress PE for the new data MDT. We can see that CSR2 receives this MDT message. The P(S,G) information is shown in yellow and the C(S,G) information is shown in green.

```
RP/0/0/CPU0:XRv2#show pim vrf EIGRP mdt cache
Core Source      Cust (Source, Group)      Core Data      Expires
13.0.0.12        (10.3.3.3, 225.13.13.13)  232.13.24.0    00:02:41
```

```
R2#show ip pim mdt receive detail | begin ^Join
Joined MDT-data [group/mdt number : source]  uptime/expires for VRF: EIGRP
[232.13.24.0 : 13.0.0.12]  00:09:04/00:02:57
(10.3.3.3, 225.13.13.13), 00:09:04/00:02:53/00:02:57, OIF count: 1, flags: TY
```

CSR2 issues the P(S,G) join towards XRv4 for this new group. I won't trace the entire RPF path as it is very basic. We know RPF is working otherwise the default MDT would have never formed; whether the MDT is default or data, the P-sources are the same. XRv2 is the source PE for this new data MDT and is the root of the tree.

```
R2#show ip mroute 232.13.24.0 13.0.0.12 | begin \
(13.0.0.12, 232.13.24.0), 00:10:49/stopped, flags: sTIZ
Incoming interface: GigabitEthernet2.524, RPF nbr 24.2.14.14
Outgoing interface list:
MVRFP EIGRP, Forward/Sparse, 00:10:49/00:01:10
```

```
RP/0/0/CPU0:XRv2#show pim topology 232.13.24.0 13.0.0.12 | begin 232
(13.0.0.12,232.13.24.0)SPT SSM Up: 00:10:06
JP: Join(never) RPF: Loopback0,13.0.0.12* Flags:
Loopback0 00:10:06 fwd LI LH
GigabitEthernet0/0/0/0.582 00:10:06 fwd Join(00:03:14)
```

Checking the C(S,G) entries, we see that CSR2's entry is mapped to the specific data MDT that was exchanged in the PIM TLV. The big 'Y' flag indicates reception of multicast traffic along a data MDT. On XRv2, the "MA" flag indicates a data MDT was assigned to this C(S,G) and the "MT" flag indicates the data MDT threshold was crossed. The only time (conceivably) you would have "MT" without "MA" is if the router exhausted its supply of data MDT groups.

```
R2#show ip mroute vrf EIGRP 225.13.13.13 10.3.3.3 | begin \
(10.3.3.3, 225.13.13.13), 00:12:24/00:01:27, flags: TY
Incoming interface: Tunnel5, RPF nbr 13.0.0.12,
MDT: [13.0.0.12,232.13.24.0]/00:02:37
Outgoing interface list:
GigabitEthernet2.512, Forward/Sparse, 00:12:24/00:03:27
```

```
RP/0/0/CPU0:XRv2#show pim vrf EIGRP topology 225.13.13.13 | begin 3,225
(10.3.3.3,225.13.13.13)SPT SM Up: 00:13:41
JP: Join(00:00:08) RPF: GigabitEthernet0/0/0/0.532,10.3.12.3 Flags: MT MA
mdtEIGRP 00:13:41 fwd Join(00:02:37)
```

As an additional check, we examine XRv3's MFIB counters. We can see many packets entering the router SW process which indicates success.

```
RP/0/0/CPU0:XRv3#show mfib route 225.13.13.13 10.3.3.3 | begin 225
(10.3.3.3,225.13.13.13),    Flags:
  Up: 00:16:09
  Last Used: 00:00:00
  SW Forwarding Counts: 968/968/96800
  SW Replication Counts: 968/0/0
  SW Failure Counts: 0/0/0/0/0
  Loopback0 Flags:  IC NS EG, Up:00:16:09
  GigabitEthernet0/0/0/0.513 Flags:  A, Up:00:16:09
```

To prove that CSR2 is actually in the transit path while XRv4 is not (technically XRv4 is a P router for this flow in the carrier network, but that is not the point), we can look at the egress PEs. CSR2 shows about the same number of packets as XRv3 (slightly more since I issued this command later) and XRv4 does not even have the C(S,G) entry. This is expected behavior given the customer RPF topology.

```
R2#show ip mroute vrf EIGRP 225.13.13.13 10.3.3.3 count | begin ^Group
Group: 225.13.13.13, Source count: 1, Packets forwarded: 1046, Packets
received: 1046
  Source: 10.3.3.3/32, Forwarding: 1045/1/142/1, Other: 1045/0/0
```

```
RP/0/0/CPU0:XRv4#show pim vrf EIGRP topology 225.13.13.13
No PIM topology table entries found.
```

8.4.3.4 MVPN – mLDP (Profile 17)

Because the remote loopbacks are leaked between ASes, building mLDP across AS boundaries should be straightforward. Before beginning, note that **XE does not support mLDP recursive FEC**. Like the PIM vector, core routers would not know how to reach remote PE loopback if BGP is used to distribute them within an AS. As such, they cannot send label mapping messages towards the mLDP roots. Just like with GRE-based MVPN, we are fortunate that AS 13 does not have any real P routers, or else this test would fail. Recursive FEC allows a router to effectively accomplish the same task the PIM vector does (update the FEC to something the core routers can reach) so that core routers are not left searching for missing remote PE routes. The XR command to enable this feature is below and is shown for demonstration purposes only. There is no XE equivalent at this time.

```
! Recursive mLDP FEC in XR
mpls ldp
  mldp
    address-family ipv4
      recursive-fec
```

Recursive FEC is less of a design/architecture and is actually an opaque type. On XR, we can see this new opaque type is an option for the mLDP database show command. The two types of recursive FEC are

basic and VPN. Just like the PIM vector, we can include the vector by itself or vector + RD. The logic is the same here, where the “recursive-rd” option includes a “root” address reachable from the core routers along with the VPN’s RD. Again, XE has no equivalent at this time, so we will not examine this in any great detail.

```
RP/0/0/CPU0:XRv1#show mpls mldp database opaquetype ?
  global-id      4 byte global LSP ID encoding
  ipv4           IPv4 opaque encoding
  ipv6           IPv6 opaque encoding
  mdt            RPF2685 VPN ID + MDT NR encoding
  recursive      Recursive opaque encoding
  recursive-rd   Recursive RD opaque encoding
  static-id      4 byte static LSP ID encoding
  vpnv4          VPNv4 opaque encoding
  vpnv6          VPNv6 opaque encoding
```

Ignoring this limitation since it doesn’t affect our particular network, we begin the basic mLDP configurations. Like the GRE-based MVPN where the default MDT group had to match between ASes, the same is true for the mLDP VPN ID. We will demonstrate this using the OSPF VPN on CSR2 and CSR8. Both of them have identical VRF configurations. As seen in the option B mLDP attempt, we should use the “inter-as” keyword to prevent BGP from automatically adding the “no-export” community to the type-1 I-PMSI routes. We could strip the community in other ways, but this is the most elegant.

```
! CSR2 and CSR8
vrf definition OSPF
  vpn id 1300:2400
  address-family ipv4
    mdt preference mldp
    mdt auto-discovery mldp inter-as
    mdt default mpls mldp p2mp
  address-family ipv6
    mdt preference mldp
    mdt auto-discovery mldp inter-as
    mdt default mpls mldp p2mp
```

Next, we must configure BGP IPv4 and IPv6 MVPN AFI. This configuration is basic on CSR2, CSR8, and XRv2. We must be careful to keep the next-hop unchanged between ASes in accordance with the option C design. Also, since this AFI uses RTs just like VPNv4/v6 and VPLS, we need to consider RT rewrite and intra-AS distribution. On XRv2, I demonstrate both retaining all RTs for one AFI and making CSR8 an RR-client for the other. This will allow XRv2 to retain all MVPN routes from CSR2 for advertisement down to CSR8. Additionally, both CSR2 and CSR8 must rewrite their RTs before advertising these routes. The RTs for the locally-originated MVPN routes match what is exported by the VRF, so the rewrites should be the same. Fortunately, we can reuse the RT rewrite policies on both CSR2 and XRv2 for this.

```
! CSR2
```

```

router bgp 24
  address-family ipv4 mvpn
    neighbor 13.0.0.12 activate
    neighbor 13.0.0.12 next-hop-unchanged
    neighbor 13.0.0.12 route-map RM_RT_REWRITE out
  address-family ipv6 mvpn
    neighbor 13.0.0.12 activate
    neighbor 13.0.0.12 next-hop-unchanged
    neighbor 13.0.0.12 route-map RM_RT_REWRITE out

! XRv2
router bgp 13
  address-family ipv4 mvpn
    retain route-target all

  address-family ipv6 mvpn

  neighbor 13.0.0.8
    address-family ipv4 mvpn
    address-family ipv6 mvpn
    route-reflector-client

  neighbor 24.0.0.2
    address-family ipv4 mvpn
    route-policy RPL_PASS in
    route-policy RPL_RT_REWRITE out
    next-hop-unchanged
  address-family ipv6 mvpn
    route-policy RPL_PASS in
    route-policy RPL_RT_REWRITE out
    next-hop-unchanged

! CSR8
router bgp 13
  neighbor 13.0.0.12 activate
  address-family ipv6 mvpn
    neighbor 13.0.0.12 activate

```

We quickly check CSR2 and XRv2 to make sure the BGP session has formed and routes are being exchanged. The reason CSR2 learns 2 routes in each AFI is because CSR8 generated one for RD 13:2 (OSPF) and one for RD 13:1 (BGP central services).

```

R2#show bgp ipv4 mvpn all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State/PfxRcd
13.0.0.12     4       13     36     44      10    0   0 00:05:06      2

R2#show bgp ipv6 mvpn all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State/PfxRcd
13.0.0.12     4       13     36     44      10    0   0 00:05:09      2

```

```
RP/0/0/CPU0:XRv2#show bgp ipv4 mvpn summary | begin ^Neigh
Neighbor      Spk    AS  MsgRcvd  MsgSent    TblVer  InQ  OutQ  Up/Down  St/PfxRcd
13.0.0.8       0    13    9229    8918        5     0     0 00:05:30      2
24.0.0.2       0    24    2261    1724        5     0     0 00:06:11      1
```

```
RP/0/0/CPU0:XRv2#show bgp ipv6 mvpn summary | begin ^Neigh
Neighbor      Spk    AS  MsgRcvd  MsgSent    TblVer  InQ  OutQ  Up/Down  St/PfxRcd
13.0.0.8       0    13    9229    8918        4     0     0 00:05:33      2
24.0.0.2       0    24    2261    1724        4     0     0 00:06:14      1
```

We will examine the MVPN route details to ensure the RT rewrite was correct. CSR2 received routes from XRv2 with RT:24:2 and XRv2 receive routes from CSR2 with RT:13:2. This is the correct RT rewrite procedure as it will enable the iBGP clients in each AS to import the MVPN routes. Other potential extended-communities should not be wiped out thanks to the “additive” keyword. We can be fancy on CSR2 by referencing the extended-community list that matches RT:24:2 to filter the output. We don’t have this capability on XR, so we will check the route details for the RT.

```
R2#show bgp ipv4 mvpn rd 24:2 extcommunity-list EXTCOML_RT_24_2 | begin
Network
```

```
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 24:2 (default for vrf OSPF)
*    [1] [24:2] [13.0.0.8]/12
      13.0.0.8              0 13 ?
*>      13.0.0.8              0 13 ?
*>    [1] [24:2] [24.0.0.2]/12
      0.0.0.0              32768 ?
```

```
RP/0/0/CPU0:XRv2#show bgp ipv4 mvpn rd 24:2 route-type intra-as-ipmsi-ad |
begin Network
```

```
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 24:2
*>    [1] [24.0.0.2]/40    24.0.0.2              0              0 24 ?
```

```
RP/0/0/CPU0:XRv2#show bgp ipv4 mvpn rd 24:2 [1] [24.0.0.2]/40 | include RT:
Extended community: RT:13:2
```

CSR2 successfully imports CSR8’s route into VRF OSPF, and vice versa. Again, the reason CSR2 sees 2 remote routes (which are identical) is because CSR8 locally imports routes from the central services VPN. Below are the detailed outputs for the IPv4 MVPN routes. Tunnel type 2 is mLDP P2MP and the root addresses in hex are highlighted in yellow as well.

```
R2#show bgp ipv4 mvpn vrf OSPF route-type 1 13.0.0.8
BGP routing table entry for [1] [24:2] [13.0.0.8]/12, version 9
Paths: (2 available, best #2, table MVPNv4-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 1
  13, imported path from [1] [13:2] [13.0.0.8]/12 (global)
```

```

13.0.0.8 (metric 25) from 13.0.0.12 (13.0.0.12)
  Origin incomplete, localpref 100, valid, external
  Extended Community: RT:24:2
  PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 17, label: exp-null,
tunnel parameters: 0600 0104 0D00 0008 0007 0100 0400 0200 00
  rx pathid: 0, tx pathid: 0
Refresh Epoch 1
13, imported path from [1][13:1][13.0.0.8]/12 (global)
13.0.0.8 (metric 25) from 13.0.0.12 (13.0.0.12)
  Origin incomplete, localpref 100, valid, external, best
  Extended Community: RT:24:2
  PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 17, label: exp-null,
tunnel parameters: 0600 0104 0D00 0008 0007 0100 0400 0200 00
  rx pathid: 0, tx pathid: 0x0

```

```

R8#show bgp ipv4 mvpn vrf OSPF route-type 1 24.0.0.2
BGP routing table entry for [1][13:2][24.0.0.2]/12, version 10
Paths: (1 available, best #1, table MVPNV4-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 1
  24, imported path from [1][24:2][24.0.0.2]/12 (global)
  24.0.0.2 (metric 20) from 13.0.0.12 (13.0.0.12)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    Extended Community: RT:13:2
    PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 17, label: exp-null,
tunnel parameters: 0600 0104 1800 0002 0007 0100 0400 0200 00
  rx pathid: 0, tx pathid: 0x0

```

For brevity, I show the IPv6 MVPN routes inside the VRF. If they are shown in the VRF table, then the RT must have been imported properly, and verifying it further is unnecessary.

```

R2#show bgp ipv6 mvpn vrf OSPF | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 24:2 (default for vrf OSPF)
*      [1][24:2][13.0.0.8]/12
              13.0.0.8              0 13 ?
*>
              13.0.0.8              0 13 ?
*> [1][24:2][24.0.0.2]/12
              ::              32768 ?

R8#show bgp ipv6 mvpn vrf OSPF | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 13:2 (default for vrf OSPF)
*> [1][13:2][13.0.0.8]/12
              ::              32768 ?
*>i [1][13:2][24.0.0.2]/12
              24.0.0.2              0    100    0 24 ?

```

Importing these BGP auto-discovery MVPN routes into the OSPF VPN means that the mLDP tree construction can begin. I will quickly trace CSR2's IPv4 P2MP tree from CSR8's perspective to see if it was successfully able to join. The MVPN section describes this in greater detail, but GIDs beginning with 0x20000 are used for IPv4 trees while 0x40000 implies IPv6 trees. LSM ID 4 is the tree we will trace as this is rooted at CSR2.

```
R8#show mpls mldp database summary
```

LSM ID	Type	Root	Decoded Opaque Value	Client Cnt.
1	P2MP	13.0.0.8	[gid 131072 (0x00020000)]	1
4	P2MP	24.0.0.2	[gid 131072 (0x00020000)]	1
2	P2MP	13.0.0.8	[gid 262144 (0x00040000)]	1
3	P2MP	24.0.0.2	[gid 262144 (0x00040000)]	1

We check the mLDP database for details about LSM ID 4. We can see the path towards 24.0.0.2 is via XRv1, which is consistent with the unicast route recursion on CSR8. CSR8 allocates downstream label 8034 to XRv1 who can use it when sending traffic down the P2MP tree from CSR2 for IPv4 multicast. When CSR8 receives it, the traffic is replicated into the OSPF VPN as expected.

```
R8#show mpls mldp database id 4
```

```
LSM ID : 4   Type: P2MP   Uptime : 00:08:55
FEC Root      : 24.0.0.2
Opaque decoded : [gid 131072 (0x00020000)]
Opaque length  : 4 bytes
Opaque value   : 01 0004 00020000
Upstream client(s) :
  13.0.0.11:0    [Active]
    Expires      : Never
    Out Label (U) : None
    Local Label (D) : 8034
    Next Hop     : 13.8.11.11
Replication client(s):
  MDT (VRF OSPF)
    Uptime       : 00:08:55
    Interface    : Lspvif0
    Path Set ID  : None
```

```
R8#show ip cef 24.0.0.2 detail
```

```
24.0.0.2/32, epoch 2, flags [rib defined all labels]
  3 RR sources [non-eos indirection, heavily shared]
    recursive via 10.6.11.6 label 6029
    nexthop 13.8.11.11 GigabitEthernet2.581 label 91008
```

When we check XRv1, we only see two trees versus the expected 4. XRv1 knows about CSR2 as a root, which implies there are clients in AS 13 that are trying to join it. Specifically, this is CSR8, and is correct. XRv1 is not aware that CSR8 is the root for any trees because it has not received any label mapping messages from CSR6. Even without looking at more detailed output, we theorize there is a problem on the transit link. LSM ID 0x2 describes the tree we are currently tracing.


```
RP/0/0/CPU0:XRv1#show mpls mldp database brief
LSM ID    Type    Root                Up Down Decoded Opaque Value
0x00002   P2MP    24.0.0.2            0  1    [global-id 131072]
0x00001   P2MP    24.0.0.2            0  1    [global-id 262144]
```

Looking at LSM ID 0x2, we can see there are no upstream neighbors. XRv1 is not the root of the tree, so we would not expect to see this output quite yet. The issue is that LDP is not enabled on the transit links, which means mLDP label mapping messages cannot be exchanged. XRv1 did correctly process the downstream label value of 8034 from CSR8.

```
RP/0/0/CPU0:XRv1#show mpls mldp database 0x2
mLDP database
LSM-ID: 0x00002    Type: P2MP    Uptime: 00:12:42
  FEC Root          : 24.0.0.2
  Opaque decoded     : [global-id 131072]
  Upstream neighbor(s) :
    None
  Downstream client(s):
    LDP 13.0.0.8:0    Uptime: 00:12:42
      Next Hop        : 13.8.11.8
      Interface       : GigabitEthernet0/0/0/0.581
      Remote label (D) : 8034
```

Before tracing anything else, I enable LDP on all of the transit interfaces. Just like with option B, it makes sense for us to adjust the LDP transport address for the TCP session to be the connected interface. There was no reason for us to advertise the ASBR loopbacks into BGP, so the LDP RIDs won't have connectivity across AS boundaries.

```
! CSR6
interface GigabitEthernet2.556
 mpls ip
 mpls ldp discovery transport-address interface

interface GigabitEthernet2.561
 mpls ip
 mpls ldp discovery transport-address interface

! CSR7
interface GigabitEthernet2.557
 mpls ip
 mpls ldp discovery transport-address interface

! CSR5
interface GigabitEthernet2.556
 mpls ip
 mpls ldp discovery transport-address interface
```

```

interface GigabitEthernet2.557
  mpls ip
  mpls ldp discovery transport-address interface

! XRv1
mpls ldp
  interface GigabitEthernet0/0/0/0.561
  address-family ipv4
  discovery transport-address interface

```

I quickly check CSR6 and CSR5 to ensure all of the LDP peers form. Notice the TCP session addressing for the inter-AS sessions below. This shows that all of the LDP sessions have formed.

```

R5#show mpls ldp neighbor | include Peer|TCP
  Peer LDP Ident: 13.0.0.11:0; Local LDP Ident 13.0.0.5:0
    TCP connection: 13.0.0.11.44214 - 13.0.0.5.646
  Peer LDP Ident: 13.0.0.8:0; Local LDP Ident 13.0.0.5:0
    TCP connection: 13.0.0.8.32139 - 13.0.0.5.646
  Peer LDP Ident: 24.0.0.7:0; Local LDP Ident 13.0.0.5:0
    TCP connection: 10.5.7.7.27954 - 10.5.7.5.646
  Peer LDP Ident: 24.0.0.6:0; Local LDP Ident 13.0.0.5:0
    TCP connection: 10.5.6.6.28926 - 10.5.6.5.646

```

```

R6#show mpls ldp neighbor | include Peer|TCP
  Peer LDP Ident: 24.0.0.7:0; Local LDP Ident 24.0.0.6:0
    TCP connection: 24.0.0.7.28768 - 24.0.0.6.646
  Peer LDP Ident: 24.0.0.14:0; Local LDP Ident 24.0.0.6:0
    TCP connection: 24.0.0.14.55519 - 24.0.0.6.646
  Peer LDP Ident: 13.0.0.11:0; Local LDP Ident 24.0.0.6:0
    TCP connection: 10.6.11.11.39776 - 10.6.11.6.646
  Peer LDP Ident: 13.0.0.5:0; Local LDP Ident 24.0.0.6:0
    TCP connection: 10.5.6.5.646 - 10.5.6.6.28926

```

As seen in option B, one issue with this approach is that labels are still being exchanged even if the peer AS does not have routes for it. Alternatively, a peer AS may have a BGP route for a prefix, making the LDP label useless. In the option B section, we developed simple label filters that only allow labels to be advertised to internal peers. On XRv1, the logic is reversed and the router allocates labels for no prefixes to a specific peer. Thus, this halts the label exchange over the inter-AS links. This only affects unicast LDP labels; it has no effect on mLDP labels.

```

! CSR6 and CSR7
ip access-list standard ACL_ANY
  permit any
ip access-list standard ACL_INTERNAL_PEERS
  permit 24.0.0.0 0.0.0.255

no mpls ldp advertise-labels

```

```

mpls ldp advertise-labels for ACL_ANY to ACL_INTERNAL_PEERS

! CSR5
ip access-list standard ACL_ANY
 permit any
ip access-list standard ACL_INTERNAL_PEERS
 permit 13.0.0.0 0.0.0.255

no mpls ldp advertise-labels
mpls ldp advertise-labels for ACL_ANY to ACL_INTERNAL_PEERS

! XRv1
ipv4 access-list ACL_DENY_ALL
 10 deny ipv4 any any

mpls ldp
 address-family ipv4
  label
  local
  advertise
    to 24.0.0.6:0 for ACL_DENY_ALL

```

For brevity, I limit my verifications to CSR6. We can see that no remote labels are learned from any AS 13 peer. The LDP session is entirely incapable of advertising IPv4 prefix to label bindings, which is the goal.

```

R6#show mpls ldp bindings neighbor 13.0.0.11
[no output]

```

```

R6#show mpls ldp bindings neighbor 13.0.0.5
[no output]

```

We will continue tracing CSR2's P2MP IPv4 tree on XRv1. Earlier, XRv1 did not have an upstream interface as LDP was disabled on the transit link towards 24.0.0.2, which is via CSR6. Checking the same LSM ID, we see CSR6 as the upstream neighbor and its LDP RIB clearly displayed. XRv1 allocates label 91011 to CSR6 for receiving multicast from CSR2. Label 91011 is swapped for label 8034 as traffic flows downstream.

```

RP/0/0/CPU0:XRv1#show mpls mldp database 0x2
mLDP database
LSM-ID: 0x00002    Type: P2MP    Uptime: 00:30:44
  FEC Root          : 24.0.0.2
  Opaque decoded     : [global-id 131072]
  Upstream neighbor(s) :
    24.0.0.6:0 [Active] Uptime: 00:13:43
    Local Label (D) : 91011
  Downstream client(s):

```

```

LDP 13.0.0.8:0      Uptime: 00:30:44
  Next Hop          : 13.8.11.8
  Interface         : GigabitEthernet0/0/0/0.581
  Remote label (D)  : 8034

```

Continuing the trace, CSR6 allocates label 6064 towards XRv4 to receive multicast traffic from CSR2. It swaps this for label 91011 received from XRv1.

```
R6#show mpls mldp database summary
```

LSM ID	Type	Root	Decoded Opaque Value	Client Cnt.
2	P2MP	13.0.0.8	[gid 131072 (0x00020000)]	1
4	P2MP	24.0.0.2	[gid 131072 (0x00020000)]	1
1	P2MP	13.0.0.8	[gid 262144 (0x00040000)]	1
3	P2MP	24.0.0.2	[gid 262144 (0x00040000)]	1

```
R6#show mpls mldp database id 4
```

```

LSM ID : 4      Type: P2MP      Uptime : 00:15:05
  FEC Root      : 24.0.0.2
  Opaque decoded : [gid 131072 (0x00020000)]
  Opaque length  : 4 bytes
  Opaque value   : 01 0004 00020000
  Upstream client(s) :
    24.0.0.14:0 [Active]
      Expires      : Never          Path Set ID : 4
      Out Label (U) : None          Interface   : GigabitEthernet2.564*
      Local Label (D) : 6064        Next Hop    : 24.6.14.14
  Replication client(s):
    13.0.0.11:0
      Uptime       : 00:15:05      Path Set ID : None
      Out label (D) : 91011
      Local label (U) : None

```

XRv4 receives label 6064 from CSR6 and sends a label mapping message to CSR2, the root, with downstream label 94019. XRv4 swaps 94019 for 6064 when traffic flows down the tree.

```
RP/0/0/CPU0:XRv4#show mpls mldp database brief
```

LSM ID	Type	Root	Up	Down	Decoded Opaque Value
0x00005	P2MP	13.0.0.8	1	1	[global-id 131072]
0x00007	P2MP	24.0.0.2	1	1	[global-id 131072]
0x00004	P2MP	13.0.0.8	1	1	[global-id 262144]
0x00006	P2MP	24.0.0.2	1	1	[global-id 262144]

```
RP/0/0/CPU0:XRv4#show mpls mldp database 0x7
```

```
mLDP database
```

```

LSM-ID: 0x00007      Type: P2MP      Uptime: 00:21:21
  FEC Root           : 24.0.0.2
  Opaque decoded      : [global-id 131072]
  Upstream neighbor(s) :

```

```

24.0.0.2:0 [Active] Uptime: 00:21:21
  Local Label (D) : 94019
Downstream client(s):
  LDP 24.0.0.6:0      Uptime: 00:21:21
    Next Hop          : 24.6.14.6
    Interface          : GigabitEthernet0/0/0/0.564
  Remote label (D) : 6064

```

CSR2 is the root of the tree and has no upstream clients. It has two downstream clients: the VPN customer and XRv4. Label 94019 is pushed onto the label stack when CSR2 receives native IPv4 multicast from CSR9.

```
R2#show mpls mldp database summary
```

LSM ID	Type	Root	Decoded Opaque Value	Client Cnt.
4	P2MP	13.0.0.8	[gid 131072 (0x00020000)]	1
1	P2MP	24.0.0.2	[gid 131072 (0x00020000)]	2
3	P2MP	13.0.0.8	[gid 262144 (0x00040000)]	1
2	P2MP	24.0.0.2	[gid 262144 (0x00040000)]	2

```
R2#show mpls mldp database id 1
```

```

LSM ID : 1    Type: P2MP    Uptime : 00:51:01
  FEC Root      : 24.0.0.2 (we are the root)
  Opaque decoded : [gid 131072 (0x00020000)]
  Opaque length  : 4 bytes
  Opaque value   : 01 0004 00020000
  Upstream client(s) :
    None
    Expires      : N/A          Path Set ID : 1
Replication client(s):
  MDT (VRF OSPF)
    Uptime      : 00:51:01      Path Set ID : None
    Interface    : Lspvif0
  24.0.0.14:0
    Uptime      : 00:23:01      Path Set ID : None
    Out label (D) : 94019      Interface   : GigabitEthernet2.524*
    Local label (U): None      Next Hop    : 24.2.14.14

```

CSR2 should be able to send multicast to CSR8 now. However, it cannot.

```

R2#ping mpls mldp p2mp 24.0.0.2 hex 0x1 00020000
p2mp Root node addr 24.0.0.2
Opaque type hex value (0x1), num hex digits 4
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
  timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]

```

Type escape sequence to abort.

Request #1

.

Received 0 replies

We moved too quickly through our verification process and skipped a critical step. On CSR6, we saw an outgoing label to XRv1, but no outgoing interface. The LFIB clearly shows this LSM traffic being dropped.

```
R6#show mpls forwarding-table labels 6064
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
6064	91011	[gid 131072 (0x00020000)]	\	0	drop

I enable mLDP debugging on CSR6 and clear the mLDP neighbor with XRv11. Because I have not seen this problem before, I enable all debugging. Focusing on this particular tree, we see label 91011 from XRv1 as usual, except the interface is “NULL”. It also shows the LDP RID as the next-hop, which doesn’t make sense. CSR6 does not have a route for this ASBR loopback, nor should it need one. The TCP session between CSR6 and XRv1 is established using directly connected interfaces and CSR6 has a valid downstream label to XRv1. Historically, reachability to the LDP RID has never been necessary for ordinary IPv4 prefix to label bindings.

```
R6#debug mpls mldp all
MLDP generic debugging is on
MLDP MRIB debugging is on
MLDP MFI debugging is on
MLDP Gracefull Restart debugging is on
MLDP neighbor debugging is on
MLDP HA debugging is on
MLDP filtering debugging is on
MLDP Route-watch notification debugging is on
```

```
MLDP: Route watch started for 24.0.0.2 topology: base ipv4
MLDP-DB: Added [gid 131072 (0x00020000)] DB Entry
MLDP-DB: [gid 131072 (0x00020000)] Added P2MP branch for 13.0.0.11:0 label 91011
MLDP-MFI: PSM-ID: 5, Add Path type: PKT, Label: 91011, Nexthop: 13.0.0.11, Interface: NULL
```

Adding a temporary static host route to CSR6 with debugging still enabled, mLDP reveals that it is actually trying to find a route to the peer LDP RID.

```
! CSR6
ip route 13.0.0.11 255.255.255.255 10.6.11.11
```

```
MLDP-NBR: 13.0.0.11:0 LDP update NH: 10.6.11.11, Interface:
GigabitEthernet2.561 Add
MLDP-NBR: 13.0.0.11:0 Check branches for path change
MLDP-DB: [gid 131072 (0x00020000)] Changing branch 13.0.0.11:0 from
Null/0.0.0.0 to GigabitEthernet2.561/10.6.11.11
```

When we check CSR6's mLDP database, now the outgoing interface shows up as XRv1 as expected. However, the LFIB remains dysfunctional. Traffic arriving from XRv4 with label 6067 is still dropped.

```
R6#show mpls mldp database summary
LSM ID      Type      Root      Decoded Opaque Value      Client
Cnt.
2           P2MP      13.0.0.8  [gid 131072 (0x00020000)] 1
7           P2MP      24.0.0.2  [gid 131072 (0x00020000)] 1
1           P2MP      13.0.0.8  [gid 262144 (0x00040000)] 1
8           P2MP      24.0.0.2  [gid 262144 (0x00040000)] 1
```

```
R6#show mpls mldp database id 7
LSM ID : 7   Type: P2MP   Uptime : 00:00:22
FEC Root      : 24.0.0.2
Opaque decoded : [gid 131072 (0x00020000)]
Opaque length  : 4 bytes
Opaque value   : 01 0004 00020000
Upstream client(s) :
  24.0.0.14:0   [Active]
    Expires      : Never      Path Set ID : 7
    Out Label (U) : None      Interface   : GigabitEthernet2.564*
    Local Label (D) : 6067    Next Hop    : 24.6.14.14
Replication client(s):
  13.0.0.11:0
    Uptime       : 00:00:22   Path Set ID : None
    Out label (D) : 91011     Interface   : GigabitEthernet2.561*
    Local label (U) : None     Next Hop    : 10.6.11.11
```

```
R6#show mpls forwarding-table labels 6067
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id  Switched     interface
6067      91011      [gid 131072 (0x00020000)] \
                                         0      drop
```

The static route helped us understand what mLDP was really trying to do but didn't solve the forwarding problem. I remove the static route and advertise the loopback into BGP on XRv1 instead.

```
! CSR6
no ip route 13.0.0.11 255.255.255.255 10.6.11.11

! XRv1
router bgp 13
```

```
address-family ipv4 unicast
network 13.0.0.11/32
```

With debugging still enabled, we see a similar set of messages from mLDP on CSR6. The interface has been updated from “NULL” to the transit link facing XRv1. So far this is identical to what we saw with our static route.

```
! CSR6
MLDP-NBR: 13.0.0.11:0 LDP update NH: 10.6.11.11, Interface:
GigabitEthernet2.561 Add
MLDP-NBR: 13.0.0.11:0 Check branchchange
MLDP-DB: [gid 131072 (0x00020000)] Changing branch 13.0.0.11:0 from
Null/0.0.0.0 to GigabitEthernet2.561/10.6.11.11
```

However, this actually did solve the problem. The LFIB is correctly populated and LSM can be forwarded from CSR2 to CSR8. This solution isn’t terrible since there are usually a small number of ASBRs, but it does modify the original option C intent.

```
R6#show mpls forwarding-table labels 6067
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Outgoing interface	Next Hop
6067	91011	[gid 131072 (0x00020000)] \	80	Gi2.561	10.6.11.11

The reason the static route did not work is because despite having a route to 13.0.0.11/32, CSR6 did not learn a unicast label for it. This suggests that mLDP requires a unicast label (from any source) for the next-hop prefixes along the tree’s downstream path. I suppose this is a mechanism to ensure LSM is not forwarded to peers that may not be distributing labels properly. Had we not applied our LDP label filtering outbound on XRv1, the static route would have solved the problem. The BGP solution is cleaner so we will continue relying on that. For completeness, I advertise all of the ASBR loopbacks into BGP (not shown) since, presumably, CSR5 and CSR7 will have the same problem during a failover event. You could potentially use static routes at the ASBR’s and keep the loopback reachability local to the ASBRs across a transit link, but this would require label bindings for those prefixes to be exchanged. Adjusting the LDP filters appropriately is a valid alternative to using BGP labeled-unicast, but since this AFI is already negotiated for option C, adding static routes makes less sense. Additionally, we could stop using the transit interface IP addresses for the LDP TCP session now that the LDP RIDs are reachable, but I do not revert this modification.

For some reason, XR was smart enough not to use the LDP RID as the next-hop in identifying replication clients. I prove that the LSM is flowing using MPLS OAM. This may indicate an issue within XE or simply a sanity check as described earlier. This will discover all other remote PEs down the P2MP tree. Beware that when using OAM on P2MP trees, pings can only be originated at the tree’s root.

```
R2#ping mpls mldp p2mp 24.0.0.2 hex 0x1 00020000
```



```

p2mp Root node addr 24.0.0.2
Opaque type hex value (0x1), num hex digits 4
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
    timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]

Request #1
! reply addr 13.8.11.8

Round-trip min/avg/max = 59/59/59 ms
Received 1 reply

```

Rather than manually trace the other 3 trees, we will use MPLS OAM to test reachability. The summarized output is below. This shows that all 4 of the trees (IPv4 and IPv6 in both directions) are functional.

```

R2#ping mpls mldp p2mp 24.0.0.2 hex 0x1 00040000
[snip]
Request #1
! reply addr 13.8.11.8

R8#ping mpls mldp p2mp 13.0.0.8 hex 0x1 00020000
[snip]
Request #1
! reply addr 24.2.14.2

R8#ping mpls mldp p2mp 13.0.0.8 hex 0x1 00040000
[snip]
Request #1
! reply addr 24.2.14.2

```

Another fast test is to check both PEs for PIM neighbors for both IPv4 and IPv6. Those outputs are shown below as further confirmation that inter-AS option C mLDP is functional.

```

R2#show ip pim vrf OSPF neighbor | begin ^Neigh
Neighbor      Interface      Uptime/Expires    Ver    DR
Address
10.2.9.9      GigabitEthernet2.529    05:59:19/00:01:41 v2      1 / DR S P G
13.0.0.8      Lspvif0                 01:13:35/00:01:34 v2      1 / S P G

R2#show ipv6 pim vrf OSPF neighbor | begin ^Neigh
Neighbor Address      Interface      Uptime    Expires    Mode DR pri
::FFFF:13.0.0.8      Lspvif0       01:13:30  00:01:25  B G      1
FE80::9              Gi2.529       05:59:19  00:01:22  B G      DR 1

R8#show ip pim vrf OSPF neighbor | begin ^Neigh
Neighbor      Interface      Uptime/Expires    Ver    DR

```

Address			Prio/Mode
10.4.8.4	GigabitEthernet2.548	06:39:51/00:01:36 v2	1 / S P G
24.0.0.2	Lspvif0	00:13:15/00:01:16 v2	1 / DR S P G

```
R8#show ipv6 pim vrf OSPF neighbor | begin ^Neigh
```

Neighbor Address	Interface	Uptime	Expires	Mode	DR	pri
::FFFF:24.0.0.2	Lspvif0	00:13:27	00:01:36	B G	DR	1
FE80::4	Gi2.548	06:39:50	00:01:38	B G		1

To test customer multicast across this MVPN instance, CSR4 has joined an IPv6 SSM group of FF33::9 from source ::10:9:9:9 (INCLUDE mode). We confirm that CSR4 is properly joined to this group and that the appropriate C(S,G) SSM join is sent towards CSR8, the egress PE. The little 's' flag in the MRIB state signifies an SSM entry.

```
R4#show ipv6 mld groups FF33::9 detail
```

```
Interface: Loopback0
Group: FF33::9
Uptime: 3d07h
Router mode: INCLUDE
Host mode: INCLUDE
Last reporter: FE80::21E:49FF:FE80:B400
Group source list:
Source Address      Uptime    Expires    Fwd  Flags
::10:9:9:9          3d07h     00:03:53   Yes  Remote Local 2D
```

```
R4#show ipv6 mroute FF33::9 ::10:9:9:9 | begin \((
```

```
(::10:9:9:9, FF33::9), 3d07h/never, flags: sLTI
Incoming interface: GigabitEthernet2.548
RPF nbr: FE80::8
Immediate Outgoing interface list:
Loopback0, Forward, 3d07h/never
```

CSR8 learns this C(S,G) state from CSR4 and installs it in the C-MRIB. The incoming interface is the PMSI. The join is transported over the default MDT, which is a collection of dynamically-built P2MP mLDLP trees, to CSR2. CSR2 receive traffic from the customer and sends it to the PMSI.

```
R8#show ipv6 mroute vrf OSPF FF33::9 ::10:9:9:9 | begin \((
```

```
(::10:9:9:9, FF33::9), 01:41:59/00:02:34, flags: sT
Incoming interface: Lspvif0
RPF nbr: ::FFFF:24.0.0.2
Immediate Outgoing interface list:
GigabitEthernet2.548, Forward, 01:41:59/00:02:34
```

```
R2#show ipv6 mroute vrf OSPF FF33::9 ::10:9:9:9 | begin \((
```

```
(::10:9:9:9, FF33::9), 00:23:39/00:02:53, flags: sT
Incoming interface: GigabitEthernet2.529
RPF nbr: FE80::9
```

```
Immediate Outgoing interface list:  
  Lspvif0, Forward, 00:23:39/00:02:53
```

CSR9 is the root of the C-SPT and expects to receive traffic from its loopback interface. Traffic is then sent towards CSR2, the ingress PE. This customer multicast signaling will never change as SSM does not rely on an RP nor SPT switchover.

```
R9#show ipv6 mroute FF33::9 ::10:9:9:9 | begin \  
(::10:9:9:9, FF33::9), 00:24:43/00:02:53, flags: sT  
  Incoming interface: Loopback0  
  RPF nbr: FE80::21E:E5FF:FEA2:5700  
  Immediate Outgoing interface list:  
    GigabitEthernet2.529, Forward, 00:24:43/00:02:53
```

CSR9 will start sending IPv6 multicast towards FF33::9 to which CSR4 is listening.

```
R9#ping ipv6  
Target IPv6 address: FF33::9  
Repeat count [5]: 10000  
Datagram size [100]:  
Timeout in seconds [2]: 1  
Extended commands? [no]: y  
Source address or interface: ::10:9:9:9  
[snip]  
Output Interface: loopback0
```

For brevity, I check the counters on CSR2 and CSR8 to ensure the traffic is being properly encapsulated and decapsulated. Packets received on CSR2 have a frame size (including layer 2) of 118 bytes, while CSR8 sees the with a single MPLS label for LSM transport. This proves that the MVPN instance is working correctly.

```
R2#show ipv6 mroute vrf OSPF FF33::9 ::10:9:9:9 count | begin ^Group  
Group: FF33::9  
  Source: ::10:9:9:9,  
    SW Forwarding: 0/0/0/0, Other: 0/0/0  
    HW Forwarding: 92/1/118/0, Other: 0/0/0  
  Totals - Source count: 1, Packet count: 92
```

```
R8#show ipv6 mroute vrf OSPF FF33::9 ::10:9:9:9 count | begin ^Group  
Group: FF33::9  
  Source: ::10:9:9:9,  
    SW Forwarding: 0/0/0/0, Other: 0/0/0  
    HW Forwarding: 96/1/126/0, Other: 0/0/0  
  Totals - Source count: 1, Packet count: 96
```

Finally, we check the packet counters on CSR4 to ensure the end-user is receiving the traffic. CSR4 sees a 118 byte packet (including layer 2 encapsulation) after the MPLS labels are removed by CSR8.

```
R4#show ipv6 mroute FF33::9 ::10:9:9:9 count | begin ^Group
Group: FF33::9
  Source: ::10:9:9:9,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 100/1/118/0, Other: 0/0/0
  Totals - Source count: 1, Packet count: 100
```

8.4.3.5 MPLS TE

Unlike option B, inter-AS MPLS TE actually makes sense for option C. Since the peer ASes are supposed to know about the remote PE loopbacks, building a tunnel directly between PEs is a valid design choice. Before continuing, I perform the same basic TE topology checks from option B to ensure all links/nodes are configured correctly. Both ASes have exactly 4 routers, and the transit links are enabled for TE as well. IS-IS IGP IDs show these like NETs with the high-order 32 bits mapped to the peers static TE ID. OSPF shows them as simple IP addresses. CSR6's link to XRv1 doesn't actually work, but I highlight it for completeness. XR does not have a "passive-interface" equivalent.

```
R2#show mpls traffic-eng topology brief | include IGP Id
IGP Id: 0000.0000.0002.00, MPLS TE Id:24.0.0.2 Router Node (isis level-2)
  link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0014.00, nbr_node_id:4, gen:54
  link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0007.00, nbr_node_id:3, gen:54
IGP Id: 0000.0000.0006.00, MPLS TE Id:24.0.0.6 Router Node (isis level-2)
  link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0014.00, nbr_node_id:4, gen:57
  link[1]: Point-to-Point, Nbr IGP Id: 0D00.0005.0000.00, nbr_node_id:4294967295,
gen:57
  link[2]: Point-to-Point, Nbr IGP Id: 0D00.000B.0000.00, nbr_node_id:4294967295,
gen:57
  link[3]: Point-to-Point, Nbr IGP Id: 0000.0000.0007.00, nbr_node_id:3, gen:57
IGP Id: 0000.0000.0007.00, MPLS TE Id:24.0.0.7 Router Node (isis level-2)
  link[0]: Point-to-Point, Nbr IGP Id: 0D00.0005.0000.00, nbr_node_id:4294967295,
gen:55
  link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0006.00, nbr_node_id:2, gen:55
  link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0002.00, nbr_node_id:1, gen:55
  link[3]: Point-to-Point, Nbr IGP Id: 0000.0000.0014.00, nbr_node_id:4, gen:55
IGP Id: 0000.0000.0014.00, MPLS TE Id:24.0.0.14 Router Node (isis level-2)
  link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0002.00, nbr_node_id:1, gen:58
  link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0006.00, nbr_node_id:2, gen:58
  link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0007.00, nbr_node_id:3, gen:58

R8#show mpls traffic-eng topology brief | include IGP Id
IGP Id: 13.0.0.5, MPLS TE Id:13.0.0.5 Router Node (ospf 13 area 0)
  link[0]: Point-to-Point, Nbr IGP Id: 13.0.0.8, nbr_node_id:1, gen:53
  link[1]: Point-to-Point, Nbr IGP Id: 13.0.0.11, nbr_node_id:3, gen:53
  link[2]: Point-to-Point, Nbr IGP Id: 24.0.0.7, nbr_node_id:4294967295, gen:53
  link[3]: Point-to-Point, Nbr IGP Id: 24.0.0.6, nbr_node_id:4294967295, gen:53
IGP Id: 13.0.0.8, MPLS TE Id:13.0.0.8 Router Node (ospf 13 area 0)
  link[0]: Point-to-Point, Nbr IGP Id: 13.0.0.11, nbr_node_id:3, gen:52
  link[1]: Point-to-Point, Nbr IGP Id: 13.0.0.12, nbr_node_id:4, gen:52
```

```

    link[2]: Point-to-Point, Nbr IGP Id: 13.0.0.5, nbr_node_id:2, gen:52
IGP Id: 13.0.0.11, MPLS TE Id:13.0.0.11 Router Node (ospf 13 area 0)
    link[0]: Point-to-Point, Nbr IGP Id: 13.0.0.5, nbr_node_id:2, gen:47
    link[1]: Point-to-Point, Nbr IGP Id: 13.0.0.12, nbr_node_id:4, gen:47
    link[2]: Point-to-Point, Nbr IGP Id: 13.0.0.8, nbr_node_id:1, gen:47
IGP Id: 13.0.0.12, MPLS TE Id:13.0.0.12 Router Node (ospf 13 area 0)
    link[0]: Point-to-Point, Nbr IGP Id: 13.0.0.8, nbr_node_id:1, gen:54
    link[1]: Point-to-Point, Nbr IGP Id: 13.0.0.11, nbr_node_id:3, gen:54

```

The normal tunnel stitching methods seen in option A and the UMPLS chapter are also valid as this gives each AS full control over inter-AS policies. Those are not examined again here. Instead, I will build a pair of inter-AS TE tunnels from XRv2 to CSR2 and XRv2 to XRv4. Below is the configuration for CSR2; the loose-hop path expansion was discussed in detail in option B. This section will details the process with debugs again. In short, the loose-hops are expanded at each router where necessary and substituted with a set of strict hops. Unlike traditional TE, PCALC is invoked at each loose-hop so they can “expand” the ERO appropriately. The design assumes that not every node sees every other node, but certain border nodes have additional information; it is somewhat similar to distance-vector routing logic. CSR2 will prefer the high-cost link directly to CSR7, then continue through XRv4 and CSR6 to exit AS 13. Once the ERO reaches CSR6, CSR6 will expand the hop to CSR7 by finding the best dynamic path. The same occurs once CSR7 receives the ERO as the path inside AS 13 will be dynamic as well. **Note: CSR6 must be a loose hop because 13.0.0.5 does not appear in the TED as a router node, despite CSR6 being statically connected to it.** I also add a 75 Mbps bandwidth reservation with custom priorities to show that this is honored across AS boundaries.

```

! CSR2
ip explicit-path name EP_2_7_14_6_L5_L12 enable
  next-address 24.0.0.7
  next-address 24.0.0.14
  next-address loose 24.0.0.6
  next-address loose 13.0.0.5
  next-address loose 13.0.0.12

interface Tunnel300
  description INTER-AS LOOSE HOP
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 13.0.0.12
  tunnel mpls traffic-eng autoroute destination
  tunnel mpls traffic-eng priority 5 3
  tunnel mpls traffic-eng bandwidth 75000
  tunnel mpls traffic-eng path-option 10 explicit name EP_2_7_14_6_L5_L12

! CSR2, CSR5, and CSR6
debug mpls traffic-eng path lookup

```

First, CSR2 invokes the regular PCALC operation to find the path to 13.0.0.12 within the TE constraints. The explicit-path was supplied versus dynamically computed and contains a combination of strict and

loose hops. The intra-AS strict hops give CSR2 (headend) more control over the internal path rather than computing any dynamic path to CSR6. CSR2 is not aware of node 13.0.0.12 in the TED but finds CSR6 as a loose-hop. This node is within the AS and the strict hops to CSR6 have been supplied so the AS 13 TE midpoints don't need to invoke PCALC; they just need to perform the RSVP signaling and update the ERO as needed. The debug is left intact for reference, but the important pieces have been discussed. The remaining outputs are related to path verification and explicit hop expansion for signaling.

```
! CSR2
TE-PCALC-API: 24.0.0.2_10->13.0.0.12_300 {7}: P2P LSP Path Lookup called
TE-PCALC: 24.0.0.2_10->13.0.0.12_300 {7}: Path Request Info
  Flags:  IP_EXPLICIT_PATH  METRIC_TE
  IP explicit-path: Supplied
    24.0.0.7
    24.0.0.14
    24.0.0.6 Loose
    13.0.0.5 Loose
    13.0.0.12 Loose
  bw 75000, min_bw 0, metric: 0
  setup_pri 5, hold_pri 3
  affinity_bits 0x0, affinity_mask 0xFFFF
TE-PCALC-PATH: 24.0.0.2_10->13.0.0.12_300 {7}: Area (isis level-2) Path
Lookup begin
TE-PCALC-PATH: Area (isis level-2): Dest ip addr 13.0.0.12 not found
TE-PCALC-PATH: lsr_exists:first Loose Hop is to addr 24.0.0.6
TE-PCALC-PATH:path_to_lsr (Loose Route after explicit-2 Segment: 24.0.0.6,
actual_hops 2
TE-PCALC-PATH:Addresses collected so far:
TE-PCALC-PATH: 24.2.7.7
TE-PCALC-PATH: 24.7.14.14
TE-PCALC-PATH: Addresses left:
TE-PCALC-PATH24.0.0.6: Loose
TE-PCALC-PATH24.0.0.6: Loose
TE-PCALC-PATH:Path from 0000.0000.0014.00 -> 0000.0000.0006.00:
  24.6.14.14->24.6.14.6 (admin_weight=10):
  num_hops 2, accumulated_aw 10, min_bw 200000
TE-PCALC-PATH: 24.0.0.2_10->13.0.0.12_300 {7}: Freeing rrr_path_setup_t
TE-PCALC-PATH: 24.0.0.2_10->13.0.0.12_300 {7}: Free all paths in path tree
TE-PCALC: Verify Path Lookup: 24.0.0.2_10->13.0.0.12_300 {7}: (protocol nil
area nil)
  Flags:  METRIC_TE
  Last Strict Router: 24.0.0.6
  sub-lsp weight:0 (Total LSP weight:10)
  Hop List:
    24.2.7.7
    24.7.14.14
    24.6.14.6
    24.0.0.6
    13.0.0.5 Loose
```

```

13.0.0.12 Loose
TE-PCALC-VERIFY: VERIFY to 24.0.0.6 BEGIN:
TE-PCALC-VERIFY: Verify:
TE-PCALC-VERIFY: 0000.0000.0002.00, 24.0.0.2 points to
TE-PCALC-VERIFY: 0000.0000.0007.00, 24.2.7.7
TE-PCALC-VERIFY: Verify:
TE-PCALC-VERIFY: 0000.0000.0007.00, 24.2.7.7 points to
TE-PCALC-VERIFY: 0000.0000.0014.00, 24.7.14.14
TE-PCALC-VERIFY: Verify:
TE-PCALC-VERIFY: 0000.0000.0014.00, 24.7.14.14 points to
TE-PCALC-VERIFY: 0000.0000.0006.00, 24.6.14.6
TE-PCALC-VERIFY: VERIFY to 24.0.0.6 PASSED
TE-PCALC-PATH: 24.0.0.2_10->13.0.0.12_300 {7}: Area (isis level-2) Path
Lookup end: path found
TE-PCALC-API: 24.0.0.2_10->13.0.0.12_300 {7}: P2P LSP Path Lookup result:
success

```

We can confirm the exact ERO message that CSR2 sends to CSR7 by checking the RSVP PATH message. The intra-AS hops have been fully expanded as CSR6's loose-hop status has been removed. CSR5 becomes the first loose-hop now.

```

R2#show ip rsvp sender detail filter session-type 7 tunnel-id 300 | section
outgoing
ERO: (outgoing)
  24.2.7.7 (Strict IPv4 Prefix, 8 bytes, /32)
  24.7.14.14 (Strict IPv4 Prefix, 8 bytes, /32)
  24.6.14.6 (Strict IPv4 Prefix, 8 bytes, /32)
  24.0.0.6 (Strict IPv4 Prefix, 8 bytes, /32)
  13.0.0.5 (Loose IPv4 Prefix, 8 bytes, /32)
  13.0.0.12 (Loose IPv4 Prefix, 8 bytes, /32)

```

Just like in option B, the LSR on the edge of the source AS (where the headend is) reports a failure to expand the loose hop. CSR6 invokes the PCALC expansion which normal TE midpoints need not do, but since the incoming ERO contained a loose-hop, CSR6 must find a path to it. Because an explicit-path to CSR5 was not supplied, CSR6 assumes any dynamic path may be used, but the bandwidth reservation and TE priorities are maintained. One thing that is NOT maintained is the affinity; between ASes, it is assumed that link coloring schemes will be totally different, so affinity is completely ignored. CSR6 also takes care not to include XRv4, CSR7, or CSR2 in the loose-hop expansion as this would create a loop. These are carried in the RSVP PATH RRO which we examine later.

```

! CSR6
TE-PCALC-API: 24.0.0.2_10->13.0.0.5_300 {7}: LSP Path Expand called
TE-PCALC: 24.0.0.2_10->13.0.0.5_300 {7}: Path Request Info
  Flags: END_SWCAP_UNKNOWN
  IP explicit-path: None (dynamic)
  bw 75000, min_bw 0, metric: 0
  setup_pri 5, hold_pri 3

```

```

    affinity_bits 0x0, affinity_mask 0x0
TE-PCALC-PATH: 24.0.0.2_10->13.0.0.5_300 {7}: rrr_pcalc_lsr_expand: Exclude
node: 24.0.0.14 (intf: 24.6.14.14)
TE-PCALC-PATH: 24.0.0.2_10->13.0.0.5_300 {7}: rrr_pcalc_lsr_expand: Exclude
node: 24.0.0.7 (intf: 24.7.14.7)
TE-PCALC-PATH: 24.0.0.2_10->13.0.0.5_300 {7}: rrr_pcalc_lsr_expand: Exclude
node: 24.0.0.2 (intf: 24.2.7.2)
TE-PCALC-PATH: 24.0.0.2_10->13.0.0.5_300 {7}: Area (isis level-2) Path
Lookup begin
TE-PCALC-PATH: expand_lsr: Dst addr 13.0.0.5 not found in area (isis level-
2)
TE-PCALC-PATH: 24.0.0.6_10->13.0.0.5_300 {7}: Area (isis level-2) Path
Lookup end: path not found
13.0.0.5 Can't Expand at this time
TE-PCALC-API: 24.0.0.6_10->13.0.0.5_300 {7}: LSP Path Expand result: failed

```

Ultimately, we see CSR6 fails to expand 13.0.0.5 into strict hops. However, CSR6 removes its local addresses from the incoming ERO and adds the transit link address (specified in the TE passive interface statement) to the outgoing ERO. Although CSR6 failed to expand 13.0.0.5, it can at least continue to signal the LSP. IT assumes that someone else down the path will be able to expand 13.0.0.5, which is a valid assumption in this case. The RRO is used to ensure, when loose-hop expansion is enabled, that a router computing a loose-hop does not select a dynamic path via a node already in the LSP. This is a stack that has new TE hops pushed as the LSP is signaled.

```

R6#show ip rsvp sender detail filter session-type 7 tunnel-id 300 | section
[ER]RO
  ERO: (incoming)
    24.6.14.6 (Strict IPv4 Prefix, 8 bytes, /32)
    24.0.0.6 (Strict IPv4 Prefix, 8 bytes, /32)
    13.0.0.5 (Loose IPv4 Prefix, 8 bytes, /32)
    13.0.0.12 (Loose IPv4 Prefix, 8 bytes, /32)
  ERO: (outgoing)
    10.5.6.5 (Strict IPv4 Prefix, 8 bytes, /32)
    13.0.0.5 (Loose IPv4 Prefix, 8 bytes, /32)
    13.0.0.12 (Loose IPv4 Prefix, 8 bytes, /32)
  RRO:
    24.6.14.14/32, Flags:0x0 (No Local Protection)
    24.7.14.7/32, Flags:0x0 (No Local Protection)
    24.2.7.2/32, Flags:0x0 (No Local Protection)

```

CSR5 receives the RSVP PATH message from CSR6 containing an ERO with its own TE ID as a loose-hop. This is loose-hop need not be expanded since it is a local address, but CSR5 must invoke the PCALC loose-hop expansion process to expand 13.0.0.12. Again, a colorless dynamic path is assumed, but the bandwidth reservations and TE priorities are honored. Being in a different AS than the other routers in the RRO, CSR5 cannot find the TE RIDs for CSR6, XRv4, CSR7, or CSR2. This is acceptable since it was only

doing this to prevent loops in the first place; being unable to find them in the graph at all guarantees that a loop cannot form. CSR5 computes the shortest path to XRv2 is via CSR8.

```
! CSR5
TE-PCALC-API: 24.0.0.2_10->13.0.0.12_300 {7}: LSP Path Expand called
TE-PCALC: 24.0.0.2_10->13.0.0.12_300 {7}: Path Request Info
  Flags:  END_SWCAP_UNKNOWN
  IP explicit-path: None (dynamic)
  bw 75000, min_bw 0, metric: 0
  setup_pri 5, hold_pri 3
  affinity_bits 0x0, affinity_mask 0x0
TE-PCALC-PATH: 24.0.0.2_10->13.0.0.12_300 {7}: rrr_pcalc_lsr_expand: Can't
get router ID addr for 10.5.6.6
TE-PCALC-PATH: 24.0.0.2_10->13.0.0.12_300 {7}: rrr_pcalc_lsr_expand: Can't
get router ID addr for 24.6.14.14
TE-PCALC-PATH: 24.0.0.2_10->13.0.0.12_300 {7}: rrr_pcalc_lsr_expand: Can't
get router ID addr for 24.7.14.7
TE-PCALC-PATH: 24.0.0.2_10->13.0.0.12_300 {7}: rrr_pcalc_lsr_expand: Can't
get router ID addr for 24.2.7.2
TE-PCALC-PATH: 24.0.0.2_10->13.0.0.12_300 {7}: Area (ospf 13 area 0) Path
Lookup begin
TE-PCALC-PATH: exclude_path: system_id 0-0-0-0-0-0-0 not known!
TE-PCALC-PATH: exclude_path: system_id 0-0-0-0-0-0-0 not known!
TE-PCALC-PATH: exclude_path: system_id 0-0-0-0-0-0-0 not known!
TE-PCALC-PATH: exclude_path: system_id 0-0-0-0-0-0-0 not known!
TE-PCALC-PATH: Path from 13.0.0.5 -> 13.0.0.12:
  13.8.12.8->13.8.12.12 (admin_weight=2):
  13.5.8.5->13.5.8.8 (admin_weight=1):
  num_hops 3, accumulated_aw 2, min_bw 200000
TE-PCALC-PATH: 13.0.0.5_10->13.0.0.12_300 {7}: Area (ospf 13 area 0) Path
Lookup end: path found
13.0.0.12 expands to:
13.5.8.8
13.8.12.12
13.0.0.12
TE-PCALC-API: 13.0.0.5_10->13.0.0.12_300 {7}: LSP Path Expand result: success
```

CSR5 is the last router that needs to expand a loose-hop. When the ERO arrived from CSR6, it contained loose-hops. CSR5 removed 13.0.0.5 without expansion (local TE-ID) and expanded 13.0.0.12 manually (final destination). The resulting ERO signaled to CSR8 via RSVP PATH messaging contains only intra-AS strict hops. No other TE LSRs in AS 13 will need to run PCALC again.

```
R5#show ip rsvp sender detail filter session-type 7 tunnel-id 300 | section
ERO
ERO: (incoming)
  10.5.6.5 (Strict IPv4 Prefix, 8 bytes, /32)
  13.0.0.5 (Loose IPv4 Prefix, 8 bytes, /32)
  13.0.0.12 (Loose IPv4 Prefix, 8 bytes, /32)
```

ERO: (outgoing)

13.5.8.8 (Strict IPv4 Prefix, 8 bytes, /32)
13.8.12.12 (Strict IPv4 Prefix, 8 bytes, /32)
13.0.0.12 (Strict IPv4 Prefix, 8 bytes, /32)

Checking XRv2, we can see that it is the tail end for a single TE LSP. The tunnel has been signaled properly with the bandwidth reservation and TE priorities maintained. The RSVP PATH RRO shows the entire LSP back to the remote AS source, which allows the tail end to see the entire path. I show all of the details for reference, but the tunnel generally looks like a normal TE tunnel.

```
RP/0/0/CPU0:XRv2#show mpls traffic-eng tunnels role tail
```

LSP Tunnel 24.0.0.2 300 [12] is signalled, Signaling State: up

Tunnel Name: INTER-AS LOOSE HOP Tunnel Role: Tail

InLabel: GigabitEthernet0/0/0/0.582, implicit-null

Signalling Info:

Src 24.0.0.2 Dst 13.0.0.12, Tun ID 300, Tun Inst 12, Ext ID 24.0.0.2

Router-IDs: upstream 13.0.0.8

local 13.0.0.12

Bandwidth: 75000 kbps (CT0) Priority: 5 3 DSTE-class: no match

Soft Preemption: None

SRLGs: not collected

Path Info:

Incoming Address: 13.8.12.12

Incoming:

Explicit Route:

Strict, 13.8.12.12

Strict, 13.0.0.12

Record Route:

IPv4 13.8.12.8, flags 0x0

IPv4 13.5.8.5, flags 0x0

IPv4 10.5.6.6, flags 0x0

IPv4 24.6.14.14, flags 0x0

IPv4 24.7.14.7, flags 0x0

IPv4 24.2.7.2, flags 0x0

Tspec: avg rate=75000 kbits, burst=1000 bytes, peak rate=75000 kbits

Session Attributes: Local Prot: Not Set, Node Prot: Not Set, BW Prot:

Not Set

Soft Preemption Desired: Not Set

Resv Info: None

Record Route: Disabled

Fspec: avg rate=75000 kbits, burst=1000 bytes, peak rate=75000 kbits

Displayed 0 (of 0) heads, 0 (of 0) midpoints, 1 (of 1) tails

Displayed 0 up, 0 down, 0 recovering, 0 recovered heads

Another nice feature of these inter-AS tunnels is the RSVP RESV RRO is automatically enabled. Though we did not configure it, CSR2 is also shown all hops along the path. I show the entire RESV message

because this also confirms the 75 Mbps bandwidth reservation for the headend. It also carries the MPLS label, and the first hop is via CSR7 using label 7093.

```
R2#show ip rsvp reservation detail filter session-type 7 tunnel-id 300
Reservation:
  Tun Dest:    13.0.0.12  Tun ID: 300  Ext Tun ID: 24.0.0.2
  Tun Sender: 24.0.0.2  LSP ID: 12
  Next Hop: 24.2.7.7 on GigabitEthernet2.527
  Label: 7093 (outgoing)
  Reservation Style is Shared-Explicit, QoS Service is Controlled-Load
  Resv ID handle: 08000408.
  Average Bitrate is 75M bits/sec, Maximum Burst is 1K bytes
  Min Policed Unit: 0 bytes, Max Pkt Size: 1500 bytes
  RRO:
    24.7.14.7/32, Flags:0x0 (No Local Protection)
    24.7.14.14/32, Flags:0x0 (No Local Protection)
    10.5.6.6/32, Flags:0x0 (No Local Protection)
    13.5.8.5/32, Flags:0x0 (No Local Protection)
    13.8.12.8/32, Flags:0x0 (No Local Protection)
    13.8.12.12/32, Flags:0x0 (No Local Protection)
  Status:
  Policy: Accepted. Policy source(s): MPLS/TE
```

We can quickly check the integrity of the TE LSP using MPLS OAM traceroute. This reveals all of the TE labels along the way. It also confirms that the tunnel formed in an acceptable manner; CSR2 as only able to influence the first few intra-AS hops. Everything after CSR6 was entirely dynamic. Notice label 7093 at the first hop.

```
R2#traceroute mpls traffic-eng tunnel 300 source 24.0.0.2 force-explicit-null
Tracing MPLS TE Label Switched Path on Tunnel300, timeout is 2 seconds
[snip]
Type escape sequence to abort.
 0 24.2.7.2 MRU 1500 [Labels: 7093/explicit-null Exp: 0/0]
L 1 24.2.7.7 MRU 1500 [Labels: 94027/explicit-null Exp: 0/0] 10 ms
L 2 24.7.14.14 MRU 1500 [Labels: 6031/explicit-null Exp: 0/0] 2 ms
L 3 24.6.14.6 MRU 1500 [Labels: 5084/explicit-null Exp: 0/0] 10 ms
L 4 10.5.6.5 MRU 1500 [Labels: 8022/explicit-null Exp: 0/0] 7 ms
L 5 13.5.8.8 MRU 1500 [Labels: explicit-null Exp: 0] 7 ms
! 6 13.8.12.12 9 ms
```

In the configuration, CSR2 used autoroute destination to automatically installed a static route to 13.0.0.12/32 via the TE tunnel. Autoroute announce and forwarding adjacency are not supported for inter-AS or inter-area tunnels, so static routes are the best option when using TE to support MPLS VPNs. The static routing database also notes that this was dynamically added via MPLS TE and is currently active.

```
R2#show ip route 13.0.0.12
Routing entry for 13.0.0.12/32
  Known via "static", distance 1, metric 0 (connected)
  Routing Descriptor Blocks:
    * directly connected, via Tunnel300
      Route metric is 0, traffic share count is 1
```

```
R2#show ip static route | begin RIB
Static local RIB for default
TE 13.0.0.12/32 [1/0] via Tunnel300 [A]
```

Since CSR2 is routing all traffic to 13.0.0.12 in this tunnel, we can assume the eBGP session packets from CSR2 to XRv2 are also inside the tunnel. Using traceroute on CSR1 inside the EIGRP VPN, we can verify that the tunnel is used. The label stack only ever consists of two labels since LDP and BGP labeled unicast are entirely bypassed with this design. We confirm label 7093 at the first hop, proving that the TE LSP is in use. The reverse LSP is whatever XRv2 was always using as the TE LSP is unidirectional.

```
R1#traceroute 10.3.3.3 source 10.1.1.1
Type escape sequence to abort.
Tracing the route to 10.3.3.3
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.2.2 3 msec 4 msec 3 msec
 2 24.2.7.7 [MPLS: Labels 7093/92006 Exp 0] 12 msec 9 msec 8 msec
 3 24.7.14.14 [MPLS: Labels 94027/92006 Exp 0] 27 msec 31 msec 32 msec
 4 24.6.14.6 [MPLS: Labels 6031/92006 Exp 0] 32 msec 31 msec 32 msec
 5 10.5.6.5 [MPLS: Labels 5084/92006 Exp 0] 31 msec 32 msec 32 msec
 6 13.5.8.8 [MPLS: Labels 8022/92006 Exp 0] 32 msec 31 msec 32 msec
 7 13.8.12.12 [MPLS: Label 92006 Exp 0] 21 msec 21 msec 20 msec
 8 10.3.12.3 20 msec 13 msec 11 msec
```

We also must ensure that this TE tunnel does not break multicast. Since CSR2 is supporting an MVPN profile 0 deployment for the EIGRP VPN, we must consider that the TE tunnel will break RPF towards 13.0.0.2. PIM is not enabled on the TE tunnel.

```
R2#show ip rpf 13.0.0.12
failed, no route exists
```

This solution is discussed in other chapters involves telling IGP not to consider TE tunnels as RPF interfaces. This does not work when static routes are used to route traffic into TE tunnels, and I prove it quickly. This technique is only valid when “autoroute announce” is used.

```
! CSR2
router isis 24
  mpls traffic-eng multicast-intact
```

```
R2#show ip rpf 13.0.0.12
failed, no route exists
```

To solve this problem, we can use BGP IPv4 multicast or a simple static multicast route. I choose the latter for simplicity. This solution can be sloppy in a network with multiple links and does not scale well.

```
! CSR2
ip mroute 13.0.0.12 255.255.255.255 24.2.14.14

R2#show ip rpf 13.0.0.12
RPF information for ? (13.0.0.12)
  RPF interface: GigabitEthernet2.524
  RPF neighbor: ? (24.2.14.14)
  RPF route/mask: 13.0.0.12/32
  RPF type: multicast (static)
  Doing distance-preferred lookups across tables
  RPF topology: ipv4 multicast base
```

With CSR2's RPF fixed, we should still be able to send multicast from CSR2 to XRv2. A quick ping test on CSR3 confirms it. CSR3 sends packets and XRv3 receives them.

```
R3#ping ip
Target IP address: 225.13.13.13
Repeat count [1]: 1000
Datagram size [100]:
Timeout in seconds [2]: 1
Extended commands [n]: y
Interface [All]: loopback0
Time to live [255]:
Source address or interface: loopback0

RP/0/0/CPU0:XRv3#show mfib route 225.13.13.13 10.3.3.3 | begin 225
(10.3.3.3,225.13.13.13),    Flags:
  Up: 00:01:17
  Last Used: 00:01:04
  SW Forwarding Counts: 13/13/1300
  SW Replication Counts: 13/0/0
  SW Failure Counts: 0/0/0/0/0
  Loopback0 Flags:  IC NS EG, Up:00:01:17
  GigabitEthernet0/0/0/0.513 Flags:  A, Up:00:01:17
```

We shut down this tunnel on CSR2 and create another [ID 301] using the “verbatim” option. This technique was discussed in detail in the TE section and is particularly useful for inter-AS TE (along with other cases such as the intermediate nodes not support TE IGP extensions). Normally, explicit paths are checked against the TED for consistency; this protects against signaling impossible paths. There is no reason to waste RSVP signaling messages trying to build a path that is nonexistent, so the local PCALC process will not even attempt to signal the LSP. The “verbatim” keyword overrides this sanity check which bypasses the TED entirely. The configured explicit-path can use TE IDs within an AS but must use

the specific link addresses in the remote AS. RSVP RESV RRO is not very valuable on explicit-path tunnels like this, but I enable it anyway to demonstrate it does work inter-AS. It is not enabled automatically on tunnels which actively use an explicit-path with the verbatim option.

```
! CSR2
ip explicit-path name EP_2_14_7_5_11_12_VERBATIM enable
  next-address 24.0.0.14
  next-address 24.0.0.7
  next-address 10.5.7.5
  next-address 13.5.11.11
  next-address 13.11.12.12
  next-address 13.0.0.12

interface Tunnel301
  description INTER-AS VERBATIM
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 13.0.0.12
  tunnel mpls traffic-eng autoroute destination
  tunnel mpls traffic-eng priority 4 2
  tunnel mpls traffic-eng bandwidth 10500
  tunnel mpls traffic-eng path-option 10 explicit name
  EP_2_14_7_5_11_12_VERBATIM verbatim
  tunnel mpls traffic-eng record-route
```

MPLS path debugging remains enabled on CSR2 but disabled on all other routers. Since there is no loose-hop expansion, no other router will invoke PCALC except CSR2. The PCALC logic is very simple as PCALC does essentially nothing at all for TE IDs it does not recognize. I highlight all the debug output that shows us the TED bypass in action. PCALC is aware that it may not be able to find all of the TE IDs for all nodes. It is able to find the hop addresses for the intra-AS hops, which allows us to use TE IDs versus exact link addressing, so PCALC does a small amount of work.

```
! CSR2
TE-PCALC-API: 24.0.0.2_35->13.0.0.12_301 {7}: P2P LSP Path Lookup called
TE-PCALC: 24.0.0.2_35->13.0.0.12_301 {7}: Path Request Info
  Flags:  IP_EXPLICIT_PATH  NO_TOPOLOGY_CHECK  METRIC_TE
  IP explicit-path: Supplied
    24.0.0.14
    24.0.0.7
    10.5.7.5
    13.5.11.11
    13.11.12.12
    13.0.0.12
  bw 10500, min_bw 0, metric: 0
  setup_pri 4, hold_pri 2
  affinity_bits 0x0, affinity_mask 0xFFFF
```

```

TE-PCALC-PATH: Get Path Common: Skip topology check for verbatim path to dest
13.0.0.12
TE-PCALC-PATH: get_path_no_check: share count=0, dest=13.0.0.12
TE-PCALC-PATH: static_path_no_check on verbatim path to 13.0.0.12
TE-PCALC-PATH: Got node info for hop: 24.0.0.14, sys-id:0000.0000.0014.00,
igmp-id out:(isis level-2), in:(isis level-2)
TE-PCALC-PATH: Got node info for hop: 24.0.0.7, sys-id:0000.0000.0007.00,
igmp-id out:(isis level-2), in:(isis level-2)
TE-PCALC-PATH: can't find node info for address 10.5.7.5
TE-PCALC-PATH: Failed to get node info for hop 10.5.7.5
TE-PCALC-PATH: can't find node info for address 13.5.11.11
TE-PCALC-PATH: Failed to get node info for hop 13.5.11.11
TE-PCALC-PATH: can't find node info for address 13.11.12.12
TE-PCALC-PATH: Failed to get node info for hop 13.11.12.12
TE-PCALC-PATH: can't find node info for address 13.0.0.12
TE-PCALC-PATH: Failed to get node info for hop 13.0.0.12
TE-PCALC-API: 24.0.0.2_35->13.0.0.12_301 {7}: P2P LSP Path Lookup result:
success

```

Looking at the ERO details, PCALC literally makes no changes other than removing the local addresses from the ERO before sending it onward. Even the TE IDs remain unexpanded, but the receiving router can quickly find the next-hop based on the intra-AS TE ID. The entire LSP is signaled very literally, hence the word “verbatim”.

```

R2#show ip rsvp sender detail filter session-type 7 tunnel-id 301 | section
ERO

```

ERO: (incoming)

```

24.0.0.2 (Strict IPv4 Prefix, 8 bytes, /32)
24.0.0.14 (Strict IPv4 Prefix, 8 bytes, /32)
24.0.0.7 (Strict IPv4 Prefix, 8 bytes, /32)
10.5.7.5 (Strict IPv4 Prefix, 8 bytes, /32)
13.5.11.11 (Strict IPv4 Prefix, 8 bytes, /32)
13.11.12.12 (Strict IPv4 Prefix, 8 bytes, /32)
13.0.0.12 (Strict IPv4 Prefix, 8 bytes, /32)

```

ERO: (outgoing)

```

24.0.0.14 (Strict IPv4 Prefix, 8 bytes, /32)
24.0.0.7 (Strict IPv4 Prefix, 8 bytes, /32)
10.5.7.5 (Strict IPv4 Prefix, 8 bytes, /32)
13.5.11.11 (Strict IPv4 Prefix, 8 bytes, /32)
13.11.12.12 (Strict IPv4 Prefix, 8 bytes, /32)
13.0.0.12 (Strict IPv4 Prefix, 8 bytes, /32)

```

Of note, CSR2 is smart enough to prevent any possible loops. Like the loose-hop expansion, verbatim paths are susceptible to loops, so CSR2 includes the RSVP PATH RRO in the first PATH it sends. We can confirm this by checking XRv4. In seeing this message, XRv4 continues to push TE hops onto the RRO stack so that ultimately XRv2 will see the entire TE LSP path. Also notice that XRv4 simply removes its

local addresses from the incoming ERO and sends out the remaining ERO addresses with no additional effort.

```
RP/0/0/CPU0:XRv4#show mpls traffic-eng tunnels | begin Path
```

```
Path Info:
  Incoming Address: 24.0.0.14
  Incoming:
  Explicit Route:
    Strict, 24.0.0.14
    Strict, 24.0.0.7
    Strict, 10.5.7.5
    Strict, 13.5.11.11
    Strict, 13.11.12.12
    Strict, 13.0.0.12
  Outgoing:
  Explicit Route:
    Strict, 24.0.0.7
    Strict, 10.5.7.5
    Strict, 13.5.11.11
    Strict, 13.11.12.12
    Strict, 13.0.0.12
  Record Route:
    IPv4 24.2.14.2, flags 0x0
```

Like the previous tunnel, XRv2 is aware of the entire forward path. I show all of the output for reference; we see that the bandwidth reservations and TE priorities were also honored between ASes.

```
RP/0/0/CPU0:XRv2#show mpls traffic-eng tunnels role tail
```

```
LSP Tunnel 24.0.0.2 301 [38] is signalled, Signaling State: up
Tunnel Name: INTER-AS VERBATIM Tunnel Role: Tail
InLabel: GigabitEthernet0/0/0/0.521, implicit-null
Signalling Info:
  Src 24.0.0.2 Dst 13.0.0.12, Tun ID 301, Tun Inst 38, Ext ID 24.0.0.2
  Router-IDs: upstream 13.0.0.11
               local 13.0.0.12
  Bandwidth: 10500 kbps (CT0) Priority: 4 2 DSTE-class: no match
  Soft Preemption: None
  SRLGs: not collected
  Path Info:
    Incoming Address: 13.11.12.12
    Incoming:
    Explicit Route:
      Strict, 13.11.12.12
      Strict, 13.0.0.12
    Record Route:
      IPv4 13.11.12.11, flags 0x0
      IPv4 13.5.11.5, flags 0x0
      IPv4 10.5.7.7, flags 0x0
```



```

IPv4 24.7.14.14, flags 0x0
IPv4 24.2.14.2, flags 0x0
Tspec: avg rate=10500 kbits, burst=1000 bytes, peak rate=10500 kbits
Session Attributes: Local Prot: Not Set, Node Prot: Not Set, BW Prot:
Not Set
Soft Preemption Desired: Not Set
Resv Info: None
Record Route: Disabled
Fspec: avg rate=10500 kbits, burst=1000 bytes, peak rate=10500 kbits
Displayed 0 (of 0) heads, 0 (of 0) midpoints, 1 (of 1) tails
Displayed 0 up, 0 down, 0 recovering, 0 recovered heads

```

A quick traceroute confirms proper operation of the tunnel. Notice that the first label is 94028 which implies traffic goes to XRv4 first, as expected.

```

R2#traceroute mpls traffic-eng tunnel 301 source 24.0.0.2
Tracing MPLS TE Label Switched Path on Tunnel301, timeout is 2 seconds
[snip]
Type escape sequence to abort.
 0 24.2.14.2 MRU 1500 [Labels: 94028 Exp: 0]
L 1 24.2.14.14 MRU 1500 [Labels: 7086 Exp: 0] 3 ms
L 2 24.7.14.7 MRU 1500 [Labels: 5026 Exp: 0] 12 ms
L 3 10.5.7.5 MRU 1500 [Labels: 91013 Exp: 0] 5 ms
L 4 13.5.11.11 MRU 1500 [Labels: implicit-null Exp: 0] 6 ms
! 5 13.11.12.12 8 ms

```

VPN traffic from CSR to CSR3 is now using this TE LSP. There is only one transport label like the previous example as LDP and BGP labeled-unicast are no longer in play.

```

R1#traceroute 10.3.3.3 source 10.1.1.1
Type escape sequence to abort.
Tracing the route to 10.3.3.3
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.2.2 4 msec 4 msec 4 msec
 2 24.2.14.14 [MPLS: Labels 94028/92006 Exp 0] 10 msec 10 msec 9 msec
 3 24.7.14.7 [MPLS: Labels 7086/92006 Exp 0] 9 msec 11 msec 22 msec
 4 10.5.7.5 [MPLS: Labels 5026/92006 Exp 0] 20 msec 21 msec 25 msec
 5 13.5.11.11 [MPLS: Labels 91013/92006 Exp 0] 18 msec 20 msec 17 msec
 6 13.11.12.12 [MPLS: Label 92006 Exp 0] 21 msec 20 msec 19 msec
 7 10.3.12.3 20 msec 9 msec 11 msec

```

The static multicast route for 13.0.0.12/32 is still configured on CSR2 and is required with verbatim paths just as it is for loose-hop expansion paths (when used inter-AS). A quick multicast test from CSR2 to XRv3 confirms that the MVPN for the EIGRP customer remains operational.

```

R3#ping ip
Target IP address: 225.13.13.13

```

```
Repeat count [1]: 10000
Datagram size [100]:
Timeout in seconds [2]: 1
Extended commands [n]: y
Interface [All]: loopback0
Time to live [255]:
Source address or interface: loopback0
```

```
RP/0/0/CPU0:XRv3#show mfib route 225.13.13.13 10.3.3.3 | begin 225
(10.3.3.3,225.13.13.13),    Flags:
  Up: 00:00:56
  Last Used: 00:00:00
  SW Forwarding Counts: 56/56/5600
  SW Replication Counts: 56/0/0
  SW Failure Counts: 0/0/0/0/0
  Loopback0 Flags:  IC NS EG, Up:00:00:56
  GigabitEthernet0/0/0/0.513 Flags:  A, Up:00:00:56
```

Before continuing, I shut down tunnel301 on CSR2 and apply an AD of 255 to the multicast route (not shown). I also verify the route to 13.0.0.12/32 is IS-IS learned again. This returns the network to its baseline operation with the TE configurations on standby (shutdown but easily enabled for future testing). I quickly confirm these configurations are correct.

```
R2#show mpls traffic-eng tunnels role head brief | begin TUNNEL
P2P TUNNELS/LSPs:
TUNNEL NAME          DESTINATION      UP IF    DOWN IF    STATE/PROT
INTER-AS LOOSE HOP    13.0.0.12        -        unknown    admin-down
INTER-AS VERBATIM     13.0.0.12        -        unknown    admin-down
Displayed 2 (of 2) heads, 0 (of 0) midpoints, 0 (of 0) tails
```

```
R2#show ip route 13.0.0.12
Routing entry for 13.0.0.12/32
  Known via "isis", distance 115, metric 30, type level-2
  Redistributing via isis 24
  Last update from 24.2.14.14 on GigabitEthernet2.524, 01:05:01 ago
  Routing Descriptor Blocks:
    * 24.2.14.14, from 24.0.0.7, 01:05:01 ago, via GigabitEthernet2.524
      Route metric is 30, traffic share count is 1
```

```
R2#show ip rpf 13.0.0.12
RPF information for ? (13.0.0.12)
  RPF interface: GigabitEthernet2.524
  RPF neighbor: ? (24.2.14.14)
  RPF route/mask: 13.0.0.12/32
  RPF type: unicast (isis 24)
  Doing distance-preferred lookups across tables
  RPF topology: ipv4 multicast base, originated from ipv4 unicast base
```

We will configure a similar set of tunnels on XRv2 going to CSR2. Tunnel300 will use loose-hop expansion, along with bandwidth reservations and TE priorities.

```
! XRv2
explicit-path name EP_12_11_8_L5_L6_L2
  index 10 next-address strict ipv4 unicast 13.0.0.11
  index 20 next-address strict ipv4 unicast 13.0.0.8
  index 30 next-address loose ipv4 unicast 13.0.0.5
  index 40 next-address loose ipv4 unicast 24.0.0.6
  index 50 next-address loose ipv4 unicast 24.0.0.2

interface tunnel-te300
  description INTER-AS LOOSE HOP
  ipv4 unnumbered Loopback0
  priority 2 2
  signalled-bandwidth 64000
  destination 24.0.0.2
  path-option 10 explicit name EP_12_11_8_L5_L6_L2
```

I enable debugging for the MPLS path lookup on all routers responsible for invoking PCALC, whether it is the headend or the loose-hop expansion nodes.

```
! XRv2, CSR5, and CSR6
debug mpls traffic-eng path lookup
```

XRv2's debug output is much more verbose and less straightforward than XE. The logic begins the same: XRv2 tries to find a way to 24.0.0.2 and waits, so it finds a path to the first loose-hop instead. It resolves the link addresses for XRv1 and CSR8 and leaves CSR5 (and all other loose addresses) inside the ERO for further processing later.

```
! XRv2
te_vif_get_excluded_addresses:24713 (Tunnel ID: 300): Getting the excluded
addr for path setup from previous PERRs
rrr_pcalc_get_path:15483 (Tunnel ID: 300): TE-PCALC_PATH: get_path:Dst ip
addr 24.0.0.2 (OSPF 13 area 0) not found
rrr_pcalc_lsr_exists:11900 (Tunnel ID: 300): TE-PCALC_PATH: lsr_exists:first
Loose Source Route is to addr 13.0.0.5
rrr_pcalc_get_path_area:14938 (Tunnel ID: 300): explicit path =
EP_12_11_8_L5_L6_L2
rrr_pcalc_show_path:9311 (Tunnel ID: 300): Tunnel300 Path Setup
[0000.0000.0000.00] 24.0.0.2: FULL_PATH (OSPF 13 area 0)
rrr_pcalc_show_path:9322 (Tunnel ID: 300): bw 64000, rev bw 64000, min_bw 0,
metric: 0
rrr_pcalc_show_path:9326 (Tunnel ID: 300): setup_pri 2, hold_pri 2
rrr_pcalc_show_path:9331 (Tunnel ID: 300): affinity_bits 0x0, affinity_mask
0xffff
rrr_pcalc_show_path:9335 (Tunnel ID: 300): path type unidirectional
```

```

rrr_pcalc_show_path:9340 (Tunnel ID: 300): endpoint switching capability psc1
rrr_pcalc_show_path:9343 (Tunnel ID: 300): endpoint encoding packet
rrr_pcalc_show_path:9347 (Tunnel ID: 300): transit switching capability psc1,
rrr_pcalc_show_path:9350 (Tunnel ID: 300): transit encoding packet
rrr_pcalc_reset_exclude_list:10099: TE-PCALC_SPF: ##### new gen 28
rrr_pcalc_path_to_lsr:11953 (Tunnel ID: 300): path_to_lsr (Loose Route after
explicit-2 Segment: 13.0.0.5, actual_hops 2
rrr_pcalc_path_to_lsr:11959 (Tunnel ID: 300): Addresses collected so far:
rrr_pcalc_path_to_lsr:11962 (Tunnel ID: 300): 13.11.12.11
rrr_pcalc_path_to_lsr:11962 (Tunnel ID: 300): 13.8.11.8
rrr_pcalc_path_to_lsr:11966 (Tunnel ID: 300): Addresses left
rrr_pcalc_path_to_lsr:11973 (Tunnel ID: 300): 13.0.0.5: Loose
rrr_pcalc_get_path_internal:9849 (Tunnel ID: 300): TE-PCALC_PATH:Path from
13.0.0.8 -> 13.0.0.5:
rrr_pcalc_get_path_internal:9883 (Tunnel ID: 300): TE-PCALC_PATH:
13.5.8.8->13.5.8.5 (admin_weight=1):
rrr_pcalc_get_path_internal:9918 (Tunnel ID: 300): TE-PCALC_PATH: Create a
path structure for new path

```

CSR5 receives the RSVP PATH message with 13.0.0.5 in the ERO and immediately removes it; this is CSR5's TE ID. CSR5 is unable to expand the hop for CSR6 so instead it leaves the ERO intact and sends the RSVP PATH message to CSR6 with 24.0.0.6 as the first loose-hop. As expected, the bandwidth reservation and TE priorities are maintained. The affinity is cleared as link colors are not expected to be consistent between ASes, much like IGP or TE metrics.

```

! CSR5
TE-PCALC-API: 13.0.0.12_10->24.0.0.6_300 {7}: LSP Path Expand called
TE-PCALC: 13.0.0.12_10->24.0.0.6_300 {7}: Path Request Info
  Flags:  END_SWCAP_UNKNOWN
  IP explicit-path: None (dynamic)
  bw 64000, min_bw 0, metric: 0
  setup_pri 2, hold_pri 2
  affinity bits 0x0, affinity mask 0x0
TE-PCALC-PATH: 13.0.0.12_10->24.0.0.6_300 {7}: Area (ospf 13 area 0) Path
Lookup begin
TE-PCALC-PATH: expand_lsr: Dst addr 24.0.0.6 not found in area (ospf 13 area
0)
TE-PCALC-PATH: 13.0.0.5_10->24.0.0.6_300 {7}: Area (ospf 13 area 0) Path
Lookup end: path not found
24.0.0.6 Can't Expand at this time
TE-PCALC-API: 13.0.0.5_10->24.0.0.6_300 {7}: LSP Path Expand result: failed

```

We can check this on CSR5 by examining the RSVP PATH details. The ERO is as I described it above, with CSR5 leaving 24.0.0.6 as the first loose-hop in the ERO for CSR6 to evaluate. Of note, there is no RSVP PATH RRO seen here. XR does not automatically enable this as XE does.

```

R5#show ip rsvp sender detail filter session-type 7 tunnel-id 300 | section
[ER]RO
  ERO: (incoming)
    13.5.8.5 (Strict IPv4 Prefix, 8 bytes, /32)
    13.0.0.5 (Strict IPv4 Prefix, 8 bytes, /32)
    24.0.0.6 (Loose IPv4 Prefix, 8 bytes, /32)
    24.0.0.2 (Loose IPv4 Prefix, 8 bytes, /32)
  ERO: (outgoing)
    10.5.6.6 (Strict IPv4 Prefix, 8 bytes, /32)
    24.0.0.6 (Loose IPv4 Prefix, 8 bytes, /32)
    24.0.0.2 (Loose IPv4 Prefix, 8 bytes, /32)

```

Because I would prefer to see the RSVP PATH RRO be evaluate by loose-hop expansion nodes, I quickly add it to XRv2's tunnel. Now, CSR5 sees this in the PATH message from CSR8 and passes it on to CSR6.

```

! XRv2
interface tunnel-te300
  record-route

```

```

R5#show ip rsvp sender detail filter session-type 7 tunnel-id 300 | section
[ER]RO
  ERO: (incoming)
    13.5.8.5 (Strict IPv4 Prefix, 8 bytes, /32)
    13.0.0.5 (Strict IPv4 Prefix, 8 bytes, /32)
    24.0.0.6 (Loose IPv4 Prefix, 8 bytes, /32)
    24.0.0.2 (Loose IPv4 Prefix, 8 bytes, /32)
  ERO: (outgoing)
    10.5.6.6 (Strict IPv4 Prefix, 8 bytes, /32)
    24.0.0.6 (Loose IPv4 Prefix, 8 bytes, /32)
    24.0.0.2 (Loose IPv4 Prefix, 8 bytes, /32)
  RRO:
    13.5.8.8/32, Flags:0x0 (No Local Protection)
    13.8.11.11/32, Flags:0x0 (No Local Protection)
    13.11.12.12/32, Flags:0x0 (No Local Protection)

```

We won't revisit CSR5's debug output after enabling RSVP PATH RRO since CSR6 shows it. CSR6 invokes the loose-hop expansion algorithm and checks the paths to the routers already in the RRO. This includes CSR5, CSR8, and XRv1, and XRv2. Since those are in another AS, it is impossible for a loop to form, so CSR6 continues with its calculation. CSR6 computes any dynamic path to CSR2 as XRv2 cannot control this with loose-hop expansion.

```

! CSR6
TE-PCALC-API: 13.0.0.12_11->24.0.0.2_300 {7}: LSP Path Expand called
TE-PCALC: 13.0.0.12_11->24.0.0.2_300 {7}: Path Request Info
  Flags:  END_SWCAP_UNKNOWN
  IP explicit-path: None (dynamic)
  bw 64000, min_bw 0, metric: 0

```

```

setup_pri 2, hold_pri 2
affinity bits 0x0, affinity_mask 0x0
TE-PCALC-PATH: 13.0.0.12_11->24.0.0.2_300 {7}: rrr_pcalc_lsr_expand: Can't
get router ID addr for 10.5.6.5
TE-PCALC-PATH: 13.0.0.12_11->24.0.0.2_300 {7}: rrr_pcalc_lsr_expand: Can't
get router ID addr for 13.5.8.8
TE-PCALC-PATH: 13.0.0.12_11->24.0.0.2_300 {7}: rrr_pcalc_lsr_expand: Can't
get router ID addr for 13.8.11.11
TE-PCALC-PATH: 13.0.0.12_11->24.0.0.2_300 {7}: rrr_pcalc_lsr_expand: Can't
get router ID addr for 13.11.12.12
TE-PCALC-PATH: 13.0.0.12_11->24.0.0.2_300 {7}: Area (isis level-2) Path
Lookup begin
TE-PCALC-PATH: exclude_path: system_id 0-0-0-0-0-0-0 not known!
TE-PCALC-PATH: exclude_path: system_id 0-0-0-0-0-0-0 not known!
TE-PCALC-PATH: exclude_path: system_id 0-0-0-0-0-0-0 not known!
TE-PCALC-PATH: exclude_path: system_id 0-0-0-0-0-0-0 not known!
TE-PCALC-PATH: Path from 0000.0000.0006.00 -> 0000.0000.0002.00:
    24.2.14.14->24.2.14.2 (admin_weight=20):
    24.6.14.6->24.6.14.14 (admin_weight=10):
    num_hops 3, accumulated_aw 20, min_bw 136000
TE-PCALC-PATH: 24.0.0.6_11->24.0.0.2_300 {7}: Area (isis level-2) Path
Lookup end: path found
24.0.0.2 expands to:
24.6.14.14
24.2.14.2
24.0.0.2
TE-PCALC-API: 24.0.0.6_11->24.0.0.2_300 {7}: LSP Path Expand result: success

```

As the tail end, CSR2 sees the tunnel as up, along with the full RSVP PATH RRO. The tail end can clearly see the entire reverse path along the LSP, and since the headend request RSVP RESV RRO as well, CSR2 adds itself to that list so the headend has the same awareness. With loose-hop expansion, this is valuable for both head and tail routers.

```

R2#show mpls traffic-eng tunnels role tail
P2P TUNNELS/LSPs:
LSP Tunnel XRv2_t300 is signalled, connection is up
  InLabel : GigabitEthernet2.524, implicit-null
  Prev Hop : 24.2.14.14
  OutLabel : -
  RSVP Signalling Info:
    Src 13.0.0.12, Dst 24.0.0.2, Tun_Id 300, Tun_Instance 11
  RSVP Path Info:
    My Address: 24.0.0.2
    Explicit Route: NONE
    Record Route: 24.2.14.14 24.6.14.6 10.5.6.5 13.5.8.8
                  13.8.11.11 13.11.12.12
    Tspec: ave rate=64000 kbits, burst=1000 bytes, peak rate=64000 kbits
  RSVP Resv Info:

```

Record Route: NONE

Fspec: ave rate=64000 kbits, burst=1000 bytes, peak rate=64000 kbits

RP/0/0/CPU0:XRv2#show mpls traffic-eng tunnels 300 detail | begin Resv

Resv Info:

Record Route:

IPv4 13.11.12.11, flags 0x0

IPv4 13.5.8.8, flags 0x0

IPv4 10.5.6.5, flags 0x0

IPv4 24.6.14.6, flags 0x0

IPv4 24.6.14.14, flags 0x0

IPv4 24.2.14.2, flags 0x0

Fspec: avg rate=64000 kbits, burst=1000 bytes, peak rate=64000 kbits

Displayed 1 (of 1) heads, 0 (of 0) midpoints, 0 (of 0) tails

Displayed 1 up, 0 down, 0 recovering, 0 recovered heads

A quick MPLS traceroute confirms that the tunnel is operational between ASes. Notice the first label is 91014 which tunnels traffic through XRv1.

RP/0/0/CPU0:XRv2#traceroute mpls traffic-eng tunnel-te 300 source 13.0.0.12

Tracing MPLS TE Label Switched Path on tunnel-te300, timeout is 2 seconds

[snip]

Type escape sequence to abort.

```
0 13.11.12.12 MRU 1500 [Labels: 91014 Exp: 0]
L 1 13.11.12.11 MRU 1500 [Labels: 8030 Exp: 0] 0 ms
L 2 13.8.11.8 MRU 1500 [Labels: 5024 Exp: 0] 10 ms
L 3 13.5.8.5 MRU 1500 [Labels: 6030 Exp: 0] 0 ms
L 4 10.5.6.6 MRU 1500 [Labels: 94022 Exp: 0] 10 ms
L 5 24.6.14.14 MRU 1500 [Labels: implicit-null Exp: 0] 10 ms
! 6 24.2.14.2 10 ms
```

Next, XRv2 needs to route traffic into the tunnel. Since autoroute announce and forwarding adjacency do not work design-wise, and autoroute destination is not supported on XRv, we resort to a static route. We already foresee that this will break MVPN for the EIGRP customer as it did on CSR2. The RPF interface shows as connected with a distance of 1 and metric of 0, which hints to a static route. The unicast route was imported into the MRIB and is currently breaking MVPN traffic entering XRv2 from CSR2.

```
! XRv2
router static
 address-family ipv4 unicast
  24.0.0.2/32 tunnel-te300
```

RP/0/0/CPU0:XRv2#show pim rpf 24.0.0.2

Table: IPv4-Multicast-default

* 24.0.0.2/32 [1/0] [Connected]

```
via Null with rpf neighbor 0.0.0.0
```

```
RP/0/0/CPU0:XRv2#show route ipv4 multicast static  
S 24.0.0.2/32 is directly connected, 00:01:00, tunnel-te300
```

Since AS 13 is using a BGP-free “core” (if core routers existed here), BGP is already running between the PEs and ASBRs. We could run BGP IPv4 multicast to exchange RPF routes, but that would also involve AS 24. Because the AS 13 ASBRs learn the remote PEs via BGP, we cannot use a network statement on them to leak the route from unicast to multicast BGP. We would have to originate the route into BGP multicast on CSR6 and CSR7, which makes the configuration rather involved. For brevity, I use a multicast static route on XRv2 as well so that the focus of this lab remains on MPLS-TE, not MVPN.

```
! XRv2  
router static  
address-family ipv4 multicast  
24.0.0.2/32 13.8.12.8
```

```
RP/0/0/CPU0:XRv2#show pim rpf 24.0.0.2  
Table: IPv4-Multicast-default  
* 24.0.0.2/32 [1/0]  
via GigabitEthernet0/0/0/0.582 with rpf neighbor 13.8.12.8
```

Next, I will test basic L3VPN from CSR2 to CSR1. With BGP labeled-unicast running in AS 13, this used to require a 3-label stack. Now, the entire LSP is 2 labels since there is a PE-PE tunnel. The first transport label is 91014 which was verified earlier using OAM. This confirms that traffic is inside the TE LSP.

```
R3#traceroute 10.1.1.1 source 10.3.3.3  
Type escape sequence to abort.  
Tracing the route to 10.1.1.1  
VRF info: (vrf in name/id, vrf out name/id)  
 1 10.3.12.12 4 msec 3 msec 1 msec  
 2 13.11.12.11 [MPLS: Labels 91014/2018 Exp 0] 11 msec 11 msec 11 msec  
 3 13.8.11.8 [MPLS: Labels 8030/2018 Exp 0] 23 msec 31 msec 31 msec  
 4 13.5.8.5 [MPLS: Labels 5024/2018 Exp 0] 29 msec 31 msec 32 msec  
 5 10.5.6.6 [MPLS: Labels 6030/2018 Exp 0] 60 msec 17 msec 16 msec  
 6 24.6.14.14 [MPLS: Labels 94022/2018 Exp 0] 30 msec 31 msec 36 msec  
 7 10.1.2.2 [MPLS: Label 2018 Exp 0] 17 msec 16 msec 15 msec  
 8 10.1.2.1 19 msec 11 msec 10 msec
```

I also test multicast from CSR3 to XRv3 again. This works as well; a design consideration of inter-AS TE is that it requires RPF fixup methods when supporting multicast transport for VPN customers.

```
R3#ping ip  
Target IP address: 225.13.13.13  
Repeat count [1]: 10000  
Datagram size [100]:
```



```

Timeout in seconds [2]: 1
Extended commands [n]: y
Interface [All]: loopback0
Time to live [255]:
Source address or interface: loopback0

RP/0/0/CPU0:Xshow mfib route 225.13.13.13 10.3.3.3 | begin 225
(10.3.3.3,225.13.13.13),    Flags:
  Up: 00:00:23
  Last Used: 00:00:00
  SW Forwarding Counts: 23/23/2300
  SW Replication Counts: 23/0/0
  SW Failure Counts: 0/0/0/0/0
  Loopback0 Flags:  IC NS EG, Up:00:00:23
  GigabitEthernet0/0/0/0.513 Flags:  A, Up:00:00:23

```

I will shut down tunnel300 on XRv2 to create a verbatim path tunnel on XR. This tunnel will be PE-P and will terminate on XRv4 rather than CSR2. This implies we need some kind of mechanism to learn XRv4's label for 24.0.0.2/32 as XRv2 will need to push this onto the label stack. If it doesn't, the MPLS service label allocated by CSR2 will be revealed to XRv4 and traffic will be dropped. We can technically use BGP labeled-unicast for this (as seen in the U MPLS section), but using LDP would be simpler. The verbatim configuration is almost identical to XE.

```

! XRv2
explicit-path name EP_12_8_5_6_7_14_VERBATIM
  index 10 next-address strict ipv4 unicast 13.0.0.8
  index 20 next-address strict ipv4 unicast 13.0.0.5
  index 30 next-address strict ipv4 unicast 10.5.6.6
  index 40 next-address strict ipv4 unicast 24.6.7.7
  index 50 next-address strict ipv4 unicast 24.7.14.14
  index 60 next-address strict ipv4 unicast 24.0.0.14

interface tunnel-te301
  description INTER-AS VERBATIM
  ipv4 unnumbered Loopback0
  priority 6 5
  signalled-bandwidth 32000
  destination 24.0.0.14
  record-route
  path-option 10 explicit name EP_12_8_5_6_7_14_VERBATIM verbatim

```

With debugging still enabled on XRv2, we can observe the PCALC process. XR sees that the path-option is "verbatim" which bypasses the TED consistency checks. The bandwidth reservation and TE priorities remain intact. Link coloring (affinity constraint) is also enforced as evidenced by the non-zero mask.

```
! XRv2
```

```

te_vif_get_excluded_addresses:24713 (Tunnel ID: 301): Getting the excluded
addr for path setup from previous PERRs
rrr_pcalc_get_path:15390 (Tunnel ID: 301): TE-PCALC_PATH: Verbatim enabled.
No topology check
rrr_pcalc_show_path:9311 (Tunnel ID: 301): Tunnel301 Path Setup
[0000.0000.0000.00] 24.0.0.14: FULL_PATH NO_TOPOLOGY_CHECK (No IGP area nil)
rrr_pcalc_show_path:9322 (Tunnel ID: 301): bw 32000, rev bw 32000, min_bw 0,
metric: 0
rrr_pcalc_show_path:9326 (Tunnel ID: 301): setup_pri 6, hold_pri 5
rrr_pcalc_show_path:9331 (Tunnel ID: 301): affinity_bits 0x0, affinity_mask
0xffff
[snip]

```

Skipping to CSR5, we can see the incoming and outgoing ERO, as well as the RRO. CSR5 removes its own address (as a TE ID) and sends the PATH to CSR6 who will forward the path based on TE link addresses. The RRO contains XRv2 and CSR8 so far as it records the reverse path to help prevent loops. This is mostly useful for loose-hop expansion as intermediate routers will run PCALC expansion, but verbatim midpoints will not.

```

R5#show ip rsvp sender detail filter session-type 7 tunnel-id 301 | section
[ER]RO
  ERO: (incoming)
    13.0.0.5 (Strict IPv4 Prefix, 8 bytes, /32)
    10.5.6.6 (Strict IPv4 Prefix, 8 bytes, /32)
    24.6.7.7 (Strict IPv4 Prefix, 8 bytes, /32)
    24.7.14.14 (Strict IPv4 Prefix, 8 bytes, /32)
    24.0.0.14 (Strict IPv4 Prefix, 8 bytes, /32)
  ERO: (outgoing)
    10.5.6.6 (Strict IPv4 Prefix, 8 bytes, /32)
    24.6.7.7 (Strict IPv4 Prefix, 8 bytes, /32)
    24.7.14.14 (Strict IPv4 Prefix, 8 bytes, /32)
    24.0.0.14 (Strict IPv4 Prefix, 8 bytes, /32)
  RRO:
    13.5.8.8/32, Flags:0x0 (No Local Protection)
    13.8.12.12/32, Flags:0x0 (No Local Protection)

```

XRv4 sees the tunnel as operational along with the reverse path back to XRv2. The bandwidth reservation was also maintained, as expected.

```

RP/0/0/CPU0:XRv4#show mpls traffic-eng tunnels role tail brief
          TUNNEL NAME          DESTINATION      STATUS  STATE
          XRv2_t301           24.0.0.14        up      up
Displayed 0 (of 0) heads, 0 (of 0) midpoints, 1 (of 1) tails
Displayed 0 up, 0 down, 0 recovering, 0 recovered heads

RP/0/0/CPU0:XRv4#show mpls traffic-eng tunnels role tail | begin Record
Record Route:

```

```

IPv4 24.7.14.7, flags 0x0
IPv4 24.6.7.6, flags 0x0
IPv4 10.5.6.5, flags 0x0
IPv4 13.5.8.8, flags 0x0
IPv4 13.8.12.12, flags 0x0
Tspec: avg rate=32000 kbits, burst=1000 bytes, peak rate=32000 kbits

```

A quick traceroute reveals that the LSP is operational in the data plane. Traffic is sent through the proper routers in accordance with the verbatim path. Notice the first label of 8030 which is allocated by CSR8 along this TE LSP. This should be the top most label for all traffic from XRv2 going into the tunnel.

```

RP/0/0/CPU0:XRv2#traceroute mpls traffic-eng tunnel-te 301 source 13.0.0.12
force-explicit-null
Tracing MPLS TE Label Switched Path on tunnel-te301, timeout is 2 seconds
[snip]
Type escape sequence to abort.

```

```

0 13.8.12.12 MRU 1500 [Labels: 8030/explicit-null Exp: 0/0]
L 1 13.8.12.8 MRU 1500 [Labels: 5072/explicit-null Exp: 0/0] 0 ms
L 2 13.5.8.5 MRU 1500 [Labels: 6056/explicit-null Exp: 0/0] 10 ms
L 3 10.5.6.6 MRU 1500 [Labels: 7084/explicit-null Exp: 0/0] 10 ms
L 4 24.6.7.7 MRU 1500 [Labels: explicit-null Exp: 0] 0 ms
! 5 24.7.14.14 20 ms

```

When we add a static route for this tunnel, VPN traffic between EIGRP customers breaks. The eBGP session remains up, which means IPv4 connectivity is maintained at least. Traceroute in the global table reveals the problem; the last hop between XRv4 and CSR2 is unlabeled. This allows IPv4 to continue working simply by virtue of XRv4 being able to receive IP packets for 24.0.0.2/32 and route them appropriately.

```

! XRv2
router static
address-family ipv4 unicast
24.0.0.2/32 tunnel-te301

```

```

R3#traceroute 10.1.1.1 source 10.3.3.3
Type escape sequence to abort.
Tracing the route to 10.1.1.1
VRF info: (vrf in name/id, vrf out name/id)
 1 10.3.12.12 3 msec 2 msec 2 msec
 2 * * *
 3 * * *

```

```

RP/0/0/CPU0:XRv2#traceroute 24.0.0.2 source 13.0.0.12
Type escape sequence to abort.
Tracing the route to 24.0.0.2

```

```

1 13.8.12.8 [MPLS: Label 8030 Exp 0] 9 msec 0 msec 0 msec
2 13.5.8.5 [MPLS: Label 5072 Exp 0] 0 msec 0 msec 0 msec
3 10.5.6.6 [MPLS: Label 6056 Exp 0] 9 msec 0 msec 0 msec
4 24.6.7.7 [MPLS: Label 7084 Exp 0] 0 msec 0 msec 0 msec
5 24.7.14.14 0 msec 0 msec 0 msec
6 24.2.14.2 0 msec 0 msec 0 msec

```

Checking XRv2's VRF-aware FIB, we see a dysfunctional label stack. The VPN label of 2018 is imposed but additional labels are not.

```

RP/0/0/CPU0:XRv2#show cef vrf EIGRP 10.1.1.1/32
10.1.1.1/32, version 392, internal 0x5000001 0x0 (ptr 0xa146a8f4) [1], 0x0
(0x0), 0x208 (0xa1553578)
Prefix Len 32, traffic index 0, precedence n/a, priority 3
  via 24.0.0.2, 5 dependencies, recursive, bgp-ext [flags 0x6020]
    path-idx 0 NHID 0x0 [0xa15bbd74 0x0]
    recursion-via-/32
    next hop VRF - 'default', table - 0xe0000000
    next hop 24.0.0.2 via 92502/0/21
    next hop 0.0.0.0/32 tt301 labels imposed {ImplNull 2018}

```

Because the tunnel destination is not the same as the traffic we are routing into it, traffic is dropped at XRv4. The VPN label allocated by CSR2 is being exposed to XRv4 who cannot label-switch it. Though 24.0.0.2/32 is learned via a TE tunnel, we cannot push the TE label just yet. XRv2 must learn XRv4's LDP label for CSR2 since XRv4 has an IGP route for 24.0.0.2/32. We can accomplish this with tLDP on the tunnel. XRv2 will enable LDP on the TE tunnel and XRv4 will permit XRv2 (and no others) to form dynamic tLDP sessions. This is a good security practice when using inter-AS TE tunnels and tLDP between core routes (very rare in real life).

```

! XRv4
ipv4 access-list ACL_XRV2
 10 permit ipv4 host 13.0.0.12 any

mpls ldp
  address-family ipv4
    discovery targeted-hello accept from ACL_XRV2

! XRv2
mpls ldp
  interface tunnel-te301
    address-family ipv4

```

An LDP neighbor will form between XRv2 and XRv4 as a result. This might be forbidden by company policy as LDP peers are generally not negotiated between ASes. We bent this rule earlier using mLDP at the ASBRs, but in this case, these could potentially be core routers. Like the ASBRs, these routers will also exchange all labels with one another by default. There is only reason for XRv4 to advertise a single

label to XRv2, and there is no reason for XRv2 to advertise any labels to XRv4. I will apply some filters for cleanup so that the TE tunnel has a minimal affect on XRv2 and XRv4.

```
! XRv2
ipv4 access-list ACL_DENY
 10 deny ipv4 any any

mpls ldp
 address-family ipv4
  label
  local
  advertise
    to 24.0.0.14:0 for ACL_DENY

! XRv4
ipv4 access-list ACL_CSR2_LOOPBACK
 10 permit ipv4 host 24.0.0.2 any

mpls ldp
 address-family ipv4
  label
  local
  advertise
    to 13.0.0.12:0 for ACL_CSR2_LOOPBACK
```

We quickly verify that the label filtering worked as expected. This will help contain labels within an AS and increase security in general.

```
RP/0/0/CPU0:XRv4#show mpls ldp bindings neighbor 13.0.0.12
```

[no output]

```
RP/0/0/CPU0:XRv2#show mpls ldp bindings neighbor 24.0.0.14
```

```
24.0.0.2/32, rev 32
  Local binding: label: 92502
  Remote bindings: (2 peers)
    Peer          Label
    -----
    24.0.0.14:0   94009
```

With this LDP label of 94009, XRv2 now has the correct label stack (not including the TE label) to account for the LDP label for 24.0.0.2/32 from XRv4 (remote PE) and the VPN label for 10.1.1.1/32 (final destination). The TE tunnel will deliver traffic to the tunnel endpoint, which is XRv4. Earlier we saw the topmost label allocated by CSR8 along the TE LSP was 8030.

```
RP/0/0/CPU0:XRv2#show cef vrf EIGRP 10.1.1.1/32
10.1.1.1/32, version 392, internal 0x5000001 0x0 (ptr 0xa146a8f4) [1], 0x0
(0x0), 0x208 (0xa1553578)
```

```

Prefix Len 32, traffic index 0, precedence n/a, priority 3
via 24.0.0.2, 5 dependencies, recursive, bgp-ext [flags 0x6020]
path-idx 0 NHID 0x0 [0xa15bbd74 0x0]
recursion-via-/32
next hop VRF - 'default', table - 0xe0000000
next hop 24.0.0.2 via 92502/0/21
next hop 0.0.0.0/32 tt301 labels imposed {94009 2018}

```

Using traceroute in the VPN, we can confirm this fixed the problem. Traffic is inside the TE tunnel as the first label is 8030, and the 3 label stack is needed all the way up until XRv4. XRv4 now receives label 94009 which allows it to forward traffic towards 24.0.0.2/32. This topic is discussed in greater detail in the TE section as well.

```

R3#traceroute 10.1.1.1 source 10.3.3.3
Type escape sequence to abort.
Tracing the route to 10.1.1.1
VRF info: (vrf in name/id, vrf out name/id)
 1 10.3.12.12 5 msec 1 msec 2 msec
 2 13.8.12.8 [MPLS: Labels 8030/94009/2018 Exp 0] 10 msec 11 msec 11 msec
 3 13.5.8.5 [MPLS: Labels 5072/94009/2018 Exp 0] 31 msec 31 msec 33 msec
 4 10.5.6.6 [MPLS: Labels 6056/94009/2018 Exp 0] 31 msec 148 msec 39 msec
 5 24.6.7.7 [MPLS: Labels 7084/94009/2018 Exp 0] 31 msec 33 msec 34 msec
 6 24.7.14.14 [MPLS: Labels 94009/2018 Exp 0] 21 msec 20 msec 24 msec
 7 10.1.2.2 [MPLS: Label 2018 Exp 0] 20 msec 21 msec 21 msec
 8 10.1.2.1 22 msec 12 msec 12 msec

```

Last, we will quickly test multicast. The RPF multicast static routes are still valid from earlier (assuming it is still enabled), so I ping from CSR3 to check for packet reception at XRv3. This proves that the MVPN is still function with the inter-AS, PE-P tunnel with tLDP on top.

```

R3#ping ip
Target IP address: 225.13.13.13
Repeat count [1]: 10000
Datagram size [100]:
Timeout in seconds [2]: 1
Extended commands [n]: y
Interface [All]: loopback0
Time to live [255]:
Source address or interface: loopback0

```

```

RP/0/0/CPU0:XRv3#show mfib route 225.13.13.13 10.3.3.3 | begin 225
(10.3.3.3,225.13.13.13),   Flags:
  Up: 00:00:31
  Last Used: 00:00:00
  SW Forwarding Counts: 31/31/3100
  SW Replication Counts: 31/0/0
  SW Failure Counts: 0/0/0/0/0

```

```
Loopback0 Flags:  IC NS EG, Up:00:00:31
GigabitEthernet0/0/0/0.513 Flags:  A, Up:00:00:31
```

As a final cleanup, I shut down all TE tunnels on XRv2 and remove the static multicast route. XR does not allow me to set the AD to 255. Shutting the tunnels down allows them to remain readily accessible for testing later. I confirm that the unicast route is iBGP learned and the RPF is derived from the unicast route. This returns the option C network to its intended architecture.

```
RP/0/0/CPU0:XRv2#show mpls traffic-eng tunnels role head brief
      TUNNEL NAME      DESTINATION      STATUS  STATE
      tunnel-te300      24.0.0.2         down    down
      tunnel-te301      24.0.0.14        down    down
Displayed 2 (of 2) heads, 0 (of 0) midpoints, 0 (of 0) tails
Displayed 0 up, 2 down, 0 recovering, 0 recovered heads
```

```
RP/0/0/CPU0:XRv2#show route 24.0.0.2
Routing entry for 24.0.0.2/32
  Known via "bgp 13", distance 200, metric 20, [ei]-bgp
  Tag 24, type internal
  Routing Descriptor Blocks
    10.6.11.6, from 13.0.0.11
      Route metric is 20
  No advertising protos.
```

```
RP/0/0/CPU0:XRv2#show pim rpf 24.0.0.2
Table: IPv4-Multicast-default
* 24.0.0.2/32 [200/20]
  via GigabitEthernet0/0/0/0.582 with rpf neighbor 13.8.12.8
```

8.4.3.6 Confederation variation

Confederations with inter-AS option C are similar to option B in that MPLS traffic exchange is expected to happen. The same considerations with respect to XR's "mpls activate" for multi-hop peers and XE's "mpls bgp forwarding" for all eBGP/intraconfederation peers still apply. Below is the basic BGP confederation configuration that is applied to all peers. This builds a confederation between AS 13 and 24, and adjusts CSR10 to peer with AS 42518, which is the confederation ASN. This configuration is identical to options A and B since it is not specific to MPLS VPN service with one exception. Since CSR2 and XRv2 peer directly, they must also identify the confederation peers. XRv4 also peers with XRv2 within the L2VPN VPLS AFI only. In this design, only CSR8 doesn't require the confederation peer list configuration.

```
! CSR2, CSR6, CSR7, and XRv4
router bgp 24
  bgp confederation identifier 42518
  bgp confederation peers 13

! CSR8
```

```

router bgp 13
  bgp confederation identifier 42518

! CSR5, XRv1, and XRv2
router bgp 13
  bgp confederation identifier 42518
  bgp confederation peers 24

! CSR10
router bgp 100
  neighbor 10.8.10.8 remote-as 42518
  neighbor FD00:10:8:10::8 remote-as 42518

```

In a textbook option C design, only BGP labeled-unicast is exchanged between the ASBRs. In this design, the IPv4 flavor of that feature is used, and we can verify these peers on CSR5 and CSR6. This proves that the confederation configuration was successful. CSR5 has an iBGP connections as well since sub-AS 13 is using labeled-unicast for transport rather than redistributing remote loopbacks into IGP.

```

R5#show bgp ipv4 unicast summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
10.5.6.6      4      24      18      19      21    0    0 00:06:17      3
10.5.7.7      4      24      19      22      21    0    0 00:06:17      3
13.0.0.12     4      13      49      51      21    0    0 00:05:50      4

```

```

R6#show bgp ipv4 unicast summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
10.5.6.5      4      13      19      18      30    0    0 00:06:25      4
10.6.11.11    4      13      16      17      30    0    0 00:06:00      4

```

One interesting observation is that when labeled-unicast is configured on an XE router, next-hop-self is assumed. Looking at CSR6, it learns two routes for 13.0.0.12/32, which is a remote PE. The route via XRv1 has the next-hop unchanged, but CSR5 changed the next-hop to self without having been configured for it.

```

R6#show bgp ipv4 unicast 13.0.0.8/32
BGP routing table entry for 13.0.0.8/32, version 108
Paths: (2 available, best #2, table default)
  Not advertised to any peer
  Refresh Epoch 1
  (13)
    10.5.6.5 from 10.5.6.5 (13.0.0.5)
      Origin IGP, metric 0, localpref 100, valid, confed-external
      mpls labels in/out nlabel/5004
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  (13)
    13.0.0.8 (metric 20) from 10.6.11.11 (13.0.0.11)

```



```
Origin IGP, metric 0, localpref 100, weight 6666, valid, confed-external, best
mpls labels in/out nolabel/imp-null
rx pathid: 0, tx pathid: 0x0
```

For some prefixes, this contributes to a routing loop with churning every few seconds. The IS-IS route has a better AD, so it beats the confed-external route's distance of 200. Normally this would be stable, but since CSR6 and CSR7 have the same problem (that is, they learn one another's IGP routes, which rely on the BGP routes first), the IS-IS route is withdraw regularly. At some point, the network may stabilize, but this is a major problem.

```
R6#debug ip routing
IP routing debugging is on
```

```
19:51:52.892: RT: closer admin distance for 13.0.0.12, flushing 1 routes
19:51:52.892: RT(multicast): delete subnet route to 13.0.0.12/32
19:51:52.892: RT: add 13.0.0.12/32 via 24.6.7.7, isis metric [115/20]
19:51:52.897: RT: rib update return code: 17
```

```
19:51:57.893: RT: del 13.0.0.12 via 24.6.7.7, isis metric [115/20]
19:51:57.893: RT: delete subnet route to 13.0.0.12/32
```

```
R6#show ip route 13.0.0.12
Routing entry for 13.0.0.12/32
  Known via "bgp 24", distance 200, metric 0
  Tag 13, type internal
  Redistributing via isis 24
  Advertised by isis 24 metric-type internal level-2 route-map RM_BGP_TO_ISIS
  Last update from 13.0.0.12 00:00:05 ago
  Routing Descriptor Blocks:
  * 13.0.0.12, from 10.6.11.11, 00:00:05 ago
    Route metric is 0, traffic share count is 1
    AS Hops 0
    Route tag 13
    MPLS label: implicit-null
```

```
R6#show ip route 13.0.0.12
Routing entry for 13.0.0.12/32
  Known via "isis", distance 115, metric 20, type level-2
  Redistributing via isis 24, bgp 24
  Last update from 24.6.7.7 on GigabitEthernet2.567, 00:00:01 ago
  Routing Descriptor Blocks:
  * 24.6.7.7, from 24.0.0.7, 00:00:01 ago, via GigabitEthernet2.567
    Route metric is 20, traffic share count is 1
```

I quickly update XRv1 to set the next-hop to self for the labeled-unicast AFI. I also take this opportunity to remove any unnecessary filters as the intraconfederation peers do not require it. The outbound RPL is valuable for loopback advertisement control, so I do not remove it.

```
! XRv1
router bgp 13
 neighbor 10.6.11.6
   address-family ipv4 labeled-unicast
   no route-policy RPL_PASS in
   ! route-policy RPL_IF_DEST_PASS(PS_LOCAL_LOOPBACKS) out
   next-hop-self
```

The loop has ended, but we still have a suboptimal routing problem. The root of the problem is more focused on the administrative distance differences. CSR7 prefers the BGP route with AD 200 and redistribute it into IGP before CSR6 runs its BGP bestpath process. CSR6, therefore, must prefer the IGP route via CSR7. This is another reason to not perform BGP/IGP redistribution and to use BGP labeled-unicast everywhere; AS 13 will never have this problem. Earlier, AS 24 never had this problem since the BGP distance for external routes was 20, which was always less than 115 for IS-IS.

```
R7#show ip route 13.0.0.12
Routing entry for 13.0.0.12/32
  Known via "bgp 24", distance 200, metric 0
  Tag 13, type internal
  Redistributing via isis 24
  Advertised by isis 24 metric-type internal level-2 route-map RM_BGP_TO_ISIS
  Last update from 10.5.7.5 00:10:10 ago
  Routing Descriptor Blocks:
  * 10.5.7.5, from 10.5.7.5, 00:10:10 ago
    Route metric is 0, traffic share count is 1
    AS Hops 0
    Route tag 13
    MPLS label: 5002
```

```
R6#show ip route 13.0.0.12
Routing entry for 13.0.0.12/32
  Known via "isis", distance 115, metric 20, type level-2
  Redistributing via isis 24, bgp 24
  Last update from 24.6.7.7 on GigabitEthernet2.567, 00:10:00 ago
  Routing Descriptor Blocks:
  * 24.6.7.7, from 24.0.0.7, 00:10:00 ago, via GigabitEthernet2.567
    Route metric is 20, traffic share count is 1
```

This problem can be fixed in two straightforward ways: lower the AD of the confed-external BGP routes representing remote loopbacks, or raise the AD of the IS-IS routes representing the remote loopbacks. CSR6 will use the first approach and CSR7 will use the second, but the effect is the same. First, I configure CSR6. This technique identifies any peer with address in the 10.0.0.0/8 range who advertises

loopbacks from 13.0.0.0/24, and sets the AD to 30. Note that BGP appears to require a “hard reset” for this distance modification to take effect. While this works, CSR7 is now preferring the IGP route via CSR6, which is the same problem. We just reversed the suboptimal routing path.

```
! CSR6
ip access-list standard ACL_REMOTE_LOOPBACKS
 permit 13.0.0.0 0.0.0.255

router bgp 24
 address-family ipv4
   distance 30 10.0.0.0 0.255.255.255 ACL_REMOTE_LOOPBACKS

R6#show ip route 13.0.0.12
Routing entry for 13.0.0.12/32
  Known via "bgp 24", distance 30, metric 0
  Tag 13, type internal
  Redistributing via isis 24
  Advertised by isis 24 metric-type internal level-2 route-map RM_BGP_TO_ISIS
  Last update from 10.6.11.11 00:00:09 ago
  Routing Descriptor Blocks:
  * 10.6.11.11, from 10.6.11.11, 00:00:09 ago
    Route metric is 0, traffic share count is 1
    AS Hops 0
    Route tag 13
    MPLS label: 91002

R7#show ip route 13.0.0.12
Routing entry for 13.0.0.12/32
  Known via "isis", distance 115, metric 25, type level-2
  Redistributing via isis 24
  Last update from 24.6.7.6 on GigabitEthernet2.567, 00:00:45 ago
  Routing Descriptor Blocks:
  * 24.6.7.6, from 24.0.0.6, 00:00:45 ago, via GigabitEthernet2.567
    Route metric is 25, traffic share count is 1
```

Configuring CSR7 with a similar approach fixes the problem entirely. The source/wildcard specification is kept generic to match any internally-generated IS-IS routes for remote loopbacks. Now, both CSR6 and CSR7 prefer confed-external BGP routes.

```
! CSR7
ip access-list standard ACL_REMOTE_LOOPBACKS
 permit 13.0.0.0 0.0.0.255

router isis 24
 distance 230 24.0.0.0 0.255.255.255 ACL_REMOTE_LOOPBACKS

R7#show ip route 13.0.0.12
Routing entry for 13.0.0.12/32
```

```

Known via "bgp 24", distance 200, metric 0
Tag 13, type internal
Redistributing via isis 24
Advertised by isis 24 metric-type internal level-2 route-map RM_BGP_TO_ISIS
Last update from 10.5.7.5 00:00:39 ago
Routing Descriptor Blocks:
* 10.5.7.5, from 10.5.7.5, 00:00:39 ago
  Route metric is 0, traffic share count is 1
  AS Hops 0
  Route tag 13
  MPLS label: 5002

```

```

R6#show ip route 13.0.0.12
Routing entry for 13.0.0.12/32
Known via "bgp 24", distance 30, metric 0
Tag 13, type internal
Redistributing via isis 24
Advertised by isis 24 metric-type internal level-2 route-map RM_BGP_TO_ISIS
Last update from 10.6.11.11 00:18:34 ago
Routing Descriptor Blocks:
* 10.6.11.11, from 10.6.11.11, 00:18:34 ago
  Route metric is 0, traffic share count is 1
  AS Hops 0
  Route tag 13
  MPLS label: 91002

```

A quick way to verify the AD adjustments on both routers, as a final sanity check, is to check the IP protocol details. The output also shows the basic confederation details, such as peer ASNs and the confederation ASN. This reveals the default admin-distances for each protocol, along with the customizations on a per-entry basis. CSR6 adjusted BGP distance while CSR7 adjusted IS-IS distance.

```

R6#show ip protocols | begin ^Routing.*bgp
Routing Protocol is "bgp 24"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  BGP confederation consists of ASs: 13
  BGP confederation advertised as AS 42518
  IGP synchronization is disabled
  Automatic route summarization is disabled
  Redistributing: isis 24
  Neighbor(s):
    Address           FiltIn FiltOut DistIn DistOut Weight RouteMap
    10.5.6.5
    10.6.11.11        6666
  Maximum path: 1
  Routing Information Sources:
    Gateway           Distance      Last Update
    10.6.11.11        30           00:19:50

```

Address	Wild mask	Distance	List
10.0.0.0	0.255.255.255	30	ACL_REMOTE_LOOPBACKS

```
R7#show ip protocols | begin ^Routing.*isis
Routing Protocol is "isis 24"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Redistributing: isis 24, bgp 24
  Address Summarization:
    None
  Maximum path: 4
  Routing for Networks:
    GigabitEthernet2.527
    GigabitEthernet2.567
    GigabitEthernet2.574
  Passive Interface(s):
    Loopback0
    Lspvif0
    Lspvif1
  Routing Information Sources:
    Gateway          Distance      Last Update
    24.0.0.14         230          00:02:36
    24.0.0.2          230          00:02:36
    24.0.0.6          230          00:02:36
```

Address	Wild mask	Distance	List
24.0.0.0	0.255.255.255	230	ACL_REMOTE_LOOPBACKS

Now that the routing is stable, I will perform some extra clean-up on the PEs. They are already configured with the proper confederation information, but many filters can be removed. The RT rewrite capabilities are still needed, but next-hop-unchanged is not. This was a critical component of the option C design but the next-hops will be preserved by default within the confederation, so this is technically not necessary. Of note, applying the confederation peer configuration to CSR2 automatically removes the next-hop-unchanged since it is only valid for true eBGP. In XR, we must do it manually.

```
! XRv2
router bgp 13
  neighbor 24.0.0.2
    address-family vpnv4 unicast
      no route-policy RPL_PASS in
      no next-hop-unchanged
      ! route-policy RPL_RT_REWRITE out

  address-family vpnv6 unicast
    no route-policy RPL_PASS in
    no next-hop-unchanged
    ! route-policy RPL_RT_REWRITE out
```

```

address-family ipv4 mvpn
  no route-policy RPL_PASS in
  no next-hop-unchanged
  ! route-policy RPL_RT_REWRITE out

address-family ipv6 mvpn
  no route-policy RPL_PASS in
  no next-hop-unchanged
  ! route-policy RPL_RT_REWRITE out

```

I also clean up the BGP-based VPLS configurations between XRv2 and XRv4. XRv4 is an RR for this AFI within sub-AS 24 since CSR2 was exhibiting undesirable label-block behavior as seen earlier. XRv2 continues to perform RT rewrite, but all other filters are removed.

```

! XRv2
router bgp 13
  neighbor 24.0.0.14
  address-family l2vpn vpls-vpws
    no route-policy RPL_PASS in
    ! route-policy RPL_RT_REWRITE out
    no next-hop-unchanged

! XRv4
router bgp 24
  neighbor 13.0.0.12
  address-family l2vpn vpls-vpws
    no route-policy RPL_PASS in
    no route-policy RPL_PASS out
    no next-hop-unchanged

```

Cisco XE 3.13.2S crashes continuously when trying to run confed-external peerings to XRv2. As soon as I “no shutdown” the BGP peer from XRv2’s perspective, CSR2 crashes shortly thereafter, probably upon receipt of some BGP updates. I also attempted Cisco XE 3.15.0S to see if it would improve, knowing that this is not the version used in the lab. The result was the same.

```

! CSR2
Exception to IOS Thread:
Frame pointer 0x7FC0677711C0, PC = 0x7FC0936F9559

UNIX-EXT-SIGNAL: Aborted(6), Process = BGP Task
-Traceback= 1#59fa9301cdbb8c697aa0b2517c9fda47  c:7FC0936C8000+31559
c:7FC0936C8000+32A09  irc:7FC093E9A000+2FFE7  :7FC0CBB4D000+5391022
:7FC0CBB4D000+538F8B1  :7FC0CBB4D000+538E65F  :7FC0CBB4D000+53888F7
:7FC0CBB4D000+542AA25  :7FC0CBB4D000+542A41D  :7FC0CBB4D000+541039B
[snip]

```

I include some debugging below for reference, but there are no obvious issues. The crash happens just before the log messages are printed due to the time it takes to print syslog messages. I suspect these events actually happen just before the crash.

```
R2#debug bgp all
BGP debugging is on for all address families
```

```
R2#debug bgp all updates
BGP updates debugging is on for all address families
```

```
BGP: 13.0.0.12 passive open to 24.0.0.2
BGP: Fetched peer 13.0.0.12 from tcb
BGP: 13.0.0.12 passive went from Idle to Connect
BGP: ses global 13.0.0.12 (0x7F6A0BAF2410:0) pas Setting open delay timer to
60 seconds.
BGP: 13.0.0.12 passive rcv message type 1, length (excl. header) 64
BGP: ses global 13.0.0.12 (0x7F6A0BAF2410:0) pas Receive OPEN
BGP: 13.0.0.12 passive rcv OPEN, version 4, holdtime 180 seconds
BGP: 13.0.0.12 passive rcv OPEN w/ OPTION parameter len: 54
BGP: 13.0.0.12 passive rcvd OPEN w/ optional parameter type 2 (Capability)
len 6
BGP: 13.0.0.12 passive OPEN has CAPABILITY code: 1, length 4
BGP: 13.0.0.12 passive OPEN has MP_EXT CAP for afi/safi: 1/128
BGP: 13.0.0.12 passive rcvd OPEN w/ optional parameter type 2 (Capability)
len 6
BGP: 13.0.0.12 passive OPEN has CAPABILITY code: 1, length 4
BGP: 13.0.0.12 passive OPEN has MP_EXT CAP for afi/safi: 2/128
BGP: 13.0.0.12 passive rcvd OPEN w/ optional parameter type 2 (Capability)
len 6
BGP: 13.0.0.12 passive OPEN has CAPABILITY code: 1, length 4
BGP: 13.0.0.12 passive OPEN has MP_EXT CAP for afi/safi: 1/5
BGP: 13.0.0.12 passive rcvd OPEN w/ optional parameter type 2 (Capability)
len 6
BGP: 13.0.0.12 passive OPEN has CAPABILITY code: 1, length 4
BGP: 13.0.0.12 passive OPEN has MP_EXT CAP for afi/safi: 2/5
BGP: 13.0.0.12 passive rcvd OPEN w/ optional parameter type 2 (Capability)
len 2
BGP: 13.0.0.12 passive OPEN has CAPABILITY code: 128, length 0
BGP: 13.0.0.12 passive OPEN has ROUTE-REFRESH capability(old) for all
address-families
BGP: 13.0.0.12 passive rcvd OPEN w/ optional parameter type 2 (Capability)
len 2
BGP: 13.0.0.12 passive OPEN has CAPABILITY code: 2, length 0
BGP: 13.0.0.12 passive OPEN has ROUTE-REFRESH capability(new) for all
address-families
BGP: 13.0.0.12 passive rcvd OPEN w/ optional parameter type 2 (Capability)
len 6
BGP: 13.0.0.12 passive OPEN has CAPABILITY code: 65, length 4
BGP: 13.0.0.12 passive OPEN has 4-byte ASN CAP for: 13
```

```

BGP: 13.0.0.12 passive rcvd OPEN w/ optional parameter type 2 (Capability)
len 4
BGP: 13.0.0.12 passive OPEN has CAPABILITY code: 64, length 2
BGP: ses global 13.0.0.12 (0x7F6A0BAF2410:0) pas NSF OPEN has GR cap
BGP: ses global 13.0.0.12 (0x7F6A0BAF2410:0) pas NSF Peer has not restarted.
Restart Time: 120
BGP: 13.0.0.12 passive rcvd OPEN w/ remote AS 13, 4-byte remote AS 13
BGP: ses global 13.0.0.12 (0x7F6A0BAF2410:0) pas Adding topology VPNv4
Unicast:base
BGP: ses global 13.0.0.12 (0x7F6A0BAF2410:0) pas Adding topology VPNv6
Unicast:base
BGP: ses global 13.0.0.12 (0x7F6A0BAF2410:0) pas Adding topology MVPNV4
Unicast:base
BGP: ses global 13.0.0.12 (0x7F6A0BAF2410:0) pas Adding topology MVPNV6
Unicast:base
BGP: ses global 13.0.0.12 (0x7F6A0BAF2410:0) pas Send OPEN
BGP: ses global 13.0.0.12 (0x7F6A0BAF2410:0) pas Building Enhanced Refresh
capability
BGP: 13.0.0.12 passive went from Connect to OpenSent
BGP: 13.0.0.12 passive sending OPEN, version 4, my as: 24, holdtime 180
seconds, ID 18000002
BGP: 13.0.0.12 passive went from OpenSent to OpenConfirm
BGP: 13.0.0.12 passive went from OpenConfirm to Established
BGP: ses global 13.0.0.12 (0x7F6A0BAF2410:1) pas Assigned ID
BGP: nbr global 13.0.0.12 Stop Active Open timer as all topologies are
allocated
BGP: ses global 13.0.0.12 (0x7F6A0BAF2410:1) Up
%BGP-5-ADJCHANGE: neighbor 13.0.0.12 Up

```

Unfortunately, this means the vast majority of our L3VPN verifications cannot be conducted since the RR for AS 24 is malfunctioning. However, we can test some L2VPN features. Since the most interesting feature of option C is the ability for the ASBRs to transport any kind of services, we can confirm the existing ATOM/CFM and BGP-based VPLS are operational. On CSR2, we can see that the remote VPLS route from CSR8 has the next-hop intact despite “next-hop-unchanged” being removed from all routers. In this case, XRv4 did not change the next-hop to self towards CSR2, and XRv2 did not change the next-hop to self towards CSR8.

```

R2#show bgp l2vpn vpls all | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 13:800
*>i 13:800:VEID-8:Blk-1/136
      13.0.0.8          0          100          0 (13) ?
Route Distinguisher: 24:200
*> 24:200:VEID-2:Blk-1/136
      0.0.0.0          32768 ?

```

```

R8#show bgp l2vpn vpls all | begin Network
      Network          Next Hop          Metric LocPrf Weight Path

```



```

Route Distinguisher: 13:800
*> 13:800:VEID-8:Blk-1/136
                                0.0.0.0                                32768 ?
Route Distinguisher: 24:200
*>i 24:200:VEID-2:Blk-1/136
                                24.0.0.2                                0      100      0 (24) ?

```

Quickly checking the VFI status, we can see a single PW is up and the in/out labels are matching on both sides. This means that the BGP signaling of VBS, VE, and LB was successful and consistent.

```

R2#show l2vpn vfi name VPLS
Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No
VFI name: VPLS, state: up, type: multipoint, signaling: BGP
VPN ID: 200, VE-ID: 2, VE-SIZE: 15
RD: 24:200, RT: 24:3
Bridge-Domain 3 attachment circuits:
Pseudo-port interface: pseudowire100001
Interface      Peer Address      VE-ID  Local Label  Remote Label  S
pseudowire100002 13.0.0.8          8      2007         8024         Y

```

```

R8#show l2vpn vfi name VPLS
Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No
VFI name: VPLS, state: up, type: multipoint, signaling: BGP
VPN ID: 800, VE-ID: 8, VE-SIZE: 10
RD: 13:800, RT: 13:3
Bridge-Domain 3 attachment circuits:
Pseudo-port interface: pseudowire100009
Interface      Peer Address      VE-ID  Local Label  Remote Label  S
pseudowire100011 24.0.0.2          2      8024         2007         Y

```

I will trace the LSP for this L2VPN in both directions. Above, we see that the remote label is 2007, which was evaluated using the math detailed in the L2VPN BGP signaling section. The next-hop for this VPLS prefix is 24.0.0.2, which has a corresponding BGP route and label. The next-hop for the BGP prefix is an IGP route with an LDP label.

```

R8#show ip route 24.0.0.2
Routing entry for 24.0.0.2/32
  Known via "bgp 13", distance 200, metric 20
  Tag 24, type internal
  Last update from 10.6.11.6 00:19:52 ago
  Routing Descriptor Blocks:
  * 10.6.11.6, from 13.0.0.12, 00:19:52 ago
    Route metric is 20, traffic share count is 1
    AS Hops 0
    Route tag 24
    MPLS label: 6013

```

```

R8#show ip route 10.6.11.6

```

Routing entry for 10.6.11.6/32

Known via "ospf 13", distance 110, metric 20

Tag 13, type extern 2, forward metric 1

Last update from 13.8.11.11 on GigabitEthernet2.581, 01:49:23 ago

Routing Descriptor Blocks:

* 13.8.11.11, from 13.0.0.11, 01:49:23 ago, via GigabitEthernet2.581

Route metric is 20, traffic share count is 1

Route tag 13

R8#show mpls ldp bindings 10.6.11.6 32 neighbor 13.0.0.11

lib entry: 10.6.11.6/32, rev 21

remote binding: lsr: 13.0.0.11:0, label: 91038

The label stack includes the VPLS label, BGP label, and LDP label, from bottom to top, as shown below. XRv1 receives label 91038, pops it, and exposes label 6013 to CSR6. This was allocated by CSR6 but not changed by XRv1, which means it must tunnel label 6013 through to a BGP router who can swap it.

R8#show l2vpn atom vc service-name VPLS detail | include label_stack

Output interface: Gi2.581, imposed label stack {91038 6013 2007}

RP/0/0/CPU0:XRv1#show mpls forwarding labels 91038

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
91038	Pop	10.6.11.6/32	Gi0/0/0/0.561	10.6.11.6	243561

CSR6 has an IGP route to 24.0.0.2, not a BGP route, so the incoming BGP label of 6013 is swapped to the outbound LDP label of 94009, which tunnels traffic through XRv4. The LFIB is smart enough to connect BGP and LDP LSPs in this way, as we have seen many times.

R6#show ip route 24.0.0.2

Routing entry for 24.0.0.2/32

Known via "isis", distance 115, metric 20

Tag 24, type level-2

Redistributing via isis 24, bgp 24

Advertised by bgp 24 level-2 route-map RM_ISIS_TO_BGP

Last update from 24.6.14.14 on GigabitEthernet2.564, 00:22:42 ago

Routing Descriptor Blocks:

* 24.6.14.14, from 24.0.0.2, 00:22:42 ago, via GigabitEthernet2.564

Route metric is 20, traffic share count is 1

Route tag 24

R6#show mpls ldp bindings 24.0.0.2 32 neighbor 24.0.0.14

lib entry: 24.0.0.2/32, rev 81

remote binding: lsr: 24.0.0.14:0, label: 94009

R6#show mpls forwarding-table labels 6013

Local	Outgoing	Prefix	Bytes	Label	Outgoing	Next Hop

Label	Label	or Tunnel Id	Switched	interface	
6013	94009	24.0.0.2/32	50680	Gi2.564	24.6.14.14

XRv4 removes label 94009 to expose the VPLS label to CSR2, who delivers it to the appropriate bridge-domain for forwarding. The BD looks at the MAC address and forwards the Ethernet frame towards CSR1.

```
RP/0/0/CPU0:XRv4#show mpls forwarding labels 94009
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94009	Pop	24.0.0.2/32	Gi0/0/0/0.524	24.2.14.2	71449

```
R2#show mpls forwarding-table labels 2007
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Label	Outgoing interface	Next Hop
2007	No Label	lbl-blk-id(1:7)	1418		none	point2point

Although this does not verify the VPLS label, all other labels are reveal with an ordinary traceroute from CSR8 to CSR2. Since the next-hops were modified at the ASBRs just like they would have been for eBGP, the behavior is the same.

```
R8#traceroute 24.0.0.2 source 13.0.0.8
```

Type escape sequence to abort.

Tracing the route to 24.0.0.2

VRF info: (vrf in name/id, vrf out name/id)

```
 1 13.8.11.11 [MPLS: Labels 91038/6013 Exp 0] 9 msec 6 msec 7 msec
 2 10.6.11.6 [MPLS: Label 6013 Exp 0] 27 msec 31 msec 30 msec
 3 24.6.14.14 [MPLS: Label 94009 Exp 0] 20 msec 20 msec 20 msec
 4 24.2.14.2 20 msec 11 msec 11 msec
```

Inside the VPLS, we verify that CSR1 and CSR3 can communicate. They are also one hop away, as expected.

```
R3#ping vrf VPLS 10.0.0.1
```

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 10.0.0.1, timeout is 2 seconds:

!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 11/11/12 ms

```
R3#traceroute vrf VPLS 10.0.0.1
```

Type escape sequence to abort.

Tracing the route to 10.0.0.1

VRF info: (vrf in name/id, vrf out name/id)

```
 1 10.0.0.1 13 msec 10 msec 10 msec
```

From CSR2 to CSR8, the logic is similar. Earlier we saw label 8024 in use for the remote PW. An LDP label of 94019 is pushed onto the stack to tunnel traffic towards CSR8. CSR2 has an IGP route for this prefix, not a BGP route, so only one new label is added. XRv4 swaps this for 6022 which describes CSR6's path to 13.0.0.8/32.

```
R2#show ip cef 13.0.0.8
13.0.0.8/32
  nexthop 24.2.14.14 GigabitEthernet2.524 label 94019
```

```
RP/0/0/CPU0:XRv4#show mpls forwarding labels 94019
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94019	6022	13.0.0.8/32	Gi0/0/0/0.564	24.6.14.6	112490

CSR6 has a BGP route for 13.0.0.8/32, not an LDP route, so CSR6 connects the LDP to the BGP LSP. The label is 91001 which is allocated by XRv1.

```
R6#show ip route 13.0.0.8
Routing entry for 13.0.0.8/32
  Known via "bgp 24", distance 30, metric 0
  Tag 13, type internal
  Redistributing via isis 24
  Advertised by isis 24 metric-type internal level-2 route-map RM_BGP_TO_ISIS
  Last update from 10.6.11.11 01:19:52 ago
  Routing Descriptor Blocks:
  * 10.6.11.11, from 10.6.11.11, 01:19:52 ago
    Route metric is 0, traffic share count is 1
    AS Hops 0
    Route tag 13
    MPLS label: 91001
```

```
R6#show mpls forwarding-table labels 6022
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
6022	91001	13.0.0.8/32	136064	Gi2.561	10.6.11.11

Since XRv1 is the penultimate hop, it removes this label to expose label 8024 to CSR8. CSR8 delivers it to the bridge-domain for switching towards CSR3.

```
RP/0/0/CPU0:XRv1#show mpls forwarding labels 91001
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
91001	Pop	13.0.0.8/32	Gi0/0/0/0.581	13.8.11.8	241390

```
R8#show mpls forwarding-table labels 8024
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
8024	No Label	lbl-blk-id(1:1)	1622	none	point2point

We can confirm the label stack using traceroute on CSR2. This confirms that the label behavior is identical to ordinary inter-AS option C.

```
R2#traceroute 13.0.0.8 source 24.0.0.2
Type escape sequence to abort.
Tracing the route to 13.0.0.8
VRF info: (vrf in name/id, vrf out name/id)
 1 24.2.14.14 [MPLS: Label 94019 Exp 0] 6 msec 6 msec 11 msec
 2 24.6.14.6 [MPLS: Label 6022 Exp 0] 31 msec 30 msec 30 msec
 3 10.6.11.11 [MPLS: Label 91001 Exp 0] 33 msec 8 msec 20 msec
 4 13.8.11.8 21 msec 11 msec 11 msec
```

We also built a dummy AToM circuit to test CFM between AS boundaries. CFM is currently up as the MEPs can communicate using CCMs. Using ping, we confirm that the E-LINE service is operational. This is identical to ordinary option C as well.

```
R2#show ethernet cfm maintenance-points remote domain C
-----
MPID  Domain Name          MacAddress          IfSt  PtSt
Lvl  Domain ID              Ingress
RDI  MA Name                Type Id             SrvcInst
     EVC Name                Age
     Local MEP Info
-----
 8    C                      001e.e64d.4dbf      Up    Up
 5    C                      Gi2:(13.0.0.8, 28)
 -    icc OPTC123456789      XCON N/A            999
     EVC_999                  2s
     MPID: 2 Domain: C MA: icc OPTC123456789
```

```
R2#ping ethernet mpid 8 domain C service icc OPTC 123456789
Type escape sequence to abort.
Sending 5 Ethernet CFM loopback messages to 001e.e64d.4dbf, timeout is 5
seconds:!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 7/7/7 ms
```

With respect to multi-hop peerings, requirements for “mpls activate” on XR are similar for option C as they are for option B. I configure CSR6 and XRv1 similarly. XRv1 configuration is simplified since only IPv4 labeled-unicast needs to be modified.

```
! CSR6
interface Loopback611
description EBGp MHOP TEST
```

```

ip address 10.6.110.110 255.255.255.255

router bgp 24
 neighbor 10.6.11.11 update-source Loopback611

! XRv1
router static
 address-family ipv4 unicast
  no 10.6.11.6/32 GigabitEthernet0/0/0/0.561
  10.6.110.110/32 GigabitEthernet0/0/0/0.561 10.6.11.6 tag 13

router bgp 13
 no neighbor 10.6.11.6
 neighbor 10.6.110.110
 remote-as 24
 ignore-connected-check
 address-family ipv4 labeled-unicast
  route-policy RPL_IF_DEST_PASS(PS_LOCAL_LOOPBACKS) out
  next-hop-self

```

Once BGP converges, we can see the route exchange works without issue. The next-hops remain reachable and labels are exchanged as well. We must include the route-tag of 13 onto the static route so that it gets redistributed into OSPF.

```

RP/0/0/CPU0:XRv1#show bgp ipv4 labeled-unicast neighbors 10.6.110.110 routes
| begin Network

```

Network	Next Hop	Metric	LocPrf	Weight	Path
*> 24.0.0.2/32	10.6.110.110	20	100	0	(24) i
*> 24.0.0.6/32	10.6.110.110	0	100	0	(24) i
*> 24.0.0.14/32	10.6.110.110	10	100	0	(24) i

```

R6#show bgp ipv4 unicast neighbors 10.6.11.11 routes | begin Network

```

Network	Next Hop	Metric	LocPrf	Weight	Path
*> 13.0.0.5/32	10.6.11.11	0	100	6666	(13) i
*> 13.0.0.8/32	10.6.11.11	0	100	6666	(13) i
*> 13.0.0.11/32	10.6.11.11	0	100	6666	(13) i
*> 13.0.0.12/32	10.6.11.11	0	100	6666	(13) i

Assuming CSR8 prefers the path to 24.0.0.2 via XRv1 (towards 10.6.11.6), the VPLS should be broken. We used weight on XRv2 to prefer the path via XRv1 to influence this. CSR8 is able to see both the BGP and LDP labels for the recursive lookups that occur, which is correct. The transport label stack becomes {91005 6006}.

```

RP/0/0/CPU0:XRv2#show bgp ipv4 labeled-unicast 24.0.0.2/32 brief | begin
Network

```

Network	Next Hop	Metric	LocPrf	Weight	Path
* i24.0.0.2/32	10.5.6.6	20	100	0	(24) i

```
*>i 10.6.110.110 20 100 2222 (24) i
```

```
R8#show ip cef 24.0.0.2 detail
24.0.0.2/32, epoch 2, flags [rib defined all labels]
 1 RR source [no flags]
 recursive via 10.6.110.110 label 6006
  nexthop 13.8.11.11 GigabitEthernet2.581 label 91005
```

This is the same issue we encountered with option B. XRv1 performs a recursive lookup on 24.0.0.2/32 to find a host-route. The prefix length is 32, which is correct, but forwarding is broken. This route has no label assigned since BGP is expected to see an LDP label for it.

```
RP/0/0/CPU0:XRv1#show route 24.0.0.2
Routing entry for 24.0.0.2/32
  Known via "bgp 13", distance 200, metric 20, labeled unicast (3107)
  Tag 24, type internal
  Routing Descriptor Blocks
    10.6.110.110, from 10.6.110.110
      Route metric is 20
  No advertising protos.
```

```
RP/0/0/CPU0:XRv1#show cef 10.6.110.110
10.6.110.110/32, version 19, internal 0x1000001 0x0 (ptr 0xa1401ef4) [3], 0x0
(0xa13cd53c), 0xa20 (0xa156d258)
local adjacency 10.6.11.6
  Prefix Len 32, traffic index 0, precedence n/a, priority 3
  via 10.6.11.6, GigabitEthernet0/0/0/0.561, 5 dependencies, weight 0, class
0 [flags 0x0]
  path-idx 0 NHID 0x0 [0xa0f371fc 0x0]
  next hop 10.6.11.6
  local adjacency
  local label 91005 labels imposed {None}
```

The less desirable LDP solution could work here, provided CSR6 and XRv1 formed an LDP session. CSR6 could advertise implicit-null bound to prefix 10.6.110.110/32 which would allow XRv1 to forward MPLS traffic correctly. This was demonstrated in the option B section. However, I will jump straight to the preferred solution of using “mpls activate”. As shown in option B, this allows BGP to perform an implicit-null binding for the BGP next-hop and is used for multi-hop (but still connected) situations like this. Once the command is applied, the FIB is updated with the implicit-null label binding as expected.

```
! XRv1
router bgp 13
 mpls activate
 interface GigabitEthernet0/0/0/0.561
```

```
RP/0/0/CPU0:XRv1#show cef 10.6.110.110
```

```

10.6.110.110/32, version 25, internal 0x1000001 0x0 (ptr 0xa1401ef4) [3], 0x0
(0xa13cd53c), 0xa20 (0xa156d280)
local adjacency 10.6.11.6
Prefix Len 32, traffic index 0, precedence n/a, priority 4
via 10.6.11.6, GigabitEthernet0/0/0/0.561, 5 dependencies, weight 0, class
0 [flags 0x0]
path-idx 0 NHID 0x0 [0xa0f371fc 0xa0f3749c]
next hop 10.6.11.6
local adjacency
local label 91005 labels imposed {ImplNull}

```

CSR2 and CSR8 are flooded with log messages indicating that the VPLS PWs have formed, CFM RMEPs have been discovered, etc. This is a good indication that the network is functional again. At a glance, CSR8 shows both the BGP-signaled VPLS and statically-configured AToM circuits as up.

```
R8#show l2vpn atom vc
```

Interface	Peer ID	VC ID	Service		Status
			Type	Name	
pw100012	2	800	vfi	VPLS	UP
pw28	24.0.0.2	28	p2p	EVPL_28	UP

A quick traceroute between PEs confirms the transport path, and a traceroute inside the VPN confirms that the VPLS is operational. I use CFM loopback messages (LBM) to ping inside the AToM circuit to test it as well. Everything is operational at this point.

```
R8#traceroute 24.0.0.2 source 13.0.0.8
```

```
Type escape sequence to abort.
```

```
Tracing the route to 24.0.0.2
```

```
VRF info: (vrf in name/id, vrf out name/id)
```

```

1 13.8.11.11 [MPLS: Labels 91005/6006 Exp 0] 7 msec 5 msec 15 msec
2 10.6.11.6 [MPLS: Label 6006 Exp 0] 31 msec 30 msec 35 msec
3 24.6.14.14 [MPLS: Label 94009 Exp 0] 16 msec 20 msec 20 msec
4 24.2.14.2 20 msec 11 msec 11 msec

```

```
R3#traceroute vrf VPLS 10.0.0.1
```

```
Type escape sequence to abort.
```

```
Tracing the route to 10.0.0.1
```

```
VRF info: (vrf in name/id, vrf out name/id)
```

```
1 10.0.0.1 12 msec 8 msec 8 msec
```

```
R8#ping ethernet mpid 2 domain C service icc OPTC 123456789
```

```
Type escape sequence to abort.
```

```
Sending 5 Ethernet CFM loopback messages to 001e.1415.dbbf, timeout is 5
seconds:!!!!
```

```
Success rate is 100 percent (5/5), round-trip min/avg/max = 7/11/30 ms
```


8.4.4 Option AB Inter-AS hybrid (AKA Option D)

Option AB is a hybrid option that attempts to offer the best characteristics of options A and B. Like option A, the VRFs are configured locally on each ASBR and traffic is exchanged as raw IP (in most cases). This keeps the VRFs isolated and allows for per-VRF QoS enforcement on the IP packets. Like option B, a single inter-AS BGP session is formed in the global table to exchange VPN prefixes. All VPN routes can be exchanged in a single connection with each individual VRF configured separately for flexibility. While still configuration intensive, it allows option A to scale better due to using a single inter-AS BGP session. This is somewhat similar to VRF route-leaking and next-hop adjustment; routes are learned via a global VPNv4 session with next-hop resolved inside of their specific VRF. Normally this is not how MPLS VPNs work, but it is a specific design case on ASBRs which are MPLS LERs and ASBRs concurrently.

This option has some significant drawbacks. First, it is only supported on IOS/XE based platforms, as there is no support for XR or other vendors. It is sometimes called Option D, which was a draft RFC to detail the procedure, but it does not appear popular (draft is also expired). Second, VPNv6 is not supported at all, so support for IPv6 VPN continues to use option A. Lastly, the testing below reveals some challenging inconsistencies with L3VPN and MVPN architectures, which make the feature more difficult to understand, deploy, and troubleshoot.

Additional Reading – Reference configurations “inter-as-mpls-ab”

8.4.4.1 L3VPN

This lab continues from the option A lab. Before beginning, I quickly verify that the proper VRFs are configured on the ASBRs. All 4 ASBRs should have all VRFs configured for redundancy.

```
R5#show vrf | include ipv4
  EIGRP          13:3          ipv4,ipv6    Tu2
  OSPF           13:2          ipv4,ipv6    Tu3

R6#show vrf | include ipv4
  EIGRP          24:3          ipv4,ipv6    Tu2
  OSPF           24:2          ipv4,ipv6    Tu3

R7#show vrf | include ipv4
  EIGRP          24:3          ipv4,ipv6    Tu2
  OSPF           24:2          ipv4,ipv6    Tu3

RP/0/0/CPU0:XRv1#show vrf all | include ^[ ^ ]
VRF          RD          RT          AFI    SAFI
  EIGRP       13:3
  OSPF        13:2
```

Checking CSR6 and CSR5, we can see VPNv4/v6 sessions for each neighbor within the VPN. This is the traditional option A design that we will be changing with option AB. For VPNv6, we will continue to use the classic option A approach as it is not supported with option AB.

```
R5#show bgp vpnv4 unicast vrf EIGRP summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
10.5.6.6      4      24      49      49      49    0    0 00:31:18      5
10.5.7.7      4      24      45      49      49    0    0 00:31:19      5
```

```
R5#show bgp vpnv4 unicast vrf OSPF summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
10.5.6.6      4      24      43      46      49    0    0 00:31:25      2
10.5.7.7      4      24      41      47      49    0    0 00:31:23      2
```

```
R5#show bgp vpnv6 unicast vrf EIGRP summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
FD00:10:5:6::6 4      24      66      66      74    0    0 00:31:40      5
FD00:10:5:7::7 4      24      46      66      74    0    0 00:31:35      5
```

```
R5#show bgp vpnv6 unicast vrf OSPF summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
FD00:10:5:6::6 4      24      58      52      74    0    0 00:31:34      2
FD00:10:5:7::7 4      24      40      53      74    0    0 00:31:28      2
```

```
R6#show bgp vpnv4 unicast vrf EIGRP summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
10.5.6.5      4      13      51      50      51    0    0 00:32:46      6
10.6.11.11    4      13      37      49      51    0    0 00:31:30      6
```

```
R6#show bgp vpnv4 unicast vrf OSPF summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
10.5.6.5      4      13      48      44      51    0    0 00:32:48      6
10.6.11.11    4      13      37      44      51    0    0 00:31:30      6
```

```
R6#show bgp vpnv6 unicast vrf EIGRP summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
FD00:10:5:6::5 4      13      67      68      104   0    0 00:32:53      6
FD00:10:6:11::11
4      13      37      54      104   0    0 00:31:20      6
```

```
R6#show bgp vpnv6 unicast vrf OSPF summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
FD00:10:5:6::5 4      13      54      59      104   0    0 00:32:53      6
FD00:10:6:11::11
4      13      37      54      104   0    0 00:31:32      6
```

A few quick traceroutes confirm that the inter-AS option A L3VPN connectivity is functional. We test VRF EIGRP, VRF OSPF, and central services. As expected, the inter-AS links are always non-MPLS encapsulated.

```
R3#traceroute 10.13.13.13 source 10.3.3.3
Type escape sequence to abort.
```

```

Tracing the route to 10.13.13.13
VRF info: (vrf in name/id, vrf out name/id)
 1 10.3.12.12 3 msec 2 msec 2 msec
 2 13.11.12.11 [MPLS: Labels 91015/5076 Exp 0] 6 msec 6 msec 6 msec
 3 10.5.6.5 [MPLS: Label 5076 Exp 0] 16 msec 16 msec 15 msec
 4 10.5.6.6 20 msec 10 msec 11 msec
 5 24.6.14.14 [MPLS: Label 94006 Exp 0] 10 msec 21 msec 18 msec
 6 10.13.14.13 21 msec 13 msec 13 msec

```

```

R9#traceroute 10.4.4.4 so 10.9.9.9
Type escape sequence to abort.
Tracing the route to 10.4.4.4
VRF info: (vrf in name/id, vrf out name/id)
 1 10.2.9.2 5 msec 5 msec 4 msec
 2 24.2.14.14 [MPLS: Labels 94008/6044 Exp 0] 7 msec 5 msec 5 msec
 3 10.5.6.6 [MPLS: Label 6044 Exp 0] 18 msec 16 msec 15 msec
 4 10.5.6.5 20 msec 10 msec 10 msec
 5 10.4.8.8 [MPLS: Label 8020 Exp 0] 11 msec 20 msec 19 msec
 6 10.4.8.4 20 msec 9 msec 12 msec

```

```

RP/0/0/CPU0:XRv3#traceroute 110.0.0.1 source 10.13.13.13
Type escape sequence to abort.
Tracing the route to 110.0.0.1
 1 10.13.14.14 0 msec 0 msec 0 msec
 2 10.5.6.6 [MPLS: Label 6038 Exp 0] 0 msec 0 msec 0 msec
 3 10.5.6.5 0 msec 0 msec 0 msec
 4 10.8.10.8 [MPLS: Label 8017 Exp 0] 0 msec 0 msec 239 msec
 5 10.8.10.10 0 msec 0 msec 0 msec

```

Next, I will enable option AB between CSR5/CSR6 and CSR5/CSR7. The first step is to create a common transit link in the global table. This is similar to option B except we don't need MPLS forwarding as it is used for control-plane traffic only. These transit links are almost identical to options B and C except MPLS and all supporting functions (RSVP, TE, LDP, etc) are disabled. There is also no reason to run IPv6 on these links at all since only VPNv4 is supported for option AB.

```

! CSR5
interface GigabitEthernet2.556
 encapsulation dot1Q 3556
 ip address 10.5.6.5 255.255.255.0
 ip pim bsr-border
 ip pim sparse-mode

interface GigabitEthernet2.557
 encapsulation dot1Q 3557
 ip address 10.5.7.5 255.255.255.0
 ip pim bsr-border
 ip pim sparse-mode

```

```

! CSR6
interface GigabitEthernet2.556
 encapsulation dot1Q 3556
 ip address 10.5.6.6 255.255.255.0
 ip pim bsr-border
 ip pim sparse-mode

```

```

! CSR7
interface GigabitEthernet2.557
 encapsulation dot1Q 3557
 ip address 10.5.7.7 255.255.255.0
 ip pim bsr-border
 ip pim sparse-mode

```

A quick way to check if the transit links are configured correctly is to check the PIM neighbors on CSR5. Although not a 100% verification, this is a quick way to see if the basic layer 2/3 reachability exists.

```

R5#show ip pim neighbor | begin ^Neighbor
Neighbor      Interface      Uptime/Expires    Ver    DR
Address
13.5.11.11    GigabitEthernet2.551    4d23h/00:01:24    v2      1 / DR P G
13.5.8.8      GigabitEthernet2.558    2d07h/00:01:25    v2      1 / DR S P G
10.5.6.6      GigabitEthernet2.556    00:00:45/00:01:28 v2      1 / DR S P G
10.5.7.7      GigabitEthernet2.557    00:00:45/00:01:28 v2      1 / DR S P G

```

Next, we will update the BGP configuration on all devices. This involves removing the VRF-aware BGP sessions for IPv4 as there is no need for BGP to be VRF-aware for this AFI with option AB. CSR6 and XRv1 will continue to run classic option A for both IPv4/v6 as XR does not support option AB at all. CSR5 and CSR7 can safely remove the entire AFIs from their configuration, while CSR6 must selectively delete specific neighbors. The peers are identified as “inter-as-hybrid” which implies option AB is used. Note that the Cisco documentation shows the command “no bgp default route-target filter” which is the preferred way to configure option B ASBRs in the configuration for option AB. This is technically not required since the ASBRs are locally importing the RTs within each AS, as well as advertising them via eBGP VPNv4. I comment them out below but leave them in the snippet as a reference.

```

! CSR5
router bgp 13
 no address-family ipv4 vrf EIGRP
 no address-family ipv4 vrf OSPF
 neighbor 10.5.6.6 remote-as 24
 neighbor 10.5.7.7 remote-as 24
 address-family vpnv4
  ! no bgp default route-target filter
 neighbor 10.5.6.6 activate
 neighbor 10.5.7.7 activate
 neighbor 10.5.6.6 inter-as-hybrid
 neighbor 10.5.7.7 inter-as-hybrid

```

```

! CSR6
router bgp 24
 neighbor 10.5.6.5 remote-as 13
 address-family ipv4 vrf EIGRP
   no neighbor 10.5.6.5
 address-family ipv4 vrf OSPF
   no neighbor 10.5.6.5
 address-family vpnv4
   ! no bgp default route-target filter
 neighbor 10.5.6.5 activate
 neighbor 10.5.6.5 inter-as-hybrid

```

```

! CSR7
router bgp 24
 neighbor 10.5.7.5 remote-as 13
 no address-family ipv4 vrf EIGRP
 no address-family ipv4 vrf OSPF
 address-family vpnv4
   ! no bgp default route-target filter
 neighbor 10.5.7.5 activate
 neighbor 10.5.7.5 inter-as-hybrid

```

We can confirm the BGP sessions are up by checking CSR5. We can see several routes being exchanged as well.

```

R5#show bgp vpnv4 unicast all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State/PfxRcd
10.5.6.6      4      24      73      75      84    0    0 00:15:17      7
10.5.7.7      4      24      43      43      84    0    0 00:12:44      7
13.0.0.12     4      13     479     507      84    0    0 01:08:39     15

```

Despite routes being learned, they cannot be imported by the AS 24 VRFs due to RT mismatches. One of the benefits of option A is that this information is contained within an AS, but with option AB, it is exchanged between AS boundaries like option B.

```

R5#show bgp vpnv4 unicast rd 24:2 10.9.9.9/32
BGP routing table entry for 24:2:10.9.9.9/32, version 83
Paths: (2 available, best #2, no table)
  Not advertised to any peer
  Refresh Epoch 2
  24
    10.5.7.7 (via default) from 10.5.7.7 (24.0.0.7)
      Origin incomplete, localpref 100, valid, external
      Extended Community: RT:24:2 OSPF ROUTER ID:10.2.9.2:0
        OSPF RT:0.0.0.0:2:0
      mpls labels in/out nlabel/7048
      rx pathid: 0, tx pathid: 0

```

```

Refresh Epoch 3
24
10.5.6.6 (via default) from 10.5.6.6 (24.0.0.6)
  Origin incomplete, localpref 100, valid, external, best
  Extended Community: RT:24:2 OSPF ROUTER ID:10.2.9.2:0
    OSPF RT:0.0.0.0:2:0
  mpls labels in/out nolabel/6002
  rx pathid: 0, tx pathid: 0x0

```

Cisco's documentation suggests adding adding "import" statements on PEs that need to import foreign RTs. This is simpler than configuring new export RTs or the RT rewrite technique.

```

! CSR5
vrf definition EIGRP
  address-family ipv4
    route-target import 24:3

vrf definition OSPF
  address-family ipv4
    route-target import 24:2

! CSR6 and CSR7
vrf definition EIGRP
  address-family ipv4
    route-target import 13:3

vrf definition OSPF
  address-family ipv4
    route-target import 13:2

```

We check a single route on CSR5 to confirm the logic of this solution. The newly imported eBGP VPNv4 route also happens to be the best path on CSR5. The iBGP-learned path was the classic option A route via XRv1, which is still functional also.

```

R5#show bgp vpnv4 unicast vrf OSPF 10.9.9.9/32
BGP routing table entry for 13:2:10.9.9.9/32, version 88
Paths: (2 available, best #1, table OSPF)
  Not advertised to any peer
  Refresh Epoch 7
  24, imported path from 24:2:10.9.9.9/32 (global)
    10.5.6.6 (via default) from 10.5.6.6 (24.0.0.6)
      Origin incomplete, localpref 100, valid, external, best
      Extended Community: RT:24:2 OSPF ROUTER ID:10.2.9.2:0
        OSPF RT:0.0.0.0:2:0
      mpls labels in/out 5063/6002
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  24

```

```

13.0.0.11 (metric 2) (via default) from 13.0.0.12 (13.0.0.12)
  Origin incomplete, localpref 100, valid, internal
  Extended Community: RT:13:2 OSPF ROUTER ID:10.2.9.2:0
    OSPF RT:0.0.0.0:2:0
  Originator: 13.0.0.11, Cluster list: 13.0.0.12
  Connector Attribute: count=1
    type 1 len 12 value 13:2:13.0.0.11
  mpls labels in/out 5063/91022
  rx pathid: 0, tx pathid: 0

```

Despite this being a best-path, CSR5 does not advertise it to XRv2. Although this may seem counter-intuitive, the reason is because CSR5 imports the route locally. Unless the peer is an RR client, eBGP VPN routes aren't advertised to iBGP peers if those routes are imported locally.

```

R5#show bgp vpnv4 unicast all neighbors 13.0.0.12 advertised-routes
Total number of prefixes 0

```

This is the symptom of an incomplete configuration. The VRF itself is not yet aware we are trying to run option AB. The VPN routes have a next-hop of 10.5.6.6; coincidentally, this is the correct next-hop in the VPN table as well as the global eBGP peer. We need to explicitly tell the VRF that it is running in "inter-as-hybrid" mode and optionally specify the VRF-aware next-hop. Normally this next-hop would be the option A eBGP peer.

```

! CSR5
vrf definition EIGRP
  address-family ipv4
    inter-as-hybrid next-hop 10.5.6.6

```

After issuing the command, the router tells us that the VRF table is being reloaded so that the changes take effect. We immediately see the EIGRP VPN routes being advertised to XRv2 as expected; the typically VPN RT import advertisement rules are relaxed in this configuration.

```

R5(config-vrf-af)#inter-as-hybrid next-hop 10.5.6.6
% The VRFs routing table is being reloaded to enforce the new configuration

```

```

R5#show bgp vpnv4 unicast all neighbors 13.0.0.12 advertised-routes | begin
Network

```

```

      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 13:3 (default for vrf EIGRP)
*> 10.1.1.1/32      10.5.6.6                      0 24 ?
*> 10.1.2.0/24      10.5.6.6                      0 24 ?
*> 10.1.13.0/24     10.5.6.6                      0 24 ?
*> 10.13.13.13/32   10.5.6.6                      0 24 ?
*> 10.13.14.0/24    10.5.6.6                      0 24 ?

```

We repeat the process for the OSPF VPN except I do not specify a next-hop. This technique works because CSR6 has the same IP address for all VPNs and the global table. Since all the VPNs are separate, this is a simple way to provision transit links for option A. It also means a simpler option AB VRF configuration as the next-hop need not be manually specified.

```
R5#show bgp vpnv4 unicast all neighbors 13.0.0.12 advertised-routes | begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 13:2 (default for vrf OSPF)					
*> 10.2.9.0/24	10.5.6.6			0 24	?
*> 10.9.9.9/32	10.5.6.6			0 24	?
Route Distinguisher: 13:3 (default for vrf EIGRP)					
*> 10.1.1.1/32	10.5.6.6			0 24	?
*> 10.1.2.0/24	10.5.6.6			0 24	?
*> 10.1.13.0/24	10.5.6.6			0 24	?
*> 10.13.13.13/32	10.5.6.6			0 24	?
*> 10.13.14.0/24	10.5.6.6			0 24	?

I quickly check XRv2 to ensure the routes are learned. One would think that a next-hop-self issue would have occurred, but option AB handles that automatically. Because it knows the data-plane is based on option A, the next-hop is automatically set to the ASBR without us having to manually configure it. This is a nice feature that simplifies configuration.

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast rd 13:2 | begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 13:2					
*>i10.2.9.0/24	13.0.0.5	0	100	0 24	?
* i	13.0.0.11		100	0 24	?
*>i10.4.4.4/32	13.0.0.8	1	100	0 ?	
*>i10.4.8.0/24	13.0.0.8	0	100	0 ?	
*>i10.9.9.9/32	13.0.0.5	0	100	0 24	?
* i	13.0.0.11		100	0 24	?

After we configured the VRF for inter-as-hybrid support, the ASBRs were able to advertise locally imported routes to iBGP peers. These routes are technically exports from the local VRF rather than pass-through advertisements from the eBGP peer. To prove it, I show the eBGP route to 10.9.9.9/32 on CSR5 and the iBGP route on CSR8. The RT advertised by CSR6/CSR7 was 24:2. CSR5 did not perform any kind of explicit rewrite but was configured to import 24:2. We did not change CSR8's RT import/export policies at all and it is still able to import this route into VRF OSPF. This is because CSR5 re-exported the route with RT:13:2 per the local RT export policy. This allows the ASBRs to import remote RTs but the remote PEs do not. This also increases option AB scalability over option B in some cases since the PE/RR is unaware of an RT change.

```
R5#show bgp vpnv4 unicast vrf OSPF 10.9.9.9/32
BGP routing table entry for 13:2:10.9.9.9/32, version 109
```



```

Paths: (1 available, best #1, table OSPF)
  Advertised to update-groups:
    3          5
  Refresh Epoch 7
  24, imported path from 24:2:10.9.9.9/32 (global)
    10.5.6.6 (via default) from 10.5.6.6 (24.0.0.6)
      Origin incomplete, localpref 100, valid, external, best
      Extended Community: RT:24:2 OSPF ROUTER ID:10.2.9.2:0
      OSPF RT:0.0.0.0:2:0
      mpls labels in/out 5052/6002
      rx pathid: 0, tx pathid: 0x0

```

```

R8#show bgp vpnv4 unicast vrf OSPF 10.9.9.9/32
BGP routing table entry for 13:2:10.9.9.9/32, version 872
Paths: (1 available, best #1, table OSPF)
  Not advertised to any peer
  Refresh Epoch 1
  24
    13.0.0.5 (metric 2) (via default) from 13.0.0.12 (13.0.0.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:13:2 OSPF ROUTER ID:10.2.9.2:0
      OSPF RT:0.0.0.0:2:0
      Originator: 13.0.0.5, Cluster list: 13.0.0.12
      Connector Attribute: count=1
        type 1 len 12 value 13:2:13.0.0.5
      mpls labels in/out nlabel/5052
      rx pathid: 0, tx pathid: 0x0

```

I quickly configure the VRFs to support option AB on CSR6 and CSR7 as well without specifying the next-hop. I temporarily shut down the BGP peer to XRv1 so that we can focus on option AB testing for now (not shown).

```

! CSR6 and CSR7
vrf definition EIGRP
  address-family ipv4
    inter-as-hybrid

vrf definition OSPF
  address-family ipv4
    inter-as-hybrid

```

CSR2 learns two routes for 10.4.4.4/32 inside the OSPF VRF: one from CSR6 and one from CSR7. These are the eBGP VPNv4 routes learned from the option B-style BGP connection to CSR5. Notice the RT:24:2 on the routes; this is a result of routes with RT:13:2 being imported on CSR6/CSR6 and being re-exported with RT:24:2 by those same ASBRs (CSR6 and CSR7).

```

R2#show bgp vpnv4 unicast vrf OSPF 10.4.4.4/32

```

```

BGP routing table entry for 24:2:10.4.4.4/32, version 1025
Paths: (2 available, best #2, table OSPF)
  Advertised to update-groups:
    13
  Refresh Epoch 1
  13, (Received from a RR-client)
    24.0.0.7 (metric 20) (via default) from 24.0.0.7 (24.0.0.7)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
      OSPF RT:0.0.0.0:2:0
      Connector Attribute: count=1
        type 1 len 12 value 13:2:13.0.0.8
      mpls labels in/out nlabel/7023
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 2
  13, (Received from a RR-client)
    24.0.0.6 (metric 20) (via default) from 24.0.0.6 (24.0.0.6)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
      OSPF RT:0.0.0.0:2:0
      Connector Attribute: count=1
        type 1 len 12 value 13:2:13.0.0.8
      mpls labels in/out nlabel/6031
      rx pathid: 0, tx pathid: 0x0

```

To simplify the initial test, the OSPF backdoor and sham-links are disabled. Both CEs have a route to one another via the MPLS network. This shows that the control-plane is correct, at least within the OSPF VPN.

```

R9#show ip cef 10.4.4.4
10.4.4.4/32
  nexthop 10.4.9.4 GigabitEthernet2.549

R4#show ip cef 10.9.9.9
10.9.9.9/32
  nexthop 10.4.8.8 GigabitEthernet2.548

```

We will manually trace the LSP from CSR9 to CSR4. On CSR2, there is a VPNv4 route from CSR6 using label 6031. The route to the VPN next-hop is learned from ISIS via XRv4, so XRv4's local LDP label is used. The label stack becomes {94008 6031} and we can quickly confirm this by checking the FIB.

```

R2#show bgp vpnv4 unicast vrf OSPF 10.4.4.4/32 bestpath
BGP routing table entry for 24:2:10.4.4.4/32, version 1032
Paths: (2 available, best #1, table OSPF)
  Advertised to update-groups:
    13
  Refresh Epoch 2

```

```

13, (Received from a RR-client)
24.0.0.6 (metric 20) (via default) from 24.0.0.6 (24.0.0.6)
Origin incomplete, metric 0, localpref 100, valid, internal, best
Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
OSPF RT:0.0.0.0:2:0
Connector Attribute: count=1
type 1 len 12 value 13:2:13.0.0.8
mpls labels in/out nolaabel/6031
rx pathid: 0, tx pathid: 0x0

```

```

R2#show ip route 24.0.0.6
Routing entry for 24.0.0.6/32
Known via "isis", distance 115, metric 20, type level-2
Redistributing via isis 24
Last update from 24.2.14.14 on GigabitEthernet2.524, 2d08h ago
Routing Descriptor Blocks:
* 24.2.14.14, from 24.0.0.6, 2d08h ago, via GigabitEthernet2.524
Route metric is 20, traffic share count is 1

```

```

R2#show mpls ldp bindings 24.0.0.6 32 neighbor 24.0.0.14
lib entry: 24.0.0.6/32, rev 12
remote binding: lsr: 24.0.0.14:0, label: 94008

```

```

R2#show ip cef vrf OSPF 10.4.4.4/32
10.4.4.4/32
nexthop 24.2.14.14 GigabitEthernet2.524 label 94008 6031

```

XRv4 is the penultimate hop so the topmost label is removed, exposing label 6031 to CSR6.

```

RP/0/0/CPU0:XRv4#show mpls forwarding labels 94008

```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94008	Pop	24.0.0.6/32	Gi0/0/0/0.564	24.6.14.6	248618

Like an ordinary option A ASBR, CSR6 removes all labels and sends the packet inside of the OSPF VPN towards CSR5. There aren't many fancy show commands to see option AB in action as most of the inner workings are done behind the scenes. It is similar to global/VRF route and traffic leaking discussed in another chapter.

```

R6#show mpls forwarding-table labels 6031 detail

```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Label	Outgoing interface	Next Hop
6031	No Label	10.4.4.4/32 [V]	0		Gi2.5562	10.5.6.5

```

MAC/Encaps=22/22, MRU=1504, Label Stack{}
005056A9DC63005056A9DE0D81000DE4810000020800
VPN route: OSPF
No output feature configured

```

CSR5 imposes a single VPN label for prefix 10.4.4.4/32 inside VRF OSPF since it is both the ingress LSR and penultimate hop. The label value is 8020; when CSR98 receives packets with this label, it removes all MPLS encapsulation and sends the IP packet to CSR4, the CE.

```
R5#show ip cef vrf OSPF 10.4.4.4
10.4.4.4/32
  nexthop 13.5.8.8 GigabitEthernet2.558 label 8020
```

```
R8#show mpls forwarding-table labels 8020 detail
Local      Outgoing    Prefix          Bytes Label    Outgoing    Next Hop
Label      Label       or Tunnel Id    Switched       interface
8020       No Label    10.4.4.4/32[V]  10308         Gi2.548     10.4.8.4
MAC/Encaps=18/18, MRU=1504, Label Stack{}
005056A92C57005056A9FB1C81000DDC0800
VPN route: OSPF
No output feature configured
```

We use traceroute to confirm this path. Just like option A, the traffic over the transit links is raw IP. Option AB is transparent to customers in this way as there is no way to tell whether option A or AB is in use based on the output below. The IPv6 traceroute, for example, is using option A and exhibit identical data-plane behavior. We did not explicitly trace the VPNv6 LSPs but this demonstrates that VPNv6 option A still functions independently from VPNv4 option AB.

```
R9#traceroute 10.4.4.4 so 10.9.9.9
Type escape sequence to abort.
Tracing the route to 10.4.4.4
VRF info: (vrf in name/id, vrf out name/id)
 1 10.2.9.2 5 msec 4 msec 4 msec
 2 24.2.14.14 [MPLS: Labels 94008/6031 Exp 0] 6 msec 5 msec 7 msec
 3 10.5.6.6 [MPLS: Label 6031 Exp 0] 16 msec 16 msec 15 msec
 4 10.5.6.5 19 msec 11 msec 10 msec
 5 10.4.8.8 [MPLS: Label 8020 Exp 0] 11 msec 20 msec 19 msec
 6 10.4.8.4 20 msec 10 msec 11 msec
```

```
R9#traceroute ipv6
Target IPv6 address: ::10:4:4:4
Source address: ::10:9:9:9
[snip]
 1 FD00:10:2:9::2 6 msec 3 msec 5 msec
 2 2024:24:2:14::14 [MPLS: Labels 94008/6015 Exp 0] 7 msec 7 msec 12 msec
 3 FD00:10:5:6::6 [MPLS: Label 6015 Exp 0] 17 msec 17 msec 20 msec
 4 FD00:10:5:6::5 22 msec 15 msec 41 msec
 5 FD00:10:4:8::8 [MPLS: Label 8027 Exp 0] 9 msec 17 msec 16 msec
 6 FD00:10:4:8::4 20 msec 10 msec 15 msec
```

Although there is nothing special required for option AB in particular, we bring up the backdoor link between CSR4 and CSR9, as well as the pair of sham-links from CSR2 to CSR8. Since the sham-link endpoints are ordinary VPNv6 prefixes from the perspective of BGP, they are carried via the traditional option A method between the ASBRs. I quickly verify that the sham-links and OSPFv3 backdoor neighbors are up.

```
R8#show ospfv3 vrf OSPF sham-links | include ^Sham
Sham Link OSPFv3_SL0 to address FD00::2 is up
Sham Link OSPFv3_SL1 to address FD00::2 is up
```

```
R4#show ospfv3 neighbor gig2.549
```

```
OSPFv3 2 address-family ipv4 (router-id 10.4.8.4)
```

Neighbor ID	Pri	State	Dead Time	Interface ID	Interface
10.4.9.9	0	FULL/ -	00:00:35	26	Gig2.549

```
OSPFv3 2 address-family ipv6 (router-id 10.4.8.4)
```

Neighbor ID	Pri	State	Dead Time	Interface ID	Interface
10.4.9.9	0	FULL/ -	00:00:37	26	Gig2.549

The sham-link signaling and construction was a result of the option A VPNv6 design, but IPv4 forwarding over the sham-link still follows the option AB mechanisms. The ASBRs still have eBGP VPNv4 routes in the global table for inter-as IPv4 prefixes, which are advertised to the RRs within each AS. We can check this quickly by comparing the VPNv4 and VPNv6 routes to CSR9. The VPNv4 route has a next-hop in the default table (which is overridden by the inter-as-hybrid functionality) while the VPNv6 route has a next-hop in VRF OSPF.

```
R5#show bgp vpnv4 unicast vrf OSPF 10.9.9.9
BGP routing table entry for 13:2:10.9.9.9/32, version 43
Paths: (1 available, best #1, table OSPF)
  Advertised to update-groups:
    1          2
  Refresh Epoch 1
  24, imported path from 24:2:10.9.9.9/32 (global)
    10.5.7.7 (via default) from 10.5.7.7 (24.0.0.7)
      Origin incomplete, localpref 100, valid, external, best
      Extended Community: RT:24:2 OSPF ROUTER ID:10.2.9.2:0
        OSPF RT:0.0.0.0:2:0
      mpls labels in/out 5006/7005
      rx pathid: 0, tx pathid: 0x0
```

```
R5#show bgp vpnv6 unicast vrf OSPF ::10:9:9:9/128
BGP routing table entry for [13:2>::10:9:9:9/128, version 341
Paths: (2 available, best #2, table OSPF)
  Advertised to update-groups:
```

```

1          3
Refresh Epoch 1
24
FD00:10:5:7::7 (FE80::7) (via vrf OSPF) from FD00:10:5:7::7 (24.0.0.7)
Origin incomplete, localpref 100, valid, external
Extended Community: RT:13:2 OSPF ROUTER ID:10.2.9.2:0
OSPF RT:0.0.0.0:2:0
mpls labels in/out 5012/nolabel
rx pathid: 0, tx pathid: 0
Refresh Epoch 1
24
FD00:10:5:6::6 (FE80::6) (via vrf OSPF) from FD00:10:5:6::6 (24.0.0.6)
Origin incomplete, localpref 100, valid, external, best
Extended Community: RT:13:2 OSPF ROUTER ID:10.2.9.2:0
OSPF RT:0.0.0.0:2:0
mpls labels in/out 5012/nolabel
rx pathid: 0, tx pathid: 0x0

```

Using traceroute, we can confirm the sham-links are operational. Both IPv4 and IPv6 have identical data-plane characteristics which resemble an option A architecture.

```

R4#traceroute 10.9.9.9 source 10.4.4.4
Type escape sequence to abort.
Tracing the route to 10.9.9.9
VRF info: (vrf in name/id, vrf out name/id)
 1 10.4.8.8 5 msec 3 msec 4 msec
 2 10.5.7.5 [MPLS: Label 5006 Exp 0] 4 msec 4 msec 4 msec
 3 10.5.7.7 6 msec 8 msec 10 msec
 4 24.7.14.14 [MPLS: Labels 94009/2017 Exp 0] 12 msec 19 msec 18 msec
 5 10.2.9.2 [MPLS: Label 2017 Exp 0] 19 msec 72 msec 14 msec
 6 10.2.9.9 16 msec 9 msec 8 msec

R4#traceroute ipv6
Target IPv6 address: ::10:9:9:9
Source address: ::10:4:4:4
[snip]
Tracing the route to ::10:9:9:9
 1 FD00:10:4:8::8 5 msec 3 msec 3 msec
 2 FD00:10:5:6::5 [MPLS: Label 5012 Exp 0] 4 msec 4 msec 4 msec
 3 FD00:10:5:6::6 23 msec 14 msec 15 msec
 4 ::FFFF:24.6.7.7 [MPLS: Labels 7004/2024 Exp 0] 17 msec 23 msec 21 msec
 5 FD00:10:2:9::2 [MPLS: Label 2024 Exp 0] 28 msec 22 msec 21 msec
 6 FD00:10:2:9::9 23 msec 17 msec 13 msec

```

To show the seamless interworking with existing option A architectures, I bring up the VRF-aware IPv4/v6 connections between CSR6 and XRv1; these were previously shutdown on CSR6 (removal of “shutdown” not shown). Using outbound RPL on XRv1, I increase the MED for routes advertised from AS

13 to AS 24. This will “hint” to AS 24 that XRv1 is a suboptimal point of ingress, which will prefer CSR5. This is so that the option AB links are preferred over the option A links.

```
! XRv1
route-policy RPL_SET_MED($MED)
  set med $MED
end-policy

router bgp 13
  vrf OSPF
    neighbor 10.6.11.6
      remote-as 24
      address-family ipv4 unicast
        route-policy RPL_SET_MED(11) out

    neighbor fd00:10:6:11::6
      address-family ipv6 unicast
        route-policy RPL_SET_MED(11) out
```

Checking CSR6 for the VPNv4 route to 10.4.4.4/32, we can see the MED value carried from XRv1. The eBGP route from CSR5 is preferred as a result of having a lower MED. So far, this has no impact on the traffic pattern since this was CSR6’s only option before. Also, we can see the next-hops are in different tables according to BGP; this is an indication that a combination of options A and AB are in use on this particular ASBR.

```
R6#show bgp vpnv4 unicast vrf OSPF 10.4.4.4/32
BGP routing table entry for 24:2:10.4.4.4/32, version 28
Paths: (2 available, best #2, table OSPF)
  Advertised to update-groups:
    3          1
  Refresh Epoch 1
  13
    10.6.11.11 (via vrf OSPF) from 10.6.11.11 (13.0.0.11)
      Origin incomplete, metric 11, localpref 100, valid, external
      Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
        OSPF RT:0.0.0.0:2:0
      mpls labels in/out 6027/nolabel
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  13, imported path from 13:2:10.4.4.4/32 (global)
    10.5.6.5 (via default) from 10.5.6.5 (13.0.0.5)
      Origin incomplete, localpref 100, valid, external, best
      Extended Community: RT:13:2 OSPF ROUTER ID:10.4.8.8:0
        OSPF RT:0.0.0.0:2:0
      Connector Attribute: count=1
        type 1 len 12 value 13:2:13.0.0.8
      mpls labels in/out 6027/5019
      rx pathid: 0, tx pathid: 0x0
```

To test XRv1 as a backup, I shut down the VPNv4 eBGP session from CSR6 to CSR5. Now, CSR7 is preferred since the MED of the iBGP learned route is still lower than the eBGP route. CSR9 will be routing to CSR5 via CSR7 now. XRv1 is still not used, which is the correct behavior. The current configuration identifies it as a backup for CSR5 entirely.

```
R6#show bgp vpnv4 unicast vrf OSPF 10.4.4.4/32
BGP routing table entry for 24:2:10.4.4.4/32, version 87
Paths: (2 available, best #1, table OSPF)
  Advertised to update-groups:
    3
  Refresh Epoch 2
  13
    24.0.0.7 (metric 10) (via default) from 24.0.0.2 (24.0.0.2)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
        OSPF RT:0.0.0.0:2:0
      Originator: 24.0.0.7, Cluster list: 24.0.0.2
      Connector Attribute: count=1
        type 1 len 12 value 13:2:13.0.0.8
      mpls labels in/out 6027/7018
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  13
    10.6.11.11 (via vrf OSPF) from 10.6.11.11 (13.0.0.11)
      Origin incomplete, metric 11, localpref 100, valid, external
      Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
        OSPF RT:0.0.0.0:2:0
      mpls labels in/out 6027/nolabel
      rx pathid: 0, tx pathid: 0
```

Using traceroute, we can confirm the new traffic pattern.

```
R9#traceroute 10.4.4.4 source 10.9.9.9
Type escape sequence to abort.
Tracing the route to 10.4.4.4
VRF info: (vrf in name/id, vrf out name/id)
 1 10.2.9.2 6 msec 4 msec 5 msec
 2 24.2.14.14 [MPLS: Labels 94010/7018 Exp 0] 6 msec 7 msec 5 msec
 3 10.5.7.7 [MPLS: Label 7018 Exp 0] 16 msec 16 msec 15 msec
 4 10.5.7.5 19 msec 11 msec 10 msec
 5 10.4.8.8 [MPLS: Label 8020 Exp 0] 11 msec 20 msec 19 msec
 6 10.4.8.4 21 msec 11 msec 10 msec
```

I also shut down CSR7's VPNv4 eBGP session to CSR5 which leaves XRv1 as the only remaining option. CSR6 reflects this accurately, and traceroute proves that the data plane flow is correction.


```

R6#show bgp vpnv4 unicast vrf OSPF 10.4.4.4/32
BGP routing table entry for 24:2:10.4.4.4/32, version 95
Paths: (1 available, best #1, table OSPF)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  13
    10.6.11.11 (via vrf OSPF) from 10.6.11.11 (13.0.0.11)
      Origin incomplete, metric 11, localpref 100, valid, external, best
      Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
      OSPF RT:0.0.0.0:2:0
      mpls labels in/out 6027/nolabel
      rx pathid: 0, tx pathid: 0x0

```

```

R9#traceroute 10.4.4.4 source 10.9.9.9
Type escape sequence to abort.
Tracing the route to 10.4.4.4
VRF info: (vrf in name/id, vrf out name/id)
 1 10.2.9.2 6 msec 4 msec 3 msec
 2 24.2.14.14 [MPLS: Labels 94008/6027 Exp 0] 5 msec 5 msec 8 msec
 3 10.6.11.6 [MPLS: Label 6027 Exp 0] 16 msec 16 msec 14 msec
 4 10.6.11.11 21 msec 13 msec 73 msec
 5 10.4.8.8 [MPLS: Label 8020 Exp 0] 11 msec 15 msec 15 msec
 6 10.4.8.4 15 msec 12 msec 11 msec

```

Before continuing, I restore all BGP sessions. Now that we have tested the OSPF VPN connectivity, I quickly check the EIGRP VPN. Both XRv3 and CSR1 are able to reach CSR3 without issue. For brevity, I do not trace the control plane signaling as it is identical in concept to the OSPF VPN.

```

R1#traceroute 10.3.3.3 source 10.1.1.1
Type escape sequence to abort.
Tracing the route to 10.3.3.3
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.2.2 5 msec 4 msec 4 msec
 2 24.2.14.14 [MPLS: Labels 94008/6026 Exp 0] 4 msec 5 msec 5 msec
 3 10.5.6.6 5 msec 8 msec 11 msec
 4 10.5.6.5 11 msec 10 msec 10 msec
 5 13.5.8.8 [MPLS: Labels 8010/92006 Exp 0] 16 msec 30 msec 19 msec
 6 13.8.12.12 [MPLS: Label 92006 Exp 0] 20 msec 19 msec 19 msec
 7 10.3.12.3 21 msec 11 msec 15 msec

```

```

RP/0/0/CPU0:XRv3#traceroute ::10:3:3:3 source ::10:13:13:13
Type escape sequence to abort.
Tracing the route to ::10:3:3:3
 1 fd00:10:13:14::14 0 msec 0 msec 0 msec
 2 fd00:10:5:6::6 [MPLS: Label 6019 Exp 0] 69 msec 0 msec 0 msec
 3 fd00:10:5:6::5 29 msec 19 msec 9 msec
 4 ::ffff:13.5.8.8 [MPLS: Labels 8010/92008 Exp 0] 9 msec 9 msec 0 msec

```

```

5  2013:13:8:12::12 [MPLS: Label 92008 Exp 0] 9 msec 0 msec 0 msec
6  fd00:10:3:12::3 0 msec 0 msec 0 msec

```

We also test central services to ensure it works over option AB. Since the VPNv4 routes advertised by the ASBRs (CSR6 and CSR7) have the local export RTs, not the eBGP-learned ones, we expect these to be accepted within an AS. CSR2 learns the VPN route from both CSR6 and CSR7. CSR6 is the best path but both VPN routes carry RT:24:3, which was exported from the ASBRs, not rewritten by BGP route-maps.

```

R2#show bgp vpnv4 unicast vrf EIGRP 110.0.0.3/32
BGP routing table entry for 24:3:110.0.0.3/32, version 1216
Paths: (2 available, best #1, table EIGRP)
  Advertised to update-groups:
    13
  Refresh Epoch 1
  13 100, (Received from a RR-client)
    24.0.0.6 (metric 20) (via default) from 24.0.0.6 (24.0.0.6)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:24:3
      Connector Attribute: count=1
        type 1 len 12 value 13:3:13.0.0.5
      mpls labels in/out nolabel/6043
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  13 100, (Received from a RR-client)
    24.0.0.7 (metric 20) (via default) from 24.0.0.7 (24.0.0.7)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Extended Community: RT:24:3
      Connector Attribute: count=1
        type 1 len 12 value 13:3:13.0.0.5
      mpls labels in/out nolabel/7038
      rx pathid: 0, tx pathid: 0

```

Option B central services is fully operational as expected.

```

R1#traceroute 110.0.0.3 source 10.1.1.1
Type escape sequence to abort.
Tracing the route to 110.0.0.3
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.2.2 2 msec 4 msec 4 msec
 2 24.2.14.14 [MPLS: Labels 94008/6043 Exp 0] 5 msec 6 msec 7 msec
 3 10.5.6.6 [MPLS: Label 6043 Exp 0] 16 msec 15 msec 16 msec
 4 10.5.6.5 20 msec 10 msec 11 msec
 5 10.8.10.8 [MPLS: Label 8019 Exp 0] 10 msec 20 msec 19 msec
 6 10.8.10.10 20 msec 10 msec 11 msec

```

```

RP/0/0/CPU0:XRv3#traceroute ::110:0:0:3 source ::10:13:13:13
Type escape sequence to abort.
Tracing the route to ::110:0:0:3

```

```

1  fd00:10:13:14::14 0 msec 0 msec 0 msec
2  fd00:10:5:6::6 [MPLS: Label 6041 Exp 0] 0 msec 0 msec 0 msec
3  fd00:10:5:6::5 0 msec 19 msec 119 msec
4  fd00:10:8:10::8 [MPLS: Label 8026 Exp 0] 29 msec 9 msec 0 msec
5  fd00:10:8:10::10 9 msec 0 msec 0 msec

```

Next, we will look at some minor variations of option AB. The design tested above is an “easy” one because the VPN next-hops and global next-hops are the same at the ASBRs. For example, CSR5 has a global eBGP VPNv4 session with 10.5.6.6, and this is also the VPN next-hop for all VRFs. This simplifies the configuration as manual option AB next-hops need not be specified. To better see the consequence of having different next-hops, I change the IPv4 address of CSR5’s transit link towards CSR6 inside both VRFs.

```

! CSR5
interface GigabitEthernet2.5562
 ip address 10.5.6.52 255.255.255.0

interface GigabitEthernet2.5563
 ip address 10.5.6.53 255.255.255.0

```

The VPN next-hop is still 10.5.6.5, and since the VRF did not specify a different next-hop, CSR6 still thinks 10.5.6.5 is accessible within the VPN. CSR6 continuously tries to ARP for this IPv4 next-hop as one would expect, but it does not exist.

```

R6#show ip route vrf OSPF 10.4.4.4
Routing Table: OSPF
Routing entry for 10.4.4.4/32
  Known via "bgp 24", distance 20, metric 0
  Tag 13, type external
  Last update from 10.5.6.5 00:16:03 ago
  Routing Descriptor Blocks:
  * 10.5.6.5, from 10.5.6.5, 00:16:03 ago
    Route metric is 0, traffic share count is 1
    AS Hops 1
    Route tag 13
    MPLS label: none

```

```

R6#show ip arp vrf OSPF 10.5.6.5
Protocol Address Age (min) Hardware Addr Type Interface
Internet 10.5.6.5 0 Incomplete ARPA

```

CSR2, the ingress LER, still prefers CSR6 as the VPN next-hop. This will blackhole all traffic in the OSPF VPN.

```

R2#show ip cef vrf OSPF 10.4.4.4 detail
10.4.4.4/32, epoch 0, flags [rib defined all labels]

```

```
recursive via 24.0.0.6 label 6046
nexthop 24.2.14.14 GigabitEthernet2.524 label 94008
```

```
R9#traceroute 10.4.4.4 source 10.9.9.9
Type escape sequence to abort.
Tracing the route to 10.4.4.4
VRF info: (vrf in name/id, vrf out name/id)
 1 10.2.9.2 4 msec 3 msec 4 msec
 2 * * *
```

The same is true for EIGRP as we changed that link address as well. CSR1 no longer has access to central service resources.

```
R2#show ip cef vrf EIGRP 110.0.0.3 detail
110.0.0.3/32, epoch 0, flags [rib defined all labels]
recursive via 24.0.0.6 label 6057
nexthop 24.2.14.14 GigabitEthernet2.524 label 94008
```

```
R1#traceroute 110.0.0.3 source 10.1.1.1
Type escape sequence to abort.
Tracing the route to 110.0.0.3
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.2.2 5 msec 3 msec 1 msec
 2 *
```

To fix it, we can use the same solution we configured initially on CSR5 as a demonstration. Under the VRFs on CSR6, I manually specify the VRF next-hops.

```
! CSR6
vrf definition EIGRP
  address-family ipv4
    inter-as-hybrid next-hop 10.5.6.53

vrf definition OSPF
  address-family ipv4
    inter-as-hybrid next-hop 10.5.6.52
```

Checking VPNv4, it is totally unaware of this change, at least from a VPN next-hop perspective. It still shows 10.5.6.5 as the VPN next-hop in the global table. Option AB overrides this and adjusts the next-hop to the value specified above.

```
R6#show bgp vpnv4 unicast vrf OSPF 10.4.4.4/32 bestpath
BGP routing table entry for 24:2:10.4.4.4/32, version 33
Paths: (2 available, best #1, table OSPF)
  Advertised to update-groups:
    7          6
  Refresh Epoch 2
```

```

13, imported path from 13:2:10.4.4.4/32 (global)
10.5.6.5 (via default) from 10.5.6.5 (13.0.0.5)
  Origin incomplete, localpref 100, valid, external, best
  Extended Community: RT:13:2 OSPF ROUTER ID:10.4.8.8:0
    OSPF RT:0.0.0.0:2:0
  Connector Attribute: count=1
    type 1 len 12 value 13:2:13.0.0.8
  mpls labels in/out 6046/5019
  rx pathid: 0, tx pathid: 0x0

```

```
R6#show ip route vrf OSPF 10.4.4.4
```

```
Routing Table: OSPF
```

```
Routing entry for 10.4.4.4/32
```

```
Known via "bgp 24", distance 20, metric 0
```

```
Tag 13, type external
```

```
Last update from 10.5.6.52 00:01:41 ago
```

```
Routing Descriptor Blocks:
```

```
* 10.5.6.52, from 10.5.6.5, 00:01:41 ago
```

```
Route metric is 0, traffic share count is 1
```

```
AS Hops 1
```

```
Route tag 13
```

```
MPLS label: none
```

Now, CSR6 can ARP for this IPv4 next-hop to bind a MAC address for encapsulation. CSR9's VPN connectivity to CSR4 and central services has been restored. The output is identical to earlier examples except the untagged inter-AS traffic transits through 10.5.6.52 rather than 10.5.6.5.

```
R6#show ip arp vrf OSPF 10.5.6.52
```

Protocol	Address	Age (min)	Hardware Addr	Type	Interface
Internet	10.5.6.52	9	0050.56a9.dc63	ARPA	Gig2.5562

```
R9#traceroute 10.4.4.4 source 10.9.9.9
```

```
Type escape sequence to abort.
```

```
Tracing the route to 10.4.4.4
```

```
VRF info: (vrf in name/id, vrf out name/id)
```

```

 1 10.2.9.2 6 msec 3 msec 4 msec
 2 24.2.14.14 [MPLS: Labels 94008/6046 Exp 0] 7 msec 7 msec 5 msec
 3 10.5.6.6 [MPLS: Label 6046 Exp 0] 5 msec 13 msec 20 msec
 4 10.5.6.52 20 msec 10 msec 10 msec
 5 10.4.8.8 [MPLS: Label 8020 Exp 0] 11 msec 19 msec 20 msec
 6 10.4.8.4 19 msec 11 msec 9 msec

```

```
R9#traceroute 110.0.0.0 source 10.9.9.9
```

```
Type escape sequence to abort.
```

```
Tracing the route to 110.0.0.0
```

```
VRF info: (vrf in name/id, vrf out name/id)
```

```

 1 10.2.9.2 4 msec 5 msec 4 msec
 2 24.2.14.14 [MPLS: Labels 94008/6048 Exp 0] 8 msec 7 msec 7 msec
 3 10.5.6.6 [MPLS: Label 6048 Exp 0] 6 msec 8 msec 20 msec

```

```

4 10.5.6.52 20 msec 11 msec 10 msec
5 10.8.10.8 [MPLS: Label 8016 Exp 0] 11 msec 19 msec 19 msec
6 10.8.10.10 18 msec 11 msec 11 msec

```

The same is true for EIGRP. On CSR6, the VPN routes to CSR3 has a global next-hop of 10.5.6.5 which was the eBGP VPNv4 peer address. The option AB hybrid configuration overrides this to correct the VRF-aware RIB, and ultimately the VRF-aware FIB. CSR6 is able to ARP for this MAC address once the VRF-aware route is updated with the manually-configured next-hop.

```

R6#show bgp vpnv4 unicast vrf EIGRP 10.3.3.3/32 bestpath
BGP routing table entry for 24:3:10.3.3.3/32, version 39
Paths: (1 available, best #1, table EIGRP)
  Advertised to update-groups:
    6
  Refresh Epoch 2
  13, imported path from 13:3:10.3.3.3/32 (global)
    10.5.6.5 (via default) from 10.5.6.5 (13.0.0.5)
      Origin incomplete, localpref 100, valid, external, best
      Extended Community: RT:13:3 0x8800:32768:0 0x8801:3:288
        0x8802:65281:2560 0x8803:1:1500 0x8806:0:167971843
      Connector Attribute: count=1
        type 1 len 12 value 13:3:13.0.0.12
      mpls labels in/out 6052/5020
      rx pathid: 0, tx pathid: 0x0

```

```

R6#show ip route vrf EIGRP 10.3.3.3
Routing Table: EIGRP
Routing entry for 10.3.3.3/32
  Known via "bgp 24", distance 20, metric 0
  Tag 13, type external
  Last update from 10.5.6.53 00:09:08 ago
  Routing Descriptor Blocks:
  * 10.5.6.53, from 10.5.6.5, 00:09:08 ago
    Route metric is 0, traffic share count is 1
    AS Hops 1
    Route tag 13
    MPLS label: none

```

```

R6#show ip arp vrf EIGRP 10.5.6.53
Protocol Address Age (min) Hardware Addr Type Interface
Internet 10.5.6.53 15 0050.56a9.dc63 ARPA Gig2.5563

```

A set of traceroutes confirms that EIGRP customers have VPN and central services reachability.

```

RP/0/0/CPU0:XRv3#traceroute 10.3.3.3 source 10.13.13.13
Type escape sequence to abort.
Tracing the route to 10.3.3.3
 1 10.13.14.14 0 msec 0 msec 0 msec

```

```

2 10.5.6.6 [MPLS: Label 6052 Exp 0] 0 msec 0 msec 0 msec
3 10.5.6.53 0 msec 0 msec 0 msec
4 13.5.8.8 [MPLS: Labels 8010/92006 Exp 0] 0 msec 0 msec 0 msec
5 13.8.12.12 [MPLS: Label 92006 Exp 0] 0 msec 0 msec 0 msec
6 10.3.12.3 9 msec 0 msec 0 msec

```

```
RP/0/0/CPU0:XRv3#traceroute 110.0.0.2 source 10.13.13.13
```

```
Type escape sequence to abort.
```

```
Tracing the route to 110.0.0.2
```

```

1 10.13.14.14 9 msec 0 msec 0 msec
2 10.5.6.6 [MPLS: Label 6056 Exp 0] 0 msec 0 msec 0 msec
3 10.5.6.53 0 msec 0 msec 0 msec
4 10.8.10.8 [MPLS: Label 8018 Exp 0] 0 msec 0 msec 0 msec
5 10.8.10.10 0 msec 0 msec 0 msec

```

Last, we can test the CSC extension to option AB. This is generally useful for when the CSC core carrier is actually two carriers. For example, if AS 13 and AS 24 were the conglomerate “core carrier” while CSR4 and CSR9 represented customer carriers, the traffic would need to remain MPLS encapsulated end-to-end. The CSC feature with option AB supports this and is very easy to configure; it is supported in conjunction with an explicit or implicit next-hop as shown below. I enable CSC for VRF OSPF only, so that VRF EIGRP will remain only IP-encapsulated over the transit link.

```

! CSR6
vrf definition OSPF
  address-family ipv4
    inter-as-hybrid csc next-hop 10.5.6.52

```

```

! CSR5 and CSR7
vrf definition OSPF
  address-family ipv4
    inter-as-hybrid csc

```

The only difference in behavior is that the MPLS labels exchanged with the eBGP VPN routes are swapped at the ASBRs, much like option B. On CSR6, we can see the next-hop in the BGP route is still incorrect as the manual configuration overrides it. However, VPN label swapping occurs like option B.

```

R6#show bgp vpnv4 unicast vrf OSPF 10.4.4.4 bestpath
BGP routing table entry for 24:2:10.4.4.4/32, version 33
Paths: (2 available, best #1, table OSPF)
  Advertised to update-groups:
    7          6
  Refresh Epoch 2
  13, imported path from 13:2:10.4.4.4/32 (global)
    10.5.6.5 (via default) from 10.5.6.5 (13.0.0.5)
      Origin incomplete, localpref 100, valid, external, best
      Extended Community: RT:13:2 OSPF ROUTER ID:10.4.8.8:0
      OSPF RT:0.0.0.0:2:0

```

```
Connector Attribute: count=1
type 1 len 12 value 13:2:13.0.0.8
mpls labels in/out 6046/5019
rx pathid: 0, tx pathid: 0x0
```

The RIB indicates an outgoing MPLS label of 5019, which is the VPN label carried in the eBGP route seen above. Despite this label always being exchanged, it was not delivered to the RIB/FIB for encapsulation when the CSC option was disabled.

```
R6#show ip route vrf OSPF 10.4.4.4
Routing Table: OSPF
Routing entry for 10.4.4.4/32
  Known via "bgp 24", distance 20, metric 0
  Tag 13, type external
  Last update from 10.5.6.52 00:01:43 ago
  Routing Descriptor Blocks:
  * 10.5.6.52, from 10.5.6.5, 00:01:43 ago
    Route metric is 0, traffic share count is 1
    AS Hops 1
    Route tag 13
    MPLS label: 5019
```

```
R6#show ip cef vrf OSPF 10.4.4.4
10.4.4.4/32
  nexthop 10.5.6.52 GigabitEthernet2.5562 label 5019
```

If we examine the VPNv4 debugging on CSR6, we can see that all of the prefixes learned from CSR5 have the same label value within each VPN. Even though we didn't even enable CSC support on VRF EIGRP, the label aggregation occurs for that VPN also. We can clearly see that CSC is not enabled for VRF EIGRP.

```
! CSR5
R5#show vrf detail | include Hybrid|^VRF
VRF EIGRP (VRF Id = 1); default RD 13:3; default VPNID <not set>
  Inter AS Hybrid mode configured, next-hop 10.5.6.6
VRF OSPF (VRF Id = 2); default RD 13:2; default VPNID <not set>
  Inter AS Hybrid mode configured, with CSC
```

This is a per-VRF label that is advertised inside of the VPNv4 update. Label aggregation is discussed in the multi-VRF CE chapter, but in summary, this is used in contract to the per-prefix label. A single label represents all VPN routes inside of a VRF, which conserves labels. The downside of this approach is that the payload of the MPLS packet must be consulted for a routing decision. CSR5's LFIB proves this; when packets arrive with label 5019, the label is removed and the OSPF RIB is consulted.

```
R6#debug bgp vpnv4 unicast updates 10.5.6.5 in
BGP updates debugging is on for neighbor 10.5.6.5 (inbound) for address
family: VPNv4 Unicast
```



```

BGP(4): 10.5.6.5 rcvd 13:2:110.0.0.1/32, label 5019
BGP(4): 10.5.6.5 rcvd UPDATE w/ attr: nexthop 10.5.6.5, origin ?, merged path
13 100, AS_PATH , extended community RT:13:2
BGP(4): 10.5.6.5 rcvd UPDATE w/ attr: nexthop 10.5.6.5, origin ?, merged path
13, AS_PATH , extended community RT:13:2 OSPF ROUTER ID:10.4.8.8:0 OSPF
RT:0.0.0.0:2:0
BGP(4): 10.5.6.5 rcvd 13:2:10.4.4.4/32, label 5019
BGP(4): 10.5.6.5 rcvd UPDATE w/ attr: nexthop 10.5.6.5, origin ?, merged path
13, AS_PATH , extended community RT:13:2 OSPF ROUTER ID:10.4.8.8:0 OSPF
RT:0.0.0.0:2:0
BGP(4): 10.5.6.5 rcvd 13:2:10.4.8.0/24, label 5019

```

```
R5#show mpls forwarding-table labels 5019
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
5019	Pop Label	IPv4 VRF[V]	6874	aggregate/OSPF	

CSR5 allocates label 5019 for VRF OSPF and 2050 for VRF EIGRP. We quickly check the VPNv4 label bindings on CSR5 to verify this. CSR6 also allocates aggregated labels per VPN for the return traffic.

```
R5#show bgp vpnv4 unicast vrf OSPF labels
```

Network	Next Hop	In label/Out label
Route Distinguisher: 13:2 (OSPF)		
10.2.9.0/24	10.5.7.7	IPv4 VRF Aggr:5019/7005
10.4.4.4/32	13.0.0.8	IPv4 VRF Aggr:5019/8020
10.4.8.0/24	13.0.0.8	IPv4 VRF Aggr:5019/8021
10.9.9.9/32	10.5.7.7	IPv4 VRF Aggr:5019/7005
110.0.0.0/32	13.0.0.8	IPv4 VRF Aggr:5019/8016
110.0.0.1/32	13.0.0.8	IPv4 VRF Aggr:5019/8017
110.0.0.2/32	13.0.0.8	IPv4 VRF Aggr:5019/8018
110.0.0.3/32	13.0.0.8	IPv4 VRF Aggr:5019/8019

```
R5#show bgp vpnv4 unicast vrf EIGRP labels
```

Network	Next Hop	In label/Out label
Route Distinguisher: 13:3 (EIGRP)		
10.1.1.1/32	10.5.7.7	5016/7006
10.1.2.0/24	10.5.7.7	5026/7006
10.1.13.0/24	10.5.7.7	5027/7006
10.3.3.3/32	13.0.0.12	IPv4 VRF Aggr:5020/92006
10.3.12.0/24	13.0.0.12	IPv4 VRF Aggr:5020/92007
10.13.13.13/32	10.5.7.7	5028/7006
10.13.14.0/24	10.5.7.7	5029/7006
110.0.0.0/32	13.0.0.8	IPv4 VRF Aggr:5020/8016
110.0.0.1/32	13.0.0.8	IPv4 VRF Aggr:5020/8017
110.0.0.2/32	13.0.0.8	IPv4 VRF Aggr:5020/8018
110.0.0.3/32	13.0.0.8	IPv4 VRF Aggr:5020/8019

The EIGRP VPN has connectivity still, as expected, since nothing really changed. We use EPC outbound on CSR6 to confirm that MPLS encapsulation is not observed. Since QinQ is used, the Ethernet payload is another dot1q frame. We do not see ethertype 0x8847 in the packet and the length of 522 suggests that the packet is not MPLS encapsulated (14 Ethernet + 4 dot1q + 4 second dot1q). The IP addresses in the hex dump are shown in green.

```
R1#ping 10.3.3.3 source 10.1.1.1 size 500
Type escape sequence to abort.
Sending 5, 500-byte ICMP Echos to 10.3.3.3, timeout is 2 seconds:
Packet sent with a source address of 10.1.1.1
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 14/61/96 ms

R6#show monitor capture CAP buffer detailed
  0 522 0.000000 00:50:56:A9:DE:0D -> 00:50:56:A9:DC:63 VLAN-tagged
frame
  0000: 005056A9 DC630050 56A9DE0D 81000DE4 .PV..c.PV.....
  0010: 81000003 08004500 01F405D0 0000FC01 .....E.....
  0020: 9F310A01 01010A03 03030800 CA300016 .1.....0..
  0030: 00000000 00003F9A 3BBBABCD ABCDABCD .....?.;.....
```

We can prove that the traffic is untagged another way. CSR2 uses VPN label 6026, which is allocated by CSR6. The LSRs along the path inside AS 24 will forward the packet to the penultimate hop, which performs PHP to reveal label 6026 to CSR6. CSR6's LFIB performs a pop, not a swap, and then uses the MPLS payload for a routing decision. This is the expected behavior for VRF-wide aggregate labels since there is no per-prefix or per-CE (per-next-hop) granularity. Forwarding traffic based on the incoming label alone is ambiguous to a second lookup is necessary. The traffic is routed without addition labels directly to CSR5 across the VRF-aware transit link.

```
R2#show ip cef vrf EIGRP 10.3.3.3
10.3.3.3/32
  nexthop 24.2.14.14 GigabitEthernet2.524 label 94008 6026

R6#show mpls forwarding-table labels 6026
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
6026      Pop Label  IPv4 VRF[V]     8296         aggregate/EIGRP

R6#show ip cef vrf EIGRP 10.3.3.3/32
10.3.3.3/32
  nexthop 10.5.6.53 GigabitEthernet2.5563
```

Pinging in VRF OSPF does not succeed and EPC reveals nothing. Since CSR6 would essentially be swapping the VPN label 6064 with the VPN aggregate label 5019, we expect MPLS to be functional. However, packets are being dropped per the LFIB. The in/out labels are correct but the LFIB does not specify an outgoing interface.

```
R9#ping 10.4.4.4 source 10.9.9.9 size 500
Type escape sequence to abort.
Sending 5, 500-byte ICMP Echos to 10.4.4.4, timeout is 2 seconds:
Packet sent with a source address of 10.9.9.9
```

```
.....
Success rate is 0 percent (0/5)
```

```
R2#show ip cef vrf OSPF 10.4.4.4/32
10.4.4.4/32
  nexthop 24.2.14.14 GigabitEthernet2.524 label 94008 6046
```

```
R6#show mpls forwarding-table labels 6046
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
6046       5019      10.4.4.4/32 [V]  0            drop
```

The solution is to enable MPLS on the VRF-aware transit links for BGP forwarding. Normally this is configured automatically with option B, but since there are no VRF-aware VPNv4 sessions, we must configure it manually on all transit links expected to support CSC. This is only needed on the VRF OSPF interfaces as VRF EIGRP is not configured for CSC support.

```
! CSR6
interface GigabitEthernet2.5562
 mpls bgp forwarding
```

```
! CSR7
interface GigabitEthernet2.5572
 mpls bgp forwarding
```

```
! CSR5
interface GigabitEthernet2.5562
 mpls bgp forwarding
```

```
interface GigabitEthernet2.5572
 mpls bgp forwarding
```

I quickly verify that BGP-based MPLS is enabled on all VRF-aware transit links. Without this command, all of the control plane functions appear functional (BGP, routing, etc) but forwarding does not work.

```
R5#show mpls interfaces vrf OSPF
Interface          IP          Tunnel  BGP Static Operational
VRF OSPF:
GigabitEthernet2.5562 No          No      Yes  No      Yes
GigabitEthernet2.5572 No          No      Yes  No      Yes
```

```
R6#show mpls interfaces vrf OSPF
```

Interface	IP	Tunnel	BGP	Static	Operational
VRF OSPF:					
GigabitEthernet2.5562	No	No	Yes	No	Yes
Lspvif1	No	No	No	No	Yes

```
R7#show mpls interfaces vrf OSPF
```

Interface	IP	Tunnel	BGP	Static	Operational
VRF OSPF:					
GigabitEthernet2.5572	No	No	Yes	No	Yes
Lspvif1	No	No	No	No	Yes

Now, when we verify the LFIB, we can see the label swap occurring properly. The LFIB is now aware that BGP is allowed to perform MPLS forwarding operations on this interface, allowing it to be added to the LFIB properly.

```
R6#show mpls forwarding-table labels 6046
```

Local	Outgoing	Prefix	Bytes	Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched		interface	
6046	5019	10.4.4.4/32 [V]	0		Gi2.5562	10.5.6.52

Using ping and EPC, we confirm that the traffic is MPLS-encapsulated with a single label value of 5019 as it is sent towards CSR5. The packet size is now 526, which is 4 bytes larger than the untagged packet in the EIGRP VPN. Label 5019 is in the stack (0x139B), which makes this a valuable tool for CSC operations. The VPN IP addresses is highlighted in green again.

```
R9#ping 10.4.4.4 source 10.9.9.9 size 500
```

```
Type escape sequence to abort.
```

```
Sending 5, 500-byte ICMP Echos to 10.4.4.4, timeout is 2 seconds:
```

```
Packet sent with a source address of 10.9.9.9
```

```
!!!!
```

```
Success rate is 100 percent (5/5), round-trip min/avg/max = 13/59/89 ms
```

```
R6#show monitor capture CAP buffer detailed
```

```
1 526 0.851998 00:50:56:A9:DE:0D -> 00:50:56:A9:DC:63 VLAN-tagged frame
```

```
0000: 005056A9 DC630050 56A9DE0D 81000DE4 .PV..c.PV.....
0010: 81000002 88470139 B1FC4500 01F4004D .....G.9..E....M
0020: 0000FE01 99A20A09 09090A04 04040800 .....
0030: 50360016 00000000 0000A525 502AABCD P6.....%P*..
```

We confirm that the LFIB counters increment when sending traffic. This is expected regardless of whether the outgoing label is an aggregate label or not. The reason it is important to verify this feature with EPC and packet counters is because VPN traceroute does not reveal this interim LSP, as shown below. I suspect the reason is because CSR5 perform its final routing lookup based on the MPLS payload (IP packet), not the MPLS aggregate label, so the reply does not include that label.

```
R6#show mpls forwarding-table labels 6046
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
6046	5019	10.4.4.4/32 [V]	7060	Gi2.5562	10.5.6.52

```
R9#traceroute 10.4.4.4 source 10.9.9.9
Type escape sequence to abort.
Tracing the route to 10.4.4.4
VRF info: (vrf in name/id, vrf out name/id)
 1 10.2.9.2 6 msec 4 msec 4 msec
 2 24.2.14.14 [MPLS: Labels 94008/6046 Exp 0] 7 msec 6 msec 6 msec
 3 10.5.6.6 [MPLS: Label 6046 Exp 0] 7 msec 12 msec 19 msec
 4 10.5.6.52 19 msec 11 msec 11 msec
 5 10.4.8.8 [MPLS: Label 8020 Exp 0] 10 msec 19 msec 20 msec
 6 10.4.8.4 20 msec 10 msec 9 msec
```

To perform a quick verification on CSR7, the backup CSC path, I shut down CSR6's BGP session to CSR5. CSR6's new best-path becomes a non-CSC enabled link to XRv1, which should never be used as it would blackhole traffic. The aggregate label was never generated on XRv1 and not even exchanged with CSR6. I leave this link operational for demonstration, but BGP MED ensures that CSR7 will be preferred as an alternate. We can also see label 7005, which we assume to be CSR7's local aggregate label for VRF OSPF.

```
R6#show bgp vpnv4 unicast vrf OSPF 10.4.4.4/32
BGP routing table entry for 24:2:10.4.4.4/32, version 162
Paths: (2 available, best #1, table OSPF)
  Advertised to update-groups:
    7
  Refresh Epoch 2
  13
    24.0.0.7 (metric 10) (via default) from 24.0.0.2 (24.0.0.2)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
      OSPF RT:0.0.0.0:2:0
      Originator: 24.0.0.7, Cluster list: 24.0.0.2
      Connector Attribute: count=1
      type 1 len 12 value 13:2:13.0.0.8
      mpls labels in/out 6046/7005
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  13
    10.6.11.11 (via vrf OSPF) from 10.6.11.11 (13.0.0.11)
      Origin incomplete, metric 11, localpref 100, valid, external
      Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
      OSPF RT:0.0.0.0:2:0
      mpls labels in/out 6046/nolabel
      rx pathid: 0, tx pathid: 0
```

We verify that label 7005 is, in fact, the local aggregate label for VRF OSPF. This should be swapped to 5019, which was CSR5's aggregate label for VRF OSPF. This hasn't changed as a result of shutting CSR6 down as CSR5 advertises it to all BGP peers for all prefixes. Since CSR2 will prefer the VPN route from CSR7 as the best path, we expect 7005 to be the VPN label for 10.4.4.4/32, as well as other VPN prefixes.

```
R7#show bgp vpnv4 unicast vrf OSPF labels
  Network          Next Hop      In label/Out label
Route Distinguisher: 24:2 (OSPF)
 10.2.9.0/24       24.0.0.2      IPv4 VRF Aggr:7005/2015
 10.4.4.4/32       10.5.7.5      IPv4 VRF Aggr:7005/5019
 10.4.8.0/24       10.5.7.5      IPv4 VRF Aggr:7005/5019
 10.9.9.9/32       24.0.0.2      IPv4 VRF Aggr:7005/2017
 110.0.0.0/32      10.5.7.5      IPv4 VRF Aggr:7005/5019
 110.0.0.1/32      10.5.7.5      IPv4 VRF Aggr:7005/5019
 110.0.0.2/32      10.5.7.5      IPv4 VRF Aggr:7005/5019
 110.0.0.3/32      10.5.7.5      IPv4 VRF Aggr:7005/5019
```

```
R2#show ip cef vrf OSPF 10.4.4.4/32
10.4.4.4/32
  nexthop 24.2.14.14 GigabitEthernet2.524 label 94010 7005
```

We notice a slight change in CSR7's local label allocation for 10.4.4.4/32 compared to CSR6. CSR6 allocated per-prefix local labels for inter-AS VPN routes, such as 6046, which could be swapped to the peer's aggregate label without invoking an IPv4 FIB lookup inside the VPN. CSR7 does not explicitly specify a VPN next-hop under the VRF configuration. Without CSC, this had no impact on performance, but in this case, CSR7 has to remove the local aggregate label, and then perform an IPv4 FIB lookup in the VPN while pushing the peer's aggregate label. The VPN-aware FIB lookup still relies on the existing 10.5.7.5 next-hop, which was unchanged, so the design is valid. This two-step operation (pop/push rather than swap) is less efficient, which is why specifying the next-hop is a good design choice when using the CSC option with option AB.

```
R7#show mpls forwarding-table labels 7005
Local      Outgoing   Prefix      Bytes Label  Outgoing   Next Hop
Label      Label      or Tunnel Id  Switched     interface
7005       Pop Label  IPv4 VRF[V]  13670        aggregate/OSPF
```

```
R7#show ip cef vrf OSPF 10.4.4.4/32
10.4.4.4/32
  nexthop 10.5.7.5 GigabitEthernet2.5572 label 5019
```

This makes the traceroute even more awkward looking inside the VPN. Since CSR7 and CSR5 are both performing FIB (not LFIB) lookups to route the VPN traffic, neither of them include the MPLS labels associated with each hop.

```
R9#traceroute 10.4.4.4 source 10.9.9.9
Type escape sequence to abort.
```

```

Tracing the route to 10.4.4.4
VRF info: (vrf in name/id, vrf out name/id)
 1 10.2.9.2 6 msec 4 msec 4 msec
 2 24.2.14.14 [MPLS: Labels 94010/7005 Exp 0] 6 msec 5 msec 6 msec
 3 10.5.7.7 5 msec 7 msec 9 msec
 4 10.5.7.5 10 msec 10 msec 9 msec
 5 10.4.8.8 [MPLS: Label 8020 Exp 0] 13 msec 22 msec 18 msec
 6 10.4.8.4 21 msec 10 msec 16 msec

```

A quick EPC check on CSR7 outbound towards CSR5 shows label 5019 in the stack. This MPLS encapsulation is required for CSC service to be operational. Before continuing, CSR6's BGP session with CSR5 is restored.

```

R7#show monitor capture CAP buffer detailed
 0 526      0.000000 00:50:56:A9:EA:77 -> 00:50:56:A9:DC:63 VLAN-tagged
frame
0000: 005056A9 DC630050 56A9EA77 81000DE5 .PV..c.PV..w....
0010: 81000002 88470139 B1FC4500 01F4005C .....G.9..E....\
0020: 0000FC01 9B930A09 09090A04 04040800 .....
0030: 18FF0019 00000000 0000A53D 8746ABCD .....=.F..

```

Some of these inconsistencies appeared strange, and furthermore, the documentation clearly states that “per-prefix” labels should be allocated when CSC is enabled. After reloading the ASBRs, the behavior changes significantly. I assume this is the result of a configuration order-of-operations issue despite the configuration above being functional. Below, I show CSR5 and CSR6 label allocations for VPNv4 inside VRF OSPF; there are no aggregate labels. On CSR6, notice that label swapping only occurs when CSR5 is the next-hop, which is accurate since XRv1 only supports option A in this architecture.

```

R6#show bgp vpnv4 unicast vrf OSPF labels
  Network          Next Hop          In label/Out label
Route Distinguisher: 24:2 (OSPF)
 10.2.9.0/24       24.0.0.2          6048/2013
 10.4.4.4/32       10.5.6.5          6036/5021
                   10.6.11.11       6036/nolabel
 10.4.8.0/24       10.5.6.5          6037/5022
                   10.6.11.11       6037/nolabel
 10.9.9.9/32       24.0.0.2          6049/2015
 110.0.0.0/32      10.5.6.5          6038/5023
                   10.6.11.11       6038/nolabel
 110.0.0.1/32      10.5.6.5          6039/5024
                   10.6.11.11       6039/nolabel
 110.0.0.2/32      10.5.6.5          6040/5025
                   10.6.11.11       6040/nolabel
 110.0.0.3/32      10.5.6.5          6041/5026
                   10.6.11.11       6041/nolabel

```

```

R5#show bgp vpnv4 unicast vrf OSPF labels

```

Network	Next Hop	In label/Out label
Route Distinguisher: 13:2 (OSPF)		
10.2.9.0/24	10.5.6.6	5009/6048
10.4.4.4/32	13.0.0.8	5021/8007
10.4.8.0/24	13.0.0.8	5022/8008
10.9.9.9/32	10.5.6.6	5010/6049
110.0.0.0/32	13.0.0.8	5023/8003
110.0.0.1/32	13.0.0.8	5024/8004
110.0.0.2/32	13.0.0.8	5025/8005
110.0.0.3/32	13.0.0.8	5026/8006

The implication of this is that the LSPs will operate more efficiently now. For example, if traffic arrives at CSR5 with label 5010 and destined for 10.9.9.9/32 in the remote AS, it will be swapped directly to 6049 in a single LFIB lookup. Label 6036 entering CSR6 along the reverse LSP should behave similarly by swapping directly to label 5021 and sending to CSR5. With a next-hop interface in the LFIB below, there is no need for another FIB lookup. The comments earlier about next-hop specification with CSC are no longer true since CSR6 and CSR5 both behave identically now. Again, this is probably the result of a configuration sequence error, but I leave the existing documentation in place as a troubleshooting aid.

```
R5#show mpls forwarding-table labels 5010
```

Local	Outgoing	Prefix	Bytes Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched	interface	
5010	6049	10.9.9.9/32 [V]	3402	Gi2.5562	10.5.6.6

```
R6#show mpls forwarding-table labels 6036
```

Local	Outgoing	Prefix	Bytes Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched	interface	
6036	5021	10.4.4.4/32 [V]	7512	Gi2.5562	10.5.6.52

Traceroute proves this as the path looks more like “option B” from a traceroute perspective. I recommend ensuring that, when inter-AS option AB with CSC is enabled, labels are allocated per-prefix. The only method that worked in this setup was an ASBR reload. This is true in both directions.

```
R4#traceroute 10.9.9.9 source 10.4.4.4
```

```
Type escape sequence to abort.
```

```
Tracing the route to 10.9.9.9
```

```
VRF info: (vrf in name/id, vrf out name/id)
```

```

 1 10.4.8.8 4 msec 4 msec 3 msec
 2 10.5.6.52 [MPLS: Label 5010 Exp 0] 7 msec 9 msec 7 msec
 3 10.5.6.6 [MPLS: Label 6049 Exp 0] 27 msec 29 msec 32 msec
 4 24.6.14.14 [MPLS: Labels 94009/2015 Exp 0] 29 msec 31 msec 32 msec
 5 10.2.9.2 [MPLS: Label 2015 Exp 0] 15 msec 16 msec 14 msec
 6 10.2.9.9 20 msec 10 msec 16 msec
```

```
R9#traceroute 10.4.4.4 source 10.9.9.9
```

```
Type escape sequence to abort.
```

```
Tracing the route to 10.4.4.4
```



```
VRF info: (vrf in name/id, vrf out name/id)
 1 10.2.9.2 4 msec 5 msec 5 msec
 2 24.2.14.14 [MPLS: Labels 94008/6036 Exp 0] 9 msec 10 msec 10 msec
 3 10.5.6.6 [MPLS: Label 6036 Exp 0] 23 msec 31 msec 31 msec
 4 10.5.6.52 [MPLS: Label 5021 Exp 0] 30 msec 31 msec 31 msec
 5 10.4.8.8 [MPLS: Label 8007 Exp 0] 19 msec 29 msec 20 msec
 6 10.4.8.4 20 msec 12 msec 11 msec
```

8.4.4.2 L2VPN

MPLS L2VPN using option AB works identically as it does for option A. Since only VPNv4 is supported, and the whole idea of option AB is to isolate customer traffic into separate VPN transit links, an option B-like forwarding mechanism would violate the general design. This section is largely similar to the option A L2VPN section but is verified briefly for completeness. We will limit our verifications to CSR5 and CSR6; no configuration changes were made between option A and option AB (this section). First, I verify that the ASBRs have a single “core PW” over MPLS to their respective PEs, as well as the VFI interface that links the bridge-domain. All connections are reported as being up.

```
R5#show l2vpn service vfi name VPLS | begin Interface
Interface          Group          Encapsulation          Prio   St   XC St
-----
VPLS name: VPLS, State: UP
pw100001           VPLS (VFI)           0           UP   UP
pw100002           core_pw              13.0.0.8:3 (MPLS)      0           UP   UP

R6#show l2vpn service vfi name VPLS | begin Interface
Interface          Group          Encapsulation          Prio   St   XC St
-----
VPLS name: VPLS, State: UP
pw100001           VPLS (VFI)           0           UP   UP
pw100002           core_pw              2:3 (MPLS)          0           UP   UP
```

Next, I check for the data-plane learning on the ASBRs. I first record the MAC address of CSR1 (yellow) and CSR3 (green) for easy identification.

```
R1#show interfaces gig2 | include bia
Hardware is CSR vNIC, address is 0050.56a9.1aaa (bia 0050.56a9.1aaa)

R3#show interfaces gig2 | include bia
Hardware is CSR vNIC, address is 0050.56a9.8ccf (bia 0050.56a9.8ccf)
```

Looking at the ASBR bridge-domains, we can see both MAC addresses are learned. On CSR5, CSR1’s MAC is learned via the VPLS PW to CSR2 while CSR3’s MAC is learned via the inter-AS EFP. The opposite is true on CSR6.

```
R5#show bridge-domain 3
Bridge-domain 3 (2 ports in all)
```

```

State: UP                               Mac learning: Enabled
Aging-Timer: 300 second(s)
  GigabitEthernet2 service instance 30
    vfi VPLS neighbor 13.0.0.8 3
AED MAC address    Policy Tag          Age Pseudoport
0  0050.56A9.1AAA  forward dynamic  221 GigabitEthernet2.EFP30
1  FFFF.FFFF.FFFF flood   static    0  OLIST_PTR:0xe80d0400
0  0050.56A9.8CCF forward dynamic  236 VPLS.1004011

```

```

R6#show bridge-domain 3
Bridge-domain 3 (2 ports in all)
State: UP                               Mac learning: Enabled
Aging-Timer: 300 second(s)
  GigabitEthernet2 service instance 30
    vfi VPLS neighbor 24.0.0.2 3
AED MAC address    Policy Tag          Age Pseudoport
0  0050.56A9.1AAA  forward dynamic  213 VPLS.1004011
1  FFFF.FFFF.FFFF flood   static    0  OLIST_PTR:0xe8125400
0  0050.56A9.8CCF forward dynamic  228 GigabitEthernet2.EFP30

```

A similar bridge-domain MAC address check on CSR2 and CSR8 confirms that the MAC addresses are being learned properly. The PE learns its local CE via the PE-CE EFP and the remote CE via the VPLS pseudoport towards the neighboring ASBR.

```

R2#show bridge-domain 3
Bridge-domain 3 (2 ports in all)
State: UP                               Mac learning: Enabled
Aging-Timer: 300 second(s)
  GigabitEthernet2 service instance 3
    vfi VPLS neighbor 24.0.0.6 3
AED MAC address    Policy Tag          Age Pseudoport
0  0050.56A9.1AAA  forward dynamic  216 GigabitEthernet2.EFP3
1  FFFF.FFFF.FFFF flood   static    0  OLIST_PTR:0xe80e9400
0  0050.56A9.8CCF forward dynamic  246 VPLS.1004011

```

```

R8#show bridge-domain 3
Bridge-domain 3 (2 ports in all)
State: UP                               Mac learning: Enabled
Aging-Timer: 300 second(s)
  GigabitEthernet2 service instance 3
    vfi VPLS neighbor 13.0.0.5 3
AED MAC address    Policy Tag          Age Pseudoport
0  0050.56A9.1AAA  forward dynamic  211 VPLS.1004011
1  FFFF.FFFF.FFFF flood   static    0  OLIST_PTR:0xe80a9800
0  0050.56A9.8CCF forward dynamic  242 GigabitEthernet2.EFP3

```

Ping and traceroute indicate the VPLS has formed and is functioning properly.

```

R3#ping vrf VPLS 10.0.0.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.1, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 9/10/14 ms

R3#traceroute vrf VPLS 10.0.0.1
Type escape sequence to abort.
Tracing the route to 10.0.0.1
VRF info: (vrf in name/id, vrf out name/id)
 1 10.0.0.1 12 msec 10 msec 9 msec

```

8.4.4.3 MVPN - GRE (Profile 0) and mLDP (Profile 1)

MVPN over option AB is similar to option A but is worth investigating because the control-plane changed for VPNv4. AS 13 uses GRE encapsulation (Profile 0) and AS 24 uses MPLS encapsulation via mLDP (Profile 1). First, I verify that the default MDT forms correctly inside AS 13. Although not recommended, I used ASM groups to demonstrate that IPv4 MDT/MVPN is not a requirement for GRE-based MVPN. For brevity, I limit the verifications to the OSPF VPN. Since the MVPN features used are deployed per-AS, not per-VPN, it is acceptable to only verify one VRF per AS for illustration. CSR8, CSR5, and XRv1 are all inside of the same default MDT. So far, this is identical to MVPN for option A. I highlighted the inter-AS PIM neighbors on CSR5 in green as well; this verification gives us the bonus information of ensuring those PIM neighbors are functional.

```

R8#show ip pim vrf OSPF neighbor | begin ^Neigh
Neighbor          Interface          Uptime/Expires    Ver    DR
Address
10.4.8.4           GigabitEthernet2.548    00:31:13/00:01:31 v2      1 / S P G
13.0.0.5           Tunnel3                 00:29:28/00:01:17 v2      1 / S P G
13.0.0.11          Tunnel3                 00:29:57/00:01:15 v2      1 / DR P G

```

```

R5#show ip pim vrf OSPF neighbor | begin ^Neigh
Neighbor          Interface          Uptime/Expires    Ver    DR
Address
10.5.6.6           GigabitEthernet2.5562   00:31:43/00:01:33 v2      1 / S P G
10.5.7.7           GigabitEthernet2.5572   00:31:52/00:01:22 v2      1 / DR S P G
13.0.0.11          Tunnel4               00:30:32/00:01:41 v2      1 / DR P G
13.0.0.8           Tunnel4               00:30:58/00:01:16 v2      1 / S P G

```

```

RP/0/0/CPU0:XRv1#show ip pim vrf OSPF neighbor | begin ^Neigh
Neighbor Address  Interface          Uptime    Expires    DR pri    Flags
10.6.11.6         GigabitEthernet0/0/0/0.5612 00:31:29  00:01:16  1        P
10.6.11.11*       GigabitEthernet0/0/0/0.5612 3d00h     00:01:25  1 (DR)    B P E
13.0.0.5          mdtOSPF            00:29:48  00:01:26  1        P
13.0.0.8          mdtOSPF            00:30:43  00:01:31  1        P
13.0.0.11*       mdtOSPF            3d00h     00:01:25  1 (DR)    P

```

Inside of AS 24, we quickly trace the MP2MP tree from root to leaves. Although mLDP trees are built upstream towards the root, I verify it in the reverse direction for brevity. Without checking the MDT database in brief format, we know the opaque is MDT and the VPN ID is 24:2 as this was identified in the VRF configuration earlier. The root is XRv4 and it has no upstream neighbors as expected. All upstream labels are local labels since XRv4 allocated these to inform downstream nodes how to forward traffic up the tree (towards the root). The remote labels are downstream labels, learned by XRv4 via each peer's label-mapping messages, which describe how to forward traffic down the tree (away from the root). The tree has 3 downstream peers: CSR6 and CSR7 are ASBRs and CSR2 is the PE. All of them appear the same to XRv4 as it is a P-router from the perspective of the data-plane.

```
RP/0/0/CPU0:XRv4#show mpls mldp database opaquetype mdt 24:2 0
mLDP database
LSM-ID: 0x00009   Type: MP2MP   Uptime: 2d18h
  FEC Root          : 24.0.0.14 (we are the root)
  Opaque decoded    : [mdt 24:2 0]
  Upstream neighbor(s) :
    None
  Downstream client(s):
    LDP 24.0.0.2:0   Uptime: 00:37:16
      Next Hop       : 24.2.14.2
      Interface      : GigabitEthernet0/0/0/0.524
      Remote label (D) : 2023           Local label (U) : 94023
    LDP 24.0.0.6:0   Uptime: 00:17:14
      Next Hop       : 24.6.14.6
      Interface      : GigabitEthernet0/0/0/0.564
      Remote label (D) : 6052           Local label (U) : 94028
    LDP 24.0.0.7:0   Uptime: 00:17:14
      Next Hop       : 24.7.14.7
      Interface      : GigabitEthernet0/0/0/0.574
      Remote label (D) : 7031           Local label (U) : 94031
```

A quick OAM check from CSR2 indicates that the tree has formed and is functioning properly.

```
R2#ping mpls mldp mp2mp 24.0.0.14 mdt 24:2 0
mp2mp Root node addr 24.0.0.14
Opaque type MDT, oui:index 0x24:02, mdtnum 0
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
  timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]
Request #1
! reply addr 24.6.14.6
! reply addr 24.7.14.7
```

Last, I ensure all ASBR/PE routers have VRF-aware PIM neighbors with all others (yellow). This ensures that the MP2MP tree is functioning correctly. On CSR6 and CSR7, we can also verify the inter-AS PIM neighbors (green).

```
R2#show ip pim vrf OSPF neighbor | begin ^Neigh
Neighbor      Interface      Uptime/Expires      Ver      DR
Address
10.2.9.9      GigabitEthernet2.529      00:46:21/00:01:40 v2      1 / DR S P G
24.0.0.6      Lspvif1                    00:44:18/00:01:43 v2      1 / S P G
24.0.0.7      Lspvif1                    00:45:17/00:01:41 v2      1 / DR S P G
```

```
R6#show ip pim vrf OSPF neighbor | begin ^Neigh
Neighbor      Interface      Uptime/Expires      Ver      DR
Address
10.5.6.52     GigabitEthernet2.5562     00:46:47/00:01:44 v2      1 / DR S P G
10.6.11.11    GigabitEthernet2.5612     00:46:46/00:01:40 v2      1 / DR P G
24.0.0.2      Lspvif1          00:44:52/00:01:43 v2      1 / S P G
24.0.0.7      Lspvif1          00:44:52/00:01:38 v2      1 / DR S P G
```

```
R7#show ip pim vrf OSPF neighbor | begin ^Neigh
Neighbor      Interface      Uptime/Expires      Ver      DR
Address
10.5.7.5      GigabitEthernet2.5572     00:46:59/00:01:34 v2      1 / S P G
24.0.0.6      Lspvif1          00:44:55/00:01:36 v2      1 / S P G
24.0.0.2      Lspvif1          00:45:54/00:01:39 v2      1 / S P G
```

Though we did not verify EIGRP, we expect that all of the PIM neighbors are in place and are functional. A quick check would be to ensure the RP information (10.3.3.3) is present on all PEs and ASBRs. We can see that all routers learn this information except CSR2.

```
RP/0/0/CPU0:XRv2#show pim vrf EIGRP rp mapping
PIM Group-to-RP Mappings
Group(s) 224.0.0.0/4
  RP 10.3.3.3 (?), v2
    Info source: 10.3.12.3 (?), elected via bsr, priority 0, holdtime 150
    Uptime: 23:21:09, expires: 00:01:40
```

```
RP/0/0/CPU0:XRv1#show pim vrf EIGRP rp mapping
PIM Group-to-RP Mappings
Group(s) 224.0.0.0/4
  RP 10.3.3.3 (?), v2
    Info source: 13.0.0.12 (?), elected via bsr, priority 0, holdtime 150
    Uptime: 23:16:21, expires: 00:02:24
```

```
R5#show ip pim vrf EIGRP rp mapping
PIM Group-to-RP Mappings
Group(s) 224.0.0.0/4
  RP 10.3.3.3 (?), v2
```

```
Info source: 10.3.3.3 (?), via bootstrap, priority 0, holdtime 150
Uptime: 00:58:02, expires: 00:02:10
```

```
R6#show ip pim vrf EIGRP rp mapping
```

```
PIM Group-to-RP Mappings
```

```
Group(s) 224.0.0.0/4
```

```
RP 10.3.3.3 (?), v2
```

```
Info source: 10.3.3.3 (?), via bootstrap, priority 0, holdtime 150
Uptime: 00:57:12, expires: 00:01:59
```

```
R7#show ip pim vrf EIGRP rp mapping
```

```
PIM Group-to-RP Mappings
```

```
Group(s) 224.0.0.0/4
```

```
RP 10.3.3.3 (?), v2
```

```
Info source: 10.3.3.3 (?), via bootstrap, priority 0, holdtime 150
Uptime: 00:57:27, expires: 00:01:43
```

```
R2#show ip pim vrf EIGRP rp mapping
```

```
PIM Group-to-RP Mappings
```

```
[no output]
```

Enabling VRF-aware PIM BSR debugging on CSR2, the issue is revealed. CSR2 receives BSR messages from both CSR6 and CSR7 over the PMSI as expected, but RPF fails. CSR2 expects to receive them from 13.0.0.12, which is in another AS. The first statement of the error message can be ignored since the BSR message did arrive on the correct interface, but from the incorrect neighbor address.

```
R2#debug ip pim vrf EIGRP bsr
```

```
PIM-BSR debugging is on
```

```
PIM(1): Received v2 Bootstrap on Lspvif0 from 24.0.0.6
```

```
PIM-BSR(1): bootstrap (10.3.3.3) on non-RPF path Lspvif0 (expected Lspvif0)
or from non-RPF neighbor 24.0.0.6 (expected 13.0.0.12) discarded
```

```
PIM(1): Received v2 Bootstrap on Lspvif0 from 24.0.0.7
```

```
PIM-BSR(1): bootstrap (10.3.3.3) on non-RPF path Lspvif0 (expected Lspvif0)
or from non-RPF neighbor 24.0.0.7 (expected 13.0.0.12) discarded
```

At a glance, this seems odd because, in option A, this inter-AS information should not be exchanged. Option AB bends those rules since the control-plane information is shared, to some degree, between ASes. This initially included VPNv4 routes, their RDs, and their extended communities (RT, etc). However, we can confirm that 13.0.0.12 is, in fact, the RPF neighbor for 10.3.3.3/32 inside the EIGRP VPN. The key to this output is the “BGP originator”. This attribute was discussed in the option B section and is used for RPF checks against multicast packets coming from the remote AS. In this case, BSR messages are rejected since multicast traffic is arriving from the ASBRs as opposed to the remote PE. The unicast FIB clearly shows 24.0.0.6 as the preferred unicast path, which is the VPN next-hop for 10.3.3.3/32.

```

R2#show ip rpf vrf EIGRP 10.3.3.3
RPF information for ? (10.3.3.3)
  RPF interface: Lspvif0
  RPF neighbor: ? (13.0.0.12)
  RPF route/mask: 10.3.3.3/32
  RPF type: unicast (bgp 24)
  Doing distance-preferred lookups across tables
  BGP originator: 13.0.0.12
  RPF topology: ipv4 multicast base, originated from ipv4 unicast base

```

```

R2#show ip cef vrf EIGRP 10.3.3.3/32 detail
10.3.3.3/32, epoch 0, flags [rib defined all labels]
  recursive via 24.0.0.6 label 6046
    nexthop 24.2.14.14 GigabitEthernet2.524 label 94008

```

This attribute cannot be overridden or modified in any obvious way. The designers of this feature likely did not consider option AB and MVPN being used together as the RPF lookup should be making decisions similar to option A. Even the multicast route, replicated from the unicast iBGP route, correctly says 24.0.0.6 is the next-hop. Only the BGP connector attribute, when present, is processed.

```

R2#sh ip route multicast vrf EIGRP 10.3.3.3
Routing Table: EIGRP:multicast
Routing entry for 10.3.3.3/32
  Known via "bgp 24", distance 200, metric 0
  Tag 13, type internal,
  replicated from topology(EIGRP)
  Last update from 24.0.0.6 01:23:23 ago
  Routing Descriptor Blocks:
  * 24.0.0.6 (default), from 24.0.0.6, 01:23:23 ago
    Route metric is 0, traffic share count is 1
    AS Hops 1
    Route tag 13
    MPLS label: 6046
    MPLS Flags: MPLS Required

```

```

R2#show bgp vpnv4 unicast vrf EIGRP 10.3.3.3/32 bestpath
BGP routing table entry for 24:3:10.3.3.3/32, version 51
Paths: (2 available, best #2, table EIGRP)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  13, (Received from a RR-client)
    24.0.0.6 (metric 20) (via default) from 24.0.0.6 (24.0.0.6)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:24:3 0x8800:32768:0 0x8801:3:288
        0x8802:65281:2560 0x8803:1:1500 0x8806:0:167971843
      Connector Attribute: count=1
      type 1 len 12 value 13:3:13.0.0.12

```

```
mpls labels in/out nolabel/6046
rx pathid: 0, tx pathid: 0x0
```

An odd behavior is observed when we test this on the opposite side of the network. CSR2 saw a BGP connector attribute from AS 13 claiming 13.0.0.12 was the originator of 10.3.3.3/32, which is true. CSR8's view of 10.1.1.1/32, which originates at 24.0.0.2, shows the local ASBR CSR5 as the originator. For this reason, MVPN should work fine from the perspective of AS 13.

```
R8#show bgp vpnv4 unicast vrf OSPF 10.9.9.9
BGP routing table entry for 13:2:10.9.9.9/32, version 76
Paths: (1 available, best #1, table OSPF)
  Not advertised to any peer
  Refresh Epoch 1
  24
    13.0.0.5 (metric 2) (via default) from 13.0.0.12 (13.0.0.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:13:2 OSPF ROUTER ID:10.2.9.2:0
      OSPF RT:0.0.0.0:2:0
      Originator: 13.0.0.5, Cluster list: 13.0.0.12
      Connector Attribute: count=1
      type 1 len 12 value 13:2:13.0.0.5
    mpls labels in/out nolabel/5010
    rx pathid: 0, tx pathid: 0x0
```

On XRv2, we can see that both XRv1 and CSR5 adjusted the connector attributes to be their own addresses. This is the result of the VPN routes being imported locally into VRFs, then re-advertised towards the RRs.

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast rd 13:2 10.9.9.9 | begin 24,
  24, (Received from a RR-client)
    13.0.0.5 (metric 3) from 13.0.0.5 (13.0.0.5)
      Received Label 5010
      Origin incomplete, metric 0, localpref 100, valid, internal, best,
      group-best, import-candidate, not-in-vrf, import suspect
      Received Path ID 0, Local Path ID 1, version 113
      Extended community: OSPF router-id:10.2.9.2 OSPF route-type:0:2:0x0
RT:13:2
  Connector: type: 1, Value:13:2:13.0.0.5
  Path #2: Received by speaker 0
  Not advertised to any peer
  24, (Received from a RR-client)
    13.0.0.11 (metric 3) from 13.0.0.11 (13.0.0.11)
      Received Label 91008
      Origin incomplete, localpref 100, valid, internal, import-candidate,
      not-in-vrf, import suspect
      Received Path ID 0, Local Path ID 0, version 0
```



```
Extended community: OSPF router-id:10.2.9.2 OSPF route-type:0:2:0x0
RT:13:2
Connector: type: 1, Value:13:2:13.0.0.11
```

There appears to be some inconsistency between when a routers applies the connector attribute or not. For example, CSR7 receives an EIGRP VPN route from XRv4 and CSR2. The one from XRv4 has a connector attribute but the one from CSR2 does not. The PE configurations are nearly identical yet the behavior appears to differ between XE and XR.

```
R7#show bgp vpnv4 unicast vrf EIGRP 10.1.1.1
BGP routing table entry for 24:3:10.1.1.1/32, version 352
Paths: (1 available, best #1, table EIGRP)
  Advertised to update-groups:
    6
  Refresh Epoch 1
  Local
    24.0.0.2 (metric 20) (via default) from 24.0.0.2 (24.0.0.2)
      Origin incomplete, metric 10880, localpref 100, valid, internal, best
      Extended Community: RT:24:3 Cost:pre-bestpath:128:10880 0x8800:32768:0
        0x8801:3:288 0x8802:65281:2560 0x8803:65281:1500 0x8806:0:167837953
      mpls labels in/out IPv4 VRF Aggr:7006/2016
      rx pathid: 0, tx pathid: 0x0
```

```
R7#show bgp vpnv4 unicast vrf EIGRP 10.13.13.13
BGP routing table entry for 24:3:10.13.13.13/32, version 416
Paths: (1 available, best #1, table EIGRP)
  Advertised to update-groups:
    6
  Refresh Epoch 1
  Local
    24.0.0.14 (metric 10) (via default) from 24.0.0.2 (24.0.0.2)
      Origin incomplete, metric 10752, localpref 100, valid, internal, best
      Extended Community: RT:24:3 Cost:pre-bestpath:128:10752 0x8800:32768:0
        0x8801:3:282 0x8802:65281:2560 0x8803:1:1500 0x8806:0:168627469
      Originator: 24.0.0.14, Cluster list: 24.0.0.2
      Connector Attribute: count=1
        type 1 len 12 value 24:3:24.0.0.14
      mpls labels in/out IPv4 VRF Aggr:7006/94006
      rx pathid: 0, tx pathid: 0x0
```

Checking CSR5, the VPN routes from CSR8 have the connector attribute set as well, which disproves the claim above.

```
R5#show bgp vpnv4 unicast vrf OSPF 10.4.4.4
BGP routing table entry for 13:2:10.4.4.4/32, version 41
Paths: (1 available, best #1, table OSPF)
  Advertised to update-groups:
    1
```

```

Refresh Epoch 1
Local
 13.0.0.8 (metric 2) (via default) from 13.0.0.12 (13.0.0.12)
  Origin incomplete, metric 1, localpref 100, valid, internal, best
  Extended Community: RT:13:2 OSPF ROUTER ID:10.4.8.8:0
  OSPF RT:0.0.0.0:2:0
  Originator: 13.0.0.8, Cluster list: 13.0.0.12
  Connector Attribute: count=1
  type 1 len 12 value 13:2:13.0.0.8
  mpls labels in/out 5021/8007
  rx pathid: 0, tx pathid: 0x0

```

I suspect these inconsistencies are related to the MVPN methods used. AS 13 uses the GRE method which appears to always allocated the BGP connector attribute, whereas the mLDP methods appear less consistent. Since MVPN ASM clearly cannot work as the RP information cannot transit the VPN, we will quickly test SSM. We expect to have a similar problem as the VPN route 10.4.4.4/32 carries a connector attribute specifying CSR8 as the originator. The RPF check expects traffic to come from CSR8, not the local ingress ASBR CSR6.

```

R2#show bgp vpnv4 unicast vrf OSPF 10.4.4.4/32 bestpath
BGP routing table entry for 24:2:10.4.4.4/32, version 35
Paths: (2 available, best #2, table OSPF)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  13, (Received from a RR-client)
  24.0.0.6 (metric 20) (via default) from 24.0.0.6 (24.0.0.6)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  Extended Community: RT:24:2 OSPF ROUTER ID:10.4.8.8:0
  OSPF RT:0.0.0.0:2:0
  Connector Attribute: count=1
  type 1 len 12 value 13:2:13.0.0.8
  mpls labels in/out nlabel/6036
  rx pathid: 0, tx pathid: 0x0

```

```

R2#show ip rpf vrf OSPF 10.4.4.4
RPF information for ? (10.4.4.4)
  RPF interface: Lspvif1
  RPF neighbor: ? (13.0.0.8)
  RPF route/mask: 10.4.4.4/32
  RPF type: unicast (bgp 24)
  Doing distance-preferred lookups across tables
  BGP originator: 13.0.0.8
  RPF topology: ipv4 multicast base, originated from ipv4 unicast base

```

We can demonstrate the problem by configuring an IGMP SSM join on CSR9 to receive traffic from CSR4. CSR2 will receive the C(S,G) join from CSR9 for (232.9.9.9, 10.4.4.4). CSR2's RPF neighbor for this C-

source is 13.0.0.8 which is accessible via the PMSI. CSR2 attempts to send the join, but cannot. The C(S,G) state looks correct on CSR2, but no other OSPF VPN routers will have this state entry. So long as the BGP connector attributes are used in an option AB network (they are supposed to be optional attributes, but modification appears impossible), MVPN will be broken at points. The attribute origination inconsistencies make it increasingly challenging to test this feature.

```
! CSR9
interface Loopback0
 ip pim sparse-mode
 ip igmp join-group 232.9.9.9 source 10.4.4.4

R2#debug ip pim vrf OSPF
PIM debugging is on

PIM(2): Received v2 Join/Prune on GigabitEthernet2.529 from 10.2.9.9, to us
PIM(2): Join-list: (10.4.4.4/32, 232.9.9.9), S-bit set
PIM(2): Add GigabitEthernet2.529/10.2.9.9 to (10.4.4.4, 232.9.9.9), Forward
state, by PIM SG Join

R2#show ip mroute vrf OSPF 232.9.9.9 | begin \
(10.4.4.4, 232.9.9.9), 00:03:34/00:02:50, flags: sT
Incoming interface: Lspvif1, RPF nbr 13.0.0.8
Outgoing interface list:
GigabitEthernet2.529, Forward/Sparse, 00:03:34/00:02:50

R5#show ip mroute vrf OSPF 232.9.9.9
Group 232.9.9.9 not found

R6#show ip mroute vrf OSPF 232.9.9.9
Group 232.9.9.9 not found

R8#show ip mroute vrf OSPF 232.9.9.9
Group 232.9.9.9 not found
```

8.4.4.4 MPLS TE

MPLS TE works identically for option AB and it does for option A. The tunnels configuration in option A are quickly examined again for completeness. Re-using tunnel100 on XRv2 and CSR2, defined in the option A section, I quickly trace the LSPs to prove the correct operation. From XRv2, we can see the TE tunnel to CSR5 is operational. Tracing the LSP for the EIGRP VPN route 10.1.1.1/32, we see that CSR5 is the best-path due to having the lower BGP RID. The VPN label bound is 5016.

```
RP/0/0/CPU0:XRv2#show mpls traffic-eng tunnels 100 brief
          TUNNEL NAME          DESTINATION      STATUS  STATE
          tunnel-te100          13.0.0.5         up      up
Displayed 1 (of 1) heads, 0 (of 0) midpoints, 0 (of 0) tails
Displayed 1 up, 0 down, 0 recovering, 0 recovered heads
```

```

RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf EIGRP 10.1.1.1 | begin 24,
24, (Received from a RR-client)
13.0.0.5 (metric 3) from 13.0.0.5 (13.0.0.5)
Received Label 5016
Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, imported, import suspect
Received Path ID 0, Local Path ID 1, version 119
Extended community: EIGRP route-info:0x8000:0 EIGRP AD:3:288 EIGRP
RHB:255:1:2560 EIGRP LM:0xff:1:1500 EIGRP VRR:0x0:1.1.1.10 RT:13:3
Connector: type: 1, Value:13:3:13.0.0.5
Source VRF: EIGRP, Source Route Distinguisher: 13:3
Path #2: Received by speaker 0
Not advertised to any peer
24, (Received from a RR-client)
13.0.0.11 (metric 3) from 13.0.0.11 (13.0.0.11)
Received Label 91003
Origin incomplete, localpref 100, valid, internal, import-candidate,
imported, import suspect
Received Path ID 0, Local Path ID 0, version 0
Extended community: EIGRP route-info:0x8000:0 EIGRP AD:3:288 EIGRP
RHB:255:1:2560 EIGRP LM:0xff:1:1500 EIGRP VRR:0x0:1.1.1.10 RT:13:3
Connector: type: 1, Value:13:3:13.0.0.11
Source VRF: EIGRP, Source Route Distinguisher: 13:3

```

The VPN next-hop is reachable via IGP through a TE tunnel, which implies the RSVP-TE label is used.

```

RP/0/0/CPU0:XRv2#show route 13.0.0.5
Routing entry for 13.0.0.5/32
Known via "ospf 13", distance 110, metric 3, type intra area
Routing Descriptor Blocks
13.0.0.5, from 13.0.0.5, via tunnel-te100
Route metric is 3
No advertising protos.

```

The TE tunnel routes via XRv1, has a bandwidth allocation of 5 Mbps, and uses label 91010. The label stack becomes {91010 5016}.

```

RP/0/0/CPU0:XRv2#show mpls traffic-eng tunnels 100 detail | begin Path Info
Path Info:
Outgoing:
Explicit Route:
Strict, 13.11.12.11
Strict, 13.5.11.5
Strict, 13.0.0.5

Record Route: Disabled
Tspec: avg rate=5000 kbits, burst=1000 bytes, peak rate=5000 kbits
[snip]

```

```
RP/0/0/CPU0:XRv2#show mpls traffic-eng tunnels 100 detail | include Label
Outgoing Interface: GigabitEthernet0/0/0/0.521, Outgoing Label: 91010
```

XRv1 is the penultimate hop, so it pops the TE label and exposes the VPN label to CSR5. Since the EIGRP VPN is not enabled for option AB CSC, the inter-AS traffic is raw IP. The entire label stack is removed on CSR5 with the IP packets sent to 10.5.6.6 inside VRF EIGRP.

```
RP/0/0/CPU0:XRv1#show mpls forwarding labels 91010
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
91010	Pop	100	Gi0/0/0/0.551	13.5.11.5	12395

```
R5#show mpls forwarding-table labels 5016 detail
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Outgoing interface	Next Hop
5016	No Label	10.1.1.1/32 [V]	0	Gi2.5563	10.5.6.6

MAC/Encaps=22/22, MRU=1504, Label Stack{}

005056A9DE0D005056A9DC6381000DE4810000030800

VPN route: EIGRP

No output feature configured

The TE tunnel in AS 24 is also operational. CSR6's best path to 10.1.1.1/32 in the EIGRP VPN is via CSR2. The outgoing label is 2016 and the next-hop is 24.0.0.2.

```
R6#show mpls traffic-eng tunnels tunnel 100 brief | begin TUNNEL
```

TUNNEL NAME	DESTINATION	UP IF	DOWN IF	STATE/PROT
R6_t100	24.0.0.2	-	Gi2.567	up/up

```
R6#show bgp vpnv4 unicast vrf EIGRP 10.1.1.1/32 bestpath
```

BGP routing table entry for 24:3:10.1.1.1/32, version 18

Paths: (1 available, best #1, table EIGRP)

Advertised to update-groups:

4 1

Refresh Epoch 1

Local

24.0.0.2 (metric 20) (via default) from 24.0.0.2 (24.0.0.2)

Origin incomplete, metric 10880, localpref 100, valid, internal, best

Extended Community: RT:24:3 Cost:pre-bestpath:128:10880 0x8800:32768:0

0x8801:3:288 0x8802:65281:2560 0x8803:65281:1500 0x8806:0:167837953

mpls labels in/out IPv4 VRF Aggr:6050/2016

rx pathid: 0, tx pathid: 0x0

The VPN next-hop is reachable via IGP, again through a TE tunnel. The RSVP-TE label is 7012 with CSR7 as the next-hop. The label stack becomes {7012 2016} at imposition since CSR6 is effectively an ingress LER.

```

R6#show ip route 24.0.0.2
Routing entry for 24.0.0.2/32
  Known via "isis", distance 115, metric 20, type level-2
  Redistributing via isis 24
  Last update from 24.0.0.2 on Tunnel100, 00:09:31 ago
  Routing Descriptor Blocks:
  * 24.0.0.2, from 24.0.0.2, 00:09:31 ago, via Tunnel100
    Route metric is 20, traffic share count is 1

R6#show ip rsvp reservation detail filter session-type 7 destination 24.0.0.2
Reservation:
  Tun Dest: 24.0.0.2 Tun ID: 100 Ext Tun ID: 24.0.0.6
  Tun Sender: 24.0.0.6 LSP ID: 1
  Next Hop: 24.6.7.7 on GigabitEthernet2.567
  Label: 7012 (outgoing)
  Reservation Style is Shared-Explicit, QoS Service is Controlled-Load
  Resv ID handle: 01000409.
  Average Bitrate is 0 bits/sec, Maximum Burst is 1K bytes
  Min Policed Unit: 0 bytes, Max Pkt Size: 1500 bytes
  Status:
  Policy: Accepted. Policy source(s): MPLS/TE

```

CSR7 performs a label pop to expose the VPN label to CSR2. CSR2 removes all labels and forwards the IP packets to CSR1 inside of the EIGRP VPN.

```

R7#show mpls forwarding-table labels 7012
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
7012      Pop Label  24.0.0.6 100 [1] 18222      Gi2.527    24.2.7.2

R2#show mpls forwarding-table labels 2016 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
2016      No Label  10.1.1.1/32 [V]  0           Gi2.512    10.1.2.1
          MAC/Encaps=18/18, MRU=1504, Label Stack{}
          005056A91AAA005056A9BE8A81000DB80800
          VPN route: EIGRP
          No output feature configured

```

We quickly verify the LSP using traceroute inside of the VPN. Everything works identically as it did for option A, since the intra-AS TE tunnels just change the routing to the egress ASBRs (also fulfilling the role of LERs when CSC is not in use). The transit link remains raw IP.

```

R3#traceroute 10.1.1.1 source 10.3.3.3
Type escape sequence to abort.
Tracing the route to 10.1.1.1
VRF info: (vrf in name/id, vrf out name/id)

```

```

1 10.3.12.12 8 msec 2 msec 2 msec
2 13.11.12.11 [MPLS: Labels 91010/5016 Exp 0] 7 msec 7 msec 6 msec
3 10.5.6.53 [MPLS: Label 5016 Exp 0] 16 msec 16 msec 15 msec
4 10.5.6.6 20 msec 9 msec 12 msec
5 24.6.7.7 [MPLS: Labels 7012/2016 Exp 0] 12 msec 17 msec 21 msec
6 10.1.2.2 [MPLS: Label 2016 Exp 0] 20 msec 18 msec 20 msec
7 10.1.2.1 21 msec 12 msec 12 msec

```

8.4.5 Confederation variation

Inter-AS option AB confederation is not currently supported in XE. Once the confederation configurations are applied, the “inter-as-hybrid” command is automatically removed. Adding it back generates the error message shown below. This chapter is a placeholder in the event this feature is supported in the future.

```

R6(config-router-af)#neighbor 10.5.6.5 inter-as-hybrid
%BGP: this command is only valid for eBGP peers

```

9. Describe multicast P2MP TE

Because this fits within the scope of MVPN profiles, it is documented there in the appropriate chapters. Specifically, it is documented under profiles 8 and 10, and both tests include configurations files.

10. Describe EVPN (EVPN and PBB-EVPN)

Ethernet VPN was developed as a next-generation L2VPN technology to overcome existing VPLS limitations. Since VPLS acts like an Ethernet switch by learning MAC addresses (data-plane learning), its scalability can be limited by the number of MAC addresses in the customers network. For this reason, many VPLS deployments have a small number of MAC addresses behind each site, which is not appropriate for data-center interconnect (DCI) service. EVPN also introduces per-flow load balancing, which is somewhat available in VPLS given the flow-aware transport PW (FAT-PW) label, but EVPN improves upon it. EVPN also supports multi-homing at layer 2 to prevent loops and duplicate packets; its lack of fully/partially meshed PWs means that it has the same scalability as MPLS L3VPN. It uses a new BGP AFI to carry new EVPN routes that essentially enables “MAC routing”. Multicast inclusion can also be signaled to indicate interest in multicast reception. Both EVPN and PBB-EVPN are discussed in this section with a limited demonstration. XRv does not support the L2VPN data-plane, and XE platforms (ASR1000, CSR1000v, etc) only support EVPN BGP features and none of the data-plane features at all. In this way, both XRv and CSR1000v make good EVPN route-reflectors. For data-plane support, ASR9000 series are the best choice. First, we must understand several key definitions.

BUM: Broadcast, multicast, or unknown unicast. Though not specific to EVPN, this is used extensively in the descriptions and is defined here for clarity. BUM traffic generally requires special handling by EVPN and is often treated like a multicast Ethernet frame.

EVI: Ethernet VPN Identifier. This is analogous to the virtual circuit ID (VCID) seen in VPLS which identifies an EVPN to BGP. An EVI may support multiple bridge-domains. Classic design options, such as

E-LINE/E-LAN/E-TREE (port-based transparent service) and EV-LINE/EV-LAN/EV-TREE (VLAN-based service) are still available, but we can also map specific C-VLANs to EVIs flexibly.

ES: Ethernet segment. This is analogous to an access circuit (AC) in VPLS which defines a link to the customer. An ES differs from an AC in that an ES actually defines a “site” which could be connected to multiple PEs. As such, multiple PEs could share the same ES if the customer is multi-homed at layer 2. Thus, the ES could be a single CE device or an entire network if the customer has multiple CE devices. A customer can follow one of these four models:

1. **Single-homed device (SHD).** This is a single device connected to a single PE which terminates the broadcast domain (like a router).
2. **Multi-homed device (MHD)** using LACP. This is a single device connected to multiple PEs. The CE devices terminate the broadcast domain.
3. **Single-homed network (SHN).** Like SHD, this is a single device connected to a single PE, but with layer 2 network behind it.
4. **Multi-homed network (MHN)** using LACP. This consists of multiple CE devices connected to multiple PE devices with a layer 2 network behind the CEs.

ESI: Ethernet Segment Identifier. Every ES is identified by a unique ESI. It is a 10 byte field that can be derived one of three ways. Of note, on singly-attached networks, the ESI is always 0; there is no compelling reason to track the ESI in these cases.

1. **LACP:** Based on the CEs information, the first 2 bytes will be the CE’s system priority, the next 6 bytes will be the CE’s LACP system-ID (like a MAC address), and the last 2 bytes will be a port key, if defined. This information is carried in the LACPDUs received from the CE.
2. **MST:** If LACP is not in use, MST can be used to create the ESI for an ES. This is based on the CIST’s root parameters: first 2 bytes is the bridge priority, next 6 bytes is the bridge MAC address, and the last 2 bytes are always 0. Essentially, the 8 byte bridge ID is used with trailing zeroes.
3. **Manual configuration:** If LACP and MST are not in use, it can be manually configured. The manual configuration only allows main interfaces to be used, not subinterfaces, and requires input that looks like LACP. One will specify the system priority, system ID, and port key for a given Ethernet interface, which represents the ES from the PE’s perspective. This would need to be the same on all PEs service the same ES.

I-SID: Instance Service identifier. An identifier telling the router where to map a customer frame. It represents a service instance like an EFP. Different I-SIDs can be mapped to the same EVI if their topologies are similar, or to different EVIs if their topologies are different. The I-SID is a 24-bit value that is used by the PBB edge domain to determine how frames should be forwarded.

There are two different redundancy models when multi-homing:

1. **Single-active:** One PE is active and all others are in standby, which means only this active PE is allowed to forward traffic towards the customer. VPLS also supports this option by way of active/backup PWs.

2. **All-active:** All PEs can forward traffic down to the customer concurrently. This does not apply to BUM traffic as it would could cause loops, but is valid for unicast frames. VPLS cannot support this functionality.

There are 4 new types of BGP routes used for the BGP EVPN AFI. Each one is referenced by its “type” number. Both EVPN and PBB-EVPN use these routes, except that type 1 routes only apply to EVPN.

1. **Ethernet active discovery (AD) route:** This route is used to signal MAC withdrawals, aliasing, and split-horizon labels. This applies to EVPN only and is not used for PBB-EVPN. These details are described in the EVPN section.
2. **MAC advertisement route:** This carries MAC addresses along with EVPN ESIs and MPLS labels.
3. **Inclusive multicast route:** This route uses provider multicast service interface (PMSI) attributes to represent ingress replication (IR); MVPN technologies are discussed at length in a dedicated section. Other multicast tunneling options can be configured as well, but IR is the default since it does not assume an LSM architecture is available. This also carries the root of the multicast deliver tree (a subcomponent of PMSI) and an MPLS label.
4. **Ethernet Segment (ES) route:** This route carries information relevant for multi-homed layer 2 connections (redundancy via ICCP) and the designated forwarder (DF) election. In networks without multi-homing, these routes are not shown.

Both EVPN and PBB-EVPN must also prevent loops via split-horizon mechanisms. Since these mechanisms differ between EVPN and PBB-EVPN, split-horizon is discussed in the individual sections.

In multi-homed deployments, the SP network must identifier designated forwarders (DF) who are allowed to send traffic towards the customer. If a BUM frame is flooded across the network between ESes, multiple remote PEs on the other end of the network may both flood the same BUM frame to the customer, which is not ideal. Instead, only one of the PEs is allowed to do this. The DF election is the same for EVPN and PBB-EVPN and operates as follows:

1. Identify each PE by its lowest IP address. The PEs connected to an ES are then assigned an index starting at 0. For example, if PE1 has address 1.11.11.11 and PE2 has address 1.12.12.12, then PE1 is assigned index 0 and PE2 is assigned index 1. This is the PE “ordered list” and has length ‘n’.
2. Evaluate $[EVI \text{ modulo } 'n']$ for each EVI. If EVIs 1000 and 1001 are used, the following occurs:
 - a. $EVI \ 1000 \text{ modulo } 2 = 0$, therefore PE1 is the DF and PE2 is the backup DF (BDF)
 - b. $EVI \ 1001 \text{ modulo } 2 = 1$, therefore PE2 is the DF and PE1 is the BDF

Aliasing is a technique used to ensure that, when operating in all-active mode, that all PEs are aware of all MACs addresses in an ES, even if they have not received traffic from that MAC address. For example, if a CE is multi-homed to PE1 and PE2 using LACP, and a unicast frame is forwarded to PE1, PE2 will not know about it. If PE2 receives a frame from the core destined to this MAC address, it won’t know how to

forward it unless PE1 tells it how. Aliasing is handled differently in EVPN and PBB-EVPN and is discussed there.

When using the active-standby (single-active) model, there needs to be a mechanism to notify the backup PE that it should become the active PE. Using BGP route withdrawal, both EVPN and PBB-EVPN signal this differently.

10.1 EVPN

After searching for some time, I was not able to find a configuration example that shows an EVPN implementation on IOS XR. It is discussed in detail on many presentations but the configuration examples all refer to PBB, probably because PBB-EVPN scales much better and there isn't much benefit to EVPN over PBB-EVPN. This section will briefly describe how EVPN works.

In VPLS, the split-horizon rule states that traffic arriving on a PW cannot be sent out of any other PW. One of the few reasons to relax this rule is to support H-VPLS, but generally it makes sense. It is also why VPLS requires a full mesh of PWs; without it, some kind of spanning-tree protocol would be needed. EVPN may suffer from similar problems in multi-homed layer 2 environments. If two PEs (PE1 and PE2) are connected to the same CE which is running multi-chassis LACP (one link to each PE), then the possibility of a loop exists when the CE sends BUM traffic. If the CE selects PE1 to send a multicast frame, the frame will be replicated to all other remote PEs, including PE2, despite PE2 serving the same ES. The solution is to use a split-horizon label; the label is included with the BUM frame and identifies the source ES of the frame. When PE2 receives it, it drops the traffic as it is already connected to the source ES via LACP.

Aliasing in EVPN, like split-horizon, is achieved using the type 1 route. When a PE learns a C-MAC from the CE, it advertises this C-MAC into BGP along with the ESI from which the MAC was learned. The PE also attached to this ES will evaluate the ESI, see a match, then know that this MAC address is reachable via the connected ES shared with the PE that originated the BGP route.

Backup path signaling is done by withdrawing the type 1 routes associated with a MAC address. Since there is a type 1 BGP route for each C-MAC, each one is withdrawn for a given ESI that signals the backup PE to become active.

10.2 PBB-EVPN

PBB-EVPN extends the scalability of EVPN by combining the 802.1ah MAC-in-MAC (or PBB) logic into EVPN. Each PE has a backbone MAC (B-MAC) which is used to identify that PE. In this way, only one type 2 (MAC advertisement) route is created per PE in each EVI, versus one per C-MAC in regular EVPN. This is why the type 1 route is not needed in PBB since MAC withdrawals and aliasing are not relevant.

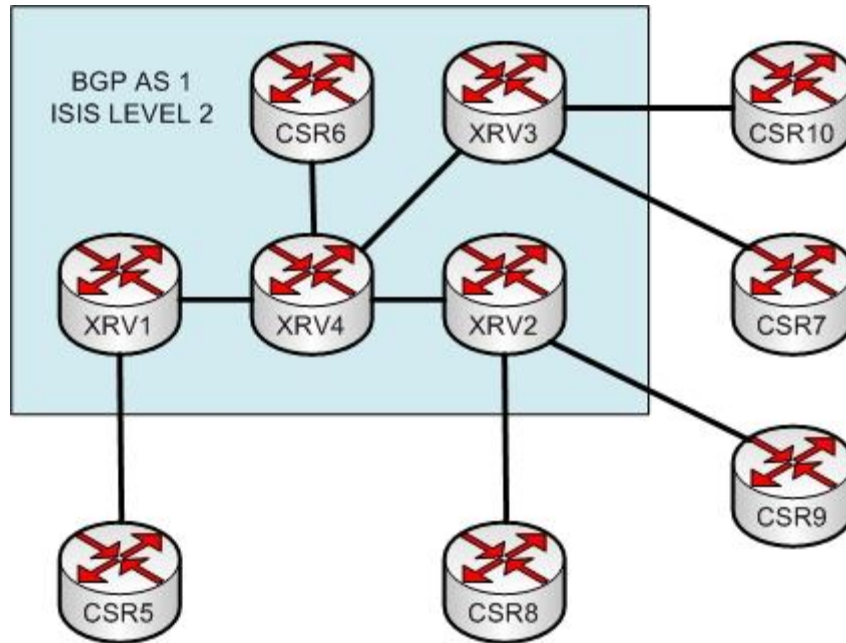
Split-horizon is also possible in PBB-EVPN as in EVPN. However, a split-horizon label is not necessary to prevent loops with PBB-EVPN. PEs connected to the same ES will use the same B-MAC for that ES: that

is, there is a 1:1 mapping between B-MAC and ESI. The receiving PEs simply check the source B-MAC on incoming BUM frames and compare it against their own. If the two match, the packet is dropped.

Aliasing in PBB-EVPN is simpler than in EVPN, much like split-horizon. Since two PEs connected to the same ES will have the same B-MAC, if a PE receives a frame destined for its own B-MAC but doesn't have the explicit C-MAC installed, it can safely assume this C-MAC is reachable via the connected ES. No explicit routing is needed since the traffic would not have been sent to this PE if the remote PE (across the core) wasn't already aware that it was mapped to the proper B-MAC shared by other PEs on the ES. Since type 1 BGP routes are not used for PBB-EVPN, a router simply withdraws its type 2 router for that ESI. Because the B-MAC is bound to that ESI with PBB, other routers will know that when a PE loses connectivity to an ES, that B-MAC cannot be used for forwarding anymore. The other PEs on the ES will have the same B-MAC so there is no need for per-MAC withdrawals as with EVPN.

PBB operations are similar to EVPN with some exceptions. First, the concept of a B-MAC is mapped to an ESI on a 1:1 basis. The B-MAC can be automatically derived from the LACP system ID of the CE by flipping the U/L bit, sort of like EUI-64 for IPv6 SLAAC. In this way, all PEs connected to an ES will have the same B-MAC. The route distinguishers (RDs) are still unique per PE as they are auto-generated based on the BGP router-IDs. An EVPN type 2 route will include several key components, include the RD, ESI, B-MAC, MPLS label, and route-target (RT). The label is a downstream label advertised to other PEs to use when sending traffic towards this B-MAC within a given EVI. When multiple PEs exist on a segment, their B-MACs will be the same, but their labels will not be. Like the RD, the RT is automatically configured based on the BGP ASN and the EVI. This is like VPLS auto-route-target and can be overridden if necessary. This ensures that all endpoints within an EVI are connected in a LAN topology by default (can adjust RTs to build a tree, etc).

Since PBB control-plane is supported on XRv, we will look at some of the PBB details. The network diagram is kept simple with 3 PBB edge routers and some CE routers as well. Both XRv4 and CSR6 are EVPN route-reflectors; this is the extent of IOS XE's support for EVPN at present. The remaining CSRs are CE routers with OSPFv3 enabled for IPv4 and IPv6. If PBB-EVPN ever suddenly starts working on XRv, then they should form neighbors according to the EVPN configuration. CSR5, CSR7, and CSR8 share a LAN, while CSR9 and CSR10 do the same in a different VPN.



The basic configurations are shown below using XRv3 as the demonstration, since XRv1 and XRv2 basic PBB configurations are identical. As discussed earlier, we identify PBB “edge” and “core” bridge-domains. PBB edge domains are assigned I-SIDs which identify ESes/VLANs and map to interfaces. PBB core domains specify the EVI used which is the basis for RT generation. Every PBB edge I-SID must map to a core bridge-domain so the router can build the internal plumbing from the EFP to the MPLS core. Using terminology from the 802.1ah (MAC in MAC) section, the PBB edge domain handles the service component (I-SID) and the PBB core domain handles the backbone component (B-MAC and B-tag). This multi-stage encapsulation allows for maximal flexibility when building EVPNs. XRv3 runs BGP to XRv4 and CSR6 which are EVPN route-reflectors. Of note, XRv3 has a second CE and thus defines a second PBB edge group. It shares the same bridge group as the other PBB edges but uses a separate bridge-domain. This will become relevant when looking at BGP routes.

```

! XRv3 (similar for XRv1 and XRv2)
interface GigabitEthernet0/0/0/0.503 l2transport
 encapsulation dot1q 3503 exact
 rewrite ingress tag pop 1 symmetric

interface GigabitEthernet0/0/0/0.573 l2transport
 encapsulation dot1q 3573 exact
 rewrite ingress tag pop 1 symmetric

l2vpn
pbb
 backbone-source-mac 0000.0013.0013

bridge group PBB_CORE
 bridge-domain PBB_CORE

```

```

    pbb core
    evpn evi 1000

bridge group PBB_EDGES
    bridge-domain PBB_EDGE_1
    pbb edge i-sid 1001 core-bridge PBB_CORE
    interface GigabitEthernet0/0/0/0.573

    bridge-domain PBB_EDGE_2
    pbb edge i-sid 1002 core-bridge PBB_CORE
    interface GigabitEthernet0/0/0/0.503

router bgp 1
    bgp router-id 1.13.13.13
    address-family l2vpn evpn

    neighbor-group IBGP
    remote-as 1
    timers 10 40
    update-source Loopback0
    address-family l2vpn evpn

    neighbor 1.6.6.6
    use neighbor-group IBGP

    neighbor 1.14.14.14
    use neighbor-group IBGP

```

There is nothing exciting about the BGP configurations on XRv4 and CSR6. They are just RRs for the L2VPN EVPN SAFI and don't do anything fancy. This is just demonstrating the RR capability on XE and XR.

```

! XRv4
router bgp 1
    bgp router-id 1.14.14.14
    bgp cluster-id 1.14.14.14
    address-family l2vpn evpn

    neighbor-group IBGP
    remote-as 1
    timers 10 40
    update-source Loopback0
    address-family l2vpn evpn
    route-reflector-client

    neighbor 1.11.11.11
    use neighbor-group IBGP

    neighbor 1.12.12.12
    use neighbor-group IBGP

```

```

neighbor 1.13.13.13
  use neighbor-group IBGP

! CSR6
router bgp 1
  no bgp default ipv4-unicast
  neighbor IBGP peer-group
  neighbor IBGP remote-as 1
  neighbor IBGP update-source Loopback0
  neighbor IBGP timers 10 40
  neighbor 1.11.11.11 peer-group IBGP
  neighbor 1.12.12.12 peer-group IBGP
  neighbor 1.13.13.13 peer-group IBGP

address-family l2vpn evpn
  neighbor IBGP route-reflector-client
  neighbor 1.11.11.11 activate
  neighbor 1.12.12.12 activate
  neighbor 1.13.13.13 activate

```

First and most simply, we can verify the B-MAC configuration worked on one of the routers. In verifying this, we can also see the “chassis” MAC which would have been used had we not configured one.

```

RP/0/0/CPU0:XRv3#show l2vpn pbb backbone-source-mac
Backbone Source MAC: 0000.0013.0013
Chassis MAC          : 0b16.212c.3742

```

We can look at XRv3 to see all routes in the topology. So far, we only see type 2 and type 3 routes. This makes sense because type 1 only applies to EVPN (which does not appear configurable on XR) and type 4 only applies to multi-homed environments (multi-chassis LACP not supported on virtual platforms). Giving this output a closer look, we notice the word “vrf” used to identify PBB_CORE. This is odd because PBB_CORE is a bridge-domain that maps to an EVI. However, the cosmetics of the output treat this like a VRF since it maps to an RD. It would make sense, then, for the administrator to name the PBB core bridge-domains based on the EVIs they support. Next, we can see that each PE generates exactly one type-2 MAC advertisement route (yellow). They carry a 48-bit MAC address which represents the manually-configured B-MAC on each router; I manually configure it for ease of understanding, but doing so is not necessary. We then see several type 3 Inclusive Multicast routes (green). We see 5 of these, so clearly PEs can advertise more than one, unlike type 2 routes. In this case, all PEs are part of I-SID 1001, which is a PBB edge domain, and thus each one advertises a route (cyan). I-SID 1002 is only present on XRv2 and XRv3, so only those two PEs generate a route for this I-SID (pink). Because these are multicast inclusion routes, this ensures that multicast traffic doesn’t leak across PBB edge domains even if they share the same PBB core domain.

```

RP/0/0/CPU0:XRv3#show bgp l2vpn evpn rd 1.13.13.13:1000 | begin Network

```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 1.13.13.13:1000 (default for vrf PBB_CORE)					
*>i [2] [0] [48] [0000.0011.0011] [0]/104	1.11.11.11		100	0	i
*>i [2] [0] [48] [0000.0012.0012] [0]/104	1.12.12.12		100	0	i
*> [2] [0] [48] [0000.0013.0013] [0]/104	0.0.0.0			0	i
*>i [3] [1001] [32] [1.11.11.11]/80	1.11.11.11		100	0	i
*>i [3] [1001] [32] [1.12.12.12]/80	1.12.12.12		100	0	i
*> [3] [1001] [32] [1.13.13.13]/80	0.0.0.0			0	i
*>i [3] [1002] [32] [1.12.12.12]/80	1.12.12.12		100	0	i
*> [3] [1002] [32] [1.13.13.13]/80	0.0.0.0			0	i

We can also view information by specifying the PBB core domain by name (versus using the RD), much like using a VRF name. In this way, we will examine the type 2 route from XRv2. Like MVPN, EVPN BGP show commands require you to enter the entire prefix string on XR. Below, we can see many of the key fields carried inside this route. The MAC address is shown in the prefix, but the MPLS label, RT, and ESI are carried inside of the route itself. Much like VPNv4/v6, the B-MAC address 0000.0012.0012 is bound to local label 92004 on XRv2. The ESI is always 0 unless multi-homing is configured, so we ignore that for now.

```
RP/0/0/CPU0:XRv3#show bgp l2vpn evpn bdomain PBB_CORE
[2][0][48][0000.0012.0012][0]/104
BGP routing table entry for [2][0][48][0000.0012.0012][0]/104, Route
Distinguisher: 1.13.13.13:1000
[snip]
Local
  1.12.12.12 (metric 20) from 1.6.6.6 (1.12.12.12)
    Received Label 92004
    Origin IGP, localpref 100, valid, internal, best, group-best, import-
candidate, imported, rib-install
    Received Path ID 0, Local Path ID 1, version 10
    Extended community: RT:1:1000
    Originator: 1.12.12.12, Cluster list: 1.6.6.6
    EVPN ESI: 0000.0000.0000.0000.0000
    Source VRF: default, Source Route Distinguisher: 1.12.12.12:1000
```

Now that we know the local label on XRv2 for receiving MPLS packets, we can check the LFIB to see what happens when that label is received. The bridge-domain ID is identified as 0 (more on this later) and the router is identified as a PE. The detailed output shows us that PW flow label is automatically

enabled which means load-sharing is inherent with EVPN. This label is popped before sending traffic to the customer's bridge-domain.

```
RP/0/0/CPU0:XRv2#show mpls forwarding labels 92004 detail
Local   Outgoing   Prefix           Outgoing   Next Hop        Bytes
Label   Label       or ID            Interface                Switched
-----
92004   Pop          No ID            BD=0 PE             point2point          0

PW Flow Label: Enabled
MAC/Encaps: 0/0, MTU: 0
Label Stack (Top -> Bottom): { }
Packets Switched: 0
```

The “BD=0” is referring to the PBB_CORE bridge-domain on XRv2. This delivers the packet from the MPLS process to the core domain.

```
RP/0/0/CPU0:XRv2#show l2vpn bridge-domain bd-name PBB_CORE brief
Legend: pp = Partially Programmed.
Bridge Group:Bridge-Domain Name  ID      State           Num ACs/up    Num PWs/up
-----
PBB_CORE:PBB_CORE                0       up              0/0           0/0
```

To see the auto-generated RD/RTs without looking at BGP, we can query the EVPN EVI details for EVI 1000. This output also shows us some interesting label information; we see a unicast and multicast label for the EVI. The unicast label is 92004 which is what we just saw. We have not seen label 92005 yet.

```
RP/0/0/CPU0:XRv2#show evpn evi vpn-id 1000 detail
EVI      Bridge Domain      Type
-----
1000     PBB_CORE                  PBB

Unicast Label   : 92004
Multicast Label : 92005
Flow Label: N
RD Config: none
RD Auto   : (auto) 1.12.12.12:1000
RT Auto   : 1:1000
Route Targets in Use      Type
-----
1:1000                    Import
1:1000                    Export
```

Checking the LFIB for label 92005, we can see the output is very similar to the unicast label except that it identifies an “inclusive multicast” (IM) entry. The output is otherwise identical, but having a separate label is important to differentiate unicast from BUM traffic.


```
RP/0/0/CPU0:XRv2#show mpls forwarding labels 92005 detail
Local   Outgoing   Prefix           Outgoing   Next Hop        Bytes
Label   Label       or ID           Interface
-----
92005   Pop          No ID           BD=0 PEIM    point2point     0
      PW Flow Label: Enabled
      MAC/Encaps: 0/0, MTU: 0
      Label Stack (Top -> Bottom): { }
      Packets Switched: 0
```

When looking at the BGP type 2 route, we did not see this label being signaled. The label is carried in the type 3 routes and is the same for all I-SIDs. This is the first time we have ever seen a non-zero MPLS label being carried within the PMSI attributes; the MVPN section did not operate this way. Looking at the type-3 routes for I-SIDs 1001 and 1002, we see the same label for both of XRv2's locally-originated BGP routes. Both of these I-SIDs (PBB edge domain) map to the same EVI (PBB core domain) so the RD/RT information is identical as well. Although not terribly significant for EVPN, the tunnel type of 6 represents ingress replication (IR) and the ID is the IP address of the LSM root, which is 1.12.12.12 (XRv2 loopback). I show this in green because it is of less importance right now.

```
RP/0/0/CPU0:XRv3#show bgp l2vpn evpn bdomain PBB_CORE
[3][1001][32][1.12.12.12$
BGP routing table entry for [3][1001][32][1.12.12.12]/80, Route
Distinguisher: 1.13.13.13:1000
[snip]
  Local
    1.12.12.12 (metric 20) from 1.6.6.6 (1.12.12.12)
      Origin IGP, localpref 100, valid, internal, best, group-best, import-
candidate, imported
      Received Path ID 0, Local Path ID 1, version 11
      Extended community: RT:1:1000
      Originator: 1.12.12.12, Cluster list: 1.6.6.6
      PMSI: flags 0x00, type 6, label 92005, ID 0x010c0c0c
      Source VRF: default, Source Route Distinguisher: 1.12.12.12:1000
```

```
RP/0/0/CPU0:XRv3#show bgp l2vpn evpn bdomain PBB_CORE
[3][1002][32][1.12.12.12$
BGP routing table entry for [3][1002][32][1.12.12.12]/80, Route
Distinguisher: 1.13.13.13:1000
[snip]
  Local
    1.12.12.12 (metric 20) from 1.6.6.6 (1.12.12.12)
      Origin IGP, localpref 100, valid, internal, best, group-best, import-
candidate, imported
      Received Path ID 0, Local Path ID 1, version 12
      Extended community: RT:1:1000
      Originator: 1.12.12.12, Cluster list: 1.6.6.6
      PMSI: flags 0x00, type 6, label 92005, ID 0x010c0c0c
      Source VRF: default, Source Route Distinguisher: 1.12.12.12:1000
```

We can see a summary of the type 2 (unicast) labels for EVPN using basic BGP show commands. Since the type 3 multicast labels are carried inside of PMSI attributes, they are not shown in summary form using BGP. Again, we can see label 92004 being used for unicast traffic towards XRv2's B-MAC of 0000.0012.0012.

```
RP/0/0/CPU0:XRv3#show bgp l2vpn evpn bdomain PBB_CORE labels | begin Network
  Network          Next Hop          Rcvd Label          Local Label
Route Distinguisher: 1.13.13.13:1000 (default for vrf PBB_CORE)
*>i [2] [0] [48] [0000.0011.0011] [0]/104
          1.11.11.11          91004          nolabel
*>i [2] [0] [48] [0000.0012.0012] [0]/104
          1.12.12.12          92004          nolabel
*> [2] [0] [48] [0000.0013.0013] [0]/104
          0.0.0.0          nolabel          93004
```

For completeness, we will examine some of the BGP show commands on XE as well. Looking at XRv2's routes, we can see the output is significantly different from XR when looking at the summary information. The type 2 route is expanded more literally on XE; the 20 zeroes in a row is the ESI, where in XR it is displayed as a single zero. The summary output also contains the MPLS label for the type 2 route. XE is clearly struggling to display information in the type 3 routes as it specifies some fields are UNKNOWN.

```
R6#show bgp l2vpn evpn rd 1.12.12.12:1000 | begin Network
  Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 1.12.12.12:1000
*>i
[2] [1.12.12.12:1000] [00000000000000000000] [0] [48] [000000120012] [0] [*] [92004] /
33
          1.12.12.12          100          0 i
*>i [3] [1.12.12.12:1000] [1001] [32] [UNKNOWN] /17
          1.12.12.12          100          0 i
*>i [3] [1.12.12.12:1000] [1002] [32] [UNKNOWN] /17
          1.12.12.12          100          0 i
```

Examining XRv2's type 2 in detail, we only see the RT. Nothing else is revealed since XR crams it into the summary information without saying much about it. Considering the feature isn't supported in XE in the data-plane at all, this is understandable, and XE is just transporting BGP information (no local processing or path-related evaluations on the payload data).

```
R6#show bgp l2vpn evpn rd 1.12.12.12:1000 route-type 2
BGP routing table entry for
[2] [1.12.12.12:1000] [00000000000000000000] [0] [48] [000000120012] [0] [*] [92004] /
33, version 8
Paths: (1 available, best #1, table EVPN-BGP-Table)
  Advertised to update-groups:
```

```

2
Refresh Epoch 1
Local, (Received from a RR-client)
  1.12.12.12 (metric 20) from 1.12.12.12 (1.12.12.12)
    Origin IGP, localpref 100, valid, internal, best
    Extended Community: RT:1:1000
    rx pathid: 0, tx pathid: 0x0

```

Looking at XRv2's type 3 routes provides some extra PMSI information. Since MVPN is fully supported on XE, we can see all the PMSI attributes, to include an option that identifies this PMSI as being "for EVPN". We also see the inclusive multicast MPLS labels, tunnel type (ingress replication), and LSM root (1.12.12.12). There are two routes as expected, one per supported I-SID on XRv2.

```

R6#show bgp l2vpn evpn rd 1.12.12.12:1000 route-type 3
BGP routing table entry for [3][1.12.12.12:1000][1001][32][UNKNOWN]/17,
version 21
Paths: (1 available, best #1, table EVPN-BGP-Table)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  Local, (Received from a RR-client)
    1.12.12.12 (metric 20) from 1.12.12.12 (1.12.12.12)
      Origin IGP, localpref 100, valid, internal, best
      Extended Community: RT:1:1000
      PMSI Attribute: for EVPN, Flags: 0x0, Tunnel type: 6, length 4, label:
92005, tunnel parameters: 010C 0C0C
      rx pathid: 0, tx pathid: 0x0
BGP routing table entry for [3][1.12.12.12:1000][1002][32][UNKNOWN]/17,
version 25
Paths: (1 available, best #1, table EVPN-BGP-Table)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  Local, (Received from a RR-client)
    1.12.12.12 (metric 20) from 1.12.12.12 (1.12.12.12)
      Origin IGP, localpref 100, valid, internal, best
      Extended Community: RT:1:1000
      PMSI Attribute: for EVPN, Flags: 0x0, Tunnel type: 6, length 4, label:
92005, tunnel parameters: 010C 0C0C
      rx pathid: 0, tx pathid: 0x0

```

We will detail inclusive multicast operations a bit more as well. We can clearly see a mapping between I-SIDs, IP addresses, and EVIs using a basic show command on a per EVI basis. In this example, we can see both I-SIDs 1001 and 1002, their participating routers, and the EVI common for all I-SIDs.

```

RP/0/0/CPU0:XRv3#show evpn evi vpn-id 1000 inclusive-multicast
ISID                Originating IP                vpn-id

```

1001	1.11.11.11	1000
1001	1.12.12.12	1000
1001	1.13.13.13	1000
1002	1.12.12.12	1000
1002	1.13.13.13	1000

The detailed output shows the BGP next-hops, along with the MPLS labels and whether the entry was locally generated or remotely learned. Again, this shows the same inclusive multicast MPLS label (as signaled as a subfield in the PMSI attribute) within each I-SID.

```
RP/0/0/CPU0:XRv3#show evpn evi vpn-id 1000 inclusive-multicast detail
ISID: 1001, Originating IP: 1.11.11.11                                1000
    Nexthop: 1.11.11.11
    Label   : 91005
    Source  : Remote
ISID: 1001, Originating IP: 1.12.12.12                                1000
    Nexthop: 1.12.12.12
    Label   : 92005
    Source  : Remote
ISID: 1001, Originating IP: 1.13.13.13                                1000
    Nexthop: ::
    Label   : 93005
    Source  : Local
ISID: 1002, Originating IP: 1.12.12.12                                1000
    Nexthop: 1.12.12.12
    Label   : 92005
    Source  : Remote
ISID: 1002, Originating IP: 1.13.13.13                                1000
    Nexthop: ::
    Label   : 93005
    Source  : Local
```

Ingress replication, as a method to transport LSM, is somewhat self-explanatory. Packets arriving from the CE are replicated to each remote PE that has expressed interest. That is to say, if the remote RT was imported for a route, it is assumed that PE is interested in receiving multicast. IR is easy to deploy because core routers treat them just like ordinary MPLS frames and don't need to run anything special, unlike RSVP P2MP-TE and mLDP, which may not be supported by all routers. Core routers won't even see the inclusive multicast labels since ordinary transport labels, such as LDP, RSVP-TE, or SR will be used to transport traffic across the network.

It is important to note that because the MPLS label is the same for multicast I-SID traffic within an EVI, the PBB encapsulation (802.1ah described later) is responsible for providing the demultiplexing capability to achieve per I-SID granularity, not the MPLS label. We can prove this by creating a second EVC between XRv1 and XRv3 using bogus ESes. For additional variety, we will configure custom RD's and RT's on XRv1 and XRv3 for this EVI. The ES doesn't actually connect a host device but for verifying the

control plane, it doesn't matter. Only XRv3 is shown, but XRv1 has a near-identical configuration. The RTs are reversed so that XRv1 and XRv3 import one another's RTs, and the RDs are different as well.

```
! XRv3
interface GigabitEthernet0/0/0/0.1113 l2transport
 encapsulation dot1q 1113 exact
 rewrite ingress tag pop 1 symmetric

evpn
 evi 1313
  bgp
   rd 0.0.11.13:13
   route-target import 1113:11
   route-target export 1113:13

l2vpn
 bridge group PBB_CORE
  bridge-domain PBB_CORE_3
  pbb core
  evpn evi 1313

bridge group PBB_EDGES
 bridge-domain PBB_EDGE_3
  pbb edge i-sid 3111 core-bridge PBB_CORE_3
  interface GigabitEthernet0/0/0/0.1113
```

Checking BGP on XRv3, we can see XRv1's type 2 and type 3 routes within this new EVI. Since the PBB core domain is different, the EVI is different, which means the RDs, RTs, and MPLS labels will be different. We can see XRv3's custom RD below. Also note that there are only two type 3 IM routes because there is only one I-SID within PBB_CORE_3 (only one PBB edge domain). The I-SID value is 3111 as expected; there is no knowledge of I-SIDs 1001 or 1002 within this PBB core domain, which is a separate EVI than the one previously configured.

```
RP/0/0/CPU0:XRv3#show bgp l2vpn evpn bdomain PBB_CORE_3 | begin Network
      Network                Next Hop                Metric LocPrf Weight Path
Route Distinguisher: 0.0.11.13:13 (default for vrf PBB_CORE_3)
*>i[2][0][48][0000.0011.0011][0]/104
      1.11.11.11                                100          0 i
*> [2][0][48][0000.0013.0013][0]/104
      0.0.0.0                                    0          0 i
*>i[3][3111][32][1.11.11.11]/80
      1.11.11.11                                100          0 i
*> [3][3111][32][1.13.13.13]/80
      0.0.0.0                                    0          0 i
```

Looking at XRv1's type 2 route, we can see the original RD was XRv1's customer RD, and the route also contains the custom RT. A new MPLS label 91006 was allocated for this new PBB core domain.

```
RP/0/0/CPU0:XRv3#show bgp l2vpn evpn bdomain PBB_CORE_3 [2][0][48][0000.0011.0011][0]/104
BGP routing table entry for [2][0][48][0000.0011.0011][0]/104, Route
Distinguisher: 0.0.11.13:13
[snip]
Local
  1.11.11.11 (metric 20) from 1.6.6.6 (1.11.11.11)
    Received Label 91006
    Origin IGP, localpref 100, valid, internal, best, group-best, import-
candidate, imported, rib-install
    Received Path ID 0, Local Path ID 1, version 60
    Extended community: RT:1113:11
    Originator: 1.11.11.11, Cluster list: 1.6.6.6
    EVPN ESI: 0000.0000.0000.0000.0000
    Source VRF: default, Source Route Distinguisher: 0.0.11.13:11
```

Before checking the LFIB on XRv1, we can verify the PBB_CORE_3 bridge-domain ID to correlate it with the LFIB next. This PBB core domain has an ID of 3. For sanity, we verify the local label by checking the EVI details as well, expecting to see value 91006 as the unicast label carried in the BGP type 2 message. The LFIB shows verifies all of this; label 91006 is popped upon reception and the packet is delivered to PBB_CORE_3, which is bridge-domain 3.

```
RP/0/0/CPU0:XRv1#show l2vpn bridge-domain bd-name PBB_CORE_3 brief
Legend: pp = Partially Programmed.
Bridge Group:Bridge-Domain Name  ID      State          Num ACs/up    Num PWs/up
-----
PBB_CORE:PBB_CORE_3             3      up              0/0            0/0
```

```
RP/0/0/CPU0:XRv1#show evpn evi vpn-id 1313 detail | include Unicast
Unicast Label   : 91006
```

```
RP/0/0/CPU0:XRv1#show mpls forwarding labels 91006
Local   Outgoing   Prefix           Outgoing   Next Hop        Bytes
Label   Label      or ID           Interface
-----
91006   Pop          No ID           BD=3 PE      point2point      0
```

The same is true for multicast labels. Since XRv3 and XRv1 are connected with two separate EVIs, we expect the multicast labels assigned for their I-SIDs to vary. In this case, we see label 91008 used for the new I-SID of 3111 (EVI 1313) and 91005 still in use for the old I-SID of 1001 (EVI 1000). Note that this verification was performed remotely on XRv3 based on received BGP type 3 routes.

```
RP/0/0/CPU0:XRv3#show evpn evi inclusive-multicast service-id 3111 orig-ip
1.11.11.11 detail
ISID: 3111, Originating IP: 1.11.11.11                      1313
Nextthop: 1.11.11.11
```

```
Label : 91007
Source : Remote
```

```
RP/0/0/CPU0:XRv3#show evpn evi inclusive-multicast service-id 1001 orig-ip
1.11.11.11 detail
ISID: 1001, Originating IP: 1.11.11.11                      1000
Nexthop: 1.11.11.11
Label : 91005
Source : Remote
```

We can confirm both entries in XRv1's LFIB. We can see that label 91005 is an include multicast label for BD 0, which is PBB_CORE seen earlier. Label 9107 is an IM label for BD 3, which is PBB_CORE_3, verified above. In this way, the MPLS label is mapped to an EVI which allows MPLS to deliver untagged packets to the proper PBB core domain. The PBB encapsulation provides the specific I-SID, not the MPLS label.

```
RP/0/0/CPU0:XRv1#show mpls forwarding labels 91005 91007
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
91005	Pop	No ID	BD=0 PEIM	point2point	0
91006	Pop	No ID	BD=3 PE	point2point	0
91007	Pop	No ID	BD=3 PEIM	point2point	0

We will finish with some additional show commands. Although I have not found a command to show this in BGP, the remote PEs signal whether they are running in single-active or all-active mode. I assume that this command confirms that each remote PE is single-homed as a result of this signaling. Notice that this output does not show any local labels.

```
RP/0/0/CPU0:XRv3#show evpn internal-label detail
```

EVI	Nexthop	Output Label
1000	1.11.11.11	91004
ESI: 0000.0000.0000.0000.0000		
Tag: 0		
Single-homed		
1000	1.12.12.12	92004
ESI: 0000.0000.0000.0000.0000		
Tag: 0		
Single-homed		
1313	1.11.11.11	91006
ESI: 0000.0000.0000.0000.0000		
Tag: 0		
Single-homed		

EVPN information can be summarized as shown below on XRv1. This can be useful for trying to determine what the resulting auto-generate RDs and RTs will be. It also shows the total number of routes seen across all EVIs (PBB core domain) and supported I-SIDs (PBB edge domain).

```
RP/0/0/CPU0:XRv1#show evpn summary
-----
Global Information
-----
Number of EVIs                : 3
Number of Local MAC Routes    : 4
Number of Remote MAC Routes   : 3
Number of Local IMCAST Routes : 2
Number of Remote IMCAST Routes: 5
Number of Internal Labels     : 3
Number of ES Entries          : 1
Number of Neighbor Entries    : 3
BGP Router ID                 : 1.11.11.11
BGP ASN                       : 1
PBB BSA MAC address           : 0000.0011.0011
Global peering timer          : 45 seconds
Global recovery timer         : 20 seconds
Global programming timer      : 1500 microseconds
Global flushagain timer       : 60 seconds
Standard Version              : draft version 06
Version Transition Timer      : 0(min) 0(sec) left
-----
High Availability Information
-----
BGP EOD                       : Y
Number of Marked MAC Routes    : 0
Number of Swept MAC Routes     : 0
Number of Marked IMCAST Routes : 0
Number of Swept IMCAST Routes  : 0
```

Although we will not see any MAC addresses, we can confirm some basic data-plane configurations are correct. First, we look at the bridge-domain summary. We see 3 PBB edge domains and 2 PBB core domains, which is consistent with the BGP verifications we performed earlier. The BGP verifications showed two separate EVIs (1000 and 1313) with three I-SIDs (1001 and 1002 mapped to EVI 1000, and 3111 mapped to EVI 1313).

```
RP/0/0/CPU0:XRv3#show l2vpn bridge-domain summary
Number of groups: 2, bridge-domains: 5, Up: 5, Shutdown: 0, Partially-
programmed: 0
Default: 0, pbb-edge: 3, pbb-core: 2
Number of ACs: 3 Up: 3, Down: 0, Partially-programmed: 0
Number of PWs: 0 Up: 0, Down: 0, Standby: 0, Partially-programmed: 0
Number of P2MP PWs: 0, Up: 0, Down: 0, other-state: 0
Number of VNIs: 0, Up: 0, Down: 0, Unresolved: 0
```


Looking at PBB_EDGE_1 as a bridge-domain, we confirm its configured as PBB-edge and has a valid I-SID. We can see it is mapped to a PBB and has one AC that is online.

```
RP/0/0/CPU0:XRv1#show l2vpn bridge-domain bd-name PBB_EDGE_1
Legend: pp = Partially Programmed.
Bridge group: PBB_EDGES, bridge-domain: PBB_EDGE_1, id: 1, state: up, ShgId:
0, MSTi: 0
Type: pbb-edge, I-SID: 1001
Aging: 300 s, MAC limit: 4000, Action: none, Notification: syslog
Filter MAC addresses: 0
ACs: 1 (1 up), VFIs: 0, PWs: 0 (0 up), PBBs: 1 (1 up)
List of PBBs:
  PBB Edge, state: up, Static MAC addresses: 0
List of ACs:
  Gi0/0/0/0.551, state: up, Static MAC addresses: 0
List of Access PWs:
List of VFIs:
```

The PBB_CORE bridge-domain is also mapped to a PBB (core type) and has one PBB-edge bridge-domain associated with it.

```
RP/0/0/CPU0:XRv1#show l2vpn bridge-domain bd-name PBB_CORE
Legend: pp = Partially Programmed.
Bridge group: PBB_CORE, bridge-domain: PBB_CORE, id: 0, state: up, ShgId: 0,
MSTi: 0
Type: pbb-core
Number of associated pbb-edge BDs: 1
Aging: 300 s, MAC limit: 4000, Action: none, Notification: syslog
Filter MAC addresses: 0
ACs: 0 (0 up), VFIs: 0, PWs: 0 (0 up), PBBs: 1 (1 up)
List of PBBs:
  PBB Core, state: up
List of ACs:
List of Access PWs:
List of VFIs:
```

To see additional details about how the PBB edge domains are mapped to PBB core domains, we use the “detail” keyword. This reveals much more information about the behavior of the bridge-domain, to including flooding and MAC operations.

```
RP/0/0/CPU0:XRv1#show l2vpn bridge-domain bd-name PBB_EDGE_1 detail
Legend: pp = Partially Programmed.
Bridge group: PBB_EDGES, bridge-domain: PBB_EDGE_1, id: 1, state: up, ShgId:
0, MSTi: 0
Coupled state: disabled
Type: pbb-edge, I-SID: 1001
Core-bridge: PBB_CORE (State: Bridge Up)
```

```

MIRP-lite: not supported
  Format: none
MAC learning: enabled
MAC withdraw: enabled
  MAC withdraw for Access PW: enabled
  MAC withdraw sent on: bridge port up
  MAC withdraw relaying (access to access): disabled
Flooding:
  Broadcast & Multicast: enabled
  Unknown unicast: enabled
[snip]

```

Using “detail” on the PBB core domain shows additional EVI information. Obviously the packet/byte counts are incorrect since no traffic is flowing, but basic BGP information (ASN, RD, etc) is shown as well. MAC and flooding operations are at the bottom and most of it is omitted for brevity.

```

RP/0/0/CPU0:XRv1#show l2vpn bridge-domain bd-name PBB_CORE detail
Legend: pp = Partially Programmed.
Bridge group: PBB_CORE, bridge-domain: PBB_CORE, id: 0, state: up, ShgId: 0,
MSTi: 0
  Coupled state: disabled
  Type: pbb-core
  Number of associated pbb-edge BDs: 1
  EVPN:
    EVI: 1000
    Route Distinguisher: (auto) 1.11.11.11:1000
    Imposition Statistics:
      Packet Count: 293221265588617217
      Byte Count   : 293183607383632832
    Disposition Statistics:
      Packet Count: 311937118322620165
      Byte Count   : 1176759140103596992
    AS Number: 1
  MAC learning: enabled
  MAC withdraw: enabled
    MAC withdraw for Access PW: enabled
    MAC withdraw sent on: bridge port up
    MAC withdraw relaying (access to access): disabled
  Flooding:
    Broadcast & Multicast: enabled
    Unknown unicast: enabled
[snip]

```

Additional Reading – Reference configurations “pbb-evpn”

11. Describe IEEE 802.1ad (QinQ), IEEE 802.1ah (Mac-in-Mac), and ITU G.8032 (REP)

11.1 802.1ad QinQ

QinQ (dot1q tunneling or provider bridging) is a method of adding additional 802.1q headers to an Ethernet frame to tunnel it across different networks. 802.1ad is an amendment to 802.1q which doesn't change any aspects of dot1q headers per se, but simply allows them to be stacked in an Ethernet frame's header. This is demonstrated many times throughout this document so a dedicated lab is not built for this section. There are many benefits to tag stacking:

1. The original specification can create 2^{12} (4096) theoretical VLANs since the VLAN-ID is a 12 bit number. VLANs 0 and 4096 cannot be used, though, so the math is imprecise so far. Adding an additional tag squares this ($2^{24} = 16777216$) which allows for greater network growth. I personally use this extensively in many labs where I don't want to provision public VLANs (I am only allowed to use VLANs 3501 – 3600 for my labs), so I tunnel my private VLANs inside of these publicly switched VLANs.
2. Tags can be added (pushed), removed (popped), or changed (translated) by network devices as necessary.
3. Provides the basis for provider bridging and provider backbone bridging (PBB) which is described later. When used in conjunction with PBB, hierarchical bridging networks can be designed to achieve very high scalability.

The standard denotes Ethertype 0x88A8 for the outer tag (service tag, carrier tag, metro tag, tag 1, or S-tag) and 0x8100 for the inner tag (customer tag, tag 2, or C-tag). This new Ethertype isn't terribly significant other than identifying the Ethernet frame as tunneled or not. Below is a copy of a packet dump from another section later in the lab that shows the QinQ stack. In this case, the S-tag in yellow has value 3544 (0xDD8) and the C-tag in green has value 460. As mentioned earlier, this can be used to tunnel VLANs not allowed by my cloud provider inside VLANs that are. Immediately following the QinQ encapsulation is the ethertype of 0x0800 (cyan), which identifies an IP packet inside of the following Ethernet frame.

```
R6#show monitor capture CAP buffer detail
 6 152      1.376039  00:50:56:A9:DE:0D -> 00:50:56:A9:EA:77 MPLS unicast
0000:  005056A9 EA770050 56A9DE0D 81000DEF      .PV..w.PV.....
0010:  884701B5 D0FF00FA A1FF0000 00000050      .G.....P
0020:  56A9862A 005056A9 EA5488A8 0DD88100      V...*.PV..T.....
0030:  01CC0800 45000064 00010000 FF01A77D      ....E..d.....}
```

Although 802.1ad does not specify more than 2 tags pushed, there is no limitation in many devices to this upper bound. In that case, the inner-most tag is the C-tag where all other tags are considered S-tags. The outer-most tag is "tag 1" and the next tag is "tag 2", which would be another S-tag if more than 2 tags exist. Another interesting change in 802.1ad is that the canonical format ID (CFI) field, formerly used when bridging token ring networks, has been recycled for drop probability use. This is similar to the frame-relay discard eligibility (DE) bit and is called the drop eligibility indicator (DEI). This can be used in conjunction with or in lieu of the 3 priority bits in the dot1q header which are also examined in later sections. Those priority bits are referenced in 802.1p and are called the priority code points (PCP).

Q-tunneling has some downsides as well. The C-MAC addresses are always exposed, even when being bridged over a service provider's network. The S-tag helps keep the C-tag intact and separates some data-plane functionality, but the provider network still must program all C-MACs within the S-VLAN CAM tables on all switches in the transit path. This leads to poor scalability for data center interconnect (DCI) applications where many customers will have many C-MACs. QinQ also makes the assumption that no two networks will ever have the same MAC address, which may not be true if custom MAC addressing is configured in software for aggregation/filtering reasons.

11.2 802.1ah MAC in MAC (Provider Backbone Bridges)

This feature was written to address some of the concerns previously identified with QinQ tunneling. It is also known as provider backbone bridging (PBB) since it introduces the concept of backbone MACs (B-MACs). Like S-tags shown in QinQ tunnel, B-MACs are service provider switch MACs that serve as the outer encapsulation for customer traffic. In this way, provider switches don't need to learn all C-MACs, only B-MACs, of which there is one per PE. This greatly enhances scalability as the B-MAC is like a tunnel or aggregate MAC. PBB was defined to offer total separation between customer and provider MAC address spaces, which is somewhat similar to MPLS L3VPN (or IP-in-IP/GRE) does for IP VPNs. PBB is comprised of three main encapsulation components:

1. **Backbone component:** Contains a destination address (B-DA), source address (B-SA), the ethertype 0x88A8 as defined in 802.1ad for QinQ tunneling, and a B-tag to represent the backbone VLAN. The B-tag is similar to the S-tag but is renamed for clarity between the specifications, since PBB technically can carry QinQ frames inside of it.
2. **Service component:** Contains ethertype 0x88E7 (defined for PBB) and an I-SID (24-bit service ID). The I-SID is used as a method of defining to which customer the frames belong, since the B-tag isn't customer specific.
3. **Original frame:** This could be an IP packet inside of an Ethernet frame (0x0800), an Ethernet frame inside of dot1q with a single C-tag (0x8100), or part of a hierarchical bridging architecture with two tags already on it (0x88A8). This is why the B-tag isn't called the S-tag, as the S-tag may actually exist within the original frame that arrived at the PBB device. This component is highly variable in length and composition, but PBB doesn't care.

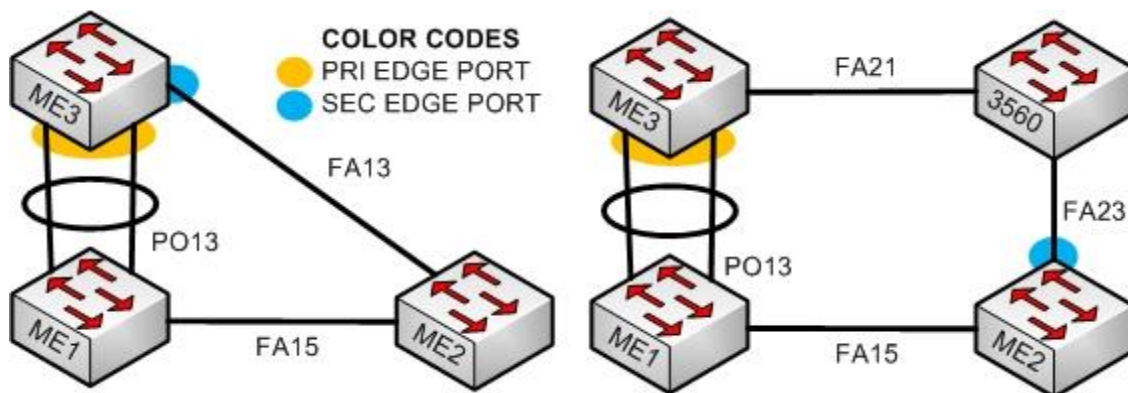
This PBB encapsulation can be used in an ordinary Ethernet switched network or as a pre-encapsulation before MPLS. In PBB-EVPN, this PBB encapsulation happens first, then the entire backbone frame is encapsulated inside MPLS labels to represent the remote PE within a VPN (BGP) and the transport labels to get there (LDP, RSVP-TE, SR, etc). I am unable to demonstrate this encapsulation due to lack of hardware devices that support it.

11.3 Ethernet Ring loop-prevention

11.3.1 Cisco Resilient Ethernet Protocol (REP)

REP is a Cisco-proprietary protocol that is used to rapidly converge a layer 2 Ethernet network arrayed in a ring topology. This is an alternative to STP which works in a very specific set of network designs, but offers very fast (50 – 300 ms) convergence time. The entire "circumference" of the ring, including the endpoints, are part of the REP domain. This domain is called a "segment" in REP terms, but I personally

find this term confusing. The segment is not like a LAN segment, nor is it a single link between two switches. Rather, the segment is the end-to-end circumference of the REP ring, which is terminated by exactly two edge ports. Each switch in the REP segment can have up to 2 ports in a segment (for intermediate nodes), and each link must have exactly one REP neighbor. To illustrate this, we will test two different REP topologies shown below. The left diagram is a “ring segment” in which one switch terminates the segment on both ends, since both edge ports are configured on this one switch. This is a simpler design but it is highly reliant on this one switch. The right diagram is an “open segment” in which the edge ports exist on disparate switches. The network in between can use STP (Cisco Catalyst 3560, for example), and since there are multiple switches with REP edge ports, this more complex design offers higher availability (more ingress/egress points).



First, we will configure the ring topology. REP is very easy to configure as there are not many commands. Before configuring any interfaces, we will discuss the REP admin VLAN. This VLAN is used for the flooding of REP messages to an ordinary MAC multicast address, which is the IEEE STP MAC 0180.c200.0000 also used for STP on VLAN 1 in PVST, for example. The reason this ordinary address is used is to improve REP performance; by using the hardware flood layer (HFL), REP bypasses the delay with software forwarding. Certain REP messages are therefore flooded throughout the entire REP admin VLAN, even beyond the REP segment. For this reason, the admin VLAN can be pruned to only exist in the REP segment, giving the operator additional control. We will configure VLAN 29 for this. It is important to note that the REP link-status layer (LSL) PDUs, which are like STP BPDUs, are always sent untagged in the native VLAN.

```
! All ME switches
vlan 29
  name REP_ADMIN

rep admin vlan 29
```

We can verify this configuration by checking the REP interface details, which is not intuitive. There are many other important fields in this output we look at later, but for now, we just verify the admin VLAN.

```
ME3#show interfaces port-channel 13 rep detail | include Admin-vlan
```

Admin-vlan: 29

There are two key requirements for enabling REP on an interface: it must be an NNI and it must be a trunk port. Any other combination of port-types and switch port modes is not supported. REP is also supported on port-channel interfaces as we shown between ME1 and ME3. Of the two edge ports, one is the “primary” edge port and one is the secondary. This is relevant later, but for now, we configure the LACP bond as the primary towards ME1 and the other port towards ME2 as the secondary. Ideally, you would work clockwise or counter-clockwise around the ring and not configure both edge ports at the same time. Moving in sequence is the best way to migrate to REP from STP, but since this is a “greenfield” deployment, I will configure all REP ports per device as I have console access and don’t care about temporary traffic blackholes.

```
! ME3
interface Port-channel13
  description LACP TO ME1
  port-type nni
  switchport mode trunk
  rep segment 1 edge primary

interface FastEthernet0/13
  description TO ME2
  port-type nni
  switchport mode trunk
  rep segment 1 edge
```

Notice that configuring REP generates a message warning you about the effects on STP. STP is totally disabled on any REP port, which means BPDUs are not sent nor received. We can prove this with a show command as well.

```
ME3(config-if)#rep segment 1 edge
Warning: Enabling REP automatically disables STP on this port.  It is
recommended to shutdown all interfaces which are not currently in use to
prevent potential bridging loops.
```

```
ME3#show spanning-tree interface fastEthernet 0/13
no spanning tree info available for FastEthernet0/13
```

Configuring the intermediate nodes is simple. All we do is specify the REP segment as there is no need to specify edge port status. I do not show the descriptions, NNI, or trunk configurations for brevity.

```
! ME2
interface FastEthernet0/13
  rep segment 1

interface FastEthernet0/15
  rep segment 1
```

```

! ME3
interface FastEthernet0/15
  rep segment 1

interface Port-channel13
  rep segment 1

```

At this point, we can use some simple show commands to verify the REP topology. Assuming the ring is complete, all nodes will have the same view of the topology, so we can issue this command anywhere. The command starts with the primary edge port and moves around the ring until it reaches the secondary edge port. Each bridge and port along the way are identified, along with their port roles. Notice that ME2 port 15 is blocking; this is similar to STP where one port in the ring must be blocking to prevent a loop. This means that traffic from ME2 to ME1 will transit ME3, just like in STP. These ports are called “alternate” ports because a break in the ring anywhere automatically means this blocked port will be opened. If the edge ports lose connectivity, there is a never a case where the ring should contain a blocked port.

```

ME3#show rep topology segment 1
REP Segment 1

```

BridgeName	PortName	Edge	Role
ME3	Po13	Pri	Open
ME1	Po13		Open
ME1	Fa0/15		Open
ME2	Fa0/15		Alt
ME2	Fa0/13		Open
ME3	Fa0/13	Sec	Open

Before we dig into detailed show commands, we will verify REP operation. Using a local SPAN session on ME3, I look at traffic on ME3’s primary edge port. At present, we can see several LSL PDUs captured by Wireshark.

```

! ME3
monitor session 1 source interface Po13
monitor session 1 destination interface Fa0/12 encapsulation replicate

```

No.	Time	Source	Destination	Protocol	Length
	20.007003000	CiscoInc_0b:16:93	Spanning-tree-(for-bridges)_00	LLC	
	20.671019000	CiscoInc_92:f7:13	Spanning-tree-(for-bridges)_00	LLC	
+ Frame 35: 590 bytes on wire (4720 bits), 590 bytes captured (4720 bits) on interface					
- IEEE 802.3 Ethernet					
+ Destination: Spanning-tree-(for-bridges)_00 (01:80:c2:00:00:00)					
+ Source: CiscoInc_92:f7:13 (00:1d:46:92:f7:13)					
Length: 576					
- Logical-Link Control					
+ DSAP: SNAP (0xaa)					
+ SSAP: SNAP (0xaa)					
+ Control field: U, func=UI (0x03)					
Organization Code: Cisco (0x00000c)					
PID: Unknown (0x0129)					
+ Data (568 bytes)					

It is interesting to see that Wireshark is calling this an STP frame, primarily because it doesn't understand REP, and really only understands the STP destination MAC address. That MAC address is highlighted below in the REP packet debug (green). I also verify the MAC address of ME1's LACP bond as this is the source MAC of the frame (yellow). This debug matches the Wireshark capture, and because the Ethertype isn't specified (it's a length of 576, which is 590 – 14 bytes of Ethernet encapsulation), the DSAP and SSAP are. Both of them are 0xAA (pink), so I am confident that this is the same type of packet seen above.

```
ME1#show interfaces port-channel 13 | include bia
Hardware is EtherChannel, address is 001d.4692.f713 (bia 001d.4692.f713)
```

```
ME3#debug rep packet
REP debug packets debugging is on
```

```
! ME2
00:43:21: REP store seq#1310 for re_tx on Fa0/13 (pak ref:2) fifo size <2>
00:43:21: REP Rx packet on Port-channel13 <len 590>
00:43:21: 0x00 0x80 0x00 0xCB 0x00 0x00 0x00 0x1A
00:43:21: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
00:43:21: 0x01 0x80 0xC2 0x00 0x00 0x00 0x00 0x1D
00:43:21: 0x46 0x92 0xF7 0x13 0x02 0x40 0xAA 0xAA
```

For fastest convergence, we can reduce the LSL age timer on each interface. We can only set the “age timer”, which is a dead timer, and the hello interval is one third of this value. The downside to using port-channels with REP is that the fastest age timer is 1 seconds.

```
ME3(config-if)#rep lsl-age-timer 120
Error: LSL Age Out timer value less than 1000 not supported on etherchannel
```

That being said, we will configure an LSL age-timer of 1 second on the port-channel interface and the minimum of 120 ms on all other interfaces. Notice that if you have mismatched timers, the REP ring

becomes unstable. By configuring only ME3 with this new interval, I've created a mismatch with ME1. ME3 is flapping up and down since ME1 is sending hellos too slowly.

```
! ME3
interface Port-channel13
  rep lsl-age-timer 1000

! ME1
%REP-4-LINKSTATUS: Port-channel13 (segment 1) is non-operational due to
neighbor mismatch

! ME3
%REP-4-LINKSTATUS: Port-channel13 (segment 1) is operational
%REP-4-LINKSTATUS: Port-channel13 (segment 1) is non-operational due to
neighbor not responding
%REP-4-LINKSTATUS: Port-channel13 (segment 1) is operational
%REP-4-LINKSTATUS: Port-channel13 (segment 1) is non-operational due to
neighbor not responding
```

We can confirm the issue by checking the REP interface details on both ME1 and ME3 to see the different timers. Filtering on "LSL" also shows us the PDU counters, similar to how STP interface-level show commands would reveal BPDU TX/RX counters.

```
ME3#show interfaces port-channel 13 rep detail | include LSL
LSL Ageout Timer: 1000 ms
LSL PDU rx: 4964, tx: 3907
BPA (STCN, LSL) TLV rx: 0, tx: 0
```

```
ME1#show interfaces port-channel 13 rep detail | include LSL
LSL Ageout Timer: 5000 ms
LSL PDU rx: 3921, tx: 5104
BPA (STCN, LSL) TLV rx: 0, tx: 0
```

To fix it, we simply match the timers on ME1 using the same configuration (not shown). Then, I adjust the timer to 120 ms on all ordinary ports (also not shown). The blinking lights on the switches are going crazy now, as expected, on the inter-switch links not including the LACP members. Below is a quick verification of all REP interfaces to show their LSL ageout timers.

```
ME1#show interfaces rep detail | include ^(Fast|Port-)|Ageout
FastEthernet0/15    REP enabled
LSL Ageout Timer: 120 ms
Port-channel13     REP enabled
LSL Ageout Timer: 1000 ms
```

```
ME2#show interfaces rep detail | include ^(Fast|Port-)|Ageout
FastEthernet0/13    REP enabled
LSL Ageout Timer: 120 ms
```

```
FastEthernet0/15    REP enabled
LSL Ageout Timer: 120 ms
```

```
ME3#show interfaces rep detail | include ^(Fast|Port-)|Ageout
```

```
FastEthernet0/13    REP enabled
LSL Ageout Timer: 120 ms
Port-channel13      REP enabled
LSL Ageout Timer: 1000 ms
```

Even with this fast timer, only about 12-16 kbps of bandwidth is used. It only consumes a small amount of CPU cycles as well, but more REP interfaces implies more CPU consumed for hello generation/processing. These are important things to check when using any kind of fast failover active detection mechanism.

```
ME1#show interfaces fastEthernet 0/15 | include put_rate
 5 minute input rate 16000 bits/sec, 23 packets/sec
 5 minute output rate 12000 bits/sec, 23 packets/sec
```

```
ME1#show processes cpu sorted
CPU utilization for five seconds: 5%/0%; one minute: 6%; five minutes: 6%
PID Runtime(ms)   Invoked      uSecs   5Sec   1Min   5Min  TTY Process
 198      10674        2175    4907  0.31%  0.08%  0.04%   0 Exec
 126       2642     125673      21  0.31%  0.35%  0.30%   0 REP LSL Hello PP
```

To avoid flooding these LSLs to our Wireshark capture, from now on we will only look at messages in the admin VLAN 29. We apply a quick SPAN filter to achieve this, which allows us to test REP failover. I verify the SPAN configuration because failing to do so may degrade my sniffer PC.

```
! ME3
monitor session 1 filter vlan 29
```

```
ME3#show monitor session 1
Session 1
-----
Type                : Local Session
Source Ports        :
    Both            : Po13
Destination Ports   : Fa0/12
    Encapsulation   : Replicate
    Ingress         : Disabled
Filter VLANs        : 29
```

Before we simulate a failure, we can trace the basic traffic flow for a number of VLANs. Test VLANs 100-103 have been defined on every switch with corresponding SVIs. We quickly check the REP topology again for reference, which indicates port 15 on ME2 is blocking. Traffic between ME1 and ME2 should therefore transit ME3.

```
ME3#show rep topology segment 1
REP Segment 1
BridgeName      PortName      Edge Role
-----
ME3              Po13          Pri  Open
ME1              Po13          Open
ME1              Fa0/15        Open
ME2              Fa0/15        Alt
ME2              Fa0/13        Open
ME3              Fa0/13        Sec  Open
```

To prove this, I ping the broadcast address from ME2. This will source broadcast pings from all connected interfaces, which should quickly discover all of the other SVIs on the network. Specifically, we want to ensure we see responses from ME1's test VLANs.

```
ME2#ping 255.255.255.255 repeat 1
```

Type escape sequence to abort.

Sending 1, 100-byte ICMP Echos to 255.255.255.255, timeout is 2 seconds:

```
Reply to request 0 from 10.0.10.91, 8 ms
Reply to request 0 from 103.0.0.3, 16 ms
Reply to request 0 from 102.0.0.3, 16 ms
Reply to request 0 from 103.0.0.1, 16 ms
Reply to request 0 from 101.0.0.3, 16 ms
Reply to request 0 from 102.0.0.1, 8 ms
Reply to request 0 from 100.0.0.3, 8 ms
Reply to request 0 from 101.0.0.1, 8 ms
Reply to request 0 from 10.0.10.93, 8 ms
Reply to request 0 from 100.0.0.1, 8 ms
```

The MAC addresses for ME1 are shown below so that we can correlate them with ME2's CAM table for those 3 test VLANs. Notice how ME3 is the outgoing port for all of them (along with the 4 SVIs on ME3), and nothing is forwarded out of port 15. This shows that the REP blocked port is working as expected.

```
ME1#show interfaces vlan 100 | include bia
Hardware is EtherSVI, address is 001d.4692.f741 (bia 001d.4692.f741)

ME1#show interfaces vlan 101 | include bia
Hardware is EtherSVI, address is 001d.4692.f742 (bia 001d.4692.f742)

ME1#show interfaces vlan 102 | include bia
Hardware is EtherSVI, address is 001d.4692.f743 (bia 001d.4692.f743)

ME1#show interfaces vlan 103 | include bia
Hardware is EtherSVI, address is 001d.4692.f744 (bia 001d.4692.f744)

ME2#show mac address-table dynamic | exclude _1_
```

Mac Address Table

Vlan	Mac Address	Type	Ports
100	001d.4692.f741	DYNAMIC	Fa0/13
100	001f.9d0b.16c1	DYNAMIC	Fa0/13
101	001d.4692.f742	DYNAMIC	Fa0/13
101	001f.9d0b.16c2	DYNAMIC	Fa0/13
102	001d.4692.f743	DYNAMIC	Fa0/13
102	001f.9d0b.16c3	DYNAMIC	Fa0/13
103	001d.4692.f744	DYNAMIC	Fa0/13
103	001f.9d0b.16c4	DYNAMIC	Fa0/13

Now, I will simulate a failure by unplugging the cable between ME2 and ME3. First, I enable pertinent REP debugs on all switches to observe the event.

```
! All switches
debug rep failure-recovery
debug rep prsm
```

Because this breaks the ring, traffic should immediately be allowed to forward over ME2's port 15. First, ME2 and ME3 receive the link-down notification from the platform after the carrier-delay expires. Both of them show the exact same debug as they both witness the layer 1 failure. The port transitions from the OPEN to FAILED state. Both switches also flush MAC addresses learned on this port from the CAM table, which will trigger temporary unknown unicast flooding until the new port is learned. Each of them notify the other switches in the segment as well.

```
! ME2 and ME3
[Fa0/13]: Link down notification received
    rep_pr Fa0/13 - pr: during state OPEN_PORT, got event 1(link_not_op)
@@@ rep_pr Fa0/13 - pr: OPEN_PORT -> FAILED_PORT[Fa0/13]rep_pr_act_fp@101:
PRSM->failed state, 0x80

HREP: Sent hfl pak on Fa0/13, vlan_id 29, di 379
REP Flush from Fa0/13 to REP, sending msg[Fa0/13]: Local failure handling
[Fa0/13]rep_pr_act_send_hw_ind@251: PRSM->sending hw failure
```

Even with debug off, both ME2 and M3 generate syslog messages to show the REP link failure.

```
! ME2 and ME3
%REP-4-LINKSTATUS: FastEthernet0/13 (segment 1) is non-operational due to
port down
```

ME1 is connected to both switches and receives this HW failure indication from both ME2 and ME3 on its LACP bond and port 15.

```

! ME1
REP LSL-OP Rx EXT Local (Fa0/15) seg:1, tc:1, frs:0) prio:
0x80 0x00 0x0F 0x04 0xC5 0xA4 0x53 0xD8
0x00
REP Flush from Fa0/15 to REP, sending msg
REP LSL-OP Rx INT Local (Po13) seg:1, tc:1, frs:0) prio:
0x80 0x00 0x0F 0x04 0xC5 0xA4 0x53 0xD8
0x00
REP Flush from Po13 to REP, sending msg

```

From a topology perspective, we can see that ME3 sees a failure in the topology. Failed ports terminate segments just as edge ports do, however since the segment is incomplete, all ports are open. We now see that ME2 port 15 is available for forwarding. We also check it from ME2's perspective which lists the nodes in the reverse order, but with the same port roles. The presence of a failed port on any segment causes REP to warn the user that there is a segment failure.

```

ME3#show rep topology segment 1
REP Segment 1
Warning: REP detects a segment failure, topology may be incomplete

```

BridgeName	PortName	Edge	Role
ME3	Po13	Sec	Open
ME1	Po13		Open
ME1	Fa0/15		Open
ME2	Fa0/15		Open
ME2	Fa0/13		Fail

```

ME2#show rep topology segment 1
REP Segment 1
Warning: REP detects a segment failure, topology may be incomplete

```

BridgeName	PortName	Edge	Role
ME2	Fa0/13		Fail
ME2	Fa0/15		Open
ME1	Fa0/15		Open
ME1	Po13		Open
ME3	Po13	Sec	Open

We ping the broadcast address from ME2 again (not shown) so that all MAC addresses in all VLANs are resolved. Check the CAM for all non-VLAN 1 entries, we can see that port 15 is used to reach ME1's test SVIs as expected.

```

ME2#show mac address-table dynamic | exclude _1_
Mac Address Table
-----

```

Vlan	Mac Address	Type	Ports
----	-----	-----	-----
100	001d.4692.f741	DYNAMIC	Fa0/15
100	001f.9d0b.16c1	DYNAMIC	Fa0/15
101	001d.4692.f742	DYNAMIC	Fa0/15
101	001f.9d0b.16c2	DYNAMIC	Fa0/15
102	001d.4692.f743	DYNAMIC	Fa0/15
102	001f.9d0b.16c3	DYNAMIC	Fa0/15
103	001d.4692.f744	DYNAMIC	Fa0/15
103	001f.9d0b.16c4	DYNAMIC	Fa0/15

With debugging enabled, we will plug the cable back in. Both ME2 and ME3 change their port state from “FAILED” to “FAILED with no external neighbor”. This is an indication that the electrical carrier is back up but REP is not operational yet.

```
! ME2 and ME3
@@@ rep_pr Fa0/13 - pr: FAILED_PORT ->
FAILED_PORT_NO_EXT_NEIGHBOR[Fa0/13]rep_pr_act_no_ext_neighbor@415: PRSM-
>fp_no_ext_neighbor state
[Fa0/13]rep_pr_lsl_event_handler@645: REP_MSG_EXT_PEER_GONE rcvd
```

After exchanging LSL PDUs, the two neighbors discover one another again, and move their ports to the alternate state. There is still no forwarder on the segment, which is incomplete.

```
! ME2 and ME3
    rep_pr Fa0/13 - pr: during state FAILED_PORT_NO_EXT_NEIGHBOR, got event
0(link_op)
@@@ rep_pr Fa0/13 - pr: FAILED_PORT_NO_EXT_NEIGHBOR ->
ALTERNATE_PORT[Fa0/13]rep_pr_act_ap@216: PRSM->alternate state
[Fa0/13]rep_pr_lsl_event_handler@625: REP_MSG_LINKOP_TRUE rcvd
```

Since one of the ports needs to be forwarding, an election process takes place during the formation of a REP ring which was not discussed until now as it is very detailed. Like STP, each port has a 64-bit priority associated with it, and in REP it is the concatenation of several fields. Towards the beginning the first 4 bits of the election include a failed bit (the most significant bit, only set on failed ports) followed by user configurable fields. The next 12 bits represent a port identifier, and the remaining 48 represent the bridge MAC address. The alternate port election only happens on these blocked ports when an “open” port must be chosen, so the switches exchange blocked port advertisements (BPAs) containing their priorities. If a port receives a BPA with a higher priority, it transitions to the open state as shown below. At the end of the link-up condition, REP displays a syslog message.

```
! ME3
    rep_pr Fa0/13 - pr: during state ALTERNATE_PORT, got event
2(pre_empty_ind)
@@@ rep_pr Fa0/13 - pr: ALTERNATE_PORT -> UNBLOCK_VLANS_ACT
```

```

    rep_pr Fa0/13 - pr: during state UNBLOCK_VLANS_ACT, got event
3(no_local_block_vlan)
@@@ rep_pr Fa0/13 - pr: UNBLOCK_VLANS_ACT ->
OPEN_PORT[Fa0/13]rep_pr_act_op@391: PRSM->active state

! ME2 and ME3
%REP-4-LINKSTATUS: FastEthernet0/13 (segment 1) is operational

```

To see the priorities, we can show the details of the REP topology. Since the link is not failed and the port priorities are both zero, the port numbers are compared. These also match, so then the MAC addresses are compared. Since ME3 port 13 receives a BPA with a better priority, it opens, and since ME2 received a BPA with a worse priorities, it remains an alternate port. BPAs are only sent by alternate ports during this election process as this information contains the VLAN information for blocked VLANs; VLAN load sharing is discussed later.

```

ME3#show rep topology segment 1 detail | begin ME2, Fa0/13
ME2, Fa0/13 (Intermediate)
  Alternate Port, some vlans blocked
  Bridge MAC: 04c5.a453.d800
  Port Number: 00F
  Port Priority: 000
  Neighbor Number: 5 / [-2]
ME3, Fa0/13 (Secondary Edge)
  Open Port, all vlans forwarding
  Bridge MAC: 001f.9d0b.1680
  Port Number: 00F
  Port Priority: 000
  Neighbor Number: 6 / [-1]

```

While we don't have much control over the exact port priority values, we can administratively configure a preferred alternate port. This is somewhat unintuitive since we normally adjust priorities to prefer something, but in this case, the opposite is true. Since the reception of a superior BPA causes a port to open, we can configure a higher priority on ME3 port 13 so that ME2 port 13 becomes open for this link. We now identify this port as the secondary edge port with the "preferred" flag. We can verify the configuration by checking the REP interface details.

```

! ME3
interface FastEthernet0/13
  rep segment 1 edge preferred

ME3#show interfaces fastEthernet 0/13 rep detail | include Pref
Segment-id: 1 (Preferred Edge)
Preferred flag: Yes

```

However, when we check the REP topology, nothing has changed. Despite having a higher priority, ME3 port 13 is still forwarding.

```
ME3#show rep topology segment 1 detail | begin ME2, Fa0/13
ME2, Fa0/13 (Intermediate)
  Alternate Port, some vlans blocked
  Bridge MAC: 04c5.a453.d800
  Port Number: 00F
  Port Priority: 000
  Neighbor Number: 5 / [-2]
ME3, Fa0/13 (Secondary Edge)
  Open Port, all vlans forwarding
  Bridge MAC: 001f.9d0b.1680
  Port Number: 00F
  Port Priority: 010
  Neighbor Number: 6 / [-1]
```

The alternate port elections only happen once during the initial negotiation when a segment is complete, and they are not preemptive. As such, I quickly disconnect and reconnect the cable which triggers the election again. Now, ME2 port 13 is forwarding and the secondary edge port is blocked. In this way, we can somewhat influence the REP convergence around the ring.

```
ME3#show rep topology segment 1 detail | begin ME2, Fa0/13
ME2, Fa0/13 (Intermediate)
  Open Port, all vlans forwarding
  Bridge MAC: 04c5.a453.d800
  Port Number: 00F
  Port Priority: 000
  Neighbor Number: 5 / [-2]
ME3, Fa0/13 (Secondary Edge)
  Alternate Port, some vlans blocked
  Bridge MAC: 001f.9d0b.1680
  Port Number: 00F
  Port Priority: 010
  Neighbor Number: 6 / [-1]
```

Unplugging the cable again, we can see the “failed” bit in action. Notice that the priority is a very high number, and nothing can ever be higher than it. As such, assuming the link was up but there was a REP communication issue, this port would always be the alternate port.

```
ME3#show rep topology segment 1 detail | begin ME2, Fa0/13
ME2, Fa0/13 (Intermediate)
  Failed Port, Reason: Physical link down
  Bridge MAC: 04c5.a453.d800
  Port Number: 00F
  Port Priority: 800
  Neighbor Number: Not available
```


Wireshark doesn't help us decode the HFL packets at all, but we note something interesting. First, despite the dot1q tag not showing up, we know that this was in VLAN 29 since the local SPAN filter is only capturing that VLAN. The destination MAC address looks similar to the one typically used by Cisco for VTP, CDP, and others, except the last character is an 'e'. Wireshark identifies this as ISL encapsulation. Either way, we know that since it isn't the IEEE STP MAC address, even STP-aware non-Cisco switches would forward this traffic throughout the network. That is why the administrative VLAN can scope the REP signaling traffic.

No.	Time	Source	Destination	Protocol	Length
27.	644018000	CiscoInc_53:d8:0f	ISL-Frame_cc:cc:ce	LLC	565
+ Frame 8: 565 bytes on wire (4520 bits), 565 bytes captured (4520 bi					
- IEEE 802.3 Ethernet					
+ Destination: ISL-Frame_cc:cc:ce (01:00:0c:cc:cc:ce)					
+ Source: CiscoInc_53:d8:0f (04:c5:a4:53:d8:0f)					

Edge ports also sent end point advertisements (EPAs) occasionally via LSL in the native VLAN (not HFL in the REP admin VLAN). We can enable debugs on ME3 to see the EPAs being sent, which contain much more information than normal LSLs. In fact, the large LSL packet we saw earlier in the packet dumps was probably an EPA. Both the primary and secondary edge ports originate these. The network is currently stable so the states aren't changing, but we can see the topology information summarized in the debug. The primary edge port lists itself first and then the hops to the secondary edge port. The secondary edge port does the opposite in the reverse direction; this is how the edge ports know the segment is complete.

```
ME3#debug rep epasm
REP debug epa sm debugging is on

! ME3
    rep_epa_edge Po13 - epa-edge: during state EDGE_PRIMARY, got event
0(epa_hello_tmo)
    @@@ rep_epa_edge Po13 - epa-edge: EDGE_PRIMARY -> EDGE_PRIMARY
EPA Append BridgeInfo TLV LEN = 13 ME3
EPA Append PortInfo TLV LEN = 18, SubType 2 Po13
EPA Create REP_TLV_EPA_INFO TLV LEN = 35 , SegNum 0
Msg Length Rx 147, Datagram Rx 175
Bridge:ME3 Port1:Po13
Bridge:ME1 Port1:Po13 Port2:Fa0/15
Bridge:ME2 Port1:Fa0/15 Port2:Fa0/13

    rep_epa_edge Fa0/13 - epa-edge: during state EDGE_SECONDARY, got event
0(epa_hello_tmo)
    @@@ rep_epa_edge Fa0/13 - epa-edge: EDGE_SECONDARY -> EDGE_SECONDARY
EPA Append BridgeInfo TLV LEN = 13 ME3
EPA Append PortInfo TLV LEN = 20, SubType 2 Fa0/13
EPA Create REP_TLV_EPA_INFO TLV LEN = 37 , SegNum 0
Msg Length Rx 149, Datagram Rx 177
```

```
Bridge:ME3  Port1:Fa0/13
Bridge:ME2  Port1:Fa0/13  Port2:Fa0/15
Bridge:ME1  Port1:Fa0/15  Port2:Po13
```

If we unplug the cable to cause a segment failure, the EPA state machine transitions the primary edge port to the secondary edge port. The reason for this is because the primary edge port is also responsible for VLAN load balancing; when there is a break in the segment, VLAN load balancing is impossible. As such, the edge port becomes a secondary port so it can effectively ignore that configuration. The debug also shows us that the triggering state change was a “segment break”.

```
! ME3
03:15:32:      rep_epa_edge Po13 - epa-edge: during state EDGE_PRIMARY, got
event 4(seg_break)
03:15:32: @@@ rep_epa_edge Po13 - epa-edge: EDGE_PRIMARY ->
EDGE_SECONDARY[Po13]rep_epa_edge_act_edge_sec@113: EPA: edge sec
[Po13]rep_epa_egde_act_update_key@208: new key=61757
```

Each port in a REP segment is numbered with offsets as well. This isn't the same as the hexadecimal port numbers seen earlier, but rather is a way of measuring distance away from edge ports. The primary edge port is number 1 and the secondary edge port is -1. Ports increment by 1 as they move away from the primary edge port and decrement by 1 as they move away from the secondary port. From any switch, we look at the topology details and filter based on the more important information. We can see that each port is assigned both a positive and negative number. The positive number measures the distance from the primary edge port and the negative number measures the distances from the secondary edge port.

```
! ME1
ME1#show rep topology segment 1 detail | include ^ME|Port,|Neighbor
ME3, Po13 (Primary Edge)
  Open Port, all vlans forwarding
  Neighbor Number: 1 / [-6]
ME1, Po13 (Intermediate)
  Open Port, all vlans forwarding
  Neighbor Number: 2 / [-5]
ME1, Fa0/15 (Intermediate)
  Open Port, all vlans forwarding
  Neighbor Number: 3 / [-4]
ME2, Fa0/15 (Intermediate)
  Open Port, all vlans forwarding
  Neighbor Number: 4 / [-3]
ME2, Fa0/13 (Intermediate)
  Open Port, all vlans forwarding
  Neighbor Number: 5 / [-2]
ME3, Fa0/13 (Secondary Edge)
  Alternate Port, some vlans blocked
  Neighbor Number: 6 / [-1]
```

Ports also have unique IDs. We can reveal these using the interface-level REP show command. Notice that this is the 64-bit field that is used in BPA elections to determine alternate ports. The port number is the 12-bit number following the first nibble. Below, the port number for ME1 port 15 is 0x011, or 17 in decimal. This makes sense since port 15 is the 17th port on the switch as the two SFPs count as the first two. Virtual interfaces like LACP bonds use system-generated values, like 160 (0x0A0) shown below.

```
ME1#show interfaces rep detail | include REP_enabled|^PortID
FastEthernet0/15    REP enabled
PortID: 0011001D4692F700
Port-channel13     REP enabled
PortID: 00A0001D4692F700
```

Using these numbers/port IDs, we can achieve VLAN load balancing. This is configured only on the primary edge port and performs two key tasks. First, the administrator specifies a port in the segment where certain VLANs should be blocked. Any VLANs not specified by this list are blocked at the primary edge port. An interface can be specified in one of three ways:

1. Using the port ID specified manually.
2. Using the neighbor offset number, positive or negative, specified manually.
3. Using the “preferred” keyword, which will reference the preferred port previously configured in the ring.

Normally, enforcing these kind of changes on a REP ring is disruptive. Cisco decided not to break the REP ring just to enable VLAN load balancing, so after configuring any of these three methods, we must enable the feature one of two ways, but in both cases, it is a disruptive event.

1. Manually pre-empt the ring from the switch with the primary edge port. This will likely result in some traffic disruption.
2. Configure a delay on the interface which begins counting down after a link failure and recovery.

We will quickly test all methods. Our first test will be using the port ID method. We will choose ME1 port 15 as the alternate port for VLANs 100 and 101, while the primary edge port (ME3 LACP bond) will be the alternate port for all other VLANs. A quick look at the topology shows no change, as expected.

```
! ME3
interface Port-channel13
  rep block port id 0011001D4692F700 vlan 100-101
```

```
ME3#show rep topology segment 1
REP Segment 1
BridgeName      PortName      Edge Role
-----
ME3             Po13          Pri  Open
ME1             Po13          Open
ME1             Fa0/15        Open
```

ME2	Fa0/15	Open
ME2	Fa0/13	Open
ME3	Fa0/13	Sec Alt

I will manually preempt this REP ring to enable VLAN load balancing. The parser warns us about traffic interruptions as well.

```
ME3#rep preempt segment 1
```

The command will cause a momentary traffic disruption.

```
Do you still want to continue? [confirm]y
```

Proceeding with Manual Preemption

Now, we look at the topology in detail. We see two alternate ports in the topology, both of which specify that “some” VLANs are blocked. The primary edge port has an increased port-priority, which means it will become the alternate port for this link. The same is true for the port we specified above, which is also an alternate port.

```
ME3#show rep topology segment 1 detail
```

```
REP Segment 1
```

ME3, Po13 (Primary Edge)

Alternate Port, some vlans blocked

Bridge MAC: 001f.9d0b.1680

Port Number: 0A0

Port Priority: 080

Neighbor Number: 1 / [-6]

ME1, Po13 (Intermediate)

Open Port, all vlans forwarding

Bridge MAC: 001d.4692.f700

Port Number: 0A0

Port Priority: 000

Neighbor Number: 2 / [-5]

ME1, Fa0/15 (Intermediate)

Alternate Port, some vlans blocked

Bridge MAC: 001d.4692.f700

Port Number: 011

Port Priority: 040

Neighbor Number: 3 / [-4]

ME2, Fa0/15 (Intermediate)

Open Port, all vlans forwarding

Bridge MAC: 04c5.a453.d800

Port Number: 011

Port Priority: 000

Neighbor Number: 4 / [-3]

ME2, Fa0/13 (Intermediate)

Open Port, all vlans forwarding

Bridge MAC: 04c5.a453.d800

Port Number: 00F

```

Port Priority: 000
Neighbor Number: 5 / [-2]
ME3, Fa0/13 (Secondary Edge)
Open Port, all vlans forwarding
Bridge MAC: 001f.9d0b.1680
Port Number: 00F
Port Priority: 010
Neighbor Number: 6 / [-1]

```

We can check the individual port-level details to see which VLANs are blocked per-port. Notice that the “block VLAN” output shows the VLANs actually blocked, while the “configured” outputs show only what is configured. The configuration only exists on the primary edge port, which is why ME1 doesn’t show it.

```

ME3#show interfaces port-channel 13 rep detail | include Block
Blocked VLAN: 1-99,102-4094
Configured Load-balancing Block Port: 0011001D4692F700
Configured Load-balancing Block VLAN: 100-101

```

```

ME1#show interfaces fast 0/15 rep detail | include Block
Blocked VLAN: 100-101
Configured Load-balancing Block Port: none
Configured Load-balancing Block VLAN: none

```

Based on this output, we quickly verify the CAM tables. Since VLAN 101 is blocked on ME1 port 15, ME1 should reach ME2’s VLAN 101 SVI via its LACP bond to ME3. That is to say, it should take the “long way” around the switched network. However, VLAN 102 traffic should go direct over port 15 to ME2. We can check the ARP cache and CAM table on ME1 to confirm this. VLAN 101 entries are in green and VLAN 102 entries are in yellow.

```

ME1#show ip arp | include 0\0\2
Internet 101.0.0.2 0 04c5.a453.d842 ARPA Vlan101
Internet 102.0.0.2 0 04c5.a453.d843 ARPA Vlan102

ME1#show mac address-table dynamic | include 04c5.a453.d84[23]
101 04c5.a453.d842 DYNAMIC Po13
102 04c5.a453.d843 DYNAMIC Fa0/15

```

Next, we will use the numbering method. We will block ME2 port 13 for VLANs 100 and 101. We can use the number 5 (distance from primary edge) or -2 (distance from secondary edge) for the same effect. The topology will not change unless we manually preempt it again, or configure a preempt delay. To trigger this, I unplug ME1 port 15, which isn’t part of this VLAN load balancing scheme. It should not matter since the BPA elections will ultimately block the port we specify below, along with the primary edge port.

```

! ME3
interface Port-channel13

```

```
rep preempt delay 15
rep block port -2 vlan 100-101
```

After plugging the cable back in, we can see that it takes 15 seconds for the VLAN load balancing to start. 15 seconds is the minimum timer, so the classic REP ring behavior will converge at first, then be changed. Ultimately, we see the primary edge port blocking, along with the port we specified. The offset of -2 specifies a port that is adjacent to the secondary edge port; we verify that by checking the topology as well.

```
ME3#show rep topology segment 1
REP Segment 1
BridgeName      PortName      Edge Role
-----
ME3             Po13          Pri  Open
ME1             Po13          Open
ME1             Fa0/15        Alt
ME2             Fa0/15        Open
ME2             Fa0/13        Open
ME3             Fa0/13        Sec  Open
```

```
ME3#show rep topology segment 1
REP Segment 1
BridgeName      PortName      Edge Role
-----
ME3             Po13          Pri  Alt
ME1             Po13          Open
ME1             Fa0/15        Open
ME2             Fa0/15        Open
ME2             Fa0/13        Alt
ME3             Fa0/13        Sec  Open
```

The primary edge port looks similar as it did before, except now it shows the configured offset of -2 instead of an explicit port ID. We see that only VLANs 100 and 101 are allowed through the primary edge port, while all other VLANs are allowed through ME2 port 13.

```
ME3#show interfaces port-channel 13 rep detail | include Block
Blocked VLAN: 1-99,102-4094
Configured Load-balancing Block Port: -2
Configured Load-balancing Block VLAN: 100-101
```

```
ME2#show interfaces fast 0/13 rep detail | include Block
Blocked VLAN: 100-101
Configured Load-balancing Block Port: none
Configured Load-balancing Block VLAN: none
```

We quickly confirm this on ME2 by looking at the MAC addresses in the CAM table for ME3 in VLAN 100 and VLAN 103. Traffic in VLAN 103 (and most other VLANs) goes directly to ME3 via port 13, but traffic for VLANs 100 and 101 goes the long way through ME1 via port 15.

```
ME2#show ip arp | include 10[03]\.0\.0\.3
Internet 100.0.0.3          6  001f.9d0b.16c1  ARPA  Vlan100
Internet 103.0.0.3          0  001f.9d0b.16c4  ARPA  Vlan103

ME2#show mac address-table dynamic | include 001f.9d0b.16c[14]
100  001f.9d0b.16c1  DYNAMIC  Fa0/15
103  001f.9d0b.16c4  DYNAMIC  Fa0/13
```

The last VLAN load balancing configuration option is to rely on the previously-configured “preferred” port. We configured this on the secondary edge port earlier. We quickly verify this again.

```
ME3#show interfaces fast 0/13 rep detail | include Prefer
Segment-id: 1 (Preferred Edge)
Preferred flag: Yes
```

On the primary edge port, we remove the offset option and use the preferred option instead. This is more dynamic since now the user can just move the “preferred” option around the ring to make adjustments, rather than use port IDs or neighbor offsets.

```
! ME3
interface Port-channel13
 rep block port preferred vlan 100-101
```

The result of this configuration is that both edge ports are alternate ports for complementary VLAN sets. I quickly unplug a cable and plug it back in on ME2; either link is fine. Looking at the REP topology summary after waiting 15 seconds for VLAN load balancing to start, this looks incorrect since both edges are blocking. One may think that the network is dysfunctional since no traffic can ever leave the REP ring, but this is normal for VLAN load balancing.

```
ME3#show rep topology segment 1
REP Segment 1
BridgeName      PortName      Edge Role
-----
ME3             Po13          Pri  Alt
ME1             Po13          Open
ME1             Fa0/15        Open
ME2             Fa0/15        Open
ME2             Fa0/13        Open
ME3             Fa0/13        Sec  Alt
```

The interface details are predictable as the configuration is based on the “prefer” flag, and the VLANs allowed are complementary.

```
ME3#show interfaces port-channel 13 rep detail | include Block
Blocked VLAN: 1-99,102-4094
Configured Load-balancing Block Port: prefer
Configured Load-balancing Block VLAN: 100-101
```

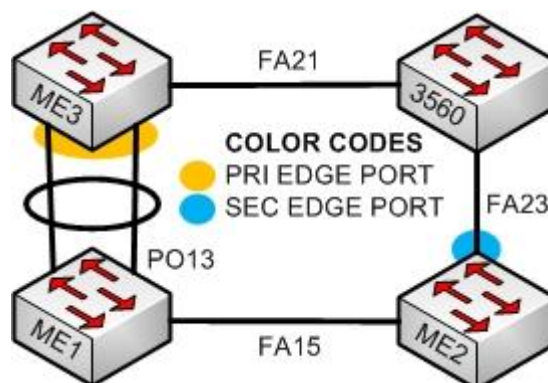
```
ME3#show interfaces fast 0/13 rep detail | include Block
Blocked VLAN: 100-101
Configured Load-balancing Block Port: none
Configured Load-balancing Block VLAN: none
```

We quickly verify that forwarding occurs according to this REP topology. ME1 in VLAN 100 is reachable via the LACP bond while ME1 in VLAN 102 is reachable via the indirect path through ME2. Personally, this kind of load balancing scheme is easiest to understand. If you make the secondary edge port the preferred port, then use the “prefer” option on the primary edge port, the forwarding paths are simple. Traffic will either flow through one edge port of the other and it’s very easy to determine which way the traffic is going. This is especially useful in “open segments” which are described next. The configurations for the REP ring segment are presented below separately.

```
ME3#show ip arp | include 10[02]\.0\.0\.1
Internet 100.0.0.1      0 001d.4692.f741  ARPA  Vlan100
Internet 102.0.0.1      0 001d.4692.f743  ARPA  Vlan102

ME3#show mac address-table dynamic | include 001d.4692.f74[13]
100 001d.4692.f741  DYNAMIC Po13
102 001d.4692.f743  DYNAMIC Fa0/13
```

Next, we modify the architecture slightly to introduce the REP open segment. A fourth switch (Cisco Catalyst 3560G) has been introduced to the ring, with produces a square. This switch is not REP-capable but will run RPVST+ and serve as the gateway into the rest of the notional layer 2 domain. Although REP is not compatible with STP, it can interwork seamlessly with it at the boundaries. ME2 and ME3 are connected to this C3560. The diagram is shown below again for reference.



We will make an edge port adjustment on ME2 by introducing a new option. We can specify the “uplink” port towards the C3560 as an edge port, but clarify that there will be no REP neighbor on it. This is loosely analogous to an IGP passive-interface, but in this case, the port is part of the REP ring. The benefit of this approach is that ME2 does not need to run STP for any VLANs now. On ME3, we will leave the primary edge port on the LACP bond to ME1, which means its uplink port will run STP. These uplink ports only carry VLANs 1 and 100-103, not the REP admin VLAN 29. This confines the BPAs to the REP ring. C3560 is the root for all VLANs and the basic configurations are shown below. Not shown are 4 test SVIs on C3560 for VLANs 100-103 which are used to test connectivity to the ME SVIs.

```
! ME2
interface FastEthernet0/15
  rep segment 1

interface FastEthernet0/23
  description TO C3560
  port-type nni
  switchport trunk allowed vlan 1,100-103
  switchport mode trunk
  rep segment 1 edge no-neighbor preferred
  rep stcn stp

! ME3
interface FastEthernet0/21
  description TO C3560
  port-type nni
  switchport trunk allowed vlan 1,100-103
  switchport mode trunk
  spanning-tree guard loop

! C3560
spanning-tree vlan 1-4094 priority 0

interface GigabitEthernet0/21
  description TO ME3
  switchport trunk encapsulation dot1q
  switchport trunk allowed vlan 1,100-103
  switchport mode trunk
  switchport nonegotiate
  spanning-tree guard root

interface GigabitEthernet0/23
  description TO ME2
  switchport trunk encapsulation dot1q
  switchport trunk allowed vlan 1,100-103
  switchport mode trunk
  switchport nonegotiate
  spanning-tree guard root
```

We first begin by verifying the REP topology. VLAN load balancing should still be in effect based on the “prefer” flag, which is still set on the secondary edge port. However, ME3 displays a syslog message that says the configuration is now invalid. In order for VLAN load balancing to work, the port specified must have a REP neighbor (I assume). The topology shows only one alternate port, which implies VLAN load balancing is disabled. The asterisk next to the edge port role means that there is no neighbor.

```
! ME3
%REP-5-PREEMPTIONFAIL: can not perform preemption on segment 1 due to invalid
Load-balancing block port
```

```
ME3#show rep topology segment 1
REP Segment 1
BridgeName      PortName      Edge Role
-----
ME3             Po13          Pri  Open
ME1             Po13          Open
ME1             Fa0/15        Open
ME2             Fa0/15        Alt
ME2             Fa0/23        Sec* Open
```

To fix this, we could move the preferred port to somewhere else, or use the port ID / neighbor offset method. I will use the neighbor offset of 4, which selects ME2 port 15. A value of -2 would have also worked.

```
ME3#show rep topology segment 1 detail | section ME2, Fa0/15
ME2, Fa0/15 (Intermediate)
  Alternate Port, some vlans blocked
  Bridge MAC: 04c5.a453.d800
  Port Number: 011
  Port Priority: 000
  Neighbor Number: 4 / [-2]
```

```
! ME3
interface Port-channel13
 rep block port 4 vlan 100-101
```

Next, we preempt the topology to enable VLAN load balancing. ME2 port 15 is now blocking VLANs 100 and 101 while the primary edge port blocks all other VLANs.

```
ME3#rep preempt segment 1
The command will cause a momentary traffic disruption.
Do you still want to continue? [confirm]y
Proceeding with Manual Preemption
```

```
ME3#show rep topology segment 1
```

REP Segment 1

BridgeName	PortName	Edge	Role
ME3	Po13	Pri	Alt
ME1	Po13		Open
ME1	Fa0/15		Open
ME2	Fa0/15		Alt
ME2	Fa0/23	Sec*	Open

Another important aspect of REP/STP interworking is topology change notification (TCN). REP has its own TCN mechanism for convergence, but STP has a TCN BPDU as well. Upon receipt of the TCN, a switch will reduce its MAC address aging timers from the default of 300 seconds to 35 seconds (max age plus forward delay time using default timers). This doesn't clear the current CAM table in regular STP, but it does in RSTP. It would be ideal if REP could inform the STP network about changes in the REP segment so that STP switches can age-out stale MAC addresses after a topology change. Because the segment has two ends, traffic that may have been flowing through one edge port was quickly transition to the other edge port, which is why the STP TCN propagation is useful. We will configure the REP segment TCN (STCN) to be reflected as an STP TCN on ME2 and ME3; notice that ME2 configures this on the port facing C3560 while ME3 does not. The command must go on the REP edge port regardless of whether the port is REP-facing (ME3) or STP-facing with no neighbor (ME2).

```
! ME2
interface FastEthernet0/23
 rep stcn stp
```

```
! ME3
interface Port-channel13
 rep stcn stp
```

We can see the details of the TCN propagation within the REP segment using the interface show commands. At present, the network is stable, so no TCNs have been sent. We can see that the STCN to STP TCN propagation has been enabled on this interfaces.

```
ME3#show interfaces port-channel 13 rep detail | include TCN
STCN Propagate to: STP
BPA (STCN, LSL) TLV rx: 0, tx: 0
BPA (STCN, HFL) TLV rx: 0, tx: 0
```

```
ME2#show interfaces fast 0/23 rep detail | include STCN
STCN Propagate to: STP
BPA (STCN, LSL) TLV rx: 0, tx: 0
BPA (STCN, HFL) TLV rx: 0, tx: 0
```

Based on the VLAN load-balancing scheme, traffic from C3560 to ME1 on VLAN 100 should flow through ME3 (primary edge port) and traffic to ME1 on VLAN 102 should flow through ME2 (secondary edge port). VLAN 100 is shown in yellow and VLAN 102 is shown in green.

```
C3560#show ip arp | include 10[02]\.0\.0\.1_
Internet 100.0.0.1          0 001d.4692.f741 ARPA Vlan100
Internet 102.0.0.1          0 001d.4692.f743 ARPA Vlan102

C3560#show mac address-table dynamic | include 001d.4692.f74[13]
100 001d.4692.f741 DYNAMIC Gi0/21
102 001d.4692.f743 DYNAMIC Gi0/23
```

On ME1, I shutdown port 15 which triggers a TCN. We can see the counter increment on both ME2 and M3 to represent this BPA activity.

```
ME3#show interfaces port-channel 13 rep detail | include TCN
STCN Propagate to: STP
BPA (STCN, LSL) TLV rx: 0, tx: 0
BPA (STCN, HFL) TLV rx: 0, tx: 1
```

```
ME2#show interfaces fast 0/23 rep detail | include STCN
STCN Propagate to: STP
BPA (STCN, LSL) TLV rx: 0, tx: 0
BPA (STCN, HFL) TLV rx: 0, tx: 1
```

With debugging enabled on C3560, we can see the TCNs being received on every VLAN allowed on the trunk. Because the network is so small, the TC event (with respect to signaling) ends immediately.

```
C3560#debug spanning-tree switch general
Spanning Tree Switch Shim general debugging is on

STP SW: VLAN1: topology change over - this bridge is root
STP SW: VLAN100: topology change over - this bridge is root
STP SW: VLAN101: topology change over - this bridge is root
STP SW: VLAN102: topology change over - this bridge is root
STP SW: VLAN103: topology change over - this bridge is root
```

Checking the STP details for VLAN 100 as an example, we can see the TC flag is still set. This flag remains set for 35 seconds (max age plus forward delay) and effectively flushes the CAM table for this VLAN when the TCN is received. If we check the command after 35 seconds, we see the flag is cleared.

```
C3560#show spanning-tree vlan 100 detail
VLAN0100 is executing the rstp compatible Spanning Tree protocol
Bridge Identifier has priority 0, sysid 100, address 0023.5dc9.5a80
Configured hello time 2, max age 20, forward delay 15, transmit holdcount 6
We are the root of the spanning tree
```

```
Topology change flag set, detected flag not set
Number of topology changes 12 last change occurred 00:00:26 ago
    from GigabitEthernet0/21
Times: hold 1, topology change 35, notification 2
    hello 2, max age 20, forward delay 15
Timers: hello 0, topology change 8, notification 0, aging 300
```

```
C3560#show spanning-tree vlan 100 detail
VLAN0100 is executing the rstp compatible Spanning Tree protocol
Bridge Identifier has priority 0, sysid 100, address 0023.5dc9.5a80
Configured hello time 2, max age 20, forward delay 15, transmit holdcount 6
We are the root of the spanning tree
Topology change flag not set, detected flag not set
Number of topology changes 12 last change occurred 00:00:37 ago
    from GigabitEthernet0/21
Times: hold 1, topology change 35, notification 2
    hello 2, max age 20, forward delay 15
Timers: hello 0, topology change 0, notification 0, aging 300
```

I've also configured a basic SPAN session on C3560 to capture these TCNs as they are flooded from the REP segment onto the C3560 designated ports. I don't need to apply a VLAN filter since I know the REP LSLs aren't being sent to the STP switches on VLAN 1.

```
! C3560
monitor session 1 source interface Gi0/21 , Gi0/23 rx
monitor session 1 destination interface Gi0/22 encapsulation replicate
```

Notice that a single TCN BPDU is sent from the REP primary and secondary edge ports (MAC addresses shown below) towards the STP switch. Even though the TCN propagation was configured on ME3 LACP bond to ME1, the actual TCN BPDU is sourced from the proper uplink interface which is STP-enabled. The exception is VLAN 1; both a PVST and IEEE TCN is sent for this VLAN, which is where the "plus" comes from in PVST+. This means that the STP-aware switch in the network could be running Cisco PVST variations (Cisco multicast MAC destination) or any other vendor that supports the IEEE STP variations without PVST. Even though the REP counters only showed 1 TCN being flooded, that really indicates one REP STCN, which may translate into several STP TCNs if running PVST variants beyond the REP ring. I use a red circle as a marker in the capture below to show the column where the VLAN IDs are shown inside of the TCN BPDUs. In total, we see 12 TCNs, 6 from each REP edge-port.

```
ME3#show interfaces fast 0/21 | include bia
Hardware is Fast Ethernet, address is 001f.9d0b.1697 (bia 001f.9d0b.1697)

ME2#show interfaces fast 0/23 | include bia
Hardware is Fast Ethernet, address is 04c5.a453.d819 (bia 04c5.a453.d819)
```

No.	Time	Source	Destination	Protocol	Length	Info
	65.684160000	00:1f:9d:0b:16:97	01:00:0c:cc:cc:cd	STP	64	RST. TC + Root = 0/1/00
	65.684196000	00:1f:9d:0b:16:97	01:80:c2:00:00:00	STP	60	RST. TC + Root = 0/1/00
	65.684584000	00:1f:9d:0b:16:97	01:00:0c:cc:cc:cd	STP	64	RST. TC + Root = 0/100/0
	65.684586000	00:1f:9d:0b:16:97	01:00:0c:cc:cc:cd	STP	64	RST. TC + Root = 0/101/0
	65.684586000	00:1f:9d:0b:16:97	01:00:0c:cc:cc:cd	STP	64	RST. TC + Root = 0/102/0
	65.684617000	00:1f:9d:0b:16:97	01:00:0c:cc:cc:cd	STP	64	RST. TC + Root = 0/103/0
	65.688174000	04:c5:a4:53:d8:19	01:00:0c:cc:cc:cd	STP	64	RST. TC + Root = 0/1/00
	65.688205000	04:c5:a4:53:d8:19	01:80:c2:00:00:00	STP	60	RST. TC + Root = 0/1/00
	65.688548000	04:c5:a4:53:d8:19	01:00:0c:cc:cc:cd	STP	64	RST. TC + Root = 0/100/0
	65.688549000	04:c5:a4:53:d8:19	01:00:0c:cc:cc:cd	STP	64	RST. TC + Root = 0/101/0
	65.688580000	04:c5:a4:53:d8:19	01:00:0c:cc:cc:cd	STP	64	RST. TC + Root = 0/102/0
	65.688907000	04:c5:a4:53:d8:19	01:00:0c:cc:cc:cd	STP	64	RST. TC + Root = 0/103/0

With the link still broken, C3560 now prefers ME3 for all traffic to ME1 as it is the only available path. Since there is a failure in the REP segment, all ports are opened, including the primary edge port which was recently blocking all VLANs except 100 and 101.

```
C3560#show mac address-table dynamic | include 001d.4692.f74[13]
```

```
100    001d.4692.f741    DYNAMIC    Gi0/21
102    001d.4692.f743    DYNAMIC    Gi0/21
```

```
ME3#show rep topology segment 1
```

```
REP Segment 1
```

```
Warning: REP detects a segment failure, topology may be incomplete
```

BridgeName	PortName	Edge	Role
ME3	Pol3	Sec	Open
ME1	Pol3		Open
ME1	Fa0/15		Fail

Before continuing, we fix the REP segment by bringing ME1 port 15 back up. There are two other kinds of TCN propagation options to discuss, but not test. We looked at the STP option in detail and saw how it worked. The remaining options are shown below.

1. **Segment:** When there are different REP segments on a single switch, this is used to propagate REP STCNs from one segment to a list of other segments. This would support their CAM table flushes for faster layer 2 data-plane convergence. The command example is below, and assuming this was applied to an interface in REP segment 1, would propagate segment 1 STCNs into the segments specified by this command.

```
rep stcn segment 2-5,6,7
```

2. **Interface:** When REP segments are not terminate on the same switch and might be only “one link” away, this command is useful. Rather than use full-blown STP on a P2P link between two switches where that link is not part of any REP segment, this interface can specify the outgoing interface for the REP STCNs to send. Assuming port 7 was a back to back link between two REP-aware switches, this command could be used to exchanged REP STCNs.

```
rep stcn interface fastEthernet 0/7
```

11.3.2 ITU G.8032

Also known as Ethernet Ring Protection Switching (ERPS), this feature works similar to Cisco's REP but is a standards-based variant. The terminology changes from REP but the general concepts remain the same. A segment is called a "ring" and an independent link between two ring nodes is called a "ring link". The ports that run this protocol on ring links are called "ring ports". The alternate (blocked) port is known as the ring protection link, or RPL. The node that is actually blocking a port is known as the "RPL owner node". Just like REP, a single port in the ring will be blocked (the RPL) to prevent loops in the ring topology. In REP, the port adjacent to the alternate port just looks like an ordinary open port, but in G.8032/ERPS, this port is called the RPL neighbor node. The loose equivalent of REP BPA/EPA messages would be the ring automatic protection switching (R-APS) series of messages. These are passed along the ring to communicate RPL information and other significant events. A failure along the ring triggers an R-APG signal failure (R-APS SF) failure, and when the RPL owner receives this message, the port is immediately unblocked. Just like when a failed port exists in a REP topology, the existing alternate port is immediately opened.

Unlike REP, G.8032 relies on existing Ethernet OAM technologies like connectivity fault management (CFM). Switches participating in ERPS must be part of the same CFM domain as a prerequisite to ERPS working; REP does not have this requirement. It is also recommended to configure CFM MEPs; although this is not required, it allows for additional CFM monitoring. ERPS registers to CFM as a client much like protocols register to BFD. In this way, the CFM process ERPS of failures. ERPS uses CFM continuity check messages (CCM) at the fast rate of one packet of 3.3 ms. This helps ERPS achieve SONET-level performance in terms of HA. Because G.8032 is not supported on the ME3400 series, it is not tested in detail here.

12. Describe broadband forum TR-101 VLAN paradigms (N:1 and 1:1)

The entire purpose of the TR-101 document is to detail how to aggregate DSL connections from the access layer into Ethernet versus ATM. The document provides several options, both for general architecture and encapsulations. This section only shows the VLAN paradigms but other notes from the document are rolled into the SP Architecture section.

TR-101 states that the access nodes (AN) must support both of the following VLAN allocation paradigms:

N:1 - Many-to-one mapping between ports and VLAN. This would mean several downstream (customer) facing ports in the same VLAN, likely because they have subscribed to the same service. For example, they all have the same destination service provider gateway device. The VLAN still has its own independent MAC address-table so that its traffic does not mix with other VLANs. The AN also must have a mechanism to prevent communication between user ports (user isolation). In Cisco ME switches, this is automatically enforced between UNI and ENI ports. In Catalyst switches, the Private VLAN (PVLAN) feature can be used to map secondary VLANs to a primary VLAN; in this case, a collection of

secondary community VLANs or a single secondary isolated VLAN would face towards the customers, while the primary VLAN promiscuous port (or the ME switch's NNI port) faces towards the provider.

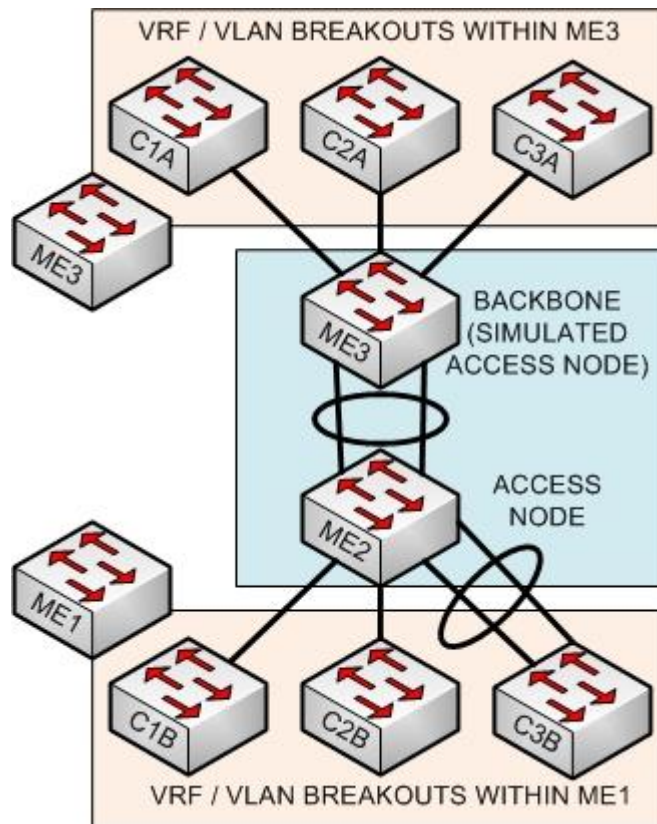
Additional restrictions are as follows:

- * Traffic may be received 802.1Q tagged on the user port
- * Traffic must be S-Tagged for C-Tag transparency within the aggregation network (QinQ)
- * The 802.1Q tag value must be preserved as the C-Tag (QinQ)
- * Access Node must apply an S-Tag. Must have a way to multiplex multiple services whether the received traffic was tagged or not.
- * The S-Tag will be common to a TLS instance if switching is performed within the aggregation network.

1:1 - One-to-one binding between a user port and a VLAN. This basically means one VLAN per customer, where a single access port facing a customer is the only UNI in that VLAN. This paradigm is required to support both single and dual 802.1Q tagged frames and also must have the capability to disable MAC address learning. The document references the Independent VLAN Learning (IVL) model described in 802.1Q. Additional restrictions are as follows:

- * Access Node must apply at least an S-Tag (outer tag)
- * Access Node may apply a C-Tag for scalability reasons (QinQ)
- * S-Tag must not be shared across Access Nodes
- * S-Tag/C-Tag pair must be unique within Access Node

The demonstration uses three hardware Cisco ME-3400-24TS-A switches. There are 3 customers which are simulated by ME1. Each of them has a dot1q trunk to the AN, which is ME2, and because I am using a single customer switch to represent 3 customers, each one uses different C-VLANs. I can still demonstrate QinQ tunneling with this approach, but I cannot demonstrate using the same C-VLANs on a physical switch. Customers 1, 2, and 3 use VLANs 10, 20, and 30 respectively. C1 and C2 are both connected to UNI ports on ME2 and use the same S-TAG of 101 for transport across the aggregation switched network. ME3 serves as a backbone switch which also simulates the customer endpoints inside VRFs using jumper cables; the setup is very sloppy at layer 1 but demonstrates the technology. Below is the logical flow, since the physical cabling is less relevant.



C3 connects to an ENI port so that LACP can run, and this uses S-VLAN 102. Since all the ports are UNI ports of some kind, they cannot talk to one another. The sharing of S-tag 101 by multiple customers C1 and C2 represents the N:1 paradigm while the dedicated S-tag of 102 for C3 represents the 1:1 paradigm. This might make sense if C1 and C2 are subscribing to a similar service and have similar SLA requirements, etc, while C3 may later have a requirement to disable MAC learning (RSPAN VLANs perhaps). The relevant configuration for ME1 is shown below. The configuration is monotonous as we must define VRFs, VLANs, and ports to the AN. Notice that C1 and C2 are regular trunk ports with a single C-VLAN allowed, while C3 uses an LACP bond to accomplish the same thing.

```
! ME1
ip routing
vrf definition C1
  address-family ipv4

vrf definition C2
  address-family ipv4

vrf definition C3
  address-family ipv4

vlan 10
  name C1
```

```

vlan 20
  name C2

vlan 30
  name C3

interface FastEthernet0/13
  description C1
  port-type nni
  switchport trunk allowed vlan 10
  switchport mode trunk
  spanning-tree portfast trunk
  spanning-tree bpdufilter enable

interface FastEthernet0/14
  description C2
  port-type nni
  switchport trunk allowed vlan 20
  switchport mode trunk
  spanning-tree portfast trunk
  spanning-tree bpdufilter enable

interface FastEthernet0/15
  description C3
  port-type nni
  switchport trunk allowed vlan 30
  switchport mode trunk
  channel-protocol lacp
  channel-group 2 mode active

interface FastEthernet0/16
  description C3
  port-type nni
  switchport trunk allowed vlan 30
  switchport mode trunk
  channel-protocol lacp
  channel-group 2 mode active

interface Port-channel2
  description LACP BOND TO ACCESS NODE, C3
  port-type nni
  switchport trunk allowed vlan 30
  switchport mode trunk
  spanning-tree portfast trunk
  spanning-tree bpdufilter enable

interface Vlan10
  vrf forwarding C1
  ip address 10.255.10.1 255.255.255.0

```

```

interface Vlan20
  vrf forwarding C2
  ip address 10.255.20.1 255.255.255.0

interface Vlan30
  vrf forwarding C3
  ip address 10.255.30.1 255.255.255.0

```

ME2 is really the focus of this test. It defines a set of S-VLANs for QinQ tunneling, but I don't name them for brevity. The port-types are actually realistic and there aren't any crazy workarounds. Facing the customer, the ports are UNI for C1 and C2 and ENI for C3. These are all user-to-network interfaces of sorts, so they cannot communicate with one another, which makes sense for customer-facing links. C1 and C2 are tunneled inside S-VLAN 101 while C3 is tunneled inside S-VLAN 102.

```

! ME2
vlan 101-109

interface FastEthernet0/13
  description C1
  switchport access vlan 101
  switchport mode dot1q-tunnel

interface FastEthernet0/14
  description C2
  switchport access vlan 101
  switchport mode dot1q-tunnel

interface FastEthernet0/15
  description C3
  port-type eni
  switchport access vlan 102
  switchport mode dot1q-tunnel
  channel-protocol lacp
  channel-group 2 mode passive

interface FastEthernet0/16
  description C3
  port-type eni
  switchport access vlan 102
  switchport mode dot1q-tunnel
  channel-protocol lacp
  channel-group 2 mode passive

interface Port-channel2
  description LACP BOND TO CUSTOMER C3
  port-type eni
  switchport access vlan 102

```

```
switchport mode dot1q-tunnel
```

For upstream connectivity to ME3, the backbone switch, we can bond the high-speed 1 Gbps interfaces together. These should be NNI ports which participate in STP, and we will enable loop-guard on the LACP bond to prevent against sudden loss of BPDUs from the upstream switch towards the root. Only S-VLANs are allowed over this link and ME2 does not need to account for any of the C-VLANs as they are tunneled inside dot1q.

```
! ME2
interface GigabitEthernet0/1
  description LACP TO GATEWAY
  port-type nni
  switchport trunk allowed vlan 101-109
  switchport mode trunk
  channel-protocol lacp
  channel-group 1 mode active

interface GigabitEthernet0/2
  description LACP TO GATEWAY
  port-type nni
  switchport trunk allowed vlan 101-109
  switchport mode trunk
  channel-protocol lacp
  channel-group 1 mode active

interface Port-channel1
  description LACP TO GATEWAY
  port-type nni
  switchport trunk allowed vlan 101-109
  switchport mode trunk
  spanning-tree guard loop
```

ME3 is the other end of the LACP gateway bond and allows the S-VLANs in. It is the root for all S-VLANs, uses root-guard on the port-channel to ensure the downstream designated port never becomes an upstream root port.

```
! ME3
spanning-tree vlan 1-4094 priority 8192

interface GigabitEthernet0/1
  description LACP TO ACCESS NODE
  port-type nni
  switchport trunk allowed vlan 101-109
  switchport mode trunk
  channel-protocol lacp
  channel-group 1 mode passive
```

```

interface GigabitEthernet0/2
  description LACP TO ACCESS NODE
  port-type nni
  switchport trunk allowed vlan 101-109
  switchport mode trunk
  channel-protocol lacp
  channel-group 1 mode passive

interface Port-channel1
  description LACP TO ACCESS NODE
  port-type nni
  switchport trunk allowed vlan 101-109
  switchport mode trunk
  spanning-tree guard root

```

In real life, this traffic would be MPLS-encapsulated and transported to the other end of the network once it hits the network core. In this case, I configure another dot1q-tunnel port to remove the outermost encapsulation from the frames. This is because ME3 is going to respond to the traffic locally for testing purposes. ME3 defines the same VRFs and C-VLANs as ME1 so that the two can communicate. Ports 21/22 and 23/24 are jumped together; ports 22 and 24 are the correctly-configured dot1q-tunnel ports at the remote end of the customer circuit, which removes the S-tag. Ports 21 and 23 simulate the customer switch by processing the customer VLANs. I used two ports because I had two separate S-tags in this test, so S-tag 101 carries VLANs 10 and 20 while S-tag 102 only carried VLAN 30. This is for testing purposes only given a lack of hardware.

```

! ME3
ip routing
vrf definition C1
  address-family ipv4

vrf definition C2
  address-family ipv4

vrf definition C3
  address-family ipv4

vlan 10
  name C1

vlan 20
  name C2

vlan 30
  name C3

interface FastEthernet0/21
  description LINK TO ACCESS NODE (JUMPER 102)

```

```

port-type nni
switchport trunk allowed vlan 30
switchport mode trunk
spanning-tree portfast trunk

interface FastEthernet0/22
description LINK TO CUSTOMER (JUMPER 102)
switchport access vlan 102
switchport mode dot1q-tunnel

interface FastEthernet0/23
description LINK TO ACCESS NODE (JUMPER 101)
port-type nni
switchport trunk allowed vlan 10,20
switchport mode trunk
spanning-tree portfast trunk

interface FastEthernet0/24
description LINK TO CUSTOMER (JUMPER 101)
switchport access vlan 101
switchport mode dot1q-tunnel

```

Before we start testing, we will capture the MAC addresses of the SVIs on ME1 and ME3. These are not configurable, or else I would have configured easily recognizable values. I will always highlight ME1's MACs in yellow with ME3's MACs in green for clarity.

```

ME1#show interfaces | include ^Vlan[123]0|EtherSVI
  Hardware is EtherSVI, address is 001d.4692.f740 (bia 001d.4692.f740)
Vlan10 is up, line protocol is up
  Hardware is EtherSVI, address is 001d.4692.f741 (bia 001d.4692.f741)
Vlan20 is up, line protocol is up
  Hardware is EtherSVI, address is 001d.4692.f742 (bia 001d.4692.f742)
Vlan30 is up, line protocol is up
  Hardware is EtherSVI, address is 001d.4692.f743 (bia 001d.4692.f743)

ME3#show interfaces | include ^Vlan.0|EtherSVI
  Hardware is EtherSVI, address is 001f.9d0b.16c0 (bia 001f.9d0b.16c0)
Vlan10 is up, line protocol is up
  Hardware is EtherSVI, address is 001f.9d0b.16c1 (bia 001f.9d0b.16c1)
Vlan20 is up, line protocol is up
  Hardware is EtherSVI, address is 001f.9d0b.16c2 (bia 001f.9d0b.16c2)
Vlan30 is up, line protocol is up
  Hardware is EtherSVI, address is 001f.9d0b.16c3 (bia 001f.9d0b.16c3)

```

First, we ping from ME1 to ME3 within VRF C1. This represents end-to-end customer connectivity inside C-VLAN 10, which was encapsulated inside S-VLAN 101 for transport between ME2 and ME3. Since the ping succeeds, we assume everything is working, and begin verification.

```
ME1#ping vrf C1 10.255.10.3
```

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 10.255.10.3, timeout is 2 seconds:

!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 1/5/9 ms

When ME1 sends traffic to ME2, it does so on port 13 which is forwarding per STP. There is no BPDU exchange between customer and AN as I've filtered BPDUs on the customer side, and the UNI ports never run STP. ENI ports support STP but it is disabled by default; I do not enable it anywhere in this test. ME1 has a CAM entry for ME3's MAC address inside the C-VLAN out of this port.

```
ME1#show spanning-tree vlan 10 interface fa0/13
```

Vlan	Role	Sts	Cost	Prio.Nbr	Type

VLAN0010	Desg	FWD	19	128.15	P2p Edge

```
ME1#show mac address-table dynamic vlan 10
```

Mac Address Table

Vlan	Mac Address	Type	Ports
10	001f.9d0b.16c1	DYNAMIC	Fa0/13

Looking at ME2's CAM for the S-VLAN, we can see this MAC address being identified as learned in VLAN 101. Even though port 13 is not really in VLAN 101, that is the S-tag for that port, so the CAM entry counts for that VLAN. This makes sense since when frames are received for this MAC, ME2 should forward them out port 13, but also remove the topmost dot1q tag. Traffic towards ME3 is forwarding out of the LACP bond towards the gateway switch, which is correct.

```
ME2#show mac address-table dynamic vlan 101
```

Mac Address Table

Vlan	Mac Address	Type	Ports
101	001d.4692.f70f	DYNAMIC	Fa0/13
101	001d.4692.f710	DYNAMIC	Fa0/14
101	001d.4692.f741	DYNAMIC	Fa0/13
101	001f.9d0b.1681	DYNAMIC	Po1
101	001f.9d0b.16c1	DYNAMIC	Po1

When ME3 receives these frames, they arrive on the high-speed Gigabit Etherchannel (GEC) from ME2. The VLAN is 101 since the frames are still double-tagged. The destination MAC address of the frames from ME1 matches the second CAM entry, which is forward out port 24.

```
ME3#show mac address-table dynamic vlan 101
```

Mac Address Table			
Vlan	Mac Address	Type	Ports
101	001d.4692.f741	DYNAMIC	Po1
101	001f.9d0b.16c1	DYNAMIC	Fa0/24

Since ME3 is also the test endpoint, it will see the C-VLAN on its jumped interface. Since ports 23 and 24 are connected, it sees the C-VLAN tag of 10 on port 23.

```
ME3#show mac address-table dynamic vlan 10
Mac Address Table
-----
Vlan    Mac Address      Type      Ports
----    -
10      001d.4692.f741   DYNAMIC   Fa0/23
Total Mac Addresses for this criterion: 1
```

We won't trace the process in the reverse direction or for the other C-VLANs since it is exactly the same. We quickly test that ME3 has reachability to ME1 for the other C-VLANs to ensure they work.

```
ME3#ping vrf C2 10.255.20.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.255.20.1, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/4/9 ms
```

```
ME3#ping vrf C3 10.255.30.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.255.30.1, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/5/9 ms
```

The downside to QinQ in this way is that the switches unaware of the C-VLAN tags, such as ME2, will continue to increase the size of their CAM tables since the C-MACs are still exposed. VLAN 101 shows 2 C-MACs per site and VLAN 102 shows 1 C-MAC per site. This can be resolved with provider backbone bridges (PBB), which are discussed in another chapter.

```
ME2#show mac address-table dynamic vlan 101
Mac Address Table
-----
Vlan    Mac Address      Type      Ports
----    -
101     001d.4692.f70f   DYNAMIC   Fa0/13
101     001d.4692.f710   DYNAMIC   Fa0/14
101     001d.4692.f741   DYNAMIC   Fa0/13
101     001d.4692.f742   DYNAMIC   Fa0/14
```



```

101    001f.9d0b.1681    DYNAMIC    Po1
101    001f.9d0b.16c1    DYNAMIC    Po1
101    001f.9d0b.16c2    DYNAMIC    Po1
Total Mac Addresses for this criterion: 7

```

```
ME2#show mac address-table dynamic vlan 102
```

Mac Address Table

```

-----
Vlan    Mac Address          Type        Ports
----    -
102     001d.4692.f711      DYNAMIC     Po2
102     001d.4692.f712      DYNAMIC     Po2
102     001d.4692.f743      DYNAMIC     Po2
102     001f.9d0b.1681      DYNAMIC     Po1
102     001f.9d0b.16c3      DYNAMIC     Po1

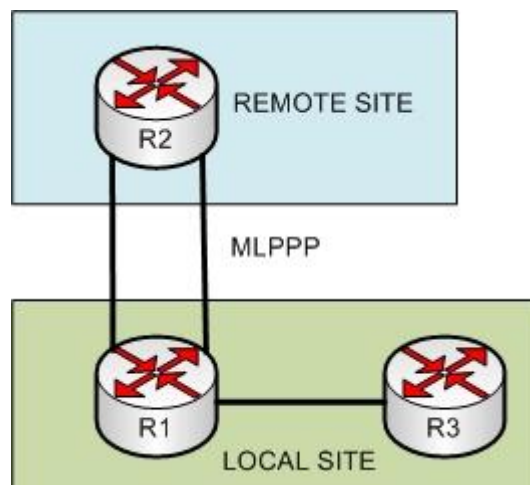
```

```
Total Mac Addresses for this criterion: 5
```

Additional Reading – Reference configurations “tr101”

13. Describe QoS link fragmentation (LFI), cRTP, and RTP

QoS link fragmentation and interleaving (LFI) is a common technique for slow WAN links to reduce latency for priority traffic. Large data packets, such as seen in an FTP download, take longer to serialize over slow WAN links than small voice packets do. The small voice packets are latency-sensitive while the FTP packets are not. PPP specifically has a mechanism to deal with this that is built into multi-link PPP (MLP or MLPPP). Because multi-link bundles aren’t supported with PPPoE, we will test this using a small GNS3 topology with Cisco 3725 images running 12.4(15)T. R1 is the customer site router with the 512 kbps WAN connection, while R2 is the “Internet” simulator and R3 is a LAN client.



The basic MLP configurations are shown below. R3 is also an NTP server and IP SLA responder which will help when approximating VOIP performance. R1 and R2 share many common configuration parameters mostly related to basic PPP/MLP configuration. The differences are shown below as well, since R2 issues

IP addresses to R1 via IPCP from a local pool. R1 installs a default route back to those addresses, and CSR2 uses a simple static route for customer reachability. R3 has an IP address and default route, which is so basic that it is not shown.

```
! R1 and R2 (common)
interface Serial0/0
  encapsulation ppp
  ppp multilink
  ppp multilink group 2

interface Serial0/1
  encapsulation ppp
  ppp multilink
  ppp multilink group 2

interface Multilink2
  bandwidth 512
  fair-queue 64 256 0
  ppp multilink
  ppp multilink group 2
  max-reserved-bandwidth 100

! R1
interface Multilink2
  ip address negotiated
  no peer default ip address
  ppp ipcp route default

! R2
ip local pool PPPOE_IPV4 10.1.2.100 10.1.2.200

interface Loopback0
  ip address 10.1.2.2 255.255.255.0

interface Multilink2
  ip unnumbered Loopback0
  peer default ip address pool PPPOE_IPV4

ip route 10.0.0.0 255.0.0.0 Multilink2
```

We will conduct basic MLP checks and ensure we have reachability from R2 to R3. R1 shows the multilink bundle as operational with Serial0/0 and Serial0/1 as member interfaces. We can clearly see the local/remote endpoints (hostnames for clarity). We also confirm reachability between the Internet and the LAN client.

```
R1#show ppp multilink active
Multilink2
  Bundle name: R2
```

```

Remote Endpoint Discriminator: [1] R2
Local Endpoint Discriminator: [1] R1
Bundle up for 00:26:14, total bandwidth 3088, load 1/255
Receive buffer limit 24000 bytes, frag timeout 1000 ms
  0/0 fragments/bytes in reassembly list
  0 lost fragments, 1 reordered
  0/0 discarded fragments/bytes, 0 lost received
  0x5D received sequence, 0x5A sent sequence
Member links: 2 active, 0 inactive (max not set, min not set)
  Se0/0, since 00:26:14
  Se0/1, since 00:26:14

```

```

R2#ping 10.1.3.3
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.1.3.3, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 28/45/104 ms

```

Since this is a simulation, it will be hard to measure real latency on this slow link, but we will work out all the math anyway. First, we must define what a “slow” WAN link is. The formula for determining serialization delay is:

$$\text{Delay (sec)} = \text{size (bits)} / \text{rate (bits/sec)}$$

As an example, a 1500 byte packet (12000 bits) going over a T1 link (1544000 bps) will take about ~8ms (0.008 seconds) to serialize. In our case, since we are using a 512 kbps link, that same sized packet will take about 3 times longer to serialize since the link is about one third as fast. Specifically, $12000 / 512000 \approx 23.4\text{ms}$. While this might not seem like a long time, imagine a tiny VOIP packet stuck behind this giant 1500 packet in the TX ring. It has to wait a relatively long time before it can be serialized.

An effective VOIP SLA will meet the following requirements as recommended by Cisco (and others):

1. Packet loss should be no more than 1 percent.
2. One-way latency (mouth to ear) should be no more than 150 ms.
3. Average one-way jitter should be less than 30 ms.

With stringent requirements like this, VOIP latency must be minimized. A general recommendation for per-link VOIP latency is 10 ms of serialization delay. This is forgivable up to 20 ms per link but is not desirable, especially for long-distance calls where there are many transit links in the path. Incurring more than that will make the cumulative 150 ms target difficult to achieve. That begin said, we can use the combination of the 10 ms per-link target and the serialization delay formula to determine what a “slow” link is. In essence, we want to answer the question “What is the slowest speed that can serialize a 1500 byte packet in 10ms or less”? If a link is faster than this number, many of the techniques discussed in this section are moot, since that “huge” 1500 byte packet serializes very quickly on fast links. To evaluate it, we use a variant of the formula above: **rate = speed * delay = 1.2 Mbps. Despite**

this value, it is commonly understood that 768 kbps (half T1) is the cutoff for a slow link.

Mathematically, a 768 kbps link would take ~15.6 ms to serialize a 1500 packet, this is considered acceptable when combined with LLQ and header compression. Since our link is 512 kbps, we assume this is a slow link.

The first tool to deal with this is an intelligent QoS policy. QoS is covered in detail in another section, but the basic idea is to treat VOIP traffic as low latency by guaranteeing bandwidth for it. In addition, low latency queuing (LLQ) also ensures that this traffic is serviced before any other queue, regardless of their class weights. This makes it a fitting solution for VOIP and other latency-sensitive flows. A sample policy is configured on R1 and attached to the MLPPP interface. For simplicity, we match the first 2 bits of the TOS byte to look for a binary value of 101; this will match IPP5, DSCP CS5, DSCP EF, and any value beginning with 101, which is generally low-latency traffic. Next, a shaper wraps this queue to limit it to 512 kbps as an average rate. The mathematics behind the priority value and shaper are described next.

```
! R1
class-map match-all CMAP_VOICE
  match precedence 5

policy-map PMAP_QUEUE
  class CMAP_VOICE
    priority 45
  class class-default
    fair-queue
    random-detect dscp-based

policy-map PMAP_SHAPE
  class class-default
    shape average 512000 5120 2560
    service-policy PMAP_QUEUE

interface Multilink2
  service-policy output PMAP_SHAPE
```

First, we analyze the value of 45 kbps for the voice reservation. This policy was built on the assumption that exactly two concurrent G.729 VOIP phone calls needed to be supported on this link. We will assume (and test via IP SLA) that the G.729 calls will be 20 byte payloads at a rate of 50 packets/second. The full packet size is much larger when we add all of the overhead: 20 payload + 12 RTP + 8 UDP + 20 IP + 8 PPP = 68 bytes total. 50 packets/second is the rate, and multiplying this by 68 yields 3400 Bytes/second. Converting to bits, this is about 27.2 kbps per phone call. 27.2 kbps multiplied by 2 is 44.4 kbps, so reserving 45 kbps for the LLQ will cover both calls. Recall that Cisco queuing strategies always account for layer 2 overhead, as well as all overhead from upper layer protocols, which is why we must include the 8 bytes of PPP encapsulation.

Next, we examine the shaper configuration. We will assume that the SP will let us burst up to 768 kbps within a single time interval if we have accumulated credit (achieved by not sending at CIR rate in

previous intervals). Since we want to achieve a 10 ms time interval to ensure the shaper “runs” and sends data every 10 ms, we must select a burst committed (Bc) value that works. The relationships between Tc, Bc, Be, CIR, and PIR are shown below.

$$Tc = Bc / CIR$$

$$Tc = (Bc + Be) / PIR$$

First, we solve for Bc using the first formula. Tc is set to 0.01 seconds (10 ms) and CIR is set to 512000 bits/second. Evaluation: $0.01 = Bc / 768000 \rightarrow 0.01 * 512000 = Bc = 5120 \text{ bits}$. Next, we solve for Be using the second formula. This will be used to determine how many bits can we accumulate and send during periods of burst; this involves the PIR which is 768 kbps. Evaluation: $0.01 = (5120 + Be) / 768000 \rightarrow 0.01 * 768000 = 5120 + Be \rightarrow 7860 = 5120 + Be = 2560 \text{ bits}$. We can show the result of our math using the show command below. We can see Tc was evaluated to be 10 ms based on the Bc we computed. The burst excess (Be) is 2560 bits but the PIR of 768 kbps is not explicitly shown.

```
R1#show policy-map interface multilink 2 output
Multilink2
```

```
Service-policy output: PMAP_SHAPE
```

```
Class-map: class-default (match-any)
```

```
139 packets, 25548 bytes
```

```
5 minute offered rate 0 bps, drop rate 0 bps
```

```
Match: any
```

```
Traffic Shaping
```

Target/Average		Byte	Sustain	Excess	Interval	Increment
Rate		Limit	bits/int	bits/int	(ms)	(bytes)
512000/512000		960	5120	2560	10	640
Adapt	Queue	Packets	Bytes	Packets	Bytes	Shaping
Active	Depth			Delayed	Delayed	Active
-	0	139	25548	0	0	no

Although testing the shaper will be difficult, we can ensure G.729 VOIP traffic is matching the voice queue we configured. We can achieve this using IP SLA on R3. We specify a 20 byte packet payload using G.729A (the ‘A’ doesn’t affect the bandwidth and relates only to payload processing) with a TOS of 160, which is IPP5. The codec-interval is 20 ms (1000ms / 50 pps = 20 ms) but that is the default and not shown in the configuration, but I show it below for completeness. Note that the codec-size of 32 is also the default, but since this isn’t a real RTP stream, we must manually account for the 12 RTP bytes. The payload of a real call would be 20 bytes with 12 RTP bytes, but in this case, the payload is 32 bytes with only an additional 8-byte UDP header.

```
! R3
ip sla 3
```

```

udp-jitter 10.1.2.2 16384 source-port 16386 codec g729a codec-size 32 codec-
interval 20
tos 160
timeout 10000
ip sla schedule 3 life forever start-time now

```

Checking R3, we can see the IP SLA operation is successful and we resolve both latency and jitter measurements. As seen later in the IP SLA section, this means that NTP is operational, as is the dynamic IP SLA responder on R2.

```

R3#show ip sla statistics 3
Round Trip Time (RTT) for          Index 3
    Latest RTT: 57 milliseconds
Latest operation start time: 01:21:57.612 UTC Fri Mar 1 2002
Latest operation return code: OK
RTT Values:
    Number Of RTT: 79                RTT Min/Avg/Max: 14/57/101
milliseconds
Latency one-way time:
    Number of Latency one-way Samples: 34
    Source to Destination Latency one way Min/Avg/Max: 3/15/42
milliseconds
    Destination to Source Latency one way Min/Avg/Max: 11/47/82
milliseconds
Jitter Time:
    Number of SD Jitter Samples: 77
    Number of DS Jitter Samples: 77
    Source to Destination Jitter Min/Avg/Max: 0/20/84 milliseconds
    Destination to Source Jitter Min/Avg/Max: 0/18/87 milliseconds
Packet Loss Values:
    Loss Source to Destination: 0          Loss Destination to Source: 0
    Out Of Sequence: 0          Tail Drop: 918
    Packet Late Arrival: 3      Packet Skipped: 0
Voice Score Values:
    Calculated Planning Impairment Factor (ICPIF): 11
MOS score: 4.06
Number of successes: 3
Number of failures: 0
Operation time to live: Forever

```

We also check the policy-map on R2 to ensure the VOIP is being classified properly. The reason it only shows as a 2 kbps flow is that the probe is not continuous and runs once every 60 seconds so my computer doesn't crash.

```

R1#show policy-map interface multilink 2 output | section VOICE
Class-map: CMAP_VOICE (match-all)
    4368 packets, 218560 bytes
    5 minute offered rate 2000 bps, drop rate 0 bps

```

```

Match: precedence 5
Queueing
  Strict Priority
  Output Queue: Conversation 40
  Bandwidth 45 (kbps) Burst 1125 (Bytes)
  (pkts matched/bytes matched) 0/0
  (total drops/bytes drops) 0/0

```

To further optimize the WAN for our slow 512 kbps link, we can enable LFI. Because large 1500 byte packets will take about 24 ms to serialize, we won't want our VOIP traffic to get stuck behind it. LFI has two main components: fragmentation and interleaving. Fragmentation is often times viewed negatively because it is usually discussed in the context of layer 3 protocols, like IP. IP fragmentation occurs end to end (unless virtual reassembly occurs in the network, which is CPU intensive) but layer 2 fragmentation is local to a link or potentially an entire layer-2 domain. MLPPP can use this to break up large packets into smaller fragments. For example, a 1500 byte packet broken into three 500-byte packets still requires 24 ms to serialize everything, but individually, they take about 8 ms. That is well within the latency range for VOIP traffic on a link, and assuming we can break up large packets, we can squeeze VOIP packets in between them. This is interleaving and is often enabled in conjunction with fragmentation with MLP. **In summary, we can fragment large data packets and interleave small VOIP packets in between them to achieve VOIP SLAs.** The downside of this approach is additional CPU resources to perform the fragmentation/reassembly and a small amount of additional overhead added to each fragment.

Before enabling fragmentation, we need to determine the fragment size. This goes back to the shaper math and the serialization delay, and should be coordinated with other QoS features appropriately. We want to maintain a 10 ms serialization delay on this link for VOIP, so we can extend that logic to ALL packets on the link. Thus, we must find out what is the largest packet we can serialize on a 512 kbps link in 10ms. Evaluation: $0.01 = \text{size} / 512000 \rightarrow 0x01 * 512000 = \text{size} = 5120 \text{ bits}$. **Converting this to bytes, we get 640 bytes.** This sounds about right, since this is a little over one third the size of a 1500 byte packet which took ~24 ms to serialize. PPP can break packets up into fragments of 640 regardless of how large they are, which guarantees that every single packet will take no more than 10 ms to serialize. VOIP packets are only 68 bytes, so those will serialize about 10 times faster at 1 ms. The goal was to ensure that a VOIP packet never gets stuck waiting for more than 10 seconds, so we have to fragment the big packets to meet this goal. Combined with the fragmentation, the shaper running every 10 ms ensures that VOIP packets do not wait long before being serialized. Configuring this on R1 and R2 is simple.

```

! R1 and R2
interface Multilink2
  ppp multilink fragment size 640

```

We can verify this took affect for all member interfaces by checking the MLP details.

```
R1#show ppp multilink active
```

```

Multilink2
  Bundle name: R2
  Remote Endpoint Discriminator: [1] R2
  Local Endpoint Discriminator: [1] R1
  Bundle up for 01:12:44, total bandwidth 3088, load 1/255
  Receive buffer limit 24000 bytes, frag timeout 1000 ms
    0/0 fragments/bytes in reassembly list
    0 lost fragments, 227 reordered
    0/0 discarded fragments/bytes, 0 lost received
    0x4739 received sequence, 0x476B sent sequence
  Member links: 2 active, 0 inactive (max not set, min not set)
    Se0/0, since 01:12:44, 5790 weight, 640 frag size
    Se0/1, since 01:12:44, 5790 weight, 640 frag size

```

MLP is very smart in that it can use multiple physical links in the bundle to serialize the fragments, further reducing latency. We can see this in the debug on R1. R3 sends a single 1000-byte packet, and R1 breaks it up. The fragment of size 648 is the largest packet that can be sent (640 + 8 PPP overhead) which takes a tiny bit longer than 10 ms to serialize. We could account for these 8 bytes by reducing the fragment size to 632, but it's not significant. We can see the fragment sequence numbers increase as well to show fragments as part of the same packet on a per-interface basis.

```

R3#ping 10.1.2.2 size 1000 repeat 1
Type escape sequence to abort.
Sending 1, 1000-byte ICMP Echos to 10.1.2.2, timeout is 2 seconds:
!
Success rate is 100 percent (1/1), round-trip min/avg/max = 64/64/64 ms

```

```

R1#debug ppp multilink fragments
Multilink fragments debugging is on
Se0/0 MLP: O frag 80005212 size 648 encsize 4
Se0/1 MLP: O frag 40005213 size 370 encsize 4
Se0/1 MLP: I frag 400051D7 size 370 encsize 4
Se0/0 MLP: I frag 800051D6 size 648 encsize 4

```

We have not yet enabled interleaving. This is a simple command that enables VOIP packets to be inserted in between other packets. The way PPP knows about packet priorities is using fair-queuing, which MUST be enabled on the multilink bundle for interleaving to work. We configured it earlier since it is off by default on multilink interfaces. Despite having a custom queuing policy on the multilink, we still need the “fair-queue” command, at least in older versions.

```

! R1 and R2
interface Multilink2
  ppp multilink interleave

```


We can verify the configuration was successful by checking the MLP details again. We can see that interleaving is now enabled, which allows the VOIP packets to be woven in between the other packets, assuming they have TOS set appropriately.

```
R1#show ppp multilink active
Multilink2
  Bundle name: R2
  Remote Endpoint Discriminator: [1] R2
  Local Endpoint Discriminator: [1] R1
  Bundle up for 01:24:18, total bandwidth 3088, load 1/255
  Receive buffer limit 24000 bytes, frag timeout 1000 ms
  Interleaving enabled
    0/0 fragments/bytes in reassembly list
    0 lost fragments, 285 reordered
    0/0 discarded fragments/bytes, 0 lost received
    0x562D received sequence, 0x566D sent sequence
  Member links: 2 active, 0 inactive (max not set, min not set)
    Se0/0, since 01:24:18, 5790 weight, 640 frag size
    Se0/1, since 01:24:18, 5790 weight, 640 frag size
```

The final technique is using header compression for RTP. Although this technique throws off some of the math we did earlier, specifically with respect to the LLQ, it isn't a big deal in most cases. First, we will discuss the basics of the RTP header. The three main components of the header, ignoring minor fields like version/flags, are the timestamp, sequence number, and synchronization source (SSRC). The SSRC is a way to identify a source, like an IP address would, but is seldom used in any meaningful way. It could be used to find a looped audio source, potentially. Of greater importance are the timestamps and sequence numbers. This is a way to determine if packets are dropped and to compute delay/latency so that software can attempt to make corrections as necessary.

Before configuring this feature, we will capture the original size of the packets on the MLPPP link. If we debug PPP fragments before header compression, we can see a ton of output for each of the VOIP packets generated by the IP SLA probe on CSR3. I recommend sending this to the buffer. Each fragment is shown as 58 bytes, but this is after the packet is split in half and sent down each MLPPP link. Don't fret on the number 58, just remember it for now.

```
! R1
Se0/0 MLP: I frag C0005018 size 58 encsize 4
Se0/0 MLP: I frag C000501A size 58 encsize 4
Se0/1 MLP: O frag C0005053 size 58 encsize 4
Se0/0 MLP: O frag C0005054 size 58 encsize 4
```

There are two formats available: IETF and IHPC, the latter of which is the default. Since we only expect two phone calls, we can also limit the header compression to conserve resources.

```
! R1 and R2
```

```
interface Multilink2
 ip rtp header-compression iphc-format
 ip rtp compression-connections 2
```

We can confirm the feature is enabled by checking the compression details. We can see most of the packets being compressed with a high hit ratio. Since the packets are small to begin with, they don't compress terribly well, but it does save several overhead bytes. Real G.729 calls would compress very poorly as the codec has already heavily compressed the voice to minimize the bandwidth requirements. It also includes the IP and UDP headers as well, which is shown in the header of the output for clarity. Generally the 40 bytes of overhead (20 IP + 8 UDP + 12 RTP) is compressed to approximately 5 bytes.

```
R1#show ip rtp header-compression multilink 2
RTP/UDP/IP header compression statistics:
Interface Multilink2 (compression on, IPHC, RTP)
  Rcvd: 2009 total, 1977 compressed, 0 errors, 0 status msgs
        0 dropped, 0 buffer copies, 0 buffer failures
  Sent: 2009 total, 1979 compressed, 0 status msgs, 1979 not predicted
        41559 bytes saved, 79157 bytes sent
        1.52 efficiency improvement factor
Connect: 2 rx slots, 2 tx slots,
          9 misses, 0 collisions, 0 negative cache hits, 2 free contexts
          99% hit ratio, five minute miss rate 0 misses/sec, 0 max
```

If we debug PPP fragments again, we can see the fragment size is now 49, which is 9 bytes smaller than before. This is a good indication that RTP header compression did a little bit for us.

```
! R2
Se0/1 MLP: I frag C0002B80 size 49 encsize 4
Se0/1 MLP: O frag C0002B84 size 49 encsize 4
Se0/0 MLP: O frag C0002B85 size 49 encsize 4
Se0/1 MLP: I frag C0002B82 size 49 encsize 4
```

Additional Reading - Reference configurations "lfi-crtip"

14. Describe Multichassis/Clustering High Availability (HA)

The ASR9000 supports many HA mechanisms. Many of these mechanisms work together during RSP/RP failure events, control-plane failure events, and other faults that could cause network outages.

SSO: Stateful switchover. The RSP/RP cards are deployed in an active/standby fashion and SSO allows the state from the active RSP/RP to be written to the backup one so that it can immediately take over if the primary one fails. SSO is complementary (and a prerequisite) for NSF/GR and NSR technologies discussed next. It synchronizes the device configuration and other critical non-routing, non-forwarding related information.

NSF: Non-stop forwarding. During an SSO event, NSF ensures that despite a brief loss of the control plane due to an RSP/RP failure, traffic can still be forwarded. The line-cards still have FIBs and can forward traffic; functions like ARP, BFD, ACL, and QoS can be handled by line-cards as well, so the impact on forwarding transit (non-exception) traffic is minimal. NSF also takes affect when soft reboots of certain software modules occurs, provided the line-cards are not affected. NSF relies on active communication with peers who “help” a failed router recover using a set of special messages that vary per protocol. NSF is also called “Graceful Restart” (GR) for some protocols, and the terms NSF and GR are interchangeable. NSF is available with almost all routing or routing-related technologies, except for RIP.

NSR: Non-stop routing. Like NSF, this maintains the routing protocol information during an SSO event. BGP sessions can remain intact to continue supporting MPLS L3VPN services on a PE router, for example. Unlike NSF/GR, NSR occurs locally on a router using a checkpoint mechanism to synchronize information between RSPs/RPs. It can be used when NSF is not supported on all devices in the network. The peer devices won’t know that anything happened on the device sustaining a failure at all. Neighbor sessions and all pertinent information are migrated between RSP/RP cards when an SSO event occurs without notifying the peer, which is different than NSF. NSR is available with OSPFv2/v3, IS-IS, BGP, LDP routing technologies currently. RVSP-TE, EIGRP, and RIP are not included.

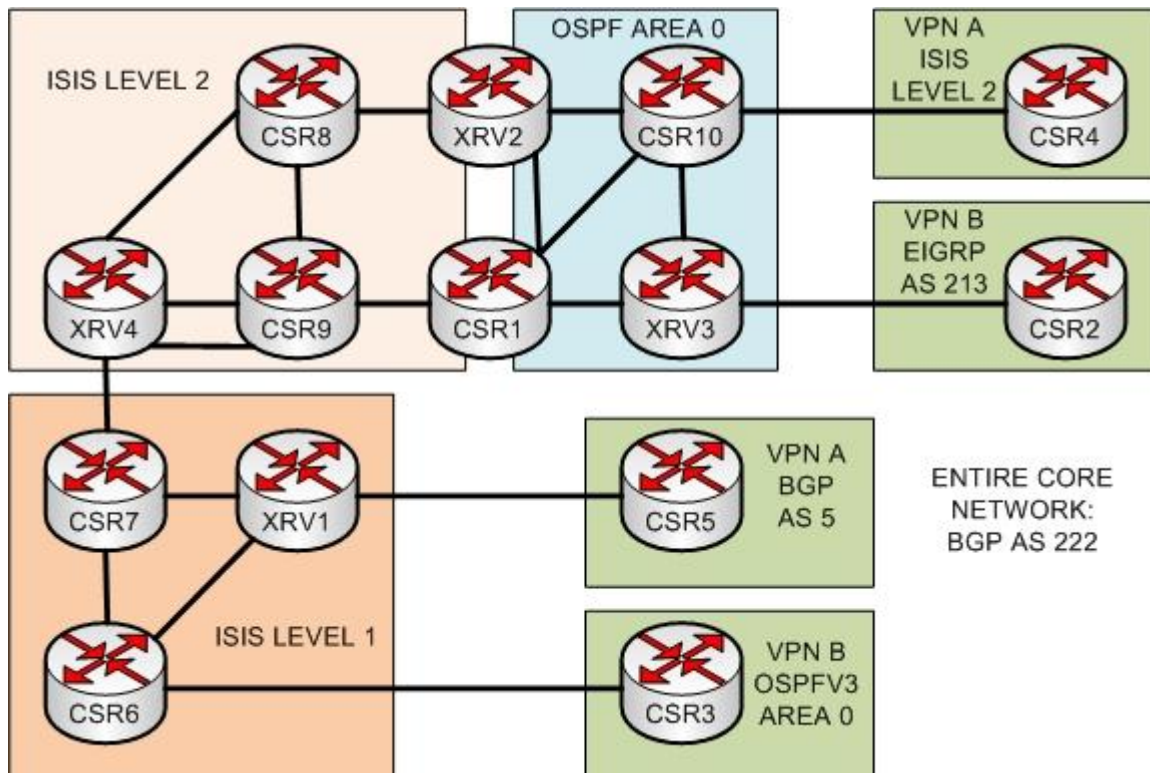
“Process restartability” isn’t so much of a feature as it is a capability. If a process has failed, it will not affect any other process, and can freely be stopped, started, or restarted at any time within IOS XR. Each process runs in a protected address space, and often times a failed process will restart itself (can be controlled manually as well).

The ASR9000 can be clustered together to create a single logical router. This negates the need for complex protocol-based HA schemes since there is only a single router with increased capabilities. The RSPs can be connected directly together using Ethernet Out-of-Band Channel (EOBC) interfaces on the RSP; each RSP has two of these. The idea is that between two routers with two RSPs each, each port will connect to a remote RSP on the other router, creating a full mesh of links between RSPs across the two platforms. This uses the same nV technology used for satellite operations on the ASR9000 and is known as “nV Edge”. These links are used for control information and not for data traffic, which is why high speed interfaces on the line cards must also be identified and connected. These are known as Inter Rack Links (IRLs) which must be at least 10 GbE and be direct layer 1 connections to the remote routers line cards. A minimum of 2 IRLs is required and preferably are connected between different pairs of line cards on each router for extra resiliency. Below are some common failover scenarios:

1. Active RSP failures on the primary node. The backup RSP on the same node detects the failure through the backplane, and becomes the active RSP for the system as a whole. The primary node then informs the backup node of the change via inter-chassis control link messaging.
2. Both the active and backup RSPs on the primary node fail (entire router lost power, for example). The failed router is detected by the UDLD running on the IRLs from the backup node. The backup node then becomes the primary node.

14.1 High Availability (HA) Demonstration (NSF/NSR/GR)

Although many of these features don't make much sense on virtual platforms, the configurations are supported in the parser and some of the feature behaviors are supported. These labs quickly examine the HA configurations for all supported protocols. Recall that GR and NSF are interchangeable terms, and the chapter headers below will use whichever feature name is supported by the parser. For example, in OSPFv2, the command "nsf" is used, while OSPFv3 uses the command "graceful-restart". All of these features share a large unified MPLS topology so that they can be tested concurrently. The network diagram is shown below.



First, I quickly skim the ISIS LSPDB and OSPF LSDB within the IGP. Not all configurations are examined since there are dedicated sections for these features. The goal is to ensure the topology shows all of the proper routers and links at a quick glance. Looking at the ABRs, we can see the entire topology quickly. CSR7 shows the output for the aggregation and core IGPs, while XRV2 shows just its aggregation IGP.

```
R7#show isis database level-2 detail | include ^[RX]|Extended
R1.00-00          0x0000000D  0xAE88          762          0/0/0
  Metric: 10      IS-Extended R9.00
R7.00-00          * 0x00000013  0x04B2          477          0/0/0
  Metric: 10      IS-Extended XRV4.00
R8.00-00          0x0000000F  0x9427          1031         0/0/0
  Metric: 10      IS-Extended R9.00
  Metric: 10      IS-Extended XRV4.00
```

```

Metric: 10          IS-Extended XRv2.00
R9.00-00           0x00000011    0x83DF          530          0/0/0
Metric: 10          IS-Extended R8.00
Metric: 10          IS-Extended R1.00
Metric: 10          IS-Extended XRv4.00
Metric: 10          IS-Extended XRv4.00
XRv2.00-00         0x00000008    0x90DE          502          0/0/0
Metric: 10          IS-Extended R8.00
XRv4.00-00         0x0000000B    0x171F          870          0/0/0
Metric: 10          IS-Extended R8.00
Metric: 10          IS-Extended R7.00
Metric: 10          IS-Extended R9.00
Metric: 10          IS-Extended R9.00

```

```
R7#show isis database level-1 detail | include ^[RX]|Extended
```

```

R6.00-00           0x0000000C    0x2DE2          1061         0/0/0
Metric: 10          IS-Extended XRv1.00
Metric: 10          IS-Extended R7.00
R7.00-00           * 0x00000016    0x7843          1053         0/0/0
Metric: 10          IS-Extended R6.00
Metric: 10          IS-Extended XRv1.00
XRv1.00-00         0x0000000C    0xFA12          975          0/0/0
Metric: 10          IS-Extended R6.00
Metric: 10          IS-Extended R7.00

```

```
RP/0/0/CPU0:XRv2#show ospf database router | utility egrep
'Advert|Neighboring'
```

```

Advertising Router: 222.0.0.1
  (Link ID) Neighboring Router ID: 222.0.0.13
  (Link ID) Neighboring Router ID: 222.0.0.12
  (Link ID) Neighboring Router ID: 222.0.0.10
Advertising Router: 222.0.0.10
  (Link ID) Neighboring Router ID: 222.0.0.1
  (Link ID) Neighboring Router ID: 222.0.0.13
  (Link ID) Neighboring Router ID: 222.0.0.12
Advertising Router: 222.0.0.12
  (Link ID) Neighboring Router ID: 222.0.0.10
  (Link ID) Neighboring Router ID: 222.0.0.1
Advertising Router: 222.0.0.13
  (Link ID) Neighboring Router ID: 222.0.0.10
  (Link ID) Neighboring Router ID: 222.0.0.1

```

I repeat the same test as above using the MPLS TED show commands. This shows that all routers are TE-enabled with the proper links in the topology. Assuming all IGP links are TE-capable, the TE topology should mirror the IGP topology.

```

R7#show mpls traffic-eng topology level-2 brief | include IGP_Id
IGP Id: 0000.0000.0001.00, MPLS TE Id:222.0.0.1 Router Node (isis level-2)
  link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0009.00, nbr_node_id:20, gen:32

```

```

IGP Id: 0000.0000.0007.00, MPLS TE Id:222.0.0.7 Router Node (isis level-2)
  link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0014.00, nbr_node_id:22, gen:31
IGP Id: 0000.0000.0008.00, MPLS TE Id:222.0.0.8 Router Node (isis level-2)
  link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0009.00, nbr_node_id:20, gen:33
  link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0014.00, nbr_node_id:22, gen:33
  link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0012.00, nbr_node_id:21, gen:33
IGP Id: 0000.0000.0009.00, MPLS TE Id:222.0.0.9 Router Node (isis level-2)
  link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0008.00, nbr_node_id:19, gen:34
  link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0001.00, nbr_node_id:18, gen:34
  link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0014.00, nbr_node_id:22, gen:34
  link[3]: Point-to-Point, Nbr IGP Id: 0000.0000.0014.00, nbr_node_id:22, gen:34
IGP Id: 0000.0000.0012.00, MPLS TE Id:222.0.0.12 Router Node (isis level-2)
  link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0008.00, nbr_node_id:19, gen:35
IGP Id: 0000.0000.0014.00, MPLS TE Id:222.0.0.14 Router Node (isis level-2)
  link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0008.00, nbr_node_id:19, gen:36
  link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0007.00, nbr_node_id:17, gen:36
  link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0009.00, nbr_node_id:20, gen:36
  link[3]: Point-to-Point, Nbr IGP Id: 0000.0000.0009.00, nbr_node_id:20, gen:36

```

```

R7#show mpls traffic-eng topology level-1 brief | include IGP_Id
IGP Id: 0000.0000.0006.00, MPLS TE Id:222.0.0.6 Router Node (isis level-1)
  link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0011.00, nbr_node_id:23, gen:38
  link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0007.00, nbr_node_id:16, gen:38
IGP Id: 0000.0000.0007.00, MPLS TE Id:222.0.0.7 Router Node (isis level-1)
  link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0006.00, nbr_node_id:15, gen:39
  link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0011.00, nbr_node_id:23, gen:39
IGP Id: 0000.0000.0011.00, MPLS TE Id:222.0.0.11 Router Node (isis level-1)
  link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0006.00, nbr_node_id:15, gen:37
  link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0007.00, nbr_node_id:16, gen:37

```

```

RP/0/0/CPU0:XRv2#show mpls traffic-eng topology ospf brief | include IGP Id
IGP Id: 222.0.0.1, MPLS TE Id: 222.0.0.1 Router Node (OSPF 222 area 0)
  Link[0]:Point-to-Point, Nbr IGP Id:222.0.0.13, Nbr Node Id:4, gen:1870
  Link[1]:Point-to-Point, Nbr IGP Id:222.0.0.12, Nbr Node Id:1, gen:1871
  Link[2]:Point-to-Point, Nbr IGP Id:222.0.0.10, Nbr Node Id:3, gen:1872
IGP Id: 222.0.0.10, MPLS TE Id: 222.0.0.10 Router Node (OSPF 222 area 0)
  Link[0]:Point-to-Point, Nbr IGP Id:222.0.0.12, Nbr Node Id:1, gen:1873
  Link[1]:Point-to-Point, Nbr IGP Id:222.0.0.13, Nbr Node Id:4, gen:1874
  Link[2]:Point-to-Point, Nbr IGP Id:222.0.0.1, Nbr Node Id:2, gen:1875
IGP Id: 222.0.0.12, MPLS TE Id: 222.0.0.12 Router Node (OSPF 222 area 0)
  Link[0]:Point-to-Point, Nbr IGP Id:222.0.0.1, Nbr Node Id:2, gen:1868
  Link[1]:Point-to-Point, Nbr IGP Id:222.0.0.10, Nbr Node Id:3, gen:1869
IGP Id: 222.0.0.13, MPLS TE Id: 222.0.0.13 Router Node (OSPF 222 area 0)
  Link[0]:Point-to-Point, Nbr IGP Id:222.0.0.1, Nbr Node Id:2, gen:1876
  Link[1]:Point-to-Point, Nbr IGP Id:222.0.0.10, Nbr Node Id:3, gen:1877

```

To ensure MPLS transport will function when TE is not configured, I also spot-check some LDP sessions. Ideally, one would verify all sessions, but for brevity I demonstrate that most of them are functional. Selecting routers with many links is a good way to verify this. Out of 15 links in the network, we can verify 10 quickly using 3 show commands.

```

R9#show mpls ldp neighbor | include Peer

```

```

Peer LDP Ident: 222.0.0.8:0; Local LDP Ident 222.0.0.9:0
Peer LDP Ident: 222.0.0.1:0; Local LDP Ident 222.0.0.9:0
Peer LDP Ident: 222.0.0.14:0; Local LDP Ident 222.0.0.9:0

```

```

R1#show mpls ldp neighbor | include Peer
Peer LDP Ident: 222.0.0.9:0; Local LDP Ident 222.0.0.1:0
Peer LDP Ident: 222.0.0.13:0; Local LDP Ident 222.0.0.1:0
Peer LDP Ident: 222.0.0.12:0; Local LDP Ident 222.0.0.1:0
Peer LDP Ident: 222.0.0.10:0; Local LDP Ident 222.0.0.1:0

```

```

R7#show mpls ldp neighbor | include Peer
Peer LDP Ident: 222.0.0.6:0; Local LDP Ident 222.0.0.7:0
Peer LDP Ident: 222.0.0.11:0; Local LDP Ident 222.0.0.7:0
Peer LDP Ident: 222.0.0.14:0; Local LDP Ident 222.0.0.7:0

```

Next, we will validate the interconnected UMPLS topology. The core IGP is IS-IS level-2 with aggregation IGPs of OSPFv2/v3 and IS-IS level-1. To override the default IS-IS behavior, I apply a pair of filters on CSR7. The attached-bit is never allowed to be set into L1 (which would create a default routing entry), while L1 routes are never allowed to be distributed into L2. Though this would normally break IPv4/v6 reachability, it is suitable for UMPLS since BGP carries those prefixes. MPLS is used to tunnel traffic through routers that don't have the IGP routes, but do have reachability to the ABRs.

```

! CSR7
clns filter-set CLNS_FILTER_CLEAR_ATT permit 49.beef

route-map RM_ATT_BIT permit 10
match clns address CLNS_FILTER_CLEAR_ATT

route-map RM_DENY_ALL deny 10

router isis 222
set-attached-bit route-map RM_ATT_BIT
redistribute isis ip level-1 into level-2 route-map RM_DENY_ALL

```

CSR8 is a traditional RR that transports prefixes between aggregation islands. CSR1, CSR7, and XRv2 are modified RRs that adjust BGP next-hops between AS boundaries. This forces BGP to swap labels (labeled-unicast AFI) between aggregation islands, allowing UMPLS to scale as IGP routes are never redistributed. The only BGP AFI that the ABRs need to support is IPv4 labeled-unicast. We quickly verify that this capability is negotiation between CSR8 and the ABRs.

```

R8#show bgp ipv4 unicast summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
222.0.0.1      4      222   520    520     19    0    0 01:20:43      2
222.0.0.7      4      222   376    374     19    0    0 00:56:37      2
222.0.0.12     4      222   291    308     19    0    0 00:46:44      1

```

```

R8#show bgp ipv4 unicast neighbors 222.0.0.1 | include ^BGP|MPLS_Label

```

```
BGP neighbor is 222.0.0.1, remote AS 222, internal link
  ipv4 MPLS Label capability: advertised and received
```

```
R8#show bgp ipv4 unicast neighbors 222.0.0.7 | include ^BGP|MPLS_Label
BGP neighbor is 222.0.0.7, remote AS 222, internal link
  ipv4 MPLS Label capability: advertised and received
```

```
R8#show bgp ipv4 unicast neighbors 222.0.0.12 | include ^BGP|MPLS_Label
BGP neighbor is 222.0.0.12, remote AS 222, internal link
  ipv4 MPLS Label capability: advertised and received
```

For L3VPN service, XRv1 and CSR10 are RRs within their aggregation IGP. Other PEs peer with those RRs, and the RRs peer with one another. The cluster-IDs must be different since these aggregation RRs do not service the same set of clients. These RRs will only have reachability to one another once the aggregation/core infrastructures are functional. This is somewhat similar to CSC where the “customer carriers”, or aggregation IGP in this example, must use MPLS encapsulation for their control plane traffic to communicate over the “core carrier”, or core IGP. Both XRv1 and CSR10 have all of their VPNv4/v6 sessions online.

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast summary | begin ^Neigh
Neighbor      Spk    AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
222.0.0.6      0    222    474    463       64    0    0  01:03:51      5
222.0.0.10     0    222    361    351       64    0    0  00:04:19      9
```

```
RP/0/0/CPU0:XRv1#show bgp vpnv6 unicast summary | begin ^Neigh
Neighbor      Spk    AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
222.0.0.6      0    222    475    464       35    0    0  01:03:58      4
222.0.0.10     0    222    362    352       35    0    0  00:04:26      5
```

```
R10#show bgp vpnv4 unicast all summary | begin ^Neigh
Neighbor      V      AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  State/PfxRcd
222.0.0.11    4      222    44    44       23    0    0  00:05:00      9
222.0.0.13    4      222    39    48       23    0    0  00:05:23      5
```

```
R10#show bgp vpnv6 unicast all summary | begin ^Neigh
Neighbor      V      AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  State/PfxRcd
222.0.0.11    4      222    44    44       22    0    0  00:05:00      8
222.0.0.13    4      222    39    48       22    0    0  00:05:23      5
```

Next, I quickly verify that CSR6 and XRv3, which are PEs, have BGP routes towards one another. The next-hops have been adjusted to the local ABRs so that label swapping occurs there. CSR6 only has one ABR servicing its aggregation IGP while XRv3 has two ABRs. Traceroute indicates that the proper number of labels are being imposed: BGP labels on the bottom with LDP labels on top, as necessary. Seeing only a single label during these tests may indicate that the BGP route recursion is not occurring, and remote loopbacks are IGP learned instead. This negatively affects the scalability of BGP.

```
R6#show bgp ipv4 unicast | begin Network
Network      Next Hop      Metric LocPrf Weight Path
```



```

*> 222.0.0.6/32      0.0.0.0      0      32768 i
*>i 222.0.0.10/32    222.0.0.7      0     100    0 i
r>i 222.0.0.11/32    222.0.0.7      0     100    0 i
*>i 222.0.0.13/32    222.0.0.7      0     100    0 i

```

```

RP/0/0/CPU0:XRv3#show bgp ipv4 labeled-unicast | begin Network
  Network      Next Hop      Metric LocPrf Weight Path
*>i222.0.0.6/32 222.0.0.1      0     100    0 i
* i            222.0.0.12    0     100    0 i
*>i222.0.0.10/32 222.0.0.1      0     100    0 i
* i            222.0.0.12    0     100    0 i
*>i222.0.0.11/32 222.0.0.1      0     100    0 i
* i            222.0.0.12    0     100    0 i
*> 222.0.0.13/32 0.0.0.0      0      32768 i

```

```

R6#traceroute 222.0.0.13 source 222.0.0.6
Type escape sequence to abort.
Tracing the route to 222.0.0.13
VRF info: (vrf in name/id, vrf out name/id)
 1 222.6.7.7 [MPLS: Label 7002 Exp 0] 10 msec 8 msec 7 msec
 2 222.7.14.14 [MPLS: Labels 94005/1003 Exp 0] 31 msec 31 msec 32 msec
 3 222.9.14.9 [MPLS: Labels 9001/1003 Exp 0] 43 msec 31 msec 31 msec
 4 222.1.9.1 [MPLS: Label 1003 Exp 0] 16 msec 21 msec 21 msec
 5 222.1.13.13 20 msec 15 msec 15 msec

```

Several different PE-CE routing protocols are demonstrate in this lab. CSR5 (BGP) and CSR4 (IS-IS) share a VPN, and CSR3 (OSPFv3) and CSR2 (EIGRP) share a VPN. A quick traceroute for IPv4 and IPv6 within each VPN confirms that the general design is functional. Note that IS-IS is not supported for IPv6 inside of a VRF, so only IPv4 is verified there. Since this lab is focused on GR/NSR, we will not trace all LSPs.

```

R5#traceroute 4.4.0.1 source 5.5.0.1
Type escape sequence to abort.
Tracing the route to 4.4.0.1
VRF info: (vrf in name/id, vrf out name/id)
 1 10.5.11.11 3 msec 2 msec 2 msec
 2 222.7.11.7 [MPLS: Labels 7005/10003 Exp 0] 11 msec 10 msec 11 msec
 3 222.7.14.14 [MPLS: Labels 94006/92001/10003 Exp 0] 23 msec 30 msec 33
msec
 4 222.8.14.8 [MPLS: Labels 8006/92001/10003 Exp 0] 37 msec 31 msec 31 msec
 5 222.8.12.12 [MPLS: Labels 92001/10003 Exp 0] 38 msec 31 msec 31 msec
 6 10.4.10.10 [MPLS: Label 10003 Exp 0] 16 msec 15 msec 16 msec
 7 10.4.10.4 19 msec 10 msec 134 msec

```

```

R3#traceroute 2.2.0.1 source 3.3.0.1
Type escape sequence to abort.
Tracing the route to 2.2.0.1
VRF info: (vrf in name/id, vrf out name/id)
 1 10.3.6.6 5 msec 4 msec 4 msec

```

```

2 222.6.7.7 [MPLS: Labels 7002/93013 Exp 0] 11 msec 10 msec 10 msec
3 222.7.14.14 [MPLS: Labels 94005/1003/93013 Exp 0] 29 msec 33 msec 32 msec
4 222.14.9.9 [MPLS: Labels 9001/1003/93013 Exp 0] 33 msec 33 msec 37 msec
5 222.1.9.1 [MPLS: Labels 1003/93013 Exp 0] 32 msec 31 msec 33 msec
6 222.1.13.13 [MPLS: Label 93013 Exp 0] 21 msec 19 msec 22 msec
7 10.2.13.2 20 msec 76 msec 13 msec

```

```

R3#traceroute ipv6
Target IPv6 address: 2002:2:2:a::1
Source address: 3003:3:3:a::1
[snip]

```

```

1 FD00:10:3:6::6 5 msec 3 msec 4 msec
2 ::FFFF:222.6.7.7 [MPLS: Labels 7002/93009 Exp 0] 12 msec 11 msec 12 msec
3 2222:222:7:14::14 [MPLS: Labels 94005/1003/93009 Exp 0] 29 msec 42 msec
36 msec
4 ::FFFF:222.9.14.9 [MPLS: Labels 9001/1003/93009 Exp 0] 35 msec 43 msec 35
msec
5 ::FFFF:222.1.9.1 [MPLS: Labels 1003/93009 Exp 0] 35 msec 46 msec 35 msec
6 2222:222:1:13::13 [MPLS: Label 93009 Exp 0] 23 msec 91 msec 9 msec
7 FD00:10:2:13::2 11 msec 11 msec 10 msec

```

Now that the basic network has been validated, we can progress to the GR/NSR tests. Note that the CSR1000v does not support SSO, which is a requirement for NSF/GR. Some applications of NSF/GR have value outside of the traditional use-case of having redundancy route processors (RP), but generally, these labs cannot be tested in detail.

Additional Reading – Reference configurations “ha”

14.1.1 IS-IS NSF and NSR

IS-IS supports both NSF and NSR. The NSF capability of IS-IS is signaled using the Restart TLV, which is carried in the hello PDUs used by IS-IS. When an IS-IS router supports NSF, this TLV is included for hellos out of all interfaces to signal support for the feature. Both XE and XR support it by default and it is true regardless of the adjacency’s IS-IS level. We confirm this on CSR7 which participates in multiple IS-IS levels and peers with XE and XR routers.

```

R7#show clns is-neighbors detail
Tag 222:
System Id      Interface  State  Type  Priority  Circuit Id      Format
R6             Gi2.567   Up     L1    0         00              Phase V
Area Address(es): 49.6711
IP Address(es):  222.6.7.6*
IPv6 Address(es): FE80::6
Uptime: 01:35:41
NSF capable
Topology: IPv4, IPv6
Interface name: GigabitEthernet2.567

```

```

XRv1          Gi2.571      Up      L1      0          00          Phase V
Area Address(es): 49.6711
IP Address(es):  222.7.11.11*
IPv6 Address(es): FE80::11
Uptime: 01:35:35
NSF capable
Topology: IPv4, IPv6
Interface name: GigabitEthernet2.571
XRv4          Gi2.574      Up      L2      0          00          Phase V
Area Address(es): 49.0000
IP Address(es):  222.7.14.14*
IPv6 Address(es): FE80::14
Uptime: 01:35:37
NSF capable
Topology: IPv4, IPv6
Interface name: GigabitEthernet2.574

```

Enabling NSF is very simple and the command is the same for XE and XR. We enable this on all IS-IS level-2 and level-1-2 routers. The “ietf” option must be used with the command to enable NSF; this is discussed more later.

```

! CSR1, CSR7, CSR8, CSR9, XRv2, and XRv4
router isis 222
nsf ietf

```

While not very exciting, NSF is not entirely enabled on these routers. Without SSO to synchronize all of the non-routing information between RPs, such as device configurations, NSF cannot realistically work. The routers will already operate in “helper” mode, which can acknowledge that a peer is performing a switchover and continue forwarding traffic (or any control-plane information required) towards it. Using basic show commands, we can verify it is enabled. XR’s only show command appears to be a simple yes/no field inside the general IS-IS protocol details.

```

R7#show isis nsf
Tag 222:
NSF is ENABLED, mode 'ietf'
NSF pdb state:
NSF L1 active interfaces: 0
NSF L1 active LSPs: 0
NSF interfaces awaiting L1 CSNP: 0
Awaiting L1 LSPs:
NSF L2 active interfaces: 0
NSF L2 active LSPs: 0
NSF interfaces awaiting L2 CSNP: 0
Awaiting L2 LSPs:
NSF T3 remaining: 0 seconds
Interface: GigabitEthernet2.567
NSF L1 Restart state: Running

```

```

NSF p2p Restart retransmissions: 0
Maximum L1 NSF Restart retransmissions: 3
Interface: GigabitEthernet2.571
NSF L1 Restart state: Running
NSF p2p Restart retransmissions: 0
Maximum L1 NSF Restart retransmissions: 3
Interface: GigabitEthernet2.574
NSF L2 Restart state: Running
NSF p2p Restart retransmissions: 0
Maximum L2 NSF Restart retransmissions: 3

RP/0/0/CPU0:XRv4#show isis protocol | include Non-stop
Non-stop forwarding: IETF NSF Restart enabled

```

Since the feature is not enabled on CSR6 and XRv1, I use the same show commands to demonstrate the output when NSF is disabled. On XE, this also shows us that SSO is not configured or operational.

```

R6#show isis nsf
Tag 222:
NSF is DISABLED
RP is ACTIVE, standby not ready, RTR chkpt peer not ready, UPD chkpt peer not
ready, bulk sync pending
NSF interval timer expired (NSF OK, checkpoint pending)
Checkpointing disabled, no errors
Local state: ACTIVE, Peer state: DISABLED, Config Mode: non-SSO, Operating
Mode: non-SSO

RP/0/0/CPU0:XRv1#show isis protocol | include Non-stop
Non-stop forwarding: Disabled

```

Debugging NSF, in a stable network, shows the adjacency timer refreshes occurring every 10 seconds, which is the P2P hello timer for IS-IS by default. I color-code the timers to show the difference of 10 seconds across 3 different interfaces.

```

R8#debug isis nsf
IS-IS NSF events debugging is on for router process 222

17:13:35.392: ISIS-NSF (222): Adjacency timer refreshed: GigabitEthernet2.584
17:13:36.781: ISIS-NSF (222): Adjacency timer refreshed: GigabitEthernet2.589
17:13:39.840: ISIS-NSF (222): Adjacency timer refreshed: GigabitEthernet2.582
17:13:44.841: ISIS-NSF (222): Adjacency timer refreshed: GigabitEthernet2.584
17:13:44.973: ISIS-NSF (222): Adjacency timer refreshed: GigabitEthernet2.589
17:13:49.049: ISIS-NSF (222): Adjacency timer refreshed: GigabitEthernet2.582

```

I enable debugging on CSR7 and also clear its ISIS process. As an L1/L2 router, we can see CSR7 trying to establish both types of adjacencies. It iterates through them, bringing up the L1 adjacencies first. The true/false markers indicate what kinds of adjacencies IS-IS needs to establish with each peer. L1

adjacencies are shown in yellow while L2 adjacencies are shown in green. XR does not appear to have an equivalent debug.

```
R7#debug isis nsf
IS-IS NSF events debugging is on for router process 222

R7#clear isis *

%CLNS-5-ADJCLEAR: ISIS (222): All adjacencies cleared
ISIS-NSF (222): Pick ADJ - XRv1 (Gi2.571):
ISIS-NSF (222): L1 BEST ADJ - XRv1 (Gi2.571) = not found => TRUE
ISIS-NSF (222): L2 BEST ADJ - XRv1 (Gi2.571) = not found => FALSE
ISIS-NSF (222): Pick ADJ - XRv1 (Gi2.571):
ISIS-NSF (222): L1 BEST ADJ - XRv1 (Gi2.571) = not found => TRUE
ISIS-NSF (222): L2 BEST ADJ - XRv1 (Gi2.571) = not found => FALSE
%CLNS-5-ADJCHANGE: ISIS (222): Adjacency to XRv1 (GigabitEthernet2.571) Up,
new adjacency
ISIS-NSF (222): Pick ADJ - R6 (Gi2.567):
ISIS-NSF (222): L1 BEST ADJ - R6 (Gi2.567) = not found => TRUE
ISIS-NSF (222): L2 BEST ADJ - R6 (Gi2.567) = not found => FALSE
ISIS-NSF (222): Pick ADJ - R6 (Gi2.567):
ISIS-NSF (222): L1 BEST ADJ - R6 (Gi2.567) = not found => TRUE
ISIS-NSF (222): L2 BEST ADJ - R6 (Gi2.567) = not found => FALSE
%CLNS-5-ADJCHANGE: ISIS (222): Adjacency to R6 (GigabitEthernet2.567) Up, new
adjacency
ISIS-NSF (222): Pick ADJ - XRv4 (Gi2.574):
ISIS-NSF (222): L1 BEST ADJ - XRv4 (Gi2.574) = not found => FALSE
ISIS-NSF (222): L2 BEST ADJ - XRv4 (Gi2.574) = not found => TRUE
ISIS-NSF (222): Pick ADJ - XRv4 (Gi2.574):
ISIS-NSF (222): L1 BEST ADJ - XRv4 (Gi2.574) = not found => FALSE
ISIS-NSF (222): L2 BEST ADJ - XRv4 (Gi2.574) = not found => TRUE
%CLNS-5-ADJCHANGE: ISIS (222): Adjacency to XRv4 (GigabitEthernet2.574) Up,
new adjacency
```

Next, we will configure several minor options. There are several timers associated with NSF. The “T3” timer is used to determine how long to wait before setting the overload bit (similar to OSPF max-metric) when a switchover occurs. “T3” is equivalent to the “restart-time” in other protocols or the “lifetime” in XR. This would force traffic to avoid a particular node by causing IGP to converge around it. The “interface” timer determines how long to wait for interfaces to come up before completing the restart/switchover. The “interval” timer is a throttle to ensure NSF restarts to do not happen too quickly, and serves as a hold-down time between them. XR allows you to define the interface timer (T1) which defines how long to wait before resending an unacknowledged restart attempt. This should be a few seconds, and I adjust it to 5. The expiration counter tells IS-IS to stop trying after 3 attempts. These timers do not appear in any obvious show commands. They are demonstrated here to show NSF’s granularity.

```

! CSR7
router isis 222
  nsf interval 7
  nsf interface wait 15
  nsf t3 manual 60

router isis 222
  nsf lifetime 60
  nsf interface-timer 5
  nsf interface-expires 3

```

Enabling NSR is less straightforward than NSF. The primary difference between IS-IS NSR and NSF is that NSF is an IETF standard that involves interactions between routers. The capability is signaled via the Restart TLV and routers are expected to know when peers switchover. NSR is local-only and does not signal anything to remote peers. The configuration for XE and XR is below. This uses the “cisco” keyword as it was developed before IETF NSF.

```

! CSR6 and XRv1
router isis 222
  nsf cisco

```

When this is configured on the CSR1000v, several errors are printed as shown below. This is followed by syslog messages every 60 seconds as well. I assume this is because the virtual platform does not support it, or possibly because SSO is not configured (also unsupported). Despite these errors, the configuration command can remain in the configuration for reference.

```

R6(config-router)#nsf cisco
Adjacency and LSP information learned from interface GigabitEthernet2.561
cannot be checkpointed due to an interface encoding error.
Adjacency and LSP information learned from interface GigabitEthernet2.567
cannot be checkpointed due to an interface encoding error.

```

Please verify that these interface types are supported for NSF in this release. If so, report the problem to the TAC.

```

%CLNS-3-NSF_CP_IDB_ENCODE_FAIL: ISIS (222): Interface Gi2.561 cannot be
encoded for nsf cisco
%CLNS-3-NSF_CP_IDB_ENCODE_FAIL: ISIS (222): Interface Gi2.567 cannot be
encoded for nsf cisco

```

CSR6 still reports NSF as being enabled, but this is effectively NSR when the mode is “cisco”.

```

R6#show isis nsf
Tag 222:
NSF is ENABLED, mode 'cisco'

```

```
RP is ACTIVE, standby not ready, RTR chkpt peer not ready, UPD chkpt peer not ready, bulk sync pending
NSF interval timer expired (NSF restart enabled)
Checkpointing disabled, no errors
Local state: ACTIVE, Peer state: DISABLED, Config Mode: non-SSO, Operating Mode: non-SSO
```

XRv1 appears to accept the command without complaint, but the feature is still inoperable.

```
RP/0/0/CPU0:XRv1#show isis protocol | include Non-stop
Non-stop forwarding: Cisco Proprietary NSF Restart enabled
```

14.1.2 OSPFv2 NSF and NSR

OSPFv2 NSF and NSR are similar to IS-IS in concept. NSF requires some extensions to OSPF (IS-IS used a Restart TLV, which isn't really an extension) so that the peer routers remain aware of the restarting router. When the router performs a switchover, the neighbors will continue to sustain the existing LSA1 information that describes the network topology. This allows traffic to continue forwarding through the router until the grace period expires. OSPF uses new "grace LSAs", which are link-local opaque LSAs containing the grace period time in their payloads. This is how NSF information is exchanged between routers, and being link-local in scope, they are not flooded throughout the area/domain. The configuration is identical to IS-IS, and is also the same on XE and XR.

```
! CSR1 and XRv2
router ospf 222
nsf ietf
```

XE has a number of verification methods while XR does not. Of note, NSF "helper" support is enabled by default in both platforms, which is valuable on virtual routers. If they were peered with hardware routers that actually supported SSO/NSF, they could honor the received grace LSAs and react accordingly. XE clearly tells us that NSF is not operating due to SSO not being configured.

```
R1#show ip ospf | include NSF|IETF
IETF Non-Stop Forwarding enabled
IETF NSF helper support enabled
Cisco NSF helper support enabled
```

```
R1#show ip ospf nsf
Routing Process "ospf 222"
IETF Non-Stop Forwarding enabled
  restart-interval limit: 120 sec
  Router is not operating in SSO mode
  Global RIB has not converged yet
IETF NSF helper support enabled
Cisco NSF helper support enabled
  OSPF restart state is NO_RESTART
Handle 140512595585760, Router ID 222.0.0.1, checkpoint Router ID 0.0.0.0
```

```
Config wait timer interval 10, timer not running
Dbase wait timer interval 120, timer not running
```

```
RP/0/0/CPU0:XRv2#show ospf | include Non-Stop
Non-Stop Forwarding enabled
```

```
RP/0/0/CPU0:XRv2#show protocols ospf | include Non-Stop
Non-Stop Forwarding: Enabled
```

We also see that the default restart interval is 2 minutes. If the neighbor does not complete switchover during that time, the helper routers will update the LSA1s to cause a reconvergence event. Setting this timer too short may not give the switching router enough time to complete the SSO process. Setting it too long may create a longer black hole if the router legitimately fails and cannot fail over, yet somehow fails to signal it. For demonstration, I increase the timer to 10 minutes on both CSR1 and XRv2. Note that this “restart-interval” is equivalent in purpose to the “T3” timer in IS-IS.

```
! CSR1
router ospf 222
 nsf ietf restart-interval 600

! XRv2
router ospf 222
 nsf lifetime 600

R1#show ip ospf nsf
Routing Process "ospf 222"
IETF Non-Stop Forwarding enabled
 restart-interval limit: 600 sec
Router is not operating in SSO mode
Global RIB has not converged yet
[snip]
```

We can also adjust a router’s willingness to be a helper for NSF. The pre-standard NSF version described by Cisco is not discussed in detail here, but unlike IS-IS, “nsf cisco” does not enable NSR. It was a proprietary Cisco implementation that extended OSPF; it was not internal-only as NSR is. XRv2 disables the helper capability for IETF NSF while CSR1 disables it for the Cisco version. You cannot disable helper support for Cisco NSF on XR.

```
! XRv2
router ospf 222
 nsf ietf helper disable

! CSR1
router ospf 222
 nsf cisco helper disable

R1#show ip ospf nsf | include helper
```



```
IETF NSF helper support enabled
Cisco NSF helper support disabled
```

Next, we configure OSPF NSR on XRv3 and CSR10. This actually uses the “nsr” command which simplifies the logic as compared to IS-IS NSR (using the command “nsf cisco”). This is good for routers in a heterogeneous environment where it is not guaranteed that the peers are NSF-capable. For example, XRv2 is not willing to be an IETF NSF helper, so CSR10 should consider using NSR versus NSF. The configuration is a single command with no arguments, and there are no other options to tune.

```
! CSR10 and XRv3
router ospf 222
nsr
```

XE has a rich set of show commands to verify NSR. I’ve highlighted several fields that suggest NSR is enabled but inoperable. NSR works by checkpointing the routing state and synchronizing that to the standby RP internally. Such checkpoints cannot be synchronized when there is only one RP.

```
R10#show ip ospf nsr
Active RP
Operating in simplex mode
Redundancy state: ACTIVE
Peer redundancy state: DISABLED
Checkpoint peer not ready
Checkpoint messages enabled
ISSU negotiation not complete
ISSU versions not compatible

Routing Process "ospf 222" with ID 222.0.0.10
NSR configured
Checkpoint message sequence number: 0
Standby synchronization state: unsynchronized
Bulk sync operations: 0
Next sync check time: -
LSA Count: 18, Checksum Sum 0x000A1D2B
```

Assuming checkpoints were being generated, the statistics below would have non-zero values. This can be used to verify the synchronization within a router is functional.

```
R10#show ip ospf nsr statistics
Pending checkpoint requests (current/max): 0/0
Pending checkpoint messages (current/max): 0/0

Routing Process "ospf 222" with ID 222.0.0.10
Pending checkpoint requests (current/max): 0/0
Pending checkpoint messages (current/max): 0/0
Time spent scheduling bulk syncs (max): 0 ms
```

```
Time spent in checkpoint loop (average/max): 0/0 ms
Checkpoint loop interruptions: 0
```

XR validation is limited to a single line of output, but this confirms the feature is enabled.

```
RP/0/0/CPU0:XRv3#show ospf | include NSR
NSR (Non-stop routing) is Enabled
```

14.1.3 OSPFv3 GR and NSR

OSPFv3 is very similar to OSPFv2 in terms of HA. One notable difference is that OSPFv3 calls NSF “graceful-restart”, but the terms NSF and GR are interchangeable. To configure NSF/GR, we generally substitute the word “nsf” for “graceful-restart” and the command syntax is similar to OSPFv2. There is no Cisco proprietary feature in OSPFv3, so differentiating Cisco from IETF is not necessary. I enable GR on all routers in the right-most aggregation IGP. I also adjust the restart-interval to 5 minutes, allowing any switching routers some extra time to complete the switchover.

```
! CSR1 and CSR10
router ospfv3 222
 graceful-restart restart-interval 300
```

```
! XRv2 and XRv3
router ospfv3 222
 graceful-restart
 graceful-restart lifetime 300
```

We can verify OSPFv3 GR using some simple show commands on XE and XR. Just like OSPFv2, XE provides more detail about the feature than XR. Both platforms also support GR helper mode by default, which could be valuable on virtual routers that peer with physical routers. There aren’t many timers to adjust with this feature, either.

```
R10#show ospfv3 graceful-restart

      OSPFv3 222 address-family ipv6 (router-id 222.0.0.10)

Graceful Restart enabled
restart-interval limit: 300 sec
Router is NOT running in SSO mode
Graceful Restart helper support enabled
Number of neighbors performing Graceful Restart is 0
```

```
RP/0/0/CPU0:XRv2#show ospfv3 | include Grace
Graceful Restart enabled
```

To test OSPFv3 NSR, I will use the PE-CE connection between CSR3 and CSR6. CSR6 disables GR helper mode, which means CSR3 should support NSR (heterogeneous environment). CSR6 could technically run GR since CSR3 did not disable helper mode, but for consistency, both PE and CE use NSR over GR.

```
! CSR6
router ospfv3 3
  nsr
  graceful-restart helper disable

! CSR3
router ospfv3 3
  nsr
```

Before verifying NSR, we verify that GR helper mode is enabled on CSR3 (default) and disabled on CSR6. This is the main motivation for wanting to use NSR on CSR3 specifically. Since NSR can be VRF-aware, we must issue the proper show command to verify this on CSR6.

```
R3#show ospfv3 graceful-restart
      OSPFv3 3 address-family ipv4 (router-id 3.3.0.1)
Graceful Restart helper support enabled
      Number of neighbors performing Graceful Restart is 0

      OSPFv3 3 address-family ipv6 (router-id 3.3.0.1)
Graceful Restart helper support enabled
      Number of neighbors performing Graceful Restart is 0

R6#show ospfv3 vrf OSPF graceful-restart
      OSPFv3 3 address-family ipv4 vrf OSPF (router-id 10.3.6.6)
Graceful Restart helper support disabled

      OSPFv3 3 address-family ipv6 vrf OSPF (router-id 10.3.6.6)
Graceful Restart helper support disabled
```

Like OSPFv2, XE provides great detail regarding NSR state and behavior. I highlight the important lines, just as in OSPFv2, which indicate that NSR is configured but not operational. NSR is configured for both IPv4 and IPv6 AFIs automatically.

```
R6#show ospfv3 vrf OSPF nsr
Active RP
Operating in simplex mode
Redundancy state: ACTIVE
Peer redundancy state: DISABLED
Checkpoint peer not ready
Checkpoint messages enabled
ISSU negotiation not complete
ISSU versions not compatible
```

```
OSPFv3 3 address-family ipv4 vrf OSPF (router-id 10.3.6.6)
NSR configured
Checkpoint message sequence number: 0
Standby synchronization state: unsynchronized
Bulk sync operations: 0
Next sync check time: -
LSA Count: 10, Checksum Sum 0x00078C5F
```

```
OSPFv3 3 address-family ipv6 vrf OSPF (router-id 10.3.6.6)
NSR configured
Checkpoint message sequence number: 0
Standby synchronization state: unsynchronized
Bulk sync operations: 0
Next sync check time: -
LSA Count: 10, Checksum Sum 0x00055A60
```

NSR statistics regarding the checkpoints are available, but unpopulated as the feature is not properly enabled.

```
R6#show ospfv3 vrf OSPF nsr statistics
Pending checkpoint requests (current/max): 0/0
Pending checkpoint messages (current/max): 0/0
```

```
OSPFv3 3 address-family ipv4 vrf OSPF (router-id 10.3.6.6)
Pending checkpoint requests (current/max): 0/0
Pending checkpoint messages (current/max): 0/0
Time spent scheduling bulk syncs (max): 0 ms
Time spent in checkpoint loop (average/max): 0/0 ms
Checkpoint loop interruptions: 0
```

```
OSPFv3 3 address-family ipv6 vrf OSPF (router-id 10.3.6.6)
Pending checkpoint requests (current/max): 0/0
Pending checkpoint messages (current/max): 0/0
Time spent scheduling bulk syncs (max): 0 ms
Time spent in checkpoint loop (average/max): 0/0 ms
Checkpoint loop interruptions: 0
```

14.1.4 BGP GR and NSR

Unlike many other forms of GR/NSF/NSR on virtual platforms, BGP GR actually has value. BGP GR's primary purpose is the same as the other protocols: don't introduce churn into the protocol while a router switches over, specify a grace period, and continue forwarding as normal. BGP accomplishes this by introducing the End-of-RIB (EoR) marker, as well as a new "Graceful Restart" capability.

RFC 4724 makes specific mention that this EoR marker can be valuable to speed up convergence as well.

Because it marks the end of RIB, it is effectively an explicit signal to the peer that the BGP UPDATE messages have ceased. Rather than wait for more updates, a BGP speaker is able to immediately begin

running best-path selections once all peers indicate EoR. Normally this period is controlled by the “update-delay” timer which specifies how long BGP can stay in read-only mode. The EoR marker is a more event-driven mechanism for starting the best-path selection process. Specifically, the EoR marker for IPv4 is an UPDATE message with no prefixes. For all other AFIs, it uses the MP_UNREACH_NLRI, a multi-protocol extension used for identifying unreachable prefixes (withdrawals). No prefixes are contained inside this message, either.

XR advertises the BGP GR capability to all peers by default. We can check CSR8 to prove this. CSR8 has GR disabled for all peers by default, yet the capability is advertised from XRv2. This is not an AFI-specific feature and applies to the BGP session in general. Checking XRv1 for VPNv4, we can see it advertises the GR capability to CSR10 and CSR6 by default even though only VPNv4/v6 is negotiated, not IPv4.

```
R8#show bgp ipv4 unicast neighbors | include ^BGP|race
BGP neighbor is 222.0.0.1, remote AS 222, internal link
  Graceful-Restart is disabled
BGP neighbor is 222.0.0.7, remote AS 222, internal link
  Graceful-Restart is disabled
BGP neighbor is 222.0.0.12, remote AS 222, internal link
  Graceful Restart Capability: received
  Graceful-Restart is disabled
```

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast neighbors | utility egrep '^BGP|race'
BGP neighbor is 222.0.0.6
  Graceful Restart (GR Awareness): advertised
BGP neighbor is 222.0.0.10
  Graceful Restart (GR Awareness): advertised
```

To be clear, XR advertises that it is “GR-aware” but the actual feature is not enabled by default in XR. Below, I show the output on XRv2 within the IPv4 labeled-unicast AFI. GR awareness is advertised to the XE peers, and bidirectionally exchanged with the XR peers.

```
RP/0/0/CPU0:XRv2#show bgp ipv4 labeled-unicast neighbors | utility egrep
'^BGP|race'
BGP neighbor is 222.0.0.8
  Graceful Restart (GR Awareness): advertised
BGP neighbor is 222.0.0.10
  Graceful Restart (GR Awareness): advertised
BGP neighbor is 222.0.0.13
  Graceful Restart (GR Awareness): advertised and received
```

Next, I enable the feature on XRv2 and re-issue the show command. GR is now “enabled” for each neighbor by default.

```
! XRv2
router bgp 222
```

```
bgp graceful-restart
```

```
RP/0/0/CPU0:XRv2#show bgp ipv4 labeled-unicast neighbors | utility egrep  
'^BGP|race'
```

```
BGP neighbor is 222.0.0.8
```

```
Graceful restart is enabled
```

```
Graceful Restart (GR Awareness): advertised
```

```
Graceful Restart capability advertised
```

```
BGP neighbor is 222.0.0.10
```

```
Graceful restart is enabled
```

```
Graceful Restart (GR Awareness): advertised
```

```
Graceful Restart capability advertised
```

```
BGP neighbor is 222.0.0.13
```

```
Graceful restart is enabled
```

```
Graceful Restart (GR Awareness): advertised and received
```

```
Graceful Restart capability advertised
```

XRv2 will assume that CSR10 is incapable of GR. Whether this is true or not does not matter; below I illustrate explicitly disabling BGP GR for a specific peer. XRv2 continues to notify CSR10 that it is GR-aware, but is not running the feature with CSR10 anymore. GR is still enabled for the other peers as shown below.

```
! XRv2
```

```
router bgp 222
```

```
neighbor 222.0.0.10
```

```
graceful-restart disable
```

```
RP/0/0/CPU0:XRv2#show bgp ipv4 labeled-unicast neighbors | utility egrep  
'^BGP|race'
```

```
BGP neighbor is 222.0.0.8
```

```
Graceful restart is enabled
```

```
Graceful Restart (GR Awareness): advertised
```

```
Graceful Restart capability advertised
```

```
BGP neighbor is 222.0.0.10
```

```
Graceful Restart (GR Awareness): advertised
```

```
BGP neighbor is 222.0.0.13
```

```
Graceful restart is enabled
```

```
Graceful Restart (GR Awareness): advertised and received
```

```
Graceful Restart capability advertised
```

Enabling BGP GR on XE is identical, except that BGP sessions must be hard-reset before the changes take effect. I begin with CSR8.

```
! CSR8
```

```
router bgp 222
```

```
bgp graceful-restart
```

```
R8(config-router)#bgp graceful-restart
```

All BGP sessions must be reset to take the new GR config

```
R8#clear bgp ipv4 unicast *
```

Now, CSR8 advertises this capability to all routers. CSR1 and CSR7 are not advertising it yet, but XRv2 always was; CSR8 and XRv2 are now capable of using BGP GR fully with one another. Again, the NSF capabilities of GR still require SSO and redundant RPs, so this would be specific to the routing optimization aspect (via the EoR marker). CSR8 claims GR is enabled for all peers, but the capability should be exchanged bidirectionally first.

```
R8#show bgp ipv4 unicast neighbors | include ^BGP|race
BGP neighbor is 222.0.0.1, remote AS 222, internal link
    Graceful Restart Capability: advertised
    Graceful-Restart is enabled, restart-time 120 seconds, stalepath-time 360
seconds
BGP neighbor is 222.0.0.7, remote AS 222, internal link
    Graceful Restart Capability: advertised
    Graceful-Restart is enabled, restart-time 120 seconds, stalepath-time 360
seconds
BGP neighbor is 222.0.0.12, remote AS 222, internal link
    Graceful Restart Capability: advertised and received
    Graceful-Restart is enabled, restart-time 120 seconds, stalepath-time 360
seconds
```

I quickly enable GR on all other BGP speakers except CSR10 and XRv3. To avoid any inconsistent behavior, I hard-clear all BGP sessions on all BGP routers. CSR8 now shows that the capability is exchanged with all peers. XRv1 exchanges it correct with CSR6, but not CSR10, which is correct. We are assuming CSR10 and XRv13 are not GR capable.

```
! CSR1, CSR6, CSR7, and XRv1
router bgp 222
    bgp graceful-restart
```

```
R8#show bgp ipv4 unicast neighbors | include ^BGP|tart_Cap
BGP neighbor is 222.0.0.1, remote AS 222, internal link
    Graceful Restart Capability: advertised and received
BGP neighbor is 222.0.0.7, remote AS 222, internal link
    Graceful Restart Capability: advertised and received
BGP neighbor is 222.0.0.12, remote AS 222, internal link
    Graceful Restart Capability: advertised and received
```

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast neighbors | utility egrep '^BGP|race'
BGP neighbor is 222.0.0.6
    Graceful restart is enabled
    Graceful Restart (GR Awareness): advertised and received
    Graceful Restart capability advertised
    Graceful Restart capability received
```

```

    Graceful Restart capability advertised
    Graceful Restart capability received
BGP neighbor is 222.0.0.10
    Graceful restart is enabled
    Graceful Restart (GR Awareness): advertised
    Graceful Restart capability advertised
    Graceful Restart capability advertised

```

For cleanup, any routers that peer with CSR10 or XRv3 will disable GR towards that peer. This was demonstrated once in XRv2, but I apply it everywhere necessary. It is hard to see the value of BGP GR on Cisco routers with respect to using EoR for routing optimization. This is because they send BGP keepalives as soon as the updates are finished sending, which allows BGP best-path to start sooner. Routers that do not behave this way (such as other vendor products) may benefit. This keepalive technique is not a BGP requirement and is specific to Cisco platforms. When used only for routing convergence optimizations, BGP GR makes more sense in multi-vendor environments. The remainder of this test will focus on the traditional use-case of GR/NSF and not the convergence optimizations.

```

! XRv1
router bgp 222
    neighbor 222.0.0.10
        graceful-restart disable

! XRv2
router bgp 222
    neighbor 222.0.0.13
        graceful-restart disable

! CSR1
router bgp 222
    neighbor 222.0.0.10 ha-mode graceful-restart disable
    neighbor 222.0.0.13 ha-mode graceful-restart disable

```

BGP GR specifies two significant timers: restart-time and stalepath-time. The restart-time begins counting down once a TCP session fails between a pair of peers during an RP switchover. The peer router (not the restarting router) must provide routing information to the new standby RP, and this timer controls how long to wait for a BGP OPEN message from the router that is switching over. It is equivalent to the IGP NSF restart-timers that limit the switchover time. When a router begins an RP switchover, all routes learned from that peer are marked as stale, yet traffic is still allowed to forward through that peer. The stalepath-time controls how long these routes can be used before they are flushed. Normally, it is longer than the restart-time because receiving an OPEN message will happen before total BGP convergence, especially in large networks. When BGP GR is enabled on XR, these timers are explicitly added to the configuration using their default values as shown below. The restart and stale timers are 2 and 6 minutes respectively.

```

! XE default values for BGP GR

```



```
router bgp 222
  bgp graceful-restart restart-time 120
  bgp graceful-restart stalepath-time 360
  bgp graceful-restart
```

This does not occur in XR, so for consistency, I configure these timers on the BGP XR routers manually. XRv3 is excluded since it notionally does not support GR.

```
! XRv1 and XRv2
router bgp 222
  bgp graceful-restart restart-time 120
  bgp graceful-restart stalepath-time 360
```

NSR is less straightforward. On XE, we adjust the HA-mode to SSO in order to enable NSR. On XR, the command is simply “nsr”. Although the goal of NSR is identical to the IGP NSR features, the BGP use-case is focused heavily on PE-CE routing. A common case would be that the CE router does not support BGP GR, which means it cannot “help” the PE. Rather than just configure NSR on the PE, which is supported, it makes sense to leave GR configured and add NSR as well. This will allow the router to be “helped” by iBGP peers within the provider’s network, while also allowing it to not rely on GR if it is unavailable. XRv1 is configured with BGP NSR below, and it’s supported CE, CSR5, was never configured with BGP GR.

```
! XRv1
router bgp 222
  nsr
```

The “show bgp nsr” command prints about 5 pages of output. For brevity, I show the first few stanzas. Most of this output isn’t NSR specific, but lists the critical BGP behaviors that are expected to be consistent when the RP changes. This includes basic fields such as the BGP ASN, BGP router ID, GR capability, and various timers. In this configuration, XRv1 can be protected by iBGP peers via GR and can protect itself using NSR.

```
RP/0/0/CPU0:XRv1#show bgp nsr
BGP Process Information:
BGP is operating in STANDALONE mode
Autonomous System number format: ASPLAIN
Autonomous System: 222
Router ID: 222.0.0.11
Default Cluster ID: 222.0.0.11
Active Cluster IDs: 222.0.0.11
Fast external fallover enabled
Neighbor logging is enabled
Enforce first AS enabled
Default local preference: 100
Default keepalive: 60
```

Graceful restart enabled

```
Restart time: 120
Stale path timeout time: 360
RIB purge timeout time: 600
Non-stop routing is enabled
Update delay: 120
Generic scan interval: 60

Address family: IPv4 Unicast
Dampening is not enabled
Client reflection is enabled in global config
Dynamic MED is Disabled
Dynamic MED interval : 10 minutes
Dynamic MED Timer : Not Running
Dynamic MED Periodic Timer : Not Running
Scan interval: 60
Total prefixes scanned: 4
Prefixes scanned per segment: 100000
Number of scan segments: 1
Nexthop resolution minimum prefix-length: 0 (not configured)
Main Table Version: 61
Table version synced to RIB: 61
Table version acked by RIB: 61
IGP notification: IGP notified
RIB has converged: version 13
RIB table prefix-limit reached ? [No], version 0
Permanent Network Unconfigured
[snip]
```

I enable NSR on all provider BGP speakers running XE, as well as XRv2 for completeness. Just like BGP GR, this requires a hard reset on all peers for which NSR is configured on XE routers.

```
! CSR1, CSR6, CSR7, and CSR8
router bgp 222
  bgp ha-mode sso

! XRv2
router bgp 222
  nsr
```

Just like before, we should not enable this feature towards CSR4 or XRv3. There is no way to control this on XR, but XE allows it to be disabled on a per-neighbor basis, just like GR.

```
! CSR1
router bgp 222
  neighbor 222.0.0.10 ha-mode sso disable
  neighbor 222.0.0.13 ha-mode sso disable
```

To verify the feature, I begin with CSR8. NSR is being used interchangeably with SSO here, and it is enabled for all peers. The XE peers show a communication failure while the XR peer shows that the feature is not enabled. This is probably a result of different implementations, and since the feature doesn't work on virtual machines anyway, I do not examine the output further.

```
R8#show bgp ipv4 unicast neighbors | include ^BGP|SSO
BGP neighbor is 222.0.0.1, remote AS 222, internal link
SSO is enabled
SSO last disable reason: Application disable (Active)
BGP neighbor is 222.0.0.7, remote AS 222, internal link
SSO is enabled
SSO last disable reason: Communication failure (Active)
BGP neighbor is 222.0.0.12, remote AS 222, internal link
SSO is enabled
SSO last disable reason: Application disable (Active)
```

Like XR, there are custom commands for NSR as well, except they are executed per AFI. The XR commands can also be issued per-AFI, but the global command shown above returns output for all AFIs. The output is very verbose, and most of it is invalid as a result of the feature not being fully supported on the CSR1000v. There are 3 sessions established with NSR enabled, but none of them support SSO.

```
R8#show bgp ipv4 unicast sso summary
Total sessions with stateful switchover support enabled: 0
Total sessions configured with NSR mode and in established state: 3
Total number of times throttle ON by TCP: 0
Total number of times throttle OFF by TCP: 0
Total number of times CF Message Transmission Failure: 0
Total number of times CF Flow OFF received: 0
Total number of times CF Flow ON received: 0
Total number of times CF Flow Warning message received: 0
Total number of times CF peer-not-ready to sync: 0
RF sent peer-comm-down notification: NO
Total number of sessions on the Enable SyncQ: 0
Total number of sessions on the Update SyncQ: 0
Total number of sessions removed from Tcp SyncQ during Bulk Sync: 0
Total number of sessions removed from Enable SyncQ during Bulk Sync: 0
Total number of sessions removed from Update SyncQ during Bulk Sync: 0
Total number of times TCP sent Sync Enable Notification: 0
Total number of times TCP sent Sync Disable Notification: 0
Total number of times TCP sent Retry SSO Notification: 0
Total APDb walk prefixes sent: 0
Total APDb walk prefix send failures: 0
Total APDb topology walk prefixes sent: 0
Total APDb topology walk prefix send failures: 0
Total APDb prefixes sent: 0
Total WPdb prefixes sent: 0
bgp_tcp_sso_enable_pending is 0
```

```
tcp_syncQ: 0, enable_syncQ: 0, update_syncQ: 0
Bulk Sync Done: NO
```

For completeness, we can see CSR6 trying to use NSR towards XRv1 inside the VPNv4/v6 AFI. Although it still fails, this shows that NSR is supported in many AFIs.

```
R6#show bgp vpnv4 unicast all neighbors | include ^BGP|SSO
BGP neighbor is 222.0.0.11, remote AS 222, internal link
SSO is enabled
SSO last disable reason: Application disable (Active)
```

```
R6#show bgp vpnv6 unicast all neighbors | include ^BGP|SSO
BGP neighbor is 222.0.0.11, remote AS 222, internal link
SSO is enabled
SSO last disable reason: Application disable (Active)
```

14.1.5 LDP GR and NSR

Although not a routing protocol, LDP is critical for most MPLS applications. It also has the concept of GR/NSF and NSR for this reason. Because LDP is an overlay protocol of sorts, it is recommended that the underlying protocols in the network that interact with LDP are also GR/NSF-enabled. This includes OSPF, IS-IS, and/or BGP. Similar to the IGP, when LDP GR is enabled without SSO being supported, the router can only operate in “helper” mode to synchronize state with a device that is actively switching over. Enabling SSO is a requirement to achieve literal “nonstop forwarding” with LDP, which we cannot test in a virtual environment. I begin by enabling the feature on all P and PE routers.

```
! CSR1, CSR6, CSR7, CSR8, CSR9, CSR10
mpls ldp graceful-restart

! XRv1, XRv2, and XRv3
mpls ldp
 graceful-restart
```

When applying these changes, XE warns us that GR will not apply for existing LDP sessions, but only new ones. I manually clear all LDP peers on all LDP GR routers to avoid any confusion. XR automatically flaps all LDP peers when the feature is configured, so a manual clearing is not required.

```
R8(config)#mpls ldp graceful-restart
% Previously established LDP sessions may not have graceful restart
protection.
```

When LDP GR is negotiated, we can see the details on both platforms. This reveals several new timers that will be discussed later. For now, we can verify that all peers on CSR9 and XRv2 are GR-enabled, as expected. There are no obvious incompatibilities between XE and XR with this feature. The “connect count” on XRv2 simply counts the number of times the session forms; if a session is cleared, or a peer fails, this number will increment.

```
R9#show mpls ldp graceful-restart
LDP Graceful Restart is enabled
Neighbor Liveness Timer: 120 seconds
Max Recovery Time: 120 seconds
Forwarding State Holding Time: 600 seconds
Down Neighbor Database (0 records):
Graceful Restart-enabled Sessions:
  VRF default:
    Peer LDP Ident: 222.0.0.1:0, State: estab
    Peer LDP Ident: 222.0.0.8:0, State: estab
    Peer LDP Ident: 222.0.0.14:0, State: estab
```

```
RP/0/0/CPU0:XRv2#show mpls ldp graceful-restart
Forwarding State Hold timer : Not Running
```

```
GR Neighbors : 3
```

Neighbor ID	Up	Connect Count	Liveness Timer	Recovery Timer
222.0.0.1	Y	1	-	-
222.0.0.8	Y	2	-	-
222.0.0.10	Y	2	-	-

Alternatively, XE has a command to see some extra detail per-neighbor regarding GR. This lists all GR-enabled peers along with their reconnection timers of 2 minutes.

```
R8#show mpls ldp neighbor graceful-restart
Peer LDP Ident: 222.0.0.9:0; Local LDP Ident 222.0.0.8:0
  TCP connection: 222.0.0.9.12152 - 222.0.0.8.646
  State: Oper; Msgs sent/rcvd: 12/12; Downstream
  Up time: 00:03:10
  Graceful Restart enabled; Peer reconnect time (msecs): 120000
Peer LDP Ident: 222.0.0.12:0; Local LDP Ident 222.0.0.8:0
  TCP connection: 222.0.0.12.14292 - 222.0.0.8.646
  State: Oper; Msgs sent/rcvd: 12/19; Downstream
  Up time: 00:03:10
  Graceful Restart enabled; Peer reconnect time (msecs): 120000
Peer LDP Ident: 222.0.0.14:0; Local LDP Ident 222.0.0.8:0
  TCP connection: 222.0.0.14.43579 - 222.0.0.8.646
  State: Oper; Msgs sent/rcvd: 12/16; Downstream
  Up time: 00:03:06
  Graceful Restart enabled; Peer reconnect time (msecs): 120000
```

To test what this reconnection timer does, we can cause an LDP failure in the network. Clearing all peers on CSR10 triggers a GR event (not shown). If we quickly check XRv3 and CSR1, we can see that their recovery timers start counting down from 2 minutes. Although the peer session forms before 2 minutes, this timer effectively represents the grace period seen in other GR/NSF implementations. If the active RP failed, the session must be restored within those 2 minutes.

```
RP/0/0/CPU0:XRv3#show mpls ldp graceful-restart
Forwarding State Hold timer : Not Running
GR Neighbors : 2
```

Neighbor ID	Up	Connect Count	Liveness Timer	Recovery Timer
222.0.0.1	Y	1	-	-
222.0.0.10	Y	2	Not running	106 sec remaining

```
R1#show mpls ldp graceful-restart
LDP Graceful Restart is enabled
Neighbor Liveness Timer: 120 seconds
Max Recovery Time: 120 seconds
Forwarding State Holding Time: 600 seconds
Down Neighbor Database (1 records):
VRF default:
  Peer LDP Ident: 222.0.0.10:0 [inst 2], Local LDP Ident: 222.0.0.1:0
  Status: recovering (25 seconds left)
  Address list contains 0 addresses:
Graceful Restart-enabled Sessions:
VRF default:
  Peer LDP Ident: 222.0.0.13:0, State: estab
  Peer LDP Ident: 222.0.0.9:0, State: estab
  Peer LDP Ident: 222.0.0.12:0, State: estab
  Peer LDP Ident: 222.0.0.10:0, State: estab
```

XE also generates log messages to show that the session failed and that GR begins its 2 minute countdown. After 2 minutes, the process completes.

```
! CSR1
21:07:49.534: %LDP-5-GR: GR session 222.0.0.10:0 (inst 2): interrupted--
recovery pending
21:07:52.995: %LDP-5-GR: GR session 222.0.0.10:0 (inst 5): starting graceful
recovery
21:09:52.995: %LDP-5-GR: GR session 222.0.0.10:0 (inst 5): completed graceful
recovery
```

The LDP GR timers are not intuitively named. The “neighbor-liveness” timer is the equivalent of the grace period, while the “max-recovery” timer determines how long to continue using stale label bindings after an LDP session is established. A third timer, the “forwarding-holding” timer, determines how long label bindings should be stored after a control plane resets. This sounds very similar to the “max-recovery” timer but begins counting down at the time of failure, not restoration. The defaults for the first two timers is 2 minutes, and the default for “forwarding-holding” is 10 minutes. To test these timers, I adjust them on CSR7 first.

```
! CSR7
mpls ldp graceful-restart timers neighbor-liveness 30
```

```
mpls ldp graceful-restart timers max-recovery 60
mpls ldp graceful-restart timers forwarding-holding 120
```

Before testing anything, we verify that the timers were accepted by the router. CSR7 still has 3 LDP neighbors and adjusting the timers did not cause them to flap.

```
R7#show mpls ldp graceful-restart
LDP Graceful Restart is enabled
Neighbor Liveness Timer: 30 seconds
Max Recovery Time: 60 seconds
Forwarding State Holding Time: 120 seconds
Down Neighbor Database (0 records):
Graceful Restart-enabled Sessions:
  VRF default:
    Peer LDP Ident: 222.0.0.6:0, State: estab
    Peer LDP Ident: 222.0.0.14:0, State: estab
    Peer LDP Ident: 222.0.0.11:0, State: estab
```

Next, I clear all of CSR7's neighbors, then check their status a few seconds later. We cannot adjust the reconnection timer of 2 minutes, but we see that the max-recovery timer begins counting down as soon as the session is established. I color-code the peers for consistency with other tests.

```
R7#show mpls ldp neighbor graceful-restart
Peer LDP Ident: 222.0.0.6:0; Local LDP Ident 222.0.0.7:0
  TCP connection: 222.0.0.6.646 - 222.0.0.7.21879
  State: Oper; Msgs sent/rcvd: 11/6; Downstream
  Up time: 00:00:17
  Graceful Restart enabled; Peer reconnect time (msecs): 120000
  Down Neighbor Information:
    Status: recovering (42 seconds left)
    Address list contains 0 addresses
Peer LDP Ident: 222.0.0.14:0; Local LDP Ident 222.0.0.7:0
  TCP connection: 222.0.0.14.49261 - 222.0.0.7.646
  State: Oper; Msgs sent/rcvd: 11/13; Downstream
  Up time: 00:00:14
  Graceful Restart enabled; Peer reconnect time (msecs): 120000
  Down Neighbor Information:
    Status: recovering (45 seconds left)
    Address list contains 0 addresses
Peer LDP Ident: 222.0.0.11:0; Local LDP Ident 222.0.0.7:0
  TCP connection: 222.0.0.11.20174 - 222.0.0.7.646
  State: Oper; Msgs sent/rcvd: 11/8; Downstream
  Up time: 00:00:14
  Graceful Restart enabled; Peer reconnect time (msecs): 120000
  Down Neighbor Information:
    Status: recovering (45 seconds left)
    Address list contains 0 addresses
```

Looking at the specific timestamps, we can see exactly 60 seconds pass between the time a peer is established to the time the graceful recovery completes. This removes the stale label bindings from the peer since it is expected that new bindings can be exchanged.

```
! CSR7
21:27:48.559: %LDP-5-GR: GR session 222.0.0.6:0 (inst 4): starting graceful
recovery
21:27:48.559: %LDP-5-NBRCHG: LDP Neighbor 222.0.0.6:0 (4) is UP

21:27:50.958: %LDP-5-GR: GR session 222.0.0.14:0 (inst 5): starting graceful
recovery
21:27:50.958: %LDP-5-NBRCHG: LDP Neighbor 222.0.0.14:0 (5) is UP

21:27:51.699: %LDP-5-GR: GR session 222.0.0.11:0 (inst 6): starting graceful
recovery
21:27:51.699: %LDP-5-NBRCHG: LDP Neighbor 222.0.0.11:0 (6) is UP

21:28:48.559: %LDP-5-GR: GR session 222.0.0.6:0 (inst 4): completed graceful
recovery
21:28:50.959: %LDP-5-GR: GR session 222.0.0.14:0 (inst 5): completed graceful
recovery
21:28:51.698: %LDP-5-GR: GR session 222.0.0.11:0 (inst 6): completed graceful
recovery
```

To see the “neighbor-liveness” timer in action, I clear all LDP peers on CSR7 again. I immediately (within 5 seconds) check the GR status to see that 2 of the neighbors are still down, and one has already recovered. The recovering router has already started its “max-recovery” time, which begins after the session establishes. The liveness timer begins counting down when the peer fails, which is still counting down for XRv1 and XRv4. When the peer is still waiting for a connection, all addresses bound to that LDP are shown in this list until the session reforms.

```
R7#show mpls ldp graceful-restart
LDP Graceful Restart is enabled
Neighbor Liveness Timer: 30 seconds
Max Recovery Time: 60 seconds
Forwarding State Holding Time: 120 seconds
Down Neighbor Database (3 records):
  VRF default:
    Peer LDP Ident: 222.0.0.6:0 [inst 4], Local LDP Ident: 222.0.0.7:0
      Status: recovering (57 seconds left)
      Address list contains 0 addresses:
    Peer LDP Ident: 222.0.0.11:0 [inst 6], Local LDP Ident: 222.0.0.7:0
      Status: waiting for reconnection (25 seconds left)
      Address list contains 3 addresses:
        222.6.11.11      222.7.11.11      222.0.0.11
    Peer LDP Ident: 222.0.0.14:0 [inst 5], Local LDP Ident: 222.0.0.7:0
      Status: waiting for reconnection (25 seconds left)
```



```
Address list contains 5 addresses:
    222.14.9.14      222.7.14.14      222.8.14.14      222.9.14.14
    222.0.0.14
```

Graceful Restart-enabled Sessions:

VRF default:

Peer LDP Ident: 222.0.0.6:0, State: estab

Seconds later, the remaining LDP sessions to XRv1 and XRv4 come up. The output is not shown again as we know the max-recovery timers begin counting down from 60 seconds after the session forms. Next, we will examine the available timers on XR. XR allows us to configure the reconnection-timeout, which is equivalent to the XE neighbor-liveness timer. This specifies how long to wait for the peer to return after a failure. The recovery timer is fixed at 2 minutes where as XE allowed us to adjust this (max-recovery). The forwarding-state-holdtime behaves identically to the forwarding-holding timer.

```
! XRv2
mpls ldp
 graceful-restart reconnect-timeout 60
 graceful-restart forwarding-state-holdtime 150
```

Clearing the LDP peers on XRv2 and immediately looking at the LDP GR status, we can see a similar output as we did on XE. CSR8 comes up quickly and begins counting down from 60, which mimics the max-recovery timer, except it is fixed at 2 minutes. The other peers that are still down, CSR1 and CSR10, count down from the reconnection-timeout of 60 seconds.

```
RP/0/0/CPU0:XRv2#show mpls ldp graceful-restart
Forwarding State Hold timer : Not Running
GR Neighbors                  : 3
Neighbor ID                   Up   Connect Count  Liveness Timer  Recovery Timer
-----
222.0.0.1                     N    1              52 sec remaining  Not running
222.0.0.8                     Y    3              Not running      118 sec remaining
222.0.0.10                    N    2              52 sec remaining  Not running
```

Last, we examine LDP NSR. Like the other NSR variants, this is used to mask RP failures on a router by handling all of the switchover locally. This can be used in environments where LDP GR is not supported by all platforms. The LDP sessions will not flap during switchover and the entire operation will appear transparent to all LDP peers.

```
! CSR10
mpls ldp nsr
```

```
! XRv3
mpls ldp
 nsr
```

We can verify the NSR configuration by checking the summary information on both routers. Interestingly, CSR10 shows XRv3 as not ready, but XRv3 shows CSR10 as ready. I assume this is an artificiality of the virtual environment.

```
R10#show mpls ldp nsr
LDP Non-Stop Routing is enabled
LDP Non-Stop Routing Sessions:
  VRF default:
    Peer LDP Ident: 222.0.0.1:0      State: Not Ready
    Peer LDP Ident: 222.0.0.12:0     State: Not Ready
    Peer LDP Ident: 222.0.0.13:0     State: Not Ready
```

```
RP/0/0/CPU0:XRv3#show mpls ldp nsr summary
Sessions:
  Total: 2, NSR-eligible: 2, Sync-ed: 0
  (2 Ready)
```

Like the other NSR implementations, we can query the routers for their NSR exchanges with their peers. Although NSR is not a negotiated capability like the GR/NSF family of features, statistics are still tracked per peer.

```
R10#show mpls ldp nsr statistics neighbor 222.0.0.13
Peer: 222.0.0.13:0
  In label Request Records created: 0, freed: 0
  In label Withdraw Records created: 0, freed: 0
  Local Address Withdraw Set: 0, Cleared: 0
  Transmit contexts enqueued: 0, dequeued: 0
```

```
RP/0/0/CPU0:XRv3#show mpls ldp nsr statistics neighbor 222.0.0.10
Peer: 222.0.0.10:0
  State: Ready for TCP sync
  Messages: 0
  Init Sync:
    Start: MON  6 03:56:14 (18:01:31 ago)
    End:   MON  6 03:56:14 (18:01:31 ago)
    Protocol: 0/0 sent/rcvd Capabilities,
              0 peer addr, 0 peer label
              0 rx-buf bytes, 0 app-data bytes
  Steady State Sync:
    Capabilities: 0 sent, 0 rcvd
    Send-Ack:0 In Lbl-WD, 0 In Lbl-Req
    Stdbdy Adj: 0 join, 0 leave
```

14.1.6 RSVP-TE GR

RSVP-TE supports GR but not NSR. RSVP GR applies only to TE applications, not ordinary IP, and is used to maintain state information when a node in the LSP performs an RP switchover. Below, I configure RSVP GR on XRv3 and CSR10. I also enable it on CSR1 and XRv2 so that all interfaces within the OSPFv2

aggregation IGP are using this feature, but those configurations are not shown for brevity. I make minor adjustments to the hello and “miss” interval; this specifies how frequently RSVP hello messages are sent. Additionally, I enable RSVP GR under all interfaces also enabled for OSPF2.

```
! XRv3
rsvp
 interface GigabitEthernet0/0/0/0.503
   signalling hello graceful-restart interface-based

 interface GigabitEthernet0/0/0/0.513
   signalling hello graceful-restart interface-based

 signalling graceful-restart
 signalling hello graceful-restart refresh misses 5
 signalling hello graceful-restart refresh interval 6000

! CSR10
interface GigabitEthernet2.502
 ip rsvp signalling hello graceful-restart

interface GigabitEthernet2.503
 ip rsvp signalling hello graceful-restart

interface GigabitEthernet2.510
 ip rsvp signalling hello graceful-restart

ip rsvp signalling hello graceful-restart mode full
ip rsvp signalling hello graceful-restart refresh interval 6000
ip rsvp signalling hello graceful-restart refresh misses 5
```

Just like TE-FRR, nothing happens until a TE tunnel is configured. RSVP is like BFD in a way where it does not have a discovery mechanism; RSVP does have hellos, but they are used for specific applications.

```
RP/0/0/CPU0:XRv2#show rsvp graceful-restart neighbors
[no output]
```

At a minimum, we can at least ensure the feature was configured properly before configuring any TE tunnels. XRv2 and CSR1 both indicate that RSVP GR is enabled with the correct timers.

```
RP/0/0/CPU0:XRv2#show rsvp graceful-restart
Graceful restart: enabled   Number of global neighbors: 0
Restart time: 120 seconds  Recovery time: 120 seconds
Recovery timer: Not running
Hello interval: 6000 milliseconds  Maximum Hello miss-count: 5
Pending states: 0

R1#show ip rsvp hello
```

Hello:

```
RSVP Hello for Fast-Reroute/Reroute: Disabled
Statistics: Disabled
BFD for Fast-Reroute/Reroute: Disabled
RSVP Hello for Graceful Restart: Enabled (full mode)
```

R1#show ip rsvp hello graceful-restart

```
Graceful Restart: Enabled (full mode)
Refresh interval: 6000 msecs
Refresh misses: 5
DSCP: 0x30
Advertised restart time: 5 msecs
Advertised recovery time: 0 msecs
Maximum wait for recovery: 3600000 msecs
```

Next, I configure a TE tunnel on XRv3 to route traffic to CSR1 over any link except the one directly connected to CSR1. This means that CSR10 will always be in the transit path. After configuring the tunnel, we verify that it forms correctly. The tunnel comes up and routes via CSR10 to CSR1 as expected. There is no need to route any traffic into the tunnel for this test.

```
! XRv3
explicit-path name EP_AVOID_DIRECT
index 10 exclude-address ipv4 unicast 222.1.13.1
index 20 exclude-address ipv4 unicast 222.1.13.13
```

```
interface tunnel-te100
ipv4 unnumbered Loopback0
destination 222.0.0.1
path-option 10 explicit name EP_AVOID_DIRECT
```

RP/0/0/CPU0:XRv3#show mpls traffic-eng tunnels 100 brief

TUNNEL NAME	DESTINATION	STATUS	STATE
tunnel-te100	222.0.0.1	up	up

Displayed 1 (of 1) heads, 0 (of 0) midpoints, 0 (of 0) tails
Displayed 1 up, 0 down, 0 recovering, 0 recovered heads

RP/0/0/CPU0:XRv3#show mpls traffic-eng tunnels 100 detail | begin Path Info

```
Path Info:
Outgoing:
Explicit Route:
Strict, 222.10.13.10
Strict, 222.1.10.1
Strict, 222.0.0.1
```

[snip]

We know that the RSVP RESV message travels from tail to head which binds RSVP labels at each hop for the TE tunnel being signaled. RSVP GR hello requests are originated at each upstream router to establish GR connectivity and a hello acknowledgement is expected in return from the downstream router. That is

to say, the hello initiation and acknowledge travels in the same direction as the RSVP PATH message. Below, we see that CSR1 received two sets of hello request messages. These are traveling down the LSP (head to tail) and originate from CSR10. One set of hellos uses TTL=255 so that the failure of an intermediate node will not prevent the RSVP hello from arriving. This session is between node IDs, much like an LDP session would be. The normal, link-based RSVP session still has TTL=1 as expected. These hellos are different than normal RSVP hellos as they contain the RESTART capability, which is supported in XR (unlike regular RSVP hellos). We will examine these two different “neighbors” later. Also, note that the recovery time is 0 (green) and restart time is 5 msec (cyan).

```
R1#debug ip rsvp dump-messages
RSVP message dump debugging is on
```

Incoming Hello:

```
version:1 flags:0000 cksum:C054 ttl:255 reserved:0 length:32
HELLO type HELLO REQUEST length 12:
  Src_Instance: 0x658183D5, Dst_Instance: 0x909D1D62
  RESTART_CAP type 1 length 12:
    Restart_Time: 0x00000005, Recovery_Time: 0x00000000
```

Outgoing Hello:

```
version:1 flags:0000 cksum:C053 ttl:255 reserved:0 length:32
HELLO type HELLO ACK length 12:
  Src_Instance: 0x909D1D62, Dst_Instance: 0x658183D5
  RESTART_CAP type 1 length 12:
    Restart_Time: 0x00000005, Recovery_Time: 0x00000000
```

Incoming Hello:

```
version:1 flags:0000 cksum:BE55 ttl:1 reserved:0 length:32
HELLO type HELLO REQUEST length 12:
  Src_Instance: 0x658183D5, Dst_Instance: 0x909D1D62
  RESTART_CAP type 1 length 12:
    Restart_Time: 0x00000005, Recovery_Time: 0x00000000
```

Outgoing Hello:

```
version:1 flags:0000 cksum:BE54 ttl:1 reserved:0 length:32
HELLO type HELLO ACK length 12:
  Src_Instance: 0x909D1D62, Dst_Instance: 0x658183D5
  RESTART_CAP type 1 length 12:
    Restart_Time: 0x00000005, Recovery_Time: 0x00000000
```

XRv3 generates a syslog message when the RSVP neighbor session is formed to CSR10. Notice that CSR10's node ID is specified, not the link ID. This is because RSVP GR only supports node failures, so it tracks the node address.

```
! XRv3
rsvp[1049]: %ROUTING-RSVP-6-RSVP_HELLO_STATE_UP : A graceful restart hello
session to 222.0.0.10 has been established
```

Starting from the headend, we will verify the RSVP GR neighbors. I show a summary and detailed view at each router along the way. Of note, XRv3 always identifies CSR10 by its node address. Hellos are successfully being sent and received, which indicates a functional configuration.

```
RP/0/0/CPU0:XRv3#show rsvp graceful-restart neighbors
Neighbor      App  State Recovery      Reason      Since      LostCnt
-----
222.0.0.10    MPLS  UP      DONE      N/A 06/12/2045 22:39:35      0
```

```
RP/0/0/CPU0:XRv3#show rsvp hello instance detail
Neighbor: 222.0.0.10 Source: 222.0.0.13 (MPLS)
State: UP (for 00:05:48)
Type: ACTIVE (sending requests)
Last Hello received on GigabitEthernet0/0/0/0.503
Hello interval (msec) (used when ACTIVE)
Configured: 6000
Src_instance 0x883a58, Dst_instance 0x658183d5
Counters:
Communication with neighbor lost:
Num of times: 0
Reasons:
Missed acks: 0
New Src_Instance received: 0
New Dst_Instance received: 0
I/f went down: 0
Neighbor disabled Hello: 0
Msgs Received: 60
Sent: 60
Suppressed: 0
```

CSR10 is a midpoint along this LSP. It shows a single LSP passing through it from 222.0.0.13 to 222.0.0.1. The details of this LSP specify the upstream and downstream nodes, which are highlighted.

```
R10#show ip rsvp hello client lsp summary
Local      Remote      tun_id lsp_id subgrp_orig      subgrp_id FLAGS
222.0.0.13 222.0.0.1   100    2      0.0.0.0          0      0xC038
```

```
R10#show ip rsvp hello client lsp detail
Hello Client LSPs (all lsp tree)
Tun Dest: 222.0.0.1 Tun ID: 100 Ext Tun ID: 222.0.0.13
Tun Sender: 222.0.0.13 LSP ID: 2
Lsp flags: 0xC038
Lsp GR UP Node nbr: 222.0.0.13 Intf nbr: 222.10.13.13
Lsp GR DN Node nbr: 222.0.0.1 Intf nbr: 222.1.10.1
```

CSR1 is the tail end of this TE tunnel. Notice that it has two neighbors. The first is the TTL=225 “multi-hop” session between the routers while the other uses the directly connected interfaces at TTL=1. This is

consistent with the debug evaluated earlier. Although GR cannot protect against link failures, the multi-hop adjacency allows the router to continue communicating, which is the reason why a second session is made. The details reveal that the single-hop session is bound to a specific interface, while the multi-hop session is not.

```
R1#show ip rsvp hello client nbr summary
Local      Remote      Type      NBR_STATE  HI_STATE  LSPs
222.0.0.1  222.0.0.10   GR        Normal     Up        1
222.1.10.1 222.1.10.10  GR        Normal     Up        1
```

```
R1#show ip rsvp hello client nbr detail
Hello Client Neighbors
  Remote addr 222.0.0.10, Local addr 222.0.0.1
  Nbr State: Normal      Type: Graceful Restart
  Nbr Hello State: Up
  LSPs protecting: 1
  I/F: Any

  Remote addr 222.1.10.10, Local addr 222.1.10.1
  Nbr State: Normal      Type: Graceful Restart
  Nbr Hello State: Up
  LSPs protecting: 1
  I/F: Gi2.510
```

To simulate a failure of CSR10, I shut down its physical interface, effectively removing it from the network entirely. Though this is not an appropriate way to simulate an RP switchover, it allows us to see how XRv3 will react. After 5 missed hellos, the neighbor is torn down and a syslog is generated. XRv3 will track the number of times the neighbor fails, along with other timestamp-related statistics.

```
! XRv3
rsvp[1049]: %ROUTING-RSVP-6-RSVP_HELLO_STATE_DOWN : Graceful restart hello
session to 222.0.0.10 has changed to the down state
```

```
RP/0/0/CPU0:XRv3#show rsvp hello instance detail
Neighbor: 222.0.0.10 Source: 222.0.0.13 (MPLS)
  State: DOWN (for 00:00:13)
  Type: ACTIVE (sending requests)
  Last Hello received on GigabitEthernet0/0/0/0.503
  Hello interval (msec) (used when ACTIVE)
  Configured: 6000
  Src_instance 0x883a58, Dst_instance 0x0
  Counters:
  Communication with neighbor lost:
    Num of times: 1 (last time at 00:00:13 ago for reason HELLO MISSED)
  Reasons:
    Missed acks: 1
    New Src_Instance received: 0
```

```

New Dst_Inst received: 0
I/f went down: 0
Neighbor disabled Hello: 0
Msgs Received: 360
Sent: 368
Suppressed: 0

```

Before continuing, I restore CSR10 and ensure that it forms its RSVP neighbors again. A total of 4 neighbors are shown: 2 single-hop sessions (yellow) and 2 multi-hop sessions (green), one of each for CSR1 and XRv3. Despite GR being configured, all routers immediately purged all TE LSP information when CSR10 failed.

```
R10#show ip rsvp hello client nbr
```

Local	Remote	Type	NBR_STATE	HI_STATE	LSPs
222.0.0.10	222.0.0.1	GR	Normal	Up	1
222.0.0.10	222.0.0.13	GR	Normal	Up	1
222.10.13.10	222.10.13.13	GR	Normal	Up	1
222.1.10.10	222.1.10.1	GR	Normal	Up	1

The restart time and recovery time observed earlier should change this behavior. A recovery time of 0 indicates that a router has no ability to preserve its forwarding information. This causes RSVP peers adjacent to a failed node to immediately delete all LSP information through that peer. A number greater than zero causes routers on either side of the failure to continue sending RSVP PATH messages (if downstream from the failed node) or RSVP RESV messages (if upstream from the failed node) so that the failure is masked. These soft-state refreshes are imperative to normal RSVP operation. The restart timer specifies how long a peer should wait for the session to establish, and the recovery time specifies how long it should take to synchronize the label forwarding constructs. XE only lets you adjust the recovery time, but XR allows you to adjust both. I use 90 seconds for both timers. XE has a hidden command for the restart time, but it doesn't even stick in the configuration. It is included below for reference.

```

! CSR1 and CSR10
ip rsvp signalling hello graceful-restart send recovery-time 90000
ip rsvp signalling hello graceful-restart send restart-time 90000 ! HIDDEN

! XRv2 and XRv3
rsvp
 signalling graceful-restart restart-time 90
 signalling graceful-restart recovery-time 90

```

Using debug on CSR1, we can verify that the timers have changed. 0x15F90 in decimal is 90,000 which represents the time in msec.

```

R1#debug ip rsvp dump-messages hello
RSVP message dump debugging is on

```


Incoming Hello:

```
version:1 flags:0000 cksum:0137 ttl:255 reserved:0 length:32
HELLO type HELLO REQUEST length 12:
Src_Instance: 0x658183D5, Dst_Instance: 0x909D1D62
RESTART_CAP type 1 length 12:
Restart_Time: 0x00015F90, Recovery_Time: 0x00015F90
```

Outgoing Hello:

```
version:1 flags:0000 cksum:0136 ttl:255 reserved:0 length:32
HELLO type HELLO ACK length 12:
Src_Instance: 0x909D1D62, Dst_Instance: 0x658183D5
RESTART_CAP type 1 length 12:
Restart_Time: 0x00015F90, Recovery_Time: 0x00015F90
```

Incoming Hello:

```
version:1 flags:0000 cksum:FF37 ttl:1 reserved:0 length:32
HELLO type HELLO REQUEST length 12:
Src_Instance: 0x658183D5, Dst_Instance: 0x909D1D62
RESTART_CAP type 1 length 12:
Restart_Time: 0x00015F90, Recovery_Time: 0x00015F90
```

Outgoing Hello:

```
version:1 flags:0000 cksum:FF36 ttl:1 reserved:0 length:32
HELLO type HELLO ACK length 12:
Src_Instance: 0x909D1D62, Dst_Instance: 0x658183D5
RESTART_CAP type 1 length 12:
Restart_Time: 0x00015F90, Recovery_Time: 0x00015F90
```

We can also use show commands. CSR10 and XRv3 both indicate the proper values. Ideally, this should give CSR10 plenty of time to recover from another failure.

```
R10#show ip rsvp hello graceful-restart
```

```
Graceful Restart: Enabled (full mode)
Refresh interval: 6000 msec
Refresh misses: 5
DSCP: 0x30
Advertised restart time: 90000 msec
Advertised recovery time: 90000 msec
Maximum wait for recovery: 3600000 msec
```

```
RP/0/0/CPU0:XRv3#show rsvp graceful-restart
```

```
Graceful restart: enabled Number of global neighbors: 1
Local MPLS router id: 222.0.0.13
Restart time: 90 seconds Recovery time: 90 seconds
Recovery timer: Not running
Hello interval: 6000 milliseconds Maximum Hello miss-count: 5
Pending states: 0
```

Since shutting down interfaces is not a valid test of an RP switchover, this lab cannot be tested further. The test results on CSR10 appear to be the same even with these new timer configurations. I was not able to see any noticeably different, however I was able to capture some new output on CSR1. We can see that when CSR10 fails, CSR1 does change the state to “Restarting” which implies, for a short time, CSR1 believed CSR10 was performing SSO. This did not last the entire 90 seconds because when IGP reachability was lost, the peer failed entirely (even though IGP converged faster than 90 seconds). I theorize that a real RP failure would have caused this behavior to reflect the timer adjustments configured earlier.

```
R1#show ip rsvp hello client nbr
```

Local	Remote	Type	NBR_STATE	HI_STATE	LSPs
222.0.0.1	222.0.0.10	GR	Restarting	Lost	1
222.1.10.1	222.1.10.10	GR	Restarting	Lost	1

14.1.7 EIGRP NSF

Like RSVP, EIGRP supports GR/NSF but not NSR. The concept is identical to all other GR/NSF features discussed so far. A router undergoing a switchover requires peers to “help” it synchronize its standby RP during an SSO event. In the case of EIGRP, it also assists with stuck-in-active (SIA) events where an “active” route has unanswered queries from at least one peer regarding its reachability status. When a restart operation begins on an EIGRP router, the restart bit (RS-bit) is set in the hello packets towards all peers. Those who receive and understand this message immediately send their EIGRP topology tables to the restarting router. When complete, an end-of-table (EoT) message is sent to signal the end of the topology transfer. Once the switching router is finished converging, it uses the EoT message towards the “helpers” to signify that convergence is complete. Interestingly, NSF is enabled by default on XR. No other NSF features are enabled by default for any other protocol on XR or XE. We can confirm this with some show commands.

```
RP/0/0/CPU0:XRv3#show protocols ipv4 eigrp vrf EIGRP | include NSF
```

```
EIGRP NSF: enabled
```

```
NSF-aware route hold timer is 480s
```

```
NSF signal timer is 20s
```

```
NSF converge timer is 300s
```

```
RP/0/0/CPU0:XRv3#show protocols ipv6 eigrp vrf EIGRP | include NSF
```

```
EIGRP NSF: enabled
```

```
NSF-aware route hold timer is 480s
```

```
NSF signal timer is 20s
```

```
NSF converge timer is 300s
```

Because we cannot “enable” NSF on XRv3, I will disable it from the IPv6 AFI. The word “disable” is the only option in the parser. After issuing this command, XRv3 issues a log message about degraded ISSU capabilities. We also verify the NSF has been fully disabled for IPv6 as a demonstration.

```

! XRv3
router eigrp VPN
  vrf EIGRP
    address-family ipv6
      nsf disable

! XRv3
eigrp[1003]: %ROUTING-EIGRP-4-ISSU_SUPPORT : EIGRP-VPN: EIGRP-v6 213: NSF
disabled. ISSU requires NSF to be enabled.

```

```

RP/0/0/CPU0:XRv3#show protocols ipv6 eigrp vrf EIGRP | include NSF
  EIGRP NSF: disabled
    NSF-aware route hold timer is 480s
    NSF signal timer is 20s
    NSF converge timer is 300s

```

To mirror XRv3's capabilities, CSR2 enables NSF for IPv4 only. We can verify this by checking the IPv4/v6 protocol details.

```

! CSR2
router eigrp VPN
  address-family ipv4 unicast autonomous-system 213
    nsf

```

```

R2#show ip protocols | include eigrp|NSF
*** IP Routing is NSF aware ***
Routing Protocol is "eigrp 213"
  NSF-aware route hold timer is 240
  EIGRP NSF enabled
    NSF signal timer is 20s
    NSF converge timer is 120s

```

```

R2#show ipv6 protocols | include eigrp|NSF
IPv6 Routing Protocol is "eigrp 213"
  NSF-aware route hold timer is 240
  EIGRP NSF disabled
    NSF signal timer is 20s
    NSF converge timer is 120s

```

As a quick test, I enabled NSF debugging on CSR2 and clear the EIGRP neighbor table on XRv3 (not shown). CSR2 cannot execute NSF because the neighbor was "hard cleared", which means that administrator was intentionally trying to reset the control-plane. At a minimum, this shows that EIGRP NSF is operational as the feature "tried to" work. We can see that EoT was received from XRv3 which indicates convergence is complete; this is carried as a special flag inside of EIGRP packets.

```

R2#debug eigrp nsf
EIGRP NSF debugging is on

```

```

EIGRP: NSF: AS213: Checking if Graceful Restart is possible with neighbor
10.2.13.13, peer_down reason 'Interface PEER-TERMINATION received'
EIGRP: NSF: Not possible: 'peer_down was called with a HARD resync flag'
EIGRP: NSF: Enqueuing NULL update to 10.2.13.13, flags 0x1:(INIT)
EIGRP: NSF: AS213, Receive EOT from 10.2.13.13, Flags 0x8:(EOT)

```

Performing the same test in the opposite direction, XRv3 indicates similar results. NSF cannot be used, but the software tries to apply the feature. This is likely the result of having a single RP with SSO unsupported. The last line of this debug probably means that an EoT was received from CSR2 as this indicates convergence from an EIGRP neighbor.

```

RP/0/0/CPU0:XRv3#debug eigrp nsf
eigrp[1003]: IP-EIGRP(VPN-213): NSF: RIB Converged notification received for
VRF EIGRP-v4
eigrp[1003]: IPv4-EIGRP(VPN-213): NSF: Self NOT in NSF restart mode. Received
a non-RS INIT Update from peer 10.2.13.2
eigrp[1003]: IPv4-EIGRP(VPN-213): NSF: Received initial updates for the peer.
Will use NSF startup mode for the peer now on
eigrp[1003]: IPv4-EIGRP(VPN-213): NSF: Self NOT in NSF restart mode. Received
a non-RS non-INIT Update from peer 10.2.13.2
eigrp[1003]: IPv4-EIGRP(VPN-213): NSF: Self NOT in NSF restart mode. Received
a non-RS non-INIT Update from peer 10.2.13.2
eigrp[1003]: IP-EIGRP(VPN-213): NSF: RIB Converged notification received for
VRF EIGRP-v4

```

Last, we will examine the EIGRP NSF timers briefly. On XE, you actually use both GR and NSF syntax, which is confusing. They are effectively describing the same feature. In the outputs above, we saw the route hold timer, signal timer, and converge timer. The route hold-timer is configured as the GR purge-time and determines how long EIGRP will keep routes for an inactive peer if EoT is not received. The signal timer indicates how long to wait after sending packets with the RS-bit set, which was described earlier. The converge timer indicates how long to wait before receiving the EoT message from all neighbors, which is similar to the BGP EoR marker. These minor details should not be adjusted unless there is a compelling reason. I adjust these to new values for variety, then confirm them with show commands on both CSR2 and XRv3.

```

! CSR2
router eigrp VPN
 address-family ipv4 unicast autonomous-system 213
   timers nsf signal 25
   timers nsf converge 100
   timers graceful-restart purge-time 150

! XRv3
router eigrp VPN
 vrf EIGRP
   address-family ipv4
     timers nsf route-hold 150

```

```
timers nsf signal 25
timers nsf converge 100
```

```
R2#show ip protocols | include eigrp|NSF
```

```
*** IP Routing is NSF aware ***
```

```
Routing Protocol is "eigrp 213"
```

```
NSF-aware route hold timer is 150
```

```
EIGRP NSF enabled
```

```
NSF signal timer is 25s
```

```
NSF converge timer is 100s
```

```
RP/0/0/CPU0:XRv3#show protocols eigrp vrf EIGRP | include NSF
```

```
EIGRP NSF: enabled
```

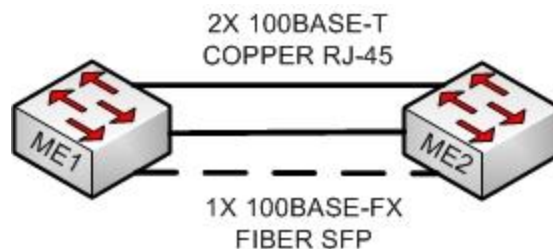
```
NSF-aware route hold timer is 150s
```

```
NSF signal timer is 25s
```

```
NSF converge timer is 100s
```

15. Describe Layer 1 failure detection

This section will be brief and covers only two features: carrier delay and link debounce. The network diagram is shown below, which uses two Cisco ME-3400-24TS-A switches with 3 links between them. Two are 100 Mbps copper NNIs and one is a 100BaseFX 100 Mbps fiber NNI. There is no link bundling.



Carrier delay is a timer that runs in software to identify when a link goes down at layer 1. This is normally set to 2 seconds to prevent against very short flaps, which may introduce churn. This slows convergence significantly, so we can adjust it to a smaller value. We can specify a value in seconds or ms, and we reduce this to 0 on ME1. The interface will go down as soon as the cable is unplugged on ME1, but on ME2, this will occur 2 seconds later. On most IOS platforms, 2 seconds is the default carrier delay (not revealed in the show command).

```
! ME1
interface FastEthernet0/15
carrier-delay msec 0
```

```
ME1#show interfaces fastEthernet 0/15 | include Carrier
```

```
Carrier delay is 0 msec
```

Before continuing, we configure basic NTP so we can use the logging timestamps to approximate the carrier delay changes. We verify the clocks are synchronized; see the NTP section for more details.

```
! ME1
ntp master
```

```
! ME2
ntp server 10.0.10.91
```

```
ME1#show ntp status | include sync
Clock is synchronized, stratum 8, reference is 127.127.1.1
```

```
ME2#show ntp status | include sync
Clock is synchronized, stratum 9, reference is 10.0.10.91
```

When the cable is unplugged, the link-light immediately turns off on both switches. However, there is a ~2 second delay between the log messages being issued. Since this is a software timer, the LEDs may turn off, but IOS doesn't notify any other process of the outage until the carrier delay expires. The timing of the syslog generation is imperfect which is why we only see a ~1.3 second delay in this case.

```
! ME1
03:45:32.510: %LINK-3-UPDOWN: Interface FastEthernet0/15, changed state to
down
03:45:33.500: %LINEPROTO-5-UPDOWN: Line protocol on Interface
FastEthernet0/15, changed state to down
```

```
! ME2
03:45:33.894: %LINEPROTO-5-UPDOWN: Line protocol on Interface
FastEthernet0/15, changed state to down
03:45:34.892: %LINK-3-UPDOWN: Interface FastEthernet0/15, changed state to
down
```

The carrier delay is synchronous by default and also controls how long to wait before declaring a link up event. Plugging the cable back in proves this, and this time is much closer to 2 seconds than the link-down time, which is somewhat coincidental.

```
! ME1
03:47:37.542: %LINK-3-UPDOWN: Interface FastEthernet0/15, changed state to up
03:47:38.549: %LINEPROTO-5-UPDOWN: Line protocol on Interface
FastEthernet0/15, changed state to up
```

```
! ME2
03:47:39.524: %LINK-3-UPDOWN: Interface FastEthernet0/15, changed state to up
03:47:40.531: %LINEPROTO-5-UPDOWN: Line protocol on Interface
FastEthernet0/15, changed state to up
```

We quickly compare this to port 16, which has the default carrier-delay on both switches. The down event happens at the same time.

```
! ME1
04:25:28.414: %LINEPROTO-5-UPDOWN: Line protocol on Interface
FastEthernet0/16, changed state to down
04:25:29.420: %LINK-3-UPDOWN: Interface FastEthernet0/16, changed state to
down
```

```
! ME2
04:25:28.389: %LINEPROTO-5-UPDOWN: Line protocol on Interface
FastEthernet0/16, changed state to down
04:25:29.395: %LINK-3-UPDOWN: Interface FastEthernet0/16, changed state to
down
```

Plugging port 16 back in, the up event happens at the same time also. This proves that the carrier delay is actually evaluated on the ME3400 platform, and since its running in software, should be platform-agnostic.

```
! ME1
04:25:50.929: %LINK-3-UPDOWN: Interface FastEthernet0/16, changed state to up
04:25:51.935: %LINEPROTO-5-UPDOWN: Line protocol on Interface
FastEthernet0/16, changed state to up

! ME2
04:25:50.946: %LINK-3-UPDOWN: Interface FastEthernet0/16, changed state to up
04:25:51.952: %LINEPROTO-5-UPDOWN: Line protocol on Interface
FastEthernet0/16, changed state to up
```

Rushing into a link-up event can be dangerous, so on some platforms we can also configure asynchronous carrier delays. The ME3400 is not one of them, which means adjusting carrier delay applies in both up and down directions at all times. For that reason, I would recommend a non-zero value, even for faster convergence, since it may lead to short-term blackholes on link-up events. An example of an “up” delay is shown below.

```
! Example (not configured on ME3400)
interface FastEthernet0/15
  carrier-delay up msec 500
```

Link debounce is not supported in the ME3400, but is used for link-down events only. It runs in firmware, not software, so it is a function of the interface driver. The default is 300 ms on copper and 10 ms on fiber interfaces, and Cisco recommends keeping it as the default. The timer is measured in ms.

```
! Example (not configured on ME3400)
interface FastEthernet0/15
  link debounce time 500
```

Additional Reading – Reference configurations “layer1-fail”

16. Describe BGPsec

This feature is discussed/tested in detail in the security section. In summary, the feature encompasses AS origin validation by way of Resource Public Key Infrastructure (RPKI). The word “resource” relates to IP resources, like prefixes, where PKI relates to existing X.509 PKI certificate infrastructure. AS origin validation seeks to ensure that only prefixes registered in a certain AS can be originated from that AS, and is most useful to prevent accidental advertisements. Attackers can easily craft BGP prefixes with AS-paths to overcome this, and BGPsec aims to stop it by having a router in each AS (only makes sense with eBGP) sign the prefix to increase its authenticity. A Resource Origination Authorization (ROA) is what binds an address space with an AS number, granting permission to that ASN to advertise prefixes in accordance with the ROA. Multiple ROAs can exist, and they may overlap; routers will consider the union of all valid ROAs when determining prefix validity. Both certificates and ROAs are published in publicly accessible repositories for general use. The time-range validity of these constructs is always relevant; if the current time does not fit within the lifetime of an ROA’s or certificate, that object cannot be used, much like a time-based key chain.

17. Describe backscatter traceback

This feature is discussed/tested in detail in the security section. In summary, the feature combines ACLs, remotely triggered blackholes (sometimes), sinkholes, and ICMP unreachables to determine the ASBR ingress routers from which an attack has entered the network. The backscatter is the forwarding of ICMP unreachables back into the core network and ultimately into a sinkhole for analysis. This is primarily employed to detect/defeat IP spoofing attacks against the SP network.

18. Describe lawful-intercept

This feature is discussed/tested in detail in the security section. In summary, this allows a mediation device to communicate to a router over SNMPv3 to setup/teardown wiretap sessions on a network device. This would be done in accordance with Government regulations and the mediation device would forward copies of “interesting” traffic back to law enforcement agencies (LEAs) that require it. Mediation devices are provided by third-party vendors and do most of the processing, but routers need to be enabled for SNMPv3 and must have the LI feature supported as required by law.

19. Describe BGP Flowspec

This feature is discussed/tested in detail in the security section. In summary, the feature is meant to add additional granularity to the remotely triggered black-hole (RTBH) feature. RTBH is a “big hammer” where it completely blackholes traffic to/from certain prefixes during a DDoS attack. BGP flowspec is like a combination of RTBH and QPPB; it can signal specific QoS actions, redirect traffic into a VRF for analysis, or drop the traffic. XE only supports it as a route-reflector (full support is introduced in version 3.15S), and XRv can only create flowspec rules; it cannot be tested in the data plane due to lacking hardware linecards.

20. Describe DDoS mitigation techniques

This feature is discussed/tested in detail in the security section. In summary, common tools include ACLs, remotely triggered black holes, unicast RPF, backscatter traceback, and BGP Flowspec.

21. Describe network event and fault management

This chapter encompasses a set of functions that detect, isolate, and correct malfunctions in the network. This includes accounting for environmental changes, examining logs and notifications, tracing/correcting faults, and running diagnostic tests. When network devices fail or suffer errors, an event-driven mechanism should be configured. For example, the device could send a syslog message to a remote syslog server or issue SNMP traps/informs to an SNMP server. This notifies the network staff of the fault and generally provides details regarding the fault. Once the condition that triggers the fault is resolved, the alarm state is cleared. Fault management systems may have internal mechanisms to map in-band severities (such as syslog priority 0, or emergencies) to another prioritization scheme based on network administrator weights. With respect to ITIL Incident Management, a fault would qualify as an incident, and is subject to the ITIL processes described in that section.

22. Describe performance management and capacity procedures

Capacity management: process of determining the network resources required to prevent a performance or availability impact on business-critical applications.

Performance management: practice of managing network service response time, consistency, and quality for individual and overall services.

Note: Performance problems are usually related to capacity; applications are slower due to congestion management and traffic conditioning (queuing and shaping).

Trending is the process of reviewing performance/capacity issues and reviewing the baselines for trends. This helps to understand future upgrade requirements.

Common areas for concern:

- a. **CPU.** Used by both control and data planes on any network device. For performance and capacity management, all devices must have sufficient CPU to service these needs. Insufficient CPU on one device can impact the entire network (BGP slow-peer, slow SPF computation blackholes, etc).
- b. **Backplane of I/O.** Refers to the total amount of traffic a device can handle. Insufficient backplane bandwidth results in dropped packets, leading to retransmissions and even more traffic on the network.
- c. **Memory and buffers.** Used by both control and data planes on any network device. Memory can often be low during periods of routing convergence.
- d. **Interface and pipe sizes.** Refers to the amount of data than be sent on any one connection. Often incorrectly referred to as speed. Congestion on these links usually results in queuing of sorts, which can increase latency.

e. **Queuing, latency, and jitter.** The larger the queue, the longer data has to wait. Tail drop occurs when the queue is full and admittance is prohibited. Acceptable for TCP, but not much else. When input queues fill, this can have a negative effect on CPU and memory as the device cannot process the incoming traffic fast enough.

f. **Speed and distance.** Data can only be forwarded up to the speed of light, which is approximately 100 miles per ms. Even with high bandwidth, low latency international links, the distance for energy to travel can be significant. Can be a large factor for certain applications if they are not tuned for specific network characteristics.

g. **Application characteristics.** Small window sizes, application keepalives, and amount of data sent over the network versus what is required is often the reason for poor application performance.

Best practices

a. **Service level management.** A deliverable that seeks to identify what service can be delivered within budgetary and staffing constraints. It can be done via an SLA between users and network organizations for a service. The service would include reports and recommendations to maintain acceptable service quality. Users must be prepared to fund the upgrades. Alternatively, the network organization can define their capacity/performance management upgrades and attempt to fund upgrades on a case-by-case basis.

b. **Network and application what-if analysis.** Used to determine the outcome of a planned change. Significantly reduces risk and helps increase network availability. Identify risk levels for individual changes and perform more detailed analyses on those that are highest risk. Cisco recommends 5 risk levels:

1. High potential impact to many users and/or business critical applications with expected downtime. Mitigation: Validation (lab) of new solution. Extensive testing and what-if analysis showing impact. New solutions require completion of an operations support document. Perform design review, create back-out plan/change process.

2. High potential impact to many users and/or business critical applications with possible downtime. Mitigation: Perform what-if analysis and review functionality. Perform design review, create back-out plan/change process.

3. Medium potential impact to less users and/or business service with possible downtime. Mitigation: Perform engineering analysis, and create implementation plan/change process.

4. Lower potential service impact, such as adding new template network modules, new WAN sites, etc. Mitigation: Create implementation plan/change process.

5. No user/service impact, such as adding individual users, changing passwords/banners/SNMP configurations, etc. Mitigation: None, all processes are optional.

General comments:

- Modeling of network topologies are recommend for all significant changes, such as routing, etc.
- Perform an application what-if analysis before deploying any business application. If you don't the application group will blame the network group for poor performance.

c. **Baselining and trending.** Allows network administrators to plan/complete network upgrades before a capacity problem arises. Can be resolved in several ways: build enough capacity so there is never an issue, divide the trend information into groups and focus on the critical network areas, or rely on reporting mechanisms to detect issues and respond to them first. All methods still require periodic review to ensure capacity demands are met over time. Graphical/chart tools are valuable for visualizing this kind of information.

d. **Exception management.** This is a reactive methodology; the network team receives notification of capacity/performance threshold violations and responds immediately. Tools like RMON and SNMP traps/informs (described in detail later) can be used for the initial reporting. Reporting must be near real-time so that the problem does not vanish before it is seen. Netflow can also be used to measure traffic, though it is passive and would require a tool to interpret the data to raise alarms.

e. **QoS management.** After baselining the network, traffic can be classified for preferential treatment. A basic design could include LLQ for voice, along with bandwidth guarantees for business critical applications. Large file transfers that are not time sensitive can be given lower priority.

Collecting and reporting capacity information: This is linked to three areas of capacity management. Each of these should have an information collection plan.

a. **What-if analyses.** Need tools to mimic the network environment and understand the effects of changes.

b. **Baselining and trending.** Need "snapshots" of current devices to shown current resource utilizations.

c. **Exception management.** Need this to alert network administrators about problems so resolution can occur.

5 steps for this process:

a. **Determine your needs.** Discover resources available and what is needed. Consider the people/expertise needed to use tools rather than just the tool/technology itself.

b. **Define a process.** Describe what happens when violations occur or what the baselining process looks like. Can be outsourced this functionality to a network services organization.

c. **Define capacity areas.** Certain areas of the network share common capacity planning strategies: LAN, WAN, field offices, critical sites, etc. LAN bandwidth is always cheaper than WAN bandwidth, so utilization should be lower. Different sites may require different SNMP MIB monitoring. Upgrades may be more difficult or time consuming in some places rather than others within the network.

d. **Define the capacity variables.** Depends on the devices/media in the network. General parameters such as CPU, memory, and link utilization are generally measured. Other things like queue depths and media-specific congestion can be valuable.

e. **Interpret the data.** Difference between peak and average is important, as peaks may be invisible based on the polling interval. Select a value that is a compromise; aggressive values cause excessive overhead while lethargic ones return less accurate information.

23. Describe maintenance and operational procedures

Maintenance procedures should be conducted in accordance with the hardware platform manual; there isn't much more to say other than that. Sustaining operations for a service provider is critical, which is why many platforms have an in-service software upgrade (ISSU) feature. This allows the administrator to upgrade/downgrade software versions of network devices while packet forwarding continues. This relies on Cisco non-stop forwarding (NSF) and stateful switchover (SSO) which implies the platform has redundant RSPs/RPs. The standby RSP/RP can be loaded with the next version, which can be manually started by the administrator while the old one is halted. Once the new version is explicitly accepted, it is written to both RSP/RPs going forward. If the new version introduces a new command or new syntax for something not recognized in the previous version, "Config Sync" is a process designed to recognize these new commands and maintain compatibility. ISSU has some limitations regarding the software versions it can support. ISSU may not be supported across major version releases if the underlying architecture changes sufficiently such that the underlying line-card FIBs must be changed, even if both versions supported ISSU. It is definitely not supported across code trains (T to S to M, for example) either. Cisco identifies code with one of three letters to denote its compatibility:

- a. **C: Compatible.** All base-level system infrastructure is supported as well as optional HW-aware subsystem components. There will be minimal service impact when performing ISSU to a version labeled compatible.
- b. **B: Base-level compatible.** ISSU will succeed but some sub-systems will not be able to maintain state during the transition. There may be some service interruption.
- c. **I: Incompatible.** The core architecture is different to the point that ISSU is not supported since SSO cannot work.

Incompatible updates are still desirable, so rather than use ISSU, one can use Fast software upgrade (FSU) which is a suboption of ISSU. By using the keyword "forced" in some platforms, you can force ISSU to perform the upgrade but it will be service impacting, so plan for that.

From a purely operational standpoint, service providers often use “runbooks” to describe their operations. This details procedures to begin, stop, supervise, and debug the system. Effective runbooks allow operations to effectively manage and troubleshoot a system, which may include automation. They are often exhaustive and include procedures for common scenarios with decision trees/flow charts to aid operators. It begins with an index of processes that are often broken down in outline form to associate processes to the operations/technologies they support. Runbook automation (RBA) is the ability to define, build, orchestrate, manage, and report on workflows that support processes that support the network.

24. Describe the network inventory management process

Network inventory management is the process of keeping records of all network assets owned by an organization, to include assets not currently in service (on the shelf, in maintenance, etc). It enables network administrators to have a record of all network equipment within the organization. Network inventory management is generally performed through asset tracking software that scans, compiles, and records data about each device/node over a network. This obviously cannot touch network devices not currently on line, but the software can be supplemented by manual inventorying done by humans.

Network inventory management would include basic information about each platform. This would include their make/model, location, serial number, network addressing (for management), software version, and license keys (along with expiration dates). Capturing this data is important for businesses. The most obvious benefit is to maintain accountability of all equipment and prevent theft. It also helps with estimating the size of a network and supports planning for growth. Knowing the network diameter also helps realize costs to maintain and grow the network.

25. Describe network change, implementation, and rollback

25.1 Processes and best practices

Configuration management best practices:

1. **Create standards.** Helps reduce network complexity, unplanned downtime, and exposure to network impacting events.

a. Software version. Deploy consistent SW versions on similar network devices. Increase interoperability, reduce SW defects and likelihood of complex issues.

b. IP addressing. Subnet allocation scheme ensures address blocks do not overlap and promotes contiguous address summarization. Identify certain hosts in the subnet for certain applications (i.e., always make the routers the first addresses, followed by switches, followed by hosts).

c. Naming conventions, DNS, and DHCP assignments. Simplifies the identification for devices and reduces the possibility of duplicate IP addresses. Adding DHCP ranges to DNS helps with identifying where hosts are also.

d. Baseline configurations and descriptors. Common configuration parameters, such as VTY settings, banners, NTP, SNMP, and AAA can be uniform on most network devices. Descriptors are used on interfaces to show the purpose and location of an interface.

e. Configuration upgrade procedures. Documenting these procedures ensures HW/SW upgrades go smoothly. The document should list all steps for the upgrade, reference vendor documentation, and provide post-upgrade validation steps for testing.

f. Solution templates. Define common solutions to common problems in modular form. Each template must be defined, tested, and documented to ensure uniformity when the templates are deployed. Templates may include HW modules, port/VLAN assignments, ACL/QoS configurations, out of band management (OOBM), and cable installations. These templates should not contain specifics like IP addresses or DNS/DHCP assignments.

2. Implement standards

3. **Maintain documentation.** Should be updated in near-real time. Cisco defines the following "critical success" factors:

a. Current device, link, and end-user inventory. Track devices, problem impact, and network change impacts. Should contain tables for similar devices, links, and user data.

b. Configuration version control system. Maintain set of current configurations and some number of previous versions for rollback, troubleshooting, or change audits. Maintain 3-5 previous versions of configuration per device.

c. TACACS configuration log. Any AAA server will do, but Cisco specifies TACACS. When used with NTP, it is easy to see who logged into a device, when it happened, and what was changed.

d. Network topology documentation. Should include both logical and physical topologies at different OSI layers for the entire network. Detail will range from routing diagrams to cabling/line-card/electrical level of detail. These are typically maintained in graphical form.

4. **Validate and audit standards.** Form a cross-functional team to measure configuration management quality. First objective is to implement change management (CM) to identify the issues.

a. Conduct integrity checks. Evaluate overall network configuration to include potential issues. This should catch inconsistent naming conventions, duplicate IP addresses, protocol mismatches, and other errors.

b. Device, protocol, and media audits. This would identify inconsistent HW/SW versions, incompatible modules, etc.

c. Standards and documentation reviews. Executed on a quarterly basis, all components of the standards (naming convention, IP subnetting plan, solution templates, etc) should be reviewed for inconsistencies and errors. New templates should be developed in accordance with new HW/SW upgrades as needed.

5. Review standards. The process restarts at "Create Standards".

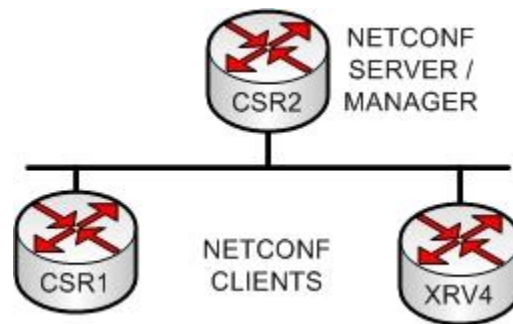
25.2 NETCONF and YANG

NETCONF is an IETF standard which is short for Network Configuration Protocol (not to be confused with PPP's Network Control Protocol (NCP) mechanism). It is a standard for configuring, deleting, and modifying configurations on network devices. YANG is used to model the network configurations and state into tree structures; it is the data model for NETCONF. Put another way, YANG is to NETCONF what MIBs are to SNMP, as YANG core models contain similar definitions for how data is structured. Like SNMP, NETCONF supports similar basic operations such as "GET", "GET-NEXT", "SET", etc, which makes it appear very similar to SNMP at a glance. The NETCONF / YANG model was intended to be vendor neutral to avoid dealing with specific CLI scripts or existing tools that were inflexible.

However, NETCONF has extra features, such as being able to test configurations before committing them, being able to configure multiple network devices concurrently, and bulk-get operations that are much faster than SNMP. Performing network-wide transactions (GET, SET, etc) is made easier since NETCONF handles the error recovery and sequencing. That is to say, the operator doesn't need to worry about the order of executing configuration commands within a given transaction. NETCONF uses SSH, which is secure, and also means that the operator doesn't need to resend requests that fail as is necessary using SNMP over UDP. NETCONF messages are encoded in XML, and individual YANG data models are included as "capabilities" in NETCONF hello packets. The device notifies the manager which capabilities it supports so the system knows the appropriate mechanisms to configure the device.

YANG data structures loosely resemble "classes" in object oriented programming or "structs" in non-OOP languages. The structure contains several "types", which include integers, Booleans, enumerations, pointers, and other information types to be processed. There are some commonly used YANG types defined in RFC 6021 which are specific to networking (or at least, well suited for it). Some of these include pre-made objects for IPv4 and IPv6 addresses and prefixes, flow-labels, port-number, MAC-address, URI, domain-name, and others. This is very convenient for managing/storing configuration data within YANG. NETCONF simply manipulates this data accordingly; provided the data from the network devices fits the YANG standard, NETCONF can modify it. If a vendor simply took their CLI configuration and wrapped it in a YANG structure somehow, this would not be effective.

NETCONF / YANG is supported in both XE and XR. The diagram for this test is shown below. CSR2 will be the NETCONF server for these examples, and it shares a LAN with CSR1 and XRv4 (NETCONF clients).



Below is an XE configuration example with basic verification. Since NETCONF requires SSH, and specifically SSHv2, we must configure a domain name as well as generate crypto keys (not shown). We can optionally add a standard ACL to identify the NETCONF managers for extra security. The NETCONF maximum sessions can be between 4 and 16 and there must be one VTY line per session, so configuring a number greater than the number of available VTY sessions does not make sense (default is 4). The lock-time identifies the maximum amount of time the NETCONF configuration is locked (cannot be changed while locked); the default is 10 seconds. The max-message option limits the size of the NETCONF messages, where by default there is no limit.

```

! CSR1
ip ssh logging events
ip ssh version 2
access-list 10 permit 10.1.14.2
netconf max-sessions 5
netconf lock-time 30
netconf max-message 65535
netconf ssh acl 10
  
```

Although the documentation suggests that entering the exec command “netconf” enters the NETCONF conversation, it doesn’t appear to work on XE version 3.13S with the CSR1000v.

```

R2#ssh -l test 10.1.14.1 netconf
Password: test
Line has invalid autocommand "netconf"
[Connection to 10.1.14.1 closed by foreign host]
  
```

Even if we test it on CSR1 natively, the command doesn’t exist. However, if it did work, we can check the sessions using the command below.

```

R1#netconf
% Bad IP address or host name address

R1#net?
% Unrecognized command

R1#show netconf session
  
```


Netconf Sessions: 0 open, maximum is 5

The XR configuration is similar except we specify the SSH port 830 (the IANA recommended port) and enable NETCONF for SSH. I add several other minor options for demonstration only to adjust the NETCONF session limits and data transfer policies. XR actually does work with NETCONF in this setup. Again, the assumption is that basic SSHv2 (domain name, keys, username, etc) has been configured.

```
! XRv4
ssh server netconf port 830

netconf agent tty
  session timeout 15
  throttle memory 400
  throttle process-rate 1500

netconf-yang agent
  ssh
  session limit 5
  session idle-timeout 10
  session absolute-timeout 15
  rate-limit 65535
```

From our NETCONF server, we will log into XRv4 using SSHv2 over TCP port 830. The “netconf” option is an auto-command that gets run immediately upon log in, which drops us into the NETCONF conversation. The NETCONF client (XRv4) sends a “hello” message to CSR2 which contains its capabilities. We could respond with a hand-crafted “hello” message, but that is way beyond the scope of this test. We can see that if CSR2 were really a NETCONF manager, this would have worked.

```
R2#ssh -l cisco -p 830 10.1.14.14 netconf
```

```
IMPORTANT:  READ CAREFULLY
[snip]
Password: cisco
```

```
<?xml version="1.0" encoding="UTF-8"?>
<hello
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><capabilities><capability>urn:ietf:params:netconf:base:1.0</capability><capability>urn:ietf:params:netconf:capability:candidate:1.0</capability><capability>urn:ietf:params:netconf:capability:notification:1.0</capability></capabilities><session-id>285212672</session-id></hello>
]]>]]>
```

Before terminating the NETCONF session, we can perform some basic verifications on XRv4. We can see the session from CSR2 in summary and detailed form. The session is idle since CSR2 didn’t send any NETCONF messages to XRv4, but we see the session is up.

```
RP/0/0/CPU0:XRv4#show netconf sessions
Session      Client      Agent  User      Date      State
11000000    10.1.14.2    tty    cisco     Sun MON DAY 15:10:11 2015 Idle
```

```
RP/0/0/CPU0:XRv4#show netconf sessions detail
1) Session      : 11000000
   Client        : 10.1.14.2:0 (VRF -)
   Agent Type    : tty
   User          : cisco
   State         : Idle
   Config Session : -
   Admin Config Session : -
   Alarm Notification : Not Registered
   Start Date     : Sun MON DAY 15:10:11 2015
   Elapsed Time   : 00:00:13
   Last State Changed : 00:00:13
```

The command below shows detailed NETCONF statistics based on configuration changes. The table is enormous so I have snipped rows and columns from it.

```
RP/0/0/CPU0:XRv4#show netconf-yang statistics
Summary statistics
# requests|          total time|    min time per request|
|
other      0|          0h 0m 0s 0ms|          0h 0m 0s 0ms|
|
close-session 0|          0h 0m 0s 0ms|          0h 0m 0s 0ms|
|
kill-session 0|          0h 0m 0s 0ms|          0h 0m 0s 0ms|
|
get-schema 0|          0h 0m 0s 0ms|          0h 0m 0s 0ms|
[snip]
```

Additional Reading – Reference configurations “netconf-yang”

26. Describe the incident management process based on the ITILv3 framework

ITIL defines an incident as any event that is not part of the standard operation of the service. The event causes, or may cause, an interruption of the service or a reduction in its quality. Failures that affect an item, even if they don't affect a service, also qualify as an incident. This could be an LACP member port failing with no significant network impacts. An incident is different than a problem in that the latter consists of recurring, related incidents. Problem Management is a different process than Incident Management. The objective of Incident Management (IM) is to restore a service to normal operations with minimal impact on the business or user, and at minimal cost.

IM inputs mostly come from users, but machines (management or detection systems) can provide information as well. The output of IM are requests for changes (RFCs) which can mitigate future incidents.

Incidents can be prioritized generally by considering (summing) the impact and the urgency. Impact refers to the business criticality of the incident and the extent that it degrades the quality of the SLA (large number of users out of service, etc). Urgency reflects the required speed of resolving an incident. The chart below summarizes the prioritization scheme. Urgency is on the y-axis with impact on the x-axis.

Priority 2 Limited damage Recover within 2 hours	Priority 1 Significant damage Recover within 1 hour
Priority 4 Limited damage Recover within 8 hours	Priority 3 Significant damage Recover within 4 hours

Incidents can be escalated functionally or hierarchically. Functional escalation moves incidents from level 1 (front line service desk support) up through levels 2 and 3. Hierarchical escalation advances the incidents upward to the Incident Manager and ultimately to the Service Manager.

IM consists of several sub processes:

- Incident detection and recording
- Classification and initial support
- Investigation and analysis
- Resolution and recovery
- Incident closure
- Incident ownership, monitoring, tracking and communication
- Establish incident framework management
- Evaluation of incident framework management

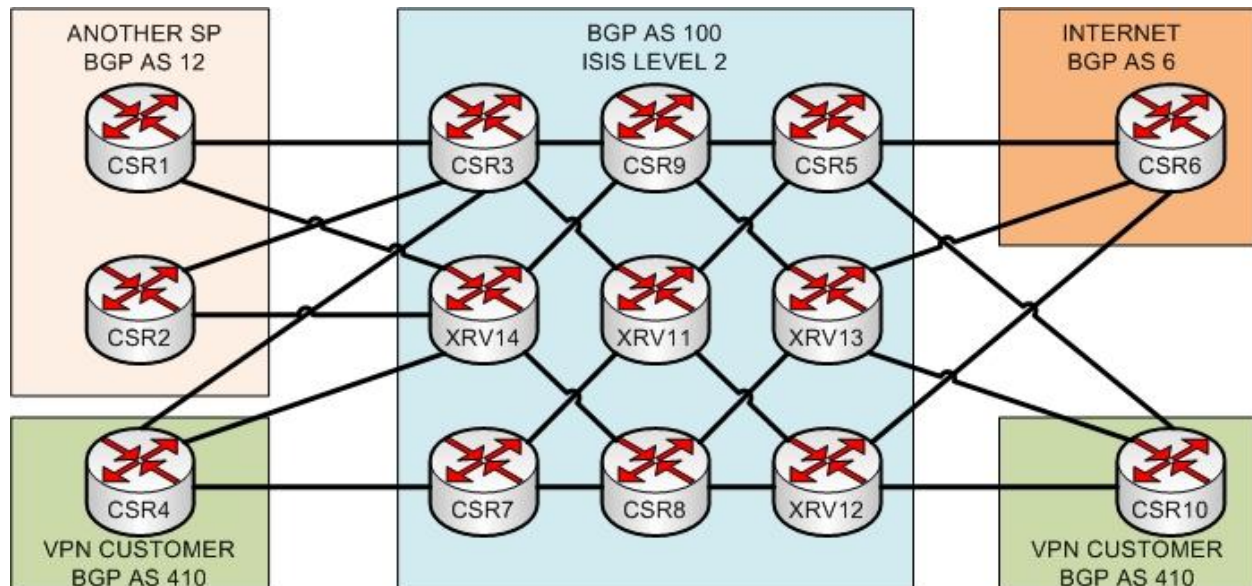
27. Describe, implement, and troubleshoot advanced BGP features

27.1 Additional Paths (add-path) and Prefix Independent Convergence (PIC)

It is a well-known behavior that using BGP route-reflection within an iBGP domain can hide topological redundancies between clients in many cases. BGP only advertises it's best path, which is true for RRs also. Often times this is desirable since a route-reflector's best path might be the best path for everyone else, also. Sometimes this is a limitation since clients with multiple paths could potentially load-share, or use additional baths for fast-reroute. BGP additional-paths, or add-path, is meant to overcome the limitations with hiding the BGP topology when using RRs.

The topology below shows a square-shaped provider network which uses IS-IS Level 2 for IP routing and LDP for MPLS transport label binding. RSVP-TE is also enabled for ad-hoc testing as needed. Initially, all

interfaces use the default IS-IS metric of 10, but we will make modifications to this as we test different features. The SP network is BGP AS 100. The Internet injects several routes into the global BGP table of the SP (AS 6). The SP transports these to another SP (AS 12) and MPLS transport isn't really the goal because we are testing BGP add-path. The SP has one L3VPN customer (AS 410) so we can test BGP add-path with VPNv4 in addition to IPv4.



Let's examine the state of the network before enabling any advanced features. CSR8, CSR9, and XRV11 are route-reflectors for both VPNv4 and IPv4 (the middle column). The routers in the left and right columns are PEs which peer with all three of the RRs; some PEs only run IPv4 while some only run VPNv4 depending on who their customers are. CSR5, XRV12, and XRV13 all receive several Internet routes from CSR6 and advertise them to the RRs. All of the RR's will have the exact same routes, but may have different best-paths due to the IGP metric. For example, looking at an Internet route, CSR9 prefers the path via CSR5 due to IGP metric and lowest RID while CSR8 prefers the path through XRV12.

```
CSR9#show bgp ipv4 unicast 28.10.45.0/25
BGP routing table entry for 28.10.45.0/25, version 50
Paths: (3 available, best #1, table default)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  6 78 454, (Received from a RR-client)
    5.5.5.5 (metric 10) from 5.5.5.5 (5.5.5.5)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  6 78 454, (Received from a RR-client)
    12.12.12.12 (metric 30) from 12.12.12.12 (12.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal
      rx pathid: 0, tx pathid: 0
```

Refresh Epoch 1

```
6 78 454, (Received from a RR-client)
13.13.13.13 (metric 10) from 13.13.13.13 (13.13.13.13)
Origin incomplete, metric 0, localpref 100, valid, internal
rx pathid: 0, tx pathid: 0
```

CSR8#show bgp ipv4 unicast 28.10.45.0/25

BGP routing table entry for 28.10.45.0/25, version 80

Paths: (3 available, best #2, table default)

Advertised to update-groups:

1

Refresh Epoch 1

```
6 78 454, (Received from a RR-client)
5.5.5.5 (metric 30) from 5.5.5.5 (5.5.5.5)
Origin incomplete, metric 0, localpref 100, valid, internal
rx pathid: 0, tx pathid: 0
```

Refresh Epoch 1

```
6 78 454, (Received from a RR-client)
12.12.12.12 (metric 10) from 12.12.12.12 (12.12.12.12)
Origin incomplete, metric 0, localpref 100, valid, internal, best
rx pathid: 0, tx pathid: 0x0
```

Refresh Epoch 1

```
6 78 454, (Received from a RR-client)
13.13.13.13 (metric 10) from 13.13.13.13 (13.13.13.13)
Origin incomplete, metric 0, localpref 100, valid, internal
rx pathid: 0, tx pathid: 0
```

CSR3, for example, is going to learn the best-path from each of the three RR's. XRv11 has selected CSR5 as the best path (lowest RID), which means CSR3 has no idea there is a third path through XRv13. CSR3 now has two copies of one path and one of another, but is missing the third.

CSR3#show bgp ipv4 unicast 28.10.45.0/25

BGP routing table entry for 28.10.45.0/25, version 97

Paths: (3 available, best #3, table default)

Advertised to update-groups:

4

Refresh Epoch 1

```
6 78 454
5.5.5.5 (metric 20) from 11.11.11.11 (11.11.11.11)
Origin incomplete, metric 0, localpref 100, valid, internal
Originator: 5.5.5.5, Cluster list: 11.11.11.11
rx pathid: 0, tx pathid: 0
```

Refresh Epoch 2

```
6 78 454
12.12.12.12 (metric 20) from 8.8.8.8 (8.8.8.8)
Origin incomplete, metric 0, localpref 100, valid, internal
Originator: 12.12.12.12, Cluster list: 8.8.8.8
rx pathid: 0, tx pathid: 0
```

Refresh Epoch 2

6 78 454

5.5.5.5 (metric 20) from 9.9.9.9 (9.9.9.9)

Origin incomplete, metric 0, localpref 100, valid, internal, best

Originator: 5.5.5.5, Cluster list: 9.9.9.9

rx pathid: 0, tx pathid: 0x0

BGP add-path has many solutions for problems like this (and others). One quick solution is to direct CSR9 to only advertise its second best path, which would be via XRv13. This way, each of the three RR's coincidentally advertise a diverse path to CSR3, XRv14, and CSR7. This is sometimes called the "shadow RR" method. Of note, the Shadow RR itself has a few steps of configuration. Specifically, it has to select the backup path as the repair path, install it in the FIB, and then advertise it to its neighbor.

! CSR9

router bgp 100

template peer-policy IPV4

advertise diverse-path backup

address-family ipv4

bgp additional-paths select backup

bgp additional-paths install

Verifications are shown below. We check the BGP RIB, FIB, and advertised routes. The "bia" codes on the BGP advertised route indicate a backup-path, iBGP learned, and additional-path respectively. Notice that the ">" symbol is not shown on this route since it is a backup path, not a best path, which seems to violate the fundamental rule of BGP route advertisement at first glance.

CSR9#show bgp ipv4 unicast 28.10.45.0/25

BGP routing table entry for 28.10.45.0/25, version 122

Paths: (3 available, best #1, table default)

Additional-path-install

Advertised to update-groups:

2

Refresh Epoch 4

6 78 454, (Received from a RR-client)

5.5.5.5 (metric 10) from 5.5.5.5 (5.5.5.5)

Origin incomplete, metric 0, localpref 100, valid, internal, best

rx pathid: 0, tx pathid: 0x0

Refresh Epoch 1

6 78 454, (Received from a RR-client)

12.12.12.12 (metric 30) from 12.12.12.12 (12.12.12.12)

Origin incomplete, metric 0, localpref 100, valid, internal

rx pathid: 0, tx pathid: 0

Refresh Epoch 1

6 78 454, (Received from a RR-client)

13.13.13.13 (metric 10) from 13.13.13.13 (13.13.13.13)

Origin incomplete, metric 0, localpref 100, valid, internal,

backup/repair

```
rx pathid: 0, tx pathid: 0
```

```
CSR9#show ip cef 28.10.45.0 detail
28.10.45.0/25, epoch 2, flags [rib only nolabel, rib defined all labels]
  recursive via 5.5.5.5
    nexthop 153.5.9.5 GigabitEthernet2.559
  recursive via 13.13.13.13, repair
    nexthop 153.9.13.13 GigabitEthernet2.593

CSR9#show bgp ipv4 unicast neighbors 3.3.3.3 advertised-routes | include
28.10.45.0/25
*bia28.10.45.0/25      13.13.13.13          0      100      0 6 78 454 ?
```

When we check CSR3, we can see it has three different next-hops for this prefix, which is ideal. Notice that the route from CSR9 isn't really CSR3's best path, since we specifically directed it to advertise its second best path. This still hasn't helped us too much, since CSR3 still only picked one best path.

```
CSR3#show bgp ipv4 unicast 28.10.45.0/25
BGP routing table entry for 28.10.45.0/25, version 112
Paths: (3 available, best #1, table default)
  Advertised to update-groups:
    4
  Refresh Epoch 1
    6 78 454
    5.5.5.5 (metric 20) from 11.11.11.11 (11.11.11.11)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Originator: 5.5.5.5, Cluster list: 11.11.11.11
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 2
    6 78 454
    12.12.12.12 (metric 20) from 8.8.8.8 (8.8.8.8)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Originator: 12.12.12.12, Cluster list: 8.8.8.8
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 5
    6 78 454
    13.13.13.13 (metric 20) from 9.9.9.9 (9.9.9.9)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Originator: 13.13.13.13, Cluster list: 9.9.9.9
      rx pathid: 0, tx pathid: 0
```

Of greater benefit to CSR3 would be the ability to use these paths for ECMP, since they all have equal IGP metrics. Let's enable iBGP multipath on CSR3 for the IPv4 AF. CSR3 still identifies a best-path because BGP always selects a single best-path, but marks the ECMP paths as "multipath". The reason for always needing exactly one best-path is because this what BGP would normally advertise on (with add-path we can change it, but this describes classic BGP behavior). The actual load sharing is interesting because the RIB installs three entries as expected, but based on the IGP routes to the BGP next-hops, we

see the FIB uses paths through XRv11 and CSR9 to service the three different routes. There is more diversity following these paths in the “core” of the network.

```
! CSR3
router bgp 100
  address-family ipv4
    maximum-paths ibgp 3

CSR3#show bgp ipv4 unicast 28.10.45.0/25
BGP routing table entry for 28.10.45.0/25, version 127
Paths: (3 available, best #1, table default)
Multipath: iBGP
  Advertised to update-groups:
    4
  Refresh Epoch 1
  6 78 454
    5.5.5.5 (metric 20) from 11.11.11.11 (11.11.11.11)
      Origin incomplete, metric 0, localpref 100, valid, internal, multipath,
best
      Originator: 5.5.5.5, Cluster list: 11.11.11.11
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 2
  6 78 454
    12.12.12.12 (metric 20) from 8.8.8.8 (8.8.8.8)
      Origin incomplete, metric 0, localpref 100, valid, internal, multipath
      Originator: 12.12.12.12, Cluster list: 8.8.8.8
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 5
  6 78 454
    13.13.13.13 (metric 20) from 9.9.9.9 (9.9.9.9)
      Origin incomplete, metric 0, localpref 100, valid, internal,
multipath(oldest)
      Originator: 13.13.13.13, Cluster list: 9.9.9.9
      rx pathid: 0, tx pathid: 0

CSR3#show ip route 28.10.45.0
Routing entry for 28.10.45.0/25
  Known via "bgp 100", distance 200, metric 0
  Tag 6, type internal
  Last update from 13.13.13.13 00:03:34 ago
  Routing Descriptor Blocks:
    13.13.13.13, from 9.9.9.9, 00:03:34 ago
      Route metric is 0, traffic share count is 1
      AS Hops 3
      Route tag 6
      MPLS label: none
    12.12.12.12, from 8.8.8.8, 00:03:34 ago
      Route metric is 0, traffic share count is 1
      AS Hops 3
```



```

Route tag 6
MPLS label: none
* 5.5.5.5, from 11.11.11.11, 00:03:34 ago
Route metric is 0, traffic share count is 1
AS Hops 3
Route tag 6
MPLS label: none

```

```

CSR3#show ip cef 28.10.45.0
28.10.45.0/25
  nexthop 153.3.9.9 GigabitEthernet2.539 label 9003
  nexthop 153.3.11.11 GigabitEthernet2.531 label 91013
  nexthop 153.3.11.11 GigabitEthernet2.531 label 91010
  nexthop 153.3.9.9 GigabitEthernet2.539 label 9000

```

Looking at the labels, we come up with the following conclusions. This is significant because the FIB ultimately determines the level of HA you achieve in a network.

1. Traffic to CSR5 can be load-shared between CSR9 and XRv11.
2. Traffic to XRv12 always flows through XRv11.
3. Traffic to XRv13 always flows through CSR9.

```

CSR3#show mpls forwarding-table 5.5.5.5 32
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
3010       9003      5.5.5.5/32      0            Gi2.539    153.3.9.9
           91013     5.5.5.5/32      0            Gi2.531    153.3.11.11

```

```

CSR3#show mpls forwarding-table 12.12.12.12 32
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
3000       91010     12.12.12.12/32  0            Gi2.531    153.3.11.11

```

```

CSR3#show mpls forwarding-table 13.13.13.13 32
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
3005       9000      13.13.13.13/32  1400         Gi2.539    153.3.9.9

```

Next, we will remove the shadow RR configuration from CSR9 and multipath configuration from CSR3 to return the network to its baseline status. A common feature to use on RR's that are spread throughout a network is disabling the IGP-metric evaluation in the BGP bestpath process. For example, CSR9 never considered XRv12 as a viable candidate because the IGP metric was 30, where the IGP metric to CSR5 and XRv13 was 10. If we disable IGP metric consideration on all RRs, then the next tie-breaker is lowest RID, and every RR will select the exact same bestpath. If we configure this feature by itself only CSR8 and CSR9 we actually cause more damage because now CSR3 only knows about the path via CSR5. We have to disable XRv11 BGP for this test since it does not appear to support ignoring the IGP metric. The

easiest way to do this is shutting down the IBGP session-group. At this point, CSR3 prefers the path via CSR5 (through CSR9) because the neighbor ID is lower. Regardless of which path CSR3 selects, it can only use CSR5 for forwarding, ignoring the other two candidate PEs.

```
! CSR8 and CSR9
router bgp 100
  address-family ipv4
    bgp bestpath igp-metric ignore

CSR3#show bgp ipv4 unicast 28.10.45.0/25
BGP routing table entry for 28.10.45.0/25, version 172
Paths: (2 available, best #1, table default)
  Advertised to update-groups:
    4
  Refresh Epoch 2
  6 78 454
    5.5.5.5 (metric 20) from 8.8.8.8 (8.8.8.8)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Originator: 5.5.5.5, Cluster list: 8.8.8.8
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 6
  6 78 454
    5.5.5.5 (metric 20) from 9.9.9.9 (9.9.9.9)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Originator: 5.5.5.5, Cluster list: 9.9.9.9
      rx pathid: 0, tx pathid: 0
```

Earlier we used the “Shadow RR” in a highly coincidental and cherry-picked situation to solve our problem. The current network is more representative of a real setup where all of the RRs pick the exact same path. In this case, we can re-enable the Shadow RR feature on CSR9 so that it picks the next best path, ignoring the IGP metric, which is XRv12. Now, CSR3 has the backup paths. This design is more resilient because changes in IGP will not affect the BGP topology; if the RR's IGP metrics to a PE change, it doesn't matter. Since the two RRs always pick the same best paths, the shadow (CSR9) knows that it's safe to advertise the second best. Without ignoring IGP metric, RRs might pick opposing paths, and without shadow RR enabled, this is fine (we just tested that). But with shadow enabled, this might result in the remote PEs receiving the same route twice due to lack of synchronization between the RRs. Although I don't show it here, iBGP multipath would be a good option on the remote PE (CSR3) or using backup paths (discussed later) to achieve fast convergence.

```
CSR9#show bgp ipv4 unicast 28.10.45.0/25
BGP routing table entry for 28.10.45.0/25, version 190
BGP Bestpath: igpmetric-ignore
Paths: (3 available, best #1, table default)
  Additional-path-install
  Advertised to update-groups:
    4
```

```

Refresh Epoch 4
6 78 454, (Received from a RR-client)
  5.5.5.5 (metric 10) from 5.5.5.5 (5.5.5.5)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    rx pathid: 0, tx pathid: 0x0
Refresh Epoch 1
6 78 454, (Received from a RR-client)
  12.12.12.12 (metric 30) from 12.12.12.12 (12.12.12.12)
    Origin incomplete, metric 0, localpref 100, valid, internal,
    backup/repair
    rx pathid: 0, tx pathid: 0
Refresh Epoch 1
6 78 454, (Received from a RR-client)
  13.13.13.13 (metric 10) from 13.13.13.13 (13.13.13.13)
    Origin incomplete, metric 0, localpref 100, valid, internal
    rx pathid: 0, tx pathid: 0

CSR3#show bgp ipv4 unicast 28.10.45.0/25
BGP routing table entry for 28.10.45.0/25, version 172
Paths: (2 available, best #1, table default)
  Advertised to update-groups:
    4
Refresh Epoch 2
6 78 454
  5.5.5.5 (metric 20) from 8.8.8.8 (8.8.8.8)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    Originator: 5.5.5.5, Cluster list: 8.8.8.8
    rx pathid: 0, tx pathid: 0x0
Refresh Epoch 7
6 78 454
  12.12.12.12 (metric 20) from 9.9.9.9 (9.9.9.9)
    Origin incomplete, metric 0, localpref 100, valid, internal
    Originator: 12.12.12.12, Cluster list: 9.9.9.9
    rx pathid: 0, tx pathid: 0

```

We will remove the shadow RR configuration from CSR9 again to test other features. We will leave the IGP-metric ignore in place and leave XRv11 BGP shutdown for now. The RR's are configured to send additional paths while the PEs are configured to receive them for IPv4. The general snippet is below for reference. Before checking the BGP tables, we can verify this negotiated capability was successful by checking the BGP neighbor details. Beware the logic on the PEs; both CSR3 and XRv14 say the "send" capability was received and the "receive" capability was sent. This is actually correct. The RR's logic is easier to digest but it's effectively the opposite of the clients.

```

! XE and XR PE
router bgp 100
  address-family ipv4
    bgp additional-paths receive

```

```

! XE and XR RR
router bgp 100
  address-family ipv4
    bgp additional-paths send

```

```

CSR3#show bgp ipv4 unicast neighbors 9.9.9.9 | include Additional
  Additional Paths send capability: received
  Additional Paths receive capability: advertised

```

```

RP/0/0/CPU0:XRv14#show bgp ipv4 unicast neighbors 9.9.9.9 | include
Additional
  Additional-paths Send: received
  Additional-paths Receive: advertised

```

```

CSR9#show bgp ipv4 unicast neighbors 3.3.3.3 | include Additional
  Additional Paths send capability: advertised
  Additional Paths receive capability: received

```

First we will test selecting the best 2 paths from the BGP RIB and advertising them to all iBGP peers on CSR9. CSR9 shows best2 path as XRv12 due to lower RID than XRv13 (remember, we ignore the IGP metric). We clearly see them being advertised to CSR3 and XRv14 as examples. Notice that with this more powerful feature, the routers are actually advertising multiple paths. The shadow RR advertised a single back up, not multiple paths.

```

! CSR9
router bgp 100
  template peer-policy IPV4
    advertise additional-paths best 2
  address-family ipv4
    bgp additional-paths select best 2

```

```

CSR9#show bgp ipv4 unicast 28.10.45.0/25
BGP routing table entry for 28.10.45.0/25, version 329
BGP Bestpath: igpmetric-ignore
Paths: (3 available, best #1, table default)
  Additional-path-install
  Path advertised to update-groups:
    6          7
  Refresh Epoch 1
  6 78 454, (Received from a RR-client)
    5.5.5.5 (metric 10) from 5.5.5.5 (5.5.5.5)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      rx pathid: 0, tx pathid: 0x0
  Path advertised to update-groups:
    7
  Refresh Epoch 1
  6 78 454, (Received from a RR-client)
    12.12.12.12 (metric 30) from 12.12.12.12 (12.12.12.12)

```

```
Origin incomplete, metric 0, localpref 100, valid, internal,
backup/repair, best2
```

```
rx pathid: 0, tx pathid: 0x1
```

```
Path not advertised to any peer
```

```
Refresh Epoch 1
```

```
6 78 454, (Received from a RR-client)
```

```
13.13.13.13 (metric 10) from 13.13.13.13 (13.13.13.13)
```

```
Origin incomplete, metric 0, localpref 100, valid, internal
```

```
rx pathid: 0, tx pathid: 0
```

```
CSR9#show bgp ipv4 unicast neighbors 3.3.3.3 advertised-routes | include
28.10.45.0/25
```

```
*>i 28.10.45.0/25      5.5.5.5              0      100      0 6 78 454 ?
```

```
*bia28.10.45.0/25     12.12.12.12           0      100      0 6 78 454 ?
```

```
CSR9#show bgp ipv4 unicast neighbors 14.14.14.14 advertised-routes | include
28.10.45.0/25
```

```
*>i 28.10.45.0/25      5.5.5.5              0      100      0 6 78 454 ?
```

```
*bia28.10.45.0/25     12.12.12.12           0      100      0 6 78 454 ?
```

Now, CSR3 and XRv14 will have three paths to choose from (CSR8 bestpath, CSR9 bestpath, and CSR9 best2 path). In order to make this effective, we should either use iBGP multipath or backup paths on the PEs. We will configure backup paths on CSR3 and XRv14 for variety. Notice that CSR3 is smart enough to pick a backup path that has a different next-hop than the best-path, although technically the "bestpath" from CSR9 (nexthop 5.5.5.5) is better than the backup due to lower originator-ID. We confirm this in the FIB. The bestpath using 5.5.5.5 as a next-hop has ECMP paths through CSR9 and XRv11 according to IGP, while the backup path has a single path via XRv11.

```
CSR3#show bgp ipv4 unicast 28.10.45.0/25
```

```
BGP routing table entry for 28.10.45.0/25, version 324
```

```
Paths: (3 available, best #2, table default)
```

```
Additional-path-install
```

```
Advertised to update-groups:
```

```
4
```

```
Refresh Epoch 2
```

```
6 78 454
```

```
12.12.12.12 (metric 20) from 9.9.9.9 (9.9.9.9)
```

```
Origin incomplete, metric 0, localpref 100, valid, internal,
```

```
backup/repair
```

```
Originator: 12.12.12.12, Cluster list: 9.9.9.9
```

```
rx pathid: 0x1, tx pathid: 0
```

```
Refresh Epoch 2
```

```
6 78 454
```

```
5.5.5.5 (metric 20) from 8.8.8.8 (8.8.8.8)
```

```
Origin incomplete, metric 0, localpref 100, valid, internal, best
```

```
Originator: 5.5.5.5, Cluster list: 8.8.8.8
```

```
rx pathid: 0x0, tx pathid: 0x0
```

```
Refresh Epoch 2
```

```

6 78 454
  5.5.5.5 (metric 20) from 9.9.9.9 (9.9.9.9)
    Origin incomplete, metric 0, localpref 100, valid, internal
    Originator: 5.5.5.5, Cluster list: 9.9.9.9
    rx pathid: 0x0, tx pathid: 0

CSR3#show ip cef 28.10.45.0 det
28.10.45.0/25, epoch 2, flags [rib only nolabel, rib defined all labels]
  recursive via 5.5.5.5
    nexthop 153.3.9.9 GigabitEthernet2.539 label 9003
    nexthop 153.3.11.11 GigabitEthernet2.531 label 91013
  recursive via 12.12.12.12, repair
    nexthop 153.3.11.11 GigabitEthernet2.531 label 91010

```

XRv14 has the exact same choices as CSR3 did. XR requires extra work, because we need to apply an RPL as an additional-path selection filter to determine not only what we want to backup, but how. Below is a sample RPL that only backs up the route in question. The configuration is shown below. I didn't have to match the specific prefix but did so for illustrative purposes. XR is also smart enough to pick a diverse BGP next-hop for its backup path. We perform a verification of both the BGP RIB and FIB.

```

! XRv14
route-policy RPL_BACKUP_45_0
  if destination in (28.10.45.0/25) then
    set path-selection backup 1 install
  endif
end-policy

router bgp 100
  address-family ipv4 unicast
    additional-paths selection route-policy RPL_BACKUP_45_0

RP/0/0/CPU0:XRv14#show bgp ipv4 unicast 28.10.45.0/25 | begin Paths
Paths: (3 available, best #1)
  Advertised to update-groups (with more than one peer):
    0.3
  Path #1: Received by speaker 0
  Advertised to update-groups (with more than one peer):
    0.3
6 78 454
  5.5.5.5 (metric 20) from 8.8.8.8 (5.5.5.5)
    Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best
  Received Path ID 0, Local Path ID 1, version 255
  Originator: 5.5.5.5, Cluster list: 8.8.8.8
  Path #2: Received by speaker 0
  Not advertised to any peer
6 78 454
  5.5.5.5 (metric 20) from 9.9.9.9 (5.5.5.5)

```

```

        Origin incomplete, metric 0, localpref 100, valid, internal
        Received Path ID 0, Local Path ID 0, version 0
        Originator: 5.5.5.5, Cluster list: 9.9.9.9
    Path #3: Received by speaker 0
    Not advertised to any peer
    6 78 454
    12.12.12.12 (metric 20) from 9.9.9.9 (12.12.12.12)
        Origin incomplete, metric 0, localpref 100, valid, internal, backup,
add-path
        Received Path ID 1, Local Path ID 2, version 266
        Originator: 12.12.12.12, Cluster list: 9.9.9.9

RP/0/0/CPU0:XRv14#show cef ipv4 28.10.45.0/25
28.10.45.0/25, version 2900, internal 0x5000001 0x0 (ptr 0xa13d2b74) [1], 0x0
(0x0), 0x0 (0x0)
local adjacency 153.9.14.9
Prefix Len 25, traffic index 0, precedence n/a, priority 4
via 5.5.5.5, 3 dependencies, recursive [flags 0x6000]
    path-idx 0 NHID 0x0 [0xa13d2574 0x0]
    next hop 5.5.5.5 via 5.5.5.5/32
via 12.12.12.12, 2 dependencies, recursive, backup [flags 0x6100]
    path-idx 1 NHID 0x0 [0xa13d2a74 0x0]
    next hop 12.12.12.12 via 12.12.12.12/32

```

We can also select the best 3 paths, which would effectively be the same as selecting all paths (both are options) in this case since we only have 3 paths. On CSR9, we will globally direct BGP to select the best 3 paths, then advertise the best 2 to all peers (default) but send the best 3 to XRv14. This is basic inheritance using the templates and overriding the settings by making changes directly on a BGP peer. Since best2 is already in the policy template for IPv4, we will override it specifically for XRv14. XRv14 now has 4 paths; one from CSR8 (it's best path) and three from CSR9. It still picks CSR8 as the best path due to lowest neighbor ID, then picks XRv12 as the backup given the lower originator ID. I highlighted the best two paths along with all three paths from CSR9.

```

! CSR9
router bgp 100
    template peer-policy IPV4
        advertise additional-paths best 2
    address-family ipv4
        bgp additional-paths select best 3
        neighbor 14.14.14.14 advertise additional-paths best 3

RP/0/0/CPU0:XRv14#show bgp ipv4 unicast 28.10.45.0/25 | begin Paths
Paths: (4 available, best #1)
    Advertised to update-groups (with more than one peer):
        0.3
    Path #1: Received by speaker 0
    Advertised to update-groups (with more than one peer):
        0.3

```

```

6 78 454
5.5.5.5 (metric 20) from 8.8.8.8 (5.5.5.5)
  Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best
  Received Path ID 0, Local Path ID 1, version 255
  Originator: 5.5.5.5, Cluster list: 8.8.8.8
Path #2: Received by speaker 0
Not advertised to any peer
6 78 454
5.5.5.5 (metric 20) from 9.9.9.9 (5.5.5.5)
  Origin incomplete, metric 0, localpref 100, valid, internal
  Received Path ID 0, Local Path ID 0, version 0
  Originator: 5.5.5.5, Cluster list: 9.9.9.9
Path #3: Received by speaker 0
Not advertised to any peer
6 78 454
12.12.12.12 (metric 20) from 9.9.9.9 (12.12.12.12)
  Origin incomplete, metric 0, localpref 100, valid, internal, backup,
add-path
  Received Path ID 1, Local Path ID 2, version 266
  Originator: 12.12.12.12, Cluster list: 9.9.9.9
Path #4: Received by speaker 0
Not advertised to any peer
6 78 454
13.13.13.13 (metric 20) from 9.9.9.9 (13.13.13.13)
  Origin incomplete, metric 0, localpref 100, valid, internal
  Received Path ID 2, Local Path ID 0, version 0
  Originator: 13.13.13.13, Cluster list: 9.9.9.9

```

This is a good opportunity to enable multipath on XRv14 to take advantage of all paths. Notice that even when enabling iBGP multipath for 4 paths, only 3 are used, since the BGP next-hop for two of them is the same. As always, we verify the BGP table, RIB, and FIB. Because I still have add-path configured, one of the paths is still considered as a backup also. Multipath and add-path are not mutually exclusive in many cases. However, notice that the FIB does not flag anything as “backup” or “repair”, because all of the paths are already in the FIB as primary, ECMP options.

```

! XRv14
router bgp 100
  address-family ipv4 unicast
    maximum-paths ibgp 4

RP/0/0/CPU0:XRv14#show bgp ipv4 unicast 28.10.45.0/25 | begin Paths
Paths: (4 available, best #1)
  Advertised to update-groups (with more than one peer):
    0.3
  Path #1: Received by speaker 0
  Advertised to update-groups (with more than one peer):
    0.3

```



```

6 78 454
  5.5.5.5 (metric 20) from 8.8.8.8 (5.5.5.5)
    Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, multipath
    Received Path ID 0, Local Path ID 1, version 272
    Originator: 5.5.5.5, Cluster list: 8.8.8.8
Path #2: Received by speaker 0
Not advertised to any peer
6 78 454
  5.5.5.5 (metric 20) from 9.9.9.9 (5.5.5.5)
    Origin incomplete, metric 0, localpref 100, valid, internal
    Received Path ID 0, Local Path ID 0, version 0
    Originator: 5.5.5.5, Cluster list: 9.9.9.9
Path #3: Received by speaker 0
Not advertised to any peer
6 78 454
  12.12.12.12 (metric 20) from 9.9.9.9 (12.12.12.12)
    Origin incomplete, metric 0, localpref 100, valid, internal, multipath,
backup, add-path
    Received Path ID 1, Local Path ID 3, version 268
    Originator: 12.12.12.12, Cluster list: 9.9.9.9
Path #4: Received by speaker 0
Not advertised to any peer
6 78 454
  13.13.13.13 (metric 20) from 9.9.9.9 (13.13.13.13)
    Origin incomplete, metric 0, localpref 100, valid, internal, multipath
    Received Path ID 2, Local Path ID 0, version 0
    Originator: 13.13.13.13, Cluster list: 9.9.9.9

RP/0/0/CPU0:XRv14#show route ipv4 unicast 28.10.45.0/25
Routing entry for 28.10.45.0/25
  Known via "bgp 100", distance 200, metric 0
  Tag 6, type internal
  Routing Descriptor Blocks
    5.5.5.5, from 8.8.8.8, BGP multi path
    Route metric is 0
    12.12.12.12, from 9.9.9.9, BGP multi path
    Route metric is 0
    13.13.13.13, from 9.9.9.9, BGP multi path
    Route metric is 0
  No advertising protos.

RP/0/0/CPU0:XRv14#show cef ipv4 28.10.45.0/25
28.10.45.0/25, version 2922, internal 0x5000001 0x0 (ptr 0xa13d2b74) [1], 0x0
(0x0), 0x0 (0x0)
  local adjacency 153.9.14.9
  Prefix Len 25, traffic index 0, precedence n/a, priority 4
  via 5.5.5.5, 2 dependencies, recursive, bgp-multipath [flags 0x6080]
    path-idx 0 NHID 0x0 [0xa13d2574 0x0]

```

```

next hop 5.5.5.5 via 5.5.5.5/32
via 12.12.12.12, 2 dependencies, recursive, bgp-multipath [flags 0x6080]
path-idx 1 NHID 0x0 [0xa13d2a74 0x0]
next hop 12.12.12.12 via 12.12.12.12/32
via 13.13.13.13, 2 dependencies, recursive, bgp-multipath [flags 0x6080]
path-idx 2 NHID 0x0 [0xa13d3174 0x0]
next hop 13.13.13.13 via 13.13.13.13/32

```

Now we will shift focus to CSR5 and XRv13. CSR5 learns two paths to 28.10.45.0/25; one via eBGP and one via CSR9. Interestingly, the only reason CSR5 learns the CSR9 route is due to CSR9 sending its best 2 paths, since the path via CSR5 is the best path for both RRs (coincidental). A quick check on CSR9 confirms this.

```

CSR9#show bgp ipv4 unicast neighbors 5.5.5.5 advertised-routes | include
28.10.45.0/25
*>i 28.10.45.0/25      5.5.5.5              0      100      0 6 78 454 ?
*bia28.10.45.0/25    12.12.12.12           0      100      0 6 78 454 ?

```

```

CSR5#show bgp ipv4 unicast 28.10.45.0/25
BGP routing table entry for 28.10.45.0/25, version 10
Paths: (2 available, best #2, table default)
  Advertised to update-groups:
    8
  Refresh Epoch 4
  6 78 454
    12.12.12.12 (metric 20) from 9.9.9.9 (9.9.9.9)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Originator: 12.12.12.12, Cluster list: 9.9.9.9
      rx pathid: 0x1, tx pathid: 0
  Refresh Epoch 2
  6 78 454
    10.5.6.6 from 10.5.6.6 (6.6.6.6)
      Origin incomplete, metric 0, localpref 100, valid, external, best
      rx pathid: 0, tx pathid: 0x0

```

XRv13, on the other hand, will show 4 routes. First comes the expected eBGP route, its best path. Both RR's select the path via CSR5 and reflect that to XRv13. Additionally, CSR9 sends its second best path to XRv13 also. Thus, two routes have a next hop of CSR5, one has a next hop of XRv12, and one is the eBGP route. The eBGP route is preferred over the iBGP ones since no BGP attributes have been modified.

```

CSR9#show bgp ipv4 unicast neighbors 13.13.13.13 advertised-routes | include
28.10.45.0/25
*>i 28.10.45.0/25      5.5.5.5              0      100      0 6 78 454 ?
*bia28.10.45.0/25    12.12.12.12           0      100      0 6 78 454 ?

RP/0/0/CPU0:XRv13#show bgp ipv4 un 28.10.45.0/25 | begin Paths
Paths: (4 available, best #4)

```

```

Not advertised to any peer
Path #1: Received by speaker 0
Not advertised to any peer
6 78 454
  5.5.5.5 (metric 20) from 8.8.8.8 (5.5.5.5)
    Origin incomplete, metric 0, localpref 100, valid, internal
    Received Path ID 0, Local Path ID 0, version 0
    Originator: 5.5.5.5, Cluster list: 8.8.8.8
Path #2: Received by speaker 0
Not advertised to any peer
6 78 454
  5.5.5.5 (metric 20) from 9.9.9.9 (5.5.5.5)
    Origin incomplete, metric 0, localpref 100, valid, internal
    Received Path ID 0, Local Path ID 0, version 0
    Originator: 5.5.5.5, Cluster list: 9.9.9.9
Path #3: Received by speaker 0
Not advertised to any peer
6 78 454
  12.12.12.12 (metric 20) from 9.9.9.9 (12.12.12.12)
    Origin incomplete, metric 0, localpref 100, valid, internal
    Received Path ID 1, Local Path ID 0, version 0
    Originator: 12.12.12.12, Cluster list: 9.9.9.9
Path #4: Received by speaker 0
Not advertised to any peer
6 78 454
  10.6.13.6 from 10.6.13.6 (6.6.6.6)
    Origin incomplete, metric 0, localpref 100, valid, external, best,
group-best
    Received Path ID 0, Local Path ID 1, version 103
    Origin-AS validity: not-found

```

Using local-preference on XRv13, will make it the desired exit point for this specific prefix. We will craft a parameterized RPL for extra practice to accomplish this. Notice that XRv13 only has one route now, its best path.

```

! XRv13
prefix-set PS_COOL_ROUTE
  28.10.45.0/25
end-set

route-policy RPL_SET_LPREF_TO_X_FOR_Y($X, $Y)
  if destination in $Y then
    set local-preference $X
  else
    pass
  endif
end-policy

router bgp 100

```

```

neighbor 10.6.13.6
  remote-as 6
  address-family ipv4 unicast
  route-policy RPL_SET_LPREF_TO_X_FOR_Y(200, PS_COOL_ROUTE) in

```

```
RP/0/0/CPU0:XRv13#show bgp ipv4 unicast 28.10.45.0/25 | begin Paths
```

```

Paths: (1 available, best #1)
  Advertised to update-groups (with more than one peer):
    0.1
  Path #1: Received by speaker 0
  Advertised to update-groups (with more than one peer):
    0.1
  6 78 454
  10.6.13.6 from 10.6.13.6 (6.6.6.6)
    Origin incomplete, metric 0, localpref 200, valid, external, best,
group-best
  Received Path ID 0, Local Path ID 1, version 110
  Origin-AS validity: not-found

```

This implies that CSR9 must not have any secondary paths or else we would have seen them reflected back to XRv13. We confirm this below. The reason is because now all of the other PEs to AS 6, such as CSR5 and XRv2, all prefer this iBGP path and do not advertise their eBGP path. Even though they have multiple iBGP paths, they are just reflections of the same local-preference 200 path. The RR's never learn anything unique, and by extension the remote PEs never learn anything unique, making the network somewhat stovepiped.

```

CSR9#show bgp ipv4 unicast 28.10.45.0/25
BGP routing table entry for 28.10.45.0/25, version 373
BGP Bestpath: igpmetric-ignore
Paths: (1 available, best #1, table default)
  Additional-path-install
  Path advertised to update-groups:
    6          7          8
  Refresh Epoch 1
  6 78 454, (Received from a RR-client)
  13.13.13.13 (metric 10) from 13.13.13.13 (13.13.13.13)
    Origin incomplete, metric 0, localpref 200, valid, internal, best
  rx pathid: 0, tx pathid: 0x0

```

```

CSR5#show bgp ipv4 unicast 28.10.45.0/25
BGP routing table entry for 28.10.45.0/25, version 54
Paths: (3 available, best #1, table default)
  Advertised to update-groups:
    7
  Refresh Epoch 2
  6 78 454
  13.13.13.13 (metric 20) from 8.8.8.8 (8.8.8.8)
    Origin incomplete, metric 0, localpref 200, valid, internal, best

```

```

    Originator: 13.13.13.13, Cluster list: 8.8.8.8
    rx pathid: 0x0, tx pathid: 0x0
Refresh Epoch 4
6 78 454
    13.13.13.13 (metric 20) from 9.9.9.9 (9.9.9.9)
    Origin incomplete, metric 0, localpref 200, valid, internal
    Originator: 13.13.13.13, Cluster list: 9.9.9.9
    rx pathid: 0x0, tx pathid: 0
Refresh Epoch 2
6 78 454
    10.5.6.6 from 10.5.6.6 (6.6.6.6)
    Origin incomplete, metric 0, localpref 100, valid, external
    rx pathid: 0, tx pathid: 0

RP/0/0/CPU0:XRv12#show bgp ipv4 unicast 28.10.45.0/25 | begin Paths
Paths: (3 available, best #1)
    Not advertised to any peer
    Path #1: Received by speaker 0
    Not advertised to any peer
    6 78 454
    13.13.13.13 (metric 20) from 8.8.8.8 (13.13.13.13)
    Origin incomplete, metric 0, localpref 200, valid, internal, best,
group-best
    Received Path ID 0, Local Path ID 1, version 73
    Originator: 13.13.13.13, Cluster list: 8.8.8.8
    Path #2: Received by speaker 0
    Not advertised to any peer
    6 78 454
    13.13.13.13 (metric 20) from 9.9.9.9 (13.13.13.13)
    Origin incomplete, metric 0, localpref 200, valid, internal
    Received Path ID 0, Local Path ID 0, version 0
    Originator: 13.13.13.13, Cluster list: 9.9.9.9
    Path #3: Received by speaker 0
    Not advertised to any peer
    6 78 454
    10.6.12.6 from 10.6.12.6 (6.6.6.6)
    Origin incomplete, metric 0, localpref 100, valid, external
    Received Path ID 0, Local Path ID 0, version 0
    Origin-AS validity: not-found

```

We can do some local damage-repair by configuring the PE's to install the eBGP route as a backup path. It doesn't really help the remote PEs on the other side of the network, like CSR3 and XRv14, but it's a small improvement. The XE command is simple for CSR5 but again, XRv12 requires a little more work. We will use a simple RPL for this one. As always, verify the FIB too (RIB verification doesn't matter because we are not load-sharing).

```

! XRv12
route-policy RPL_BACKUP_ALL

```

```

    set path-selection backup 1 install
end-policy

router bgp 100
  address-family ipv4 unicast
    additional-paths selection route-policy RPL_BACKUP_ALL

RP/0/0/CPU0:XRv12#show bgp ipv4 unicast 28.10.45.0/25 | begin Paths
Paths: (3 available, best #1)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  6 78 454
    13.13.13.13 (metric 20) from 8.8.8.8 (13.13.13.13)
      Origin incomplete, metric 0, localpref 200, valid, internal, best,
group-best
      Received Path ID 0, Local Path ID 1, version 73
      Originator: 13.13.13.13, Cluster list: 8.8.8.8
  Path #2: Received by speaker 0
  Not advertised to any peer
  6 78 454
    13.13.13.13 (metric 20) from 9.9.9.9 (13.13.13.13)
      Origin incomplete, metric 0, localpref 200, valid, internal
      Received Path ID 0, Local Path ID 0, version 0
      Originator: 13.13.13.13, Cluster list: 9.9.9.9
  Path #3: Received by speaker 0
  Not advertised to any peer
  6 78 454
    10.6.12.6 from 10.6.12.6 (6.6.6.6)
      Origin incomplete, metric 0, localpref 100, valid, external, backup,
add-path
      Received Path ID 0, Local Path ID 2, version 96
      Origin-AS validity: not-found

CSR5#show bgp ipv4 unicast 28.10.45.0/25
BGP routing table entry for 28.10.45.0/25, version 60
Paths: (3 available, best #1, table default)
  Additional-path-install
  Advertised to update-groups:
    7
  Refresh Epoch 2
  6 78 454
    13.13.13.13 (metric 20) from 8.8.8.8 (8.8.8.8)
      Origin incomplete, metric 0, localpref 200, valid, internal, best
      Originator: 13.13.13.13, Cluster list: 8.8.8.8
      rx pathid: 0x0, tx pathid: 0x0
  Refresh Epoch 4
  6 78 454
    13.13.13.13 (metric 20) from 9.9.9.9 (9.9.9.9)

```

```

        Origin incomplete, metric 0, localpref 200, valid, internal
        Originator: 13.13.13.13, Cluster list: 9.9.9.9
        rx pathid: 0x0, tx pathid: 0
Refresh Epoch 2
6 78 454
    10.5.6.6 from 10.5.6.6 (6.6.6.6)
        Origin incomplete, metric 0, localpref 100, valid, external,
backup/repair , recursive-via-connected
        rx pathid: 0, tx pathid: 0

CSR5#show ip cef 28.10.45.0 detail
28.10.45.0/25, epoch 2, flags [rib only nolabel, rib defined all labels]
    recursive via 13.13.13.13
        nexthop 153.5.9.9 GigabitEthernet2.559 label 9000
    recursive via 10.5.6.6, repair
        attached to GigabitEthernet2.556

RP/0/0/CPU0:XRv12#show cef ipv4 28.10.45.0/25
28.10.45.0/25, version 1286, internal 0x5000001 0x0 (ptr 0xa14803f4) [1], 0x0
(0x0), 0x0 (0x0)
local adjacency 153.8.12.8
Prefix Len 25, traffic index 0, precedence n/a, priority 4
    via 10.6.12.6, 3 dependencies, recursive, bgp-ext, backup [flags 0x6120]
        path-idx 0 NHID 0x0 [0xa147f5f4 0x0]
        next hop 10.6.12.6 via 10.6.12.6/32
    via 13.13.13.13, 2 dependencies, recursive [flags 0x6000]
        path-idx 1 NHID 0x0 [0xa147ef74 0x0]
        next hop 13.13.13.13 via 13.13.13.13/32

```

A good option for introducing more path diversity into the iBGP domain is allowing the "loser" PE's to advertise their best external route. You can enable it on all the PE's in case path selections change, but in this case, I just enable it on CSR5 and XRv12. Beware that in order for best-external to work, you must remove the negotiated add-path capability, since features like this and the shadow RR are independent (and sometimes mutually exclusive) from full-blown add-path. Now CSR5 marks this as its best eBGP path and still treats it as a fast-reroute backup.

```

CSR5(config-router-ptmp)#advertise best-external
% BGP: Addpath capability present. When negotiated, it will cause advertise
best-external to be silently disabled. Inheritance accepted anyway for
8.8.8.8

```

```

! CSR5
router bgp 100
template peer-policy IPV4
    advertise best-external
address-family ipv4
    bgp additional-paths select best-external
    no bgp additional-paths receive

```

```

CSR5#show bgp ipv4 unicast 28.10.45.0/25
BGP routing table entry for 28.10.45.0/25, version 136
Paths: (3 available, best #1, table default)
  Additional-path-install
  Advertised to update-groups:
    11
  Refresh Epoch 1
  6 78 454
    13.13.13.13 (metric 20) from 8.8.8.8 (8.8.8.8)
      Origin incomplete, metric 0, localpref 200, valid, internal, best
      Originator: 13.13.13.13, Cluster list: 8.8.8.8
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  6 78 454
    13.13.13.13 (metric 20) from 9.9.9.9 (9.9.9.9)
      Origin incomplete, metric 0, localpref 200, valid, internal
      Originator: 13.13.13.13, Cluster list: 9.9.9.9
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 2
  6 78 454
    10.5.6.6 from 10.5.6.6 (6.6.6.6)
      Origin incomplete, metric 0, localpref 100, valid, external,
      backup/repair, advertise-best-external , recursive-via-connected
      rx pathid: 0, tx pathid: 0

```

Similarly, we configure the feature on XRv12 so the RRs can learn its eBGP route also. XR seems content to advertise the best-external even when add-path capability is negotiated, unlike XE. It marks its best (and only) eBGP path as best external, and it's still a valid backup path.

```

! XRv12
router bgp 100
  address-family ipv4 unicast
    advertise best-external

RP/0/0/CPU0:XRv12#show bgp ipv4 unicast 28.10.45.0/25
BGP routing table entry for 28.10.45.0/25
Versions:
  Process          bRIB/RIB  SendTblVer
  Speaker          164      164
Paths: (4 available, best #1)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  6 78 454
    13.13.13.13 (metric 20) from 8.8.8.8 (13.13.13.13)
      Origin incomplete, metric 0, localpref 200, valid, internal, best,
group-best
  Received Path ID 0, Local Path ID 1, version 73

```



```

    Originator: 13.13.13.13, Cluster list: 8.8.8.8
Path #2: Received by speaker 0
Not advertised to any peer
6 78 454
    13.13.13.13 (metric 20) from 9.9.9.9 (13.13.13.13)
        Origin incomplete, metric 0, localpref 200, valid, internal
        Received Path ID 0, Local Path ID 0, version 0
        Originator: 13.13.13.13, Cluster list: 9.9.9.9
Path #3: Received by speaker 0
Not advertised to any peer
6 78 454
    5.5.5.5 (metric 20) from 9.9.9.9 (5.5.5.5)
        Origin incomplete, metric 0, localpref 100, valid, internal
        Received Path ID 1, Local Path ID 0, version 0
        Originator: 5.5.5.5, Cluster list: 9.9.9.9
Path #4: Received by speaker 0
Advertised to update-groups (with more than one peer):
    0.1
6 78 454
    10.6.12.6 from 10.6.12.6 (6.6.6.6)
        Origin incomplete, metric 0, localpref 100, valid, external, best-
external, backup, add-path
        Received Path ID 0, Local Path ID 3, version 164
        Origin-AS validity: not-found

```

A quick check of CSR8 and CSR9 show the paths. We did not configure any add-path options on CSR8, other than the basic negotiation, so it is still a classic RR in that it only sends its best path. CSR9, however, has now selected its three best paths. Without advertising the best-external paths from the “loser” PEs in the first place, the RRs would have never seen these additional paths in the first place. We can tell these are the “loser” paths because of the relatively low local-preference value.

```

CSR8#show bgp ipv4 unicast 28.10.45.0/25
BGP routing table entry for 28.10.45.0/25, version 185
BGP Bestpath: igpmetric-ignore
Paths: (3 available, best #3, table default)
    Advertised to update-groups:
        3          4
Refresh Epoch 1
6 78 454, (Received from a RR-client)
    12.12.12.12 (metric 10) from 12.12.12.12 (12.12.12.12)
        Origin incomplete, metric 0, localpref 100, valid, internal
        rx pathid: 0, tx pathid: 0
Refresh Epoch 2
6 78 454, (Received from a RR-client)
    5.5.5.5 (metric 30) from 5.5.5.5 (5.5.5.5)
        Origin incomplete, metric 0, localpref 100, valid, internal
        rx pathid: 0, tx pathid: 0
Refresh Epoch 1

```

```

6 78 454, (Received from a RR-client)
13.13.13.13 (metric 10) from 13.13.13.13 (13.13.13.13)
Origin incomplete, metric 0, localpref 200, valid, internal, best
rx pathid: 0, tx pathid: 0x0

CSR9#show bgp ipv4 unicast 28.10.45.0/25
BGP routing table entry for 28.10.45.0/25, version 433
BGP Bestpath: igpmetric-ignore
Paths: (3 available, best #3, table default)
Additional-path-install
Path advertised to update-groups:
8
Refresh Epoch 1
6 78 454, (Received from a RR-client)
12.12.12.12 (metric 30) from 12.12.12.12 (12.12.12.12)
Origin incomplete, metric 0, localpref 100, valid, internal, best3
rx pathid: 0, tx pathid: 0x2
Path advertised to update-groups:
7      8
Refresh Epoch 1
6 78 454, (Received from a RR-client)
5.5.5.5 (metric 10) from 5.5.5.5 (5.5.5.5)
Origin incomplete, metric 0, localpref 100, valid, internal,
backup/repair, best2
rx pathid: 0, tx pathid: 0x1
Path advertised to update-groups:
7      8      9
Refresh Epoch 1
6 78 454, (Received from a RR-client)
13.13.13.13 (metric 10) from 13.13.13.13 (13.13.13.13)
Origin incomplete, metric 0, localpref 200, valid, internal, best
rx pathid: 0, tx pathid: 0x0

```

Now, CSR3 and XRv14 can see the additional paths in the network from the RRs. They still select the proper "best" path because the whole reason one uses local-preference in the first place is for traffic engineering, and add-path honors that. The backup paths they select are the next-best paths with varying next-hops as we expected. However, even with this feature enabled, you should also have either add-path or shadow RR configured, otherwise all the RRs will simply pick the higher local-preference and ignore all the other best-externals. If add-path is not supported, you can use the shadow method by itself, which we examine in VPNv4 tests. Here, we can clearly see XRv13 as the primary path with CSR5 as the backup.

```

CSR3#show bgp ipv4 unicast 28.10.45.0/25
BGP routing table entry for 28.10.45.0/25, version 467
Paths: (3 available, best #2, table default)
Additional-path-install
Advertised to update-groups:
4

```

```

Refresh Epoch 4
6 78 454
5.5.5.5 (metric 20) from 9.9.9.9 (9.9.9.9)
  Origin incomplete, metric 0, localpref 100, valid, internal,
backup/repair
  Originator: 5.5.5.5, Cluster list: 9.9.9.9
  rx pathid: 0x1, tx pathid: 0
Refresh Epoch 2
6 78 454
13.13.13.13 (metric 20) from 8.8.8.8 (8.8.8.8)
  Origin incomplete, metric 0, localpref 200, valid, internal, best
  Originator: 13.13.13.13, Cluster list: 8.8.8.8
  rx pathid: 0x0, tx pathid: 0x0
Refresh Epoch 4
6 78 454
13.13.13.13 (metric 20) from 9.9.9.9 (9.9.9.9)
  Origin incomplete, metric 0, localpref 200, valid, internal
  Originator: 13.13.13.13, Cluster list: 9.9.9.9
  rx pathid: 0x0, tx pathid: 0

RP/0/0/CPU0:XRv14#show bgp ipv4 unicast 28.10.45.0/25
BGP routing table entry for 28.10.45.0/25
Versions:
  Process          bRIB/RIB  SendTblVer
  Speaker          394      394
Paths: (4 available, best #1)
  Advertised to update-groups (with more than one peer):
    0.3
  Path #1: Received by speaker 0
  Advertised to update-groups (with more than one peer):
    0.3
6 78 454
13.13.13.13 (metric 20) from 8.8.8.8 (13.13.13.13)
  Origin incomplete, metric 0, localpref 200, valid, internal, best,
group-best
  Received Path ID 0, Local Path ID 1, version 314
  Originator: 13.13.13.13, Cluster list: 8.8.8.8
  Path #2: Received by speaker 0
  Not advertised to any peer
6 78 454
13.13.13.13 (metric 20) from 9.9.9.9 (13.13.13.13)
  Origin incomplete, metric 0, localpref 200, valid, internal
  Received Path ID 0, Local Path ID 0, version 0
  Originator: 13.13.13.13, Cluster list: 9.9.9.9
  Path #3: Received by speaker 0
  Not advertised to any peer
6 78 454
5.5.5.5 (metric 20) from 9.9.9.9 (5.5.5.5)

```

```

Origin incomplete, metric 0, localpref 100, valid, internal, backup,
add-path
Received Path ID 1, Local Path ID 4, version 394
Originator: 5.5.5.5, Cluster list: 9.9.9.9
Path #4: Received by speaker 0
Not advertised to any peer
6 78 454
12.12.12.12 (metric 20) from 9.9.9.9 (12.12.12.12)
Origin incomplete, metric 0, localpref 100, valid, internal
Received Path ID 2, Local Path ID 0, version 0
Originator: 12.12.12.12, Cluster list: 9.9.9.9

```

Another useful feature is group-best. This feature groups incoming routes together by AS, selects a best path, then picks the next best path from the group NOT from the same AS. CSR1, CSR2, and CSR4 all advertise 124.0.0.1/32 to CSR3. I've created three basic route-maps that set the local-preference to 101, 102, and 104 on this route inbound from CSR1, CSR2, and CSR4 respectively. CSR4 is the best path, with CSR2 as the "group-best" for AS 12. Another key point with group-best is that it is technically an additional-path and requires the negotiated capability. On CSR3 only, we enabled send/receive, along with CSR9. The other PE's are receive only and the other RRs are send only. CSR3's changes are below.

```

!CSR3
router bgp 100
template peer-policy IPV4
advertise additional-paths group-best
address-family ipv4
bgp additional-paths select group-best
bgp additional-paths send receive
neighbor 10.1.3.1 route-map RM_LPREF_GROUP_TEST_CSR1 in
neighbor 10.2.3.2 route-map RM_LPREF_GROUP_TEST_CSR2 in
neighbor 10.3.4.4 route-map RM_LPREF_GROUP_TEST_CSR4 in

CSR3#show bgp ipv4 unicast 124.0.0.1/32
BGP routing table entry for 124.0.0.1/32, version 787
Paths: (3 available, best #3, table default)
Additional-path-install
Flag: 0x820
Path not advertised to any peer
Refresh Epoch 10
12
10.1.3.1 from 10.1.3.1 (1.1.1.1)
Origin incomplete, metric 0, localpref 101, valid, external ,
recursive-via-connected
rx pathid: 0, tx pathid: 0
Path advertised to update-groups: (Pending Update Generation)
11
Refresh Epoch 10
12
10.2.3.2 from 10.2.3.2 (2.2.2.2)

```

```

Origin incomplete, metric 0, localpref 102, valid, external,
backup/repair, group-best , recursive-via-connected
rx pathid: 0, tx pathid: 0x2
Path advertised to update-groups: (Pending Update Generation)
4          10          11
Refresh Epoch 8
410
10.3.4.4 from 10.3.4.4 (4.4.4.4)
Origin incomplete, metric 0, localpref 104, valid, external, best ,
recursive-via-connected
rx pathid: 0, tx pathid: 0x0

```

As expected, CSR3 will advertise this group-best path to CSR9 as a backup, additional path (b and a flags). Notice that CSR3 does not advertise it to CSR8 because CSR8 is not configured to receive additional paths. CSR9 advertises the ability to send/receive, while CSR8 only advertises the ability to send. Since each capability is distinctly different, I recommend enabling both everywhere to avoid the confusion if possible.

```

CSR3#show bgp ipv4 unicast neighbors 9.9.9.9 | include Addition
Additional Paths send capability: advertised and received
Additional Paths receive capability: advertised and received

```

```

CSR3#show bgp ipv4 unicast neighbors 9.9.9.9 advertised-routes | include
124.0.0.1/32
*> 124.0.0.1/32      10.3.4.4          0      104      0 410 ?
*b a124.0.0.1/32    10.2.3.2          0      102      0 12 ?

```

```

CSR3#show bgp ipv4 unicast neighbors 8.8.8.8 | include Addition
Additional Paths send capability: advertised and received
Additional Paths receive capability: advertised

```

```

CSR3#show bgp ipv4 unicast neighbors 8.8.8.8 advertised-routes | include
124.0.0.1/32
*> 124.0.0.1/32      10.3.4.4          0      104      0 410 ?

```

To demonstrate the behavior of group-best specifically, we will reduce the local-preference of CSR4 to 90. The best path becomes CSR2, but BGP cannot consider CSR1 as the next best path despite its local-preference being higher than CSR4. This is because CSR1 and CSR2 share an AS, so the group-best should have a different next-hop AS; this is a fault-tolerance mechanism. What is also interesting is that CSR3 uses the path through CSR1 as its local backup path because it's the second best, but advertises the group-best, which is CSR4. We can prove both points by checking the BGP RIB, FIB (for backup path), and neighbor advertised routes (for group-best). The advertised group-best route is no longer flagged with 'b' for backup, because it isn't one. I also include the "brief" view of the BGP table to demonstrate the differences between the three routes in summary form. All three routes are different: one best, one backup, and one additional group-best.

```

CSR3#show bgp ipv4 unicast 124.0.0.1/32
BGP routing table entry for 124.0.0.1/32, version 788
Paths: (3 available, best #2, table default)
  Additional-path-install
  Path not advertised to any peer
  Refresh Epoch 11
  12
    10.1.3.1 from 10.1.3.1 (1.1.1.1)
      Origin incomplete, metric 0, localpref 101, valid, external,
      backup/repair , recursive-via-connected
      rx pathid: 0, tx pathid: 0
      Path advertised to update-groups:
        4          10          11
      Refresh Epoch 11
  12
    10.2.3.2 from 10.2.3.2 (2.2.2.2)
      Origin incomplete, metric 0, localpref 102, valid, external, best ,
      recursive-via-connected
      rx pathid: 0, tx pathid: 0x0
      Path advertised to update-groups:
        11
      Refresh Epoch 9
  410
    10.3.4.4 from 10.3.4.4 (4.4.4.4)
      Origin incomplete, metric 0, localpref 90, valid, external, group-best
      , recursive-via-connected
      rx pathid: 0, tx pathid: 0x2

CSR3#show ip cef 124.0.0.1 detail
124.0.0.1/32, epoch 2, flags [rib only nolabel, rib defined all labels]
  recursive via 10.2.3.2
    attached to GigabitEthernet2.523
  recursive via 10.1.3.1, repair
    attached to GigabitEthernet2.513

CSR3#show bgp ipv4 unicast neighbors 9.9.9.9 advertised-routes | include
124.0.0.1/32
*> 124.0.0.1/32      10.2.3.2          0      102      0 12 ?
*  a124.0.0.1/32    10.3.4.4          0       90      0 410 ?

CSR3#show bgp ipv4 unicast | begin 124.0.0.1
*b 124.0.0.1/32      10.1.3.1          0      101      0 12 ?
*>                10.2.3.2          0      102      0 12 ?
*  a                10.3.4.4          0       90      0 410 ?

```

So far XRv11 has not been participating much in this test, mostly because of its inability to disable the IGP metric from its best path selection. This has a more significant effect on non-add-path features like shadow RR and best-external. To test XR as a RR with add-path, we will allow CSR8 and CSR9 to consider

IGP metric in their calculations again just so all RRs behave the same. We also disable ""best-external" from CSR5 so we are allowed to enable add-path negotiations. We then disable the add-path advertisements on CSR9. Essentially, we are doing some basic clean-up so that we can focus on XR.

```
! CSR5
router bgp 100
  template peer-policy IPV4
    no advertise best-external
  address-family ipv4
    no bgp additional-paths select best-external
    bgp additional-paths receive

! CSR8
router bgp 100
  address-family ipv4
    no bgp bestpath igp-metric ignore

! CSR9
router bgp 100
  template peer-policy IPV4
  advertise additional-paths best 2
  address-family ipv4
    no bgp bestpath igp-metric ignore
    no bgp additional-paths select best 3
    no neighbor 14.14.14.14 advertise additional-paths best 3

! XRv11
router bgp 100
  session-group IBGP
  no shutdown

! XRv12
router bgp 100
  address-family ipv4 unicast
  no advertise best-external
```

Let's quickly verify that XRv11 has properly negotiated the add-path capability, only because we haven't used it in awhile. Remember that we added the "send" capability on CSR3, the only PE with it configured. The remaining PEs are receive-capable while XRv11 advertises send capability.

```
RP/0/0/CPU0:XRv11#show bgp ipv4 unicast neighbors | utility egrep '^BGP
neigh|Additional'
BGP neighbor is 3.3.3.3
  Additional-paths Send: advertised and received
  Additional-paths Receive: received
  Additional-paths operation: Send
BGP neighbor is 5.5.5.5
  Additional-paths Send: advertised
```

```

    Additional-paths Receive: received
    Additional-paths operation: Send
    Additional-paths operation: None
BGP neighbor is 12.12.12.12
    Additional-paths Send: advertised
    Additional-paths Receive: received
    Additional-paths operation: Send
    Additional-paths operation: None
BGP neighbor is 13.13.13.13
    Additional-paths Send: advertised
    Additional-paths Receive: received
    Additional-paths operation: Send
    Additional-paths operation: None
BGP neighbor is 14.14.14.14
    Additional-paths Send: advertised
    Additional-paths Receive: received
    Additional-paths operation: Send
    Additional-paths operation: None

```

We solved the local-preference 200 problem earlier using advertise-best-external, which allowed CSR5 and XRv12 to send their eBGP routes to the RRs. This time, we will enable BGP add-path receive capability on XRv11 and add-path send capability on CSR5 / XRv12. Then, we verify the negotiations were successful. We see that with CSR5 and XRv12, both send and receive capabilities are negotiated bidirectionally. The configuration is not shown as we have seen it many times so far.

```

RP/0/0/CPU0:XRv11#show bgp ipv4 unicast neighbors | utility egrep '^BGP
neigh|Additional'
BGP neighbor is 3.3.3.3
    Additional-paths Send: advertised and received
    Additional-paths Receive: advertised and received
    Additional-paths operation: Send and Receive
BGP neighbor is 5.5.5.5
    Additional-paths Send: advertised and received
    Additional-paths Receive: advertised and received
    Additional-paths operation: Send and Receive
    Additional-paths operation: None
BGP neighbor is 12.12.12.12
    Additional-paths Send: advertised and received
    Additional-paths Receive: advertised and received
    Additional-paths operation: Send and Receive
    Additional-paths operation: None
BGP neighbor is 13.13.13.13
    Additional-paths Send: advertised
    Additional-paths Receive: advertised and received
    Additional-paths operation: Send
    Additional-paths operation: None
BGP neighbor is 14.14.14.14
    Additional-paths Send: advertised

```


Additional-paths Receive: advertised and received
Additional-paths operation: Send
Additional-paths operation: None

Recall that XRv13 is setting local-preference 200 inbound for the prefix 28.10.45.0/25, making it the preferred exit point from AS 100. All of the PE's on the other side of the SP network are going to have three copies of that route, one from each RR, since each RR only learns this one path, it is the only thing it can advertise. The PE's also have their less-preferred eBGP routes which can be considered additional paths. We check the detailed output on CSR3 and the brief output on XRv14 to prove it.

```
CSR5#show bgp ipv4 un 28.10.45.0/25
BGP routing table entry for 28.10.45.0/25, version 444
Paths: (4 available, best #2, table default)
  Additional-path-install
  Advertised to update-groups:
    13
  Refresh Epoch 1
  6 78 454
    13.13.13.13 (metric 20) from 11.11.11.11 (11.11.11.11)
      Origin incomplete, metric 0, localpref 200, valid, internal
      Originator: 13.13.13.13, Cluster list: 11.11.11.11
      rx pathid: 0x1, tx pathid: 0
  Refresh Epoch 2
  6 78 454
    13.13.13.13 (metric 20) from 8.8.8.8 (8.8.8.8)
      Origin incomplete, metric 0, localpref 200, valid, internal, best
      Originator: 13.13.13.13, Cluster list: 8.8.8.8
      rx pathid: 0x0, tx pathid: 0x0
  Refresh Epoch 2
  6 78 454
    13.13.13.13 (metric 20) from 9.9.9.9 (9.9.9.9)
      Origin incomplete, metric 0, localpref 200, valid, internal
      Originator: 13.13.13.13, Cluster list: 9.9.9.9
      rx pathid: 0x0, tx pathid: 0
  Refresh Epoch 2
  6 78 454
    10.5.6.6 from 10.5.6.6 (6.6.6.6)
      Origin incomplete, metric 0, localpref 100, valid, external,
      backup/repair, recursive-via-connected
      rx pathid: 0, tx pathid: 0
```

```
RP/0/0/CPU0:XRv12#show bgp ipv4 unicast 28.10.45.0/25 brief | begin Network
  Network          Next Hop          Metric LocPrf Weight Path
*>i28.10.45.0/25    13.13.13.13          0      200      0 6 78 454 ?
* i                13.13.13.13          0      200      0 6 78 454 ?
*                  10.6.12.6            0              0 6 78 454 ?
* i                13.13.13.13          0      200      0 6 78 454 ?
```

Next, we configure both CSR5 and XRv12 to select all paths as additional paths. XRv12 automatically advertises them to neighbors without a neighbor-level command, while CSR5 requires it. BGP is smart enough to know that the three copies of the exact same route on CSR5 and XRv12 do not qualify as additional paths, and additionally, the PE's are not RRs (so basic iBGP says we cannot advertise them further). We have effectively allowed the PE's to advertise their best external routes using add-path capabilities as an alternative method to achieve the same result. A quick look at XRv11 confirms this; notice the variance in next-hops and local-preference values. BGP still selects the proper best-path, though. There is still a problem; XRv11 is not advertising any of its additional-paths to clients on the other side of the network, such as CSR3 and XRv14. BGP add-path is less effective if only the RR's have visibility of the additional paths.

```
RP/0/0/CPU0:XRv11#show bgp ipv4 unicast 28.10.45.0/25 brief | begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
* i28.10.45.0/25	5.5.5.5	0	100	0	6 78 454 ?
* i	12.12.12.12	0	100	0	6 78 454 ?
*>i	13.13.13.13	0	200	0	6 78 454 ?

```
RP/0/0/CPU0:XRv11#show bgp ipv4 unicast neighbors 3.3.3.3 advertised-routes
[snip]
```

Network	Next Hop	Metric	LocPrf	Weight	Path
28.10.44.128/25	5.5.5.5	5.5.5.5	6	78	454?
28.10.45.0/25	13.13.13.13	13.13.13.13	6	78	454?
28.10.45.128/25	5.5.5.5	5.5.5.5	6	78	454?

```
[snip]
```

We can correct this by specifically telling XRv11 to advertise backup paths. For some reason, XRv only lets me use the number 1, even with parameterization, but the documentation claims it can be between 0 and 7 (3 bits). After applying this configuration snippet, XRv11 selects a single back-up path via CSR5 (lower RID than XRv12) and advertises this to its neighbors. We verify the advertisement and also that the route is marked as "backup" and "add-path" in the BGP table. This is equivalent to "best2" in XE.

```
! XRv11
route-policy RPL_ADV_ADD_PATHS
  if destination in (28.10.45.0/25) then
    set path-selection backup 1 advertise
  endif
end-policy

router bgp 100
  address-family ipv4 unicast
    no additional-paths selection route-policy RPL
    additional-paths selection route-policy RPL_ADV_ADD_PATHS
```

```
RP/0/0/CPU0:XRv11#show bgp ipv4 unicast neighbors 3.3.3.3 advertised-routes |
begin 28.10.45.0/25
```

Network	Next Hop	Metric	LocPrf	Weight	Path
28.10.45.0/25	13.13.13.13	13.13.13.13	6	78	454?
	5.5.5.5	5.5.5.5	6	78	454?

[snip]

RP/0/0/CPU0:XRv11#show bgp ipv4 unicast 28.10.45.0/25

BGP routing table entry for 28.10.45.0/25

Versions:

Process	bRIB/RIB	SendTblVer
Speaker	290	290

Paths: (3 available, best #3)

Advertised to update-groups (with more than one peer):

0.1

Path #1: Received by speaker 0

Advertised to update-groups (with more than one peer):

0.1

6 78 454, (Received from a RR-client)

5.5.5.5 (metric 10) from 5.5.5.5 (5.5.5.5)

Origin incomplete, metric 0, localpref 100, valid, internal, backup,
add-path

Received Path ID 1, Local Path ID 2, version 290

Path #2: Received by speaker 0

Not advertised to any peer

6 78 454, (Received from a RR-client)

12.12.12.12 (metric 10) from 12.12.12.12 (12.12.12.12)

Origin incomplete, metric 0, localpref 100, valid, internal

Received Path ID 4, Local Path ID 0, version 0

Path #3: Received by speaker 0

Advertised to update-groups (with more than one peer):

0.1

6 78 454, (Received from a RR-client)

13.13.13.13 (metric 30) from 13.13.13.13 (13.13.13.13)

Origin incomplete, metric 0, localpref 200, valid, internal, best,
group-best

Received Path ID 0, Local Path ID 1, version 247

A summary view of CSR3's BGP table shows this additional route with the 'b' flag for backup. CSR3 was already configured to install these backup paths and was just waiting for XRv11 to advertise it.

CSR3#show bgp ipv4 unicast | begin 28.10.45.0/25

*bi 28.10.45.0/25	5.5.5.5	0	100	0 6 78 454 ?
* i	13.13.13.13	0	200	0 6 78 454 ?
* i	13.13.13.13	0	200	0 6 78 454 ?
*>i	13.13.13.13	0	200	0 6 78 454 ?

[snip]

CSR3#show bgp ipv4 unicast neighbors 11.11.11.11 routes | begin 28.10.45.0/25

*bi 28.10.45.0/25	5.5.5.5	0	100	0 6 78 454 ?
* i	13.13.13.13	0	200	0 6 78 454 ?

It would be even better if we could get the path from XRv12 advertised to our left-side PEs also via XRv11. A simple RPL modification to select all available paths (there doesn't appear to be a best3 equivalent ... its 0, 1, or all) solves this. CSR3 still has 3 copies of the best-path, but has two additional paths received from XRv11. I show the RPL configuration, XRv11 advertisement to CSR3, CSR3 BGP table displaying all 5 paths, and the route reception on CSR3 from XRv11. The path via XRv12 may not be a hot-standby but having it locally can still be valuable for future design changes, such as ECMP, without needing to reconverge BGP.

```
! XRv11
route-policy RPL_ADV_ADD_PATHS
  if destination in (28.10.45.0/25) then
    set path-selection all advertise
  endif
end-policy
```

```
RP/0/0/CPU0:XRv11#show bgp ipv4 unicast neighbors 3.3.3.3 advertised-routes |
begin 28.10.45.0/25
28.10.45.0/25      13.13.13.13      13.13.13.13      6 78 454?
                  5.5.5.5          5.5.5.5          6 78 454?
                  12.12.12.12      12.12.12.12      6 78 454?

[snip]
```

```
CSR3#show bgp ipv4 unicast | begin 28.10.45.0/25
* i 28.10.45.0/25 12.12.12.12      0      100      0 6 78 454 ?
*b i             5.5.5.5          0      100      0 6 78 454 ?
* i             13.13.13.13      0      200      0 6 78 454 ?
* i             13.13.13.13      0      200      0 6 78 454 ?
*>i             13.13.13.13      0      200      0 6 78 454 ?
```

```
CSR3#show bgp ipv4 unicast neighbors 11.11.11.11 routes | begin 28.10.45.0/25
* i 28.10.45.0/25 12.12.12.12      0      100      0 6 78 454 ?
*b i             5.5.5.5          0      100      0 6 78 454 ?
* i             13.13.13.13      0      200      0 6 78 454 ?
```

So far, we have focused on IPv4 unicast. IPv6 unicast works identically, and one would expect all AFs to work this way. Unfortunately, current XE code only supports the send/receive negotiated capability for IPV4/IPv6 unicast/multicast. All other AFs, such as VPNv4, VPNv6, L2VPN variations, RT-filter, NSAP, etc do not currently support this capability. The CSR does support backup path selection and best-external-advertisement for VPNv4 and VPNv6 though, but supports no BGP add-path features for the remaining AFs. XR appears to fully support the feature for all AFs, even new AFs such as BGP-LS and BGP flowspec. That being said, we will quickly test IPv6 unicast in our setup using 6PE as the provider core is not IPv6 enabled. XRv11 BGP process is shutdown for this test as it does not support ignoring the IGP metric in the bestpath calculation.

```
! CSR8 and CSR9
router bgp 100
```

```
address-family ipv6 unicast
  bgp bestpath igp-metric ignore
```

At this point, both CSR8 and CSR9 select CSR5 as their best path due to having the lowest BGP RID. The result on CSR3 and other remote PEs is that they receive multiple copies of the same path with the same BGP next-hop. A failure of CSR5 would break 6PE despite the existence of alternative paths.

```
CSR3#show bgp ipv6 unicast 2001:28:14:170::/64
BGP routing table entry for 2001:28:14:170::/64, version 333
Paths: (2 available, best #2, table default)
  Advertised to update-groups:
    5          6          7
Refresh Epoch 1
6
  ::FFFF:5.5.5.5 (metric 20) from 9.9.9.9 (9.9.9.9)
    Origin incomplete, metric 0, localpref 100, valid, internal
    Originator: 5.5.5.5, Cluster list: 9.9.9.9
    mpls labels in/out nolabel/5044
    rx pathid: 0, tx pathid: 0
Refresh Epoch 1
6
  ::FFFF:5.5.5.5 (metric 20) from 8.8.8.8 (8.8.8.8)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    Originator: 5.5.5.5, Cluster list: 8.8.8.8
    mpls labels in/out nolabel/5044
    rx pathid: 0, tx pathid: 0x0
```

We will configure CSR9 to select its backup path for advertisement using the minimum number of commands. CSR9 is instructed to select its backup path as the additional path, then advertise it to all IPv6 labeled-unicast neighbors. CSR3 still picks CSR5 as the exit point due to lower originator ID, but the backup path is XRv12. Ideally, we should configure CSR9 to install this as a backup path also, but that technically isn't a function of the shadow RR method, just an additional optimization. We verify that this backup path exists in the BGP table and the FIB. CSR3 will ECMP to the primary destination of CSR5 via CSR9 and XRv11 while the backup path routes via XRv11 exclusively.

```
! CSR9
router bgp 100
  template peer-policy IPV6_LUCAST
  advertise diverse-path backup
  address-family ipv6 unicast
  bgp additional-paths select backup

! CSR3
router bgp 100
  address-family ipv6
  bgp additional-paths install
```

```

CSR3#show bgp ipv6 unicast 2001:28:14:170::/64
BGP routing table entry for 2001:28:14:170::/64, version 347
Paths: (2 available, best #2, table default)
  Additional-path-install
  Advertised to update-groups:
    5          6          7
  Refresh Epoch 4
  6
    ::FFFF:12.12.12.12 (metric 20) from 9.9.9.9 (9.9.9.9)
      Origin incomplete, metric 0, localpref 100, valid, internal,
  backup/repair
      Originator: 12.12.12.12, Cluster list: 9.9.9.9
      mpls labels in/out nolabel/92026
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 3
  6
    ::FFFF:5.5.5.5 (metric 20) from 8.8.8.8 (8.8.8.8)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Originator: 5.5.5.5, Cluster list: 8.8.8.8
      mpls labels in/out nolabel/5044
      rx pathid: 0, tx pathid: 0x0

CSR3#show ipv6 cef 2001:28:14:170::/64 detail
2001:28:14:170::/64, epoch 0, flags [rib defined all labels]
  recursive via 5.5.5.5 label 5044
    nexthop 153.3.9.9 GigabitEthernet2.539 label 9037
    nexthop 153.3.11.11 GigabitEthernet2.531 label 91018
  recursive via 12.12.12.12 label 92026, repair
    nexthop 153.3.11.11 GigabitEthernet2.531 label 91010

```

Next, we will demonstrate the PE's each advertising their best-external routes as this does not require add-path negotiation. XRv13 will set the local preference on the prefix 2001:28:14:170::/64 to 200 so that it is the preferred exit point from AS 100. We demonstrate the power of RPL again by using our highly generic existing configuration. Both CSR5 and XRv12 consider this their best path, and because it is iBGP learned (and these PE's are not RRs), it cannot be advertised to the RRs.

```

! XRv13
prefix-set PS_COOL_ROUTE_IPV6
  2001:28:14:170::/64
end-set

router bgp 100
  neighbor 2001:10:6:13::6
  remote-as 6
  address-family ipv6 unicast
    route-policy RPL_SET_LPRF_TO_X_FOR_Y(200, PS_COOL_ROUTE_IPV6) in

RP/0/0/CPU0:XRv12#show bgp ipv6 unicast 2001:28:14:170::/64 | begin Paths

```

```

Paths: (3 available, best #1)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  6
    13.13.13.13 (metric 20) from 8.8.8.8 (13.13.13.13)
      Received Label 93009
      Origin incomplete, metric 0, localpref 200, valid, internal, best,
group-best
      Received Path ID 0, Local Path ID 1, version 22
      Originator: 13.13.13.13, Cluster list: 8.8.8.8
  Path #2: Received by speaker 0
  Not advertised to any peer
  6
    13.13.13.13 (metric 20) from 9.9.9.9 (13.13.13.13)
      Received Label 93009
      Origin incomplete, metric 0, localpref 200, valid, internal
      Received Path ID 0, Local Path ID 0, version 0
      Originator: 13.13.13.13, Cluster list: 9.9.9.9
  Path #3: Received by speaker 0
  Not advertised to any peer
  6
    2001:10:6:12::6 from 2001:10:6:12::6 (6.6.6.6)
      Origin incomplete, metric 0, localpref 100, valid, external
      Received Path ID 0, Local Path ID 0, version 0
      Origin-AS validity: not-found

CSR5#show bgp ipv6 unicast 2001:28:14:170::/64 | begin Paths
Paths: (3 available, best #2, table default)
  Advertised to update-groups:
    6
  Refresh Epoch 2
  6
    ::FFFF:13.13.13.13 (metric 20) from 9.9.9.9 (9.9.9.9)
      Origin incomplete, metric 0, localpref 200, valid, internal
      Originator: 13.13.13.13, Cluster list: 9.9.9.9
      mpls labels in/out 5044/93009
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  6
    ::FFFF:13.13.13.13 (metric 20) from 8.8.8.8 (8.8.8.8)
      Origin incomplete, metric 0, localpref 200, valid, internal, best
      Originator: 13.13.13.13, Cluster list: 8.8.8.8
      mpls labels in/out 5044/93009
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  6
    2001:10:5:6::6 (FE80::6) from 10.5.6.6 (6.6.6.6)
      Origin incomplete, metric 0, localpref 100, valid, external

```

```
mpls labels in/out 5044/nolabel
rx pathid: 0, tx pathid: 0
```

The result is that the RR's only have a single route to exit the AS, and this is all they distribute to their peers. The remote PEs, such as CSR3, are going to see the same copy of that single route. Our existing solution of using shadow RR in isolation won't solve the problem because the RR's only see one path in the first place.

```
CSR9#show bgp ipv6 unicast 2001:28:14:170::/64
BGP routing table entry for 2001:28:14:170::/64, version 31
BGP Bestpath: igpmetric-ignore
Paths: (1 available, best #1, table default)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  6, (Received from a RR-client)
    ::FFFF:13.13.13.13 (metric 10) from 13.13.13.13 (13.13.13.13)
      Origin incomplete, metric 0, localpref 200, valid, internal, best
      mpls labels in/out nolabel/93009
      rx pathid: 0, tx pathid: 0x0
```

```
CSR3#show bgp ipv6 unicast 2001:28:14:170::/64
BGP routing table entry for 2001:28:14:170::/64, version 355
Paths: (2 available, best #2, table default)
  Additional-path-install
  Advertised to update-groups:
    5          6          7
  Refresh Epoch 5
  6
    ::FFFF:13.13.13.13 (metric 20) from 9.9.9.9 (9.9.9.9)
      Origin incomplete, metric 0, localpref 200, valid, internal
      Originator: 13.13.13.13, Cluster list: 9.9.9.9
      mpls labels in/out nolabel/93009
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 4
  6
    ::FFFF:13.13.13.13 (metric 20) from 8.8.8.8 (8.8.8.8)
      Origin incomplete, metric 0, localpref 200, valid, internal, best
      Originator: 13.13.13.13, Cluster list: 8.8.8.8
      mpls labels in/out nolabel/93009
      rx pathid: 0, tx pathid: 0x0
```

As seen earlier, we can use best-external to solve this. CSR5, XRv12, and XRv13 are configured to advertise their best external routes to the RRs, giving them a better view of the network. Then, our existing shadow RR configuration ensures that a diverse path is advertised to the remote PEs. Note that these PE's must also install this path as a backup before they are able to advertise it to the RR. Unfortunately, this "selection" keyword on XR seems to only apply to the negotiated add-path

capability. XRv12 claims the path was selected as a backup but was not advertised to the RRs, probably because it expects the full negotiated capability to be exchanged.

```
! CSR5
router bgp 100
  template peer-policy IPV6_INTERNAL
    advertise best-external
  address-family ipv6 unicast
    bgp additional-paths select best-external
    bgp additional-paths install

! XRv12 (should be on XRv13 also to account for failovers also)
route-policy RPL_BACKUP_1
  set path-selection backup 1 advertise install
end-policy

router bgp 100
  address-family ipv6 unicast
    additional-paths selection route-policy RPL_BACKUP_1
```

RP/0/0/CPU0:XRv12#show bgp ipv6 labeled-unicast neighbors 9.9.9.9 advertised-routes

Network	Next Hop	From	AS Path
::6:6:6:6/128	12.12.12.12	2001:10:6:12::6	6?
2001:28:10:44::/64	12.12.12.12	2001:10:6:12::6	6?
2001:28:10:45::/64	12.12.12.12	2001:10:6:12::6	6?
2001:28:10:46::/64	12.12.12.12	2001:10:6:12::6	6?
2001:28:10:47::/64	12.12.12.12	2001:10:6:12::6	6?

RP/0/0/CPU0:XRv12#show bgp ipv6 unicast 2001:28:14:170::/64 | begin Paths

Paths: (3 available, best #1)

Not advertised to any peer

Path #1: Received by speaker 0

Not advertised to any peer

6

13.13.13.13 (metric 20) from 8.8.8.8 (13.13.13.13)

Received Label 93009

Origin incomplete, metric 0, localpref 200, valid, internal, best,

group-best

Received Path ID 0, Local Path ID 1, version 22

Originator: 13.13.13.13, Cluster list: 8.8.8.8

Path #2: Received by speaker 0

Not advertised to any peer

6

13.13.13.13 (metric 20) from 9.9.9.9 (13.13.13.13)

Received Label 93009

Origin incomplete, metric 0, localpref 200, valid, internal

Received Path ID 0, Local Path ID 0, version 0

Originator: 13.13.13.13, Cluster list: 9.9.9.9

Path #3: Received by speaker 0

Not advertised to any peer

6

2001:10:6:12::6 from 2001:10:6:12::6 (6.6.6.6)

Origin incomplete, metric 0, localpref 100, valid, external, backup,

add-path

Received Path ID 0, Local Path ID 2, version 30

Origin-AS validity: not-found

At a minimum, the RR's receive the best-external from CSR5, which can be advertised to the remote PEs using the diverse-path (shadow RR) method. Another very odd issue can be observed on CSR9. Notice that the IPv6 next-hop is reachable, but is inaccessible. This is because there is no MPLS label associated with the 6PE prefix. I don't have a great reason for this, but the debug on CSR9 reveals that the MPLS label for this backup path was not advertised with the prefix from CSR5. Notice that all the other routes have labels.

```
CSR9#show bgp ipv6 unicast 2001:28:14:170::/64
BGP routing table entry for 2001:28:14:170::/64, version 70
BGP Bestpath: igpmetric-ignore
Paths: (2 available, best #2, table default)
  Additional-path-install
  Advertised to update-groups:
    2
  Refresh Epoch 2
  6, (Received from a RR-client)
    ::FFFF:5.5.5.5 (inaccessible) from 5.5.5.5 (5.5.5.5)
      Origin incomplete, metric 0, localpref 100, valid, internal
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  6, (Received from a RR-client)
    ::FFFF:13.13.13.13 (metric 10) from 13.13.13.13 (13.13.13.13)
      Origin incomplete, metric 0, localpref 200, valid, internal, best
      mpls labels in/out nlabel/93009
      rx pathid: 0, tx pathid: 0x0
```

!CSR9

BGP(1): 5.5.5.5 rcvd ::6:6:6:6/128, label 5046

BGP(1): 5.5.5.5 rcvd 2001:28:14:170::/64

BGP(1): 5.5.5.5 rcvd 2001:28:10:44::/64, label 5047

BGP(1): 5.5.5.5 rcvd 2001:28:10:45::/64, label 5048

BGP(1): 5.5.5.5 rcvd 2001:28:10:46::/64, label 5049

BGP(1): 5.5.5.5 rcvd 2001:28:10:47::/64, label 5050

On CSR5, when debugging outbound updates, we see that the label was never sent. Digging deeper, we see that local label was not even allocated. This is true on both CSR5 and XRv12, and I am concluding that the older, non-negotiated add-path features were not designed to work with labeled-unicast. MPLS wasn't supposed to reinvent the wheel, but rely on existing IGP and BGP routing, and since the shadow

RR method is breaking rules without any active communication, MPLS may not understand. BGP may see no reason to allocate labels (the local labels are highlighted) for paths that are not best. So, using additional-path best-external with labeled-unicast AFI does not appear to be a valid design.

```
! CSR5
BGP(1): (base) 8.8.8.8 send UPDATE (format) ::6:6:6:6/128, next
::FFFF:5.5.5.5, label 5046, metric 0, path 6, label 5046
BGP(1): (base) 8.8.8.8 send UPDATE (format) 2001:28:14:170::/64, next
::FFFF:5.5.5.5, metric 0, path 6
```

```
CSR5#show bgp ipv6 unicast labels | begin 2001:28:14:170::/64
2001:28:14:170::/64
          ::FFFF:13.13.13.13
                        nolabel/93009
          ::FFFF:13.13.13.13
                        nolabel/93009
          2001:10:5:6::6   nolabel/nolabel
```

```
RP/0/0/CPU0:XRv12#show bgp ipv6 labeled-unicast labels | begin
2001:28:14:170::/64
*>i2001:28:14:170::/64
          13.13.13.13      93009      nolabel
* i       13.13.13.13      93009      nolabel
```

Let's try to salvage this design by removing the best-external configuration from CSR5 and enabling the BGP add-path negotiation. We will enable send/receive capabilities on the PEs and RRs so we can do all our testing bidirectionally as needed. For completeness we will apply these changes on all relevant devices. This also means we can bring XRv11 back into the network. The snippets are shown below, along with the basic neighbor negotiation verifications. I am only showing outputs from the RR's to save time; if they see the proper negotiations sent/received them it's a fair assumption to assume the PE's are configured properly for add-path. It can be time consuming to do these basic verifications but trying to troubleshoot why additional-paths aren't being advertised is much more difficult.

```
! XRv11, XRv12, XRv13, XRv14
router bgp 100
  address-family ipv6 unicast
    additional-paths send
    additional-paths receive

! CSR3, CSR5, CSR8, CSR9
router bgp 100
  address-family ipv6 unicast
    bgp additional-paths send receive
```

```
CSR8#show bgp ipv6 unicast neighbors 3.3.3.3 | include IPv6_U|Additional
For address family: IPv6 Unicast
Additional Paths send capability: advertised and received
```

Additional Paths receive capability: advertised and received

```
CSR8#show bgp ipv6 unicast neighbors 5.5.5.5 | include IPv6_U|Additional
For address family: IPv6 Unicast
Additional Paths send capability: advertised and received
Additional Paths receive capability: advertised and received
```

```
CSR8#show bgp ipv6 unicast neighbors 14.14.14.14 | include IPv6_U|Additional
For address family: IPv6 Unicast
Additional Paths send capability: advertised and received
Additional Paths receive capability: advertised and received
```

```
CSR8#show bgp ipv6 unicast neighbors 13.13.13.13 | include IPv6_U|Additional
For address family: IPv6 Unicast
Additional Paths send capability: advertised and received
Additional Paths receive capability: advertised and received
```

```
CSR8#show bgp ipv6 unicast neighbors 12.12.12.12 | include IPv6_U|Additional
For address family: IPv6 Unicast
Additional Paths send capability: advertised and received
Additional Paths receive capability: advertised and received
```

```
CSR9#show bgp ipv6 unicast neighbors 3.3.3.3 | include IPv6_U|Additional
For address family: IPv6 Unicast
Additional Paths send capability: advertised and received
Additional Paths receive capability: advertised and received
```

```
CSR9#show bgp ipv6 unicast neighbors 5.5.5.5 | include IPv6_U|Additional
For address family: IPv6 Unicast
Additional Paths send capability: advertised and received
Additional Paths receive capability: advertised and received
```

```
CSR9#show bgp ipv6 unicast neighbors 14.14.14.14 | include IPv6_U|Additional
For address family: IPv6 Unicast
Additional Paths send capability: advertised and received
Additional Paths receive capability: advertised and received
```

```
CSR9#show bgp ipv6 unicast neighbors 13.13.13.13 | include IPv6_U|Additional
For address family: IPv6 Unicast
Additional Paths send capability: advertised and received
Additional Paths receive capability: advertised and received
```

```
CSR9#show bgp ipv6 unicast neighbors 12.12.12.12 | include IPv6_U|Additional
For address family: IPv6 Unicast
Additional Paths send capability: advertised and received
Additional Paths receive capability: advertised and received
```

```
RP/0/0/CPU0:XRv11#show bgp ipv6 labeled-unicast neighbors 3.3.3.3 | utility
egrep 'IPv6 Lab|Additional'
```

```
For Address Family: IPv6 Labeled-unicast
  Additional-paths Send: advertised and received
  Additional-paths Receive: advertised and received
  Additional-paths operation: Send and Receive
```

```
RP/0/0/CPU0:XRv11#show bgp ipv6 labeled-unicast neighbors 5.5.5.5 | utility
egrep 'IPv6 Lab|Additional'
```

```
For Address Family: IPv6 Labeled-unicast
  Additional-paths Send: advertised and received
  Additional-paths Receive: advertised and received
  Additional-paths operation: Send and Receive
  Additional-paths operation: None
```

```
RP/0/0/CPU0:XRv11#show bgp ipv6 labeled-unicast neighbors 14.14.14.14 |
utility egrep 'IPv6 Lab|Additional'
```

```
For Address Family: IPv6 Labeled-unicast
  Additional-paths Send: advertised and received
  Additional-paths Receive: advertised and received
  Additional-paths operation: Send and Receive
  Additional-paths operation: None
```

```
RP/0/0/CPU0:XRv11#show bgp ipv6 labeled-unicast neighbors 13.13.13.13 |
utility egrep 'IPv6 Lab|Additional'
```

```
For Address Family: IPv6 Labeled-unicast
  Additional-paths Send: advertised and received
  Additional-paths Receive: advertised and received
  Additional-paths operation: Send and Receive
  Additional-paths operation: None
```

```
RP/0/0/CPU0:XRv11#show bgp ipv6 labeled-unicast neighbors 12.12.12.12 |
utility egrep 'IPv6 Lab|Additional'
```

```
For Address Family: IPv6 Labeled-unicast
  Additional-paths Send: advertised and received
  Additional-paths Receive: advertised and received
  Additional-paths operation: Send and Receive
  Additional-paths operation: None
```

Before checking any specific BGP routing information, let's see if CSR5 allocates a label for this backup path now. It does, using label value 5007, which was BGP 6PE in action.

```
! CSR5
router bgp 100
  template peer-policy IPV6_INTERNAL
    advertise additional-paths best 2
  address-family ipv6 unicast
    bgp additional-paths select best 2
```

```
CSR5#show bgp ipv6 unicast labels | begin 2001:28:14:170::/64
2001:28:14:170::/64
```

```

::FFFF:13.13.13.13
                    5007/93009
::FFFF:13.13.13.13
                    5007/93009
::FFFF:13.13.13.13
                    5007/93009
2001:10:5:6::6     5007/nolabel

! XRv12
router bgp 100
  address-family ipv6 unicast
    additional-paths selection route-policy RPL_BACKUP_ALL

RP/0/0/CPU0:XRv12#show bgp ipv6 labeled-unicast labels | begin
2001:28:14:170::/64
*>i2001:28:14:170::/64
      13.13.13.13      93009      92002
* i      13.13.13.13      93009      92002
* i      5.5.5.5        5007      92002
* i      13.13.13.13      93009      92002
*      2001:10:6:12::6 nolabel      92002

```

CSR9 has the backup routes from CSR5 and XRv12 now. Technically, the existing selection and advertisement of the diverse-path could work, but with add-path negotiation enabled, it makes more sense to use that. The parser also tells you that the non-negotiated add-paths features are silently ignored when negotiated add-path is configured. I will configure CSR9 to select its best 3 paths for backup, and advertise 2 towards remote PEs by default. XRv14 will be an exception and receive all 3 paths. We can see the "best2" path from CSR5 and XRv12 on the RR (CSR9). We see that CSR3 has 3 copies of the best-path, and selects the CSR5 path as the backup. XRv14 also has 3 copies of the best path plus backups via CSR5 and XRv12 (on XRv14 sees the path via XRv12, though).

```

! CSR9
router bgp 100
  template peer-policy IPV6_LUCAST
    no advertise diverse-path backup
    advertise additional-paths best 2
  address-family ipv6 unicast
    no bgp additional-paths select backup
    bgp additional-paths select best 3
    neighbor 14.14.14.14 advertise additional-paths best 3

CSR9#show bgp ipv6 unicast 2001:28:14:170::/64
BGP routing table entry for 2001:28:14:170::/64, version 32
Paths: (3 available, best #2, table default)
  Additional-path-install
  Path advertised to update-groups:
    4
  Refresh Epoch 1

```

```

6, (Received from a RR-client)
::FFFF:12.12.12.12 (metric 30) from 12.12.12.12 (12.12.12.12)
  Origin incomplete, metric 0, localpref 100, valid, internal, best3
  mpls labels in/out nolabel/92002
  rx pathid: 0x2, tx pathid: 0x2
Path advertised to update-groups:
  3          4
Refresh Epoch 1
6, (Received from a RR-client)
::FFFF:13.13.13.13 (metric 10) from 13.13.13.13 (13.13.13.13)
  Origin incomplete, metric 0, localpref 200, valid, internal, best
  mpls labels in/out nolabel/93009
  rx pathid: 0x1, tx pathid: 0x0
Path advertised to update-groups:
  3          4
Refresh Epoch 2
6, (Received from a RR-client)
::FFFF:5.5.5.5 (metric 10) from 5.5.5.5 (5.5.5.5)
  Origin incomplete, metric 0, localpref 100, valid, internal,
backup/repair, best2
  mpls labels in/out nolabel/5007
  rx pathid: 0x1, tx pathid: 0x1

CSR3#show bgp ipv6 unicast 2001:28:14:170::/64
BGP routing table entry for 2001:28:14:170::/64, version 694
Paths: (4 available, best #4, table default)
  Additional-path-install
  Advertised to update-groups:
    5          6          7
Refresh Epoch 1
6
  ::FFFF:5.5.5.5 (metric 20) from 9.9.9.9 (9.9.9.9)
  Origin incomplete, metric 0, localpref 100, valid, internal,
backup/repair
  Originator: 5.5.5.5, Cluster list: 9.9.9.9
  mpls labels in/out nolabel/5007
  rx pathid: 0x1, tx pathid: 0
Refresh Epoch 1
6
  ::FFFF:13.13.13.13 (metric 20) from 9.9.9.9 (9.9.9.9)
  Origin incomplete, metric 0, localpref 200, valid, internal
  Originator: 13.13.13.13, Cluster list: 9.9.9.9
  mpls labels in/out nolabel/93009
  rx pathid: 0x0, tx pathid: 0
Refresh Epoch 1
6
  ::FFFF:13.13.13.13 (metric 20) from 11.11.11.11 (11.11.11.11)
  Origin incomplete, metric 0, localpref 200, valid, internal
  Originator: 13.13.13.13, Cluster list: 11.11.11.11

```

```

mpls labels in/out nlabel/93009
rx pathid: 0x1, tx pathid: 0
Refresh Epoch 2
6
::FFFF:13.13.13.13 (metric 20) from 8.8.8.8 (8.8.8.8)
Origin incomplete, metric 0, localpref 200, valid, internal, best
Originator: 13.13.13.13, Cluster list: 8.8.8.8
mpls labels in/out nlabel/93009
rx pathid: 0x0, tx pathid: 0x0

```

```
RP/0/0/CPU0:XRv14#show bgp ipv6 unicast 2001:28:14:170::/64 brief
```

```

Status codes: s suppressed, d damped, h history, * valid, > best
                i - internal, r RIB-failure, S stale, N Nexthop-discard
Origin codes: i - IGP, e - EGP, ? - incomplete

```

	Network	Next Hop	Metric	LocPrf	Weight	Path
*>i	2001:28:14:170::/64					
		13.13.13.13	0	200	0 6 ?	
* i		13.13.13.13	0	200	0 6 ?	
* i		5.5.5.5	0	100	0 6 ?	
* i		12.12.12.12	0	100	0 6 ?	
* i		13.13.13.13	0	200	0 6 ?	

Next, we will examine VPNv4 and VPNv6 tests briefly. This will be short because the CSR does not support the negotiated capabilities, but just the diverse-path, best-external, and backup install mechanisms. CSR4 and CSR10 are the VPNv4/v6 customers. An interesting design note with VPN service is that we can achieve additional-paths simply by having unique RD's. For example, CSR5 uses RD 100:5, while XRv13 and XRv12 use RD 100:13. CSR9 doesn't have any VRFs locally configured, but by breaking the routes apart by RD, we see one route in table 100:5 and two in table 100:13. The RR runs best-path on these routes within an RD, never across, so there are two best-paths advertised to remote PEs. One is 100:13:10.10.10.10/32 via XRv13 and one is 100:5:10.10.10.10/32 via CSR5. We could have had three different paths if XRv12 had a unique RD, such as 100:12, but I wanted to combine it with XRv13 to demonstrate the point.

```

CSR9#show bgp vpnv4 unicast rd 100:13 10.10.10.10/32
BGP routing table entry for 100:13:10.10.10.10/32, version 39
Paths: (2 available, best #2, no table)
Advertised to update-groups:
1
Refresh Epoch 1
410, (Received from a RR-client)
12.12.12.12 (metric 30) (via default) from 12.12.12.12 (12.12.12.12)
Origin incomplete, metric 0, localpref 100, valid, internal
Extended Community: RT:100:410
mpls labels in/out nlabel/92008
rx pathid: 0, tx pathid: 0

```



```

Refresh Epoch 1
410, (Received from a RR-client)
13.13.13.13 (metric 10) (via default) from 13.13.13.13 (13.13.13.13)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  Extended Community: RT:100:410
  mpls labels in/out nolabel/93011
  rx pathid: 0, tx pathid: 0x0

```

```

CSR9#show bgp vpnv4 unicast rd 100:5 10.10.10.10/32
BGP routing table entry for 100:5:10.10.10.10/32, version 38
Paths: (1 available, best #1, no table)
  Advertised to update-groups:
    1

```

```

Refresh Epoch 2
410, (Received from a RR-client)
5.5.5.5 (metric 10) (via default) from 5.5.5.5 (5.5.5.5)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  Extended Community: RT:100:410
  mpls labels in/out nolabel/5045
  rx pathid: 0, tx pathid: 0x0

```

We can verify that both of these routes were advertised to CSR7.

```

CSR9#show bgp vpnv4 unicast rd 100:13 neighbors 7.7.7.7 advertised-routes |
begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 100:13
*>i 10.10.10.10/32  13.13.13.13          0    100      0 410 ?

```

```

CSR9#show bgp vpnv4 unicast rd 100:5 neighbors 7.7.7.7 advertised-routes |
begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 100:5
*>i 10.10.10.10/32  5.5.5.5          0    100      0 410 ?

```

If we look at a different RR in the network, such as CSR8, we see that it select a different best-path in RD 100:13. Specifically, it picks XRv12 because its IGP metric is lower.

```

CSR8#show bgp vpnv4 unicast rd 100:5 neighbors 7.7.7.7 advertised-routes |
begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 100:5
*>i 10.10.10.10/32  5.5.5.5          0    100      0 410 ?

```

```

Total number of prefixes 1
CSR8#show bgp vpnv4 unicast rd 100:13 neighbors 7.7.7.7 advertised-routes |
begin Network
      Network      Next Hop      Metric LocPrf Weight Path

```

```
Route Distinguisher: 100:13
*>i 10.10.10.10/32 12.12.12.12 0 100 0 410 ?
```

CSR7 has received both RD 100:13 routes with next-hops of XRv12 and XRv13. Best-path runs on these routes first, before import into the VRF, so that only one VPN route is imported per RD. Because both XRv11 and CSR8 selected XRv12 as the best path, CSR3 has a copy of each route. Ultimately, the route through CSR8 wins due to lower BGP peer ID (the last tie breaker) and progresses to the "next round".

```
CSR7#show bgp vpnv4 unicast rd 100:13 10.10.10.10/32
BGP routing table entry for 100:13:10.10.10.10/32, version 151
Paths: (3 available, best #3, no table)
  Not advertised to any peer
  Refresh Epoch 3
  410
    13.13.13.13 (metric 20) (via default) from 9.9.9.9 (9.9.9.9)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Extended Community: RT:100:410
      Originator: 13.13.13.13, Cluster list: 9.9.9.9
      mpls labels in/out nolabel/93011
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  410
    12.12.12.12 (metric 20) (via default) from 11.11.11.11 (11.11.11.11)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Extended Community: RT:100:410
      Originator: 12.12.12.12, Cluster list: 11.11.11.11
      mpls labels in/out nolabel/92008
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  410
    12.12.12.12 (metric 20) (via default) from 8.8.8.8 (8.8.8.8)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:100:410
      Originator: 12.12.12.12, Cluster list: 8.8.8.8
      mpls labels in/out nolabel/92008
      rx pathid: 0, tx pathid: 0x0
```

Best-path within RD 100:5 is much less interesting because all three RR's selected the same best path (the only path) which is via CSR5. The path selected was based on the lowest neighbor ID (the last tie breaker), indicating the routes are equivalent in every way. This best path advances to the "next round".

```
CSR7#show bgp vpnv4 unicast rd 100:5 10.10.10.10/32
BGP routing table entry for 100:5:10.10.10.10/32, version 128
Paths: (3 available, best #3, no table)
  Not advertised to any peer
  Refresh Epoch 3
  410
```

```

5.5.5.5 (metric 20) (via default) from 9.9.9.9 (9.9.9.9)
  Origin incomplete, metric 0, localpref 100, valid, internal
  Extended Community: RT:100:410
  Originator: 5.5.5.5, Cluster list: 9.9.9.9
  mpls labels in/out nolabel/5045
  rx pathid: 0, tx pathid: 0
Refresh Epoch 1
410
5.5.5.5 (metric 20) (via default) from 11.11.11.11 (11.11.11.11)
  Origin incomplete, metric 0, localpref 100, valid, internal
  Extended Community: RT:100:410
  Originator: 5.5.5.5, Cluster list: 11.11.11.11
  mpls labels in/out nolabel/5045
  rx pathid: 0, tx pathid: 0
Refresh Epoch 1
410
5.5.5.5 (metric 20) (via default) from 8.8.8.8 (8.8.8.8)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  Extended Community: RT:100:410
  Originator: 5.5.5.5, Cluster list: 8.8.8.8
  mpls labels in/out nolabel/5045
  rx pathid: 0, tx pathid: 0x0

```

CSR7 imports these "best within the RD" VPN routes into the VRF for further processing, which allows it to directly compare all three routes since they are competing for a spot in the VRF RIB; they already exist in the VRF BGP table though. CSR5 wins due to lower originator ID. Enabling additional-path install and backup selection gives us fast-reroute without having to do anything fancy with the route-reflectors. This is a good design case for using different RD's at each PE, even for a single customer VRF. It is simple and does not require much additional complexity in the SP network.

```

CSR7#show bgp vpnv4 unicast vrf B 10.10.10.10/32
BGP routing table entry for 100:7:10.10.10.10/32, version 154
Paths: (2 available, best #2, table B)
  Additional-path-install
  Advertised to update-groups:
    8
Refresh Epoch 1
410, imported path from 100:13:10.10.10.10/32 (global)
  12.12.12.12 (metric 20) (via default) from 8.8.8.8 (8.8.8.8)
  Origin incomplete, metric 0, localpref 100, valid, internal,
  backup/repair
  Extended Community: RT:100:410
  Originator: 12.12.12.12, Cluster list: 8.8.8.8 , recursive-via-host
  mpls labels in/out nolabel/92008
  rx pathid: 0, tx pathid: 0
Refresh Epoch 1
410, imported path from 100:5:10.10.10.10/32 (global)
  5.5.5.5 (metric 20) (via default) from 8.8.8.8 (8.8.8.8)

```

```
Origin incomplete, metric 0, localpref 100, valid, internal, best
Extended Community: RT:100:410
Originator: 5.5.5.5, Cluster list: 8.8.8.8 , recursive-via-host
mpls labels in/out nolabel/5045
rx pathid: 0, tx pathid: 0x0
```

We will quickly demonstrate the shadow RR feature. This involves using the same RD on CSR5, XRv12, and XRv13, which forces the RR to only advertise one path. We change CSR5's RD to 100:13 to accomplish this. Beware that deleting the RD from the VRF automatically removes VRF-related BGP configuration and RT import/export policies, so backup the configuration before attempting the change. We already know that the route-reflectors are going to get three copies of the same route, pick a best path, then advertise that single path to CSR7. This creates the inherent problem add-path was meant to solve.

```
CSR9#show bgp vpnv4 unicast rd 100:13 10.10.10.10/32
BGP routing table entry for 100:13:10.10.10.10/32, version 43
Paths: (3 available, best #1, no table)
  Advertised to update-groups:
    1
  Refresh Epoch 2
  410, (Received from a RR-client)
    5.5.5.5 (metric 10) (via default) from 5.5.5.5 (5.5.5.5)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:100:410
      mpls labels in/out nolabel/5003
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  410, (Received from a RR-client)
    12.12.12.12 (metric 30) (via default) from 12.12.12.12 (12.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Extended Community: RT:100:410
      mpls labels in/out nolabel/92008
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  410, (Received from a RR-client)
    13.13.13.13 (metric 10) (via default) from 13.13.13.13 (13.13.13.13)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Extended Community: RT:100:410
      mpls labels in/out nolabel/93011
      rx pathid: 0, tx pathid: 0
```

As usual, we ignore the IGP metric in the RR for VPNv4/v6 so that RR's have the exact same topology regardless of IGP changes. Then, we configure CSR to be the shadow RR so it can advertise a diverse, second best path. CSR8, as usual, advertises its single best path, while CSR9 advertises a backup, diverse path (which is not marked as best).

! CSR8

```
router bgp 100
  address-family vpnv4
    bgp bestpath igp-metric ignore
```

! CSR9

```
router bgp 100
  template peer-policy VPNV4
    advertise diverse-path backup
  address-family vpnv4
    bgp bestpath igp-metric ignore
    bgp additional-paths select backup
```

```
CSR8#show bgp vpnv4 unicast rd 100:13 neighbors 7.7.7.7 advertised-routes |
begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 100:13					
*>i 10.10.10.10/32	5.5.5.5	0	100	0	410 ?

```
CSR9#show bgp vpnv4 unicast rd 100:13 neighbors 7.7.7.7 advertised-routes |
begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 100:13					
*bia 10.10.10.10/32	12.12.12.12	0	100	0	410 ?

As expected, CSR7 runs best-path within the RD and identifies the best and backup paths. The issue is that only one of these paths is actually imported into the VRF table, which means FRR is not achieved. Even if I configured add-path install under the VRF, it did not work. I have seen online blogs and my colleagues configured it successfully, but I have personally never seen it work, and so I do not document it here.

```
CSR7#show bgp vpnv4 unicast rd 100:13 10.10.10.10/32
```

```
BGP routing table entry for 100:13:10.10.10.10/32, version 170
```

```
Paths: (3 available, best #3, no table)
```

```
  Additional-path-install
```

```
  Not advertised to any peer
```

```
  Refresh Epoch 4
```

```
  410
```

```
    12.12.12.12 (metric 20) (via default) from 9.9.9.9 (9.9.9.9)
```

```
      Origin incomplete, metric 0, localpref 100, valid, internal,
```

```
  backup/repair
```

```
    Extended Community: RT:100:410
```

```
    Originator: 12.12.12.12, Cluster list: 9.9.9.9 , recursive-via-host
```

```
    mpls labels in/out nlabel/92008
```

```
    rx pathid: 0, tx pathid: 0
```

```
  Refresh Epoch 1
```

```
  410
```

```
    5.5.5.5 (metric 20) (via default) from 11.11.11.11 (11.11.11.11)
```

```
      Origin incomplete, metric 0, localpref 100, valid, internal
```

```

Extended Community: RT:100:410
Originator: 5.5.5.5, Cluster list: 11.11.11.11 , recursive-via-host
mpls labels in/out nolabel/5003
rx pathid: 0, tx pathid: 0
Refresh Epoch 1
410
5.5.5.5 (metric 20) (via default) from 8.8.8.8 (8.8.8.8)
Origin incomplete, metric 0, localpref 100, valid, internal, best
Extended Community: RT:100:410
Originator: 5.5.5.5, Cluster list: 8.8.8.8 , recursive-via-host
mpls labels in/out nolabel/5003
rx pathid: 0, tx pathid: 0x0

```

Only one route is imported into the VRF table, which means the FIB will only have one entry. While the VRF import process is probably very fast given the backup path, the FIB not being pre-programmed is a fundamental principle of FRR. The multiple RD method is simpler, more effective, and seems to be recommended by Cisco.

```

CSR7#show bgp vpnv4 unicast vrf B 10.10.10.10/32
BGP routing table entry for 100:7:10.10.10.10/32, version 171
Paths: (1 available, best #1, table B)
Additional-path-install
Advertised to update-groups:
8
Refresh Epoch 1
410, imported path from 100:13:10.10.10.10/32 (global)
5.5.5.5 (metric 20) (via default) from 8.8.8.8 (8.8.8.8)
Origin incomplete, metric 0, localpref 100, valid, internal, best
Extended Community: RT:100:410
Originator: 5.5.5.5, Cluster list: 8.8.8.8 , recursive-via-host
mpls labels in/out nolabel/5003
rx pathid: 0, tx pathid: 0x0

CSR7#show ip cef vrf B 10.10.10.10 detail
10.10.10.10/32, epoch 0, flags [rib defined all labels]
recursive via 5.5.5.5 label 5003
nexthop 153.7.11.11 GigabitEthernet2.571 label 91018

```

Given the density of XRv, I will attempt to use the negotiated capability between XRv11, XRv12, and XRv14 to demonstrate the equivalent of "best-external". XRv3 sets the local-preference of its exported VPN routes to 200 and advertises this to its iBGP peers. It is now the preferred exit point from AS 100 for VPN customer traffic. XRv12 selects this as its best path, giving the RR's no indicating of an alternative. The eBGP path is considered an "import suspect" as it was not a candidate for import into BGP (export from VRF).

```

! XRv13
route-policy RPL_EXPORT_VPNV4($LPREF)

```

```

set extcommunity rt (100:410)
set local-preference $LPREF
end-policy

```

```

vrf B
address-family ipv4 unicast
export route-policy RPL_EXPORT_VPNV4(200)
no export route-target

```

```

RP/0/0/CPU0:XRv12#show bgp vpnv4 unicast rd 100:13 10.10.10.10/32
BGP routing table entry for 10.10.10.10/32, Route Distinguisher: 100:13
Versions:

```

```

Process          bRIB/RIB   SendTblVer
Speaker          250       250

```

```

Local Label: 92008

```

```

Paths: (3 available, best #1)

```

```

Not advertised to any peer

```

```

Path #1: Received by speaker 0

```

```

Not advertised to any peer

```

```

410

```

```

13.13.13.13 (metric 20) from 8.8.8.8 (13.13.13.13)

```

```

Received Label 93011

```

```

Origin incomplete, metric 0, localpref 200, valid, internal, best,

```

```

group-best, import-candidate, imported, import suspect

```

```

Received Path ID 0, Local Path ID 1, version 250

```

```

Extended community: RT:100:410

```

```

Originator: 13.13.13.13, Cluster list: 8.8.8.8

```

```

Source VRF: B, Source Route Distinguisher: 100:13

```

```

Path #2: Received by speaker 0

```

```

Not advertised to any peer

```

```

410

```

```

13.13.13.13 (metric 20) from 9.9.9.9 (13.13.13.13)

```

```

Received Label 93011

```

```

Origin incomplete, metric 0, localpref 200, valid, internal, import-
candidate, imported, import suspect

```

```

Received Path ID 0, Local Path ID 0, version 0

```

```

Extended community: RT:100:410

```

```

Originator: 13.13.13.13, Cluster list: 9.9.9.9

```

```

Source VRF: B, Source Route Distinguisher: 100:13

```

```

Path #3: Received by speaker 0

```

```

Not advertised to any peer

```

```

410

```

```

10.10.12.10 from 10.10.12.10 (10.10.10.10)

```

```

Origin incomplete, metric 0, localpref 100, valid, external, backup,
add-path, import suspect

```

```

Received Path ID 0, Local Path ID 2, version 233

```

```

Extended community: RT:100:410

```

To fix it, we need quite a bit of configuration. We need to enable the dynamic add-path capability on XRv11, XRv12, and XRv14. Each router needs to select its backup paths using RPL, and they need to be advertised to the RRs (XRv12 to XRv11), then advertised and installed on the remote PE (XRv14).

```
! XRv11
route-policy RPL_VPN_ADD_PATH
  set path-selection all advertise
end-policy

router bgp 100
  address-family vpnv4 unicast
    additional-paths send
    additional-paths receive
    additional-paths selection route-policy RPL_VPN_ADD_PATH

! XRv12
router bgp 100
  address-family vpnv4 unicast
    additional-paths receive
    additional-paths send
    additional-paths selection route-policy RPL_BACKUP_ALL

! XRv14
route-policy RPL_ADD_BACKUP
  set path-selection backup 1 install
end-policy

router bgp 100
  address-family vpnv4 unicast
    additional-paths receive
    additional-paths send
    additional-paths selection route-policy RPL_ADD_BACKUP
```

XRv12 now considers this a valid add-path and advertises it to XRv11. The RR sees the additional path from XRv12 and can advertise it to XRv14.

```
RP/0/0/CPU0:XRv12#show bgp vpnv4 unicast rd 100:13 10.10.10.10/32 | begin
Paths
Paths: (4 available, best #1)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  410
    13.13.13.13 (metric 20) from 8.8.8.8 (13.13.13.13)
      Received Label 93011
      Origin incomplete, metric 0, localpref 200, valid, internal, best,
group-best, import-candidate, imported
      Received Path ID 0, Local Path ID 1, version 291
```



```

    Extended community: RT:100:410
    Originator: 13.13.13.13, Cluster list: 8.8.8.8
    Source VRF: B, Source Route Distinguisher: 100:13
Path #2: Received by speaker 0
Not advertised to any peer
410
    13.13.13.13 (metric 20) from 9.9.9.9 (13.13.13.13)
    Received Label 93011
    Origin incomplete, metric 0, localpref 200, valid, internal, import-
candidate, imported
    Received Path ID 0, Local Path ID 0, version 0
    Extended community: RT:100:410
    Originator: 13.13.13.13, Cluster list: 9.9.9.9
    Source VRF: B, Source Route Distinguisher: 100:13
Path #3: Received by speaker 0
Advertised to peers (in unique update groups):
    11.11.11.11
410
    10.10.12.10 from 10.10.12.10 (10.10.10.10)
    Origin incomplete, metric 0, localpref 100, valid, external, backup,
add-path
    Received Path ID 0, Local Path ID 2, version 280
    Extended community: RT:100:410
Path #4: Received by speaker 0
Not advertised to any peer
410
    13.13.13.13 (metric 20) from 11.11.11.11 (13.13.13.13)
    Received Label 93011
    Origin incomplete, metric 0, localpref 200, valid, internal, import-
candidate, imported
    Received Path ID 1, Local Path ID 0, version 0
    Extended community: RT:100:410
    Originator: 13.13.13.13, Cluster list: 11.11.11.11
    Source VRF: B, Source Route Distinguisher: 100:13

RP/0/0/CPU0:XRv11#show bgp vpnv4 unicast rd 100:13 10.10.10.10/32 | begin
Paths
Paths: (2 available, best #2)
    Advertised to update-groups (with more than one peer):
        0.2
    Advertised to peers (in unique update groups):
        12.12.12.12
Path #1: Received by speaker 0
Not advertised to any peer
410, (Received from a RR-client)
    12.12.12.12 (metric 10) from 12.12.12.12 (12.12.12.12)
    Received Label 92008
    Origin incomplete, metric 0, localpref 100, valid, internal, not-in-vrf
    Received Path ID 2, Local Path ID 0, version 0

```

```

    Extended community: RT:100:410
    Path #2: Received by speaker 0
    Advertised to update-groups (with more than one peer):
    0.2
    Advertised to peers (in unique update groups):
    12.12.12.12
    410, (Received from a RR-client)
    13.13.13.13 (metric 30) from 13.13.13.13 (13.13.13.13)
    Received Label 93011
    Origin incomplete, metric 0, localpref 200, valid, internal, best,
group-best, import-candidate, not-in-vrf
    Received Path ID 0, Local Path ID 1, version 45
    Extended community: RT:100:410

RP/0/0/CPU0:XRv11#show bgp vpnv4 unicast neighbors 14.14.14.14 advertised-
routes
Network                Next Hop                From                AS Path
[snip]
Route Distinguisher: 100:13
10.10.10.10/32          13.13.13.13            13.13.13.13        410?
                        12.12.12.12            12.12.12.12        410?
[snip]

```

Finally, the remote PE gets both routes and marks the less preferred route (lower local-preference) as a backup path. We still have the same issue as earlier, which is that only a single route gets installed in the VRF RIB and thus in the FIB. This is why unique RD is still the best solution in my eyes. The reason the FIB shows two paths is due to IGP ECMP towards 13.13.13.13; don't be fooled.

```

RP/0/0/CPU0:XRv14#show bgp vpnv4 unicast rd 100:13 10.10.10.10/32 | begin
Paths
Paths: (4 available, best #1)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  410
  13.13.13.13 (metric 20) from 8.8.8.8 (13.13.13.13)
  Received Label 93011
  Origin incomplete, metric 0, localpref 200, valid, internal, best,
group-best, import-candidate, not-in-vrf
  Received Path ID 0, Local Path ID 1, version 169
  Extended community: RT:100:410
  Originator: 13.13.13.13, Cluster list: 8.8.8.8
  Path #2: Received by speaker 0
  Not advertised to any peer
  410
  13.13.13.13 (metric 20) from 9.9.9.9 (13.13.13.13)
  Received Label 93011
  Origin incomplete, metric 0, localpref 200, valid, internal, not-in-vrf

```

```

    Received Path ID 0, Local Path ID 0, version 0
    Extended community: RT:100:410
    Originator: 13.13.13.13, Cluster list: 9.9.9.9
Path #3: Received by speaker 0
Not advertised to any peer
410
    13.13.13.13 (metric 20) from 11.11.11.11 (13.13.13.13)
    Received Label 93011
    Origin incomplete, metric 0, localpref 200, valid, internal, not-in-vrf
    Received Path ID 1, Local Path ID 0, version 0
    Extended community: RT:100:410
    Originator: 13.13.13.13, Cluster list: 11.11.11.11
Path #4: Received by speaker 0
Not advertised to any peer
410
    12.12.12.12 (metric 20) from 11.11.11.11 (12.12.12.12)
    Received Label 92008
    Origin incomplete, metric 0, localpref 100, valid, internal, backup,
add-path, not-in-vrf
    Received Path ID 2, Local Path ID 2, version 177
    Extended community: RT:100:410
    Originator: 12.12.12.12, Cluster list: 11.11.11.11

RP/0/0/CPU0:XRv14#show bgp vpnv4 unicast vrf B 10.10.10.10/32 | beign Paths
Paths: (1 available, best #1)
    Not advertised to any peer
    Path #1: Received by speaker 0
    Not advertised to any peer
410
    13.13.13.13 (metric 20) from 8.8.8.8 (13.13.13.13)
    Received Label 93011
    Origin incomplete, metric 0, localpref 200, valid, internal, best,
group-best, import-candidate, imported
    Received Path ID 0, Local Path ID 1, version 174
    Extended community: RT:100:410
    Originator: 13.13.13.13, Cluster list: 8.8.8.8
    Source VRF: default, Source Route Distinguisher: 100:13

RP/0/0/CPU0:XRv14#show cef vrf B ipv4 10.10.10.10/32
10.10.10.10/32, version 162, internal 0x5000001 0x0 (ptr 0xa13d4774) [1], 0x0
(0x0), 0x208 (0xa152c3c0)
Prefix Len 32, traffic index 0, precedence n/a, priority 3
    via 13.13.13.13, 5 dependencies, recursive [flags 0x6000]
    path-idx 0 NHID 0x0 [0xa1594b74 0x0]
    recursion-via-/32
    next hop VRF - 'default', table - 0xe0000000
    next hop 13.13.13.13 via 94000/0/21
    next hop 153.8.14.8/32 Gi0/0/0/0.584 labels imposed {8009 93011}
    next hop 153.9.14.9/32 Gi0/0/0/0.594 labels imposed {9006 93011}

```

Last, we will examine VPNv6 with add-path. CSR5, XRv12, and XRv13 all still use the same RD, so we will test add-path first before testing diverse RDs. Looking quickly at CSR9, we see that the bestpath to ::10:10:10:10/128 is via CSR5 due to lowest metric, and then due to neighbor RID. CSR8 picks XRv12 for the same reason. Fortunately, we can achieve a little bit of redundancy based on NOT ignoring the IGP metric in this specific situation. The problem is that changes in the IGP topology may negatively influence our BGP topology with designs like this. Without even looking at XRv11, we know that it will select CSR5 over XRv12 due to lower RID, and rejects XRv13 due to higher IGP cost.

```
CSR9#show bgp vpnv6 unicast rd 100:13 ::10:10:10:10/128
BGP routing table entry for [100:13]::10:10:10:10/128, version 33
Paths: (3 available, best #2, no table)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  410, (Received from a RR-client)
    ::FFFF:12.12.12.12 (metric 30) (via default) from 12.12.12.12
(12.12.12.12)
  Origin incomplete, metric 0, localpref 100, valid, internal
  Extended Community: RT:100:410
  mpls labels in/out nolabel/92012
  rx pathid: 0, tx pathid: 0
  Refresh Epoch 2
  410, (Received from a RR-client)
    ::FFFF:5.5.5.5 (metric 10) (via default) from 5.5.5.5 (5.5.5.5)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  Extended Community: RT:100:410
  mpls labels in/out nolabel/5010
  rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  410, (Received from a RR-client)
    ::FFFF:13.13.13.13 (metric 10) (via default) from 13.13.13.13
(13.13.13.13)
  Origin incomplete, metric 0, localpref 100, valid, internal
  Extended Community: RT:100:410
  mpls labels in/out nolabel/93013
  rx pathid: 0, tx pathid: 0

CSR8#show bgp vpnv6 unicast rd 100:13 ::10:10:10:10/128
BGP routing table entry for [100:13]::10:10:10:10/128, version 46
Paths: (3 available, best #1, no table)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  410, (Received from a RR-client)
    ::FFFF:12.12.12.12 (metric 10) (via default) from 12.12.12.12
(12.12.12.12)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
```

```

Extended Community: RT:100:410
mpls labels in/out nolabel/92012
rx pathid: 0, tx pathid: 0x0
Refresh Epoch 1
410, (Received from a RR-client)
::FFFF:5.5.5.5 (metric 30) (via default) from 5.5.5.5 (5.5.5.5)
Origin incomplete, metric 0, localpref 100, valid, internal
Extended Community: RT:100:410
mpls labels in/out nolabel/5010
rx pathid: 0, tx pathid: 0
Refresh Epoch 1
410, (Received from a RR-client)
::FFFF:13.13.13.13 (metric 10) (via default) from 13.13.13.13
(13.13.13.13)
Origin incomplete, metric 0, localpref 100, valid, internal
Extended Community: RT:100:410
mpls labels in/out nolabel/93013
rx pathid: 0, tx pathid: 0

```

On CSR7, we will receive two copies of CSR5's route (via CSR9 and XRv11) and one copy of XRv12's route (via CSR8). Activating add-path install allows the route to XRv12 to be considered a backup, but not installed in the VRF FIB, because RD-level bestpath happens before VRF import from BGP. This is the same issue we saw before.

```

! CSR7
router bgp 100
address-family vpnv6
  bgp additional-paths install

CSR7#show bgp vpnv6 unicast rd 100:13 ::10:10:10:10/128
BGP routing table entry for [100:13]::10:10:10:10/128, version 209
Paths: (3 available, best #2, no table)
  Additional-path-install
  Not advertised to any peer
  Refresh Epoch 1
  410
    ::FFFF:5.5.5.5 (metric 20) (via default) from 11.11.11.11 (11.11.11.11)
    Origin incomplete, metric 0, localpref 100, valid, internal
    Extended Community: RT:100:410
    Originator: 5.5.5.5, Cluster list: 11.11.11.11 , recursive-via-host
    mpls labels in/out nolabel/5010
    rx pathid: 0, tx pathid: 0
  Refresh Epoch 3
  410
    ::FFFF:5.5.5.5 (metric 20) (via default) from 9.9.9.9 (9.9.9.9)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    Extended Community: RT:100:410
    Originator: 5.5.5.5, Cluster list: 9.9.9.9 , recursive-via-host

```

```

mpls labels in/out nolabel/5010
rx pathid: 0, tx pathid: 0x0
Refresh Epoch 1
410
::FFFF:12.12.12.12 (metric 20) (via default) from 8.8.8.8 (8.8.8.8)
Origin incomplete, metric 0, localpref 100, valid, internal,
backup/repair
Extended Community: RT:100:410
Originator: 12.12.12.12, Cluster list: 8.8.8.8 , recursive-via-host
mpls labels in/out nolabel/92012
rx pathid: 0, tx pathid: 0

CSR7#show bgp vpnv6 unicast vrf B ::10:10:10:10/128
BGP routing table entry for [100:7]::10:10:10:10/128, version 208
Paths: (1 available, best #1, table B)
Additional-path-install
Advertised to update-groups:
6
Refresh Epoch 3
410, imported path from [100:13]::10:10:10:10/128 (global)
::FFFF:5.5.5.5 (metric 20) (via default) from 9.9.9.9 (9.9.9.9)
Origin incomplete, metric 0, localpref 100, valid, internal, best
Extended Community: RT:100:410
Originator: 5.5.5.5, Cluster list: 9.9.9.9 , recursive-via-host
mpls labels in/out nolabel/5010
rx pathid: 0, tx pathid: 0x0

CSR7#show ipv6 cef vrf B ::10:10:10:10 detail
::10:10:10:10/128, epoch 0, flags [rib defined all labels]
recursive via 5.5.5.5 label 5010
nexthop 153.7.11.11 GigabitEthernet2.571 label 91018

```

We can achieve a similar, and somewhat better result, by ignoring the IGP metric on CSR8 and CSR9 (we will disable XRv11 for this since it doesn't support that command), and directing CSR9 to advertise diverse paths. This is the shadow RR feature we have tested many times. CSR8 and CSR9 both select CSR5 as the bestpath given the lowest RID. In terms of advertisements, CSR8 advertises the bestpath (classic RR behavior), while CSR9 advertises its backup path (shadow RR behavior). Notice that the route advertised by CSR9 is marked as an additional and backup path, but is not the bestpath.

```

! CSR8
router bgp 100
address-family vpnv6 unicast
bgp bestpath igp-metric ignore

! CSR9
router bgp 100
template peer-policy VPNV6
advertise diverse-path backup

```

```

address-family vpnv6 unicast
  bgp bestpath igp-metric ignore

CSR8#show bgp vpnv6 unicast rd 100:13 ::10:10:10:10/128
BGP routing table entry for [100:13]::10:10:10:10/128, version 48
BGP Bestpath: igpmetric-ignore
Paths: (3 available, best #2, no table)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  410, (Received from a RR-client)
    ::FFFF:12.12.12.12 (metric 10) (via default) from 12.12.12.12
(12.12.12.12)
  Origin incomplete, metric 0, localpref 100, valid, internal
  Extended Community: RT:100:410
  mpls labels in/out nolabel/92012
  rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  410, (Received from a RR-client)
    ::FFFF:5.5.5.5 (metric 30) (via default) from 5.5.5.5 (5.5.5.5)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  Extended Community: RT:100:410
  mpls labels in/out nolabel/5010
  rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  410, (Received from a RR-client)
    ::FFFF:13.13.13.13 (metric 10) (via default) from 13.13.13.13
(13.13.13.13)
  Origin incomplete, metric 0, localpref 100, valid, internal
  Extended Community: RT:100:410
  mpls labels in/out nolabel/93013
  rx pathid: 0, tx pathid: 0

CSR8#show bgp vpnv6 unicast rd 100:13 neighbors 7.7.7.7 advertised-routes |
begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 100:13
*>i ::10:10:10:10/128
      ::FFFF:5.5.5.5          0      100      0 410 ?

CSR9#show bgp vpnv6 unicast rd 100:13 ::10:10:10:10/128
BGP routing table entry for [100:13]::10:10:10:10/128, version 39
BGP Bestpath: igpmetric-ignore
Paths: (3 available, best #2, no table)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  410, (Received from a RR-client)

```

```

::FFFF:12.12.12.12 (metric 30) (via default) from 12.12.12.12
(12.12.12.12)
  Origin incomplete, metric 0, localpref 100, valid, internal,
  backup/repair
    Extended Community: RT:100:410
    mpls labels in/out nolabel/92012
    rx pathid: 0, tx pathid: 0
  Refresh Epoch 2
  410, (Received from a RR-client)
    ::FFFF:5.5.5.5 (metric 10) (via default) from 5.5.5.5 (5.5.5.5)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    Extended Community: RT:100:410
    mpls labels in/out nolabel/5010
    rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  410, (Received from a RR-client)
    ::FFFF:13.13.13.13 (metric 10) (via default) from 13.13.13.13
(13.13.13.13)
  Origin incomplete, metric 0, localpref 100, valid, internal
  Extended Community: RT:100:410
  mpls labels in/out nolabel/93013
  rx pathid: 0, tx pathid: 0

CSR9#show bgp vpnv6 unicast rd 100:13 neighbors 7.7.7.7 advertised-routes |
begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 100:13
*bia::10:10:10:10/128
               ::FFFF:12.12.12.12
                                0      100      0 410 ?

```

We still have the same ineffective result on CSR7, but at least now its IGP-independent using the shadow RR method. Neither XR v3.15 nor XR v5.3.0 supports multipath for VPNv6.

```

CSR7#show bgp vpnv6 unicast rd 100:13 ::10:10:10:10/128
BGP routing table entry for [100:13>::10:10:10:10/128, version 213
Paths: (2 available, best #2, no table)
  Additional-path-install
  Not advertised to any peer
  Refresh Epoch 4
  410
    ::FFFF:12.12.12.12 (metric 20) (via default) from 9.9.9.9 (9.9.9.9)
    Origin incomplete, metric 0, localpref 100, valid, internal,
    backup/repair
      Extended Community: RT:100:410
      Originator: 12.12.12.12, Cluster list: 9.9.9.9 , recursive-via-host
      mpls labels in/out nolabel/92012
      rx pathid: 0, tx pathid: 0
    Refresh Epoch 1

```


410

```
::FFFF:5.5.5.5 (metric 20) (via default) from 8.8.8.8 (8.8.8.8)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  Extended Community: RT:100:410
  Originator: 5.5.5.5, Cluster list: 8.8.8.8 , recursive-via-host
  mpls labels in/out nolabel/5010
  rx pathid: 0, tx pathid: 0x0
```

```
CSR7#show bgp vpnv6 unicast vrf B ::10:10:10:10/128
BGP routing table entry for [100:7]::10:10:10:10/128, version 212
Paths: (1 available, best #1, table B)
  Additional-path-install
  Advertised to update-groups:
    6
  Refresh Epoch 1
  410, imported path from [100:13]::10:10:10:10/128 (global)
    ::FFFF:5.5.5.5 (metric 20) (via default) from 8.8.8.8 (8.8.8.8)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:100:410
      Originator: 5.5.5.5, Cluster list: 8.8.8.8 , recursive-via-host
      mpls labels in/out nolabel/5010
      rx pathid: 0, tx pathid: 0x0
```

Last, we will change CSR5's RD back to 100:5 to achieve "optimal" path diversity without using BGP add-path on the RRs. Remember that it will wipe out all of the RT import/export policies and related BGP configurations. We will combine it with changing the local-preference on XRv13 to 200 for ::10:10:10:10/128 so that the remote PE's have a common exit point from the AS, but the VRF table should have the backup path installed in the FIB also.

```
! XRv13
vrf B
  address-family ipv6 unicast
    export route-policy RPL_EXPORT_VPNV4(166)
    no export route-target

! CSR5
vrf definition B
  no rd 100:13
  rd 100:5

router bgp 100
  address-family vpnv6 unicast
    bgp additional-paths install
    bgp additional-paths select best-external
  address-family vpnv6
    neighbor 8.8.8.8 advertise best-external
    neighbor 9.9.9.9 advertise best-external
    neighbor 11.11.11.11 advertise best-external
```

```

CSR5#show bgp vpnv6 unicast rd 100:5 ::10:10:10:10/128
BGP routing table entry for [100:5]::10:10:10:10/128, version 297
Paths: (2 available, best #2, table B)
  Advertised to update-groups:
    10
  Refresh Epoch 1
  410
    2001:10:5:10::10 (FE80::10) (via vrf B) from 10.5.10.10 (10.10.10.10)
      Origin incomplete, metric 0, localpref 100, valid, external, advertise-
best-external
      Extended Community: RT:100:410 , recursive-via-connected
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 7
  410, imported path from [100:13]::10:10:10:10/128 (global)
    ::FFFF:13.13.13.13 (metric 20) (via default) from 8.8.8.8 (8.8.8.8)
      Origin incomplete, metric 0, localpref 166, valid, internal, best
      Extended Community: RT:100:410
      Originator: 13.13.13.13, Cluster list: 8.8.8.8 , recursive-via-host
      mpls labels in/out nolabel/93013
      rx pathid: 0, tx pathid: 0x0

```

Now, each RR will have two paths, one via CSR5 and one via XRv13. On CSR9, we look at the RDs individually, and on CSR8 we look at them with a single command. XRv11's verification is shown in brief form for variety.

```

CSR9#show bgp vpnv6 unicast rd 100:5 ::10:10:10:10/128
BGP routing table entry for [100:5]::10:10:10:10/128, version 47
BGP Bestpath: igpmetric-ignore
Paths: (1 available, best #1, no table)
  Advertised to update-groups:
    2
  Refresh Epoch 4
  410, (Received from a RR-client)
    ::FFFF:5.5.5.5 (metric 10) (via default) from 5.5.5.5 (5.5.5.5)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:100:410
      mpls labels in/out nolabel/5044
      rx pathid: 0, tx pathid: 0x0

```

```

CSR9#show bgp vpnv6 unicast rd 100:13 ::10:10:10:10/128
BGP routing table entry for [100:13]::10:10:10:10/128, version 46
BGP Bestpath: igpmetric-ignore
Paths: (1 available, best #1, no table)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  410, (Received from a RR-client)

```

```

::FFFF:13.13.13.13 (metric 10) (via default) from 13.13.13.13
(13.13.13.13)
    Origin incomplete, metric 0, localpref 166, valid, internal, best
    Extended Community: RT:100:410
    mpls labels in/out nolabel/93013
    rx pathid: 0, tx pathid: 0x0

CSR8#show bgp vpnv6 unicast all ::10:10:10:10/128
BGP routing table entry for [100:5]::10:10:10:10/128, version 54
BGP Bestpath: igpmetric-ignore
Paths: (1 available, best #1, no table)
    Advertised to update-groups:
        1
    Refresh Epoch 2
    410, (Received from a RR-client)
        ::FFFF:5.5.5.5 (metric 30) (via default) from 5.5.5.5 (5.5.5.5)
            Origin incomplete, metric 0, localpref 100, valid, internal, best
            Extended Community: RT:100:410
            mpls labels in/out nolabel/5044
            rx pathid: 0, tx pathid: 0x0
BGP routing table entry for [100:13]::10:10:10:10/128, version 52
BGP Bestpath: igpmetric-ignore
Paths: (1 available, best #1, no table)
    Advertised to update-groups:
        1
    Refresh Epoch 1
    410, (Received from a RR-client)
        ::FFFF:13.13.13.13 (metric 10) (via default) from 13.13.13.13
(13.13.13.13)
    Origin incomplete, metric 0, localpref 166, valid, internal, best
    Extended Community: RT:100:410
    mpls labels in/out nolabel/93013
    rx pathid: 0, tx pathid: 0x0

RP/0/0/CPU0:XRv11#show bgp vpnv6 unicast rd all ::10:10:10:10/128

Status codes: s suppressed, d damped, h history, * valid, > best
                i - internal, r RIB-failure, S stale, N Nexthop-discard
Origin codes: i - IGP, e - EGP, ? - incomplete
   Network                Next Hop                Metric LocPrf Weight Path
Route Distinguisher: 100:5
*>i::10:10:10:10/128  5.5.5.5                        0      100          0 410 ?
Route Distinguisher: 100:13
*>i::10:10:10:10/128  13.13.13.13                       0      166          0 410 ?

```

Now, within CSR7 and XRv14 VRF table, we have multiple BGP routes and pre-installed backup paths in the FIB. Be sure to add the proper configurations to CSR7 and XRv14 so the backup paths can be installed.

```

! CSR7
router bgp 100
  address-family vpnv6
    bgp additional-paths install

CSR7#show bgp vpnv6 unicast vrf B ::10:10:10:10/128
BGP routing table entry for [100:7]::10:10:10:10/128, version 239
Paths: (2 available, best #2, table B)
  Additional-path-install
  Advertised to update-groups:
    6
  Refresh Epoch 1
  410, imported path from [100:5]::10:10:10:10/128 (global)
    ::FFFF:5.5.5.5 (metric 20) (via default) from 8.8.8.8 (8.8.8.8)
    Origin incomplete, metric 0, localpref 100, valid, internal,
    backup/repair
    Extended Community: RT:100:410
    Originator: 5.5.5.5, Cluster list: 8.8.8.8 , recursive-via-host
    mpls labels in/out nolabel/5044
    rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  410, imported path from [100:13]::10:10:10:10/128 (global)
    ::FFFF:13.13.13.13 (metric 20) (via default) from 8.8.8.8 (8.8.8.8)
    Origin incomplete, metric 0, localpref 166, valid, internal, best
    Extended Community: RT:100:410
    Originator: 13.13.13.13, Cluster list: 8.8.8.8 , recursive-via-host
    mpls labels in/out nolabel/93013
    rx pathid: 0, tx pathid: 0x0

CSR7#show ipv6 cef vrf B ::10:10:10:10/128 detail
::10:10:10:10/128, epoch 0, flags [rib defined all labels]
  recursive via 13.13.13.13 label 93013
    nexthop 153.7.8.8 GigabitEthernet2.578 label 8009
  recursive via 5.5.5.5 label 5044, repair
    nexthop 153.7.11.11 GigabitEthernet2.571 label 91018

! XRv14
router bgp 100
  address-family vpnv6 unicast
    additional-paths selection route-policy RPL_ADD_BACKUP

RP/0/0/CPU0:XRv14#show bgp vpnv6 unicast vrf B ::10:10:10:10/128 brief |
begin Network
      Network                Next Hop                Metric LocPrf Weight Path
Route Distinguisher: 100:14 (default for vrf B)
* i::10:10:10:10/128  5.5.5.5                0      100          0 410 ?
*>i                  13.13.13.13             0      166          0 410 ?

```

```

RP/0/0/CPU0:XRv14#show cef vrf B ipv6 ::10:10:10:10/128
::10:10:10:10/128, version 155, internal 0x5000001 0x0 (ptr 0xa13ba2f4) [1],
0x0 (0x0), 0x208 (0xa17fa12c)
Prefix Len 128, traffic index 0, precedence n/a, priority 3
via ::ffff:5.5.5.5, 4 dependencies, recursive, backup [flags 0x6100]
  path-idx 0 NHID 0x0 [0xa18ef0bc 0x0]
  recursion-via-/128
  next hop VRF - 'default', table - 0xe0000000
  next hop ::ffff:5.5.5.5 via ::ffff:5.5.5.5:0
  next hop 153.9.14.9/32 Gi0/0/0/0.594 labels imposed {9001 5044}
via ::ffff:13.13.13.13, 5 dependencies, recursive [flags 0x6000]
  path-idx 1 NHID 0x0 [0xa18ef200 0x0]
  recursion-via-/128
  next hop VRF - 'default', table - 0xe0000000
  next hop ::ffff:13.13.13.13 via ::ffff:13.13.13.13:0
  next hop 153.8.14.8/32 Gi0/0/0/0.584 labels imposed {8009 93013}
  next hop 153.9.14.9/32 Gi0/0/0/0.594 labels imposed {9006 93013}

```

BGP add-path is a very involved feature and there are many minor options worth testing. PIC, on the other hand, is a simple concept with almost no configuration involved. The idea of PIC is to create a hierarchical FIB to speed up convergence. Legacy FIBs were two-tiered, in which a prefix was mapped to an outgoing interface and next-hop value. Because the number of routes in a large router was orders of magnitude greater than the number of connected routes (possible next-hops), a reconvergence event would require a tremendous amount of updates to the FIB which took minutes to accomplish. For example, an Internet router with 400,000 routes may have 10 connected interfaces or so. This means an average of 40,000 routes all have the same next-hop, yet we still have to make 400,000 updates to each individual FIB entry. It would be much more efficient to update the next-hop references (using a pointer of sorts) while the FIB just references the pointer itself. This is called a three-tier or hierarchical FIB which greatly speeds convergence. A change in next-hop status due to IGP failure would result in the FIB updating a next-hop pointer, which is referenced by 40,000 routes. This is theoretically 40,000 times faster to converge since there is one change for 40,000 prefixes, rather than 40,000 next-hop updates for 40,000 prefixes. Enabling PIC is often done simply by designing a BGP add-path architecture.

To change the way the FIB is built, you can adjust the CEF table construction for fast convergence or efficient memory use. The two are mutually exclusive, but on modern routers the general preference (and the default) is to favor speed. We can adjust this in global configuration, and also verify the behavior as well. The default is shown below (convergence speed) while memory utilization is another option.

```

CSR1#show cef table | section favor
Output chain build favors:
platform:      not configured
CLI:           not configured
operational:    convergence-speed

```

```

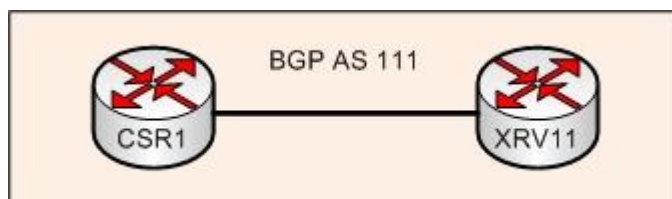
CSR1(config)#cef table output-chain build favor ?
convergence-speed    Favor faster convergence
memory-utilization   Favor smaller memory utilization

```

Additional Reading – Reference configurations “bgp-add-path”

27.2 BGP RT-filter unicast / IPv4 RT-filter feature

Most designs have a BGP hierarchy using VPN RRs. If a PE is only imported a subset of RTs, the RR will reflect all VPN routes to a PE where the majority of them are rejected due to the RT-import not being locally configured for any VRF. This is generally desirable, but would be better if it could have been filtered outbound on the sender versus inbound on the receiver. The feature is also called "RT constrained route distribution". Features like aggregation (notwithstanding the default route), redistribution, and iBGP synchronization are not supported for this feature. They don't really make sense anyway since this is meant to reduce routing advertisement based on RTs. When the feature is not negotiated, the default filter states that all RTs should be sent to a peer, regardless of RT value. The local router will just reject the ones it doesn't actually need. Ensure your routers have enough memory to store all these RT-filter "routes" (also called RTC prefixes) as it could create a sizable footprint in the BGP LocRIB on the RRs. This feature is loosely analogous to Outbound Route Filtering (ORF).



Without enabling the feature yet, we configure a basic VPNv4 route exchange with proper RT import/export policies. Both routers show the prefix attributes and the VPNv4 allocated label. This was successfully imported into at least one VRF, or at a minimum, not filtered from the VPNv4 LocRIB.

```

CSR1#debug bgp vpnv4 unicast updates in
BGP(4): 11.0.0.11 rcvd UPDATE w/ attr: nexthop 11.0.0.11, origin ?, localpref
100, metric 0, extended community RT:1:111
BGP(4): 11.0.0.11 rcvd 1:1:100.11.11.11/32, label 91000
BGP(4): Revise route installing 1 of 1 routes for 100.11.11.11/32 ->
11.0.0.11(A) to A IP table

```

We see similar output in XR, except that it is a little harder to read.

```

RP/0/0/CPU0:XRv11#debug bgp update vpnv4 unicast
bgp[1046]: [default-rtr] (vpn4u): nexthop 11.0.0.1/32, origin ?, localpref
100, metric 0, extended community RT:1:101
bgp[1046]: [default-rtr] (vpn4u): Received prefix 2ASN:1:1:100.1.1.1/32 (path
ID: none) with MPLS label 1002 from neighbor 11.0.0.1

```

```
bgp[1046]: [default-upd] VRF A: table-attr walk for table TBL:A (1/1), resume
version 0, subgrp version 8, target version 10
```

On CSR1, we remove the RT import for 1:111, so CSR1 should reject it. The debug doesn't explicitly say the RT was rejected, but the generic "extended community not supported" normally indicates an RT import issue.

```
! CSR1
BGP(4): 11.0.0.11 rcvd UPDATE w/ attr: nexthop 11.0.0.11, origin ?, localpref
100, metric 0, extended community RT:1:111
BGP(4): 11.0.0.11 rcvd 1:1:100.11.11.11/32, label 91000 -- DENIED due to:
extended community not supported;
BGP(4): no valid path for 1:1:100.11.11.11/32
BGP: topo A:VPNv4 Unicast:base Remove_fwdroute for 1:1:100.11.11.11/32
```

Likewise on XRv11, we remove the RT import for 1:101. The XR debug is more explicit and tells you there is an issue with the RT import policy.

```
! XRv11
bgp[1046]: [default-rtr] (vpn4u): nexthop 11.0.0.1/32, origin ?, localpref
100, metric 0, extended community RT:1:101
bgp[1046]: [default-rtr] (vpn4u): Received prefix 2ASN:1:1:100.1.1.1/32 (path
ID: none) with MPLS label 1002 from neighbor 11.0.0.1
[default-rtr] (vpn4u): Prefix 2ASN:1:1:100.1.1.1/32 (path ID: none) received
from 11.0.0.1 DENIED RT extended community is not imported locally
bgp[1046]: [default-upd] (vpn4u): No unreachable (no path is available) sent
to sub-group 0.1 (Regular) with 2ASN:1:1:100.1.1.1/32 - already withdrawn
```

Assuming that these routers don't want one another's routes, not importing the RTs ensures the VPN topology is functional but still requires the receiving PE to process each route individually. This can consume router resources and waste network bandwidth. We will enable the rt-filter feature on both routers. It is called different things between XE and XR, though. Specifically, the XE AF is "rt-filter unicast" and the XR AF is "ipv4 rt-filter". Assuming a semi-worthless configuration where neither router imports any RTs, the LocRIB for the RT-filter AF is empty, which implicitly says "my neighbor requested no RTs, so I send him nothing". The debug on CSR1 indicates that the prefix was denied, but doesn't give much detail. We know it's due to the dynamic RT filter.

```
! CSR1
BGP(4): (base) 11.0.0.11 send UPDATE (format) 1:1:100.1.1.1/32, next
11.0.0.1, label 1002, metric 0, path Local, extended community RT:1:101
BGP(4): (base) 11.0.0.11 Peer based policy member(format) 1:1:100.1.1.1/32,
next 11.0.0.1 result(denied)
```

The debug on XRv11 indicates that the update was not sent to CSR1 and is more detailed than the CSR debug.

```
!XRv11
bgp[1046]: [default-upd] (vpn4u): Deny UPDATE to filter-group 0.2 (Regular,
pelem Regular) for 2ASN:1:1:100.11.11.11/32 (changedfl=0x1000000/0x0), path
reason Dropped by RT Filter
bgp[1046]: [default-upd] (vpn4u): No unreachable (Dropped by RT Filter) sent
to sub-group 0.1 (Regular) with 2ASN:1:1:100.11.11.11/32 - already withdrawn
```

Next, we will add the baseline RT import policy to CSR1 but not to XRv11 yet. The dynamic RT filter should be updated so that the RTs can be advertised between the neighbors. We will activate debug for both VPNv4 and rt-filter AFIs. BGP(11) in this case is rt-filter and BGP(4) is VPNv4.

```
CSR1#debug bgp vpnv4 unicast updates in
CSR1#debug bgp rt-filter unicast updates in
BGP: 111:2:1:111 bestpath, is sourced, is not multipath
BGP(11): 11.0.0.11 NEXT_HOP self is set for sourced RT Filter for net
111:2:1:111
BGP(11): (base) 11.0.0.11 send UPDATE (format) 111:2:1:111, next 11.0.0.1,
metric 0, path Local
BGP(4): 11.0.0.11 rcvd UPDATE w/ attr: nexthop 11.0.0.11, origin ?, localpref
100, metric 0, extended community RT:1:111
BGP(4): 11.0.0.11 rcvd 1:1:100.11.11.11/32, label 91000
BGP(4): Revise route installing 1 of 1 routes for 100.11.11.11/32 ->
11.0.0.11(A) to A IP table
```

First, CSR1 creates an rt-filter RTC prefix that contains A:B:C:D ...

A = BGP AS number

B = Usually 2 for the extended-community type code

C:D = The route target we are importing

```
CSR1#show bgp rtfiler unicast all detail
BGP routing table entry for 111:2:1:111, version 2
  Paths: (1 available, best #1)
  Advertised to update-groups:
    3
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (1.1.1.1)
      Origin IGP, localpref 100, weight 32768, valid, sourced, local, best
      RT generation: import
      rx pathid: 0, tx pathid: 0x0
```

We can also see this on XRv12 as being received from CSR1, both in debug and show commands. XR will consult this table before advertising VPN routes to a peer to ensure those RT's are requested. This is a similar operation to the outbound route filtering (ORF) capability used to communicate an inbound prefix filter to the remote peer.


```

! XRv11
bgp[1046]: [default-upd] (vpn4u): Permit UPDATE to filter-group 0.2 (Regular,
pelem Regular) for 2ASN:1:1:100.11.11.11/32 (changedfl=0x1000000/0x0), path

RP/0/0/CPU0:XRv11#show bgp ipv4 rt-filter 111:2:1:111/96
BGP routing table entry for 111:2:1:111/96
Versions:
  Process          bRIB/RIB  SendTblVer
  Speaker          2         2
Paths: (1 available, best #1)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  Local
    11.0.0.1 from 11.0.0.1 (1.1.1.1)
    Origin IGP, metric 0, localpref 100, valid, internal, best, group-best
    Received Path ID 0, Local Path ID 1, version 2

```

Next, let's import some more bogus RTs on CSR1. The router creates three new RTC prefixes in the rt-filter unicast AF and advertises them to XRv11.

```

CSR1(config-vrf)#route-target import 1:901
CSR1(config-vrf)#route-target import 1:902
CSR1(config-vrf)#route-target import 1:903

! CSR1
BGP: 111:2:1:901 bestpath, is sourced, is not multipath
BGP(11): 11.0.0.11 NEXT_HOP self is set for sourced RT Filter for net
111:2:1:901
(base) 11.0.0.11 send UPDATE (format) 111:2:1:901, next 11.0.0.1, metric 0,
path Local

BGP(11): 111:2:1:902 bestpath, is sourced, is not multipath
BGP(11): 11.0.0.11 NEXT_HOP self is set for sourced RT Filter for net
111:2:1:902
(base) 11.0.0.11 send UPDATE (format) 111:2:1:902, next 11.0.0.1, metric 0,
path Local

BGP(11): 111:2:1:903 bestpath, is sourced, is not multipath
BGP(11): 11.0.0.11 NEXT_HOP self is set for sourced RT Filter for net
111:2:1:903
BGP(11): (base) 11.0.0.11 send UPDATE (format) 111:2:1:903, next 11.0.0.1,
metric 0, path Local

```

XRv11 isn't going to do anything with those as no one is exporting VPN routes with those RTs, but we can see a 1:1 correlation between imported RT's and what is advertised between the neighbors.

```
RP/0/0/CPU0:XRv11#show bgp ipv4 rt-filter
[snip]
Status codes: s suppressed, d damped, h history, * valid, > best
                i - internal, r RIB-failure, S stale, N Nexthop-discard
Origin codes: i - IGP, e - EGP, ? - incomplete
   Network                Next Hop                Metric LocPrf Weight Path
*>i111:2:1:111/96          11.0.0.1                  0      100      0 i
*>i111:2:1:901/96          11.0.0.1                  0      100      0 i
*>i111:2:1:902/96          11.0.0.1                  0      100      0 i
*>i111:2:1:903/96          11.0.0.1                  0      100      0 i
```

On CSR1, we can look at the details of one of these RTs to see an extended community "RT generation" set to value "import", which commands the VRF to import routes with the matching RT.

```
CSR1#show bgp rtfilter unicast rt 1:901
BGP routing table entry for 111:2:1:901, version 3
Paths: (1 available, best #1)
  Advertised to update-groups:
    4
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (1.1.1.1)
    Origin IGP, localpref 100, weight 32768, valid, sourced, local, best
    RT generation: import
    rx pathid: 0, tx pathid: 0x0
```

For good measure, we will create bogus RT's on XRv11 to observe any relevant debug messages. The XR debugs are very detailed so irrelevant parts have been omitted.

```
RP/0/0/CPU0:XRv11(config-vrf-import-rt)#1:801
RP/0/0/CPU0:XRv11(config-vrf-import-rt)#1:802
RP/0/0/CPU0:XRv11(config-vrf-import-rt)#1:803

! XRv11
bgp[1046]: [default-upd] (ipv4rtf): ===UPDATE===: tbl=TBL:default (1/132),
afi=11: [snip]: net=111:2:1:803/96 [snip]
bgp[1046]: [default-upd] (ipv4rtf): Permit UPDATE to filter-group 0.2
(Regular, pelem Regular) for 111:2:1:803/96 (changedfl=0x0/0x1000), path

bgp[1046]: [default-upd] (ipv4rtf): ===UPDATE===: tbl=TBL:default (1/132),
afi=11: [snip]: net=111:2:1:802/96 [snip]
bgp[1046]: [default-upd] (ipv4rtf): Permit UPDATE to filter-group 0.2
(Regular, pelem Regular) for 111:2:1:802/96 (changedfl=0x0/0x1000), path

bgp[1046]: [default-upd] (ipv4rtf): ===UPDATE===: tbl=TBL:default (1/132),
afi=11: [snip]: net=111:2:1:801/96 [snip]
bgp[1046]: [default-upd] (ipv4rtf): Permit UPDATE to filter-group 0.2
(Regular, pelem Regular) for 111:2:1:801/96 (changedfl=0x0/0x1000), path
```

Verify that CSR1 has received them. We will look at one of them as an example. No surprises. This effectively says "11.0.0.11 wants RT:1:801".

```
CSR1#show bgp rtfilter unicast rt 1:801
BGP routing table entry for 111:2:1:801, version 8
Paths: (1 available, best #1)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    11.0.0.11 from 11.0.0.11 (11.11.11.11)
      Origin IGP, localpref 100, valid, internal, best
      rx pathid: 0, tx pathid: 0x0
```

As a final test, let's pretend XRv11 is a route-reflector for VPNv4 and the RT-filter AFIs. Ideally, CSR1 should be sending all RTs it exports to XRv11 without XRv11 having to explicitly request it via individual RTC prefixes. The "default-originate" command violates many of the formatting rules defined earlier, such as encoding the BGP AS number, community code, and RT. The zeroes in the RTC instruct the RR client to send all VPN routes with any RTs to the BGP peer, which is a RR. **Notwithstanding the default route, aggregation is not supported with the RT filter feature.**

```
! XRv11
router bgp 111
 neighbor 11.0.0.1
   address-family vpnv4 unicast
     route-reflector-client
   address-family ipv4 rt-filter
     route-reflector-client
   default-originate
```

CSR1 shows this "default" RTC and uses this to send all RT's to the peer 11.0.0.11, which is the RR.

```
CSR1#show bgp rtfilter unicast all
[snip]
```

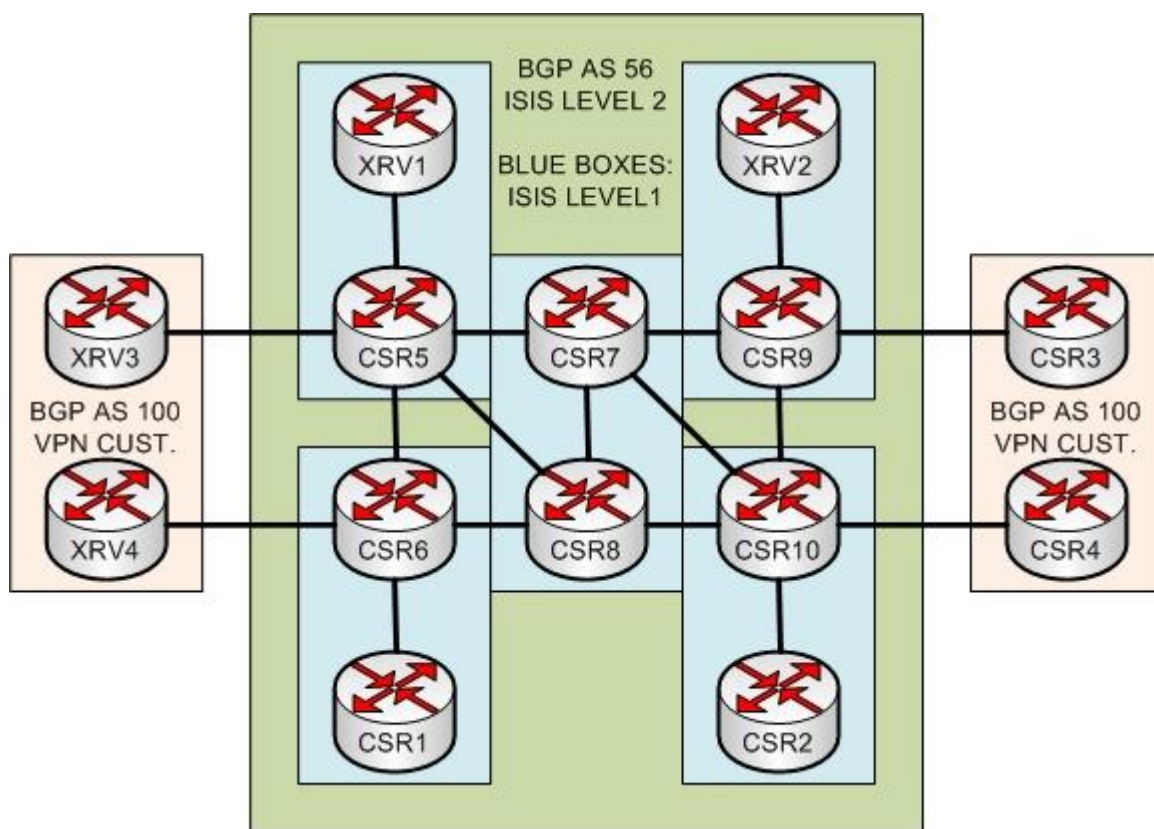
	Network	Next Hop	Metric	LocPrf	Weight	Path
*>i	0:0:0:0	11.0.0.11		100	0	i
*>	111:2:1:111	0.0.0.0			32768	i
*>	111:2:1:901	0.0.0.0			32768	i
*>	111:2:1:902	0.0.0.0			32768	i
*>	111:2:1:903	0.0.0.0			32768	i

Additional Reading – Reference configurations "rtfilter" and "rtfilter-rr"

27.3 BGP RR-group and Selective RT Retention

This lab investigates how to enable efficient route retention on BGP RRs when using VPNv4 and/or VPNv6. Traditionally, BGP RRs for VPN routes must always maintain all routes from RR-clients, regardless

of their RT values. This is because the RR expects the PE to filter RTs that are unwanted. The feature in the previous chapter is one solution to the problem which actively enables a communication method between devices. This lab uses a passive feature that is local only, requires no additional AFI negotiation, and can save memory on RRs/PEs in a highly distributed RR architecture. The diagram is shown below. Several unrelated advanced technologies have been enabled for further testing in other sections, such as TE auto-primary tunnels and MVPN, but those are not the focus of this lab. CSR1, CSR2, XRv1, and XRv2 are BGP RRs for VPNv4 and VPNv6. CSRs 5 through 10 are LSRs within BGP AS 56. The IS-IS areas are built in such a way to shield the RRs from the “large” IS-IS level-2 domain. This design attempts to move BGP RRs entirely out of the control plane, both from a forwarding and feature perspective (not running MPLS, PIM, etc). All PEs peer with all RRs for VPNv4 and VPNv6. There are 2 VPNs, RED and BLUE. Each CE runs VRF-lite to support both VPNs and each PE provides service for both VPNs. VPN RED uses BGP for IPv4/v6 routing while VPN BLUE uses RIPv2 and OSPFv3 for IPv4 and IPv6 routing, respectively.



The basic configurations, such as interfaces, IS-IS routing, and BGP peers are not discussed in detail. Each router runs IS-IS on all interfaces as shown in the diagram with the loopback interfaces advertised. Instead, we quickly verify the network using cursory show commands. On CSR6, we verify the IS-IS level-2 database. In a single show command, I reveal all of the IS-IS level-2 routers and their links while hiding the IP prefixes. Quickly skimming this output, we confirm that it corresponds with the diagram above.

```
R6#show isis database level-2 detail | include ^[RX]|Extended
```

R5.00-00	0x0000001A	0xFCCA	653	0/0/0
Metric: 10	IS-Extended	R7.00		
Metric: 10	IS-Extended	R6.00		
Metric: 10	IS-Extended	R8.00		
R6.00-00	* 0x00000018	0xFDEF	416	0/0/0
Metric: 10	IS-Extended	R8.00		
Metric: 10	IS-Extended	R5.00		
R7.00-00	0x0000001A	0x2682	1122	0/0/0
Metric: 10	IS-Extended	R10.00		
Metric: 10	IS-Extended	R9.00		
Metric: 10	IS-Extended	R5.00		
Metric: 10	IS-Extended	R8.00		
R8.00-00	0x0000001B	0x0B9D	975	0/0/0
Metric: 10	IS-Extended	R7.00		
Metric: 10	IS-Extended	R6.00		
Metric: 10	IS-Extended	R10.00		
Metric: 10	IS-Extended	R5.00		
R9.00-00	0x00000019	0x7AE0	562	0/0/0
Metric: 10	IS-Extended	R10.00		
Metric: 10	IS-Extended	R7.00		
R10.00-00	0x0000001C	0x8E89	996	0/0/0
Metric: 10	IS-Extended	R7.00		
Metric: 10	IS-Extended	R8.00		
Metric: 10	IS-Extended	R9.00		

Since CSR6 is an L1L2 router, it also has a level-1 LSPDB. This is much smaller, but we expect to see CSR1 along with a transit link. CSR6 sets the attached-bit so that CSR1 installs a default route via CSR6. The other three L1L2 routers do this as well (not shown below).

```
R6#show isis database level-1 detail | include ^[RX]|Extended
R1.00-00      0x0000000B  0xCDE7      603      0/0/0
Metric: 10    IS-Extended R6.00
R6.00-00      * 0x0000000B  0x8844     1161      1/0/0
Metric: 10    IS-Extended R1.00
```

MPLS TE is enabled for IS-IS L2 only. This output should mirror the IS-IS level-2 LSPDB exactly as TE is enabled on all nodes and all links. This is a critical part of the MPLS transport infrastructure as LDP is not enabled on IS-IS interfaces.

```
R6#show mpls traffic-eng topology brief | include IGP_Id
IGP Id: 0000.0000.0005.00, MPLS TE Id:56.0.0.5 Router Node (isis level-2)
  link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0007.00, nbr_node_id:14, gen:126
  link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0006.00, nbr_node_id:11, gen:126
  link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0008.00, nbr_node_id:15, gen:126
IGP Id: 0000.0000.0006.00, MPLS TE Id:56.0.0.6 Router Node (isis level-2)
  link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0008.00, nbr_node_id:15, gen:123
  link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0005.00, nbr_node_id:10, gen:123
IGP Id: 0000.0000.0007.00, MPLS TE Id:56.0.0.7 Router Node (isis level-2)
  link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0010.00, nbr_node_id:16, gen:125
```

```

link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0009.00, nbr_node_id:12, gen:125
link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0005.00, nbr_node_id:10, gen:125
link[3]: Point-to-Point, Nbr IGP Id: 0000.0000.0008.00, nbr_node_id:15, gen:125
IGP Id: 0000.0000.0008.00, MPLS TE Id:56.0.0.8 Router Node (isis level-2)
link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0007.00, nbr_node_id:14, gen:127
link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0006.00, nbr_node_id:11, gen:127
link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0010.00, nbr_node_id:16, gen:127
link[3]: Point-to-Point, Nbr IGP Id: 0000.0000.0005.00, nbr_node_id:10, gen:127
IGP Id: 0000.0000.0009.00, MPLS TE Id:56.0.0.9 Router Node (isis level-2)
link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0010.00, nbr_node_id:16, gen:116
link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0007.00, nbr_node_id:14, gen:116
IGP Id: 0000.0000.0010.00, MPLS TE Id:56.0.0.10 Router Node (isis level-2)
link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0007.00, nbr_node_id:14, gen:117
link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0008.00, nbr_node_id:15, gen:117
link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0009.00, nbr_node_id:12, gen:117

```

MPLS LDP is not enabled directly on the transit links. Instead, TE auto-primary tunnels are configured. These tunnels are automatically created to every one-hop peer as determined by the TED. The tunnels have autoroute announce and FRR automatically configured as well. I enable tLDP on them so that routers more than one hop away (CSR6 to CSR9, for example) can still communicate. This feature is discussed in detail in the TE section, but in summary, the idea is to “protect every link” in the topology. This offers protection to LDP and mLDP LSPs, not just TE LSPs, and it scales quite well. CSR6, for example, is the headend for tunnels to CSR5 and CSR8. It is also the tail end for the complementary tunnels from those routers along the reverse path. I do not use this technique often which is why I configure it for variety here. Normally this feature is paired with auto-backup tunnels for TE-FRR, but I omit that configuration for clarity.

```

! All LSRs
mpls traffic-eng auto-tunnel primary onehop
mpls traffic-eng auto-tunnel primary config unnumbered-interface Loopback0
mpls traffic-eng auto-tunnel primary config mpls ip
mpls traffic-eng auto-tunnel primary tunnel-num min 500 max 599

```

```

R6#show mpls traffic-eng auto-tunnel primary
State: Enabled
  Auto primary tunnels: 2 (up: 2, down: 0)
  Tunnel ID Range: 500 - 599
  Check for deletion of FRR Active onehop tunnels every:0 Sec
  Config:
    unnumbered-interface: Loopback0
    mpls ip: TRUE

```

```
R6#show mpls traffic-eng tunnels brief | begin TUNNEL
```

P2P TUNNELS/LSPs:

TUNNEL NAME STATE/PROT	DESTINATION	UP IF	DOWN IF	
R6_t500	56.0.0.5	-	Gi2.556	up/up
R6_t501	56.0.0.8	-	Gi2.568	up/up
R5_t500	56.0.0.6	Gi2.556	-	up/up

```
R8_t500          56.0.0.6          Gi2.568    -          up/up
Displayed 2 (of 2) heads, 0 (of 0) midpoints, 2 (of 2) tails
```

The one-hop tunnels always bind implicit-null as the headend is always the penultimate hop. For this reason, many tunnels are PE-P or P-P, requiring an additional label in the stack. Thanks to tLDP, CSR6 is able to send MPLS-encapsulated traffic to CSR10, which is multiple hops away. This is not specific to the features addressed in this lab, but is the basis for all MPLS forwarding and must be verified.

```
R6#show ip route 56.0.0.10
Routing entry for 56.0.0.10/32
  Known via "isis", distance 115, metric 20
  Tag 56, type level-2
  Redistributing via isis 56
  Last update from 56.0.0.8 on Tunnel501, 01:12:19 ago
  Routing Descriptor Blocks:
  * 56.0.0.8, from 56.0.0.10, 01:12:19 ago, via Tunnel501
    Route metric is 20, traffic share count is 1
    Route tag 56

R6#show mpls ldp bindings 56.0.0.10 32 neighbor 56.0.0.8
  lib entry: 56.0.0.10/32, rev 8
    remote binding: lsr: 56.0.0.8:0, label: 8010

R6#show mpls traffic-eng tunnels tunnel 501 | include Label
  InLabel   : -
  OutLabel  : GigabitEthernet2.568, implicit-null

R6#traceroute 56.0.0.10 source 56.0.0.6
Type escape sequence to abort.
Tracing the route to 56.0.0.10
VRF info: (vrf in name/id, vrf out name/id)
 1 56.6.8.8 [MPLS: Label 8010 Exp 0] 5 msec 3 msec 3 msec
 2 56.8.10.10 3 msec 4 msec 3 msec
```

Next, we configure BGP towards the RRs and the configuration on all four PEs is identical. The configuration is simple as all PEs peer to all four RRs using all supported AFIs. The VRFs will be configured after the core infrastructure is built. IPv4 MDT is also negotiated to support PIM/GRE MVPN, which is beyond the scope of this test as well. Since that AFI has no concept of RT, there is no ability to “filter” prefixes from IPv4 MDT based on RT extended communities.

```
! CSR5, CSR6, CSR9, CSR10
router bgp 56
  template peer-session IBGP
    remote-as 56
    update-source Loopback0
    timers 10 40
```

```

no bgp default ipv4-unicast
neighbor 56.0.0.1 inherit peer-session IBGP
neighbor 56.0.0.2 inherit peer-session IBGP
neighbor 56.0.0.11 inherit peer-session IBGP
neighbor 56.0.0.12 inherit peer-session IBGP

```

```

address-family vpnv4
neighbor 56.0.0.1 activate
neighbor 56.0.0.2 activate
neighbor 56.0.0.11 activate
neighbor 56.0.0.12 activate

```

```

address-family ipv4 mdt
neighbor 56.0.0.1 activate
neighbor 56.0.0.2 activate
neighbor 56.0.0.11 activate
neighbor 56.0.0.12 activate

```

```

address-family vpnv6
neighbor 56.0.0.1 activate
neighbor 56.0.0.2 activate
neighbor 56.0.0.11 activate
neighbor 56.0.0.12 activate

```

The configuration on CSR1 and CSR2 is identical as well. The RRs peer with all PEs and negotiate VPNv4, VPNv6, and IPv4 MDT. This is almost identical to the PE configuration except all peers are RR clients.

```

! CSR1 and CSR2
router bgp 56
template peer-policy RR_CLIENT
route-reflector-client
send-community both

template peer-session IBGP
remote-as 56
update-source Loopback0
timers 10 40

no bgp default ipv4-unicast
neighbor 56.0.0.5 inherit peer-session IBGP
neighbor 56.0.0.6 inherit peer-session IBGP
neighbor 56.0.0.9 inherit peer-session IBGP
neighbor 56.0.0.10 inherit peer-session IBGP

address-family vpnv4
neighbor 56.0.0.5 activate
neighbor 56.0.0.5 inherit peer-policy RR_CLIENT
neighbor 56.0.0.6 activate
neighbor 56.0.0.6 inherit peer-policy RR_CLIENT

```



```

neighbor 56.0.0.9 activate
neighbor 56.0.0.9 inherit peer-policy RR_CLIENT
neighbor 56.0.0.10 activate
neighbor 56.0.0.10 inherit peer-policy RR_CLIENT

```

```

address-family ipv4 mdt
neighbor 56.0.0.5 activate
neighbor 56.0.0.5 inherit peer-policy RR_CLIENT
neighbor 56.0.0.6 activate
neighbor 56.0.0.6 inherit peer-policy RR_CLIENT
neighbor 56.0.0.9 activate
neighbor 56.0.0.9 inherit peer-policy RR_CLIENT
neighbor 56.0.0.10 activate
neighbor 56.0.0.10 inherit peer-policy RR_CLIENT

```

```

address-family vpnv6
bgp rr-group EXTCOML_RR_RED
neighbor 56.0.0.5 activate
neighbor 56.0.0.5 inherit peer-policy RR_CLIENT
neighbor 56.0.0.6 activate
neighbor 56.0.0.6 inherit peer-policy RR_CLIENT
neighbor 56.0.0.9 activate
neighbor 56.0.0.9 inherit peer-policy RR_CLIENT
neighbor 56.0.0.10 activate
neighbor 56.0.0.10 inherit peer-policy RR_CLIENT

```

From a logical standpoint, XRv1 and XRv2 perform the same function. Their configuration is identical as well and effectively mirrors the XE configuration above. The configuration is long only due to the number of clients and AFIs, not due to complexity.

```

! XRv1 and XRv2
router bgp 56
  bgp cluster-id 56.0.0.11
  address-family vpnv4 unicast
  address-family vpnv6 unicast
  address-family ipv4 mdt

  af-group AF_VPNV4 address-family vpnv4 unicast
    route-reflector-client
  af-group AF_VPNV6 address-family vpnv6 unicast
    route-reflector-client
  af-group AF_IPV4_MDT address-family ipv4 mdt
    route-reflector-client

  session-group IBGP
    remote-as 56
    timers 10 40
    update-source Loopback0

```

```

neighbor 56.0.0.5
  use session-group IBGP
  address-family vpnv4 unicast
    use af-group AF_VPNV4
  address-family vpnv6 unicast
    use af-group AF_VPNV6
  address-family ipv4 mdt
    use af-group AF_IPV4_MDT

neighbor 56.0.0.6
  use session-group IBGP
  address-family vpnv4 unicast
    use af-group AF_VPNV4
  address-family vpnv6 unicast
    use af-group AF_VPNV6
  address-family ipv4 mdt
    use af-group AF_IPV4_MDT

neighbor 56.0.0.9
  use session-group IBGP
  address-family vpnv4 unicast
    use af-group AF_VPNV4
  address-family vpnv6 unicast
    use af-group AF_VPNV6
  address-family ipv4 mdt
    use af-group AF_IPV4_MDT

neighbor 56.0.0.10
  use session-group IBGP
  address-family vpnv4 unicast
    use af-group AF_VPNV4
  address-family vpnv6 unicast
    use af-group AF_VPNV6
  address-family ipv4 mdt
    use af-group AF_IPV4_MDT

```

At this point, all BGP sessions should form. **Note that BGP (XE and XR) will never initiate a session when the longest-match route to a peer is the default route.** This means all of the RRs, at this point, will be passively listening for BGP OPEN messages from their peers. They can reply along the default route, but not initiate. Once the sessions form, we quickly verify them. Rather than document all sessions on all RRs for all AFIs, I show CSR1 and XRv1 only. All sessions are verified as up but are not documented. No routes are currently received by the RRs as the VRFs have not yet been configured.

```

R1#show bgp vpnv4 unicast all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ  Up/Down  State/PfxRcd
56.0.0.5      4      56    825    895      77    0    0 02:02:09      0
56.0.0.6      4      56    816    834      77    0    0 02:02:24      0

```

```

56.0.0.9      4      56      834      836      77      0      0 02:02:16      0
56.0.0.10     4      56      821      832      77      0      0 02:02:13      0

R1#show bgp vpnv6 unicast all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent    TblVer  InQ  OutQ  Up/Down  State/PfxRcd
56.0.0.5      4      56      826      895      101    0      0 02:02:12      0
56.0.0.6      4      56      816      834      101    0      0 02:02:27      0
56.0.0.9      4      56      834      836      101    0      0 02:02:19      0
56.0.0.10     4      56      822      832      101    0      0 02:02:16      0

R1#show bgp ipv4 mdt all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent    TblVer  InQ  OutQ  Up/Down  State/PfxRcd
56.0.0.5      4      56      826      896      5      0      0 02:02:18      0
56.0.0.6      4      56      817      835      5      0      0 02:02:33      0
56.0.0.9      4      56      835      837      5      0      0 02:02:26      0
56.0.0.10     4      56      822      833      5      0      0 02:02:22      0

RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast summary | begin ^Neigh
Neighbor      Spk    AS MsgRcvd MsgSent    TblVer  InQ  OutQ  Up/Down  St/PfxRcd
56.0.0.5      0      56      730      724      59      0      0 01:34:55      0
56.0.0.6      0      56      712      690      59      0      0 01:34:42      0
56.0.0.9      0      56      736      690      59      0      0 01:34:46      0
56.0.0.10     0      56      728      690      59      0      0 01:34:45      0

RP/0/0/CPU0:XRv1#show bgp vpnv6 unicast summary | begin ^Neigh
Neighbor      Spk    AS MsgRcvd MsgSent    TblVer  InQ  OutQ  Up/Down  St/PfxRcd
56.0.0.5      0      56      731      725      49      0      0 01:34:58      0
56.0.0.6      0      56      713      691      49      0      0 01:34:44      0
56.0.0.9      0      56      737      691      49      0      0 01:34:49      0
56.0.0.10     0      56      729      691      49      0      0 01:34:48      0

RP/0/0/CPU0:XRv1#show bgp ipv4 mdt summary | begin ^Neigh
Neighbor      Spk    AS MsgRcvd MsgSent    TblVer  InQ  OutQ  Up/Down  St/PfxRcd
56.0.0.5      0      56      731      725      7      0      0 01:35:02      0
56.0.0.6      0      56      713      691      7      0      0 01:34:49      0
56.0.0.9      0      56      737      691      7      0      0 01:34:53      0
56.0.0.10     0      56      729      691      7      0      0 01:34:52      0

```

To prove that BGP cannot initiate sessions via a default route, we can check the TCP brief on CSR1 and XRv1. Note that each RR will have one /32 route to the connected L1L2 router, so it is possible that one session may be initiated by the RR. For example, CSR1 has a route to 56.0.0.5/32 and XRv1 has a route to 56.0.0.6/32 as a result of being connected to CSR5 and CSR6, respectively. XRv1 initiated the connection to CSR6 coincidentally, while CSR1 did not. The other 3 sessions that are reachable via the default route are highlighted; the RRs are servers (not initiators) for all of them.

```

R1#show tcp brief
TCB          Local Address          Foreign Address          (state)
7FCBA18381D0 56.0.0.1.179            56.0.0.10.40173         ESTAB
7FCBAC1EB498 56.0.0.1.179            56.0.0.6.49282          ESTAB
7FCBA1837850 56.0.0.1.179            56.0.0.9.20797          ESTAB
7FCBA1889000 56.0.0.1.179            56.0.0.5.44310          ESTAB

```

```
RP/0/0/CPU0:XRv1#show tcp brief | include ESTAB
0x10180278 0x60000000 0 0 56.0.0.11:179 56.0.0.9:42848 ESTAB
0x1017c368 0x60000000 0 0 56.0.0.11:30859 56.0.0.5:179 ESTAB
0x10145f9c 0x60000000 0 0 56.0.0.11:179 56.0.0.10:25813 ESTAB
0x1016b444 0x60000000 0 0 56.0.0.11:179 56.0.0.6:32154 ESTAB
```

The PE-CE setup is configuration intensive but very simple. As described earlier, each CE (CSR3, CSR4, XRv3, and XRv4) runs VRF-lite so it can represent multiple CE routers. VPN RED uses BGP while VPN BLUE uses RIPv2/OSPFv3. The CE configurations are not shown here, and only snippets from the PEs are shown. There are 2 RDs and 8 RTs in use. VPN RED uses RD 56:100 and VPN BLUE uses RD 56:200 on all PEs. Individual PEs use different import/export RTs shown below in the configuration. CSR5, for example, exports RT:56:213 and imports 3 others. The other 3 represent the remote PEs. The same is true inside both VPNs and both AFls, totaling 8 RTs. Having RT diversity is important to test the RR-group and RT retention features. The MVPN configuration is shown for reference only. It is important to note that RED RTs contain the BGP ASN and a number within the 100-199 range. VPN BLUE is similar except the number is in the 200-299 range, which is important for matching.

```
! CSR5
vrf definition BLUE
  rd 56:200
  vpn id 56:200
  route-target export 56:213
  route-target import 56:214
  route-target import 56:203
  route-target import 56:204

  address-family ipv4
    mdt preference mldp
    mdt default mpls mldp 56.0.0.7
  address-family ipv6
    mdt preference mldp
    mdt default mpls mldp 56.0.0.7

vrf definition RED
  rd 56:100
  route-target export 56:113
  route-target import 56:114
  route-target import 56:103
  route-target import 56:104

  address-family ipv4
    mdt default 232.0.0.100
  address-family ipv6
    mdt default 232.0.0.100
```

All of the PEs use a very similar configuration except the RT import/export policies are adjusted slightly. The full-mesh L3VPN is the end-state. The basic PE-CE routing is shown below, along with the VRF-aware

RIPv2 and OSPFv3 process inside VPN BLUE. This is basic L3VPN and is not discussed in detail. The CE configurations are not shown, but they are configured similarly to exchange routes with the PE.

```
! CSR5
route-map RM_CONN_BGP deny 10

router bgp 56
 address-family ipv4 vrf BLUE
   redistribute connected route-map RM_CONN_BGP
   redistribute rip

 address-family ipv6 vrf BLUE
   redistribute ospf 56

 address-family ipv4 vrf RED
   neighbor 10.5.13.13 remote-as 100
   neighbor 10.5.13.13 activate
   neighbor 10.5.13.13 as-override

 address-family ipv6 vrf RED
   neighbor FD00:10:5:13::13 remote-as 100
   neighbor FD00:10:5:13::13 activate
   neighbor FD00:10:5:13::13 as-override

router ospfv3 56
 address-family ipv6 unicast vrf BLUE
   redistribute bgp 56
   router-id 56.0.0.5
   prefix-suppression

router rip
 address-family ipv4 vrf BLUE
   redistribute bgp 56 metric transparent
   network 10.0.0.0
   no auto-summary
   version 2
```

Assuming the CEs are configured, the PE should be learning IPv4 and IPv6 routes inside of both RED and BLUE VRFs. All of these routes would be advertised to all four RRs. Checking CSR1 and XRv1, we can see all of the routes are present from all PEs (each PE injects 4 routes per VPN, per AFI). For completeness, I show the entire VPNv4 routing table on CSR1, where it clearly depicts 4 routes from each PE within each RD. These are the routes within the two VPNs (recall that RED uses RD 56:100 and BLUE uses RD 56:200). This makes for 8 total VPN routes from each PE. Using the VPNV6 summary command, we can see this as well, which serves as a cursory check.

```
R1#show bgp vpnv4 unicast all | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
```

Route Distinguisher: 56:100

*>i 100.3.0.0/18	56.0.0.9	0	100	0 100 ?
*>i 100.3.64.0/18	56.0.0.9	0	100	0 100 ?
*>i 100.3.128.0/18	56.0.0.9	0	100	0 100 ?
*>i 100.3.192.0/18	56.0.0.9	0	100	0 100 ?
*>i 100.4.0.0/18	56.0.0.10	0	100	0 100 ?
*>i 100.4.64.0/18	56.0.0.10	0	100	0 100 ?
*>i 100.4.128.0/18	56.0.0.10	0	100	0 100 ?
*>i 100.4.192.0/18	56.0.0.10	0	100	0 100 ?
*>i 100.13.0.0/18	56.0.0.5	0	100	0 100 ?
*>i 100.13.64.0/18	56.0.0.5	0	100	0 100 ?
*>i 100.13.128.0/18	56.0.0.5	0	100	0 100 ?
*>i 100.13.192.0/18	56.0.0.5	0	100	0 100 ?
*>i 100.14.0.0/18	56.0.0.6	0	100	0 100 ?
*>i 100.14.64.0/18	56.0.0.6	0	100	0 100 ?
*>i 100.14.128.0/18	56.0.0.6	0	100	0 100 ?
*>i 100.14.192.0/18	56.0.0.6	0	100	0 100 ?

Route Distinguisher: 56:200

*>i 200.3.0.0/18	56.0.0.9	1	100	0 ?
*>i 200.3.64.0/18	56.0.0.9	1	100	0 ?
*>i 200.3.128.0/18	56.0.0.9	1	100	0 ?
*>i 200.3.192.0/18	56.0.0.9	1	100	0 ?
*>i 200.4.0.0/18	56.0.0.10	1	100	0 ?
*>i 200.4.64.0/18	56.0.0.10	1	100	0 ?
*>i 200.4.128.0/18	56.0.0.10	1	100	0 ?
*>i 200.4.192.0/18	56.0.0.10	1	100	0 ?
*>i 200.13.0.0/18	56.0.0.5	1	100	0 ?
*>i 200.13.64.0/18	56.0.0.5	1	100	0 ?
*>i 200.13.128.0/18	56.0.0.5	1	100	0 ?
*>i 200.13.192.0/18	56.0.0.5	1	100	0 ?
*>i 200.14.0.0/18	56.0.0.6	1	100	0 ?
*>i 200.14.64.0/18	56.0.0.6	1	100	0 ?
*>i 200.14.128.0/18	56.0.0.6	1	100	0 ?
*>i 200.14.192.0/18	56.0.0.6	1	100	0 ?

```
R1#show bgp vpnv6 unicast all summary | begin ^Neigh
```

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
56.0.0.5	4	56	1038	1128	117	0	0	02:30:49	8
56.0.0.6	4	56	1025	1057	117	0	0	02:31:05	8
56.0.0.9	4	56	1042	1057	117	0	0	02:30:57	8
56.0.0.10	4	56	1030	1052	117	0	0	02:30:54	8

Checking XRv1, we see an interesting problem. None of the VPN routes are marked as best path. Upon closer inspection, XRv1 claims the VPN next-hop is unreachable. This is because XR expects a non-default route to the BGP next-hop, in addition to the peering restriction discussed earlier. XE did not seem to care (CSR1 and CSR2) that the only route to the VPN next-hop was a default route. XRv1 and XRv2 are ineffective as VPN RRs in the design at present.

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast | begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 56:100					
*i100.3.0.0/18	56.0.0.9	0	100	0	100 ?
*i100.3.64.0/18	56.0.0.9	0	100	0	100 ?
*i100.3.128.0/18	56.0.0.9	0	100	0	100 ?
*i100.3.192.0/18	56.0.0.9	0	100	0	100 ?

[snip]

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast rd 56:100 100.3.0.0/18 | begin 100,
100, (Received from a RR-client)
56.0.0.9 (inaccessible) from 56.0.0.9 (56.0.0.9)
Received Label 9026
Origin incomplete, metric 0, localpref 100, valid, internal, not-in-vrf
Received Path ID 0, Local Path ID 0, version 0
Extended community: RT:56:103
Connector: type: 1, Value:56:100:56.0.0.9
```

Since this issue is not the focus of this lab, we can quickly resolve it by leaking PE loopbacks (not core loopbacks) into IS-IS level-1 areas on CSR5 and CSR9. To do this semi-dynamically, I apply a route tag of 56 to the PE loopbacks. The route-map on CSR5 and CSR9 matches this tag and allows those loopbacks to leak into level-1. This has the added benefit of allowing XRv1 and XRv2 initiate BGP sessions to the PEs as there is a non-default route to reach them.

```
! All PEs
interface Loopback0
isis tag 56

! CSR5 and CSR9
route-map RM_LEAK_LOOP_L1 permit 10
match tag 56

router isis 56
redistribute isis ip level-2 into level-1 route-map RM_LEAK_LOOP_L1
```

This immediately fixes the problem on XRv1 as shown below. We also check to ensure that XRv1 is receiving VPNv6 routes. Like CSR1, it receives exactly 8, as expected.

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast | begin Network
Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 56:100
*>i100.3.0.0/18    56.0.0.9      0      100      0 100 ?
*>i100.3.64.0/18   56.0.0.9      0      100      0 100 ?
*>i100.3.128.0/18  56.0.0.9      0      100      0 100 ?
*>i100.3.192.0/18  56.0.0.9      0      100      0 100 ?

RP/0/0/CPU0:XRv1#show bgp vpnv6 unicast summary | begin ^Neigh
Neighbor      Spk    AS  MsgRcvd  MsgSent  TblVer  InQ  OutQ  Up/Down  St/PfxRcd
56.0.0.5      0      56    991    1006    113     0     0 02:15:01      8
```

56.0.0.6	0	56	971	962	113	0	0	02:14:47	8
56.0.0.9	0	56	996	962	113	0	0	02:14:51	8
56.0.0.10	0	56	988	962	113	0	0	02:14:50	8

With the BGP configuration complete, I quickly test VPN connectivity on a few routers. Although MPLS forwarding is not the focus of this lab, this will ensure BGP was configured properly. We will perform more detailed BGP verification later. Using nested TCL, I can execute 16 pings between each pair of addresses within a given VPN. One example is shown below, and it succeeds.

```
! CSR3
tclsh
foreach x {
100.13.0.1
100.13.64.1
100.13.128.1
100.13.224.1
} { foreach y {
100.3.0.1
100.3.64.1
100.3.128.1
100.3.224.1
} { ping vrf RED $x source $y repeat 3 timeout 1 } }
```

The issue with this design is that the PEs have many copies of the exact same route. CSR10, for example, receives a copy from all four RRs. Upon deeper inspection, we see that the route from CSR1 is always preferred as it has the lowest BGP peer ID (the final tie-breaker). This is irrelevant since all routes are identical, but it is a waste of memory on both the RRs and PEs.

```
R10#show bgp vpnv4 unicast vrf RED | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 56:100 (default for vrf RED)
* i 100.3.0.0/18    56.0.0.9      0      100      0 100 ?
* i                56.0.0.9      0      100      0 100 ?
* i                56.0.0.9      0      100      0 100 ?
*>i               56.0.0.9      0      100      0 100 ?
* i 100.3.64.0/18   56.0.0.9      0      100      0 100 ?
* i                56.0.0.9      0      100      0 100 ?
* i                56.0.0.9      0      100      0 100 ?
*>i               56.0.0.9      0      100      0 100 ?
[snip]
```

```
R10#show bgp vpnv4 unicast vrf RED 100.3.0.0/18 | include from|valid
56.0.0.9 (metric 10) (via default) from 56.0.0.11 (56.0.0.11)
Origin incomplete, metric 0, localpref 100, valid, internal
56.0.0.9 (metric 10) (via default) from 56.0.0.12 (56.0.0.12)
Origin incomplete, metric 0, localpref 100, valid, internal
56.0.0.9 (metric 10) (via default) from 56.0.0.2 (56.0.0.2)
```



```
Origin incomplete, metric 0, localpref 100, valid, internal
56.0.0.9 (metric 10) (via default) from 56.0.0.1 (56.0.0.1)
Origin incomplete, metric 0, localpref 100, valid, internal, best
```

With such a distributed RR infrastructure, it would make sense if different RRs could service different VPNs. This is accomplished by selecting which RTs an RR can retain as discussed earlier. On XE, the feature is known as BGP “RR-groups” which are not to be confused with RR clusters. This is a simple configuration which uses an extended-community list of RTs as a pass/fail filter. A certain RR can be configured to retain only specific RTs that match the filter and reject all others. RRs assume that, by default, all RTs must be retained since any PE may need them. Using this method when there are multiple/redundant RRs in the network is a good way to increase scale and conserve memory on RRs and PEs. On CSR1 and CSR2, I configure opposing policies. CSR1 is allowed to retain only RED RTs while CSR2 is allowed to retain anything except RED RTs. In this design, there is only one non-RED VPN, but if there were more, CSR2 would retain those other RTs as well. This illustrates that both positive and negative logic are supported for the RR-group filter. On XE, the feature is supported for VPNv4, VPNv6, MVPNV4, and MVPNV6. It is not supported for L2VPN VPLS. CSR1 is examined first.

```
! CSR1
ip extcommunity-list expanded EXTCOML_RR_RED permit RT:56:1..$

router bgp 56
 address-family vpnv4
   bgp rr-group EXTCOML_RR_RED
 address-family vpnv6
   bgp rr-group EXTCOML_RR_RED
```

Applying this configuration immediately purges any VPN routes with rejected RTs. CSR1 now has no routes for RD 56:200.

```
R1#show bgp vpnv4 unicast rd 56:200
[no output]
```

If more routes are advertised to the RRs later containing rejected RTs, they are never installed. The debug output shown below looks familiar as the rejection looks identical to any RT rejection. The same output would be present if, for example, a PE failed to explicitly import a RT. I highly the RD 56:200 as well as this identifies the routes as BLUE. Make no mistake; the filter is not matching the RD, but the RTs carried by the routes. These are highlighted in green.

```
R1#debug bgp vpnv4 unicast updates in
BGP updates debugging is on (inbound) for address family: VPNv4 Unicast

BGP(4): 56.0.0.9 rcvd UPDATE w/ attr: nexthop 56.0.0.9, origin ?, localpref
100, metric 1, extended community RT:56:203
BGP(4): 56.0.0.9 rcvd 56:200:200.3.0.0/18, label 9034 -- DENIED due to:
extended community not supported;
```

```
BGP(4): 56.0.0.9 rcvd 56:200:200.3.64.0/18, label 9035 -- DENIED due to:
extended community not supported;
BGP(4): 56.0.0.9 rcvd 56:200:200.3.128.0/18, label 9036 -- DENIED due to:
extended community not supported;
BGP(4): 56.0.0.9 rcvd 56:200:200.3.192.0/18, label 9037 -- DENIED due to:
extended community not supported;
```

Examining VPN BLUE on CSR10, there are only 3 copies learned. CSR1 no longer advertises one as it cannot even retain the routes in the first place. This indicates that the feature is working correctly. CSR2 is the new best-path given the next lower BGP peer ID.

```
R10#show bgp vpnv4 unicast vrf BLUE 200.3.0.0/18 | include from|valid
56.0.0.9 (metric 10) (via default) from 56.0.0.11 (56.0.0.11)
Origin incomplete, metric 1, localpref 100, valid, internal
56.0.0.9 (metric 10) (via default) from 56.0.0.12 (56.0.0.12)
Origin incomplete, metric 1, localpref 100, valid, internal
56.0.0.9 (metric 10) (via default) from 56.0.0.2 (56.0.0.2)
Origin incomplete, metric 1, localpref 100, valid, internal, best
```

Next, we will configure CSR2 to reject RED RTs while permitting anything else. The IOS-regex logic is straightforward and applying the filter is identical as it was on CSR1. After applying the configuration, we quickly verify that CSR2 is not retaining any RED routes.

```
! CSR2
ip extcommunity-list expanded EXTCOML_RR_NOT_RED deny RT:56:1..$
ip extcommunity-list expanded EXTCOML_RR_NOT_RED permit RT:[0-9]+:[0-9]+

router bgp 56
 address-family vpnv4
   bgp rr-group EXTCOML_RR_NOT_RED
 address-family vpnv6
   bgp rr-group EXTCOML_RR_NOT_RED
```

```
R2#show bgp vpnv4 unicast rd 56:100
[no output]
```

On CSR10 within the RED VPN, CSR2 is no longer advertising reachability for RED prefixes. CSR1 remains the best path, but of greater significance is that CSR2 is not advertising any routes.

```
R10#show bgp vpnv4 unicast vrf RED 100.3.0.0/18 | include from|valid
56.0.0.9 (metric 10) (via default) from 56.0.0.11 (56.0.0.11)
Origin incomplete, metric 0, localpref 100, valid, internal
56.0.0.9 (metric 10) (via default) from 56.0.0.12 (56.0.0.12)
Origin incomplete, metric 0, localpref 100, valid, internal
56.0.0.9 (metric 10) (via default) from 56.0.0.1 (56.0.0.1)
Origin incomplete, metric 0, localpref 100, valid, internal, best
```

To further clean up the network, we will configure similar policies on XRv1 and XRv2. XRv1 will team up with CSR1 to service the RED VPN, while XRv2 will team up with CSR2 to service anything not in the RED VPN. Rather than create a custom command for this, XR uses the “retain route-target” syntax. This is often used for inter-AS MPLS option B ASBRs so that they can retain all RTs without having to configure the VRF locally or identify iBGP peers as RR-clients. When a peer is identified as an RR-client, all RTs are retained from those peers by default. XR allows the user to restrict this using RPL, so the feature has value on RRs as well as option B ASBRs. Looking at XRv1 first, we create an RT-set using an IOS-regex to match RED RTs. This is passed into a parameterized RPL for flexibility. Last, the RPL is invoked by VPNv4 and VPNv6 AFIs under BGP.

```
! XRv1
extcommunity-set rt RT_RED
  ios-regex '56:1[01][34]${'
end-set

route-policy RPL_RETAIN_IF_MATCH($RT)
  if extcommunity rt matches-any $RT then
    pass
  endif
end-policy

router bgp 56
  address-family vpnv4 unicast
    retain route-target route-policy RPL_RETAIN_IF_MATCH(RT_RED)
  address-family vpnv6 unicast
    retain route-target route-policy RPL_RETAIN_IF_MATCH(RT_RED)
```

Like XE, any existing routes with unmatched RTs are immediately purged. XRv1 has no need to maintain a separate table for RD 56:200, which saves memory. Checking the BLUE VRF on CSR10 reveals only 2 routes as both CSR1 and XRv1 have stopped servicing the BLUE VPN. CSR10 still learns RED routes from XRv1, also shown below.

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast rd 56:200
[no output]
```

```
R10#show bgp vpnv4 unicast vrf BLUE 200.3.0.0/18 | include from|valid
56.0.0.9 (metric 10) (via default) from 56.0.0.12 (56.0.0.12)
  Origin incomplete, metric 1, localpref 100, valid, internal
56.0.0.9 (metric 10) (via default) from 56.0.0.2 (56.0.0.2)
  Origin incomplete, metric 1, localpref 100, valid, internal, best

R10#show bgp vpnv4 unicast vrf RED 100.3.0.0/18 | include from|valid
56.0.0.9 (metric 10) (via default) from 56.0.0.11 (56.0.0.11)
  Origin incomplete, metric 0, localpref 100, valid, internal
56.0.0.9 (metric 10) (via default) from 56.0.0.12 (56.0.0.12)
  Origin incomplete, metric 0, localpref 100, valid, internal
```

```
56.0.0.9 (metric 10) (via default) from 56.0.0.1 (56.0.0.1)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
```

Last, we configure XRv2 to reject RED routes. RPL helps simplify positive/negative logic by embedding the logic in the conditional construct rather than the list construct. In XE, the permit/deny action was defined in the extended-community list. XRv2 uses the same RT-set as XRv1, but inverts the RPL logic. The RPL rejects the route if it contains any RT matched within the RED list.

```
! XRv2
extcommunity-set rt RT_RED
  ios-regex '56:1[01][34]${'
end-set

route-policy RPL_REJECT_IF_MATCH($RT)
  if extcommunity rt matches-any $RT then
    drop
  else
    pass
  end
end-policy

router bgp 56
  address-family vpnv4 unicast
    retain route-target route-policy RPL_REJECT_IF_MATCH(RT_RED)
  address-family vpnv6 unicast
    retain route-target route-policy RPL_REJECT_IF_MATCH(RT_RED)
```

XRv2 no longer retains any RED routes and has no need to create a table for RD 56:100. CSR10 now only sees 2 RED routes (CSR1 and XRv1) which means CSR2 and XRv2 are no longer servicing RED VPN clients. In this design, we still have HA since there are 2 RRs servicing each VPN. If the PEs were multi-homed BGP PIC could be used for additional fault tolerance. This design reduced the memory requirements on all 4 RRs and all 4 PEs by 50% by load-splitting the VPN routes between available RRs.

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast rd 56:100
[no output]
```

```
R10#show bgp vpnv4 unicast vrf RED 100.3.0.0/18 | include from|valid
56.0.0.9 (metric 10) (via default) from 56.0.0.11 (56.0.0.11)
  Origin incomplete, metric 0, localpref 100, valid, internal
56.0.0.9 (metric 10) (via default) from 56.0.0.1 (56.0.0.1)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
```

To confirm the claim above with respect to RR memory requirements, we check CSR1 and XRv1 for their VPN route counts per client. These numbers were 8 earlier, but now are 4, since each RR is only accepting one VPN route “color”.

```
R1#show bgp vpnv4 unicast all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent   TblVer  InQ  OutQ Up/Down  State/PfxRcd
56.0.0.5      4      56   1384   1476     117    0    0 03:24:26      4
56.0.0.6      4      56   1372   1403     117    0    0 03:24:42      4
56.0.0.9      4      56   1388   1404     117    0    0 03:24:34      4
56.0.0.10     4      56   1379   1398     117    0    0 03:24:30      4
```

```
R1#show bgp vpnv6 unicast all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent   TblVer  InQ  OutQ Up/Down  State/PfxRcd
56.0.0.5      4      56   1385   1477     133    0    0 03:24:31      4
56.0.0.6      4      56   1372   1403     133    0    0 03:24:46      4
56.0.0.9      4      56   1389   1404     133    0    0 03:24:38      4
56.0.0.10     4      56   1379   1398     133    0    0 03:24:35      4
```

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast summary | begin ^Neigh
Neighbor      Spk    AS MsgRcvd MsgSent   TblVer  InQ  OutQ Up/Down  St/PfxRcd
56.0.0.5      0     56   1261   1263     147    0    0 02:57:39      4
56.0.0.6      0     56   1240   1219     147    0    0 02:57:25      4
56.0.0.9      0     56   1265   1219     147    0    0 02:57:29      4
56.0.0.10     0     56   1257   1219     147    0    0 02:57:28      4
```

```
RP/0/0/CPU0:XRv1#show bgp vpnv6 unicast summary | begin ^Neigh
Neighbor      Spk    AS MsgRcvd MsgSent   TblVer  InQ  OutQ Up/Down  St/PfxRcd
56.0.0.5      0     56   1262   1263     129    0    0 02:57:41      4
56.0.0.6      0     56   1241   1219     129    0    0 02:57:28      4
56.0.0.9      0     56   1265   1219     129    0    0 02:57:32      4
56.0.0.10     0     56   1258   1219     129    0    0 02:57:31      4
```

Additional Reading – Reference configurations “bgp-rr-group”

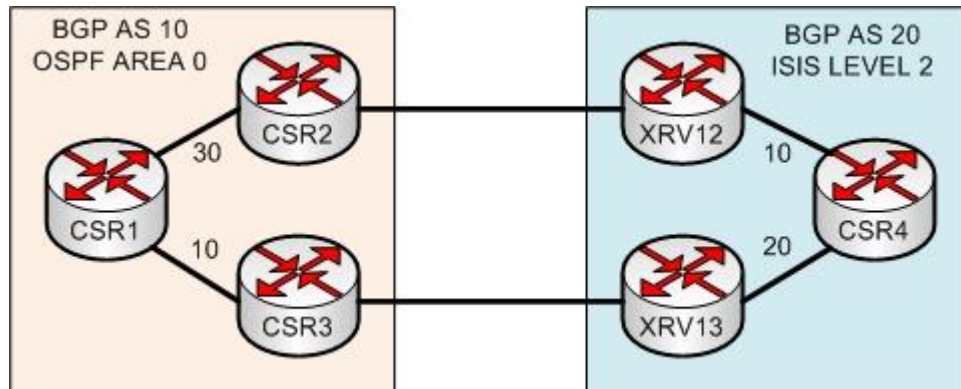
27.4 Accumulated IGP attribute

If you have multiple sub-ASes within a confederation (or you control the IGP metrics for both ASes), you can essentially compare the end to end cost using accumulated IGP (AIGP). The remote AS will write the AIGP value, normally by copying the IGP metric (similar to how MED is populated), then advertise it to the BGP neighbor as a new attribute. This new attribute can be added to the IGP metric to the BGP next-hop to determine the total path cost; this cannot be done with MED or cost-communities, although AIGP can be transported via those methods for legacy devices. AIGP is evaluated immediately following local-preference in the best path selection process (so, it's evaluated very early). Beware that grossly incompatible IGP metrics between the two ASes, confed-external or true eBGP, will break the usefulness of this feature. EIGRP in one AS versus RIP in another will always be bound by the EIGRP metric, since the RIP metrics will contribute comparatively little to the AIGP. As an odd workaround, you could encode AIGP inside cost-community, use the pre-bestpath POI, and apply the transitive option. This would allow one AS to override the policies of another, even for true eBGP.

27.4.1 Basic AIGP

This lab explores the basics of AIGP and how it is exchanged/processed in a BGP network. The network diagram is shown below. There are two ASes pictured with iBGP internally and eBGP between one another. We are only evaluating prefixes from the perspective of CSR1 and CSR4 in these labs. Most of

the community manipulation happens on the ASBRs. The numbers associated with the links are the IGP costs, which are symmetric.



Below I show the configurations for CSR2 and XRV3, but the equivalent AIGP configuration exists on all 4 ASBRs. We need to activate AIGP on all neighbors to pass the attribute. We also have to set the AIGP metric manually, which can be done dynamically based on the IGP cost. This can be applied to prefixes being redistributed into BGP or as an attribute modification within the “network” statement. The RPL construct in XR is very similar. Of note, the IGP costs are constantly adjusted in this lab as well.

```

! CSR2
route-map AIGP_METRIC permit 10
  set aigp-metric igp-metric

router bgp 10
  no bgp default ipv4-unicast
  address-family ipv4
    network 100.1.1.1 mask 255.255.255.255 route-map AIGP_METRIC
  neighbor 1.1.1.1 aigp
  neighbor 22.0.0.12 aigp

! XRV12
route-policy SET_AIGP
  set aigp-metric igp-cost

router bgp 20
  address-family ipv4 unicast
    network 100.4.4.4/32 route-policy SET_AIGP

  neighbor 33.0.0.3
    address-family ipv4 unicast
    aigp
  neighbor 4.4.4.4
    address-family ipv4 unicast
    aigp

```

Before looking at any routing information, we will verify the configuration between XRv13 and CSR4 as a demonstration. AIGP is a neighbor-specific feature but is not a negotiated capability nor a community value. Rather, it is like an additional BGP path-attribute, much like local-preference or MED. These commands confirm that AIGP was configured correctly.

```
CSR4#show bgp ipv4 unicast neighbors 12.12.12.12 | begin For address family
For address family: IPv4 Unicast
  Session: 12.12.12.12
  BGP table version 1, neighbor version 1/0
  Output queue size : 0
  Index 1, Advertise bit 0
  AIGP is enabled
  1 update-group member
```

```
RP/0/0/CPU0:XRv2#show bgp ipv4 unicast neighbors 4.4.4.4 | begin For Address
Family
For Address Family: IPv4 Unicast
  BGP neighbor version 2
  Update group: 0.2 Filter-group: 0.1 No Refresh request being processed
  Route refresh request: received 0, sent 0
  0 accepted prefixes, 0 are bestpaths
  Cumulative no. of prefixes denied: 0.
  Prefix advertised 0, suppressed 0, withdrawn 0
  Maximum prefixes allowed 1048576
  Threshold for warning message 75%, restart interval 0 min
  AIGP is enabled
  An EoR was received during read-only mode
  [snip]
```

A quick debug on CSR2 reveals that CSR4's loopback received from XRv12 now carries a new AIGP-metric field, which is currently set to XRv12 IGP metric to 100.4.4.4/32.

```
CSR2#debug bgp ipv4 unicast update
BGP updates debugging is on for address family: IPv4 Unicast
BGP(0): 22.0.0.12 rcvd UPDATE w/ attr: nexthop 22.0.0.12, origin i, metric
10, aigp-metric 10, merged path 20, AS_PATH
BGP(0): 22.0.0.12 rcvd 100.4.4.4/32
```

We will look at CSR4's route to CSR1's loopback to start. The AIGP metric is better via XRv13 (11), but AIGP + local IGP metric (42) is worse than the sum of the same values carried by the XRv12 route (41). This is what makes AIGP special; unlike MED, it is combined with the IGP cost to form a pseudo end-to-end metric across AS boundaries.

```
CSR4#show bgp ipv4 unicast 100.1.1.1/32
BGP routing table entry for 100.1.1.1/32, version 12
Paths: (2 available, best #2, table default)
  Not advertised to any peer
```

```

Refresh Epoch 1
10
  13.13.13.13 (metric 31) from 13.13.13.13 (13.13.13.13)
    Origin IGP, aigp-metric 11, metric 11, localpref 100, valid, internal
    rx pathid: 0, tx pathid: 0
Refresh Epoch 1
10
  12.12.12.12 (metric 10) from 12.12.12.12 (12.12.12.12)
    Origin IGP, aigp-metric 31, metric 31, localpref 100, valid, internal,
best
    rx pathid: 0, tx pathid: 0x0

```

Next, we reduce CSR4's cost to XRv13 to 28. The AIGP metric is better via XRv13 (11), even though the IGP metric (29) is worse. The sum is still less for the best route (40 vs. 41). Notice that even though AIGP is compared early in the path selection process, this comparison is the sum of the AIGP and IGP costs, not just comparing the AIGP cost, as proven below.

```

CSR4#show bgp ipv4 unicast 100.1.1.1/32
BGP routing table entry for 100.1.1.1/32, version 11
Paths: (2 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  10
    13.13.13.13 (metric 29) from 13.13.13.13 (13.13.13.13)
      Origin IGP, aigp-metric 11, metric 11, localpref 100, valid, internal,
best
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  10
    12.12.12.12 (metric 10) from 12.12.12.12 (12.12.12.12)
      Origin IGP, aigp-metric 31, metric 31, localpref 100, valid, internal
      rx pathid: 0, tx pathid: 0

```

Below is an example where AIGP + local IGP is the same in both cases (41). Best-path declares a tie and continues with the selection process. In most cases, the AIGP metric was copied from the IGP metric, just like MED. Therefore, MED determines that XRv13 is the better route, and generally speaking, the lower remote IGP cost will break the tie.

```

CSR4#show bgp ipv4 unicast 100.1.1.1/32
BGP routing table entry for 100.1.1.1/32, version 13
Paths: (2 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  10
    13.13.13.13 (metric 30) from 13.13.13.13 (13.13.13.13)
      Origin IGP, aigp-metric 11, metric 11, localpref 100, valid, internal,
best

```



```

    rx pathid: 0, tx pathid: 0x0
Refresh Epoch 1
10
12.12.12.12 (metric 10) from 12.12.12.12 (12.12.12.12)
    Origin IGP, aigp-metric 31, metric 31, localpref 100, valid, internal
    rx pathid: 0, tx pathid: 0

```

If AIGP is not present, AIGP automatically wins (assuming weight and local preference are equal), even at maximum value. This is a little odd as I personally like the behavior of the cost-community where the lack of a value assumes the midpoint between minimum and maximum.

```

CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 31
Paths: (2 available, best #1, table default)
    Not advertised to any peer
    Refresh Epoch 6
    20
    2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
        Origin IGP, aigp-metric 4294967295, metric 10, localpref 100, valid,
internal, best
        rx pathid: 0, tx pathid: 0x0
    Refresh Epoch 6
    20
    3.3.3.3 (metric 11) from 3.3.3.3 (3.3.3.3)
        Origin IGP, metric 20, localpref 100, valid, internal
        rx pathid: 0, tx pathid: 0

```

Just to prove where AIGP sits in the best path, local preference has been increased for the route via CSR3, and now it is preferred since AIGP is not evaluated. By extension, modifying weight would also work (no example shown).

```

CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 10
Paths: (2 available, best #1, table default)
    Not advertised to any peer
    Refresh Epoch 2
    20
    3.3.3.3 (metric 11) from 3.3.3.3 (3.3.3.3)
        Origin IGP, metric 20, localpref 200, valid, internal, best
        rx pathid: 0, tx pathid: 0x0
    Refresh Epoch 1
    20
    2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
        Origin IGP, aigp-metric 4294967295, metric 10, localpref 100, valid,
internal
        rx pathid: 0, tx pathid: 0

```

Also, let's increase the AS path length of the CSR2 route. The CSR2 route stills wins because AIGP is processed first.

```
CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 12
Paths: (2 available, best #2, table default)
  Not advertised to any peer
  Refresh Epoch 3
  20
    3.3.3.3 (metric 11) from 3.3.3.3 (3.3.3.3)
      Origin IGP, metric 20, localpref 100, valid, internal
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  20 20 20 20
    2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
      Origin IGP, aigp-metric 4294967295, metric 10, localpref 100, valid,
internal, best
      rx pathid: 0, tx pathid: 0x0
```

Below is another example where a lower AIGP does not necessary imply best-path. AIGP is just a component of the metric as the local IGP metric still counts, assuming AIGP attribute is present on all contending routes. The best path has a total metric of 31 while the path with lower AIGP has a total metric of 41.

```
CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 29
Paths: (2 available, best #2, table default)
  Not advertised to any peer
  Refresh Epoch 6
  20
    2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
      Origin IGP, aigp-metric 10, metric 10, localpref 100, valid, internal
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 6
  20
    3.3.3.3 (metric 11) from 3.3.3.3 (3.3.3.3)
      Origin IGP, aigp-metric 20, metric 20, localpref 100, valid, internal,
best
      rx pathid: 0, tx pathid: 0x0
```

Last, I show another example of an AIGP + local IGP metric tie (31). The route through CSR2 is selected due to lower MED.

```
CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 33
Paths: (2 available, best #1, table default)
  Not advertised to any peer
```

```

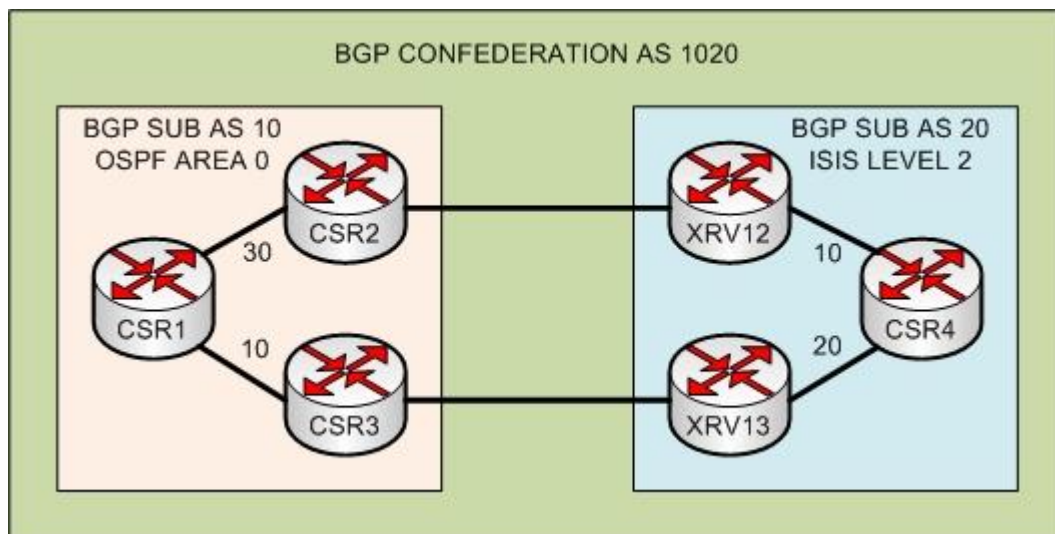
Refresh Epoch 7
20
  2.2.2.2 (metric 21) from 2.2.2.2 (2.2.2.2)
    Origin IGP, aigp-metric 10, metric 10, localpref 100, valid, internal,
best
    rx pathid: 0, tx pathid: 0x0
Refresh Epoch 7
20
  3.3.3.3 (metric 11) from 3.3.3.3 (3.3.3.3)
    Origin IGP, aigp-metric 20, metric 20, localpref 100, valid, internal
    rx pathid: 0, tx pathid: 0

```

Additional Reading – Reference configurations "aigp-basic"

27.4.2 AIGP with cost-communities and BGP confederations

AIGP can also be transported inside the cost-community. Note: As seen in the cost-community section, this does not work for true eBGP peers, only confederations and iBGP. The network has been modified to place AS 10 and AS 20 in a confederation together. The exception is using the "transitive" option which is discussed next, but oddly, there is no way to pass classic cost-communities across eBGP boundaries. It appears to be an AIGP-only extension. The network diagram is below; it is the same design with minor BGP modifications to construct a confederation between AS 10 and AS 20.



Although much less effective than real AIGP, you can encode the AIGP metric in a cost-community. If used with the pre-bestpath POI, it effectively trumps all local policies set by the receiving router and is even more powerful than ordinary AIGP. Below are the outputs from CSR1 and CSR4, respectively. Using it with igp-cost POI is probably most realistic as it comes after the IGP metric to BGP next-hop, but the two are not added together. In XR, a special command is needed to exchange standard/extended communities with eBGP peers; within iBGP, communities are automatically exchanged. In IOS, community exchange must be specified per neighbor, per type, regardless of the remote AS number.

The configuration snippet from XRv13 (the other three ASBRs have similar configurations to correspond with the show commands following this) is below.

```
! XRv13
router bgp 20
  af-group AF_IPV4 address-family ipv4 unicast
    aigp send cost-community 44 poi pre-bestpath
    next-hop-self
  neighbor 33.0.0.3
    address-family ipv4 unicast
      use af-group AF_IPV4
      send-extended-community-ebgp
```

```
CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 33
Paths: (2 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 6
  (20)
    2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, confed-internal, best
      Extended Community: Cost:pre-bestpath:44:10
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 5
  (20)
    3.3.3.3 (metric 11) from 3.3.3.3 (3.3.3.3)
      Origin IGP, metric 20, localpref 100, valid, confed-internal
      Extended Community: Cost:pre-bestpath:44:20
      rx pathid: 0, tx pathid: 0
```

```
CSR4#show bgp ipv4 unicast 100.1.1.1/32
BGP routing table entry for 100.1.1.1/32, version 63
Paths: (2 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  (10)
    13.13.13.13 (metric 20) from 13.13.13.13 (13.13.13.13)
      Origin IGP, metric 11, localpref 100, valid, internal, best
      Extended Community: Cost:pre-bestpath:99:11
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  (10)
    12.12.12.12 (metric 10) from 12.12.12.12 (12.12.12.12)
      Origin IGP, metric 31, localpref 100, valid, internal
      Extended Community: Cost:pre-bestpath:99:31
      rx pathid: 0, tx pathid: 0
```

If you tell CSR2 and CSR3 to send AIGP inside cost-community towards CSR1, it will add a second cost-community with metric 0. CSR1 sees both communities. The communities are compared in sequence which is discussed in detail in the cost-community section. Essentially, the lowest community IDs are compared together first. Since there is no tie, the higher community IDs are never evaluated.

```
CSR1#show bgp ipv4 unicast 100.4.4.4/32
Paths: (2 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 6
  (20)
    2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, confed-internal, best
      Extended Community: Cost:pre-bestpath:44:10 Cost:pre-bestpath:99:0
      rx pathid: 0, tx pathid: 0x0
      Refresh Epoch 5
  (20)
    3.3.3.3 (metric 11) from 3.3.3.3 (3.3.3.3)
      Origin IGP, metric 20, localpref 100, valid, confed-internal
      Extended Community: Cost:pre-bestpath:44:20 Cost:pre-bestpath:99:0
      rx pathid: 0, tx pathid: 0
```

I am assuming this is because the first non-BGP route that the recursive lookup finds is a connected route. The IGP cost to the BGP next-hop, for eBGP peers in this network, is zero. CSR2's perspective is shown below.

```
CSR2#show ip route 100.4.4.4
Routing entry for 100.4.4.4/32
  Known via "bgp 10", distance 200, metric 10
  Tag 20, type internal
  Last update from 22.0.0.12 00:09:43 ago
  Routing Descriptor Blocks:
  * 22.0.0.12, from 22.0.0.12, 00:09:43 ago
    Route metric is 10, traffic share count is 1
    AS Hops 0
    Route tag 20
    MPLS label: none
```

```
CSR2#show ip route 22.0.0.12
Routing entry for 22.0.0.0/8
  Known via "connected", distance 0, metric 0 (connected, via interface)
  Routing Descriptor Blocks:
  * directly connected, via GigabitEthernet2.522
    Route metric is 0, traffic share count is 1
```

AIGP is also available inside the cost-community with the transitive option. Using the transitive keyword, we can transport the cost-community across true eBGP boundaries (configuration not shown). The

confederation has been removed and AS 10 and AS 20 are separate ASes now. All the same cost-community rules apply. I have not found a way to do this with regular cost-communities.

```
! XRv12
router bgp 20
  af-group AF_IPV4 address-family ipv4 unicast
    aigp send cost-community 44 poi pre-bestpath transitive
    next-hop-self

CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 6
Paths: (2 available, best #2, table default)
  Not advertised to any peer
  Refresh Epoch 1
  20
    3.3.3.3 (metric 11) from 3.3.3.3 (3.3.3.3)
      Origin IGP, metric 20, localpref 100, valid, internal
      Extended Community: Cost(transitive):pre-bestpath:44:20
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  20
    2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, internal, best
      Extended Community: Cost(transitive):pre-bestpath:44:10
      rx pathid: 0, tx pathid: 0x0
```

Additional Reading – Reference configurations "aigp-costcom"

27.5 Cost-Community / Point Of Insertion (POI)

Somewhat similar to AIGP, the cost-community was developed as a mechanism to carry cost information around BGP networks. It has many different purposes which are described in this chapter, along with different points of insertion (POI) that determine when this community is evaluated. This feature is quite old, so newer features like accumulated IGP (AIGP) can use it to transport AIGP metrics for older routers. AIGP is more advanced and requires newer code versions (15.4(2)S for IOS, 3.12 for XE, 4.0.0 for XR).

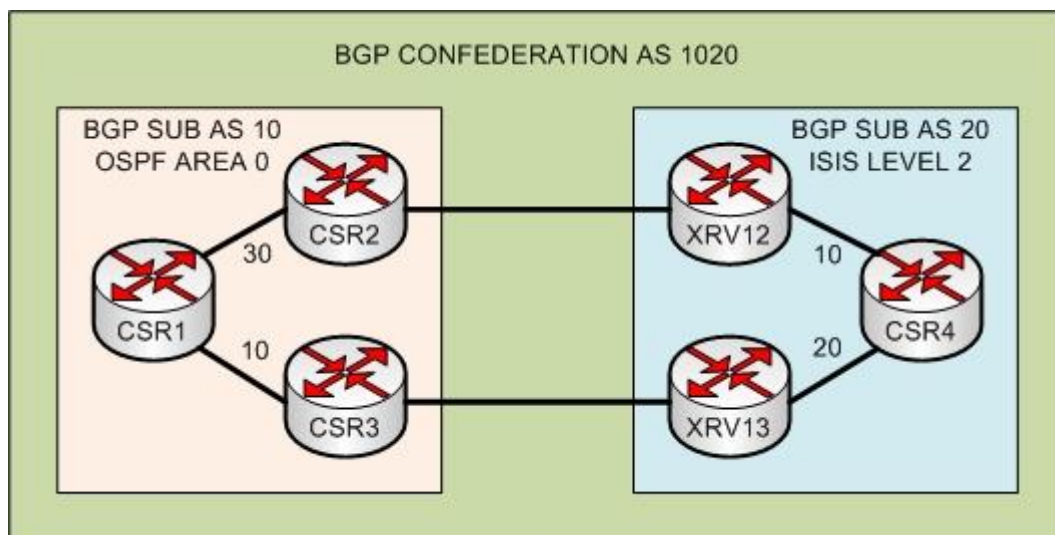
Pre-bestpath POI was designed for EIGRP MPLS L3VPN architectures where backdoor links exist. Normally BGP would prefer the locally sourced routes over the BGP learned ones due to the automatic 32,768 weight value applied. This would lead to a looping condition or suboptimal routing. Using the cost-community fixes it because if a prefix loops in the control-plane back to a PE, the cost-community would have increased across the customer network and will be less than whatever the original VPNv4 prefix cost-community carried from the source PE. This is why cost-community is valuable for overriding the weight attribute.

Another use for pre-bestpath is for extreme "hot potato" routing, where getting the traffic out of the AS as quickly as possible is desired. Because it pre-empts even the weight attribute, it can be used to change path selection within an AS or between confederations (not possible with true eBGP peers). There is one exception where the cost-community is fully transitive, and that is when the AIGP metric is transported in the cost-community with the "transitive" keyword. I have not found a way to establish transitivity without the AIGP application. The ordinary IOS route-map and XR RPL does not have an option and the attribute is simply not exchanged.

IGP-cost is the default POI for cost-community with the exception of EIGRP MPLS L3VPN, which uses pre-bestpath (and is fully automatic; requires no configuration). The rules for the cost-community processing are the same (many are illustrated below) but the IGP-cost POI is evaluated after the IGP metric to the BGP next-hop criteria. It is effectively a new tie-breaker to occur before the router-ID comparison which applies to iBGP, confed-internal, and confed-external.

Hierarchical cost-community selection is achievable by affixing multiple cost-communities to a specific BGP prefix. It should be supported for all AFI/SAFI on all platforms that understand the cost-community in general. To protect your network from an accidental hijack, you can use "bgp bestpath cost-community ignore". BGP simply ignores the values and proceeds as if they never existed.

This feature is tested using a similar topology as AIGP with two BGP sub-ASes wrapped into a confederation. Because the cost-community is not transitive, we cannot test it with true eBGP.



These examples will not show the configuration changes each time. For all of them, the cost-community application on XRv12 and/or XRv13 was changed as the text describes. Here are the snippets from XRv12, but the actual community values vary during the experiments. Similar configurations are applied to XRv13. Additional communities and component routes are added to the configurations over time, but the general construct is shown below. The "COST_COMM_SET" is changed constantly in these

experiments, but the other constructs generally remain the same. The basic network configurations are also not shown.

```
! XRv12
extcommunity-set cost COST_COMM_SET
  IGP:21:100
end-set

route-policy COST_COMM
  set extcommunity cost COST_COMM_SET
end-policy

router bgp 20
  address-family ipv4 unicast
    network 100.4.4.4/32 route-policy COST_COMM
```

Before completing the confederation configuration, I quickly prove that the cost-community is not transitive. Below I attempt true eBGP between AS 10 and AS 20. Notice how CSR2 sets the cost-community locally and "debug bgp updates" verifies it was sent to XRv12.

```
CSR2#show bgp ipv4 unicast 100.1.1.1/32
BGP routing table entry for 100.1.1.1/32, version 2
Paths: (1 available, best #1, table default)
  Advertised to update-groups:
    10          11
  Refresh Epoch 1
  Local
    12.0.0.1 from 0.0.0.0 (2.2.2.2)
      Origin IGP, metric 31, localpref 100, weight 32768, valid, sourced,
local, best
      Extended Community: Cost:pre-bestpath:22:30
      rx pathid: 0, tx pathid: 0x0

CSR2#debug bgp ipv4 unicast update
BGP(0): (base) 22.0.0.12 send UPDATE (format) 100.1.1.1/32, next 22.0.0.2,
metric 31, path Local, extended community Cost:pre-bestpath:22:30
```

Despite receiving the update successfully, XRv12 never honors that value it. The same was true in the other direction, despite extended communities being enabled between the neighbors.

```
XRv12#show bgp ipv4 unicast 100.1.1.1/32 | begin Paths
Paths: (1 available, best #1)
  Advertised to peers (in unique update groups):
    4.4.4.4
  Path #1: Received by speaker 0
  Advertised to peers (in unique update groups):
    4.4.4.4
```


10

```
22.0.0.2 from 22.0.0.2 (2.2.2.2)
  Origin IGP, metric 31, localpref 100, valid, external, best, group-best
  Received Path ID 0, Local Path ID 1, version 23
  Origin-AS validity: not-found
```

When AS 10 and AS 20 are placed in a confederation together (with appropriate next-hop changes), the cost-community immediately appears. Cisco clearly states eBGP is not supported as the cost-community is not transitive. Notice that XR makes no attempt to decode the “pre-bestpath” code of 128. This number is specific to the pre-bestpath POI.

```
XRv12#show bgp ipv4 unicast 100.1.1.1/32 | begin Paths
Paths: (1 available, best #1)
  Advertised to peers (in unique update groups):
    4.4.4.4
  Path #1: Received by speaker 0
  Advertised to peers (in unique update groups):
    4.4.4.4
  (10)
    22.0.0.2 from 22.0.0.2 (2.2.2.2)
      Origin IGP, metric 31, localpref 100, valid, confed-external, best,
group-best
      Received Path ID 0, Local Path ID 1, version 27
      Extended community: COST:128:22:30
```

With respect to the community ID, the lower ID wins, even if cost is worse. This assumes that each prefix only has one cost-community with the same POI. Below, I demonstrate that 22:10 is better than 33:5 due to having a lower community ID.

```
CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 46
Paths: (2 available, best #2, table default)
  Not advertised to any peer
  Refresh Epoch 3
  (20)
    3.3.3.3 (metric 11) from 3.3.3.3 (3.3.3.3)
      Origin IGP, metric 20, localpref 100, valid, confed-internal
      Extended Community: Cost:pre-bestpath:33:5
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 5
  (20)
    2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, confed-internal, best
      Extended Community: Cost:pre-bestpath:22:10
      rx pathid: 0, tx pathid: 0x0
```

When the IDs match, a tie is declared, and the cost value is compared. The lower cost always wins in this case, as expected. In short, it is an ordered comparison of ID first, cost value second.

```
CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 47
Paths: (2 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 3
  (20)
    3.3.3.3 (metric 11) from 3.3.3.3 (3.3.3.3)
      Origin IGP, metric 20, localpref 100, valid, confed-internal, best
      Extended Community: Cost:pre-bestpath:22:5
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 5
  (20)
    2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, confed-internal
      Extended Community: Cost:pre-bestpath:22:10
      rx pathid: 0, tx pathid: 0
```

When there are multiple cost communities, the lowest IDs are compared first, then incrementally so. In this example, the loser route has an additional cost-community with very low cost, but it was never considered since the ID 22 cost-community was already defeated. So, even though the winner route doesn't have a second cost-community for comparison, it doesn't matter.

```
CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 47
Paths: (2 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 3
  (20)
    3.3.3.3 (metric 11) from 3.3.3.3 (3.3.3.3)
      Origin IGP, metric 20, localpref 100, valid, confed-internal, best
      Extended Community: Cost:pre-bestpath:22:5
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 5
  (20)
    2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, confed-internal
      Extended Community: Cost:pre-bestpath:22:10 Cost:pre-bestpath:33:1
      rx pathid: 0, tx pathid: 0
```

Just to prove it, here is an example where both paths have two cost communities but only the first was compared. The first route still wins as the second cost-community was never compared.

```
CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 48
```

```

Paths: (2 available, best #1, table default)
Not advertised to any peer
Refresh Epoch 3
(20)
  3.3.3.3 (metric 11) from 3.3.3.3 (3.3.3.3)
    Origin IGP, metric 20, localpref 100, valid, confed-internal, best
    Extended Community: Cost:pre-bestpath:22:5 Cost:pre-bestpath:33:33
    rx pathid: 0, tx pathid: 0x0
Refresh Epoch 5
(20)
  2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
    Origin IGP, metric 10, localpref 100, valid, confed-internal
    Extended Community: Cost:pre-bestpath:22:10 Cost:pre-bestpath:33:1
    rx pathid: 0, tx pathid: 0

```

When the first cost communities are tied, only then do the following cost communities act as tiebreakers. Now the route via CSR3 is the best based on the next-lowest cost-community ID / cost pair.

```

CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 49
Paths: (2 available, best #2, table default)
Not advertised to any peer
Refresh Epoch 3
(20)
  3.3.3.3 (metric 11) from 3.3.3.3 (3.3.3.3)
    Origin IGP, metric 20, localpref 100, valid, confed-internal
    Extended Community: Cost:pre-bestpath:22:10 Cost:pre-bestpath:33:33
    rx pathid: 0, tx pathid: 0
Refresh Epoch 5
(20)
  2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
    Origin IGP, metric 10, localpref 100, valid, confed-internal, best
    Extended Community: Cost:pre-bestpath:22:10 Cost:pre-bestpath:33:1
    rx pathid: 0, tx pathid: 0x0

```

Paths without the cost-community have value set to $2^{32} / 2$ or 21^{31} (2147483647). This is considered the "default" value. The feature makes the assumption that if you don't specify anything, the middle value is used for comparison. This is in contrast to AIGP which automatically wins if it is present, even at its maximum value. Path through CSR2 wins with the value set to default – 1, but just barely so.

```

CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 52
Paths: (2 available, best #2, table default)
Not advertised to any peer
Refresh Epoch 3
(20)
  3.3.3.3 (metric 11) from 3.3.3.3 (3.3.3.3)
    Origin IGP, metric 20, localpref 100, valid, confed-internal

```

```

    rx pathid: 0, tx pathid: 0
Refresh Epoch 5
(20)
2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
  Origin IGP, metric 10, localpref 100, valid, confed-internal, best
  Extended Community: Cost:pre-bestpath:22:2147483646 (default-1)
  rx pathid: 0, tx pathid: 0x0

```

If we increase the cost value to default + 1, CSR3 becomes the best path despite not carrying the community at all.

```

CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 53
Paths: (2 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 3
  (20)
    3.3.3.3 (metric 11) from 3.3.3.3 (3.3.3.3)
      Origin IGP, metric 20, localpref 100, valid, confed-internal, best
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 5
  (20)
    2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, confed-internal
      Extended Community: Cost:pre-bestpath:22:2147483648 (default+1)
      rx pathid: 0, tx pathid: 0

```

When one prefix has a default cost-community (explicit default) and the other has no community at all (implicit default0, the BGP best-path selection process simply continues as this evaluation criterion ties. Below is an example; CSR3 wins in a tie due to lower IGP metric to BGP next hop. You may have expected CSR2 to win based on lower MED, but due to the BGP confederation, the MED is ignored in this case. You can use "bgp bestpath med confed" to select CSR2 over CSR3 in this case, but this is not the default behavior. We can tell the peers are in a confederation based on the parenthetical AS path.

```

CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 53
Paths: (2 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 3
  (20)
    3.3.3.3 (metric 11) from 3.3.3.3 (3.3.3.3)
      Origin IGP, metric 20, localpref 100, valid, confed-internal, best
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 5
  (20)
    2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, confed-internal

```

```
Extended Community: Cost:pre-bestpath:22:2147483647 (default)
rx pathid: 0, tx pathid: 0
```

So far we have only looked at pre-bestpath, which is a very extreme way to change the path selection before any other attributes are considered, even weight. Below shows an attempt to override pre-bestpath behavior using weight; it fails. CSR2 is still the bestpath.

```
CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 6
Paths: (2 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 2
  (20)
    2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, confed-internal, best
      Extended Community: Cost:pre-bestpath:22:214
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 2
  (20)
    3.3.3.3 (metric 11) from 3.3.3.3 (3.3.3.3)
      Origin IGP, metric 20, localpref 100, weight 99, valid, confed-internal
      rx pathid: 0, tx pathid: 0
```

When we change the cost-community back to the default value to create a tie, weight takes effect and wins. I also changed the IGP metric to BGP next-hop for the CSR3 route to 1000 in order to demonstrate that the CSR3 route did not win due to that value, as it did earlier. When the cost-community is compared against a route with no cost-community at the same POI, the community ID means nothing.

```
CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 7
Paths: (2 available, best #2, table default)
  Not advertised to any peer
  Refresh Epoch 2
  (20)
    2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, confed-internal
      Extended Community: Cost:pre-bestpath:22:2147483647 (default)
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 2
  (20)
    3.3.3.3 (metric 1000) from 3.3.3.3 (3.3.3.3)
      Origin IGP, metric 20, localpref 100, weight 99, valid, confed-internal, best
      rx pathid: 0, tx pathid: 0x0
```

Next, we evaluate the IGP POI. This considers the IGP cost-community just after the IGP metric to BGP next-hop comparison. It's essentially a tie-breaker for routes before comparing the oldest route

(external) or router ID (confed-internal, confed-external, or internal). The only way eBGP routes would carry this attribute is by way of the AIGP inside cost-community with transitive option, so realistically, it is preempting the lowest RID (or originator ID with route-reflector) comparisons. As a quick comparison, when the type of cost-community varies, they are not compared directly. CSR3 wins despite having a larger "cost value" but it was considered pre-bestpath against CSR2's pre-bestpath implicit-default value of 2,147,483,647.

```
CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 11
Paths: (2 available, best #2, table default)
  Not advertised to any peer
  Refresh Epoch 5
  (20)
    2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, confed-internal
      Extended Community: Cost:igp:22:12
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 5
  (20)
    3.3.3.3 (metric 31) from 3.3.3.3 (3.3.3.3)
      Origin IGP, metric 20, localpref 100, valid, confed-internal, best
      Extended Community: Cost:pre-bestpath:22:13
      rx pathid: 0, tx pathid: 0x0
```

When the two are the same, they are considered only when the IGP metric to BGP next-hop ties. CSR2 wins in a tie due to lower RID (again, because MED is not compared within the confederation).

```
CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 12
Paths: (2 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 5
  (20)
    2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, confed-internal, best
      Extended Community: Cost:igp:22:12
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 5
  (20)
    3.3.3.3 (metric 31) from 3.3.3.3 (3.3.3.3)
      Origin IGP, metric 20, localpref 100, valid, confed-internal
      Extended Community: Cost:igp:22:12
      rx pathid: 0, tx pathid: 0
```

If we adjust the cost to a higher number for CSR2's prefix, CSR3 wins due to a lower cost-community value. The community ID remains 22 so that the costs can be compared directly.

```

CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 13
Paths: (2 available, best #2, table default)
  Not advertised to any peer
  Refresh Epoch 5
  (20)
    2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, confed-internal
      Extended Community: Cost:igp:22:999
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 5
  (20)
    3.3.3.3 (metric 31) from 3.3.3.3 (3.3.3.3)
      Origin IGP, metric 20, localpref 100, valid, confed-internal, best
      Extended Community: Cost:igp:22:12
      rx pathid: 0, tx pathid: 0x0

```

Just like pre-bestpath, the lack of the IGP-cost cost-community compares against the default value. CSR3 wins because CSR2's value is 2,147,483,647 (default).

```

CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 13
Paths: (2 available, best #2, table default)
  Not advertised to any peer
  Refresh Epoch 5
  (20)
    2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, confed-internal
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 5
  (20)
    3.3.3.3 (metric 31) from 3.3.3.3 (3.3.3.3)
      Origin IGP, metric 20, localpref 100, valid, confed-internal, best
      Extended Community: Cost:igp:22:12
      rx pathid: 0, tx pathid: 0x0

```

As expected, CSR2 wins in a tie due to lower RID when CSR3 has the default cost-community metric. This is slightly different than the earlier example to prove the behavior of the default metric when using IGP cost POI.

```

CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 14
Paths: (2 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 5
  (20)
    2.2.2.2 (metric 31) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, confed-internal, best

```

```

    rx pathid: 0, tx pathid: 0x0
Refresh Epoch 5
(20)
3.3.3.3 (metric 31) from 3.3.3.3 (3.3.3.3)
    Origin IGP, metric 20, localpref 100, valid, confed-internal
    Extended Community: Cost:igp:22:2147483647 (default)
    rx pathid: 0, tx pathid: 0

```

The next example proves that the IGP cost comparison happens after the ordinary IGP metric to BGP next-hop. The cost to CSR2 has been reduced but CSR3 has a better cost-community. CSR2 is the best path, though; the IGP cost cost-community is only compared when the IGP metric to BGP next-hop is a tie. It can be viewed as the “first tie breaker” ahead of the BGP RID. This application of this POI was designed to make a less arbitrary best-path decision by assigning administrator-defined values that take effect when the tie-breakers begin.

```

CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 16
Paths: (2 available, best #1, table default)
    Not advertised to any peer
Refresh Epoch 5
(20)
2.2.2.2 (metric 30) from 2.2.2.2 (2.2.2.2)
    Origin IGP, metric 10, localpref 100, valid, confed-internal, best
    rx pathid: 0, tx pathid: 0x0
Refresh Epoch 5
(20)
3.3.3.3 (metric 31) from 3.3.3.3 (3.3.3.3)
    Origin IGP, metric 20, localpref 100, valid, confed-internal
    Extended Community: Cost:igp:22:21
    rx pathid: 0, tx pathid: 0

```

The cost-community also interacts with BGP route aggregation. The aggregate route selects the highest cost-community value, but only if the IDs match AND the "as-set" flag is used on the aggregate. If "as-set" is not applied, no communities are carried over. The first two outputs show two component /32 routes with varying cost-communities. The aggregate /24 route uses the larger one.

```

CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 18
Paths: (1 available, best #1, table default)
    Not advertised to any peer
Refresh Epoch 6
(20)
3.3.3.3 (metric 31) from 3.3.3.3 (3.3.3.3)
    Origin IGP, metric 20, localpref 100, valid, confed-internal, best
    Extended Community: Cost:igp:22:300
    rx pathid: 0, tx pathid: 0x0

```



```

CSR1#show bgp ipv4 unicast 100.4.4.44/32
BGP routing table entry for 100.4.4.44/32, version 19
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 6
  (20)
    2.2.2.2 (metric 30) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, confed-internal, best
      Extended Community: Cost:igp:22:100
      rx pathid: 0, tx pathid: 0x0

CSR1#show bgp ipv4 unicast 100.4.4.0/24
BGP routing table entry for 100.4.4.0/24, version 21
Paths: (1 available, best #1, table default)
  Advertised to update-groups:
    5
  Refresh Epoch 1
  (20), (aggregated by 1020 100.1.1.1)
    0.0.0.0 from 0.0.0.0 (100.1.1.1)
      Origin IGP, localpref 100, weight 32768, valid, aggregated, local, best
      Extended Community: Cost:igp:22:300
      rx pathid: 0, tx pathid: 0x0

```

If one or more component routes don't have a cost-community, the largest explicit value is applied. The documentation says the default value is applied, but I did not observe this. The route via CSR2 does not have any cost-community and the aggregate selected the value from CSR3's route.

```

CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 3
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  (20)
    3.3.3.3 (metric 31) from 3.3.3.3 (3.3.3.3)
      Origin IGP, metric 20, localpref 100, valid, confed-internal, best
      Extended Community: Cost:igp:22:300
      rx pathid: 0, tx pathid: 0x0

CSR1#show bgp ipv4 unicast 100.4.4.44/32
BGP routing table entry for 100.4.4.44/32, version 4
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  (20)
    2.2.2.2 (metric 30) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, confed-internal, best
      rx pathid: 0, tx pathid: 0x0

```

```

CSR1#show bgp ipv4 unicast 100.4.4.0/24
BGP routing table entry for 100.4.4.0/24, version 5
Paths: (1 available, best #1, table default)
  Advertised to update-groups:
    6
  Refresh Epoch 1
  (20), (aggregated by 1020 100.1.1.1)
    0.0.0.0 from 0.0.0.0 (100.1.1.1)
      Origin IGP, localpref 100, weight 32768, valid, aggregated, local, best
      Extended Community: Cost:igp:22:300
      rx pathid: 0, tx pathid: 0x0

```

If one or more component routes have mismatched cost-community IDs, individual cost-communities are added to the aggregate for each unique ID with a default cost value. CSR2 and CSR3 have mismatched IDs and the aggregate has two IGP cost-communities, both with default cost values, per unique ID. This isn't very useful for the aggregate, since the presence of the default cost-community effectively does nothing for BGP best-path comparisons. At this point, the cost-community values just become "tags" that would be used to identify the community IDs carried by the component routes.

```

CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 3
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  (20)
    3.3.3.3 (metric 31) from 3.3.3.3 (3.3.3.3)
      Origin IGP, metric 20, localpref 100, valid, confed-internal, best
      Extended Community: Cost:igp:22:300
      rx pathid: 0, tx pathid: 0x0

```

```

CSR1#show bgp ipv4 unicast 100.4.4.44/32
BGP routing table entry for 100.4.4.44/32, version 6
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  (20)
    2.2.2.2 (metric 30) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, confed-internal, best
      Extended Community: Cost:igp:21:100
      rx pathid: 0, tx pathid: 0x0

```

```

CSR1#show bgp ipv4 unicast 100.4.4.0/24
BGP routing table entry for 100.4.4.0/24, version 7
Paths: (1 available, best #1, table default)
  Advertised to update-groups:
    6
  Refresh Epoch 1
  (20), (aggregated by 1020 100.1.1.1)

```

```

0.0.0.0 from 0.0.0.0 (100.1.1.1)
  Origin IGP, localpref 100, weight 32768, valid, aggregated, local, best
  Extended Community: Cost:igp:21:2147483647 (default)
    Cost:igp:22:2147483647 (default)
  rx pathid: 0, tx pathid: 0x0

```

If one or more component routes have varying POI types (pre-bestpath versus IGP cost), the same rules apply as demonstrated above. If the IDs do not match, the metric is set to the default, and each custom ID is added to the aggregate. The only difference is that the POI is maintained. The above example has been extended with a third component route to illustrate this. The aggregate has two IGP and one pre-bestpath communities, each with varying IDs, and all with default metric.

```

CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 3
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  (20)
    3.3.3.3 (metric 31) from 3.3.3.3 (3.3.3.3)
      Origin IGP, metric 20, localpref 100, valid, confed-internal, best
      Extended Community: Cost:igp:22:300
      rx pathid: 0, tx pathid: 0x0

```

```

CSR1#show bgp ipv4 unicast 100.4.4.44/32
BGP routing table entry for 100.4.4.44/32, version 6
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  (20)
    2.2.2.2 (metric 30) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, confed-internal, best
      Extended Community: Cost:igp:21:100
      rx pathid: 0, tx pathid: 0x0

```

```

CSR1#show bgp ipv4 unicast 100.4.4.144/32
BGP routing table entry for 100.4.4.144/32, version 8
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  (20)
    2.2.2.2 (metric 30) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 10, localpref 100, valid, confed-internal, best
      Extended Community: Cost:pre-bestpath:7:7
      rx pathid: 0, tx pathid: 0x0

```

```

CSR1#show bgp ipv4 unicast 100.4.4.0/24
BGP routing table entry for 100.4.4.0/24, version 9
Paths: (1 available, best #1, table default)

```

Advertised to update-groups:

```
6
Refresh Epoch 1
(20), (aggregated by 1020 100.1.1.1)
0.0.0.0 from 0.0.0.0 (100.1.1.1)
Origin IGP, localpref 100, weight 32768, valid, aggregated, local, best
Extended Community: Cost:pre-bestpath:7:2147483647 (default)
Cost:igp:21:2147483647 (default) Cost:igp:22:2147483647 (default)
rx pathid: 0, tx pathid: 0x0
```

Even if the ID matches between a pre-bestpath and IGP cost POI, the two are incompatible and incomparable, and both are added to the aggregate. The outputs are almost identical to the above example, except the pre-bestpath POI component route IDs matches one of the IGP cost-community IDs.

```
CSR1#show bgp ipv4 unicast 100.4.4.4/32
BGP routing table entry for 100.4.4.4/32, version 3
Paths: (1 available, best #1, table default)
Not advertised to any peer
Refresh Epoch 1
(20)
3.3.3.3 (metric 31) from 3.3.3.3 (3.3.3.3)
Origin IGP, metric 20, localpref 100, valid, confed-internal, best
Extended Community: Cost:igp:22:300
rx pathid: 0, tx pathid: 0x0
```

```
CSR1#show bgp ipv4 unicast 100.4.4.44/32
BGP routing table entry for 100.4.4.44/32, version 6
Paths: (1 available, best #1, table default)
Not advertised to any peer
Refresh Epoch 1
(20)
2.2.2.2 (metric 30) from 2.2.2.2 (2.2.2.2)
Origin IGP, metric 10, localpref 100, valid, confed-internal, best
Extended Community: Cost:igp:21:100
rx pathid: 0, tx pathid: 0x0
```

```
CSR1#show bgp ipv4 unicast 100.4.4.144/32
BGP routing table entry for 100.4.4.144/32, version 10
Paths: (1 available, best #1, table default)
Not advertised to any peer
Refresh Epoch 1
(20)
2.2.2.2 (metric 30) from 2.2.2.2 (2.2.2.2)
Origin IGP, metric 10, localpref 100, valid, confed-internal, best
Extended Community: Cost:pre-bestpath:22:7
rx pathid: 0, tx pathid: 0x0
```

```
CSR1#show bgp ipv4 unicast 100.4.4.0/24
```

```

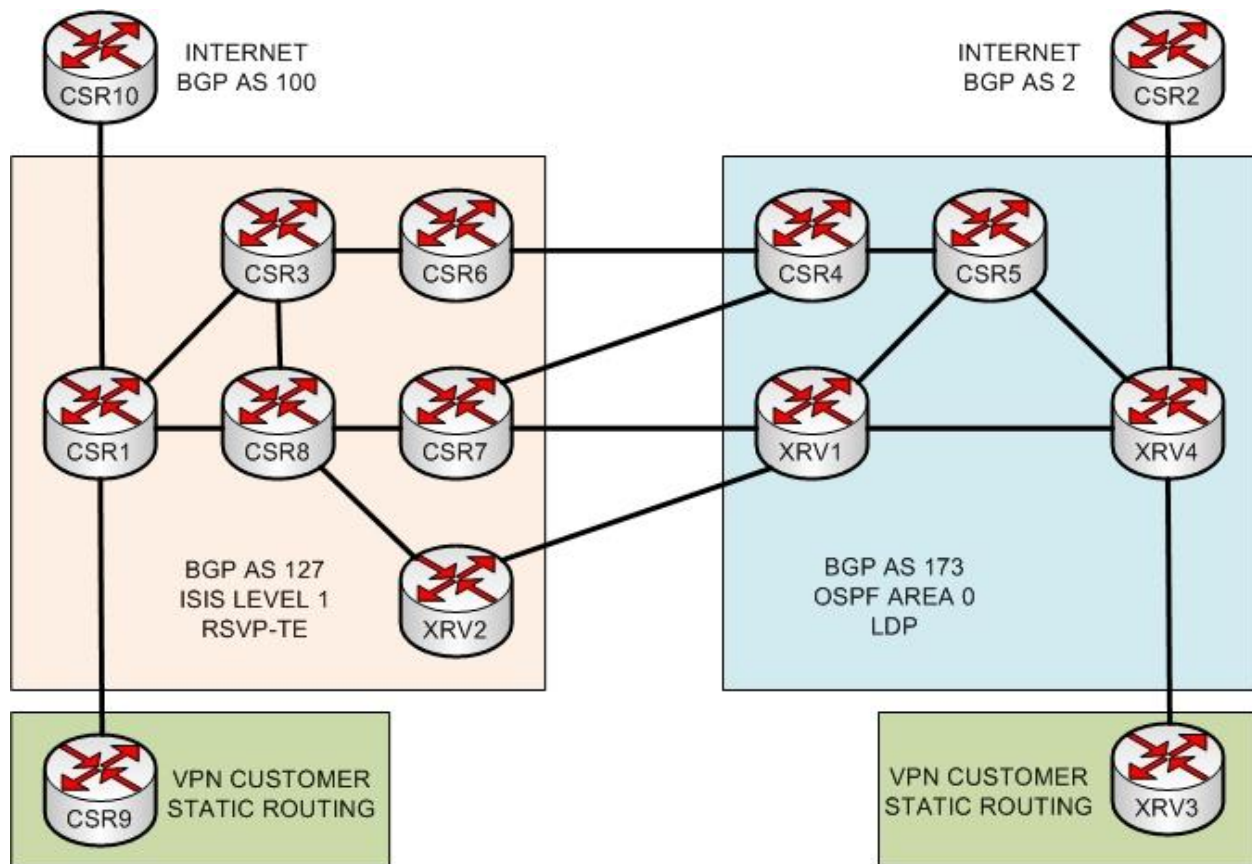
BGP routing table entry for 100.4.4.0/24, version 11
Paths: (1 available, best #1, table default)
  Advertised to update-groups:
    6
  Refresh Epoch 1
  (20), (aggregated by 1020 100.1.1.1)
    0.0.0.0 from 0.0.0.0 (100.1.1.1)
      Origin IGP, localpref 100, weight 32768, valid, aggregated, local, best
      Extended Community: Cost:pre-bestpath:22:2147483647 (default)
                          Cost:igp:21:2147483647 (default) Cost:igp:22:2147483647 (default)
      rx pathid: 0, tx pathid: 0x0

```

Additional Reading – Reference configurations “costcom”

27.6 DMZ Link Bandwidth

The DMZ link bandwidth feature is used to enable unequal cost load sharing in BGP. DMZ links are the eBGP links that interconnect BGP ASes, which this document also identifies as “transit links”. This feature encodes the bandwidth of those links into an extended community (also carries the BGP ASN for administrative purposes) which is non-transitive. Routers that learn the remote BGP routes via iBGP can perform load sharing when sending traffic to those remote destinations based on the bandwidth of the DMZ links. On XE, the feature is fully supported for IPv4 and IPv6, but only supported at the edge routers (eBGP) for VPNv4. The XE routers on the AS boundary can encode the link bandwidth (LB) community but cannot use it for load sharing for VPN-iBGP paths at this time. There is currently no VPNv6 support. XR appears to fully support all four of these AFIs. This feature is implemented very differently between XE and XR which makes interworking very difficult. Between the two, the units of measurement are vastly different and are not often converted when crossing between the two. Coupled with a limited set of show/debug commands, this makes troubleshooting very challenging. The network below has two ASes using Inter AS MPLS option B for VPNv4/v6, along with ordinary IPv4/v6. CSR9 and XRv3 are VPN customers while CSR10 and CSR2 are remote Internet ASes. CSR1 and XRv4 have iBGP neighbors to each of their ASBRs, and each ASBR has a direct eBGP peering to the other ASBR at the end of the link. AS 137 uses IS-IS and MPLS-TE for transport (no LDP) while AS 173 uses OSPFv2 and LDP. This is unrelated to DMZ link bandwidth and is introduced for variety.



First, we will quickly check the BGP peers on CSR1 and XRV4 to verify the peers are up and the proper capabilities were exchanged. IPv6 uses labeled unicast (6PE) to provide IPv6 transit across an IPv4 core. Normally, we would verify each neighbor individually as we do between CSR1 and CSR6, but in this case, I will just look at the summary for each AFI for brevity. The asterisk indicates that these are dynamic neighbors using the BGP listen command.

```
R1#show bgp ipv4 unicast neighbors 137.0.0.6 | section capabilities:
```

```
Neighbor capabilities:
```

```
Route refresh: advertised and received(new)
```

```
Four-octets ASN Capability: advertised and received
```

```
Address family IPv4 Unicast: advertised and received
```

```
Address family IPv6 Unicast: advertised and received
```

```
ipv6 MPLS Label capability: advertised and received
```

```
Address family VPNv4 Unicast: advertised and received
```

```
Address family VPNv6 Unicast: advertised and received
```

```
Enhanced Refresh Capability: advertised and received
```

```
Multisession Capability:
```

```
Stateful switchover support enabled: NO for session 1
```

```
R1#show bgp ipv4 unicast summary | begin ^Neighbor
```

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
10.1.10.10	4	100	3098	3036	353	0	0	1d21h	6
*137.0.0.6	4	137	3120	3164	353	0	0	1d21h	7

```
*137.0.0.7      4      137      3148      3163      353      0      0 1d21h      7
*137.0.0.12     4      137       30       29       353      0      0 00:15:44     7
* Dynamically created based on a listen range command
Dynamically created neighbors: 3, Subnet ranges: 1
```

```
BGP peergroup IBGP listen range group members:
 137.0.0.0/28
```

Total dynamically created neighbors: 3/(3 max), Subnet ranges: 1

```
R1#show bgp ipv6 unicast summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down  State/PfxRcd
10.1.10.10    4     100    3098    3036     499    0    0 1d21h      8
*137.0.0.6    4     137    3120    3165     499    0    0 1d21h      9
*137.0.0.7    4     137    3148    3164     499    0    0 1d21h      9
*137.0.0.12   4     137      30      30     499    0    0 00:15:52    9
[snip]

R1#show bgp vpnv4 unicast all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down  State/PfxRcd
*137.0.0.6    4     137    3120    3165     131    0    0 1d21h      1
*137.0.0.7    4     137    3148    3164     131    0    0 1d21h      1
*137.0.0.12   4     137      30      30     131    0    0 00:15:59    1
[snip]

R1#show bgp vpnv6 unicast all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down  State/PfxRcd
*137.0.0.6    4     137    3120    3165     128    0    0 1d21h      1
*137.0.0.7    4     137    3148    3164     128    0    0 1d21h      1
*137.0.0.12   4     137      30      30     128    0    0 00:16:03    1
[snip]
```

We quickly do the same on XRv4 for the routers in AS 173. Notice that the IPv4 unicast AFI only includes the eBGP peer to CSR2; the iBGP peers are technically running IPv6 labeled-unicast which XR treats as a different AFI. This is technically more correct than the XE implementation.

```
RP/0/0/CPU0:XRv4#show bgp ipv4 unicast summary | begin ^Neighbor
Neighbor      Spk    AS MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
10.2.14.2     0      2    1358    1226     22    0    0 20:18:16    7
173.0.0.4     0     173    1365    1228     22    0    0 20:18:08    6
173.0.0.11    0     173    1238    1228     22    0    0 20:18:12    6

RP/0/0/CPU0:XRv4#show bgp ipv6 unicast summary | begin ^Neighbor
Neighbor      Spk    AS MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
fd00:10:2:14::2  0      2    1351    1228     32    0    0 20:18:24    9

RP/0/0/CPU0:XRv4#show bgp ipv6 labeled-unicast summary | begin ^Neighbor
Neighbor      Spk    AS MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
173.0.0.4     0     173    1365    1228     32    0    0 20:18:17    8
173.0.0.11    0     173    1239    1228     32    0    0 20:18:22    8

RP/0/0/CPU0:XRv4#show bgp vpnv4 unicast summary | begin ^Neighbor
Neighbor      Spk    AS MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
173.0.0.4     0     173    1365    1228      9    0    0 20:18:24    1
```

```
173.0.0.11      0    173    1239    1228      9    0    0 20:18:28      1
```

```
RP/0/0/CPU0:XRv4#show bgp vpnv6 unicast summary | begin ^Neighbor
```

```
Neighbor      Spk    AS MsgRcvd MsgSent  TblVer  InQ  OutQ  Up/Down  St/PfxRcd
173.0.0.4      0    173    1365    1228      9    0    0 20:18:27      1
173.0.0.11     0    173    1239    1228      9    0    0 20:18:31      1
```

We would continue this process for all of the eBGP peers as well, but skip it for the sake of brevity. The number of prefixes exchanged in CSR1 and XRv4 indicates that there is at least some connectivity. We will begin in AS 137 first. The goal is to load-share traffic as follows: traffic to 2.128.0.0/16 prefixes should be shared in ratio 5:3:8 using CSR6, CSR7, and XRv12 respectively. Traffic to 2.129.0.0/16 prefixes should be shared in ratio 5:3:16 using CSR6, CSR7, and XRv12 respectively. We will assign some bandwidth values to the XE routers to represent 5 Mbps and 3 Mbps on CSR6 and CSR7, respectively. There is nothing fancy about this configuration. We also need to instruct these routers to “look” at the transit links and encode this value into an extended community. Last, we need to ensure extended communities can be exchanged with CSR1, which is automatic for VPNv4/6 but not IPv4/6. I also use BGP RID comparison on CSR7 so that CSR4 always wins the tie-breaker, ignoring the oldest eBGP route.

```
! CSR6
interface GigabitEthernet2.546
  bandwidth 5000

router bgp 137
  address-family ipv4
    neighbor 10.4.6.4 dmzlink-bw
    neighbor 137.0.0.1 send-community both

  address-family vpnv4
    neighbor 10.4.6.4 dmzlink-bw

  address-family ipv6
    neighbor 137.0.0.1 send-community both
    neighbor FD00:10:4:6::4 dmzlink-bw

  address-family vpnv6
    neighbor 10.4.6.4 dmzlink-bw

! CSR7
interface GigabitEthernet2.547
  bandwidth 3000

interface GigabitEthernet2.571
  bandwidth 16000

router bgp 137
  bgp bestpath compare-routerid
  address-family ipv4
    neighbor 10.4.7.4 dmzlink-bw
```



```
neighbor 10.7.11.11 dmzlink-bw
neighbor 137.0.0.1 send-community both
```

```
address-family vpnv4
neighbor 10.4.7.4 dmzlink-bw
neighbor 10.7.11.11 dmzlink-bw
```

```
address-family ipv6
neighbor 137.0.0.1 send-community both
neighbor FD00:10:4:7::4 dmzlink-bw
neighbor FD00:10:7:11::11 dmzlink-bw
```

```
address-family vpnv6
neighbor 10.4.7.4 dmzlink-bw
neighbor 10.7.11.11 dmzlink-bw
```

We can locally check CSR6 and CSR7 to ensure this community was encoded using any test prefix from AS 2. As we know, the bandwidth applied to the link-level was in kbps. In XE, this is converted to kbps (bytes) when it is received from an XE router. The CSR7-XRv1 link has bandwidth 16 Mbps but this is not a best path, and is not advertised to CSR1.

```
R6#show bgp ipv4 unicast 2.128.192.0/20
BGP routing table entry for 2.128.192.0/20, version 86
Paths: (1 available, best #1, table default)
  Advertised to update-groups:
    9
  Refresh Epoch 8
  173 2
    10.4.6.4 from 10.4.6.4 (173.0.0.4)
      Origin incomplete, localpref 100, valid, external, best
      DMZ-Link Bw 625 kbytes
      rx pathid: 0, tx pathid: 0x0
```

```
R7#show bgp ipv4 unicast 2.128.192.0/20
BGP routing table entry for 2.128.192.0/20, version 105
BGP Bestpath: compare-routerid
Paths: (2 available, best #2, table default)
  Advertised to update-groups:
    3      8
  Refresh Epoch 1
  173 2
    10.7.11.11 from 10.7.11.11 (173.0.0.11)
      Origin incomplete, localpref 100, valid, external
      DMZ-Link Bw 2000 kbytes
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 6
  173 2
    10.4.7.4 from 10.4.7.4 (173.0.0.4)
```

```
Origin incomplete, localpref 100, valid, external, best
DMZ-Link Bw 375 kbytes
rx pathid: 0, tx pathid: 0x0
```

We verify that CSR1 receives this information. If CSR1 did not receive it, that would imply that the ASBRs were not properly sending extended communities to CSR1. CSR1 is not yet configured for load-sharing; we will come back to this after configuring XRv2. I want to point out that the bandwidth values are the same as they were on the XE ASBRs, as expected.

```
R1#show bgp ipv4 unicast 2.129.0.0
BGP routing table entry for 2.129.0.0/17, version 358
Paths: (3 available, best #3, table default)
  Advertised to update-groups:
    6
  Refresh Epoch 1
  173 2
    137.0.0.12 from *137.0.0.12 (137.0.0.12)
      Origin incomplete, localpref 100, valid, internal
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 5
  173 2
    137.0.0.7 from *137.0.0.7 (137.0.0.7)
      Origin incomplete, metric 0, localpref 100, valid, internal
      DMZ-Link Bw 375 kbytes
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 7
  173 2
    137.0.0.6 from *137.0.0.6 (137.0.0.6)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      DMZ-Link Bw 625 kbytes
      rx pathid: 0, tx pathid: 0x0
```

XR allows the LB community to be tuned based on an RPL, providing additional flexibility not present in XE. To meet the requirement stated earlier, we configure a simple RPL that changes the bandwidth based on the prefix on XRv2. Notice that the values being set are in kbps; when XE receives the LB community from XR, it does not do any conversation, so these are raw kbps values.

```
! XRv2
prefix-set PL_2_128
  2.128.0.0/16 le 24
end-set

prefix-set PL_2_129
  2.129.0.0/16 le 24
end-set

route-policy RPL_DMZ
```

```

if destination in PL_2_128 then
    set extcommunity bandwidth (137:1000)
elseif destination in PL_2_129 then
    set extcommunity bandwidth (137:2000)
else
    pass
endif
end-policy

router bgp 137
neighbor 10.11.12.11
address-family ipv4 unicast
route-policy RPL_DMZ in

```

We verify that this configuration was successful on XRv2. XR shows the extended community in the form of LB:ASN:value. The 137 was automatically added, where XE leaves this as zero. **The value 8 is confusing; this actually represents Mbps. The link was configured at 1000 kbps = 8000 kbps = 8 Mbps.** The same is true for the second prefix, where 2000 kbps was configured, yet we see 16 Mbps. This at least proves that the RPL is working properly.

```

RP/0/0/CPU0:XRv2#show bgp ipv4 unicast 2.128.192.0/20
[snip]
173 2
10.11.12.11 from 10.11.12.11 (173.0.0.11)
Origin incomplete, localpref 100, valid, external, best, group-best
Received Path ID 0, Local Path ID 1, version 3
Extended community: LB:137:8
Origin-AS validity: not-found

```

```

RP/0/0/CPU0:XRv2#show bgp ipv4 unicast 2.129.0.0/17
[snip]
173 2
10.11.12.11 from 10.11.12.11 (173.0.0.11)
Origin incomplete, localpref 100, valid, external, best, group-best
Received Path ID 0, Local Path ID 1, version 7
Extended community: LB:137:16
Origin-AS validity: not-found

```

Checking CSR1 again and only focusing on the prefix from XRv2, we see the update now carries the LB community. XE leaves this value as configured, which leads me to believe XR wants the value to be configured in kbps.

```

R1#show bgp ipv4 unicast 2.129.0.0
BGP routing table entry for 2.129.0.0/17, version 358
Paths: (3 available, best #3, table default)
Advertised to update-groups:
6

```

```

Refresh Epoch 1
173 2
  137.0.0.12 from *137.0.0.12 (137.0.0.12)
    Origin incomplete, localpref 100, valid, internal
    DMZ-Link Bw 2000 kbytes
    rx pathid: 0, tx pathid: 0

```

Next, we will enable CSR1 to use these three paths proportionally. On a per-AF basis, we can enable this feature. We instruct BGP to account for the LB community (default is to ignore it) and use up to 4 iBGP paths for load sharing. Now, we can see the paths are all marked as multipath. BGP still needs to select a best-path for advertisement onward.

```

! CSR1
router bgp 137
  address-family ipv4
    bgp dmzlink-bw
    maximum-paths ibgp 4

```

```

R1#show bgp ipv4 unicast 2.129.0.0
BGP routing table entry for 2.129.0.0/17, version 365
Paths: (3 available, best #3, table default)
Multipath: iBGP
  Advertised to update-groups:
    6
  Refresh Epoch 1
  173 2
    137.0.0.12 from *137.0.0.12 (137.0.0.12)
      Origin incomplete, localpref 100, valid, internal, multipath
      DMZ-Link Bw 2000 kbytes
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 5
  173 2
    137.0.0.7 from *137.0.0.7 (137.0.0.7)
      Origin incomplete, metric 0, localpref 100, valid, internal,
multipath(oldest)
      DMZ-Link Bw 375 kbytes
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 7
  173 2
    137.0.0.6 from *137.0.0.6 (137.0.0.6)
      Origin incomplete, metric 0, localpref 100, valid, internal, multipath,
best
      DMZ-Link Bw 625 kbytes
      rx pathid: 0, tx pathid: 0x0

```

We can verify proper operation by checking the RIB and FIB for share counts. Considering the bandwidth values were in the ratio of 5:3:16, the traffic share counts are not terribly important. CEF can split traffic 16 different ways, so there isn't tremendous granularity. The 16 buckets show an approximate

proportion of 5:3:16. We can see most of the traffic going towards XRv2 via its TE tunnel (11/16), significantly less going to CSR6 (3/16), and the least going to CSR7 (2/16). Given the configured bandwidth values, this appears correct.

```
R1#show ip route 2.129.0.0
Routing entry for 2.129.0.0/17
  Known via "bgp 137", distance 200, metric 0
  Tag 173, type internal
  Last update from 137.0.0.6 00:02:15 ago
  Routing Descriptor Blocks:
  * 137.0.0.12, from 137.0.0.12, 00:02:15 ago
    Route metric is 0, traffic share count is 240
    AS Hops 2
    Route tag 173
    MPLS label: none
  137.0.0.7, from 137.0.0.7, 00:02:15 ago
    Route metric is 0, traffic share count is 44
    AS Hops 2
    Route tag 173
    MPLS label: none
  137.0.0.6, from 137.0.0.6, 00:02:15 ago
    Route metric is 0, traffic share count is 75
    AS Hops 2
    Route tag 173
    MPLS label: none
```

```
R1#show ip cef 2.129.0.0 internal | include midchain
    attached to Tunnel17, IP midchain out of Tunnel17 7F7C756E5F58
    attached to Tunnel16, IP midchain out of Tunnel16 7F7C756E7038
    attached to Tunnel112, IP midchain out of Tunnel112 7F7C756E6318
< 0 > IP midchain out of Tunnel17 7F7C756E5F58
< 1 > IP midchain out of Tunnel16 7F7C756E7038
< 2 > IP midchain out of Tunnel112 7F7C756E6318
< 3 > IP midchain out of Tunnel17 7F7C756E5F58
< 4 > IP midchain out of Tunnel16 7F7C756E7038
< 5 > IP midchain out of Tunnel112 7F7C756E6318
< 6 > IP midchain out of Tunnel16 7F7C756E7038
< 7 > IP midchain out of Tunnel112 7F7C756E6318
< 8 > IP midchain out of Tunnel112 7F7C756E6318
< 9 > IP midchain out of Tunnel112 7F7C756E6318
<10 > IP midchain out of Tunnel112 7F7C756E6318
<11 > IP midchain out of Tunnel112 7F7C756E6318
<12 > IP midchain out of Tunnel112 7F7C756E6318
<13 > IP midchain out of Tunnel112 7F7C756E6318
<14 > IP midchain out of Tunnel112 7F7C756E6318
<15 > IP midchain out of Tunnel112 7F7C756E6318
```

We quickly check a 2.128.0.0/16 prefix to verify a 5:3:8 ratio. Essentially, we should see less buckets assigned to XRv2 and more assigned to CSR6 and CSR7. Notice the traffic share counts are twice as high for CSR6 and CSR7, mostly because the shares for XRv2 were halved as the bandwidth is halved.

```
R1#show bgp ipv4 unicast 2.128.192.0/20
BGP routing table entry for 2.128.192.0/20, version 361
Paths: (3 available, best #3, table default)
Multipath: iBGP
  Advertised to update-groups:
    6
  Refresh Epoch 1
  173 2
    137.0.0.12 from *137.0.0.12 (137.0.0.12)
      Origin incomplete, localpref 100, valid, internal, multipath
      DMZ-Link Bw 1000 kbytes
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 5
  173 2
    137.0.0.7 from *137.0.0.7 (137.0.0.7)
      Origin incomplete, metric 0, localpref 100, valid, internal,
multipath(oldest)
      DMZ-Link Bw 375 kbytes
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 7
  173 2
    137.0.0.6 from *137.0.0.6 (137.0.0.6)
      Origin incomplete, metric 0, localpref 100, valid, internal, multipath,
best
      DMZ-Link Bw 625 kbytes
      rx pathid: 0, tx pathid: 0x0
```

```
R1#show ip route 2.128.192.0
Routing entry for 2.128.192.0/20
  Known via "bgp 137", distance 200, metric 0
  Tag 173, type internal
  Last update from 137.0.0.6 00:15:25 ago
  Routing Descriptor Blocks:
  * 137.0.0.12, from 137.0.0.12, 00:15:25 ago
    Route metric is 0, traffic share count is 240
    AS Hops 2
    Route tag 173
    MPLS label: none
  137.0.0.7, from 137.0.0.7, 00:15:25 ago
    Route metric is 0, traffic share count is 89
    AS Hops 2
    Route tag 173
    MPLS label: none
  137.0.0.6, from 137.0.0.6, 00:15:25 ago
    Route metric is 0, traffic share count is 150
```

```
AS Hops 2
Route tag 173
MPLS label: none
```

This time, only 8/16 buckets are allocated for XRv2, while there are 5/16 for CSR6 and 3/16 for CSR7. When the ratio numbers add up to 16 as they do here, you end up with a very clean distribution scheme. This CEF load sharing scheme matches the target ratio perfectly.

```
R1#show ip cef 2.128.192.0 internal | include midchain
    attached to Tunnel17, IP midchain out of Tunnel17 7F7C756E5F58
    attached to Tunnel16, IP midchain out of Tunnel16 7F7C756E7038
    attached to Tunnel112, IP midchain out of Tunnel112 7F7C756E6318
< 0 > IP midchain out of Tunnel17 7F7C756E5F58
< 1 > IP midchain out of Tunnel16 7F7C756E7038
< 2 > IP midchain out of Tunnel112 7F7C756E6318
< 3 > IP midchain out of Tunnel17 7F7C756E5F58
< 4 > IP midchain out of Tunnel16 7F7C756E7038
< 5 > IP midchain out of Tunnel112 7F7C756E6318
< 6 > IP midchain out of Tunnel17 7F7C756E5F58
< 7 > IP midchain out of Tunnel16 7F7C756E7038
< 8 > IP midchain out of Tunnel112 7F7C756E6318
< 9 > IP midchain out of Tunnel16 7F7C756E7038
<10 > IP midchain out of Tunnel112 7F7C756E6318
<11 > IP midchain out of Tunnel16 7F7C756E7038
<12 > IP midchain out of Tunnel112 7F7C756E6318
<13 > IP midchain out of Tunnel112 7F7C756E6318
<14 > IP midchain out of Tunnel112 7F7C756E6318
<15 > IP midchain out of Tunnel112 7F7C756E6318
```

The process is identical for IPv6. We will use the legacy method on XRv2, which is a neighbor-level command to enable DMZ LB. This applies to all AFs for which there is not a more specific RPL attachment. Unfortunately, no matter what you set the actual bandwidth on the link, XR always uses value 500,000. This means that CSR1 will interpret this as 500,000 kbps, which is worthless in the context of the current low-speed bandwidths. Since XE does not have the per-AF granularity to make adjustments, I use this as a demonstration of a bad LB design. You also cannot verify it locally on XR due to what I consider an order-of-operations issue. CSR1 clearly shows this bogus value.

```
! XRv2
interface GigabitEthernet0/0/0/0.512
    bandwidth 4000
```

```
router bgp 137
    neighbor fd00:10:11:12::11
        dmz-link-bandwidth
```

```
R1#show bgp ipv6 unicast 2001:2:128:190::/62
BGP routing table entry for 2001:2:128:190::/62, version 499
```

```

Paths: (3 available, best #3, table default)
Multipath: iBGP
  Advertised to update-groups:
    6
  Refresh Epoch 1
173 2
  ::FFFF:137.0.0.12 from *137.0.0.12 (137.0.0.12)
    Origin incomplete, localpref 100, valid, internal, multipath
    DMZ-Link Bw 500000 kbytes
    mpls labels in/out nolabel/92006
    rx pathid: 0, tx pathid: 0
  Refresh Epoch 13
173 2
  ::FFFF:137.0.0.7 from *137.0.0.7 (137.0.0.7)
    Origin incomplete, metric 0, localpref 100, valid, internal,
multipath(oldest)
    DMZ-Link Bw 375 kbytes
    mpls labels in/out nolabel/7002
    rx pathid: 0, tx pathid: 0
  Refresh Epoch 13
173 2
  ::FFFF:137.0.0.6 from *137.0.0.6 (137.0.0.6)
    Origin incomplete, metric 0, localpref 100, valid, internal, multipath,
best
    DMZ-Link Bw 625 kbytes
    mpls labels in/out nolabel/6017
    rx pathid: 0, tx pathid: 0x0

```

As expected, the load balancing weights are heavily tilted and all traffic should be forwarded out of XRv2. I am not sure if this is an XE issue or a function of 6PE, but the CEF table shows equal distribution between all of the buckets. The IPv6 CEF has a nice distribution output that IPv4 CEF used to have. Even without knowing what the paths are, we can see they are processed in round-robin fashion (each path gets a number). This is obviously incorrect despite having the correct configuration.

```

R1#show ipv6 cef 2001:2:128:190::/62 internal | include Path index
  Path index      [ 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 ]

```

VPNv4 link bandwidth load-sharing is limited to eBGP in current XE code. On CSR7, we can demonstrate this balance between the 3 Mbps link to CSR4 and 16 Mbps link to XRv1. Take note of the inbound label.

```

! CSR7
address-family vpnv4
  bgp dmzlink-bw
  maximum-paths 2

```

```

R7#show bgp vpnv4 unicast rd 137:173 10.13.14.0/24
BGP routing table entry for 137:173:10.13.14.0/24, version 38

```



```

BGP Bestpath: compare-routerid
Paths: (2 available, best #2, no table)
  Advertised to update-groups:
    3          7
Refresh Epoch 1
173
  10.7.11.11 (via default) from 10.7.11.11 (173.0.0.11)
    Origin incomplete, localpref 100, valid, external, multipath(oldest)
    Extended Community: RT:137:173
    DMZ-Link Bw 2000 kbytes
    mpls labels in/out 7003/91005
    rx pathid: 0, tx pathid: 0
Refresh Epoch 3
173
  10.4.7.4 (via default) from 10.4.7.4 (173.0.0.4)
    Origin incomplete, localpref 100, valid, external, multipath, best
    Extended Community: RT:137:173
    DMZ-Link Bw 375 kbytes
    mpls labels in/out 7003/4015
    rx pathid: 0, tx pathid: 0x0

```

Since there is no VRF locally configured on CSR7 (Option B), there won't be a RIB or FIB entry. Traffic will be label-switched, with BGP performing a VPN label swap, at the ASBRs. Like IPv6, the feature does not appear to work very well, and could be MPLS related. The distribution appears 1:1 despite enabling DMZ LB feature for this AFI.

```

R7#show mpls forwarding-table labels 7003 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
7003      4015      137:173:10.13.14.0/24  \
                                         0          Gi2.547    10.4.7.4
MAC/Encaps=18/22, MRU=1500, Label Stack{4015}
005056A92C57005056A9EA7781000DDB8847 00FAF000
No output feature configured
Per-destination load-sharing, slots: 0 2 4 6 8 10 12 14
          91005      137:173:10.13.14.0/24  \
                                         0          Gi2.571    10.7.11.11
MAC/Encaps=18/22, MRU=1500, Label Stack{91005}
005056A99C60005056A9EA7781000DF38847 1637D000
No output feature configured
Per-destination load-sharing, slots: 1 3 5 7 9 11 13 15

```

Next, we will configure AS 173. The goal is to balance traffic in a 2:1 ratio preferring CSR4 twice as often as XRv1. When XRv4 receives the LB community from an XE ASBR, it divides it by 1000. Since the bandwidth was specified in kbps on CSR4, XR wants to view it in Mbps, so dividing by 1000 is logical. XR also cannot process LB values less than 1000, and since XR interprets the value as Mbps, this means that only 1 Gbps links (1000 Mbps) or faster will even register as having an LB value. As such, I set the

bandwidth to 8 Gbps on CSR4 to achieve a value of 4000 on XRv4. Notice how the value is wildly different on CSR4 from XRv4. In AS 137, values set by XR were untouched by XE, but values set by XE are divided by 1000 when received by XR. XE reads the value as kbps and displays it in kbps. XR reads the value as kbps and displays it in Mbps with a lower bound of 1000 Mbps. There is also an unidentifiable extended community sent by CSR4 that XR cannot read. A quick search did not reveal obvious documentation for extended community 0x0006. Also notice that CSR4 did not set the ASN inside the LB community and it remains 0, whereas XR as an ASBR set the value to the local ASN.

```
! CSR4
interface GigabitEthernet2.546
  bandwidth 8000000

R4#show bgp ipv4 unicast 100.64.32.0/20
BGP routing table entry for 100.64.32.0/20, version 91
Paths: (2 available, best #2, table default)
  Advertised to update-groups:
    3          17
  Refresh Epoch 6
  137 100
    10.4.7.7 from 10.4.7.7 (137.0.0.7)
      Origin incomplete, localpref 100, valid, external
      DMZ-Link Bw 375 kbytes
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 8
  137 100
    10.4.6.6 from 10.4.6.6 (137.0.0.6)
      Origin incomplete, localpref 100, valid, external, best
      DMZ-Link Bw 1000000 kbytes
      rx pathid: 0, tx pathid: 0x0
```

```
RP/0/0/CPU0:XRv4#show bgp ipv4 unicast 100.64.32.0/20
[snip]
  137 100
    173.0.0.4 (metric 3) from 173.0.0.4 (173.0.0.4)
      Origin incomplete, metric 0, localpref 100, valid, internal, multipath
      Received Path ID 0, Local Path ID 0, version 0
      Extended community: 0x0006:0:1232348160 LB:0:8000
[snip]
```

Just like XE-to-XE, XR-to-XR is a little easier. While it still doesn't read the interface bandwidth properly, the value on XRv4 corresponds to what we saw earlier. CSR1 showed an LB value of 500,000 kbps when it received LB from XRv2 using the legacy configuration (no RPL). XRv4 shows a value of 4,000 Mbps; these two values are equivalent to 4 Gbps. I am not sure where XR is getting this number, but at least it is consistent. Notice that there is no unknown extended community and the ASN is encoded, unlike XE.

```
RP/0/0/CPU0:XRv4#show bgp ipv4 unicast 100.64.32.0/20
```

```
[snip]
Path #1: Received by speaker 0
Not advertised to any peer
137 100
  173.0.0.4 (metric 3) from 173.0.0.4 (173.0.0.4)
    Origin incomplete, metric 0, localpref 100, valid, internal, multipath
    Received Path ID 0, Local Path ID 0, version 0
    Extended community: 0x0006:0:1232348160 LB:0:8000
Path #2: Received by speaker 0
Advertised to peers (in unique update groups):
  10.2.14.2
137 100
  173.0.0.11 (metric 2) from 173.0.0.11 (173.0.0.11)
    Origin incomplete, localpref 100, valid, internal, best, group-best,
multipath
    Received Path ID 0, Local Path ID 1, version 15
    Extended community: LB:173:4000
```

To enable XRv4 to use this information, we need to enable maximum-paths. I use unequal-cost since, unlike AS 137, there are different IGP costs to the BGP next-hops, and I need to multipath across the two. AS 137 used static routes with RSVP-TE so this was a non-issue. We can clearly see the weights of 8000 and 4000 being applied to the candidate routes. The XR show commands for CEF distribution are very clean. XRv4 builds 3 buckets for more accurate load sharing, rather than forcing a 2:1 ratio to be split evenly over 16 buckets, which is impossible. The load distribution shows path 0 (CSR4) being used twice as often as path 1 (XRv1).

```
! XRv4
router bgp 173
  address-family ipv4 unicast
    maximum-paths ibgp 2 unequal-cost

RP/0/0/CPU0:XRv4#show route ipv4 unicast 100.64.32.0/20
Routing entry for 100.64.32.0/20
  Known via "bgp 173", distance 200, metric 0
  Tag 137, type internal
  Routing Descriptor Blocks
    173.0.0.4, from 173.0.0.4, BGP multi path
      Route metric is 0, Wt is 8000
    173.0.0.11, from 173.0.0.11, BGP multi path
      Route metric is 0, Wt is 4000
  No advertising protos.

RP/0/0/CPU0:XRv4#show cef ipv4 100.64.32.0/20 detail | begin Weight
Weight distribution:
  slot 0, weight 8000, normalized_weight 2, class 0
  slot 1, weight 4000, normalized_weight 1, class 0

  Load distribution: 0 0 1 (refcount 1)
```

Hash	OK	Interface	Address
0	Y	GigabitEthernet0/0/0/0.554	173.5.14.5
1	Y	GigabitEthernet0/0/0/0.554	173.5.14.5
2	Y	GigabitEthernet0/0/0/0.514	173.11.14.11

We will turn the tables and use XRv1 twice as often when CSR4 picks CSR7 as the best path. By setting the link to 2 Gbps, we will achieve a value of 2000 on XR, which is half of XRv1's value of 4000. XR needs to specify the maximum-paths under the VRF IPv4/v6 AFI, not the VPNv4 AFI.

```
! CSR4
interface GigabitEthernet2.547
  bandwidth 2000000

router bgp 173
  vrf DMZ
    address-family ipv4 unicast
      maximum-paths ibgp 2 unequal-cost
```

We verify the communities below. We see that both were imported, are multipath, and have the proper 2:1 ratio of XRv1:CSR4 for load sharing.

```
RP/0/0/CPU0:XRv4#show bgp vpnv4 unicast vrf DMZ 10.1.9.0/24
[snip]
  Path #1: Received by speaker 0
  Not advertised to any peer
  137
    173.0.0.4 (metric 3) from 173.0.0.4 (173.0.0.4)
      Received Label 4000
      Origin incomplete, metric 0, localpref 100, valid, internal, multipath,
import-candidate, imported
      Received Path ID 0, Local Path ID 0, version 0
      Extended community: 0x0006:0:1215570944 LB:0:2000 RT:137:173
      Source VRF: DMZ, Source Route Distinguisher: 137:173
  Path #2: Received by speaker 0
  Not advertised to any peer
  137
    173.0.0.11 (metric 2) from 173.0.0.11 (173.0.0.11)
      Received Label 91008
      Origin incomplete, localpref 100, valid, internal, best, group-best,
multipath, import-candidate, imported
      Received Path ID 0, Local Path ID 1, version 11
      Extended community: LB:173:4000 RT:137:173
      Source VRF: DMZ, Source Route Distinguisher: 137:173
```

We check the routing information within the VPN to verify it is correct. Since multipath was configured under the VRF, both routes can be imported into the VPN for multipath.

```

RP/0/0/CPU0:XRv4#show route vrf DMZ ipv4 unicast 10.1.9.0/24
Routing entry for 10.1.9.0/24
  Known via "bgp 173", distance 200, metric 0
  Tag 137, type internal
  Routing Descriptor Blocks
    173.0.0.4, from 173.0.0.4, BGP multi path
      Nexthop in Vrf: "default", Table: "default", IPv4 Unicast, Table Id:
0xe0000000
        Route metric is 0, Wt is 2000
    173.0.0.11, from 173.0.0.11, BGP multi path
      Nexthop in Vrf: "default", Table: "default", IPv4 Unicast, Table Id:
0xe0000000
        Route metric is 0, Wt is 4000
  No advertising protos.

```

Since these are VPN routes, the label imposition is shown, not the outgoing interface. We can connect the two easily by checking the LFIB. Label 94000 corresponds to CSR4 and 92002 corresponds to XRv1, so the load distribution appears correct.

```

RP/0/0/CPU0:XRv4#show cef vrf DMZ ipv4 10.1.9.0/24 detail | begin Weight
Weight distribution:
slot 0, weight 2000, normalized_weight 1, class 0
slot 1, weight 4000, normalized_weight 2, class 0

Load distribution: 0 1 1 (refcount 1)

```

Hash	OK	Interface	Address
0	Y	Unknown	94000/0
1	Y	Unknown	94002/0
2	Y	Unknown	94002/0

```

RP/0/0/CPU0:XRv4#show mpls forwarding labels 94000 94002

```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94000	5007	173.0.0.4/32	Gi0/0/0/0.554	173.5.14.5	199672
94001	Pop	173.0.0.5/32	Gi0/0/0/0.554	173.5.14.5	144626
94002	Pop	173.0.0.11/32	Gi0/0/0/0.514	173.11.14.11	327767

Additional Reading – Reference configurations “bgp-dmz-lb”

27.7 BGP Multicast VPN (MVPN) Theory

MVPN is a generalized name meant to encompass all of the solutions meant to transport multicast traffic from place to place in a VPN environment. In most cases, the “VPN environment” specifically means MPLS 3VPN. The main business driver for this is the explosion of Internet video over the past

years, and these video services are expected to increase further still. A very long set of solutions has been developed and adopted as industry standards to transport multicast across MPLS L3VPN.

The original solution, known commonly as Draft-Rosen, used GRE tunnels to connect all of the PE devices in a given MVPN instance (generally within a given L3VPN, though extranets are supported). The destination of these tunnels was multicast also, so the SP core needed to run PIM to transport it. It has worked well for many years and one of its greatest benefits is that it doesn't introduce any particularly new or difficult concepts. Both the provider and customer use PIM, except the provider tunnels (P-tunnels) just carry customer multicast inside. It can be protected with multicast-only FRR (MoFRR) and also works with PIM-ASM, PIM-SSM, or PIM-Bidir in the core. It can also transport any of these three traffic types used by the customer, along with customer BSR messaging for RP discovery. It carries IPv6 multicast inside these IPv4 P-tunnels as well. Overall, it is a good solution, and despite it being "legacy" it is still considered an MVPN solution (profile 0).

The main issue with this solution is that it does not take advantage of MPLS transport. It would be ideal if both unicast and multicast traffic used a common data plane in the SP core, ultimately removing the need to run PIM there. Label switched multicast (LSM) can be candidate for FRR using the methods already used for unicast LSPs. Two main technologies were extended to support this: RSVP-TE and LDP.

RSVP was already extended to support TE (RSVP-TE), so extending it again to transport multicast was logical. Using P2MP RSVP-TE, a multipoint tunnel can be built from headend to multiple tail ends, which is perfect for transporting one-way multicast. Like a normal TE tunnel, each sub-tunnel (also called sub-LSP) can request FRR treatment, make bandwidth reservations, and do any other kind of TE CSPF parameter tuning. This is Juniper's preferred model and it is the only model that supports bandwidth reservations and simplified FRR. XE never supported this feature (the IOS S-train did), but XR does.

LDP was extended to support this also; it is called multicast LDP (mLDP). This works like LDP where labels are exchanged on a hop-by-hop basis. mLDP can support P2MP trees, which are unidirectional kind of like RSVP-TE P2MP trees. It can also support MP2MP trees, which are similar to PIM-Bidir where traffic can flow in either direction towards the root, which is the main junction point. In P2MP trees, only downstream labels are allocated, whereas in MP2MP trees, upstream and downstream labels are exchanged. P2MP trees are more efficient (like PIM-SSM) but MP2MP trees use less state (like PIM-Bidir). Unlike RSVP LSPs, mLDP LSPs are built from tail to head as label-mapping messages are sent towards the root of the delivery tree. This is Cisco's preferred model as it has less "core state", even for P2MP trees, and generally scales better. It supports FRR but it's less straightforward than RSVP-TE P2MP, and there is no support for making bandwidth reservations.

Some solutions permit dynamic discovery of PE endpoints. Rather than statically configure RSVP-TE P2MP endpoints or mLDP MP2MP roots, BGP can automatically discover endpoints. This is often called BGP auto-discovery (AD). BGP introduces 5 new route-types under the IPv4/v6 MVPN SAFI to accomplish this. The concept of Provider Multicast Service Interface (PMSI) is an abstraction for the router. As long as the C-MRIB is smart enough to know "I need to deliver this packet to the PMSI", the MVPN process

knows what to do next. The PMSI actions will typically include some kind of encapsulation, be it IP/GRE or MPLS, for transport across the core. Traffic exiting the core traverses the PMSI as well where the encapsulation is removed. Different types of BGP routes mean different things as described below.

Type 1: Intra-AS Inclusive PMSI (I-PMSI) AD: Originated by each PE inside an AS; used to learn MVPN membership. It is called “inclusive” because it includes everyone in the MVPN, whether they want traffic or not. I-PMSI can be thought of as similar to a default MDT.

Type 2: Inter-AS I-PMSI AD: Originated by each ASBR; used to learn MVPN membership. This is examined in detail when testing inter-AS MVPN.

Type 3: Selective (S-PMSI) AD: Originated by the source (ingress) PE and used for signaling a specific P2MP tree or data MDT. This is meant to selectively target receivers that want multicast for a given C(S,G).

Type 4: Leaf AD: If the Type-3 requests it, the receiver (Egress) PEs will explicitly show interest in a specific C(S,G). Normally this is not needed for the vast majority of profiles, but has relevance for RSVP P2MP-TE auto-discovered tunnels.

Type 5: Source Active AD: Originated by the source (ingress) PE when it learns about an active source and assist with SPT switchover. This is only relevant when the customer is using PIM-ASM.

The method by which customers exchange multicast routing has always been PIM. In Draft-Rosen, the PEs would form PIM neighbors inside the VRF and the tunnel-mesh would look like an emulated LAN. The same is true for basic mLDP MP2MP trees. This works well and most people are familiar with PIM operations on a LAN. To achieve better scalability, BGP was extended to do customer multicast (c-mcast) signaling. BGP introduces 2 new route-types under the IPv4/v6 MVPN SAFI to accomplish this; basically the PM equivalent of a (*,G) join and an (S,G) join. Receiving a PIM (*,G) or (S,G) join from the CE will trigger the creation of a BGP Type-6 or Type-7 route, respectively. Receiving a prune simply triggers the withdrawal of that route. BGP allows for massive scalability in MVPNs as the concept of a PIM “emulated LAN” for overlay signaling is removed.

Type 6: Shared Tree Join: Specifies the RP and group, and is meant to signal C-PIM-ASM interest in a particular group. This replaces the PIM (*,G) signaling.

Type 7: Source Tree Join: Looks identical to a Type-6 except the RP’s IP address is substituted for the source’s address. It can also be sent upon receipt of a Type-5 Source Active AD message. This replaces the PIM (S,G) signaling.

mLDP is also capable of carrying the c-mcast signaling in-band. That is, PIM/BGP is not needed to exchange customer multicast information since the actually (S,G) information is encoded in the mLDP label mapping messages as an opaque value. This only works with PIM-SSM, since there isn’t a

mechanism for (*,G) signaling nor carrying BSR messages across the VPN. It also scales poorly, but is a very simple solution that eliminates the need for an overlay protocol entirely. For customers using SSM that only have a few, high-bandwidth multicast flows (which is common), and this is a good solution.

A more rudimentary mechanism for transporting multicast includes performing the multicast replication at the head-end and reusing the unicast LSPs. This automatically gives all the flows FRR (provided the unicast LSPs were already protected), doesn't require new signaling, and also overcomes any vendor incompatibility issues. Ingress Replication (IR) is not commonly used (and is supported on XR only), but can be used in specific corner-cases. PBB-EVPN uses this by default for its inclusive multicast exchanges as it assumes that the core is not MVPN-capable by default, which is a good assumption.

In order for the router to know which kind of P-tunnel to use for a particular multicast flow (assuming BGP AD is used), it needs to look at the I-PMSI or S-PMSI to determine the tunnel type. Each transport method has a different tunnel type. This information is carried inside the PMSI attributes within BGP AD routes; once a BGP AD route is bound to a particular C(S,G), the tunnel type specified in the PMSI attribute is used for transport. A mismatch between endpoints of tunnel-types is not supported. This is how remote endpoints agree on how to encapsulate/decapsulate traffic traversing their local PMSIs.

Type 0: No tunnel specified

Type 1: RSVP-TE P2MP

Type 2: mLDP P2MP

Type 3: PIM-SSM

Type 4: PIM-ASM

Type 5: PIM-Bidir

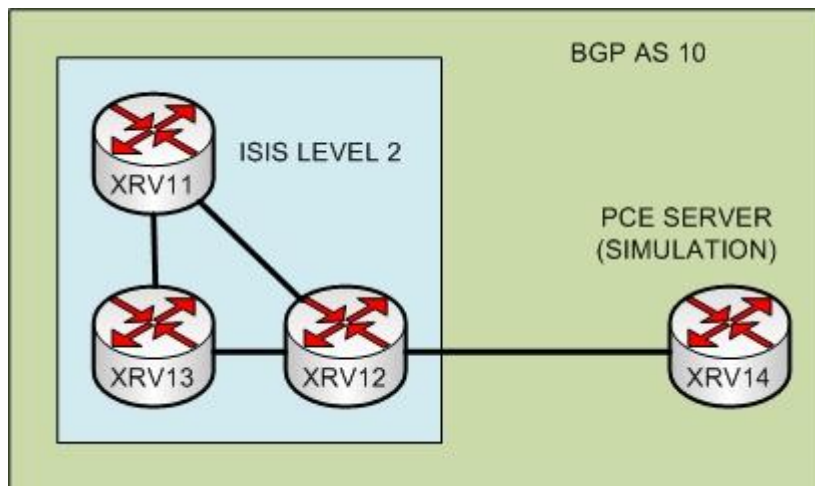
Type 6: Ingress Replication (IR)

Type 7: mLDP MP2MP

As you can see, there are many options; many of the combinations of transport and overlay signaling are valid. **Note: This chapter lays the foundation for the MVPN tests. All of the detailed verification happens there.**

27.8 BGP Link State AF and Path Computation Element (PCE)

BGP-LS is designed to capture link state IGP and TE information and transport it across the network. Specifically, passing it to an SDN/PCE device allows the network to be controlled remotely, enabling network-wide TE and other features. BGP cannot actually use the IGP link-state information for any kind of path selection, but it is easily readable from the CLI. BGP is truly a transport mechanism as it scales much better than IGP and can peer directly with SDN controllers or PCE devices. The feature is simple to configure and only supported in IOS XR at present. The documentation says the peer address must be private but that isn't enforced in XR version 5.3.0. After identifying a domain-distinguisher, just enable the AF per neighbor as usual. The network diagram is shown below.



Below is the configuration of our simulated PCE server. Essentially, it is going to accept BGP-LS information from XRV12 and do nothing with it.

```
! XRV14
router bgp 10
  address-family link-state link-state
    domain-distinguisher 10:14.14.14.14

  neighbor 12.12.12.12
    address-family link-state link-state
```

The configuration is similar on XRV12, with the addition of the IS-IS LSP distribution into BGP. OSPFv2 is also supported, but OSPFv3 does not appear supported yet.

```
! XRV12
router isis 1
  distribute bgp-ls level 2

router bgp 10
  address-family link-state link-state
    domain-distinguisher 10:12.12.12.12

  neighbor 12.12.12.12
    address-family link-state link-state
```

A quick look at the BGP table for this AFI shows all of the IS-IS topology information from level 2 that was injected by XRV12. We will select a few different route types to examine. The BGP codes in the output are excellent and help to decipher the output. The [V] route represents the "vertex" for XRV12 itself. The [L2] is the IS-IS level, and since this isn't a link in the graph, there is a local node [N] but no remote node [R]. The NET is 0000.0000.0012.00 and the BGP RID that originated the prefix was 12.12.12.12 (XRV12). The [E] route, or "edge", is similar except it has a remote node, which is

0000.0000.0012.03. This is an IS-IS DIS for one of XRv12's links. The [T] route contains the prefix for XRv12's loopback0 which is 12.12.12.12/32. The prefix-codes are very useful; don't ignore them.

```
RP/0/0/CPU0:XRv14#show bgp link-state link-state
[snip]
Status codes: s suppressed, d damped, h history, * valid, > best
                i - internal, r RIB-failure, S stale, N Nexthop-discard
Origin codes: i - IGP, e - EGP, ? - incomplete
Prefix codes: E link, V node, T IP reachable route, u/U unknown
                I Identifier, N local node, R remote node, L link, P prefix
                L1/L2 ISIS level-1/level-2, O OSPF, D direct, S static
                a area-ID, l link-ID, t topology-ID, s ISO-ID,
                c confed-ID/ASN, b bgp-identifier, r router-ID,
                i if-address, n nbr-address, o OSPF Route-type, p IP-prefix
                d designated router address

      Network          Next Hop          Metric LocPrf Weight Path
[snip]
*>i[V] [L2] [I0x0] [N[c10] [b12.12.12.12] [s0000.0000.0012.00]]/328
                        12.12.12.12          100      0 i
[snip]
*>i[E] [L2] [I0x0] [N[c10] [b12.12.12.12] [s0000.0000.0012.00]] [R[c10] [b12.12.12.12] [s0000.
0000.0012.03]]/576
                        12.12.12.12          100      0 i
[snip]
*>i[T] [L2] [I0x0] [N[c10] [b12.12.12.12] [s0000.0000.0012.00]] [P[p12.12.12.12/32]]/400
                        12.12.12.12          100      0 i
[snip]
```

The details of the [V] route help to summarize our findings, also. The IS-IS area is not encoded anywhere in the summary information, so one would have to use the detailed show command for it. It is also nice to see the hostname (the LSP name for IS-IS at least) when trying to identify the prefixes. The TE-ID is also important since one of the main uses of this feature is MPLS TE.

```
RP/0/0/CPU0:XRv14#show bgp link-state link-state
[V] [L2] [I0x0] [N[c10] [b12.12.12.12] [s0000.0000.0012.00]]/328
BGP routing table entry for
[V] [L2] [I0x0] [N[c10] [b12.12.12.12] [s0000.0000.0012.00]]/328
Versions:
  Process          bRIB/RIB   SendTblVer
  Speaker          22         22
Paths: (1 available, best #1)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  Local
    12.12.12.12 from 12.12.12.12 (12.12.12.12)
    Origin IGP, localpref 100, valid, internal, best, group-best
    Received Path ID 0, Local Path ID 1, version 22
    Link-state: Node-name: XRv12, ISIS area: 00, Local TE Router-ID:
```

12.12.12.12

The [E] route details reveal some extra information, such as TE metric, IGP metric and TE RID. The show command is very long so I snip the entire prefix, but BGP repeats it (highlighted) when the output is displayed.

```
RP/0/0/CPU0:XRv14#show bgp link-state link-state [E] [L2] [I0x0] [N[c10] [b12.12.12.12] [snip]
```

BGP routing table entry for

```
[E] [L2] [I0x0] [N[c10] [b12.12.12.12] [s0000.0000.0012.00]] [R[c10] [b12.12.12.12] [s0000.0000.0012.03]]/576
```

Versions:

Process	bRIB/RIB	SendTblVer
Speaker	7	7

Paths: (1 available, best #1)

Not advertised to any peer

Path #1: Received by speaker 0

Not advertised to any peer

Local

12.12.12.12 from 12.12.12.12 (12.12.12.12)

Origin IGP, localpref 100, valid, internal, best, group-best

Received Path ID 0, Local Path ID 1, version 7

Link-state: Local TE Router-ID: 12.12.12.12, TE-default-metric: 100
metric: 100

The [T] route details don't really reveal anything interesting other than the IGP metric. This makes sense because this is just an IP route and doesn't carry any interesting TE information like the "vertex" and "edge" routes do, which are critical components in the link-state graph.

```
RP/0/0/CPU0:XRv14#show bgp link-state link-state [T] [L2] [I0x0] [N[c10] [b12.12.12.12] [snip]
```

BGP routing table entry for

```
[T] [L2] [I0x0] [N[c10] [b12.12.12.12] [s0000.0000.0012.00]] [P[p12.12.12.12/32]]/400
```

Versions:

Process	bRIB/RIB	SendTblVer
Speaker	33	33

Paths: (1 available, best #1)

Not advertised to any peer

Path #1: Received by speaker 0

Not advertised to any peer

Local

12.12.12.12 from 12.12.12.12 (12.12.12.12)

Origin IGP, localpref 100, valid, internal, best, group-best

Received Path ID 0, Local Path ID 1, version 33

Link-state: Metric: 0

PCE Vocabulary and Definitions

PCE: Path Computation Element is a network entity that can compute paths based on the topological graph and a set of constraints. It's basically an enhanced version of the "TE PCALC" and is often externally hosted (not composite). Paths that span multiple domains could be controlled by a single PCE (centralized) or multiple PCE (distributed).

PCC: The Path Computation Client generally represents an LSR who can request a path computation.

PCEP: PCE protocol is used between PCE and PCC and is defined in RFC 5440. It is just a TCP session whereby path requests, replies, commands, and error messages are exchanged.

TED: Traffic Engineering Database is what the PCE uses to determine paths through the network. The TED is not a new construct as RSVP-TE used the same information as input to the "TE PCALC" algorithm, ultimately generating an ERO to feed RSVP-TE PATH messages.

PCE systems can operate in one of two modes

Stateful PCE: The PCE has a strict synchronization with its PCC devices to include topology and resource information. This allows for optimal path computation network-wide. The downside is the significant control-plane overhead between PCE/PCC, and potentially multiple PCEs (more SDN-like). With the delegation option enabled, you also give PCE the authority to change your statically configured tunnels also, not just the automatically created PCE ones. The automatically created ones, like other auto-tunnels for TE, will be spawned and re-used based on the range of numbers you specify.

Stateless PCE: Each request for a path is computed independent of one another and the PCE does not remember any path information. Less control-plane overhead but much less capability. This is less SDN-like and might be better for PCE newcomers initially.

IGP extensions to support PCE discovery: IS-IS introduces "PCED sub-TLV type 5" while OSPF includes sub-TLVs inside the Router Information LSA (Opaque Type 4, Opaque ID 0) with TLV PCEC type 6. There are more sub-TLVs included to define specific PCE parameters covered in RFCs 5088 and 5089 for OSPF and IS-IS, respectively.

Three main PCE deployment models

Composite PCE: The PCE is hosted locally on an LSR. The PCE reads the TED and controls the LSP signaling within one box. I don't believe this is supported on XR as it would effectively turn the router into a TE controller.

External PCE: The PCE is hosted on an external platform like a server or out of band LSR. The TED is learned via BGP-LS and the PCE controls the LSP signaling using PCE protocol (PCEP) to control the PCCs. This seems like an appropriate model for a small SP or a test setup.

Multiple PCE: Typically uses external PCEs interconnected to share information, either for availability or scalability. This would be appropriate for a large, distributed SP.

The lab for this feature is very basic. In this case, the PCE source address for the TCP session is XRv12 loopback, and two peers are defined. The precedence value is self explanatory and lower numbers indicate higher priority. 14.14.14.14/32 is XRv14's loopback (remember, we have a BGP-LS peering with it, since it's our imitation PCE) and 15.15.15.15/32 is a non-existent address just for demonstration. PCE is operating in stateful mode with delegation of TE tunnels. The instantiation keyword allows the PCE to create new dynamic tunnels. The last stanza is outside of the PCE hierarchy but allows the PCC (this router) to generate auto-tunnels in a specified index range. Auto-tunnels come in many varieties and are discussed in detail in the MPLS TE section of this book.

```
! XRv12
mpls traffic-eng
  auto-tunnel pcc
    tunnel-id min 100 max 199
  pce
    peer source ipv4 12.12.12.12
    peer ipv4 14.14.14.14
      precedence 1

    peer ipv4 15.15.15.15
      precedence 2

  logging events peer-status
  stateful-client
  instantiation
  delegation
```

When we verify our PCE sessions, we see the session is pending, since we have no PCE. The remaining show commands for PCE are related to specific LSPs and tunnels, of which we have zero. I didn't actually set up the entire PCE environment as I believe it is beyond the scope of the CCIE SP exam.

```
RP/0/0/CPU0:XRv12#show mpls traffic-eng pce peer
```

Address	Precedence	State	Learned From
14.14.14.14	1	TCP Pending	Static config
15.15.15.15	2	TCP Pending	Static config

Additional Reading – Reference configurations "bgp-ls"

28. Describe, implement, and troubleshoot MVPN

This section walks through many advanced MVPN configurations. The topology is shown below; XRv3 and XRv4 are core routers, with the other six as VPNv4/v6 PEs. XRv2 is a BGP RR for VPNv4/v6 and MVPNv4/v6 (or IPv4/v6 MVPN, same thing). I tried to keep XRv away from the edge since its MVPN capabilities are limited, so most of the testing will occur on the five CSR PEs. CSR10 is the RP for all ASM groups, while the other nodes are senders and/or receivers. The SSM range of 232.0.0.0/8 is also used by the MVPN clients. All 6 sites have full IPv4 unicast reachability, and IPv6 reachability is limited to the top three CEs and bottom three CEs in a segmented fashion. The core runs IS-IS level 2 with MPLS TE enabled on all links.

For all profiles, the following (*,G) and (S,G) streams are joined by the specified CE devices:

XRv1: (, 225.0.0.1)*

CSR10: (2001:10:4:6::4, FF33::1),

CSR9: (, 225.0.0.1), (2001:10:4:6::4, FF33::1)*

CSR4: (, 225.0.0.1), (10.2.7.2, 232.0.0.5)*

CSR3: (, 225.0.0.1), (10.2.7.2, 232.0.0.5), (2001:10:4:6::4, FF33::1), (*, FF7E:240:2001:10:2:7:0:1)*

CSR2: (2001:10:4:6::4, FF33::1)

Anywhere MDT default groups are used, the format is as follows:

First octet: 239 for ASM, 238 for Bidir, and 232 for SSM. Remaining octets: X.255.0.1

Anywhere MDT data groups are used, the format is as follows:

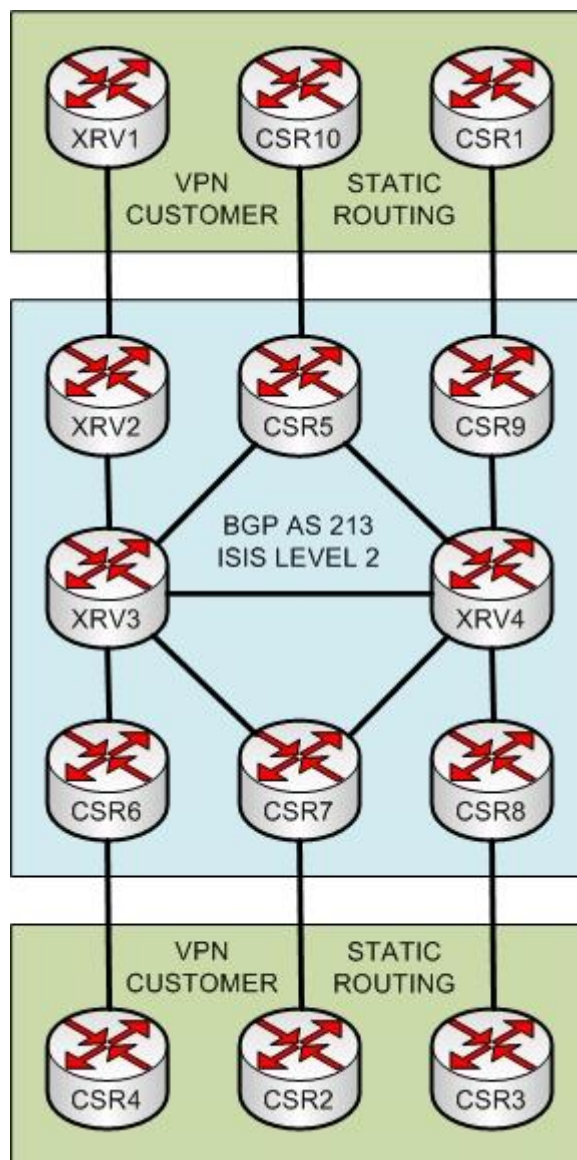
First octet: 239 for ASM, 238 for Bidir, and 232 for SSM

Second octet: 4 for IPv4, 6 for IPv6

Third octet: Router number (1, 5, 6, 7, 8, 12) for each PE

Fourth octet: Variable, 0 - 255

The network diagram used in all profiles is shown below. The routers within the box make up the SP core. XRv2 is the BGP RR for all AFI that are in use. The 6 routers not inside the box are CE devices. Each of them has an IPv4/IPv6 static default route pointing towards the PEs, while the PEs redistribute connected PE-CE links into BGP for VPNv4/v6 reachability. VPNv4 and VPNv6 are always enabled. MVPNv4, MVPNv6, and IPv4 MDT will be enabled based on the profile being tested. The SP core runs PIM only for non-LSM profiles as a best practice.



28.1 Profile 0: Default MDT – GRE – PIM C–mcast Signaling (Traditional Draft-Rosen)

Because the type of PIM used in the core can vary and Profile 0 generally recommends SSM, we will look at all of the options. The baseline configuration has XRv4 as the RP for 239.0.0.0/8 groups using ASM and 238.0.0.0/8 groups using PIM-Bidir. BGP IPv4 MDT and IPv4/v6 MVPN families are disabled everywhere. PIM-SM is enabled on all transit links and MDT endpoints on PEs. The basic multicast configuration is not shown in this section. Rather than show the same basic verification over and over for the different flavors of profile 0, the baseline verifications are done here. First, we will check PIM neighbor counts on XRv3 and XRv4; because there are no neighbor filters, I make the assumption that the neighbors are bidirectional with all the PEs for brevity. There are 2 neighbors per link because XR considers the its own interface in the link (also why the loopback0 shows 1 neighbor as well).

```
RP/0/0/CPU0:XRv3#show pim neighbor count
PIM neighbors in VRF default
```

Interface	Nbr count
GigabitEthernet0_0_0_0.523	2
GigabitEthernet0_0_0_0.534	2
GigabitEthernet0_0_0_0.553	2
GigabitEthernet0_0_0_0.563	2
GigabitEthernet0_0_0_0.573	2
AMT	1
Loopback0	1

RP/0/0/CPU0:XRv4#show pim neighbor count

PIM neighbors in VRF default

Interface	Nbr count
GigabitEthernet0_0_0_0.514	2
GigabitEthernet0_0_0_0.534	2
GigabitEthernet0_0_0_0.554	2
GigabitEthernet0_0_0_0.574	2
GigabitEthernet0_0_0_0.584	2
AMT	1
Loopback0	1

Next, I check the status of the RP on a few select nodes. 239.0.0.0/8 is the ASM range, 238.0.0.0/8 is the Bidir range, and 232.0.0.0/8 is the default SSM range (no RP required). XRv4, the BSR/RP, shows the ranges. I spot-check XRv3 and CSR8 to ensure they learned the mappings properly, and are using the right type of PIM for each RP (ASM or Bidir). This completes the basic multicast transport verification for the core network.

RP/0/0/CPU0:XRv4#show pim bsr rp-cache

PIM BSR Candidate RP Cache

Group(s) 238.0.0.0/8, RP count 1

RP-addr	Proto	Priority	Holdtime(s)	Uptime	Expires
213.14.14.14	BD	192	150	00:06:27	00:02:02

Group(s) 239.0.0.0/8, RP count 1

RP-addr	Proto	Priority	Holdtime(s)	Uptime	Expires
213.14.14.14	SM	192	150	00:06:27	00:02:02

RP/0/0/CPU0:XRv3#show pim rp mapping

PIM Group-to-RP Mappings

Group(s) 238.0.0.0/8

RP 213.14.14.14 (?), v2, bidir

Info source: 213.13.14.14 (?), elected via bsr, priority 192, holdtime 150

Uptime: 00:07:36, expires: 00:02:08

Group(s) 239.0.0.0/8

RP 213.14.14.14 (?), v2

Info source: 213.13.14.14 (?), elected via bsr, priority 192, holdtime 150

Uptime: 00:07:36, expires: 00:02:08


```

R8#show ip pim rp mapping
PIM Group-to-RP Mappings
Group(s) 238.0.0.0/8
  RP 213.14.14.14 (?), v2, bidir
    Info source: 213.14.14.14 (?), via bootstrap, priority 192, holdtime 150
    Uptime: 00:07:57, expires: 00:01:44
Group(s) 239.0.0.0/8
  RP 213.14.14.14 (?), v2
    Info source: 213.14.14.14 (?), via bootstrap, priority 192, holdtime 150
    Uptime: 00:07:57, expires: 00:01:44

```

28.1.1 PIM-ASM in the core

This variation starts with the baseline configuration and does not require any modification. The default MDT for VRF MC uses group 239.255.0.1 which, at present, is an ASM group. This default MDT is used for signaling between the nodes so C-PIM information, such as joins, prunes, hellos, and BSR messages can transit the SP core. Each PE loopback participating in this MDT is a source for group 239.255.0.1. We can check XRv4, the RP, and we should see a P(S,G) state entry for each PE using the default MDT group. Because this yields a lot of output, I will show a summary view. The P(*,G) entry is always in the table provided there is at least one receiver (we will discuss receivers next). Each P(S,G) is duplicated for a shortest path tree (SPT) and an RP tree (RPT, or root path tree).

Note: Cisco documentation clearly states the need to run BGP IPv4 MDT between sites for this profile to work. I found out that is not true as this entire section was done without this feature being configured. This may be because these ASM/Bidir variations are not technically compliant with Cisco's definition of the "profile", but they are functionally valid.

```

RP/0/0/CPU0:XRv4#show pim topology 239.255.0.1 | include 239.255.0.1
(*,239.255.0.1) SM Up: 00:33:44 RP: 213.14.14.14*
(213.1.1.1,239.255.0.1) RPT SM Up: 00:33:44 RP: 213.14.14.14*
(213.1.1.1,239.255.0.1) SPT SM Up: 00:33:31
(213.5.5.5,239.255.0.1) RPT SM Up: 00:33:38 RP: 213.14.14.14*
(213.5.5.5,239.255.0.1) SPT SM Up: 00:33:38
(213.6.6.6,239.255.0.1) RPT SM Up: 00:33:36 RP: 213.14.14.14*
(213.6.6.6,239.255.0.1) SPT SM Up: 00:33:36
(213.7.7.7,239.255.0.1) RPT SM Up: 00:33:26 RP: 213.14.14.14*
(213.7.7.7,239.255.0.1) SPT SM Up: 00:33:26
(213.8.8.8,239.255.0.1) RPT SM Up: 00:33:32 RP: 213.14.14.14*
(213.8.8.8,239.255.0.1) SPT SM Up: 00:33:32
(213.12.12.12,239.255.0.1) RPT SM Up: 00:01:23 RP: 213.14.14.14*
(213.12.12.12,239.255.0.1) SPT SM Up: 00:01:27

```

We will zoom in on CSR8's related state information on the P-RP. The RPT entries represent the state created when a source registered with the RP; this is why the RPF interface is a decapsulation tunnel since that is the mechanism by which "native" multicast is viewed by the router. The SPT is the P(S,G) state we normally see in XE that shows how native multicast is forwarded (not encapsulated register

messages). This traffic is forwarded out all of XRv4's other interfaces towards the remote PE's in the same MDT.

```
RP/0/0/CPU0:XRv4#show pim topology 239.255.0.1 213.8.8.8 | begin RPT SM
(213.8.8.8,239.255.0.1)RPT SM Up: 00:17:22 RP: 213.14.14.14*
JP: Prune(never) RPF: Decapstunnel0,213.14.14.14 Flags: KAT(00:02:42) RA RR
(00:03:45)
GigabitEthernet0/0/0/0.584 00:17:22 off Prune(00:02:35)

(213.8.8.8,239.255.0.1)SPT SM Up: 00:17:22
JP: Join(00:00:25) RPF: GigabitEthernet0/0/0/0.584,213.8.14.8 Flags:
KAT(00:02:42) RA RR (00:03:45)
GigabitEthernet0/0/0/0.514 00:16:18 fwd Join(00:02:55)
GigabitEthernet0/0/0/0.534 00:17:17 fwd Join(00:03:12)
GigabitEthernet0/0/0/0.554 00:16:18 fwd Join(00:02:57)
GigabitEthernet0/0/0/0.574 00:16:18 fwd Join(00:02:53)
```

The reason XRv4 knows the forward the traffic to all receivers is because each PE loopback is also a receiver for the same group. This is part of the reason why PIM must be enabled on the MDT source; an IGMP join is automatically applied to that interface, which triggers a PIM P(*,G) join towards the SP-core's RP for that group. For completeness, I show some debug from CSR8 which shows the periodic join. CSR8 also tells the RP that it doesn't want traffic from the source 213.8.8.8, which is itself. All of the other PEs perform similar operations.

```
R8#show ip igmp groups 239.255.0.1
IGMP Connected Group Membership
Group Address      Interface          Uptime    Expires    Last Reporter
Group Accounted
239.255.0.1        Loopback0          00:40:49  stopped    213.8.8.8

! CSR8
PIM(0): Building Periodic (*,G) Join / (S,G,RP-bit) Prune message for
239.255.0.1
PIM(0): Insert (*,239.255.0.1) join in nbr 213.8.14.14's queue
PIM(0): Insert (213.8.8.8,239.255.0.1) sgr prune in nbr 213.8.14.14's queue
PIM(0): Building Join/Prune packet for nbr 213.8.14.14
```

Focusing on CSR8, we can check its global MRIB to see how it treats this default MDT group. The summary view gives us most of the information, and I remove some of the expanded output because many entries are nearly identical. We can see the big 'Z' flag set on these groups which represents a multicast tunnel. The last group is the locally originated traffic which was locally registered (big 'F') after arriving on the loopback0 (MDT source). The OIL for these tunnels includes the MVRF MC, which is the VRF to which this MDT is bound.

```
R8#show ip mroute 239.255.0.1 summary | begin \((
```

```
(*, 239.255.0.1), 00:52:33/stopped, RP 213.14.14.14, OIF count: 1, flags:
SJCFZ
(213.12.12.12, 239.255.0.1), 00:00:09/00:02:50, OIF count: 1, flags: JTZ
(213.5.5.5, 239.255.0.1), 00:31:41/00:02:36, OIF count: 1, flags: JTZ
(213.1.1.1, 239.255.0.1), 00:31:42/00:01:47, OIF count: 1, flags: JTZ
(213.7.7.7, 239.255.0.1), 00:31:54/00:02:43, OIF count: 1, flags: JTZ
(213.6.6.6, 239.255.0.1), 00:32:12/00:01:44, OIF count: 1, flags: JTZ
(213.8.8.8, 239.255.0.1), 00:32:17/00:02:03, OIF count: 1, flags: FT
```

```
R8#show ip mroute 239.255.0.1 | begin \
(*, 239.255.0.1), 00:53:05/stopped, RP 213.14.14.14, flags: SJCFZ
Incoming interface: GigabitEthernet2.584, RPF nbr 213.8.14.14
Outgoing interface list:
MVRF MC, Forward/Sparse, 00:53:03/stopped
```

```
(213.12.12.12, 239.255.0.1), 00:00:40/00:02:19, flags: JTZ
Incoming interface: GigabitEthernet2.584, RPF nbr 213.8.14.14
Outgoing interface list:
MVRF MC, Forward/Sparse, 00:00:40/00:02:19
```

[snip]

```
(213.8.8.8, 239.255.0.1), 00:27:24/00:02:56, flags: FT
Incoming interface: Loopback0, RPF nbr 0.0.0.0
Outgoing interface list:
GigabitEthernet2.584, Forward/Sparse, 00:27:24/00:03:04
```

With the multicast tunnels operational, we can check the C-PIM information to verify we have PIM neighbors across the tunnel (in the overlay) with the other five PEs. This PIM signaling is always carried by the default MDT and cannot be optimized into a data MDT, because it needs to go to all nodes. The default MDT is often called an “emulated LAN” since all of the multi-access PIM mechanisms, such as Assert, will occur over this tunnel mesh.

```
R8#show ip pim vrf MC neighbor | begin ^Neighbor
Neighbor          Interface          Uptime/Expires    Ver    DR
Address
213.12.12.12      Tunnel2            00:04:22/00:01:21 v2     1 / DR B P G
213.7.7.7         Tunnel2            00:35:59/00:01:17 v2     1 / B S P G
213.5.5.5         Tunnel2            00:36:29/00:01:17 v2     1 / B S P G
213.1.1.1         Tunnel2            00:36:29/00:01:18 v2     1 / B S P G
213.6.6.6         Tunnel2            00:36:29/00:01:19 v2     1 / B S P G
```

The C-MRIB on CSR8 shows two interesting entries currently. The first is the IGMPv3 join for the SSM pair. The big ‘Y’ is set, meaning this group should be receiving traffic on a data MDT. We can specifically see a new data MDT was spawned using group 239.4.7.0 specifically to carry this traffic. The third octet represents the router that is sending the traffic, so we can see the CSR7 is the root (source) of this flow. This occurs because this C(S,G) was matched by the ACL defining what traffic was candidate for data

MDT transport. The second entry is the C(*,G) entry for customer ASM. CSR10 is the C-RP, and this information was learned via BSR across the default MDT. There hasn't been any traffic to this group yet.

```
R8#show ip mroute vrf MC | begin \  
(10.2.7.2, 232.0.0.5), 00:00:43/00:02:27, flags: sTIY  
  Incoming interface: Tunnel2, RPF nbr 213.7.7.7, MDT:239.4.7.0/00:02:26  
  Outgoing interface list:  
    GigabitEthernet2.538, Forward/Sparse, 00:00:33/00:02:27  
  
(* , 225.0.0.1), 00:08:05/00:02:16, RP 10.5.10.10, flags: SJC  
  Incoming interface: Tunnel2, RPF nbr 213.5.5.5  
  Outgoing interface list:  
    GigabitEthernet2.538, Forward/Sparse, 00:08:05/00:02:16
```

For verification, we will confirm our understanding of the data MDT by checking CSR6. It should have joined the same data MDT group.

```
R6#show ip mroute vrf MC 232.0.0.5 10.2.7.2 | begin \  
(10.2.7.2, 232.0.0.5), 00:00:38/00:02:43, flags: sTIY  
  Incoming interface: Tunnel0, RPF nbr 213.7.7.7, MDT:239.4.7.0/00:02:43  
  Outgoing interface list:  
    GigabitEthernet2.546, Forward/Sparse, 00:00:38/00:02:43
```

It is not a coincidence that both routers knew to pick the same group. This was carried inside the MDT signaling in the overlay. Although the debugs don't reveal the contents of the overlay, we can see that it happened. We can also look at the send/receive MDT information in a separate table. CSR7 was the originator, and CSR6 /CSR8 were the receivers.

```
! CSR6  
PIM(1): Receive MDT Packet (13613) from 213.7.7.7 (Tunnel0), length (ip: 44,  
udp: 24), ttl: 1  
PIM(1): TLV type: 1 length: 16 MDT Packet length: 16
```

```
R6#show ip pim mdt receive  
Joined MDT-data [group/mdt number : source]  uptime/expires for VRF: MC  
[239.4.7.0 : 213.7.7.7]  00:14:54/00:02:05
```

```
R8#show ip pim mdt receive  
Joined MDT-data [group/mdt number : source]  uptime/expires for VRF: MC  
[239.4.7.0 : 213.7.7.7]  00:15:06/00:02:53
```

```
R7#show ip pim mdt send  
MDT-data send list for VRF: MC  
  (source, group)                MDT-data group/num  ref_count  
  (10.2.7.2, 232.0.0.5)          239.4.7.0           1
```

An unfortunate drawback of the ASM model is that our data MDTs are also ASM groups. That means even though the overlay signaling was SSM, the underlying transport is not, so hopefully the SP is allowing SPT construction in the core for those flows.

```
R6#show ip mroute 239.4.7.0 | begin \  
(* , 239.4.7.0) , 00:13:51/stopped, RP 213.14.14.14, flags: SJCZ  
  Incoming interface: GigabitEthernet2.563, RPF nbr 213.6.13.13  
  Outgoing interface list:  
    MVRF MC, Forward/Sparse, 00:08:40/stopped
```

Now, we will sent traffic from CSR2 to group 232.0.0.5. We don't expect any additional PIM signaling to occur within the overlay, since the deliver trees have been built. However, in the underlay, CSR7 will have to register this new (S,G) with XRv4, and the SPT construction happens per normal PIM-SM rules. The S remains the MDT source (loopback0), but the G will be the new data MDT group assigned for this flow. XRv4 receives the register and sends a (S,G) join towards the source, CSR7's loopback. XR PIM debugs are extensive so I've only cherry-picked the valuable parts.

```
! CSR7  
PIM(0): Adding register encap tunnel (Tunnel3) as forwarding interface of  
(213.7.7.7, 239.4.7.0).  
PIM(0): Received v2 Join/Prune on GigabitEthernet2.574 from 213.7.14.14, to  
us  
PIM(0): Join-list: (213.7.7.7/32, 239.4.7.0), S-bit set  
PIM(0): Add GigabitEthernet2.574/213.7.14.14 to (213.7.7.7, 239.4.7.0),  
Forward state, by PIM SG Join
```

```
! XRv4  
pim[1152]: [13] VRF : default (213.7.7.7,239.4.7.0) Received Register from  
213.7.14.7 Register count 6, Register limit 20000  
pim[1152]: [13] Route create: VRF default (213.7.7.7,239.4.7.0)  
pim[1152]: [13] VRF : default (213.7.7.7,239.4.7.0) SEND J/P adding Join on  
Gi0/0/0/0.574, target 213.7.14.7  
pim[1152]: [13] Sending J/P message for neighbor 213.7.14.7 on Gi0/0/0/0.574  
for 1 groups, size 34 (MTU=1480)
```

XRv4 decapsulates the register message and sends it down the shared tree. CSR6 and CSR8 receive it and issue (S,G) joins towards CSR7 to leave the shared tree.

```
! CSR6  
PIM(0): Insert (213.7.7.7,239.4.7.0) join in nbr 213.6.13.13's queue  
PIM(0): Building Join/Prune packet for nbr 213.6.13.13  
PIM(0): Adding v2 (213.7.7.7/32, 239.4.7.0), S-bit Join  
PIM(0): Send v2 join/prune to 213.6.13.13 (GigabitEthernet2.563)  
  
! CSR8  
PIM(0): Insert (213.7.7.7,239.4.7.0) join in nbr 213.8.14.14's queue
```

```
PIM(0): Building Join/Prune packet for nbr 213.8.14.14
PIM(0): Adding v2 (213.7.7.7/32, 239.4.7.0), S-bit Join
PIM(0): Send v2 join/prune to 213.8.14.14 (GigabitEthernet2.584)
```

CSR7 receives the SPT join from XRv3 (originated by CSR6) and adds it to its delivery tree.

```
! CSR7
PIM(0): Received v2 Join/Prune on GigabitEthernet2.573 from 213.7.13.13, to
us
PIM(0): Join-list: (213.7.7.7/32, 239.4.7.0), S-bit set
PIM(0): Add GigabitEthernet2.573/213.7.13.13 to (213.7.7.7, 239.4.7.0),
Forward state, by PIM SG Join
```

CSR7 has not been told to stop registering, so it sends another one to the RP, along with native multicast. When XRv4 sees native multicast on the SPT, it issues the register-stop.

```
! CSR7
PIM(0): Received v2 Register-Stop on GigabitEthernet2.574 from 213.14.14.14
PIM(0): for source 213.7.7.7, group 239.4.7.0
PIM(0): Removing register encap tunnel (Tunnel3) as forwarding interface of
(213.7.7.7, 239.4.7.0).
PIM(0): Clear Registering flag to 213.14.14.14 for (213.7.7.7/32, 239.4.7.0)
PIM(0): Building Periodic (*,G) Join / (S,G,RP-bit) Prune message for
239.4.7.0
```

```
! XRv4
pim[1152]: [13] VRF : default (213.7.7.7,239.4.7.0) Received Register from
213.7.14.7 Register count 7, Register limit 20000
pim[1152]: [13] VRF : default (213.7.7.7,239.4.7.0) Send Register-Stop to
213.7.14.7
```

At this point, CSR7 should be delivering (213.7.7.7, 239.4.7.0) out of both interfaces towards CSR6 and CSR8. The little 'z' means this is the outer encapsulation for a data MDT group.

```
R7#show ip mroute 239.4.7.0 213.7.7.7 | begin \
(213.7.7.7, 239.4.7.0), 00:12:07/00:01:48, flags: FTz
Incoming interface: Loopback0, RPF nbr 0.0.0.0
Outgoing interface list:
GigabitEthernet2.573, Forward/Sparse, 00:12:05/00:03:19
GigabitEthernet2.574, Forward/Sparse, 00:12:07/00:03:22
```

We already know there is no new signaling in the overlay. Now, we can check packet counters on all three PEs; CSR7 sends packets to the MDT while CSR6 and CSR8 receive it from the MDT.

```
R6#show ip mroute 239.4.7.0 213.7.7.7 count | begin ^Group
Group: 239.4.7.0, Source count: 1, Packets forwarded: 412, Packets received:
412
```

```
Source: 213.7.7.7/32, Forwarding: 411/0/142/0, Other: 411/0/0
```

```
R7#show ip mroute 239.4.7.0 213.7.7.7 count | begin ^Group
Group: 239.4.7.0, Source count: 1, Packets forwarded: 409, Packets received:
410
```

```
Source: 213.7.7.7/32, Forwarding: 409/0/124/0, Other: 410/1/0
```

```
R8#show ip mroute 239.4.7.0 213.7.7.7 count | begin ^Group
Group: 239.4.7.0, Source count: 1, Packets forwarded: 406, Packets received:
406
```

```
Source: 213.7.7.7/32, Forwarding: 405/0/142/0, Other: 405/0/0
```

Fortunately, native multicast counters work in XRv. We can check the packet counters on XRv3 and XRv4 for extra verification. The “A” flag means accept (inbound), the “EG” flag means egress (outbound), and the “NS” flag means negate signal (not related to this).

```
RP/0/0/CPU0:XRv3#show mfib route 239.4.7.0 sources-only | begin 213
(213.7.7.7,239.4.7.0), Flags:
Up: 00:15:41
Last Used: 00:00:00
SW Forwarding Counts: 472/470/58280
SW Replication Counts: 472/470/58280
SW Failure Counts: 1/0/0/0/0
GigabitEthernet0/0/0/0.563 Flags: NS EG, Up:00:15:41
GigabitEthernet0/0/0/0.573 Flags: A, Up:00:15:41
```

```
RP/0/0/CPU0:XRv4#show mfib route 239.4.7.0 sources-only | begin 213
(213.7.7.7,239.4.7.0), Flags:
Up: 00:16:11
Last Used: 00:00:00
SW Forwarding Counts: 486/485/60140
SW Replication Counts: 486/487/60388
SW Failure Counts: 0/0/0/0/0
GigabitEthernet0/0/0/0.574 Flags: A, Up:00:16:11
GigabitEthernet0/0/0/0.584 Flags: NS EG, Up:00:16:11
```

CSR6 will perform a final verification using EPC. It will capture traffic entering from the SP core (multicast ICMP inside GRE) and also exiting towards the customer (multicast ICMP only). Since the capture will look at both interfaces concurrently, the timestamps for the two packets should be very close together. It looks like the router was extremely fast in switching this packet (no time difference). In both packets, I highlighted the ICMP packet source/destination addresses. We can see the difference in length is exactly 24 (142-118=24), the length of a basic GRE header. This further strengthens our confidence that it is working.

```
3 142 2.000992 213.7.7.7 -> 239.4.7.0 GRE
0000: 01005E04 07000050 56A9DB37 8100CDEB ..^....PV..7....
0010: 08004500 007C024B 0000FE2F E7F4D507 ..E...|.K.../....
```

```

0020: 0707EF04 07000000 08004500 00641CB3 .....E..d..
0030: 0000FE01 A6DC0A02 0702E800 00050800 .....

 4 118 2.000992 10.2.7.2 -> 232.0.0.5 ICMP
0000: 01005E00 00050050 56A9DE0D 8100DDA ..^....PV.....
0010: 08004500 00641CB3 0000FD01 A7DC0A02 ..E..d.....
0020: 0702E800 00050800 3A150007 02420000 .....:.....B..
0030: 0000142C 2DC0ABCD ABCDABCD ABCDABCD ...,~.....

```

Next we will examine IPv6 ASM. This is significantly more complex because we are doing ASM in both the customer and SP networks. XE routers only allowing debugging in one VRF at a time, so because we already examined what happens in the SP network, we will look at the C-PIM signaling for customer ASM traffic with IPv6. CSR2 is the embedded RP and only CSR3 is a receiver for the group specified below.

```

R8#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 | begin \
(*, FF7E:240:2001:10:2:7:0:1), 01:00:00/never, RP 2001:10:2:7::2, flags: SCJ
Incoming interface: Tunnel2
RPF nbr: ::FFFF:213.7.7.7
Immediate Outgoing interface list:
GigabitEthernet2.538, Forward, 01:00:00/never

```

Quickly checking for IPv6 PIM neighbors, we see that no PE has any neighbors over the MDT. Below is a spot check.

```

RP/0/0/CPU0:XRv2#show pim vrf MC ipv6 neighbor | begin mdt
mdtMC
Neighbor Address          Uptime    Expires   DR pri   DR
Flags
::ffff:213.12.12.12*      01:21:53  00:01:44  1        (DR)    P

```

```

R6#show ipv6 pim vrf MC neighbor
No neighbors found.

```

```

R1#show ipv6 pim vrf MC neighbor
No neighbors found.

```

```

R8#show ipv6 pim vrf MC neighbor
No neighbors found.

```

Cisco requires IPv6 multicast enabled in the global process also; when this is enabled globally, it is enabled for all interfaces by default. Be sure to manually remove it (no ipv6 pim) from core-facing links so it only remains enabled on the loopback0 (MDT source). You can quickly check to see which interfaces have PIMv6 enabled. Below is a spot check of some routers, along with their newly-built PIM adjacencies.


```
RP/0/0/CPU0:XRv2#show pim vrf MC ipv6 neighbor | begin mdt
mdtMC
Neighbor Address                               Uptime    Expires   DR pri   DR
Flags
::ffff:213.1.1.1                             00:02:39  00:01:16  1
::ffff:213.5.5.5                             00:01:57  00:01:19  1
::ffff:213.6.6.6                             00:12:15  00:01:17  1
::ffff:213.7.7.7                             00:02:09  00:01:17  1
::ffff:213.8.8.8                             00:02:54  00:01:16  1
::ffff:213.12.12.12*                         01:52:26  00:01:34  1          (DR)  P

RP/0/0/CPU0:XRv2#show pim interface state-on
PIM interfaces in VRF default
Address                               Interface                                PIM  Nbr    Hello   DR      DR
Count Intvl Prior
213.12.12.12                         Loopback0                             on   1      30      1
this system
213.12.13.12                         GigabitEthernet0/0/0/0.523            on   2      30      1
213.12.13.13

R8#show ipv6 pim vrf MC neighbor
PIM Neighbor Table
Mode: B - Bidir Capable, G - GenID Capable
Neighbor Address                               Interface                                Uptime    Expires   Mode DR pri
::FFFF:213.1.1.1                             Tunnel2                                00:03:26  00:01:29  B G      1
::FFFF:213.5.5.5                             Tunnel2                                00:02:44  00:01:31  B G      1
::FFFF:213.6.6.6                             Tunnel2                                00:03:40  00:01:28  B G      1
::FFFF:213.7.7.7                             Tunnel2                                00:02:57  00:01:29  B G      1
::FFFF:213.12.12.12                         Tunnel2                                00:03:37  00:01:16  B G      DR 1

R8#show ipv6 pim int
R8#show ipv6 pim interface state-on
Interface                                PIM  Nbr    Hello   DR
Count Intvl Prior
Loopback0                             on   0      30      1
Address: FE80::21E:E6FF:FE4D:4D00
DR      : this system
```

An interesting behavior of the CSR is that if the PIM DR is within a VRF and the RP is embedded, the register message never gets sent. Since CSR4 will be the source, we will tell CSR6 to use a static RP. We know embedded RP works from the receiver side since CSR2 has the C(*,G) join from CSR8. IPv6 (*,G) messages have an incoming interface of the PIM decapsulation tunnel for register messages as opposed to "null" for IPv4. The logic is the same, though.

```
! CSR6
ipv6 pim vrf MC rp-address 2001:10:2:7::2
```

```
ipv6 pim vrf MC register-source GigabitEthernet2.546
no ipv6 pim vrf MC rp embedded
```

```
R2#show ipv6 mroute FF7E:240:2001:10:2:7:0:1 | begin \
(*, FF7E:240:2001:10:2:7:0:1), 00:04:30/00:02:58, RP 2001:10:2:7::2, flags: S
  Incoming interface: Tunnel2
  RPF nbr: 2001:10:2:7::2
  Immediate Outgoing interface list:
    GigabitEthernet2.527, Forward, 00:04:30/00:02:58
```

```
R2#show ipv6 pim tunnel tunnel2
Tunnel2*
  Type   : PIM Decap
  RP     : 2001:10:2:7::2*
  Source: -
```

CSR4 will be sending traffic to this group. We expect CSR6 to register the source with CSR2, who will decapsulate the register message and send it down the shared tree, then immediately join the SPT. Given the location of CSR2 in the network, it sends a (S,G) prune to CSR7 instead, which is sufficient to build the (S,G) state there.

```
! CSR6
IPv6 PIM: [MC] (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) Start registering to
2001:10:2:7::2

! CSR2
IPv6 PIM: J/P entry: Prune root: 2001:10:4:6::4 group:
FF7E:240:2001:10:2:7:0:1 Sender: FE80::7 flags: RPT S
IPv6 PIM: (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1)RPT Create entry
(2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) J/P adding Prune on
GigabitEthernet2.527
```

Upon receiving the traffic, CSR8 will join the SPT by sending a PIMv6 join directed to CSR6 over the default MDT.

```
! CSR8
IPv6 PIM: [MC] (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) J/P adding Join on
Tunnel2
```

Whether triggered by CSR7's SPT join from CSR2 or CSR8's SPT join, CSR6 will switch over to the data MDT since this group is candidate for optimization. CSR6 will eventually receive joins from both. At this point, CSR6 sends multicast both as register messages to the C-RP and as native multicast.

```
! CSR6
IPv6 PIM: [MC] J/P entry: Join root: 2001:10:4:6::4 group:
FF7E:240:2001:10:2:7:0:1 Sender: ::FFFF:213.7.7.7 flags:
```

```
IPv6 PIM: [MC] J/P entry: Join root: 2001:10:2:7::2 group:
FF7E:240:2001:10:2:7:0:1 Sender: ::FFFF:213.8.8.8 flags:
```

CSR2 receives another register then tells CSR6 to stop registering. CSR6 stops sending traffic down its PIM encapsulation tunnel.

```
! CSR2
IPv6 PIM: (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) Received Register from
2001:10:4:6::6
IPv6 PIM: (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) Send Register-Stop to
2001:10:4:6::6
```

```
! CSR6
IPv6 PIM: [MC] (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) Received Register-
Stop
IPv6 PIM: [MC] (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) Stop registering
IPv6 PIM: [MC] (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) Tunnel5 J/P state
changed from Join to Null
IPv6 PIM: [MC] (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) Tunnel5 FWD state
change from Forward to Prune
```

```
R6#show ipv6 pim vrf MC tunnel
Tunnel5*
Type : PIM Encap
RP : 2001:10:2:7::2
Source: 2001:10:4:6::6
```

At this point, CSR6 and CSR8 should have valid C(S,G) information. CSR6 claims to be forwarding traffic to multiple tunnels, but Tunnel2 doesn't exist. I am assuming this feature is not supported fully in a virtual environment. The router says it is forwarding to an MDT data group, which is correct (little 'y'). CSR8 specifically lists the MDT to which this C(S,G) is bound, which uses the SP-core group 239.6.6.0, rooted at CSR6.

```
R6#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 2001:10:4:6::4 | begin \
(2001:10:4:6::4, FF7E:240:2001:10:2:7:0:1), 00:31:35/00:00:07, flags: SFTy
Incoming interface: GigabitEthernet2.546
RPF nbr: 2001:10:4:6::4
Immediate Outgoing interface list:
Tunnel2, Forward, 00:31:35/never
Tunnel0, Forward, 00:20:38/00:03:08
```

```
R6#show ip interface brief | include Tunnel
Tunnel0          213.6.6.6        YES unset up      up
Tunnel1          unassigned       YES unset up      up
Tunnel3          213.6.13.6       YES unset up      up
Tunnel4          10.4.6.6         YES unset up      up
Tunnel5          unassigned       YES unset up      up
```

```
R8#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 2001:10:4:6::4 | begin \
(2001:10:4:6::4, FF7E:240:2001:10:2:7:0:1), 00:17:47/now, flags: SJY
Incoming interface: Tunnel2 MDT: 239.6.6.0/00:02:23
RPF nbr: ::FFFF:213.6.6.6
Inherited Outgoing interface list:
GigabitEthernet2.538, Forward, 02:30:53/never
```

A quick check of the packet counters indicates traffic is flowing from end to end. The reason for the 280 failures was a failure on my part; the “issue” I described regarding anycast RP in a VRF slowed me down, and I did not clear counters once I got things working. You can see the difference in packet size is exactly 24 (142 – 118) bytes also.

```
R6#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 2001:10:4:6::4 count |
include HW
HW Forwarding: 903/0/118/0, Other: 280/0/280
```

```
R8#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 2001:10:4:6::4 count |
include HW
HW Forwarding: 895/0/142/0, Other: 0/0/0
```

In the underlay, CSR6 has spawned a data MDT for this flow, which was carried over the default MDT in the PIM signaling information. CSR6 sends the MDT information and CSR8 receives it.

```
R6#show ipv6 pim mdt send
MDT-data send list for VRF: MC
  (source, group)                MDT-data group/num  ref_count
  (2001:10:4:6::4,
  FF7E:240:2001:10:2:7:0:1)      239.6.6.0          1
```

```
R8#show ipv6 pim mdt receive
Joined MDT-data [group/mdt number : source]  uptime/expires for VRF: MC
[239.6.6.0 : 213.6.6.6]  00:25:50/00:02:14
```

Quickly checking XRv3 and XRv4, we can see this new MDT group exists in the core and appears to be functional.

```
RP/0/0/CPU0:XRv3#show mfib route 239.6.6.0 sources-only | begin 213
(213.6.6.6,239.6.6.0),  Flags:
Up: 00:31:48
Last Used: 00:00:00
SW Forwarding Counts: 954/954/118296
SW Replication Counts: 954/1907/236468
SW Failure Counts: 0/0/0/0/0
GigabitEthernet0/0/0/0.534 Flags:  NS EG, Up:00:31:48
GigabitEthernet0/0/0/0.563 Flags:  A, Up:00:31:48
GigabitEthernet0/0/0/0.573 Flags:  NS EG, Up:00:31:46
```

```
RP/0/0/CPU0:XRv4#show mfib route 239.6.6.0 sources-only | begin 213
(213.6.6.6,239.6.6.0),    Flags:
  Up: 00:32:09
  Last Used: 00:00:00
  SW Forwarding Counts: 965/964/119536
  SW Replication Counts: 965/966/119784
  SW Failure Counts: 0/0/0/0/0
  GigabitEthernet0/0/0/0.534 Flags:  A, Up:00:32:09
  GigabitEthernet0/0/0/0.584 Flags:  NS EG, Up:00:32:09
```

We will configure EPC on CSR8 to look at GRE multicast incoming from the core and ICMP multicast going out towards the customer. Like with IPv4 SSM, the capture should show both packets almost simultaneously. The first packet is the GRE packet (0x2F is protocol 47) and the P(S,G) is (213.6.6.6, 239.6.6.0). This is how the service provider transports the packet across the core. The GRE payload is 0x86DD which is IPv6 that has C(S,G) equal to (2001:10:4:6::4, FF7E:240:2001:10:2:7:0:1). The second packet is identical to the first except without the outer IPv4/GRE headers. If you count the difference in length, you will get 24 (0x8E - 0x76 = 0x18 = 24).

```
R8#show monitor capture CAP buffer dump
0
0000: 01005E06 06000050 56A9DE77 8100CE00 ..^....PV..w....
0010: 08004500 007C0C58 0000FD2F E0E7D506 ..E...|.X.../....
0020: 0606EF06 06000000 86DD6000 0000003C .....`....<
0030: 3A3F2001 00100004 00060000 00000000 :? .....
0040: 0004FF7E 02402001 00100002 00070000 ...~.@ .....
0050: 00018000 AE3A0B3D 047F7F80 81828384 .....:.=.....
0060: 85868788 898A8B8C 8D8E8F90 91929394 .....
0070: 95969798 999A9B9C 9D9E9FA0 A1A2A3A4 .....
0080: A5A6A7A8 A9AAABAC ADAEAFB0 B1B2 .....

1
0000: 33330000 00010050 56A9FB1C 8100DD2 33.....PV.....
0010: 86DD6000 0000003C 3A3F2001 00100004 ..`....<:? .....
0020: 00060000 00000000 0004FF7E 02402001 .....~.@ .
0030: 00100002 00070000 00018000 AE3A0B3D .....:.=.....
0040: 047F7F80 81828384 85868788 898A8B8C .....
0050: 8D8E8F90 91929394 95969798 999A9B9C .....
0060: 9D9E9FA0 A1A2A3A4 A5A6A7A8 A9AAABAC .....
0070: ADAEAFB0 B1B2 .....
```

Additional Reading – Reference configurations “mvpn-0-asm”

28.1.2 PIM-SSM in the core

This variation starts with the baseline configuration and will require some additional BGP features, along with MDT updates. First, we must configure the IPv4 MDT AFI on all PEs. Once BGP converges with this new AF, we see routing information that looks like a normal IPv4 prefix. Normally this MDT AFI is used

for auto-discovery of MDT endpoints for SSM. The prefix is the MDT source and the MDT default group is carried inside the message; it's very straightforward and much less involved than LSM BGP-AD. This is just a way for MDT endpoints to discover one another without relying the ASM mechanism (i.e., the RP).

```
! Basic BGP MDT configuration on XE, apply to all PEs
router bgp 213
  address-family ipv4 mdt
  neighbor 213.12.12.12 activate

! XR2
router bgp 213
  address-family ipv4 mdt
  neighbor-group IBGP
  address-family ipv4 mdt
  route-reflector-client
```

Before continuing, we ensure the MDT capability was negotiated between neighbors. The fastest way to check this is on the RR, XRv2. It should be advertised and received for all five peers. We won't look at any of the routing information yet.

```
RP/0/0/CPU0:XRv2#show bgp ipv4 mdt neighbors | utility egrep 'MDT:|^BGP nei'
BGP neighbor is 213.1.1.1
  Address family IPv4 MDT: advertised and received
BGP neighbor is 213.5.5.5
  Address family IPv4 MDT: advertised and received
BGP neighbor is 213.6.6.6
  Address family IPv4 MDT: advertised and received
BGP neighbor is 213.7.7.7
  Address family IPv4 MDT: advertised and received
BGP neighbor is 213.8.8.8
  Address family IPv4 MDT: advertised and received
```

Next, we will update the MDT default/data group addresses to be within the SSM range. We will swap the first octet from 239 to 232, leaving everything else the same. Be sure to remove the default MDT configuration from all address-families before adding new ones.

```
! Changing from ASM to SSM on all XE PEs
vrf definition MC
  address-family ipv4
    no mdt default 239.255.0.1
    mdt default 232.255.0.1
    mdt data 232.4.1.0 0.0.0.255 list ACL_DATA_MDT
  address-family ipv6
    no mdt default 239.255.0.1
    mdt default 232.255.0.1
    mdt data 232.6.1.0 0.0.0.255 list ACL_DATA_MDT_IPV6
```

```

! Changing from ASM to SSM on XRv2
multicast-routing
vrf MC
  address-family ipv4
    no mdt default ipv4 239.255.0.1
    mdt default ipv4 232.255.0.1
    no mdt data 239.4.12.0/24 ACL_DATA_MDT immediate-switch
    mdt data 232.4.12.0/24 ACL_DATA_MDT immediate-switch
  !
  address-family ipv6
    no mdt default ipv4 239.255.0.1
    mdt default ipv4 232.255.0.1
    no mdt data 239.6.12.0/24 ACL_DATA_MDT_IPV6 immediate-switch
    mdt data 232.6.12.0/24 ACL_DATA_MDT_IPV6 immediate-switch

```

A quick check of the PIM neighbors on a few PEs will validate the construction of the default MDT.

```

RP/0/0/CPU0:XRv2#show pim vrf MC neighbor | begin mdt
213.1.1.1                mdtMC                00:12:17  00:01:40  1          P
213.5.5.5                mdtMC                00:11:38  00:01:42  1          P
213.6.6.6                mdtMC                00:11:15  00:01:42  1          P
213.7.7.7                mdtMC                00:11:02  00:01:41  1          P
213.8.8.8                mdtMC                00:10:52  00:01:42  1          P
213.12.12.12*           mdtMC                00:12:52  00:01:26  1 (DR)    P

```

```

R7#show ip pim vrf MC neighbor | begin ^Neighbor
Neighbor      Interface      Uptime/Expires      Ver      DR
Address
213.8.8.8      Tunnel2      00:11:22/00:01:41  v2       1 / B S P G
213.12.12.12   Tunnel2      00:11:28/00:01:25  v2       1 / DR B P G
213.1.1.1      Tunnel2      00:11:32/00:01:40  v2       1 / B S P G
213.5.5.5      Tunnel2      00:11:32/00:01:41  v2       1 / B S P G
213.6.6.6      Tunnel2      00:11:32/00:01:41  v2       1 / B S P G

```

```

R1#show ip pim vrf MC neighbor | begin ^Neighbor
Neighbor      Interface      Uptime/Expires      Ver      DR
Address
213.8.8.8      Tunnel2      00:11:38/00:01:25  v2       1 / B S P G
213.7.7.7      Tunnel2      00:11:48/00:01:24  v2       1 / B S P G
213.6.6.6      Tunnel2      00:12:00/00:01:26  v2       1 / B S P G
213.5.5.5      Tunnel2      00:12:24/00:01:25  v2       1 / B S P G
213.12.12.12   Tunnel2      00:13:01/00:01:40  v2       1 / DR B P G

```

We also check the PIMv6 neighbors for good measure. Interestingly, the IPv6 show command for XE has a “count” option but IPv4 does not.

```

RP/0/0/CPU0:XRv2#show pim vrf MC ipv6 neighbor | begin mdt
mdtMC

```

Neighbor Address	Uptime	Expires	DR	pri	DR
Flags					
::ffff:213.1.1.1	00:13:27	00:01:21	1		
::ffff:213.5.5.5	00:12:38	00:01:24	1		
::ffff:213.6.6.6	00:12:17	00:01:23	1		
::ffff:213.7.7.7	00:12:16	00:01:22	1		
::ffff:213.8.8.8	00:11:45	00:01:20	1		
::ffff:213.12.12.12*	00:14:05	00:01:26	1		(DR) P

R7#show ipv6 pim vrf MC neighbor | begin ^Neighbor

Neighbor Address	Interface	Uptime	Expires	Mode	DR	pri
::FFFF:213.1.1.1	Tunnel2	00:12:31	00:01:37	B G		1
::FFFF:213.5.5.5	Tunnel2	00:12:31	00:01:39	B G		1
::FFFF:213.6.6.6	Tunnel2	00:12:32	00:01:38	B G		1
::FFFF:213.8.8.8	Tunnel2	00:11:59	00:01:36	B G		1
::FFFF:213.12.12.12	Tunnel2	00:12:32	00:01:41	B G	DR	1
FE80::2	Gi2.527	05:29:27	00:01:42	B G		1

R1#show ipv6 pim vrf MC neighbor | begin ^Neighbor

Neighbor Address	Interface	Uptime	Expires	Mode	DR	pri
::FFFF:213.5.5.5	Tunnel2	00:13:08	00:01:24	B G		1
::FFFF:213.6.6.6	Tunnel2	00:12:47	00:01:23	B G		1
::FFFF:213.7.7.7	Tunnel2	00:12:46	00:01:23	B G		1
::FFFF:213.8.8.8	Tunnel2	00:12:14	00:01:21	B G		1
::FFFF:213.12.12.12	Tunnel2	00:13:56	00:01:26	B G	DR	1

R1#show ipv6 pim vrf MC neighbor count

Interface	Nbr count
Tunnel2	5
Gi2.519	0
Total Nbrs	5

A glance at CSR1's P-MRIB shows all of the expected P(S,G) entries for the MDT peers. The little 's' flag on each entry indicates SSM, which means there is no RP in use, and thus no P(*,G) or register messages in play. The big 'Z' indicates that this is a multicast tunnel, which is also correct. The big 'I' indicates that this entry is the result of having received a source-specific IGMP join, which happened on the loopback0.

R1#show ip mroute summary | include 232\255

```
(213.8.8.8, 232.255.0.1), 00:15:19/stopped, OIF count: 1, flags: sTIZ
(213.7.7.7, 232.255.0.1), 00:15:29/stopped, OIF count: 1, flags: sTIZ
(213.6.6.6, 232.255.0.1), 00:15:42/stopped, OIF count: 1, flags: sTIZ
(213.5.5.5, 232.255.0.1), 00:16:06/stopped, OIF count: 1, flags: sTIZ
(213.1.1.1, 232.255.0.1), 00:17:00/00:02:33, OIF count: 1, flags: sT
(213.12.12.12, 232.255.0.1), 00:17:00/stopped, OIF count: 1, flags: sTIZ
```

R1#show ip igmp groups 232.255.0.1
IGMP Connected Group Membership

Group Address	Interface	Uptime	Expires	Last Reporter
Group Accounted				
232.255.0.1	Loopback0	00:18:41	stopped	213.1.1.1

The challenge at this point is answering how CSR1 knew to originate PIM P(S,G) joins for all of the remote peers. A quick debug of PIM shows us that CSR1 is, in fact, sending these joins.

```

! CSR1
PIM(0): Insert (213.7.7.7,232.255.0.1) join in nbr 213.1.14.14's queue
PIM(0): Building Join/Prune packet for nbr 213.1.14.14
PIM(0): Adding v2 (213.7.7.7/32, 232.255.0.1), S-bit Join
PIM(0): Send v2 join/prune to 213.1.14.14 (GigabitEthernet2.514)

PIM(0): Insert (213.8.8.8,232.255.0.1) join in nbr 213.1.14.14's queue
PIM(0): Building Join/Prune packet for nbr 213.1.14.14
PIM(0): Adding v2 (213.8.8.8/32, 232.255.0.1), S-bit Join
PIM(0): Send v2 join/prune to 213.1.14.14 (GigabitEthernet2.514)

PIM(0): Insert (213.5.5.5,232.255.0.1) join in nbr 213.1.14.14's queue
PIM(0): Building Join/Prune packet for nbr 213.1.14.14
PIM(0): Adding v2 (213.5.5.5/32, 232.255.0.1), S-bit Join
PIM(0): Send v2 join/prune to 213.1.14.14 (GigabitEthernet2.514)

```

The answer is BGP MDT. The MDT route carries the MDT endpoint, along with the RD, and the MDT group. This is a simple mapping that makes it very easy for remote routers to construct P(S,G) joins, which is why SSM is the preferred method of operation for this MVPN profile. Looking at CSR1, it learns all of the other endpoints this way. The route doesn't carry a route target because it's not bringing new routing information or targeting a specific device. We will look at CSR5 as an example.

```

R1#show bgp ipv4 mdt all | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 213:1 (default for vrf MC)
 *> 213.1.1.1/32    0.0.0.0                        0 ?
Route Distinguisher: 213:5
 *>i 213.5.5.5/32   213.5.5.5                      0 100 0 ?
Route Distinguisher: 213:6
 *>i 213.6.6.6/32   213.6.6.6                      0 100 0 ?
Route Distinguisher: 213:7
 *>i 213.7.7.7/32   213.7.7.7                      0 100 0 ?
Route Distinguisher: 213:8
 *>i 213.8.8.8/32   213.8.8.8                      0 100 0 ?
Route Distinguisher: 213:12
 *>i 213.12.12.12/32 213.12.12.12                  100 0 i

R1#show bgp ipv4 mdt rd 213:8 213.8.8.8/32
BGP routing table entry for 213:8:213.8.8.8/32 version 19
Paths: (1 available, best #1, table IPv4-MDT-BGP-Table)

```

```

Not advertised to any peer
Refresh Epoch 1
Local
213.8.8.8 from 213.12.12.12 (213.12.12.12)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  Originator: 213.8.8.8, Cluster list: 213.12.12.12,
  MDT group address: 232.255.0.1
  rx pathid: 0, tx pathid: 0x0

```

Checking the BGP learned MDT information, we can get a summary view of all routers participating in a given MDT. This shows 5 other routers using the MDT 232.255.0.1.

```

R1#show ip pim mdt bgp
MDT (Route Distinguisher + IPv4)
MDT group 232.255.0.1
  213:1:213.5.5.5      213.12.12.12      213.5.5.5
  213:1:213.6.6.6      213.12.12.12      213.6.6.6
  213:1:213.7.7.7      213.12.12.12      213.7.7.7
  213:1:213.8.8.8      213.12.12.12      213.8.8.8
  213:1:213.12.12.12   213.12.12.12      213.12.12.12

```

Note: Although not yet relevant (will become relevant for Inter-AS models), the BGP connector attribute has been added to the VPNv4 prefixes. This is the result of using any profile 0 variant for MVPN. This attribute is meant to override the BGP next-hop when performing RPF checks. Even if the BGP next-hop changes, the original address is maintained so there is a mechanism to “connect” potential multicast sources to their MDT endpoints.

```

R1#show ip rpf vrf MC 10.2.7.0
RPF information for ? (10.2.7.0)
  RPF interface: Tunnel2
  RPF neighbor: ? (213.7.7.7)
  RPF route/mask: 10.2.7.0/24
  RPF type: unicast (bgp 213)
  Doing distance-preferred lookups across tables
  BGP originator: 213.7.7.7
  RPF topology: ipv4 multicast base, originated from ipv4 unicast base

```

```

R1#show bgp vpnv4 unicast vrf MC 10.2.7.0/24
BGP routing table entry for 213:1:10.2.7.0/24, version 475
Paths: (1 available, best #1, table MC)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from 213:7:10.2.7.0/24 (global)
    213.7.7.7 (metric 20) (via default) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:213:7
      Originator: 213.7.7.7, Cluster list: 213.12.12.12

```

```
Connector Attribute: count=1
type 1 len 12 value 213:7:213.7.7.7
mpls labels in/out nolabel/7006
rx pathid: 0, tx pathid: 0x0
```

At this point, everything is the same with respect to normal multicast forwarding. Because both the default and data MDTs are SSM, the P-mcast signaling has been simplified and the customer can still use ASM. First, we will test IPv4 SSM across the customer network. CSR2 is the source and CSR3/CSR4 are receivers. This means that CSR6, CSR7, and CSR8 should all have C(S,G) state for this group. The group is also candidate for data MDT treatment, so that selective delivery tree should have been signaled. The little 'y' on the egress PE means the traffic is being sent to an MDT data group, while the big 'Y' on the receivers means it is receiving data from an MDT data group.

```
R6#show ip mroute vrf MC 232.0.0.5 10.2.7.2 | begin \
(10.2.7.2, 232.0.0.5), 06:40:59/00:02:25, flags: sTIY
Incoming interface: Tunnel0, RPF nbr 213.7.7.7,
MDT:[213.7.7.7,232.4.7.0]/00:02:19
Outgoing interface list:
GigabitEthernet2.546, Forward/Sparse, 06:35:48/00:02:25
```

```
R8#show ip mroute vrf MC 232.0.0.5 10.2.7.2 | begin \
(10.2.7.2, 232.0.0.5), 06:47:32/00:02:18, flags: sTIY
Incoming interface: Tunnel2, RPF nbr 213.7.7.7,
MDT:[213.7.7.7,232.4.7.0]/00:02:23
Outgoing interface list:
GigabitEthernet2.538, Forward/Sparse, 06:47:22/00:02:18
```

```
R7#show ip mroute vrf MC 232.0.0.5 10.2.7.2 | begin \
(10.2.7.2, 232.0.0.5), 00:01:48/00:02:40, flags: sTY
Incoming interface: GigabitEthernet2.527, RPF nbr 0.0.0.0
Outgoing interface list:
Tunnel2, Forward/Sparse, 00:01:48/00:02:40
```

We can also verify the MDT signaling is correct by looking at the sent/received MDT information. This is used for signaling the dynamic data group assignments. The data MDT information is carried across the default MDT in a special PIM TLV that effectively notifies the egress LSR to join a specific data MDT.

```
R6#show ip pim mdt receive
Joined MDT-data [group/mdt number : source] uptime/expires for VRF: MC
[232.4.7.0 : 213.7.7.7] 00:40:22/00:02:01
```

```
R8#show ip pim mdt receive
Joined MDT-data [group/mdt number : source] uptime/expires for VRF: MC
[232.4.7.0 : 213.7.7.7] 00:40:57/00:02:26
```

```
R7#show ip pim mdt send
MDT-data send list for VRF: MC
```

(source, group)	MDT-data group/num	ref_count
(10.2.7.2, 232.0.0.5)	232.4.7.0	1

CSR2 will begin sending traffic now. There should be no more PIM signaling in either the provider or customer networks since all (S,G) trees are fully built. The counters have never been cleared on CSR6 or CSR8. CSR7 was rebooted in between tests to clear an error, but the counters continue to increment regardless, showing that data is flowing. We can also see the size difference is 24; when CSR7 measures the packet, it does so before GRE encapsulation. When the receivers measure it, they do so before GRE decapsulation.

```
R7#show ip mroute vrf MC 232.0.0.5 10.2.7.2 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 35, Packets received:
35
Source: 10.2.7.2/32, Forwarding: 35/1/118/0, Other: 35/0/0
```

```
R6#show ip mroute vrf MC 232.0.0.5 10.2.7.2 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 732, Packets received:
732
Source: 10.2.7.2/32, Forwarding: 732/1/142/1, Other: 732/0/0
```

```
R8#show ip mroute vrf MC 232.0.0.5 10.2.7.2 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 730, Packets received:
730
Source: 10.2.7.2/32, Forwarding: 730/1/142/1, Other: 730/0/0
```

One final check involves using EPC on CSR8 to measure traffic entering from the SP and exiting to the customer. We should see GRE multicast entering and ICMP multicast exiting. The source/destination addresses have been identified in the buffer dump to prove the data MDT is working. The difference in size between the packets is exactly 24 and the time difference is immeasurable by the router.

```
R8#show monitor capture CAP buffer detailed
 1  142    0.059002    213.7.7.7    -> 232.4.7.0    GRE
0000:  01005E04 07000050 56A9DE77 8100CE00    ..^....PV..w....
0010:  08004500 007C0337 0000FE2F EE08D507    ..E..|.7.../....
0020:  0707E804 07000000 08004500 00641D9C    .....E..d..
0030:  0000FE01 A5F30A02 0702E800 00050800    .....

 2  118    0.059002    10.2.7.2    -> 232.0.0.5    ICMP
0000:  01005E00 00050050 56A9FB1C 8100DD2    ..^....PV.....
0010:  08004500 00641D9C 0000FD01 A6F30A02    ..E..d.....
0020:  0702E800 00050800 C002000A 006A0000    .....j..
0030:  00001584 A84FABCD ABCDABCD ABCDABCD    .....O.....
```

To ensure IPv6 multicast is working, we will test IPv6 SSM using the customer pair (2001:10:4:6::4, FF33::1). Let's check the C-MRIB on the ingress/egress PEs to ensure the C(S,G) state is there. Absent

from the C(S,G) states are 'y' or 'Y' flags; this group is not candidate for MDT treatment per the IPv6 ACL. This traffic will always transit the SP network along the default MDT, meaning all PEs will receive it.

```
R6#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 | begin \  
(2001:10:4:6::4, FF33::1), 00:57:44/00:03:25, flags: sT  
  Incoming interface: GigabitEthernet2.546  
  RPF nbr: 2001:10:4:6::4  
  Immediate Outgoing interface list:  
    Tunnel0, Forward, 00:57:44/00:03:25
```

```
R7#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 | begin \  
(2001:10:4:6::4, FF33::1), 07:17:33/never, flags: sTI  
  Incoming interface: Tunnel2  
  RPF nbr: ::FFFF:213.6.6.6  
  Immediate Outgoing interface list:  
    GigabitEthernet2.527, Forward, 07:17:33/never
```

```
R8#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 | begin \  
(2001:10:4:6::4, FF33::1), 07:17:08/never, flags: sTI  
  Incoming interface: Tunnel2  
  RPF nbr: ::FFFF:213.6.6.6  
  Immediate Outgoing interface list:  
    GigabitEthernet2.538, Forward, 07:17:08/never
```

CSR4 begins sending traffic. There should be no change to the signaling, so next we verify the packet counters for each C(S,G) entry.

```
R6#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count | include HW  
  HW Forwarding: 73/1/118/0, Other: 0/0/0
```

```
R7#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count | include HW  
  HW Forwarding: 86/1/142/1, Other: 0/0/0
```

```
R8#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count | include HW  
  HW Forwarding: 81/1/142/1, Other: 0/0/0
```

Even routers like CSR1 and CSR5 are receiving (and dropping) this flow. This is the downside to using the default MDT for high-bandwidth flows; inefficient forwarding and wasted PE resources are allocated just to drop the traffic. The IPv6 VPN network is split so these other PEs don't even have unicast reachability to the source, so RPF fails.

```
R1#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 | begin \  
(2001:10:4:6::4, FF33::1), 07:23:47/never, flags: sTI  
  Incoming interface: Null  
  RPF nbr: ::  
  Immediate Outgoing interface list:  
    GigabitEthernet2.519, Forward, 07:23:47/never
```

```
R1#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count | include HW
HW Forwarding: 0/0/0/0, Other: 215/215/0
```

```
R5#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 | begin \((
(2001:10:4:6::4, FF33::1), 07:25:14/never, flags: STI
Incoming interface: Null
RPF nbr: ::
Immediate Outgoing interface list:
GigabitEthernet2.550, Forward, 07:25:14/never
```

```
R5#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count | include HW
HW Forwarding: 0/0/0/0, Other: 224/224/0
```

Packet counting in the core is less effective, but is still another tool to add to our confidence. Checking XRv4, we see many more packets from CSR6 than we do from CSR1 along the default MDT. This is another good sign that the data is flowing along the path we suspect.

```
RP/0/0/CPU0:XRv4#show mfib route 232.255.0.1 213.6.6.6 | begin 213
(213.6.6.6,232.255.0.1), Flags:
Up: 01:15:09
Last Used: 00:00:05
SW Forwarding Counts: 1142/1142/128736
SW Replication Counts: 1142/2263/255640
[snip]
```

```
RP/0/0/CPU0:XRv4#show mfib route 232.255.0.1 213.1.1.1 | begin 213
(213.1.1.1,232.255.0.1), Flags:
Up: 00:52:02
Last Used: 00:00:04
SW Forwarding Counts: 286/286/24636
SW Replication Counts: 286/1144/98544
[snip]
```

Reusing the EPC configuration on CSR8, we validate that traffic is being successfully decapsulated from the default MDT in the SP network to the original ICMP packet, and forwarded to the customer. The GRE IPv4 header shows a source of 213.6.6.6 and a destination of 232.255.0.1 (Default MDT). The inner IPv6 ICMP header shows a source of 2001:10:4:6::4 and a destination of FF33::1.

```
R8#show monitor capture CAP buffer dump
0
0000: 01005E7F 00010050 56A9DE77 8100CE00 ..^....PV..w....
0010: 08004500 007C197E 0000FD2F DFC7D506 ..E..|.~.../....
0020: 0606E8FF 00010000 86DD6000 0000003C .....`....<
0030: 3A3F2001 00100004 00060000 00000000 :? .....
0040: 0004FF33 00000000 00000000 00000000 ...3.....
0050: 00018000 54521D4F 018D8D8E 8F909192 ....TR.O.....
0060: 93949596 9798999A 9B9C9D9E 9FA0A1A2 .....
0070: A3A4A5A6 A7A8A9AA ABACADAE AFB0B1B2 .....
```

```

0080:  B3B4B5B6 B7B8B9BA BBBCBDBE BFC0      .....
1
0000:  33330000 00010050 56A9FB1C 8100DD2    33.....PV.....
0010:  86DD6000 0000003C 3A3F2001 00100004    ..`.....<:? .....
0020:  00060000 00000000 0004FF33 00000000    .....3.....
0030:  00000000 00000000 00018000 54521D4F    .....TR.O
0040:  018D8D8E 8F909192 93949596 9798999A    .....
0050:  9B9C9D9E 9FA0A1A2 A3A4A5A6 A7A8A9AA    .....
0060:  ABACADAE AFB0B1B2 B3B4B5B6 B7B8B9BA    .....
0070:  BBBCBDBE BFC0      .....

```

Additional Reading – Reference configurations “mvpn-0-ssm”

28.1.3 PIM-Bidir in the core

Continuing on with the SSM configuration, we will use a hybrid of Bidir and SSM. Just like we saw in the ASM demonstration, it does not make (much) sense to use ASM or Bidir for data MDTs. The whole purpose of data MDTs is to optimize for bandwidth, so having an RP in the signaling path is not an effective way to do that. Since the data MDT receivers are known, SSM is the best option. Technically, ASM can offer benefits to data MDTs since ultimately the SPT is built, but for Bidir it makes no sense unless there were multiple Bidir RPs placed intelligently around the network where high-bandwidth flows were known to be (seems too complicated). Realistically, only these three models make sense:

1. **SSM for default and data MDTs.** This is the best design, is most common, but requires BGP. Since BGP is already running on all the PEs for VPNv4 or other MPLS services, activating another AF is not a big deal.
2. **ASM for default and SSM for data MDTs.** A valid combination that wouldn't require BGP. Allow PE endpoints to discover one another and use SSM paths for high-bandwidth flows.
3. **Bidir for default and SSM for data MDTs.** A valid combination that wouldn't require BGP. So long as the RP is central in the data plane, this reduces default MDT (S,G) state in the network if there are many PEs. Use SSM paths for high-bandwidth flows.

We did not explicitly test #2, but in this demonstration we will explore option #3. That means the data groups will remain 232.x.x.x while the default MDT will become 238.255.0.1. I will also disable BGP MDT to prove that it is not required for this model to function.

```

! Basic XE PE configuration to switch from SSM to Bidir default MDT
vrf definition MC
 address-family ipv4
   no mdt default 232.255.0.1
   mdt default 238.255.0.1
   mdt data 232.4.1.0 0.0.0.255 list ACL_DATA_MDT
 address-family ipv6
   no mdt default 232.255.0.1
   mdt default 238.255.0.1

```

```

mdt data 232.6.1.0 0.0.0.255 list ACL_DATA_MDT_IPV6

! XRv2 configuration to switch from SSM to Bidir default MDT
multicast-routing
vrf MC
  address-family ipv4
    no mdt default ipv4 232.255.0.1
    mdt default ipv4 238.255.0.1
  address-family ipv6
    no mdt default ipv4 232.255.0.1
    mdt default ipv4 238.255.0.1

```

Next, we disable BGP MDT on all routers. We should not need it to build the default MDT since all traffic flows to XRv4 as the Bidir-RP. Removing it from XRv2 is enough since it breaks the entire MDT route exchange, but to keep the configuration clean, I remove it everywhere.

```

! Removing BGP MDT from XE PEs
router bgp 213
  no address-family ipv4 mdt

! Removing BGP MDT from XRv2
router bgp 213
  no address-family ipv4 mdt
  neighbor-group IBGP
    no address-family ipv4 mdt
  address-family ipv4 mdt
  no route-reflector-client

```

Taking a cavalier attitude towards verifying PIM neighbors, I will check the neighbor counts on XRv2 and assume everyone has connectivity. I spot check a CSR1 as well. XRv2 reports 6 neighbors because it counts itself as another neighbor.

```

RP/0/0/CPU0:XRv2#show pim vrf MC neighbor count | include mdt
mdtMC 6

```

```

RP/0/0/CPU0:XRv2#sh pim vrf MC ipv6 neighbor count | include mdt
mdtMC 6

```

```

R1#show ip pim vrf MC neighbor | begin ^Neighbor
Neighbor      Interface      Uptime/Expires    Ver    DR
Address
213.12.12.12   Tunnel2          00:06:10/00:01:21 v2      1 / DR B P G
213.8.8.8       Tunnel2          00:06:45/00:01:29 v2      1 / B S P G
213.7.7.7       Tunnel2          00:06:55/00:01:26 v2      1 / B S P G
213.6.6.6       Tunnel2          00:07:04/00:01:26 v2      1 / B S P G
213.5.5.5       Tunnel2          00:07:13/00:01:27 v2      1 / B S P G

```


Seeing PIM neighbors within the customer VRF means the default MDT is working. Looking at CSR6 and XRv2, we can see the select their core interfaces as upstream (towards the root of the shared tree, the Bidir-RP) and their “VRF” interfaces as downstream, towards the customer. The big ‘Z’ on XE means this is a multicast tunnel, which is correct. The XR flags ‘LH’ and DSS’ mean “last hop” and “don’t signal sources”. The former means the router is a leaf in the delivery tree and the latter is self-explanatory when using Bidir. The ‘LI’ interface flag means local interest since this router is requesting the flow for itself; it was not signaled by a receiver.

```
R6#show ip mroute 238.255.0.1 | begin \
(*, 238.255.0.1), 10:19:30/stopped, RP 213.14.14.14, flags: BCZ
  Bidir-Upstream: GigabitEthernet2.563, RPF nbr 213.6.13.13
  Outgoing interface list:
    MVRF MC, Forward/Sparse, 10:19:30/stopped
    GigabitEthernet2.563, Bidir-Upstream/Sparse, 10:19:30/stopped
```

```
RP/0/0/CPU0:XRv2#show pim topology 238.255.0.1 | begin 238
(*,238.255.0.1) BD Up: 10:20:10 RP: 213.14.14.14
JP: Join(00:00:42) RPF: GigabitEthernet0/0/0/0.523,213.12.13.13 Flags: LH DSS
  Loopback0 10:20:10 fwd LI LH
```

Because traffic on a Bidir tree always goes to the RP, there has to be a mechanism to send traffic upstream also (that’s why it’s called Bidir). This is comparable to an mLDP MP2MP tree where traffic flows upstream and downstream, where normal PIM ASM/SSM (or P2MP RSVP-TE) is comparable to an mLDP P2MP tree where traffic only flows downstream from the root. The designated forwarder (DF) is elected on each LAN and only this router is allowed to send traffic upstream to the root. This ensures the forwarding mechanism is loop-free. The output below shows XRv3 and CSR6 DF election results. The DF on the link between them is XRv3 since XRv3 is closer to the root (RP). CSR6 has a metric of 20 which is worse than XRv3’s metric of 10, so if Bidir multicast traffic is forwarded upstream to this LAN segment, only XRv3 is allowed to forward it upstream. This is what allows CSR6 to send traffic to the RP and ultimately have it be seen by the rest of the PEs. The XRv output is technically more correct since it shows the administrative distance also, which is evaluated before the metric.

```
RP/0/0/CPU0:XRv3#show pim df winner
```

RP	Interface	DF Winner	Metrics
213.14.14.14	GigabitEthernet0/0/0/0.573	213.7.13.13	[115/10]
213.14.14.14	GigabitEthernet0/0/0/0.563	213.6.13.13	[115/10]
213.14.14.14	GigabitEthernet0/0/0/0.553	213.5.13.13	[115/10]
213.14.14.14	GigabitEthernet0/0/0/0.534	213.13.14.14	[0/0]
213.14.14.14	GigabitEthernet0/0/0/0.523	213.12.13.13	[115/10]
213.14.14.14	Loopback0	213.13.13.13	[115/10]

```
R6#show ip pim interface df
* implies this system is the DF
```

Interface	RP	DF Winner	Metric	Uptime
Loopback0	213.14.14.14	*213.6.6.6	20	19:00:35
GigabitEthernet2.563	213.14.14.14	213.6.13.13	10	19:00:35

```
R6#show ip route 213.14.14.14 | include Known
Known via "isis", distance 115, metric 20, type level-2
```

Quick debugging on CSR6 reveal this process. The DF announcement packets are sent every ~10 seconds.

```
! CSR6
PIM(0): Received v2 DF on GigabitEthernet2.563 from 213.6.13.13
PIM(0): Receive DF Winner message from 213.6.13.13 on GigabitEthernet2.563
(Non-DF)
PIM(0): RP 213.14.14.14, pref 115, metric 10
PIM(0): Metric is better
```

The data MDT information for the IPv4 SSM pair (10.2.7.2, 232.0.0.5) is signaled across the default MDT from CSR7 to CSR6 and CSR8. We quickly verify this is still working as it was with the prior tests. When the 'y' flag says the C(S,G) is sending to a data MDT, this is the mapping table that shows which C(S,G) goes to which data MDT group.

```
R7#show ip pim mdt send
MDT-data send list for VRF: MC
  (source, group)                MDT-data group/num  ref_count
  (10.2.7.2, 232.0.0.5)          232.4.7.0          1
```

```
R6#show ip pim mdt receive
Joined MDT-data [group/mdt number : source]  uptime/expires for VRF: MC
[232.4.7.0 : 213.7.7.7]  10:26:12/00:02:47
```

```
R8#show ip pim mdt receive
Joined MDT-data [group/mdt number : source]  uptime/expires for VRF: MC
[232.4.7.0 : 213.7.7.7]  10:25:16/00:02:43
```

Like earlier, the debugging shows this information being transmitted over the default MDT.

```
! CSR6
PIM(1): Receive MDT Packet (41878) from 213.7.7.7 (Tunnel0), length (ip: 44,
udp: 24), ttl: 1
PIM(1): TLV type: 1 length: 16 MDT Packet length: 16
```

All three of these routers should show C-MRIB entries for this IPv4 SSM group. The ingress PE (CSR7) has little 'y' to indicate sending to a data MDT group, while the egress PEs (CSR6 and CSR8) have a big 'Y' to indicate receiving from it. The egress PEs also show the MDT mapping in their C-MRIB entries. On CSR7, we saw earlier how the C(S,G) maps to a data MDT when we looked at the "sent MDT" listing.

```
R7#show ip mroute vrf MC 10.2.7.2 232.0.0.5 | begin \
(10.2.7.2, 232.0.0.5), 11:22:32/00:03:11, flags: sTy
Incoming interface: GigabitEthernet2.527, RPF nbr 0.0.0.0
```

Outgoing interface list:

Tunnel2, Forward/Sparse, 10:41:13/00:03:11

```
R6#show ip mroute vrf MC 10.2.7.2 232.0.0.5 | begin \  
(10.2.7.2, 232.0.0.5), 18:12:58/00:02:27, flags: sTIY
```

Incoming interface: Tunnel0, RPF nbr 213.7.7.7,

MDT:[213.7.7.7,232.4.7.0]/00:02:31

Outgoing interface list:

GigabitEthernet2.546, Forward/Sparse, 18:07:46/00:02:27

```
R8#show ip mroute vrf MC 10.2.7.2 232.0.0.5 | begin \  
(10.2.7.2, 232.0.0.5), 18:19:43/00:02:10, flags: sTIY
```

Incoming interface: Tunnel2, RPF nbr 213.7.7.7,

MDT:[213.7.7.7,232.4.7.0]/00:02:22

Outgoing interface list:

GigabitEthernet2.538, Forward/Sparse, 18:19:33/00:02:10

Each router should have a corresponding P(S,G) of (213.7.7.7, 232.4.7.0) in their global MRIB also. This is how the traffic appears on the carrier's network. The little 'z' means this group is transport for a data MDT (true on CSR7), and on the others it is flagged as a multicast tunnel with a big 'Z'. Notice that the group is SSM, not Bidir, so this will have a positive performance impact on the customer multicast flow.

```
R7#show ip mroute 232.4.7.0 213.7.7.7 | begin \  
(213.7.7.7, 232.4.7.0), 12:11:26/00:02:33, flags: sTz
```

Incoming interface: Loopback0, RPF nbr 0.0.0.0

Outgoing interface list:

GigabitEthernet2.574, Forward/Sparse, 10:44:54/00:02:33

GigabitEthernet2.573, Forward/Sparse, 10:44:55/00:02:30

```
R6#show ip mroute 232.4.7.0 213.7.7.7 | begin \  
(213.7.7.7, 232.4.7.0), 12:12:02/stopped, flags: sTIZ
```

Incoming interface: GigabitEthernet2.563, RPF nbr 213.6.13.13

Outgoing interface list:

MVRF MC, Forward/Sparse, 00:07:50/00:01:09

```
R8#show ip mroute 232.4.7.0 213.7.7.7 | begin \  
(213.7.7.7, 232.4.7.0), 12:11:03/stopped, flags: sTIZ
```

Incoming interface: GigabitEthernet2.584, RPF nbr 213.8.14.14

Outgoing interface list:

MVRF MC, Forward/Sparse, 10:45:31/00:02:28

We should check to ensure XRv3 and XRv4 are not sending this traffic to any other PEs other than CSR6 and CSR8, respectively. Being SSM groups, only the SPT is used, and we can see the OILs are built correctly. Disinterested PEs will not receive any of this traffic.

```
RP/0/0/CPU0:XRv3#show pim topology 232.4.7.0 213.7.7.7 | begin 213  
(213.7.7.7,232.4.7.0) SPT SSM Up: 10:48:09  
JP: Join(00:00:36) RPF: GigabitEthernet0/0/0/0.573,213.7.13.7 Flags:
```

```
GigabitEthernet0/0/0/0.563 10:48:09 fwd Join(00:03:20)
```

```
RP/0/0/CPU0:XRv4#show pim topology 232.4.7.0 213.7.7.7 | begin 213
(213.7.7.7,232.4.7.0)SPT SSM Up: 10:48:15
JP: Join(00:00:32) RPF: GigabitEthernet0/0/0/0.574,213.7.14.7 Flags:
GigabitEthernet0/0/0/0.584 10:48:15 fwd Join(00:03:20)
```

CSR2 starts sending traffic. We check the counters on all three PE routers to ensure traffic is flowing.

```
R7#show ip mroute 232.4.7.0 213.7.7.7 count | begin ^Group
Group: 232.4.7.0, Source count: 1, Packets forwarded: 624, Packets received:
624
Source: 213.7.7.7/32, Forwarding: 624/1/124/0, Other: 624/0/0
```

```
R6#show ip mroute 232.4.7.0 213.7.7.7 count | begin ^Group
Group: 232.4.7.0, Source count: 1, Packets forwarded: 632, Packets received:
632
Source: 213.7.7.7/32, Forwarding: 632/1/142/1, Other: 632/0/0
```

```
R8#show ip mroute 232.4.7.0 213.7.7.7 count | begin ^Group
Group: 232.4.7.0, Source count: 1, Packets forwarded: 630, Packets received:
630
Source: 213.7.7.7/32, Forwarding: 630/1/142/1, Other: 630/0/0
```

As a final verification, we will use EPC on CSR8 to look at traffic entering from the core (GRE) and exiting to the customer (ICMP). As we have seen many times, the timestamps are equal, the difference in length is 24, and the P(S,G) and C(S,G) information is as we expect.

```
R8#show monitor capture CAP buffer detailed
0 142 0.000000 213.7.7.7 -> 232.4.7.0 GRE
0000: 01005E04 07000050 56A9DE77 8100CE00 ..^....PV..w....
0010: 08004500 007C0597 0000FE2F EBA8D507 ..E..|...../....
0020: 0707E804 07000000 08004500 00641FFC .....E..d..
0030: 0000FE01 A3930A02 0702E800 00050800 .....

1 118 0.000000 10.2.7.2 -> 232.0.0.5 ICMP
0000: 01005E00 00050050 56A9FB1C 8100DD2 ..^....PV.....
0010: 08004500 00641FFC 0000FD01 A4930A02 ..E..d.....
0020: 0702E800 00050800 FA97000B 006D0000 .....m..
0030: 000017FD 6B3DABCD ABCDABCD ABCDABCD ....k=.....
```

For good measure, here is a PIM signaling packet transiting the default MDT. We know it is on the default MDT based on the destination group address 238.255.0.1. The TTL is 1 (0x01) and the protocol is 103/PIM (0x67). This packet in particular came from CSR5 destined to 224.0.0.13 (highlighted). It is probably a PIM Hello, BSR message, or periodic join/prune. We know that this packet transited the Bidir shared tree to reach CSR8.

```
R8#show monitor capture CAP buffer detailed
14  98  67.960009  213.5.5.5  -> 238.255.0.1  GRE
0000:  01005E7F 00010050 56A9DE77 8100CE00  ..^....PV..w....
0010:  080045C0 00500AAA 0000FE2F E809D505  ..E..P...../....
0020:  0505EEFF 00010000 080045C0 003884DA  .....E..8..
0030:  00000167 79AD505 0505E000 000D2400  ...gy.....$.
```

Next we will test IPv6 SSM as we did earlier. The only difference with this delivery mechanism is that the IPv6 SSM pair (2001:10:4:6::4, FF33::1) is not candidate for MDT optimization as it does not match the ACL we applied. This means the traffic will always transit the default MDT. First, we verify that CSR7 and CSR8 receive the MLDv2 joins from their customers and that the C(S,G) tree was properly built back to the root, CSR6. The only significant difference between this set of outputs and the IPv4 SSM set is the lack of any 'y' or 'Y' flags. This is because the data MDT operation is not involved with this C(S,G) at all.

```
R6#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 | begin \
(2001:10:4:6::4, FF33::1), 10:58:30/00:03:05, flags: sT
Incoming interface: GigabitEthernet2.546
RPF nbr: 2001:10:4:6::4
Immediate Outgoing interface list:
Tunnel0, Forward, 10:58:30/00:03:05
```

```
R7#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 | begin \
(2001:10:4:6::4, FF33::1), 18:44:37/never, flags: sTI
Incoming interface: Tunnel2
RPF nbr: ::FFFF:213.6.6.6
Immediate Outgoing interface list:
GigabitEthernet2.527, Forward, 18:44:37/never
```

```
R8#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 | begin \
(2001:10:4:6::4, FF33::1), 18:44:15/never, flags: sTI
Incoming interface: Tunnel2
RPF nbr: ::FFFF:213.6.6.6
Immediate Outgoing interface list:
GigabitEthernet2.538, Forward, 18:44:15/never
```

Since there is no data MDT, and the default MDT has been verified, we can safely assume this traffic will transport inside the default MDT. This means that XRv4 will always receive traffic for this flow. When CSR4 starts sending traffic, we check the packet counters on the three PEs to ensure it is working. The packet counters are all increasing as traffic flows.

```
R6#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count | include HW
HW Forwarding: 790/1/118/0, Other: 0/0/0
```

```
R7#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count | include HW
HW Forwarding: 797/1/142/1, Other: 0/0/0
```

```
R8#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count | include HW
```

```
HW Forwarding: 792/1/142/1, Other: 0/0/0
```

To show the drawbacks of using PIM Bidir with default MDTs, I will instruct CSR3 to stop requesting this flow. CSR8 will send a C(S,G) prune towards CSR6 but it has no effect since CSR7 is still on the “emulated LAN” requesting traffic. When CSR7 sees the C(S,G) prune from CSR8, it immediately issues a C(S,G) join so that CSR6 will cancel its removal of the MDT from its OIL. Since there is no P(S,G) prune mechanism for the default MDT, traffic will still be sent to XRv4 and dropped. This is a prune override operation in action.

! CSR8

```
IPv6 PIM: [MC] (2001:10:4:6::4,FF33::1) GigabitEthernet2.538 Local state
changed from Join to Null
IPv6 PIM: [MC] (2001:10:4:6::4,FF33::1) GigabitEthernet2.538 FWD state change
from Forward to Prune
IPv6 PIM: [MC] FF33::1 Delete Group
```

! CSR7

```
IPv6 PIM: [MC] J/P entry: Prune root: 2001:10:4:6::4 group: FF33::1 Sender:
::FFFF:213.8.8.8 flags: S
IPv6 PIM: [MC] (2001:10:4:6::4,FF33::1) J/P adding Join on Tunnel2
```

! CSR6

```
IPv6 PIM: [MC] J/P entry: Prune root: 2001:10:4:6::4 group: FF33::1 Sender:
::FFFF:213.8.8.8 flags: S
IPv6 PIM: [MC] (2001:10:4:6::4,FF33::1) Tunnel0 Prune scheduled
IPv6 PIM: [MC] J/P entry: Join root: 2001:10:4:6::4 group: FF33::1 Sender:
::FFFF:213.7.7.7 flags: S
IPv6 PIM: [MC] (2001:10:4:6::4,FF33::1) Tunnel0 Canceling scheduled prune
```

Let’s ensure that CSR7 is still receiving the traffic while CSR8 is not. CSR8 doesn’t even have the C-MRIB entry anymore let alone statistical information for the data plane. CSR7 is still receiving traffic, though.

```
R7#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count | include HW
HW Forwarding: 1237/1/142/1, Other: 0/0/0
```

```
R8#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count | include HW
[no output]
```

To prove packets are entering CSR6, we will do two things. We will check the counters on the P(S,G) for the default MDT on CSR8 to show they are being received. This method isn’t fool-proof since you are basically trying to guess how many packets are along the default MDT that aren’t signaling, but the two outputs below were sampled 10 seconds apart. The source is sending 1 pps, so seeing ~12 packets in 10 seconds is probably an indication that the traffic is being received.

```
R8#show ip mroute 238.255.0.1 count | begin ^Group
```

```
Group: 238.255.0.1, Source count: 0, Packets forwarded: 20742, Packets
received: 20745
```

```
RP-tree: Forwarding: 20742/1/107/1, Other: 20745/3/0
```

```
R8#show ip mroute 238.255.0.1 count | begin ^Group
```

```
Group: 238.255.0.1, Source count: 0, Packets forwarded: 20754, Packets
received: 20757
```

```
RP-tree: Forwarding: 20754/1/107/1, Other: 20757/3/0
```

We can also use EPC to show the packets entering and not being sent down to the customer. Below we see two packets spaced exactly one second apart, but there is no raw ICMPv6 being sent to the customer. We can see the beginning of the IPv6 source address as 2001:10:4:6, which we know is the customer behind CSR6.

```
R8#show monitor capture CAP buffer detailed
```

```
1 142 0.413980 213.6.6.6 -> 238.255.0.1 GRE
0000: 01005E7F 00010050 56A9DE77 8100CE00 ..^....PV..w....
0010: 08004500 007C1E5C 0000FD2F D4E9D506 ..E...|.\\.../....
0020: 0606EEFF 00010000 86DD6000 0000003C .....`....<
0030: 3A3F2001 00100004 00060000 00000000 :? .....

2 142 1.413980 213.6.6.6 -> 238.255.0.1 GRE
0000: 01005E7F 00010050 56A9DE77 8100CE00 ..^....PV..w....
0010: 08004500 007C1E5D 0000FD2F D4E8D506 ..E...|.\\.../....
0020: 0606EEFF 00010000 86DD6000 0000003C .....`....<
0030: 3A3F2001 00100004 00060000 00000000 :? .....


```

To confirm that the flow can be repaired, we re-add the MLDv2 join to CSR3 and we now see the ICMPv6 packets flowing to the customer.

```
R8#show monitor capture CAP buffer detailed
```

```
0 142 0.000000 213.6.6.6 -> 238.255.0.1 GRE
0000: 01005E7F 00010050 56A9DE77 8100CE00 ..^....PV..w....
0010: 08004500 007C1F51 0000FD2F D3F4D506 ..E...|.Q.../....
0020: 0606EEFF 00010000 86DD6000 0000003C .....`....<
0030: 3A3F2001 00100004 00060000 00000000 :? .....

1 118 0.000000 2001:::0004 -> FF33:::0001 IPv6-ICMP
0000: 33330000 00010050 56A9FB1C 8100DD2 33.....PV.....
0010: 86DD6000 0000003C 3A3F2001 00100004 ..`....<:?.
0020: 00060000 00000000 0004FF33 00000000 .....3....
0030: 00000000 00000000 00018000 96921B72 .....r


```

Looking at the brief buffer information also shows this, but in less detail.

```
R8#show monitor capture CAP buffer brief
```

```
-----
```

#	size	timestamp	source		destination	protocol
0	142	0.000000	213.6.6.6	->	238.255.0.1	GRE
1	118	0.000000	2001:::0004	->	FF33:::0001	IPv6-ICMP

Additional Reading – Reference configurations “mvpn-0-bidir”

28.2 Profile 1: Default MDT – MLDP MP2MP – PIM C–mcast Signaling (Basic mLDP)

This profile is the one most people think of when mLDP is discussed. It is a very simple configuration and is basically a one-for-one replacement with the GRE-Rosen method in terms of MDT methodology. The profile specifically calls out MP2MP trees, but we examine P2MP MDT trees also. The basic mLDP configuration is very straightforward, and I will configure support for the MP2MP tree and P2MP trees together. The MP2MP tree is a bidirectional tree that represents and emulated LAN. When used with C-PIM signaling, it forms a full-mesh of all VPN participants, which is determined by the VRF’s VPN-ID. This is not to be confused with the RD, which can (and does) vary between the PEs. An mLDP MP2MP tree is logically comparable to a PIM Bidir RP. Before doing anything, we should verify the routers are mLDP-capable. XRv lies and always says it is mLDP capable even when the process is disabled. Maybe this “capability” is loosely interpreted as the feature being supported, but this is advertised to LDP peers even when mLDP is off.

```
RP/0/0/CPU0:XRv4#show mpls ldp capabilities
```

Type	Description	Owner
0x50b	Typed Wildcard FEC	LDP
0x3eff	Cisco IOS-XR	LDP
0x508	MP: Point-to-Multipoint (P2MP)	mLDP
0x509	MP: Multipoint-to-Multipoint (MP2MP)	mLDP
0x703	P2MP PW	L2VPN-AToM

```
RP/0/0/CPU0:XRv4#show mpls mldp database
```

```
mLDP is not running (1082230272)
```

The configuration below enables mLDP in XR. I enable logging as well, but that is not required for mLDP to work.

```
! XRv4
mpls ldp
  mldp
  logging notifications
```

I have selected XRv4 to be the “root” of the mLDP “shared tree” and instructed the VRF to prefer mLDP over PIM. I also show a snippet for how to enable basic mLDP in XR because it is not on by default, as it is with XE. XR is much more complicated by introducing “core-trees”, which are essentially RPF lookup mechanisms. We specifically need to tell PIM to use the mLDP-based tree for RPF.


```

! XE basic PE configuration for all PEs
vrf definition MC
  vpn id 213:213
  address-family ipv4
    mdt preference mldp
    mdt default mpls mldp 213.14.14.14

! XRv2 basic PE configuration
vrf MC
  vpn id 213:213

multicast-routing
  address-family ipv4
    interface Loopback0
      enable
    mdt source Loopback0

vrf MC
  address-family ipv4
    mdt default mldp ipv4 213.14.14.14

route-policy RPL_MLDP
  set core-tree mldp-default
end-policy

router pim
  vrf MC
    address-family ipv4
      rpf topology route-policy RPL_MLDP

```

As soon as a node is told about a root, the mLDP process adds this to its database. Behind the scenes, it creates an LSPvif for forwarding LSM. It automatically inherits the Loopback0 (LDP RID) configuration as the MDT source (XR needs to be explicitly told, XE does not) and enables PIM on it. A MP2MP default MDT is always identified by the number 0 contained within the opaque ID. This is the “MDT number” and as P2MP trees are spawned, the number increments.

```

R6#debug mpls mldp all
MLDP-MFI: Enabled MLDP MFI client on Lspvif0; status = ok
MLDP: Reclaimed success lsp vif: Lspvif0 address: 0.0.0.0 application: MDT
vrf_id: 1
MLDP: Enabled IPv4 on Lspvif0 unnumbered with Loopback0
MLDP: Enable pim on lsp vif: Lspvif0
MLDP: Add success lsp vif: Lspvif0 address: 0.0.0.0 application: MDT vrf_id:
1
MLDP-MDT: [mdt 213:213 0] wavl insert success
MLDP: LDP root 213.14.14.14 added

```

Next, the router needs to signal to the root that it is a leaf in the tree. It does this by sending a label mapping message towards the root based on the shortest IGP path. This is not based on the multicast RPF lookups, but uses the actual unicast routing table. We see that shortest path to XRv4 is via XRv3. We also notice that mLDP allocated a local label of 6005. This is the downstream label that CSR6 advertises to XRv3 to say “when you want to send me traffic inside this specific VPN along the MP2MP tree, use this label”. ID:5 at the end is the LSM path ID, which is another way to reference the mLDP database entry.

```
! CSR6
MLDP: nhop 213.6.13.13 added
MLDP-NBR: 213.13.13.13:0 mapped to next_hop: 213.6.13.13
MLDP: Root 213.14.14.14 old paths: 0 new paths: 1
MLDP-DB: [mdt 213:213 0] Changing peer from none to 213.13.13.13:0
MLDP-DB: [mdt 213:213 0] Add accepting element nbr: 213.13.13.13:0
MLDP-MFI: For accepting element nbr: 213.13.13.13:0, [mdt 213:213 0] MFI
allocated label 6005
MLDP: [mdt 213:213 0] label mapping msg sent to 213.13.13.13:0 success
MLDP-DB: [mdt 213:213 0] path to peer: 213.13.13.13:0 changed None:0.0.0.0 to
GigabitEthernet2.563:213.6.13.13
MLDP-MFI: After the MFI call For accepting element nbr: 213.13.13.13:0, Bind
local label 6005 to PSM ID: 5, returned: 6005
```

If we have advertised our downstream local label to the upstream router, surely the upstream router will give us an upstream label so we can reach the root. This only occurs with MP2MP trees. This upstream label is a local label on the upstream router which says “when you want to send traffic through me to reach the root, use this label”.

```
! CSR6
MLDP-LDP: [mdt 213:213 0] label mapping from: 213.13.13.13:0 label: 93004
root: 213.14.14.14 Opaque_len: 14 sess_hdl: 0xF
MLDP-DB: [mdt 213:213 0] Set upstream label from no_label to 93004
```

We can look at the mLDP database for the MP2MP tree (ID=0) for the VPN in question to verify these labels mapping. We also see the LSM ID:5 which corresponds to the debug messages. Notice that the local label is downstream (XRv3 uses this to send traffic away from the root) and the outbound label is upstream (CSR6 uses this label to send traffic towards the root). The output also shows “replication clients” which are essentially downstream clients. In this case, it’s the customer VRF.

```
R6#show mpls mldp database opaque_type mdt 213:213 0
LSM ID : 5 (RNR LSM ID: 6) Type: MP2MP Uptime : 00:08:58
FEC Root : 213.14.14.14
Opaque decoded : [mdt 213:213 0]
Opaque length : 11 bytes
Opaque value : 02 000B 0002130000021300000000
RNR active LSP : (this entry)
```

```

Upstream client(s) :
  213.13.13.13:0    [Active]
    Expires         : Never           Path Set ID   : 5
    Out Label (U)   : 93004           Interface    : GigabitEthernet2.563*
    Local Label (D) : 6005            Next Hop     : 213.6.13.13
Replication client(s):
  MDT (VRF MC)
    Uptime          : 00:08:58        Path Set ID   : 6
    Interface       : Lspvif0

```

XRv3 also generates a log message which shows similar information. The log is incomplete in that it says no local label was assigned, but if we check the database, we clearly see it. The log also shows us the LSM-ID, a long hexadecimal number, which I use the reference the mLDP database entry. XRv3 also has a second downstream client, XRv2. The same process occurs for all clients, and the label mapping process happens all the way until the root is reached.

```

! XRv3
mpls_ldp[1042]: %ROUTING-MLDP-5-BRANCH_ADD : 0x00001 [mdt 213:213 0] MP2MP
213.14.14.14, Add LDP 213.6.6.6:0 branch remote label 6005, local label
no_label

```

```

RP/0/0/CPU0:XRv3#show mpls mldp database 0x00001
mLDP database
LSM-ID: 0x00001    Type: MP2MP    Uptime: 01:21:25
  FEC Root          : 213.14.14.14
  Opaque decoded    : [mdt 213:213 0]
  Upstream neighbor(s) :
    213.14.14.14:0 [Active] Uptime: 00:22:10
      Next Hop       : 213.13.14.14
      Interface      : GigabitEthernet0/0/0/0.534
      Local Label (D) : 93005           Remote Label (U) : 94007
  Downstream client(s):
    LDP 213.6.6.6:0 Uptime: 00:17:55
      Next Hop       : 213.6.13.6
      Interface      : GigabitEthernet0/0/0/0.563
      Remote label (D) : 6005           Local label (U) : 93004
    LDP 213.12.12.12:0 Uptime: 00:22:31
      Next Hop       : 213.12.13.12
      Interface      : GigabitEthernet0/0/0/0.523
      Remote label (D) : 92003           Local label (U) : 93012

```

A quick way to see if the root is reachable with a properly built tree is to look at the root information directly. The output is similar in XE and XR. Recall that we never configured the root on XRv3, it learned it from the label mapping messages from the downstream clients (PEs).

```

R6#show mpls mldp root 213.14.14.14
Root node      : 213.14.14.14
Metric         : 20

```

```

Distance      : 115
Interface     : GigabitEthernet2.563 (via unicast RT)
FEC count     : 1
Path count    : 1
Path(s)       : 213.6.13.13          LDP nbr: 213.13.13.13:0
GigabitEthernet2.563

```

```

RP/0/0/CPU0:XRv3#show mpls mldp root 213.14.14.14
Root node     : 213.14.14.14
Metric        : 10
Distance      : 115
FEC count     : 1
Path count    : 1
Path(s)       : 213.13.14.14        LDP nbr: 213.14.14.14:0

```

The root also knows that is it the root using the same method as XRv3. When the metric/distance are 0, there are no LDP neighbors towards the root, and the router knows it is the root. At no time did we explicitly configure XRv4 to be root on XRv4 itself.

```

RP/0/0/CPU0:XRv4#show mpls mldp root 213.14.14.14
Root node     : 213.14.14.14 (We are the root)
Metric        : 0
Distance      : 0
FEC count     : 1
Path count    : 1
Path(s)       : 213.14.14.14        LDP nbr: none

```

The PEs are able to communicate over this MP2MP tree which allows them to form PIM adjacencies across their virtual interfaces within the VPN. On XRv3, when label 93005 comes in, the traffic is replicated towards both CSR6 and XRv2 (downstream clients). Label 93005 is the downstream label that XRv3 allocated and advertised towards XRv4.

```

RP/0/0/CPU0:XRv3#show mpls mldp database opaquetype mdt 213:213 0 | begin
Upst
Upstream neighbor(s) :
  213.14.14.14:0 [Active] Uptime: 00:33:08
    Next Hop       : 213.13.14.14
    Interface      : GigabitEthernet0/0/0/0.534
    Local Label (D) : 93005           Remote Label (U): 94007

```

```

RP/0/0/CPU0:XRv3#show mpls forwarding labels 93005

```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
93005	6005	MLDP: 0x00001	Gi0/0/0/0.563	213.6.13.6	0
	92003	MLDP: 0x00001	Gi0/0/0/0.523	213.12.13.12	0

CSR6 and XRv2, upon receiving this, associate it with the proper VPN. That is why PHP cannot be used for LSM, since the label determines in which VPN the traffic belongs. XR takes a few show commands to verify that the traffic is part of a specific MDT.

```
R6#show mpls forwarding-table labels 6005
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Label	Outgoing interface	Next Hop
6005	[T] No Label	[mdt 213:213 0] [V]	\			
	0	aggregate/MC				

```
RP/0/0/CPU0:XRv2#show mpls forwarding labels 92003
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
92003	Unlabelled	MLDP: 0x00003			

```
RP/0/0/CPU0:XRv2#show mpls mldp database 0x00003 brief
```

LSM ID	Type	Root	Up	Down	Decoded	Opaque	Value
0x00003	MP2MP	213.14.14.14	1	1	[mdt 213:213 0]		

We can verify the PIM adjacencies over the tunnel also. This happens within the VRF since the global process is PIM-free.

```
RP/0/0/CPU0:XRv2#show pim vrf MC neighbor | begin ^Neighbor
```

Neighbor Address	Interface	Uptime	Expires	DR	pri	Flags
10.11.12.11	Gig0/0/0/0.512	02:00:40	00:01:33	1	B	P
10.11.12.12*	Gig0/0/0/0.512	02:27:50	00:01:33	1	(DR)	B P E
213.1.1.1	LmdtMC	00:36:37	00:01:40	1	P	
213.5.5.5	LmdtMC	00:36:37	00:01:44	1	P	
213.6.6.6	LmdtMC	01:35:35	00:01:43	1	P	
213.7.7.7	LmdtMC	00:36:37	00:01:41	1	P	
213.8.8.8	LmdtMC	00:36:37	00:01:15	1	P	
213.12.12.12*	LmdtMC	01:35:39	00:01:34	1	(DR)	P

```
R6#show ip pim vrf MC neighbor | begin ^Neighbor
```

Neighbor Address	Interface	Uptime/Expires	Ver	DR Prio/Mode
213.12.12.12	Lspvif0	00:34:34/00:01:27	v2	1 / DR P G
213.5.5.5	Lspvif0	00:34:35/00:01:36	v2	1 / S P G
213.7.7.7	Lspvif0	00:34:35/00:01:33	v2	1 / S P G
213.1.1.1	Lspvif0	00:34:35/00:01:34	v2	1 / S P G
213.8.8.8	Lspvif0	00:34:35/00:01:37	v2	1 / S P G

With the PIM neighbors up, the PEs should also learn that CSR10 is an RP. If you forget to change the core-tree on XR PEs, they will drop the BSR messages due to failed RPF. The “info source” is different but this is just cosmetic between XE and XR.

```
RP/0/0/CPU0:XRv2#show pim vrf MC rp mapping
```

PIM Group-to-RP Mappings

Group(s) 224.0.0.0/4

RP 10.5.10.10 (?), v2

Info source: 213.5.5.5 (?), elected via bsr, priority 0, holdtime 150

Uptime: 00:02:35, expires: 00:01:56

R6#show ip pim vrf MC rp mapping

PIM Group-to-RP Mappings

Group(s) 224.0.0.0/4

RP 10.5.10.10 (?), v2

Info source: 10.5.10.10 (?), via bootstrap, priority 0, holdtime 150

Uptime: 00:40:26, expires: 00:01:31

At this point, I configure CSR4, CSR3, CSR9, and XRv1 to be receivers for 225.0.0.1. This is an ASM group so each PE should issues a C(*,G) join towards the C-RP, CSR10. I do not show the IGMP configuration because it is simple. Below is a quick check of the PEs to ensure they see the C-RP via the LSPvif0 interface with proper RPF information.

R1#show ip mroute vrf MC sparse

(*, 225.0.0.1), 01:37:11/00:02:48, RP 10.5.10.10, flags: SJC

Incoming interface: Lspvif0, RPF nbr 213.5.5.5

Outgoing interface list:

GigabitEthernet2.519, Forward/Sparse, 01:21:09/00:02:48

R6#show ip mroute vrf MC sparse

(*, 225.0.0.1), 01:37:29/00:02:46, RP 10.5.10.10, flags: SJC

Incoming interface: Lspvif0, RPF nbr 213.5.5.5

Outgoing interface list:

GigabitEthernet2.546, Forward/Sparse, 01:21:13/00:02:46

R8#show ip mroute vrf MC sparse

(*, 225.0.0.1), 01:37:23/00:02:17, RP 10.5.10.10, flags: SJC

Incoming interface: Lspvif0, RPF nbr 213.5.5.5

Outgoing interface list:

GigabitEthernet2.538, Forward/Sparse, 01:37:23/00:02:17

RP/0/0/CPU0:XRv2#show pim vrf MC topology 225.0.0.1

(*,225.0.0.1) SM Up: 00:00:35 RP: 10.5.10.10

JP: Join(00:00:14) RPF: LmdtMC,213.5.5.5 Flags: LH

GigabitEthernet0/0/0/0.512 00:00:35 fwd LI LH

Next, we check the ingress PE and C-RP, CSR5 and CSR10, to ensure they see the C(*,G) join. CSR5 is the only router that will have the LSPvif0 in the OIL since the RP is behind it. CSR10 has no incoming interface since it is the root of the shared tree, and is running PIM in the global table.

R5#show ip mroute vrf MC sparse

(*, 225.0.0.1), 01:45:03/00:03:11, RP 10.5.10.10, flags: S

```
Incoming interface: GigabitEthernet2.550, RPF nbr 10.5.10.10
Outgoing interface list:
  Lspvif0, Forward/Sparse, 01:45:03/00:03:11
```

```
R10#show ip mroute sparse
(*, 225.0.0.1), 01:45:53/00:02:54, RP 10.5.10.10, flags: S
  Incoming interface: Null, RPF nbr 0.0.0.0
  Outgoing interface list:
    GigabitEthernet2.550, Forward/Sparse, 01:45:53/00:02:54
```

Assuming CSR2 is the source and is sending traffic, we expect CSR7 to send a register message to the RP. The RP will forward the traffic down the shared tree until it can build the C(S,G) tree back to CSR7. Once it does that, it will prune itself from the C(S,G) tree since it is not in the transit path for any of those flows. All of the PEs will have C(S,G) state for the source.

```
CSR10#debug ip pim
PIM(0): Received v2 Register on GigabitEthernet2.550 from 10.2.7.7
        for 10.2.7.2, group 225.0.0.1
PIM(0): Insert (10.2.7.2,225.0.0.1) join in nbr 10.5.10.5's queue
PIM(0): Send v2 join/prune to 10.5.10.5 (GigabitEthernet2.550)
PIM(0): Prune-list: (10.2.7.2/32, 225.0.0.1) RPT-bit set

PIM(0): Received v2 Register on GigabitEthernet2.550 from 10.2.7.7
        for 10.2.7.2, group 225.0.0.1
PIM(0): Send v2 Register-Stop to 10.2.7.7 for 10.2.7.2, group 225.0.0.1
PIM(0): Insert (10.2.7.2,225.0.0.1) prune in nbr 10.5.10.5's queue
PIM(0): Send v2 join/prune to 10.5.10.5 (GigabitEthernet2.550)
```

We will verify that the PEs are actually forwarding traffic towards the customers. XRv does not appear to work in the data plane, though.

```
R1#show ip mroute vrf MC 225.0.0.1 count | begin Group
Group: 225.0.0.1, Source count: 1, Packets forwarded: 26, Packets received:
26
  RP-tree: Forwarding: 3/0/122/0, Other: 3/0/0
  Source: 10.2.7.2/32, Forwarding: 23/0/122/0, Other: 23/0/0
```

```
R6#show ip mroute vrf MC 225.0.0.1 count | begin Group
Group: 225.0.0.1, Source count: 1, Packets forwarded: 101, Packets received:
101
  RP-tree: Forwarding: 2/0/122/0, Other: 2/0/0
  Source: 10.2.7.2/32, Forwarding: 99/0/122/0, Other: 99/0/0
```

```
R8#show ip mroute vrf MC 225.0.0.1 count | begin Group
Group: 225.0.0.1, Source count: 1, Packets forwarded: 36, Packets received:
36
  RP-tree: Forwarding: 2/0/122/0, Other: 2/0/0
  Source: 10.2.7.2/32, Forwarding: 34/1/122/0, Other: 34/0/0
```

```
RP/0/0/CPU0:XRv2#show mfib vrf MC interface GigabitEthernet0/0/0/0.512
Interface : GigabitEthernet0/0/0/0.512 (Enabled)
SW Mcast pkts in : 0, SW Mcast pkts out : 0
TTL Threshold : 0
Ref Count : 4
```

XRv LSM LFIB counters don't appear to work, but the debugs do. On XRv3, we can see traffic arriving from CSR7 and being replicated towards XRv2 and CSR6. XRv3 spits out a nasty message that the P2MP traffic is being dropped, but it actually isn't. We saw the counters increment on CSR6 as it forward traffic towards the customer. The LFIB has this properly programmed but the counters don't increase. Notice that this MPLS packet came in from XRv4, not direct from CSR7. This is because we are using the MP2MP tree, so transit traffic goes through the root. There are ways to optimize this, which are examined later.

```
RP/0/0/CPU0:XRv3#debug mpls packet
netio[309]: mpls_switch: GigabitEthernet0_0_0_0.534, mpls eos 1, ttl 253, len
122, inlabel 93005, tbl_id=0x0, vrf_id=0x0 in=0xe00
netio[309]: mpls_p2mp_switch: dir: ingress
netio[309]: mpls_rewrite
netio[309]: mpls_rewrite: tos 0 eos 1 ttl 252, out_label 92003, #labels 1, RA
- 0, out_intf GigabitEthernet0_0_0_0.523
netio[309]: mpls_rewrite
netio[309]: mpls_rewrite: tos 0 eos 1 ttl 252, out_label 6005, #labels 1, RA
- 0, out_intf GigabitEthernet0_0_0_0.563
netio[309]: [mpls_p2mp_switch_egress:819] Pkt Drop: P2MP MPLS2IP disposition:
egress - LKUP flag not set, input ifh:0x300
```

```
RP/0/0/CPU0:XRv3#show mpls forwarding labels 93005
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
93005	6005	MLDP: 0x00001	Gi0/0/0/0.563	213.6.13.6	0
	92003	MLDP: 0x00001	Gi0/0/0/0.523	213.12.13.12	0

Additionally, I confirm it with EPC on CSR6. Immediately following the 0x8847 ethertype, label 0x1775 (6005) is seen entering the router, which is the correct local downstream label allocated by CSR6 towards XRv3. The packet has EXP 0 and is the bottom-most label. The IP packet has source 10.2.7.2 (green) and destination 225.0.0.1 (pink).

```
R6#show monitor capture CAP buffer detailed
 4 122 2.286987 00:50:56:A9:DB:37 -> 00:50:56:A9:DE:0D MPLS unicast
0000: 005056A9 DE0D0050 56A9DB37 81000DEB .PV....PV..7....
0010: 88470177 51FC4500 0064012B 0000FE01 .G.wQ.E..d.+....
0020: C9680A02 0702E100 00010800 1E7A0018 .h.....z..
0030: 00020000 00004958 165EABCD ABCDABCD .....IX.^.....
```


Although not strictly part of profile 1, data MDTs are optional. These are used to optimize bandwidth by only sending traffic to PEs that want it. The MP2MP tree is bidirectional and when traffic is sent onto that tree, the core routers consult their LFIB and deliver traffic to each node. PEs without C-multicast clients interested in that C-group will simply drop the traffic. The benefit of only using MP2MP trees is that it minimizes state since there is one delivery tree. It is best used for low-volume/inclusive traffic, whereas data MDTs are more appropriate for high-volume/selective traffic. In this example, only CSR4 and CSR3 request a high-volume SSM flow using C-group 232.0.0.5. This is the only high-volume flow in the network; all other SSM and ASM flows should use the MP2MP tree. We will instruct all PEs routers to allow a maximum of 10 data MDTs using only this C-group. I use SSM for variation, not out of necessity.

```
! XE PE configuration
ip access-list extended ACL_DATA_MDT
 permit ip any host 232.0.0.5

vrf definition MC
 address-family ipv4
  mdt data mpls mldp 10 list ACL_DATA_MDT immediate-switch

! XRV2 PE configuration
ipv4 access-list ACL_DATA_MDT
 10 permit ipv4 any host 232.0.0.5

multicast-routing
 vrf MC
  address-family ipv4
   mdt data mldp 10 ACL_DATA_MDT immediate-switch
```

This will spawn a new MDT in the core of the network, with a non-zero identifier. Since PIM is still used for C-multicast signaling, the PIM TLV process reviewed in profile 0 is used again. Looking at CSR6's MRIB topology, we can quickly figure out the new MDT number and look into the mLDp database. Notice that the RPF interface for the C(S,G) join is the same, but the RPF neighbor is CSR7 since this is an SSM join. The MDT number is 1. The Y/y flags mean joined/sending to MDT data groups, respectively.

```
R6#show ip mroute vrf MC ssm
(10.2.7.2, 232.0.0.5), 02:10:05/00:02:37, flags: sTIY
  Incoming interface: Lspvif0, RPF nbr 213.7.7.7, MDT: [1, 213.7.7.7]/
00:02:54
  Outgoing interface list:
    GigabitEthernet2.546, Forward/Sparse, 02:10:05/00:02:37
```

CSR6 indicates that traffic sent downstream from the root (CSR7) towards the customer should use label 6000. This was advertised to XRV3. Notice there is no outgoing/upstream label. Data MDTs are not only P2MP, they are unidirectional. A downstream LSR can never send data towards the root of the tree.

```

R6#show mpls mldp database opaque_type mdt 213:213 1
LSM ID : 7   Type: P2MP   Uptime : 01:33:18
FEC Root      : 213.7.7.7
Opaque decoded : [mdt 213:213 1]
Opaque length  : 11 bytes
Opaque value   : 02 000B 0002130000021300000001
Upstream client(s) :
  213.13.13.13:0 [Active]
    Expires      : Never      Path Set ID : 7
    Out Label (U) : None      Interface  : GigabitEthernet2.563*
    Local Label (D) : 6000     Next Hop   : 213.6.13.13
Replication client(s):
  MDT (VRF MC)
    Uptime      : 01:33:18    Path Set ID : None
    Interface   : Lspvif0

```

On XRv3, we will look at its P2MP trees, since we know there is only one. The output doesn't even show an upstream label field, let alone a value. The remote label from CSR6 is 6000, which is what XRv3 will use to deliver LSM towards CSR6 (downstream away from the root). The local label 93002 is what XRv3 advertised to CSR7 to receive this downstream traffic. XRv3 doesn't need to replicate anything because there is only one client, CSR6. If there were more clients, their label mapping messages would intersect the existing P2MP tree somewhere and be grafted (like tree branches). The LFIB does not list XRv2 for this MDT as expected.

```

RP/0/0/CPU0:XRv3#show mpls mldp database p2mp
mLDP database
LSM-ID: 0x00002   Type: P2MP   Uptime: 01:35:51
FEC Root      : 213.7.7.7
Opaque decoded : [mdt 213:213 1]
Upstream neighbor(s) :
  213.7.7.7:0 [Active] Uptime: 01:35:51
    Local Label (D) : 93002
Downstream client(s):
  LDP 213.6.6.6:0   Uptime: 01:35:51
    Next Hop        : 213.6.13.6
    Interface       : GigabitEthernet0/0/0/0.563
    Remote label (D) : 6000

```

```

RP/0/0/CPU0:XRv3#show mpls forwarding labels 93002
Local   Outgoing   Prefix           Outgoing         Next Hop          Bytes
Label   Label      or ID            Interface        Interface        Switched
-----
93002   6000           MLDP: 0x00002    Gi0/0/0/0.563   213.6.13.6       0

```

We will use a new command on CSR7 as a quick verification. Checking the mLDP bindings shows the downstream labels used to deliver traffic. CSR7 has to replicate it towards XRv3 and XRv4. Notice label 93002 in use for traffic sent towards XRv3 in this VPN as it matches the verification performed above.

We also check the mLDP database summary which shows us all the roots and their VPNs, along with client counts. There can be multiple roots in the network, even for the same VPN.

```
R7#show mpls mldp bindings opaque_type mdt 213:213 1
System ID: 3
Type: P2MP, Root Node: 213.7.7.7, Opaque Len: 14
Opaque value: [mdt 213:213 1]
lsr: 213.14.14.14:0, remote binding[D]: 94018
lsr: 213.13.13.13:0, remote binding[D]: 93002
```

```
R7#show mpls mldp database summary
```

LSM ID	Type	Root	Decoded Opaque Value	Client
3	P2MP	213.7.7.7	[mdt 213:213 1]	3
1	MP2MP	213.14.14.14	[mdt 213:213 0]	1

We will send traffic from CSR2 again, this time to the SSM group. We expect CSR6 and CSR8 to see packets being forwarded to their customers.

```
R6#show ip mroute vrf MC 232.0.0.5 count | begin Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 48, Packets received:
48
Source: 10.2.7.2/32, Forwarding: 48/1/122/0, Other: 48/0/0
```

```
R8#show ip mroute vrf MC 232.0.0.5 count | begin Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 60, Packets received:
60
Source: 10.2.7.2/32, Forwarding: 60/1/122/0, Other: 60/0/0
```

Earlier, I mentioned there can be multiple roots. Specifically, we can manually configure multiple roots for the MP2MP tree also. Both of them use the MDT ID of 0. On CSR6 and CSR7, I temporarily configure CSR5 as a MP2MP root in addition to XRv4. Examining CSR7, we can prove this.

```
! CSR6 and CSR7
vrf definition MC
address-family ipv4
mdt default mpls mldp 213.5.5.5
address-family ipv6
mdt default mpls mldp 213.5.5.5
```

```
R7#show mpls mldp database summary
```

LSM ID	Type	Root	Decoded Opaque Value	Client
3	P2MP	213.7.7.7	[mdt 213:213 1]	3
4	MP2MP	213.5.5.5	[mdt 213:213 0]	1
1	MP2MP	213.14.14.14	[mdt 213:213 0]	1

CSR7 selects XRv4 as the root, not wanting to use CSR5. This is because the lowest cost to the root, when multiple roots are configured, is the default behavior. CSR5 is a hot-standby, since the label mapping exchange already occurred. This secondary root is listed as a “candidate”.

```
R7#show mpls mldp database opaque_type mdt 213:213 0 | include RNR|FEC_Root
LSM ID : 4 (RNR LSM ID: 2)    Type: MP2MP    Uptime : 00:01:51
    FEC Root                  : 213.5.5.5
    RNR active LSP            : 1 (root: 213.14.14.14)
LSM ID : 1 (RNR LSM ID: 2)    Type: MP2MP    Uptime : 03:11:48
    FEC Root                  : 213.14.14.14
    RNR active LSP            : (this entry)
    Candidate RNR ID(s) : 4
```

```
R7#show mpls mldp root 213.14.14.14 | include Metric
    Metric : 10
R7#show mpls mldp root 213.5.5.5 | include Metric
    Metric : 20
```

In the case of a tie, the highest IP address wins. In this case, XRv4 remains the root. The LSM-ID 8 output shows that “A” is the active LSP. This hexadecimal number 0xA represents the second database entry that maps to 213.14.14.14 as the root.

```
R6#show mpls mldp database opaque_type mdt 213:213 0 | include RNR|FEC_Root
LSM ID : 8 (RNR LSM ID: 6)    Type: MP2MP    Uptime : 00:15:40
    FEC Root                  : 213.5.5.5
    RNR active LSP            : A (root: 213.14.14.14)
LSM ID : A (RNR LSM ID: 6)    Type: MP2MP    Uptime : 00:01:45
    FEC Root                  : 213.14.14.14
    RNR active LSP            : (this entry)
    Candidate RNR ID(s) : 8
```

```
R6#show mpls mldp root 213.14.14.14 | include Metric
    Metric : 20
R6#show mpls mldp root 213.5.5.5 | include Metric
    Metric : 20
```

To make things complicated, I increase the IS-IS cost on XRv3’s link to XRv4 from 10 to 15. This makes CSR6 choose CSR5 as its root, while CSR7 still selects XRv4. In fact, every other router selects XRv4, since no one else is configured to use CSR5. At a glance, it appears like CSR6 will not have reachability to any other nodes since they cannot agree on the MP2MP root.

```
R6#show mpls mldp root 213.14.14.14 | include Metric
    Metric : 25
R6# show mpls mldp root 213.5.5.5 | include Metric
    Metric : 20

R6# show mpls mldp database opaque_type mdt 213:213 0 | include RNR|FEC_Root
```

```

LSM ID : 8 (RNR LSM ID: 6)    Type: MP2MP    Uptime : 00:17:30
  FEC Root      : 213.5.5.5
  RNR active LSP : (this entry)
Candidate RNR ID(s): A
LSM ID : A (RNR LSM ID: 6)    Type: MP2MP    Uptime : 00:03:36
  FEC Root      : 213.14.14.14
  RNR active LSP : 8 (root: 213.5.5.5)

```

From CSR6's perspective, nothing looks wrong as it sees all its PIM neighbors. Recall that PIM neighbors are unidirectional, and the reception of a PIM hello is sufficient to form a neighborhood. This implies that every other router is able to reach CSR6.

```

R6#show ip pim vrf MC neighbor | begin ^Neighbor
Neighbor          Interface          Uptime/Expires    Ver    DR
Address
213.12.12.12      Lspvif0                          04:50:19/00:01:32 v2      1 / DR P G
213.5.5.5         Lspvif0                          04:50:20/00:01:42 v2      1 / S P G
213.7.7.7         Lspvif0                          04:50:20/00:01:43 v2      1 / S P G
213.1.1.1         Lspvif0                          04:50:20/00:01:19 v2      1 / S P G
213.8.8.8         Lspvif0                          04:50:20/00:01:18 v2      1 / S P G

```

The issue is that CSR6 is not able to reach all the other PEs. Specifically, it cannot reach XRv2, CSR1, or CSR8. Tracing the MP2MP tree from CSR6, CSR6 is using label 93007 to send traffic upstream towards CSR5 via XRv3. I use the ID number of 8, seen earlier, to reference this MP2MP tree.

```

R6#show mpls mldp database id 8 | section Upstream
Upstream client(s) :
  213.13.13.13:0    [Active]
    Expires         : Never          Path Set ID   : 8
    Out Label (U)   : 93007          Interface     : GigabitEthernet2.563*
    Local Label (D): 6002            Next Hop      : 213.6.13.13

```

XRv3 forwards this towards the root, which is CSR5, and also down to CSR7, which is part of the same MP2MP tree. XRv3 does not forward it to XRv2 since that PE never send a label mapping for the MP2MP tree using CSR5 as the root. Thus, XRv2 cannot see PIM hellos from CSR6.

```

RP/0/0/CPU0:XRv3#show mpls forwarding labels 93007
Local  Outgoing  Prefix          Outgoing        Next Hop        Bytes
Label  Label       or ID           Interface       Interface       Switched
-----
93007  5001         MLDP: 0x00003   Gi0/0/0/0.553  213.5.13.5      0
       7006         MLDP: 0x00003   Gi0/0/0/0.573  213.7.13.7      0

```

Knowing that CSR7 is a leaf in the tree, we normally expect CSR5 (the root) to forward the packet onward towards the other side of the network. However, CSR1 and CSR8 did not issue label mappings for this MP2MP tree, and thus no "branches" exist from CSR5. The only downstream client is XRv3 with

inbound traffic using label 5001 as expected. This means CSR1 and CSR8 will never see traffic from CSR6, but CSR7 will. CSR5 also will not see traffic from CSR6; even though it's the root, it did not configure itself as the root. In performing PE duties, it is not aware of that traffic coming from CSR6 within the MVPN as a result.

```
R5#show mpls mldp database summary
```

LSM ID	Type	Root	Decoded Opaque Value	Client
3	MP2MP	213.5.5.5	[mdt 213:213 0]	1
1	MP2MP	213.14.14.14	[mdt 213:213 0]	1

```
R5#show mpls mldp database id 3 | section Replication
```

```
Replication client(s):
```

```
213.13.13.13:0
```

```
Uptime : 03:00:24
```

```
Path Set ID : 4
```

```
Out label (D) : 93010
```

```
Interface : GigabitEthernet2.553*
```

```
Local label (U): 5001
```

```
Next Hop : 213.5.13.13
```

We verify our findings by checking PIM neighbors on all the PEs. As expected, only CSR7 have a PIM neighborship with CSR6. More correctly, the opposite is true regarding reachability, since CSR6 can only reach CSR7. Everyone can reach CSR6. A node only sends traffic upstream to one root, but is capable of receiving traffic from all configured roots. That is why using multiple roots works, so long as everyone has the same list of roots. The LSPs are signaled to each root at the time of configuration to enable this failover behavior.

```
RP/0/0/CPU0:XRv2#show pim vrf MC neighbor | include 213.6.6.6
```

```
[no output]
```

```
R5#show ip pim vrf MC neighbor | include 213.6.6.6
```

```
[no output]
```

```
R1#show ip pim vrf MC neighbor | include 213.6.6.6
```

```
[no output]
```

```
R8#show ip pim vrf MC neighbor | include 213.6.6.6
```

```
[no output]
```

```
R7#show ip pim vrf MC neighbor | include 213.6.6.6
```

```
213.6.6.6 Lspvif0 05:05:01/00:01:24 v2 1 / S P G
```

To fix it, we will configure every node to agree on the list of roots. For simplicity, I will remove the CSR5 root from CSR6 and CSR7, as well as revert the IS-IS cost on XRv3. Multiple mLDP root testing is not the focus of this lab. Next, we will retest this MVPN profile with IPv6. We will use XRv4 as root on all devices and use Embedded RP for variety. CSR2 will be the Embedded RP for all IPv6 ASM traffic; for this test, we will use the ASM group FF7E:240:2001:10:2:7::1 and SSM group FF33::1. The SSM group will use the MP2MP default tree and the ASM group will use the P2MP data trees (opposite of what we tested

earlier). The VPN ID does not change but it shown again because without it, mLDP does not work at all. It is critical to identify a VPN-ID under the VRF. Because the MP2MP tree carries the FEC information of the VPN ID and root address, there is no differentiation between IPv4 and IPv6, so the roots must match between the two. If we wanted to use multiple roots, we would have to do so consistently between both AFs.

```
! XE PE configuration on all PEs
ipv6 access-list ACL_DATA_MDT_IPV6
  permit ipv6 any FF7E:240:2001:10:2:7::/96

vrf definition MC
  vpn id 213:213
  address-family ipv6
    mdt preference mldp
    mdt default mpls mldp 213.14.14.14
    mdt data mpls mldp 10 list ACL_DATA_MDT_IPV6 immediate-switch

! XRv2 PE configuration
ipv6 access-list ACL_DATA_MDT_IPV6
  10 permit ipv6 any ff7e:240:2001:10:2:7::/96

multicast-routing
  vrf MC
    address-family ipv6
      mdt default mldp ipv4 213.14.14.14
      mdt data mldp 10 ACL_DATA_MDT_IPV6 immediate-switch
```

We should verify that our IPv6 PIM neighbors have formed. We see the “emulated LAN” behavior in play; even though the unicast RTs did not create this full-mesh of L3VPN reachability, the VPN ID is the same on all of the nodes.

```
RP/0/0/CPU0:XRv2#show pim vrf MC ipv6 neighbor | begin Lmdt
LmdtMC
Neighbor Address                               Uptime    Expires   DR pri   DR
Flags
::ffff:213.1.1.1                               00:03:58  00:01:43  1
::ffff:213.5.5.5                               00:03:58  00:01:43  1
::ffff:213.6.6.6                               00:03:57  00:01:44  1
::ffff:213.7.7.7                               00:03:58  00:01:43  1
::ffff:213.8.8.8                               00:03:58  00:01:42  1
::ffff:213.12.12.12*                          00:04:01  00:01:32  1          (DR)   P

R5#show ipv6 pim vrf MC neighbor | begin ^Neighbor
Neighbor Address      Interface      Uptime    Expires   Mode DR pri
::FFFF:213.1.1.1      Lspvif0       00:06:13  00:01:41  B G    1
::FFFF:213.6.6.6      Lspvif0       00:06:17  00:01:42  B G    1
::FFFF:213.7.7.7      Lspvif0       00:06:17  00:01:41  B G    1
```

```

::FFFF:213.8.8.8      Lspvif0      00:06:18  00:01:39 B G      1
::FFFF:213.12.12.12   Lspvif0      00:04:31  00:01:30 B G     DR 1

```

R1#show ipv6 pim vrf MC neighbor | begin ^Neighbor

```

Neighbor Address      Interface      Uptime      Expires     Mode DR pri
::FFFF:213.5.5.5      Lspvif0      00:06:47    00:01:36 B G      1
::FFFF:213.6.6.6      Lspvif0      00:06:47    00:01:37 B G      1
::FFFF:213.7.7.7      Lspvif0      00:06:47    00:01:37 B G      1
::FFFF:213.8.8.8      Lspvif0      00:06:46    00:01:35 B G      1
::FFFF:213.12.12.12   Lspvif0      00:05:05    00:01:26 B G     DR 1

```

R6#show ipv6 pim vrf MC neighbor | begin ^Neighbor

```

Neighbor Address      Interface      Uptime      Expires     Mode DR pri
::FFFF:213.1.1.1      Lspvif0      00:06:33    00:01:21 B G      1
::FFFF:213.5.5.5      Lspvif0      00:06:38    00:01:21 B G      1
::FFFF:213.7.7.7      Lspvif0      00:07:25    00:01:21 B G      1
::FFFF:213.8.8.8      Lspvif0      00:07:20    00:01:20 B G      1
::FFFF:213.12.12.12   Lspvif0      00:04:51    00:01:41 B G     DR 1

```

R7#show ipv6 pim vrf MC neighbor | begin ^Neighbor

```

Neighbor Address      Interface      Uptime      Expires     Mode DR pri
FE80::2              Gi2.527       00:19:29    00:01:30 B G      1
::FFFF:213.1.1.1      Lspvif0      00:06:45    00:01:39 B G      1
::FFFF:213.5.5.5      Lspvif0      00:06:50    00:01:38 B G      1
::FFFF:213.6.6.6      Lspvif0      00:07:36    00:01:40 B G      1
::FFFF:213.8.8.8      Lspvif0      00:07:31    00:01:37 B G      1
::FFFF:213.12.12.12   Lspvif0      00:05:02    00:01:29 B G     DR 1

```

R8#show ipv6 pim vrf MC neighbor | begin ^Neighbor

```

Neighbor Address      Interface      Uptime      Expires     Mode DR pri
::FFFF:213.1.1.1      Lspvif0      00:06:42    00:01:42 B G      1
::FFFF:213.5.5.5      Lspvif0      00:06:47    00:01:42 B G      1
::FFFF:213.6.6.6      Lspvif0      00:07:28    00:01:43 B G      1
::FFFF:213.7.7.7      Lspvif0      00:07:28    00:01:42 B G      1
::FFFF:213.12.12.12   Lspvif0      00:04:59    00:01:32 B G     DR 1

```

We will configure MLD joins on all of the PEs to request traffic for FF33:1 from CSR4 and this will use the MP2MP default tree. CSR2 and CSR3 will issue MLD joins for FF7E:240:2001:10:2:7::1. The MLD configuration is not shown, but we verify the PIMv6 signaling between the PEs. First, we will verify the top three PEs (XRv2, CSR5, and CSR1). All of them receive the MLDv2 join from their CEs, but their RPF interface is null. The reason for this is that none of them have a unicast IPv6 route to 2001:10:4:6::4, because the unicast topology does not allow it. Their OILs are properly built, but ultimately traffic cannot flow towards the customers, even if it is flooded to the PEs via the MP2MP tree. There is nothing magical about MVPN as it still relies heavily on converged unicast VPN routing.

```

R1#show ipv6 mroute vrf MC FF33::1 | begin \
(2001:10:4:6::4, FF33::1), 00:04:57/never, flags: sTI
Incoming interface: Null

```



```
RPF nbr: ::
```

```
Immediate Outgoing interface list:
```

```
GigabitEthernet2.519, Forward, 00:04:57/never
```

```
R5#show ipv6 mroute vrf MC FF33::1 | begin \  
(2001:10:4:6::4, FF33::1), 00:06:25/never, flags: sTI
```

```
Incoming interface: Null
```

```
RPF nbr: ::
```

```
Immediate Outgoing interface list:
```

```
GigabitEthernet2.550, Forward, 00:06:25/never
```

```
RP/0/0/CPU0:XRv2#show pim vrf MC ipv6 topology | begin SPT$
```

```
(2001:10:4:6::4, ff33::1) SPT
```

```
SSM Up: 00:09:05 JP: Join(00:00:59) Flags:
```

```
RPF: Null, ::
```

```
GigabitEthernet0/0/0/0.512 00:09:05 fwd LI LH
```

Next, we verify CSR7 and CSR8, which have unicast reachability back to CSR4. Their C(S,G) trees are properly constructed. A quick verification on CSR6 shows the LSPvif as the “LAN” interface in the OIL, as expected. This virtual interface represents the PMSI.

```
R7#show ipv6 mroute vrf MC FF33::1 | begin \  
(2001:10:4:6::4, FF33::1), 00:00:31/never, flags: sTI
```

```
Incoming interface: Lspvif0
```

```
RPF nbr: ::FFFF:213.6.6.6
```

```
Immediate Outgoing interface list:
```

```
GigabitEthernet2.527, Forward, 00:00:31/never
```

```
R8#show ipv6 mroute vrf MC FF33::1 | begin \  
(2001:10:4:6::4, FF33::1), 00:00:43/never, flags: sTI
```

```
Incoming interface: Lspvif0
```

```
RPF nbr: ::FFFF:213.6.6.6
```

```
Immediate Outgoing interface list:
```

```
GigabitEthernet2.538, Forward, 00:00:43/never
```

```
R6#show ipv6 mroute vrf MC FF33::1 | begin \  
(2001:10:4:6::4, FF33::1), 00:01:32/00:03:10, flags: sT
```

```
Incoming interface: GigabitEthernet2.546
```

```
RPF nbr: 2001:10:4:6::4
```

```
Immediate Outgoing interface list:
```

```
Lspvif0, Forward, 00:01:32/00:03:10
```

Traffic sent to LSPvif0 will be send down the MP2MP tree, ultimately reaching all destinations. We acknowledge this is wasteful in this scenario, because there is no connectivity between some receivers and the source. We can prove that mLDP is not smart enough to understand this by tracing the LSP. CSR6’s mLDP database uses upstream label 93004 to send traffic towards the root. XRv3 is the next-hop.

```
R6#show mpls mldp database opaque_type mdt 213:213 0 | section Upstream
```

```

Upstream client(s) :
  213.13.13.13:0    [Active]
    Expires          : Never          Path Set ID   : B
    Out Label (U)    : 93004          Interface     : GigabitEthernet2.563*
    Local Label (D)  : 6004           Next Hop      : 213.6.13.13

```

On XRv3, the mLDP database shows XRv2 as another downstream client of the MP2MP tree, and replicates traffic towards it using label 92003. We verify this by checking the LFIB also. At the same time, XRv3 sends traffic upstream to the root using XRv4's local label of 94007.

```

RP/0/0/CPU0:XRv3#show mpls mldp database opaque type mdt 213:213 0
mLDP database
LSM-ID: 0x00001   Type: MP2MP   Uptime: 18:45:34
FEC Root          : 213.14.14.14
Opaque decoded     : [mdt 213:213 0]
Upstream neighbor(s) :
  213.14.14.14:0 [Active] Uptime: 17:46:19
    Next Hop       : 213.13.14.14
    Interface      : GigabitEthernet0/0/0/0.534
    Local Label (D) : 93005          Remote Label (U) : 94007
Downstream client(s):
  LDP 213.6.6.6:0   Uptime: 15:29:10
    Next Hop        : 213.6.13.6
    Interface       : GigabitEthernet0/0/0/0.563
    Remote label (D) : 6004          Local label (U) : 93004
  LDP 213.12.12.12:0 Uptime: 17:46:40
    Next Hop        : 213.12.13.12
    Interface       : GigabitEthernet0/0/0/0.523
    Remote label (D) : 92003          Local label (U) : 93012

```

```

RP/0/0/CPU0:XRv3#show mpls forwarding labels 93004
Local  Outgoing  Prefix      Outgoing  Next Hop  Bytes
Label  Label     or ID       Interface                Switched
-----
93004  92003        MLDP: 0x00001  Gi0/0/0/0.523  213.12.13.12  0
      94007        MLDP: 0x00001  Gi0/0/0/0.534  213.13.14.14  0

```

Traffic reaching XRv4 from XRv3 (using label 94007) is replicated to many downstream peers. Specifically, it includes CSR1, CSR5, CSR7, and CSR8. Only the last two will be able to forward the traffic further. These four downstream peers are all listed in the LFIB also.

```

RP/0/0/CPU0:XRv4#show mpls mldp database opaque type mdt 213:213 0 | begin
Down
Downstream client(s):
  LDP 213.1.1.1:0   Uptime: 17:46:49
    Next Hop        : 213.1.14.1
    Interface       : GigabitEthernet0/0/0/0.514
    Remote label (D) : 1009          Local label (U) : 94016

```

```

LDP 213.5.5.5:0      Uptime: 17:46:49
  Next Hop           : 213.5.14.5
  Interface          : GigabitEthernet0/0/0/0.554
  Remote label (D)   : 5015                      Local label (U) : 94015
LDP 213.7.7.7:0      Uptime: 17:46:49
  Next Hop           : 213.7.14.7
  Interface          : GigabitEthernet0/0/0/0.574
  Remote label (D)   : 7004                      Local label (U) : 94017
LDP 213.8.8.8:0      Uptime: 17:46:49
  Next Hop           : 213.8.14.8
  Interface          : GigabitEthernet0/0/0/0.584
  Remote label (D)   : 8035                      Local label (U) : 94011
LDP 213.13.13.13:0   Uptime: 17:46:49
  Next Hop           : 213.13.14.13
  Interface          : GigabitEthernet0/0/0/0.534
  Remote label (D)   : 93005                      Local label (U) : 94007

```

RP/0/0/CPU0:XRv4#show mpls forwarding labels 94007

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94007	1009	MLDP: 0x00003	Gi0/0/0/0.514	213.1.14.1	0
	5015	MLDP: 0x00003	Gi0/0/0/0.554	213.5.14.5	0
	7004	MLDP: 0x00003	Gi0/0/0/0.574	213.7.14.7	0
	8035	MLDP: 0x00003	Gi0/0/0/0.584	213.8.14.8	0

CSR4 will begin sending traffic now. Examining XRv2, CSR5, and CSR1, they show RPF failures when traffic arrives, since there is no RPF interface but the C(S,G) matches the C-MRIB. Because MVPN LSM isn't supported on egress XRv PEs, all the counters are zero, but normally it would show here. The RPF counters are proof that mLDP is flooding the traffic throughout the entire MP2MP tree.

R4#ping FF33::1 repeat 10

Output Interface: GigabitEthernet2.546

R5#show ipv6 mroute vrf MC FF33::1 count

Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second

Other counts: Total/RPF failed/Other drops(OIF-null, rate-limit etc)

VRF MC

53 routes, 6 (*,G)s, 46 (*,G/m)s

Group: FF33::1

Source: 2001:10:4:6::4,

SW Forwarding: 0/0/0/0, Other: 0/0/0

HW Forwarding: 0/0/0/0, Other: 10/10/0

Totals - Source count: 1, Packet count: 0

R1#show ipv6 mroute vrf MC FF33::1 count

Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second

Other counts: Total/RPF failed/Other drops(OIF-null, rate-limit etc)

```

VRF MC
 53 routes, 6 (*,G)s, 46 (*,G/m)s
Group: FF33::1
  Source: 2001:10:4:6::4,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 0/0/0/0, Other: 10/10/0
  Totals - Source count: 1, Packet count: 0

RP/0/0/CPU0:XRv2#show mfib vrf MC ipv6 route FF33::1/128 | begin Failure
Failure Counts: RPF / TTL / Empty Olist / Encap RL / Other
(2001:10:4:6::4,ff33::1)
  Flags:
  Up: 00:39:35
  Last Used: never
  SW Forwarding Counts: 0/0/0
  SW Replication Counts: 0/0/0
  SW Failure Counts: 0/0/0/0/0
  GigabitEthernet0/0/0/0.512 Flags: NS EG, Up:00:39:35

```

The operation was successful on CSR7 and CSR8, as expected.

```

R8#show ipv6 mroute vrf MC FF33::1 count
Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second
Other counts:      Total/RPF failed/Other drops(OIF-null, rate-limit etc)
VRF MC
 53 routes, 6 (*,G)s, 46 (*,G/m)s
Group: FF33::1
  Source: 2001:10:4:6::4,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 10/0/126/0, Other: 0/0/0
  Totals - Source count: 1, Packet count: 10

R7#show ipv6 mroute vrf MC FF33::1 count
Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second
Other counts:      Total/RPF failed/Other drops(OIF-null, rate-limit etc)
VRF MC
 53 routes, 6 (*,G)s, 46 (*,G/m)s
Group: FF33::1
  Source: 2001:10:4:6::4,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 10/0/126/0, Other: 0/0/0
  Totals - Source count: 1, Packet count: 10

```

Next, we test IPv6 ASM using the P2MP data MDT. Knowing that the top half of the network does not have a route to the Embedded RP, I won't configure the MLD join on them; we already tested the failures there. Instead, CSR3 joins the group FF7E:240:2001:10:2:7::1 and CSR8 sends a C(*,G) join towards the C-RP, which is CSR2.

```

R8#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 | begin \
(*, FF7E:240:2001:10:2:7:0:1), 00:00:26/never, RP 2001:10:2:7::2, flags: SCJ
Incoming interface: Lspvif0
RPF nbr: ::FFFF:213.7.7.7
Immediate Outgoing interface list:
    GigabitEthernet2.538, Forward, 00:00:26/never

R7#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 | begin \
(*, FF7E:240:2001:10:2:7:0:1), 00:01:57/00:02:36, RP 2001:10:2:7::2, flags: S
Incoming interface: GigabitEthernet2.527
RPF nbr: FE80::2
Immediate Outgoing interface list:
    Lspvif0, Forward, 00:01:57/00:02:36

```

CSR2 is the C-RP, and is the root of the C(*,G) tree. Its incoming interface is the PIM decapsulation tunnel, which was automatically created.

```

R2#show ipv6 mroute FF7E:240:2001:10:2:7::1 | begin \
(*, FF7E:240:2001:10:2:7:0:1), 00:02:31/00:03:00, RP 2001:10:2:7::2, flags: S
Incoming interface: Tunnel2
RPF nbr: 2001:10:2:7::2
Immediate Outgoing interface list:
    GigabitEthernet2.527, Forward, 00:02:31/00:03:00

R2#show ipv6 pim tunnel tunnel2
Tunnel2*
Type   : PIM Decap
RP     : 2001:10:2:7::2*
Source: -

```

```

R2#show derived-config interface tunnel2
interface Tunnel2
description Pim Register Tunnel (Decap) for RP 2001:10:2:7::2
no ip address
ipv6 unnumbered GigabitEthernet2.527
ipv6 enable
tunnel source GigabitEthernet2.527
tunnel destination 2001:10:2:7::2

```

At this point, no new mLDP P2MP trees have been built, despite us specifying so in the ACL. This is because a C(S,G) join is generally required for the P2MP tree to be built, since the intent is to build it directly to the source. This is why P2MP mLDP trees are better suited for SSM than ASM. As observed earlier, Embedded RP does not appear to work in a VRF when deployed with MVPN on XE. The PE never sends the register message. The configuration looks right, it just never works. There is no debug output on CSR2 (C-RP), nor any MPLS unicast packets seen in the core during this process.

```

CSR6#debug ipv6 pim

```

```
IPv6 PIM: [MC] (2001:10:4:6::4, FF7E:240:2001:10:2:7:0:1/128)
GigabitEthernet2.546 MRIB update (f=20,c=20)
IPv6 PIM: [MC] (2001:10:4:6::4, FF7E:240:2001:10:2:7:0:1) Signal present on
GigabitEthernet2.546
IPv6 PIM: [MC] (2001:10:4:6::4, FF7E:240:2001:10:2:7:0:1) Create entry
IPv6 PIM: [MC] (2001:10:4:6::4, FF7E:240:2001:10:2:7:0:1) RPF changed from
::/- to 2001:10:4:6::4/GigabitEthernet2.546
IPv6 PIM: [MC] (2001:10:4:6::4, FF7E:240:2001:10:2:7:0:1) Connected status
changed from off to on
IPv6 PIM: [MC] (2001:10:4:6::4, FF7E:240:2001:10:2:7:0:1) Start registering to
2001:10:2:7::2
```

```
R6#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 | begin \
(2001:10:4:6::4, FF7E:240:2001:10:2:7:0:1), 00:03:43/00:00:51, flags: SFT
Incoming interface: GigabitEthernet2.546
RPF nbr: 2001:10:4:6::4, Registering
Immediate Outgoing interface list:
Tunnell, Forward, 00:03:43/never
```

For the sake of actually seeing this work, I configure the RP statically on CSR6 only, disable embedded RP support, and set a valid register source just to be certain. This was the same workaround conducted for profile 0.

```
! CSR6
ipv6 pim vrf MC rp-address 2001:10:2:7::2
ipv6 pim vrf MC register-source GigabitEthernet2.546
no ipv6 pim vrf MC rp embedded
```

Now, the first-hop router can successfully register traffic with the C-RP. The little 'y' flag means sending traffic to an MDT data group. This implies the P2MP MDT was built, but we will verify that later. We also know the MP2MP is not in use because the MDT ID is non-zero and a new LSM-ID was assigned. The counters show several packets forwarded also.

```
R6#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 | begin \
(2001:10:4:6::4, FF7E:240:2001:10:2:7:0:1), 00:19:10/00:00:33, flags: SFTy
Incoming interface: GigabitEthernet2.546
RPF nbr: 2001:10:4:6::4
MDT TX nr: 2147483649 (0x80000001) LSM-ID: 0xD
Immediate Outgoing interface list:
Tunnell, Forward, 00:19:10/never
Lspvif0, Forward, 00:14:47/00:03:05
```

```
R6#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 2001:10:4:6::4 count
Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second
Other counts:      Total/RPF failed/Other drops(OIF-null, rate-limit etc)
VRF MC
54 routes, 6 (*,G)s, 46 (*,G/m)s
Group: FF7E:240:2001:10:2:7:0:1
```

```
Source: 2001:10:4:6::4,
SW Forwarding: 0/0/0/0, Other: 0/0/0
HW Forwarding: 786/1/118/0, Other: 10/0/10
Totals - Source count: 1, Packet count: 786
```

Checking the C-RP quickly, we see the C(S,G) entry so we know registration was successful. Because CSR2 is not in the transit path between CSR4 and CSR3, it is pruned from the C(S,G) tree.

```
R2#show ipv6 mroute FF7E:240:2001:10:2:7:0:1 | begin \
(*, FF7E:240:2001:10:2:7:0:1), 00:45:09/00:03:22, RP 2001:10:2:7::2, flags: S
Incoming interface: Tunnel2
RPF nbr: 2001:10:2:7::2
Immediate Outgoing interface list:
GigabitEthernet2.527, Forward, 00:45:09/00:03:22
```

```
(2001:10:4:6::4, FF7E:240:2001:10:2:7:0:1), 00:02:10/00:03:10, RP
2001:10:2:7::2, flags: SPR
Incoming interface: Tunnel2
RPF nbr: 2001:10:2:7::2
Immediate Outgoing interface list:
GigabitEthernet2.527, Null, 00:02:10/00:03:22
```

The egress PE shows traffic being received from the LSPvif and forwarded to the customer correctly. The same can be observed on CSR7. It is also considered an egress PE (tail-end of the P2MP tree), probably because the RP is behind it issuing C(S,G) joins, although the traffic is dropped due to a null OIL. It is awkward that the packet counters increase on CSR7 with a null OIL. The big 'Y' means traffic is being received from an MDT data group.

```
R8#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 | begin \
(*, FF7E:240:2001:10:2:7:0:1), 00:46:06/never, RP 2001:10:2:7::2, flags: SCJ
Incoming interface: Lspvif0
RPF nbr: ::FFFF:213.7.7.7
Immediate Outgoing interface list:
GigabitEthernet2.538, Forward, 00:46:06/never
```

```
(2001:10:4:6::4, FF7E:240:2001:10:2:7:0:1), 00:03:07/00:00:22, flags: SJY
Incoming interface: Lspvif0, MDT: [2147483649 (0x80000001),
213.6.6.6]/00:02:02
RPF nbr: ::FFFF:213.6.6.6
Inherited Outgoing interface list:
GigabitEthernet2.538, Forward, 00:46:06/never
```

```
R8#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 2001:10:4:6::4 count
Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second
Other counts:      Total/RPF failed/Other drops(OIF-null, rate-limit etc)
VRF MC
56 routes, 7 (*,G)s, 47 (*,G/m)s
```

```

Group: FF7E:240:2001:10:2:7:0:1
  Source: 2001:10:4:6::4,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 167/1/126/0, Other: 0/0/0
  Totals - Source count: 1, Packet count: 167

R7#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 | begin \
(*, FF7E:240:2001:10:2:7:0:1), 00:50:12/00:03:20, RP 2001:10:2:7::2, flags: S
  Incoming interface: GigabitEthernet2.527
  RPF nbr: FE80::2
  Immediate Outgoing interface list:
    Lspvif0, Forward, 00:50:12/00:03:20

(2001:10:4:6::4, FF7E:240:2001:10:2:7:0:1), 00:08:31/00:03:16, flags: STY
  Incoming interface: Lspvif0, MDT: [2147483649 (0x80000001),
213.6.6.6]/00:02:56
  RPF nbr: ::FFFF:213.6.6.6
  Outgoing interface list: Null

R7#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 2001:10:4:6::4 count
Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second
Other counts:      Total/RPF failed/Other drops(OIF-null, rate-limit etc)
VRF MC
  56 routes, 7 (*,G)s, 47 (*,G/m)s
Group: FF7E:240:2001:10:2:7:0:1
  Source: 2001:10:4:6::4,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 404/1/126/0, Other: 1/1/0
  Totals - Source count: 1, Packet count: 404

```

Before all of this occurred, the new P2MP tree was signaled. We will examine the debugs from CSR8. One significant difference between these debugs and the MP2MP debugs is that there is no upstream label mapping message from the core, as expected.

```

! CSR8
MLDP-MDT: [mdt 213:213 2147483649] wavl insert success
MLDP: LDP root 213.6.6.6 added
MLDP-DB: [mdt 213:213 2147483649] Added P2MP branch for MDT label
MLDP: Root 213.6.6.6 old paths: 0 new paths: 1
MLDP-DB: [mdt 213:213 2147483649] Changing peer from none to 213.14.14.14:0
MLDP-DB: [mdt 213:213 2147483649] Add accepting element nbr: 213.14.14.14:0
MLDP-MFI: For accepting element nbr: 213.14.14.14:0, [mdt 213:213 2147483649]
MFI allocated label 8038
MLDP: [mdt 213:213 2147483649] label mapping msg sent to 213.14.14.14:0
success
MLDP-DB: [mdt 213:213 2147483649] path to peer: 213.14.14.14:0 changed
None:0.0.0.0 to GigabitEthernet2.584:213.8.14

```


MLDP-MFI: After the MFI call For accepting element nbr: 213.14.14.14:0, Bind local label 8038 to **PSM ID: 6, returned: 8038**

Next, we will trace the LSP. Checking CSR8, we confirm a new P2MP tree was built.

```
R8#show mpls mldp database summary
```

LSM ID Cnt.	Type	Root	Decoded Opaque Value	Client
4	P2MP	213.7.7.7	[mdt 213:213 1]	1
5	P2MP	213.6.6.6	[mdt 213:213 2147483649]	1
1	MP2MP	213.14.14.14	[mdt 213:213 0]	1

Looking at the mLDP database, CSR8 advertises label 8038 so XRv4 can send traffic downstream. There is no upstream label since mLDP P2MP trees are downstream only.

```
R8#show mpls mldp database opaque_type mdt 213:213 2147483649
```

```
LSM ID : 5    Type: P2MP    Uptime : 00:03:40
FEC Root      : 213.6.6.6
Opaque decoded : [mdt 213:213 2147483649]
Opaque length  : 11 bytes
Opaque value   : 02 000B 0002130000021380000001
Upstream client(s) :
  213.14.14.14:0 [Active]
    Expires      : Never      Path Set ID : 5
    Out Label (U) : None      Interface  : GigabitEthernet2.584*
    Local Label (D) : 8038    Next Hop   : 213.8.14.14
Replication client(s) :
  MDT (VRF MC)
    Uptime      : 00:03:40    Path Set ID : None
    Interface   : Lspvif0
```

On the other end, CSR6 is the root of the new P2MP MDT.

```
R6#show mpls mldp database summary
```

LSM ID Cnt.	Type	Root	Decoded Opaque Value	Client
B	P2MP	213.7.7.7	[mdt 213:213 1]	1
D	P2MP	213.6.6.6	[mdt 213:213 2147483649]	2
A	MP2MP	213.14.14.14	[mdt 213:213 0]	1

There are no upstream clients from the root, and CSR6 replicates the traffic downstream to XRv3 using label 93007.

```
R6#show mpls mldp database opaque_type mdt 213:213 2147483649
```

```
LSM ID : D    Type: P2MP    Uptime : 00:03:52
FEC Root      : 213.6.6.6 (we are the root)
Opaque decoded : [mdt 213:213 2147483649]
```

```

Opaque length      : 11 bytes
Opaque value       : 02 000B 0002130000021380000001
Upstream client(s) :
  None
    Expires        : N/A                Path Set ID   : 10
Replication client(s):
  MDT (VRF MC)
    Uptime         : 00:03:52          Path Set ID   : None
    Interface      : Lspvif0
    213.13.13.13:0
    Uptime         : 00:03:52          Path Set ID   : None
    Out label (D)  : 93007             Interface    : GigabitEthernet2.563*
    Local label (U): None              Next Hop     : 213.6.13.13

```

Upon receipt of label 93007, XRv3 replicates the traffic to both CSR7 (C-RP is behind it) and XRv4 (downstream towards CSR8).

```

RP/0/0/CPU0:XRv3#show mpls forwarding labels 93007
Local  Outgoing  Prefix      Outgoing  Next Hop      Bytes
Label  Label      or ID       Interface  Hop           Switched
-----
93007  7008        MLDP: 0x00005  Gi0/0/0/0.573 213.7.13.7    0
          94019        MLDP: 0x00005  Gi0/0/0/0.534 213.13.14.14  0

```

Upon receipt of label 94019, XRv4 replicates the traffic to CSR8 (in front of the C-receiver).

```

RP/0/0/CPU0:XRv4#show mpls forwarding labels 94019
Local  Outgoing  Prefix      Outgoing  Next Hop      Bytes
Label  Label      or ID       Interface  Hop           Switched
-----
94019  8038        MLDP: 0x00005  Gi0/0/0/0.584 213.8.14.8    0

```

Last, we will examine OAM for MP2MP and P2MP trees in this profile. Unlike Profile 12 where we were limited only to unidirectional P2MP trees (the root specified had to be local), with MP2MP pings we can test from any endpoint; the tree is bidirectional. We specify the FEC and opaque type and can quickly discover all the endpoints in the MVPN on this MDT.

```

R6# show mpls mldp database summary
LSM ID    Type    Root          Decoded Opaque Value      Client
Cnt.
16        P2MP    213.7.7.7     [mdt 213:213 1]           1
13        MP2MP   213.14.14.14  [mdt 213:213 0]           1

```

```

R6#ping mpls mldp mp2mp 213.14.14.14 mdt 213:213 0
mp2mp Root node addr 213.14.14.14
Opaque type MDT, oui:index 0x213:0213, mdtnum 0
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,

```

```
        timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]
```

```
Request #1
! reply addr 213.12.13.12
! reply addr 213.7.13.7
! reply addr 213.1.14.1
! reply addr 213.5.14.5
! reply addr 213.8.14.8
```

```
Round-trip min/avg/max = 16/68/127 ms
```

```
R7#show mpls mldp database summary
```

LSM ID	Type	Root	Decoded Opaque Value	Client
Cnt.				
16	P2MP	213.7.7.7	[mdt 213:213 1]	3
13	MP2MP	213.14.14.14	[mdt 213:213 0]	2

```
R7#ping mpls mldp p2mp 213.7.7.7 mdt 213:213 1
p2mp Root node addr 213.7.7.7
Opaque type MDT, oui:index 0x213:0213, mdtnum 1
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
        timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]
```

```
Type escape sequence to abort.
```

```
Request #1
! reply addr 213.8.14.8
! reply addr 213.6.13.6
```

```
Round-trip min/avg/max = 126/136/146 ms
```

Additional Reading – Reference configurations "mvpn-1"

28.3 Profile 3: Default MDT – GRE – BGP-AD – PIM C-mcast Signaling

For this test, begin with the MVPN Profile 0 configuration using the Bidir model where PIM is set up already for both the customer and provider networks. The default MDT uses PIM-Bidir while the data MDTs use PIM-SSM. This will allow us to see more varieties in the P-tunnel types. The only significant difference here is enabling BGP auto-discovery (AD). This modification similar to jumping from profile 1 to profile 9 (using mLDP, not GRE). I do not anticipate seeing much benefit to this since the default MDT is still specified statically and signaling is done via PIM. IPv4 MDT SAFI is no longer needed; instead we configure MVPN.

```

! Basic XE configuration to enable BGP AD
vrf definition MC
  address-family ipv4
    mdt auto-discovery pim
  address-family ipv6
    mdt auto-discovery pim

router bgp 213
  no address-family ipv4 mdt
  address-family ipv4 mvpn
    neighbor 213.12.12.12 activate
    neighbor 213.12.12.12 send-community extended
  address-family ipv6 mvpn
    neighbor 213.12.12.12 activate
    neighbor 213.12.12.12 send-community extended

! Basic XR configuration to enable BGP AD (specific to XRv2 for BGP config)
route-policy RPL_PIM
  set core-tree pim-default
end-policy

multicast-routing
vrf MC
  address-family ipv4
    bgp auto-discovery pim
  address-family ipv6
    bgp auto-discovery pim

router pim
  address-family ipv4
    rpf topology route-policy RPL_PIM
  address-family ipv6
    rpf topology route-policy RPL_PIM

router bgp 213
  no address-family ipv4 mdt
  address-family ipv4 mvpn
  address-family ipv6 mvpn

  no address-family ipv4 mdt
  neighbor-group IBGP
    address-family ipv4 mvpn
    route-reflector-client
  address-family ipv6 mvpn
    route-reflector-client

vrf MC
  address-family ipv4 mvpn

```

```
address-family ipv6 mvpn
```

After configuring BGP, we look at CSR1 to see if it learned any BGP AD routes. It sees a Type-1 I-PMSI route from every PE in the MVPN (IPv4) and the same type of route for the “south-side” routers on IPv6. This is because the VPNv6 L3VPN is split horizontally. We also see a Type-3 S-PMSI route from CSR7 which describes a data MDT.

```
R1#show bgp ipv4 mvpn vrf MC | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 213:1 (default for vrf MC)
*> [1][213:1][213.1.1.1]/12
      0.0.0.0                      32768 ?
*>i [1][213:1][213.5.5.5]/12
      213.5.5.5                      0    100    0 ?
*>i [1][213:1][213.6.6.6]/12
      213.6.6.6                      0    100    0 ?
*>i [1][213:1][213.7.7.7]/12
      213.7.7.7                      0    100    0 ?
*>i [1][213:1][213.8.8.8]/12
      213.8.8.8                      0    100    0 ?
*>i [1][213:1][213.12.12.12]/12
      213.12.12.12                  100    0 i
*>i [3][213:1][10.2.7.2][232.0.0.5][213.7.7.7]/22
      213.7.7.7                      0    100    0 ?
```

```
R1#show bgp ipv6 mvpn vrf MC | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 213:1 (default for vrf MC)
*> [1][213:1][213.1.1.1]/12
      ::                          32768 ?
*>i [1][213:1][213.5.5.5]/12
      213.5.5.5                      0    100    0 ?
*>i [1][213:1][213.12.12.12]/12
      213.12.12.12                  100    0 i
```

Looking at the I-PMSI and S-PMSI for CSR7 side by side, we see one uses type 5 and one uses type 3 provider tunnels (green). Type-5 is PIM-Bidir and Type-3 is PIM-SSM, which is correct given our multicast group allocations defined for MDT default/data within this MVPN. If we were using ASM, we would have seen Type-4 also. In this case, both trees are rooted at CSR7 (yellow). The only difference is the destination group (cyan) encoded in the tunnel parameters. This is essentially the same information that IPv4 MDT carries, except it is more verbose and capable.

```
R1#show bgp ipv4 mvpn vrf MC route-type 1 213.7.7.7
BGP routing table entry for [1][213:1][213.7.7.7]/12, version 376
Paths: (1 available, best #1, table MVPNv4-BGP-Table, not advertised to EBGp
peer)
  Not advertised to any peer
```

```

Refresh Epoch 1
Local, imported path from [1][213:7][213.7.7.7]/12 (global)
  213.7.7.7 (metric 20) from 213.12.12.12 (213.12.12.12)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    Community: no-export
    Extended Community: RT:213:7
    Originator: 213.7.7.7, Cluster list: 213.12.12.12
    PMSI Attribute: Flags: 0x0, Tunnel type: 5, length 8, label: exp-null,
tunnel parameters: D507 0707 EEEF 0001
    rx pathid: 0, tx pathid: 0x0

```

```

R1#show bgp ipv4 mvpn vrf MC route-type 3 10.2.7.2 232.0.0.5 213.7.7.7
BGP routing table entry for [3][213:1][10.2.7.2][232.0.0.5][213.7.7.7]/22,
version 378
Paths: (1 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGp
peer)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from [3][213:7][10.2.7.2][232.0.0.5][213.7.7.7]/22
(global)
    213.7.7.7 (metric 20) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Community: no-export
      Extended Community: RT:213:7
      Originator: 213.7.7.7, Cluster list: 213.12.12.12
      PMSI Attribute: Flags: 0x0, Tunnel type: 3, length 8, label: exp-null,
tunnel parameters: D507 0707 E804 0700
      rx pathid: 0, tx pathid: 0x0

```

Because the Type-3 S-PMSI was created once CSR7 received a C(S,G) join for a flow candidate for MDT optimization, this implies the entire signaling path is already set up. Assuming CSR2 is the source with CSR3 and CSR4 as receivers, we can verify the C-MRIB entries on the three relevant PEs. The “verbose” option on the egress PE shows us the GRE MDT to which this is bound; the S-PMSI route told us the P-tunnel was PIM (GRE) and the group was 232.4.7.0. We also see a little ‘y’ which means this is sending to a data MDT group. The ingress PEs, CSR6 and CSR7, also show the MDT to which they are bound (using the Type-3 S-PMSI from CSR7). The big ‘Y’ means they are joined to (receiving) an MDT data group.

```

R7#show ip mroute vrf MC 232.0.0.5 10.2.7.2 verbose | begin \(
(10.2.7.2, 232.0.0.5), 00:05:18/00:03:06, flags: sTyp
  Incoming interface: GigabitEthernet2.527, RPF nbr 0.0.0.0
  Outgoing interface list:
    Tunnel2, GRE MDT: 232.4.7.0 (data), Forward/Sparse, 00:05:19/00:03:06, p

R6#show ip mroute vrf MC 232.0.0.5 10.2.7.2 | begin \(
(10.2.7.2, 232.0.0.5), 00:00:26/00:02:33, flags: sTIY
  Incoming interface: Tunnel3, RPF nbr 213.7.7.7,
MDT: [213.7.7.7,232.4.7.0]/never
  Outgoing interface list:

```

```
GigabitEthernet2.546, Forward/Sparse, 00:00:26/00:02:33
```

```
R8#show ip mroute vrf MC 232.0.0.5 10.2.7.2 | begin \  
(10.2.7.2, 232.0.0.5), 00:36:19/00:02:48, flags: sTIY  
Incoming interface: Tunnel3, RPF nbr 213.7.7.7,  
MDT:[213.7.7.7,232.4.7.0]/never  
Outgoing interface list:  
GigabitEthernet2.538, Forward/Sparse, 00:36:18/00:02:48
```

We can verify the data MDT is in play by checking the routers MDT caches. This information is derived from BGP and helps device the binding between BGP AD routes (PMSI) and C-MRIB entries. All of them know about the data MDT. CSR7 originated the message so the next-hop (last column) is blank.

```
R7#show ip pim mdt bgp all | section 232  
MDT group 232.4.7.0  
213:7:213.7.7.7                213.7.7.7                0.0.0.0  
  
R6#show ip pim mdt bgp all | section 232  
MDT group 232.4.7.0  
213:6:213.7.7.7                213.12.12.12            213.7.7.7  
  
R8#show ip pim mdt bgp all | section 232  
MDT group 232.4.7.0  
213:8:213.7.7.7                213.12.12.12            213.7.7.7
```

We can also quickly verify the P(S,G) creation of the MDT data group. We also must ensure XRv3 and XRv4, core routers, have this P(S,G) state. CSR7 is performing replication to both XRv3 and XRv4 and the little 'z' means this is a multicast tunnel bound to an MDT data group in the MVPN. The core routers show that they are sending the traffic only two their downstream clients and not all PEs in the MVPN, which is the whole point of data MDTs. The core routers have no special MVPN-style flags for the P(S,G) state because they are not aware of anything special. Last, the egress PEs accept this traffic from the core and forward it to their MVPN customers.

```
R7#show ip mroute 232.4.7.0 213.7.7.7 | begin \  
(213.7.7.7, 232.4.7.0), 00:12:19/00:03:14, flags: sTz  
Incoming interface: Loopback0, RPF nbr 0.0.0.0  
Outgoing interface list:  
GigabitEthernet2.573, Forward/Sparse, 00:03:18/00:03:13  
GigabitEthernet2.574, Forward/Sparse, 00:12:19/00:03:14
```

```
RP/0/0/CPU0:XRv3#show pim topology 232.4.7.0 213.7.7.7 | begin 213  
(213.7.7.7,232.4.7.0)SPT SSM Up: 00:04:47  
JP: Join(00:00:04) RPF: GigabitEthernet0/0/0/0.573,213.7.13.7 Flags:  
GigabitEthernet0/0/0/0.563 00:04:47 fwd Join(00:02:40)
```

```
RP/0/0/CPU0:XRv4#show pim topology 232.4.7.0 213.7.7.7 | begin 213  
(213.7.7.7,232.4.7.0)SPT SSM Up: 00:14:02  
JP: Join(now) RPF: GigabitEthernet0/0/0/0.574,213.7.14.7 Flags:
```

```
GigabitEthernet0/0/0/0.584 00:14:02 fwd Join(00:03:13)
```

```
R6#show ip mroute 232.4.7.0 213.7.7.7 | begin \  
(213.7.7.7, 232.4.7.0), 00:05:13/stopped, flags: sTIZ  
Incoming interface: GigabitEthernet2.563, RPF nbr 213.6.13.13  
Outgoing interface list:  
MVRF MC, Forward/Sparse, 00:05:13/00:00:46
```

```
R8#show ip mroute 232.4.7.0 213.7.7.7 | begin \  
(213.7.7.7, 232.4.7.0), 00:14:16/stopped, flags: sTIZ  
Incoming interface: GigabitEthernet2.584, RPF nbr 213.8.14.14  
Outgoing interface list:  
MVRF MC, Forward/Sparse, 00:14:16/00:00:43
```

We begin sending traffic from CSR2 and check the packet counters on all routers. CSR7 is sending them to the PMSI while CSR6 and CSR8 are receiving them from the PMSI. This occurs within the MVPN.

```
R7#show ip mroute vrf MC 232.0.0.5 10.2.7.2 count | begin ^Group  
Group: 232.0.0.5, Source count: 1, Packets forwarded: 318, Packets received:  
318  
Source: 10.2.7.2/32, Forwarding: 318/1/118/0, Other: 318/0/0
```

```
R6#show ip mroute vrf MC 232.0.0.5 10.2.7.2 count | begin ^Group  
Group: 232.0.0.5, Source count: 1, Packets forwarded: 326, Packets received:  
326  
Source: 10.2.7.2/32, Forwarding: 326/1/142/1, Other: 326/0/0
```

```
R8#show ip mroute vrf MC 232.0.0.5 10.2.7.2 count | begin ^Group  
Group: 232.0.0.5, Source count: 1, Packets forwarded: 324, Packets received:  
324  
Source: 10.2.7.2/32, Forwarding: 324/1/142/1, Other: 324/0/0
```

Next, we check counters on the ingress and egress PEs to ensure they are seeing the MDT-encapsulated packets along the PMSI.

```
R7#show ip mroute 232.4.7.0 213.7.7.7 count | begin ^Group  
Group: 232.4.7.0, Source count: 1, Packets forwarded: 179, Packets received:  
179  
Source: 213.7.7.7/32, Forwarding: 179/1/124/0, Other: 179/0/0
```

```
R6#show ip mroute 232.4.7.0 213.7.7.7 count | begin ^Group  
Group: 232.4.7.0, Source count: 1, Packets forwarded: 186, Packets received:  
186  
Source: 213.7.7.7/32, Forwarding: 186/1/142/1, Other: 186/0/0
```

```
R8#show ip mroute 232.4.7.0 213.7.7.7 count | begin ^Group  
Group: 232.4.7.0, Source count: 1, Packets forwarded: 184, Packets received:  
184
```


Source: 213.7.7.7/32, Forwarding: 184/1/142/1, Other: 184/0/0

Although not necessary, we will check the core routers for good measure. Both XRv3 and XRv4 show transit traffic within the data MDT, although they are totally unaware that MVPN is in play.

```
RP/0/0/CPU0:XRv3#show mfib route 232.4.7.0 213.7.7.7 | begin 213
(213.7.7.7,232.4.7.0),   Flags:
Up: 00:21:32
Last Used: 00:00:00
SW Forwarding Counts: 407/407/50468
SW Replication Counts: 407/407/50468
SW Failure Counts: 0/0/0/0/0
GigabitEthernet0/0/0/0.563 Flags:  NS EG, Up:00:21:32
GigabitEthernet0/0/0/0.573 Flags:  A, Up:00:21:32
```

```
RP/0/0/CPU0:XRv4#show mfib route 232.4.7.0 213.7.7.7 | begin 213
(213.7.7.7,232.4.7.0),   Flags:
Up: 00:30:52
Last Used: 00:00:00
SW Forwarding Counts: 426/426/52824
SW Replication Counts: 426/426/52824
SW Failure Counts: 0/0/0/0/0
GigabitEthernet0/0/0/0.574 Flags:  A, Up:00:30:52
GigabitEthernet0/0/0/0.584 Flags:  NS EG, Up:00:30:52
```

As a final check, we can use EPC to measure packets coming in from the core and going out towards the customer. The two packets are shown below in that sequence. The difference in length is exactly 24 bytes (142-118) and the IP addresses are shown plainly in the output. The outer IP addressing is highlighted in yellow and the inner IP addressing is green.

```
R8#show monitor capture CAP buffer detailed
 0  142      0.000000  213.7.7.7      -> 232.4.7.0      GRE
0000:  01005E04 07000050 56A9DE77 8100CE00  ..^....PV..w....
0010:  08004500 007C05E4 0000FE2F EB5BD507  ..E..|...../.[..
0020:  0707E804 07000000 08004500 00642049  .....E..d I
0030:  0000FE01 A3460A02 0702E800 00050800  .....F.....

 1  118      0.000000  10.2.7.2      -> 232.0.0.5      ICMP
0000:  01005E00 00050050 56A9FB1C 8100DD2  ..^....PV.....
0010:  08004500 00642049 0000FD01 A4460A02  ..E..d I.....F..
0020:  0702E800 00050800 06DC000D 00080000  .....
0030:  00002431 5328ABCD ABCDABCD ABCDABCD  ..$1S(.....
```

Next, we will test IPv6 SSM. ASM introduces much more complex signaling in the MVPN customer network and is tested many times in many profiles. IPv6 SSM is interesting because the C(S,G) we use, which is (2001:10:4:6::4, FF33::1), is not candidate for MDT optimization per our IPv6 ACL. This means that it will transit the default MDT, which uses PIM-Bidir in the underlay. All PEs will receive the traffic in

their underlay (core) networks and many will drop it, which can be wasteful both on bandwidth and router resources. First, we verify the C-PIM signaling is working by checking the three relevant PEs. CSR6 should have received the C(S,G) join from CSR7 and CSR8, then added its MDT interface to the OIL.

```
R6#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 | begin \  
(2001:10:4:6::4, FF33::1), 00:37:09/00:03:19, flags: sT  
  Incoming interface: GigabitEthernet2.546  
  RPF nbr: 2001:10:4:6::4  
  Immediate Outgoing interface list:  
    Tunnel3, Forward, 00:37:09/00:03:19
```

```
R7#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 | begin \  
(2001:10:4:6::4, FF33::1), 01:06:42/never, flags: sTI  
  Incoming interface: Tunnel2  
  RPF nbr: ::FFFF:213.6.6.6  
  Immediate Outgoing interface list:  
    GigabitEthernet2.527, Forward, 01:06:42/never
```

```
R8#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 | begin \  
(2001:10:4:6::4, FF33::1), 01:06:41/never, flags: sTI  
  Incoming interface: Tunnel3  
  RPF nbr: ::FFFF:213.6.6.6  
  Immediate Outgoing interface list:  
    GigabitEthernet2.538, Forward, 01:06:41/never
```

We can verify that the default MDT is in use a few ways. First, none of the routers claim to be doing any kind of data MDT. CSR6 does not send and CSR7/CSR8 do not receive and data MDT information. No one has entry idea about any 232 groups; in our case, we can correlate this with no data MDTs.

```
R6#show ipv6 pim mdt bgp all | section 232  
[no output]
```

```
R7#show ipv6 pim mdt bgp all | section 232  
[no output]
```

```
R8#show ipv6 pim mdt bgp all | section 232  
[no output]
```

We check BGP IPv6 MVPN for a matching S-PMSI route for this specific C(S,G) on the router who would have originated it, which is CSR6. There is no such network. However, we do have an I-PMSI match using CSR6 as the root (green) using a type-5 Bidir tunnel (yellow) to group 238.255.0.1 (cyan).

```
R6#show bgp ipv6 mvpn vrf MC route-type 3 2001:10:4:6::4 FF33::1 213.6.6.6  
% Network not in table
```

```
R6#show bgp ipv6 mvpn vrf MC route-type 1 213.6.6.6  
BGP routing table entry for [1][213:6][213.6.6.6]/12, version 32
```

```

Paths: (1 available, best #1, table MVPNV6-BGP-Table, not advertised to EBGp
peer)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  Local
    :: from 0.0.0.0 (213.6.6.6)
    Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
    Community: no-export
    Extended Community: RT:213:6
    PMSI Attribute: Flags: 0x0, Tunnel type: 5, length 8, label: exp-null,
tunnel parameters: D506 0606 EEEF 0001
    rx pathid: 0, tx pathid: 0x0

```

Because the default MDT is working (if it were not, none of the C-MRIB state would be exchanged), we need not validate it in the core again. It is a basic PIM-Bidir configuration with XRv4 as the root and the core has no knowledge about the MDT traffic at all. At this point, we begin sending traffic from CSR4. We check packet counters on the ingress/egress PEs to ensure it is working.

```

R6#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count | include HW
HW Forwarding: 13/0/118/0, Other: 0/0/0

```

```

R7#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count | include HW
HW Forwarding: 25/1/142/1, Other: 0/0/0

```

```

R8#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count | include HW
HW Forwarding: 31/1/142/1, Other: 0/0/0

```

A quick EPC effort on CSR8 from the core and to the customer shows the encapsulations plainly. In summary, this profile is almost identical to profile 0 except replaces IPv4 MDT with IPv4/v6 MVPN using BGP AD. This is the “new way” of configuring PIM/GRE-based MVPN.

```

R8#show monitor capture CAP buffer detailed
11 142 1.998993 213.6.6.6 -> 238.255.0.1 GRE
0000: 01005E7F 00010050 56A9DE77 8100CE00 ..^....PV..w....
0010: 08004500 007C2143 0000FD2F D202D506 ..E..|!C.../....
0020: 0606EEFF 00010000 86DD6000 0000003C .....`....<
0030: 3A3F2001 00100004 00060000 00000000 :? .....

12 118 1.998993 2001:*:0004 -> FF33:*:0001 IPv6-ICMP
0000: 33330000 00010050 56A9FB1C 8100DD2 33.....PV.....
0010: 86DD6000 0000003C 3A3F2001 00100004 ..`....<:?.? ....
0020: 00060000 00000000 0004FF33 00000000 .....3....
0030: 00000000 00000000 00018000 9A970661 .....a

```

Additional Reading – Reference configurations “mvpn-3”

28.4 Profile 6: VRF MLDP – In-band Signaling

For this test, begin with the MVPN Profile 1 configuration where all of the MVPN BGP is set up already, plus the MP2MP/P2MP trees are built. We will be disabling the MDT model and transporting the C-multicast state inside mLDP. The XR configuration is more logical than XE for this profile; in XE, none of the configuration goes under the VRF at all. In XR, the configuration still goes under the VRF and the core-tree is updated to reflect the usage of in-band signaling for mLDP. The configuration in XE uses global PIM commands and references the VRF in question, which is difficult to follow.

```
! Basic XE configuration to enable mLDP in-band signaling
vrf definition MC
  address-family ipv4
    no mdt default mpls mldp 213.14.14.14
    no mdt preference mldp
  address-family ipv6
    no mdt default mpls mldp 213.14.14.14
    no mdt preference mldp

ip multicast vrf MC mpls mldp
ip pim vrf MC mpls source lo0
ipv6 multicast vrf MC mpls mldp
ipv6 multicast vrf MC mpls source loopback 0

! Basic XR configuration to enable mLDP in-band signaling (XRv2)
multicast-routing
  vrf MC
    address-family ipv4
      no mdt default mldp ipv4 213.14.14.14
      mdt mldp in-band-signaling ipv4
    address-family ipv6
      no mdt default mldp ipv4 213.14.14.14
      mdt mldp in-band-signaling ipv4

route-policy RPL_MLDP
  set core-tree mldp-inband
end-policy
```

There is no BGP AD, BGP c-mcast signaling, or PIM c-mcast signaling used in this model. A quick check on CSR6 and XRv2 confirms this. XRv2 shows some self-neighbors that I assume are an internal function of the mLDP in-band signaling. Suffice it to say that there is no more “overlay” across the MVPN.

```
R6#show bgp ipv4 mvpn all
[no output]
```

```
R6#show ip pim vrf MC neighbor | begin ^Neighbor
Neighbor          Interface          Uptime/Expires    Ver    DR
Address                                     Prio/Mode
[no additional output]
```

```
RP/0/0/CPU0:XRv2#show bgp ipv4 mvpn
[no output]
```

```
RP/0/0/CPU0:XRv2#srf MC neighbor | begin Neighbor
Neighbor Address      Interface      Uptime        Expires      DR pri  Flags
10.11.12.11           GigabitEthernet0/0/0/0.512 3d00h        00:01:19 1      B P
10.11.12.12*          GigabitEthernet0/0/0/0.512 3d17h        00:01:32 1 (DR) B P E
213.12.12.12*         GImdtMC       01:13:36     now         1 (DR) E
213.12.12.12*         ImdtMC        01:13:36     now         1 (DR) E
```

Many other MVPN profiles use “out of band” signaling via some kind of c-mcast overlay (PIM or BGP). In this model, the C(S,G) information is directly encoded inside the opaque value. **Note that only SSM is supported for this profile, as there currently is no mechanism to dynamically discover sources.** A quick look at CSR6 shows an IGMP join received from the customer but the C(*,G) goes nowhere, lacking a C-RP. With an overlay in place, CSR6 would have learned this via PIM or BGP c-mcast signaling.

```
R6#show ip mroute vrf MC 225.0.0.1 | begin \
(*, 225.0.0.1), 01:11:02/00:02:01, RP 0.0.0.0, flags: SJC
Incoming interface: Null, RPF nbr 0.0.0.0
Outgoing interface list:
GigabitEthernet2.546, Forward/Sparse, 01:11:02/00:02:01
```

```
R6#show ip pim vrf MC rp mapping
PIM Group-to-RP Mappings
[no output]
```

However, CSR6 does have a customer requesting traffic for (10.2.7.2, 232.0.0.5), which is a valid SSM group. We can check the C-MRIB to verify the entry was installed properly. A new LSPvif was spawned which is used for RPF. Because BGP AD did not advertise any PMSI information and there are no traditional MDTs, we don’t fully see how the traffic is actually being transported yet (the P-tunnel is a mystery at this point).

```
R6#show ip mroute vrf MC 232.0.0.5 | begin \
(10.2.7.2, 232.0.0.5), 01:13:19/00:02:46, flags: sTI
Incoming interface: Lspvif2, RPF nbr 213.7.7.7
Outgoing interface list:
GigabitEthernet2.546, Forward/Sparse, 01:13:19/00:02:46
```

Looking at the current P2MP trees that have been built, we see one rooted in CSR6 and one rooted in CSR7. As discussed earlier, the entire C(S,G) information is carried inside the opaque value, so we can see the IPv4 SSM and IPv6 SSM groups mapped here. The opaque type is VPNv4/v6 since that is the mechanism by which the routes were learned. In this case, we know the source is behind CSR7, so we look at the details for that tree. We can see the mapping to LSPvif2 and also the incoming downstream (local) label of 6008. The length of the opaque makes this less desirable to decode in hex like we did for

other profiles (the global-ID ones, for example), but it's still possible. The C-S, C-G, and RD are shown in cyan, green, and yellow respectively. The length is 16 (0x10) bytes and the type is 0xFA.

```
R6#show mpls mldp database summary
```

LSM ID	Type	Root	Decoded Opaque Value	Client
1	P2MP	213.7.7.7	[vpn4 10.2.7.2 232.0.0.5 213:7]	
2	P2MP	213.6.6.6	[vpn6 2001:10:4:6::4 FF33::1 1]	

```
R6#show mpls mldp database opaque_type vpn4 10.2.7.2 232.0.0.5 213:7
```

```
LSM ID : 1    Type: P2MP    Uptime : 01:15:43
  FEC Root      : 213.7.7.7
  Opaque decoded : [vpn4 10.2.7.2 232.0.0.5 213:7]
  Opaque length  : 16 bytes
  Opaque value   : FA 0010 0A020702E8000005000000D500000007
  Upstream client(s) :
    213.13.13.13:0 [Active]
      Expires      : Never      Path Set ID   : 1
      Out Label (U) : None      Interface    : GigabitEthernet2.563*
      Local Label (D) : 6008    Next Hop     : 213.6.13.13
  Replication client(s):
    MRIBv4(1) (VRF MC)
      Uptime       : 01:15:43    Path Set ID   : None
      Interface    : Lspvif2
```

We also expect to see similar outputs on CSR8. Here, I show the C-MRIB and mLDP database for brevity. CSR8 allocates the downstream label value 8010 to receive traffic on this P2MP tree.

```
R8#show ip mroute vrf MC 232.0.0.5 | begin \((
```

```
(10.2.7.2, 232.0.0.5), 00:08:29/00:02:36, flags: sTI
  Incoming interface: Lspvif2, RPF nbr 213.7.7.7
  Outgoing interface list:
    GigabitEthernet2.538, Forward/Sparse, 00:08:29/00:02:36
```

```
R8#show mpls mldp database opaque_type vpn4 10.2.7.2 232.0.0.5 213:7
```

```
LSM ID : 2    Type: P2MP    Uptime : 00:08:08
  FEC Root      : 213.7.7.7
  Opaque decoded : [vpn4 10.2.7.2 232.0.0.5 213:7]
  Opaque length  : 16 bytes
  Opaque value   : FA 0010 0A020702E8000005000000D500000007
  Upstream client(s) :
    213.14.14.14:0 [Active]
      Expires      : Never      Path Set ID   : 2
      Out Label (U) : None      Interface    : GigabitEthernet2.584*
      Local Label (D) : 8010    Next Hop     : 213.8.14.14
  Replication client(s):
    MRIBv4(1) (VRF MC)
      Uptime       : 00:08:08    Path Set ID   : None
```

Interface : Lspvif2

Clearing the mLDP neighbors on CSR6 and debugging mLDP at the same time reveals the process. CSR6 receives the IGMP join for (10.2.7.2, 232.0.0.5) and adds an entry to its database for the VPNv4 opaque containing the C-S, C-G, and RD. Various other multicast components, such as RPF interfaces, are also updated. The process is identical for IPv6 SSM groups.

```
! CSR6
MLDP-MRIB-RW: RW Tracking started for: 10.2.7.2
MLDP-MRIB-IP(MC): Route watch started for 10.2.7.2 topology: base ipv4
multicast
MLDP-MRIB-IP(MC): Change RPF neighbor from 0.0.0.0 to 213.7.7.7
MLDP-MRIB-IP(MC): (10.2.7.2,232.0.0.5/32) update idb = Lspvif2, (f=2,c=2)
MLDP-MRIB-IP(MC): add accepting interface: Lspvif2 root: 213.7.7.7
MLDP-MRIB-IP(MC): Route watch stopped for 10.2.7.2 topology: base ipv4
multicast
MLDP-MRIB-RW: Sending RW notification message to process: MLDP MRIB Process
MLDP-MRIB-RW: RW Tracking started for: 10.2.7.2
MLDP-MRIB-IP(MC): Route watch started for 10.2.7.2 topology: base ipv4
multicast
MLDP-MRIB-IP(MC): change interface from NULL to Lspvif2
MLDP-MRIB-IP(MC): Route Watch update for 10.2.7.2/32, paths: 2, topology:
base ipv4 multicast
MLDP-MRIB-IP(MC): Found RD 213:7 for source 10.2.7.2 rpf 213.7.7.7
MLDP: LDP root 213.7.7.7 added
mLDP-RW: Sending RW notification message to process: mLDP Process
mLDP-RW: RW Tracking started for: 213.7.7.7
MLDP: Route watch started for 213.7.7.7 topology: base ipv4
MLDP-DB: Added [vpn4 10.2.7.2 232.0.0.5 213:7] DB Entry
MLDP-DB: [vpn4 10.2.7.2 232.0.0.5 213:7] Added P2MP branch for MRIBv4(1)
label
```

When the neighbor with XRv3 comes up, CSR6 sends a label mapping message for this entry using label 6008. Because the tree is unidirectional, there is no upstream label advertised from XRv3 to CSR6.

```
! CSR6
mLDP-RW: Sending RW notification message to process: mLDP Process
mLDP-RW: RW Tracking started for: 213.13.13.13
MLDP-NBR: Route watch started for 213.13.13.13 topology: base ipv4
MLDP-NBR: 213.13.13.13:0 old paths: 0 new paths: 1
MLDP-NBR: 213.13.13.13:0 ask LDP for adjacencies
MLDP-HA Ignoring peer path sync request: No sync required in current RP
state.
MLDP-NBR: 213.13.13.13:0 LDP update NH: 213.6.13.13, Interface:
GigabitEthernet2.563 Add
MLDP-HA: Ignoring mldp db sync request: No sync required in current RP state.
MLDP-NBR: 213.13.13.13:0 mapped to next_hop: 213.6.13.13
MLDP: Reevaluating peers for nhop: 213.6.13.13
```

```
MLDP-DB: [vpn4 10.2.7.2 232.0.0.5 213:7] Changing peer from none to
213.13.13.13:0
MLDP-DB: [vpn4 10.2.7.2 232.0.0.5 213:7] Add accepting element nbr:
213.13.13.13:0
MLDP-MFI: For accepting element nbr: 213.13.13.13:0, [vpn4 10.2.7.2
232.0.0.5 213:7] MFI allocated label 6008
MLDP: [vpn4 10.2.7.2 232.0.0.5 213:7] label mapping msg sent to
213.13.13.13:0 success
```

XRv3 accepts the binding and the process is complete. The LSM-ID is 4 now, and we can confirm this by checking the database. The label allocation is the same but the LSM-ID changed from the output earlier.

```
! CSR6
MLDP-DB: [vpn4 10.2.7.2 232.0.0.5 213:7] path to peer: 213.13.13.13:0
changed None:0.0.0.0 to GigabitEthernet2.563:213.6.13.13
MLDP-MFI: After the MFI call For accepting element nbr: 213.13.13.13:0, Bind
local label 6008 to PSM ID: 4, returned: 6008
MLDP-DB: [vpn4 10.2.7.2 232.0.0.5 213:7] Activate accepting element nbr:
213.13.13.13:0
```

```
R6#show mpls mldp database id 4
LSM ID : 4    Type: P2MP    Uptime : 00:06:14
FEC Root      : 213.7.7.7
Opaque decoded : [vpn4 10.2.7.2 232.0.0.5 213:7]
Opaque length  : 16 bytes
Opaque value   : FA 0010 0A020702E8000005000000D500000007
Upstream client(s) :
  213.13.13.13:0    [Active]
    Expires        : Never          Path Set ID   : 4
    Out Label (U)  : None           Interface    : GigabitEthernet2.563*
    Local Label (D): 6008           Next Hop     : 213.6.13.13
Replication client(s):
  MRIBv4(1) (VRF MC)
    Uptime        : 00:06:14        Path Set ID   : None
    Interface     : Lspvif2
```

Next, we can check CSR7 to see if it has the information about this tree. mLDP should have carried the C(S,G) information in-band upstream towards the root in the label mapping messages. A quick look at the database confirms this. We see that both XRv3 and XRv4 are downstream in the path to CSR6 and CSR8 respectively. It also has the appropriate C(S,G) entry; using the “verbose” keyword we can see it is using LSM-ID 6, which is correct.

```
R7#show mpls mldp database summary | include 232.0.0.5
6          P2MP      213.7.7.7          [vpn4 10.2.7.2 232.0.0.5 213:2

R7#show mpls mldp database id 6
LSM ID : 6    Type: P2MP    Uptime : 00:10:03
```



```

FEC Root          : 213.7.7.7 (we are the root)
Opaque decoded    : [vpn4 10.2.7.2 232.0.0.5 213:7]
Opaque length     : 16 bytes
Opaque value      : FA 0010 0A020702E8000005000000D500000007
Upstream client(s) :
  None
    Expires       : N/A          Path Set ID   : 6
Replication client(s):
  213.14.14.14:0
    Uptime        : 00:10:03     Path Set ID   : None
    Out label (D) : 94002        Interface    : GigabitEthernet2.574*
    Local label (U): None        Next Hop     : 213.7.14.14
  213.13.13.13:0
    Uptime        : 00:09:59     Path Set ID   : None
    Out label (D) : 93000        Interface    : GigabitEthernet2.573*
    Local label (U): None        Next Hop     : 213.7.13.13

```

```

R7#show ip mroute vrf MC 232.0.0.5 verbose | begin \
(10.2.7.2, 232.0.0.5), 01:44:34/stopped, flags: sTI
  Incoming interface: GigabitEthernet2.527, RPF nbr 0.0.0.0
  Outgoing interface list:
    Lspvif0, LSM ID: 6, Forward/Sparse, 00:19:32/00:01:27

```

Checking the LFIBs on XRv3 and XRv4, we see normal MPLS forwarding without any replication. This is because there aren't clients on other PEs, such as CSR1, that are interested in this flow.

```

RP/0/0/CPU0:XRv3#show mpls forwarding labels 93000
Local   Outgoing   Prefix           Outgoing         Next Hop         Bytes
Label   Label      or ID           Interface        Interface        Switched
-----
93000   6008           MLDP: 0x0002b   Gi0/0/0/0.563   213.6.13.6       0

RP/0/0/CPU0:XRv4#show mpls forwarding labels 94002
Local   Outgoing   Prefix           Outgoing         Next Hop         Bytes
Label   Label      or ID           Interface        Interface        Switched
-----
94002   8010           MLDP: 0x0002e   Gi0/0/0/0.584   213.8.14.8       0

```

When CSR2 begins sending traffic, there is no more signaling that occurs. The tree has been fully signaled (as is common when using SSM in most profiles) so routers must only forward the LSM.

```

R6#show ip mroute vrf MC 232.0.0.5 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 103, Packets received:
103
  Source: 10.2.7.2/32, Forwarding: 103/0/122/0, Other: 103/0/0

```

```

R7#show ip mroute vrf MC 232.0.0.5 count | begin ^Group

```

Group: 232.0.0.5, Source count: 1, Packets forwarded: 104, Packets received: 104

Source: 10.2.7.2/32, Forwarding: 104/0/118/0, Other: 104/0/0

R8#show ip mroute vrf MC 232.0.0.5 count | begin ^Group

Group: 232.0.0.5, Source count: 1, Packets forwarded: 106, Packets received: 106

Source: 10.2.7.2/32, Forwarding: 106/0/122/0, Other: 106/0/0

A quick OAM check is also handy to ensure there are no “rogue” endpoints receiving this traffic.

R7#ping mpls mldp p2mp 213.7.7.7 vpnv4 213:7 10.2.7.2 232.0.0.5

p2mp Root node addr 213.7.7.7

Opaque type VPNv4, source 10.2.7.2, group 232.0.0.5

Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
[snip]

Type escape sequence to abort.

Request #1

! reply addr 213.6.13.6

! reply addr 213.8.14.8

Round-trip min/avg/max = 111/155/200 ms

Next we will test IPv6 SSM. We expect the behavior to be identical to IPv4 SSM. We will use different method of verifying the flows. First, we will ensure all of the PEs have the proper C-MRIB information. Ingress PE CSR6 sends traffic to the PMSI while the egress PEs (CSR7 and CSR8) accept information arriving from the PMSI.

R6#show ipv6 mroute vrf MC FF33::1 | begin \((

(2001:10:4:6::4, FF33::1), 00:28:46/never, flags: sT

Incoming interface: GigabitEthernet2.546

RPF nbr: 2001:10:4:6::4

Immediate Outgoing interface list:

Lspvif3, Forward, 00:28:46/never

R7#show ipv6 mroute vrf MC FF33::1 | begin \((

(2001:10:4:6::4, FF33::1), 01:55:08/never, flags: sTI

Incoming interface: Lspvif1

RPF nbr: ::FFFF:213.6.6.6

Immediate Outgoing interface list:

GigabitEthernet2.527, Forward, 01:55:08/never

R8#show ipv6 mroute vrf MC FF33::1 | begin \((

(2001:10:4:6::4, FF33::1), 00:29:46/never, flags: sTI

Incoming interface: Lspvif3

We will send traffic from CSR4. We already know the downstream label is 93015. XRv3 will replicate this to CSR7, and endpoint, and also towards CSR8 via XRv4.

```
R4#ping FF33::1 rep 10000
Output Interface: GigabitEthernet2.546
```

```
RP/0/0/CPU0:XRv3#show mpls forwarding labels 93015
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
93015	7010	MLDP: 0x0002d	Gi0/0/0/0.573	213.7.13.7	0
	94003	MLDP: 0x0002d	Gi0/0/0/0.534	213.13.14.14	0

Upon receipt, CSR7 delivers the traffic to the customer in accordance with the VPNv6 opaque value carried inside the mLDP mapping messages. XRv4 forwards it down to CSR8. The byte counter increases on the XE boxes, which is another verification method. These LFIB counters do not appear to work for MVPN traffic on XRv.

```
R7#show mpls forwarding-table labels 7010
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
7010	[T] No Label	[vpn6 2001:10:4:6::4 FF33::1 213:6]	11834	[V] \	aggregate/MC

```
RP/0/0/CPU0:XRv4#show mpls forwarding labels 94003
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94003	8011	MLDP: 0x0002f	Gi0/0/0/0.584	213.8.14.8	0

Finally, Like CSR7, CSR8 forwards it to its customer based on the VPNv6 opaque information. We also see packet matches, like on CSR7.

```
R8#show mpls forwarding-table labels 8011
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
8011	[T] No Label	[vpn6 2001:10:4:6::4 FF33::1 213:6]	15738	[V] \	aggregate/MC

We will quickly check the IPv6 C-multicast counters on all of the PEs to ensure there are no issues there. I also use EPC to validate the traffic entering on CSR7 to ensure it is using the correct label and has the correct C(S,G) information. The label is 0x1B62 highlighted in yellow (7010) and the IPv6 information is correct also. The source address is highlighted in cyan and the group is in green.

```
R6#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count | include HW
HW Forwarding: 336/0/118/0, Other: 0/0/0
```

```
R7#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count | include HW
HW Forwarding: 337/0/122/0, Other: 0/0/0
```

```
R8#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count | include HW
HW Forwarding: 339/0/122/0, Other: 0/0/0
```

```
R7#show monitor capture CAP buffer dump
0
0000: 005056A9 EA770050 56A9DB37 81000DF5 .PV..w.PV..7....
0010: 884701B6 213E6000 0000003C 3A3F2001 .G...!>`....<:? .
0020: 00100004 00060000 00000000 0004FF33 .....3
0030: 00000000 00000000 00000000 00018000 .....
0040: FA971C2A 01111112 13141516 1718191A ...*.....
0050: 1B1C1D1E 1F202122 23242526 2728292A ..... !"#$$%&'()* *
0060: 2B2C2D2E 2F303132 33343536 3738393A +,-./0123456789:
0070: 3B3C3D3E 3F404142 4344 ;<=>?@ABCD
```

Additional Reading – Reference configurations “mvpn-6”

28.5 Profile 7: Global MLDP In-band Signaling

This profile is very similar to Profile 6 except performs mLDP in-band signaling within the global table. BGP VPNv4/v6 and MVPN v4/v6 have been disabled which should obviate the need for carrying an RD within the opaque. Normal IPv6 unicast and IPv6 labeled-unicast (6PE) reachability is provided between the customer sites in this design. PIM is not enabled on the core routers (XRv3 and XRv4). There was significant rework to BGP to support this, but that rework was not technically challenging and is not shown here. Only the significant components of mLDP in-band signaling are shown. The same 4 commands are used to enable the feature for IPv4 and IPv6, except the “VRF” modifier is not used. This is a very simple configuration. XR still uses the same core tree RPL; it is shown again for reference only.

```
! Basic XE configuration to enable mLDP in-band signaling
ip multicast mpls mldp
ip pim mpls source lo0
ipv6 multicast mpls mldp
ipv6 multicast mpls source loopback 0
```

```
! Basic XR configuration to enable mLDP in-band signaling
multicast-routing
address-family ipv4
 mdt mldp in-band-signaling ipv4
address-family ipv6
 mdt mldp in-band-signaling ipv4

route-policy RPL_MLDP
 set core-tree mldp-inband
end-policy
```

Because mLDP in-band signaling carries the C(S,G) information inside of its opaque, only PIM-SSM is supported. There is no signaling mechanism in the overlay to carry information like the BSR mappings or PIM hellos; this is identical to profile 6 seen earlier. A quick check on CSR8 shows no PIM neighbors and no RP information. CSR5 learns the RP only because they are connected, but it is not used for anything significant in this test.

```
R8#show ip pim neighbor | begin ^Neighbor
Neighbor          Interface          Uptime/Expires    Ver    DR
Address                                     Prio/Mode
[no output]
```

```
R8#show ip pim rp mapping
PIM Group-to-RP Mappings
[no output]
```

```
R5#show ip pim rp mapping
PIM Group-to-RP Mappings
Group(s) 224.0.0.0/4
  RP 10.5.10.10 (?), v2
    Info source: 10.5.10.10 (?), via bootstrap, priority 0, holdtime 150
    Uptime: 00:24:25, expires: 00:02:23
```

As a final verification to prove PIM-ASM is broken, CSR10 is the RP and does not have the C(*,G) join for group 225.0.0.1. CSR8 receives the IGMP join for this group but has no RP assigned.

```
R8#show ip mroute 225.0.0.1 | begin \
(*, 225.0.0.1), 00:24:54/00:02:12, RP 0.0.0.0, flags: SJC
  Incoming interface: Null, RPF nbr 0.0.0.0
  Outgoing interface list:
    GigabitEthernet2.538, Forward/Sparse, 00:24:54/00:02:12
```

```
R5#show ip mroute 225.0.0.1
Group 225.0.0.1 not found
```

PIM-SSM, however, appears to be functioning from a PIM standpoint. Both CSR6 and CSR8 have clients joining (10.2.7.2, 232.0.0.5) and have PIM C(S,G) tries that indicate a valid incoming interface (PMSI). The incoming interface will be mapped to the mLDP delivery tree inside the mLDP database (shown later).

```
R6#show ip mroute 232.0.0.5 10.2.7.2 | begin \
(10.2.7.2, 232.0.0.5), 00:28:06/00:02:55, flags: sTI
  Incoming interface: Lspvif6, RPF nbr 213.7.7.7
  Outgoing interface list:
    GigabitEthernet2.546, Forward/Sparse, 00:28:06/00:02:55
```

```
R8#show ip mroute 232.0.0.5 10.2.7.2 | begin \
(10.2.7.2, 232.0.0.5), 00:26:37/00:02:20, flags: sTI
```

```
Incoming interface: Lspvif3, RPF nbr 213.7.7.7
Outgoing interface list:
  GigabitEthernet2.538, Forward/Sparse, 00:26:37/00:02:20
```

Looking at the mLDP trees on CSR6, we see that each one has two P2MP trees. These are unidirectional (downstream only) trees rooted at CSR7. mLDP in-band signaling clearly shows us what the opaque values are, and we see the IPv4 and IPv6 multicast (S,G) embedded inside. This in-band signaling mechanism creates a P2MP tree for every PIM-SSM sender in the network, and all the leaves join it via label-mapping messages. The output is very similar to the VPNv4/v6 opaque messages seen earlier except without an RD.

```
R6#show mpls mldp database summary
LSM ID      Type      Root      Decoded Opaque Value      Client
Cnt.
7           P2MP      213.7.7.7      [ipv4 10.2.7.2 232.0.0.5]      1
8           P2MP      213.6.6.6      [ipv6 2001:10:4:6::4 FF33::1]  1
```

CSR8 shows an IPv6 ASM group only because the RP is embedded. The C(*,G) join actually does make it to the C-RP, which is CSR2. The issue is that there is no way to prune specific sources from the shared tree once the flow starts without an out-of-band signaling protocol. So technically, some components of PIM-ASM could work, but its general operation is compromised due to the lack of an overlay. For now, we will focus on the IPv4 (S,G).

```
R8#show mpls mldp database summary
LSM ID      Type      Root      Decoded Opaque Value      Client
Cnt.
3           P2MP      213.7.7.7      [ipv4 10.2.7.2 232.0.0.5]      1
1           P2MP      213.7.7.7      [ipv6 :: FF7E:240:2001:10:2:7:1]  1
2           P2MP      213.6.6.6      [ipv6 2001:10:4:6::4 FF33::1]  1
```

Looking into more detail on the IPv4 entry for CSR8, we see the low-level opaque type is 0x03, the field is 8 bytes long, and the information is simply the concatenated string of C-S and C-G (yellow and green, respectively). From CSR8's perspective, there is no traffic going outbound (upstream) towards CSR7 since this tree is unidirectional, so no upstream labels are allocated. CSR8 should be receiving traffic with label 8010 for this C(S,G). Notice that the replication client lists Lspvif3, which is the same interface shown above in the C-MRIB output.

```
R8#show mpls mldp database opaque_type ipv4 10.2.7.2 232.0.0.5
LSM ID : 3      Type: P2MP      Uptime : 00:06:47
  FEC Root      : 213.7.7.7
  Opaque decoded : [ipv4 10.2.7.2 232.0.0.5]
  Opaque length  : 8 bytes
  Opaque value   : 03 0008 0A020702E8000005
  Upstream client(s) :
    213.14.14.14:0 [Active]
    Expires       : Never      Path Set ID : 3
```

```

Out Label (U) : None           Interface : GigabitEthernet2.584*
Local Label (D): 8010         Next Hop  : 213.8.14.14
Replication client(s):
MRIBv4(0)
Uptime          : 00:06:47      Path Set ID   : None
Interface       : Lspvif3

```

The same output exists on CSR6 since it joined the same P2MP tree for the same opaque value. The only differences (potentially) will be the incoming label and replication interfaces. Traffic arriving to CSR6 will use label 6007.

```

R6#show mpls mldp database opaque_type ipv4 10.2.7.2 232.0.0.5
LSM ID : 7   Type: P2MP   Uptime : 00:52:36
FEC Root      : 213.7.7.7
Opaque decoded : [ipv4 10.2.7.2 232.0.0.5]
Opaque length  : 8 bytes
Opaque value   : 03 0008 0A020702E8000005
Upstream client(s) :
213.13.13.13:0 [Active]
Expires       : Never          Path Set ID   : 7
Out Label (U) : None          Interface     : GigabitEthernet2.563*
Local Label (D): 6007         Next Hop      : 213.6.13.13
Replication client(s):
MRIBv4(0)
Uptime          : 00:52:36      Path Set ID   : None
Interface       : Lspvif6

```

CSR7 also has a single entry for this tree. CSR7 replicates the traffic towards both XRv3 and XRv4 since there are clients behind each of the core routers. Note the outgoing downstream labels of 93002 and 94002 as we will use these to verify the LFIB on the core routers.

```

R7#show mpls mldp database opaque_type ipv4 10.2.7.2 232.0.0.5
LSM ID : 8   Type: P2MP   Uptime : 00:24:31
FEC Root      : 213.7.7.7 (we are the root)
Opaque decoded : [ipv4 10.2.7.2 232.0.0.5]
Opaque length  : 8 bytes
Opaque value   : 03 0008 0A020702E8000005
Upstream client(s) :
None
Expires       : N/A           Path Set ID   : 8
Replication client(s):
213.13.13.13:0
Uptime          : 00:24:31      Path Set ID   : None
Out label (D)   : 93002         Interface     : GigabitEthernet2.573*
Local label (U) : None          Next Hop      : 213.7.13.13
213.14.14.14:0
Uptime          : 00:08:33      Path Set ID   : None
Out label (D)   : 94002         Interface     : GigabitEthernet2.574*

```


Local label (U): None

Next Hop : 213.7.14.14

A quick OAM check from CSR7 helps us discover the neighbors. Recall that MPLS OAM is only valuable at the root when probing P2MP mLDP trees as they are unidirectional. I demonstrate the “easy” way using built-in commands and the “hard” way using hexadecimal codes. The hex method is required for some verification as the XE parser does not list all supported opaque types (seen later).

```
R7#ping mpls mldp p2mp 213.7.7.7 ipv4 10.2.7.2 232.0.0.5
p2mp Root node addr 213.7.7.7
Opaque type IPv4, source 10.2.7.2, group 232.0.0.5
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
    timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]
Request #1
! reply addr 213.8.14.8
! reply addr 213.6.13.6

Round-trip min/avg/max = 48/116/185 ms
Received 2 replies
```

```
R7#ping mpls mldp p2mp 213.7.7.7 hex 0x03 0A020702E8000005
p2mp Root node addr 213.7.7.7
Opaque type hex value (0x3), num hex digits 8
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
    timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]

Request #1
! reply addr 213.6.13.6
! reply addr 213.8.14.8

Round-trip min/avg/max = 128/132/137 ms
Received 2 replies
```

Because the C-PIM trees and subsequent LSPs are built, sending traffic from CSR2 should trigger no additional signaling. We check C-MRIB packet counters on all three routers to ensure the multicast process is able to send packets to (CSR7) and receive packets from (CSR6 and CSR8) the PMSI.

```
R7#show ip mroute 232.0.0.5 10.2.7.2 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 33, Packets received:
33
    Source: 10.2.7.2/32, Forwarding: 33/1/118/0, Other: 33/0/0

R6#show ip mroute 232.0.0.5 10.2.7.2 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 32, Packets received:
32
```

Source: 10.2.7.2/32, Forwarding: 32/1/122/0, Other: 32/0/0

```
R8#show ip mroute 232.0.0.5 10.2.7.2 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 30, Packets received:
30
Source: 10.2.7.2/32, Forwarding: 30/1/122/0, Other: 30/0/0
```

The LFIBs of the core routers indicate that this traffic is being sent downstream towards the egress PEs. The transport is efficient since these are P2MP trees only; there is no MP2MP tree in use. The outbound downstream labels from CSR7 are 93002 and 94002, which we saw from CSR7's mLDP database. Unfortunately, XRv packet counters do not appear to work for LSM. We can see labels 6007 and 8010 in use which we verified to be the local labels for this LSP on CSR6 and CSR8, respectively.

```
RP/0/0/CPU0:XRv3#show mpls forwarding labels 93002
Local   Outgoing   Prefix           Outgoing   Next Hop        Bytes
Label   Label       or ID            Interface
-----
93002   6007        MLDP: 0x0002f    Gi0/0/0/0.563 213.6.13.6      0

RP/0/0/CPU0:XRv4#show mpls forwarding labels 94002
Local   Outgoing   Prefix           Outgoing   Next Hop        Bytes
Label   Label       or ID            Interface
-----
94002   8010        MLDP: 0x00030    Gi0/0/0/0.584 213.8.14.8      0
```

When CSR6 receives the packet it removes all MPLS encapsulation. The LFIB entry is bound to the mLDP opaque value which defines the (S,G), so when the packet is passed to the PIM process, there is no question as to which C-MRIB entry this flow belongs.

```
R6#show mpls forwarding-table labels 6007
Local      Outgoing   Prefix           Bytes Label   Outgoing   Next Hop
Label      Label       or Tunnel Id     Switched      interface
6007 [T] No Label   [ipv4 10.2.7.2 232.0.0.5] \
                                     35624      aggregate
```

On CSR8, we will verify with EPC. Capturing inbound, we see the LSM arriving with label 8010 (0x1F4A). Going outbound to the customer, we see the raw ICMP multicast packet. The difference in packet size is exactly 4 bytes which accounts for the MPLS shim header containing one label (yellow). The (S,G) information of (10.2.7.2, 232.0.0.5) is highlighted in green.

```
R8#show monitor capture CAP buffer detailed
 0 122 0.000000 00:50:56:A9:DE:77 -> 00:50:56:A9:FB:1C MPLS unicast
0000: 005056A9 FB1C0050 56A9DE77 81000E00 .PV....PV..w....
0010: 884701F4 A1FD4500 0064500C 0000FE01 .G....E..dP.....
0020: 73830A02 0702E800 00050800 93290018 s.....) ..
0030: 018B0000 00002E85 BAF8ABCD ABCDABCD .....
```

```

1 118 0.000000 10.2.7.2 -> 232.0.0.5 ICMP
0000: 01005E00 00050050 56A9FB1C 81000DD2 ..^....PV.....
0010: 08004500 0064500C 0000FC01 75830A02 ..E..dP.....u...
0020: 0702E800 00050800 93290018 018B0000 .....)).....
0030: 00002E85 BAF8ABCD ABCDABCD ABCDABCD .....

```

Next, we test IPv6 SSM. CSR4 is the source and CSR2/CSR3 are the receivers. We saw earlier from our summary view that some of the routers had IPv6 opaque state for certain flows. Revisiting that in detail, we see the egress PEs (CSR7 and CSR8) both received MLDv2 joins from their clients for this group.

```
R7#show ipv6 mld groups FF33::1 detail
```

```
Interface: GigabitEthernet2.527
```

```
Group: FF33::1
```

```
Uptime: 01:19:58
```

```
Router mode: INCLUDE
```

```
Host mode: INCLUDE
```

```
Last reporter: FE80::2
```

```
Group source list:
```

Source Address	Uptime	Expires	Fwd	Flags
2001:10:4:6::4	01:19:58	00:02:25	Yes	Remote 4

```
R8#show ipv6 mld groups FF33::1 detail
```

```
Interface: GigabitEthernet2.538
```

```
Group: FF33::1
```

```
Uptime: 00:28:48
```

```
Router mode: INCLUDE
```

```
Host mode: INCLUDE
```

```
Last reporter: FE80::3
```

```
Group source list:
```

Source Address	Uptime	Expires	Fwd	Flags
2001:10:4:6::4	00:28:48	00:03:07	Yes	Remote 4

These get written to the C-MFIB as (S,G) entries and the router, knowing that it is enabled for mLDP in-band signaling for IPv6, originates a label mapping message for this (S,G) towards the root, which is CSR6. Debugging mLDP on CSR8 reveals this process. First, there are IPv6-specific things that occur with respect to the C(S,G). The join is received and RPF is updated.

```
! CSR8
```

```
MLDP-MRIB-IPv6: (2001:10:4:6::4,FF33::1/128) update (t=1) RPF:
```

```
::FFFF:213.6.6.6
```

```
MLDP-MRIB-IPv6: (2001:10:4:6::4,FF33::1/128) set rpf nbr: ::FFFF:213.6.6.6
```

```
MLDP-MRIB-IPv6: wavl insert success (2001:10:4:6::4, FF33::1)
```

```
MLDP-MRIB-IPv6: RPF neighbor from :: to ::FFFF:213.6.6.6
```

```
MLDP-MRIB-IPv6: (2001:10:4:6::4,FF33::1/128) update idb = Lspvif2, (f=2,c=2)
```

```
MLDP-MRIB-IPv6: add accepting interface: Lspvif2 root: ::FFFF:213.6.6.6
```

```
MLDP-MRIB-IPv6: change interface from NULL to Lspvif2
```

Next, mLDP builds the tree, which is universal for any mLDP tree. The only difference is the opaque value (that mLDP doesn't care about) represents IPv6 versus IPv4. The router allocates label 8009 and sends this in a label-mapping message upstream to XRv4. This tells XRv4 which label to use when sending traffic down the mLDP P2MP delivery tree.

```
! CSR8
MLDP: LDP root 213.6.6.6 added
mLDP-RW: Sending RW notification message to process: mLDP Process
mLDP-RW: RW Tracking started for: 213.6.6.6
MLDP: Route watch started for 213.6.6.6 topology: base ipv4
MLDP-DB: Added [ipv6 2001:10:4:6::4 FF33::1] DB Entry
MLDP-DB: [ipv6 2001:10:4:6::4 FF33::1] Added P2MP branch for MRIBv6(0) label
MLDP-MFI: PSM-ID: 4, Add Path type: REC_IF, Interface: Lspvif2
MLDP: Root 213.6.6.6 old paths: 0 new paths: 1
MLDP-DB: [ipv6 2001:10:4:6::4 FF33::1] Changing peer from none to
213.14.14.14:0
MLDP-DB: [ipv6 2001:10:4:6::4 FF33::1] Add accepting element nbr:
213.14.14.14:0
MLDP-MFI: For accepting element nbr: 213.14.14.14:0, [ipv6 2001:10:4:6::4
FF33::1] MFI allocated label 8009
MLDP: [ipv6 2001:10:4:6::4 FF33::1] label mapping msg sent to 213.14.14.14:0
success
```

Checking the C-MRIB on all three routers, we can see the proper (S,G) information. CSR6 receives multicast traffic from the customer and sends it to the PMSI, while CSR7 and CSR8 do the opposite. Even though this is a P2MP tree, it is technically not an MDT-data group. There are no "MDT" mechanisms in play, so the 'Y' or 'y' flags do not apply.

```
R6#show ipv6 mroute FF33::1 2001:10:4:6::4 | begin \
(2001:10:4:6::4, FF33::1), 00:49:56/never, flags: sT
Incoming interface: GigabitEthernet2.546
RPF nbr: 2001:10:4:6::4
Immediate Outgoing interface list:
Lspvif7, Forward, 00:49:56/never

R7#show ipv6 mroute FF33::1 2001:10:4:6::4 | begin \
(2001:10:4:6::4, FF33::1), 01:26:39/never, flags: sTI
Incoming interface: Lspvif4
RPF nbr: ::FFFF:213.6.6.6
Immediate Outgoing interface list:
GigabitEthernet2.527, Forward, 01:26:39/never

R8#show ipv6 mroute FF33::1 2001:10:4:6::4 | begin \
(2001:10:4:6::4, FF33::1), 00:04:42/never, flags: sTI
Incoming interface: Lspvif2
RPF nbr: ::FFFF:213.6.6.6
```

Immediate Outgoing interface list:

GigabitEthernet2.538, Forward, 00:04:42/never

In terms of labels, CSR7 allocates 7005 and CSR8 allocates 8009 upstream towards the core as they build their trees. This makes XRv3 a branch point since it has two downstream clients: CSR7 and XRv4. Notice that CSR7 has bound its LSM path to Lspvif4 and CSR8 did the same with Lspvif2. This must match what the C-MRIB says. Decoding the IPv6 opaque is also simple. The hex code is now 0x04 (versus 0x03 for IPv4) with the concatenation of the C(S,G) information.

```
R7#show mpls mldp database opaque_type ipv6 2001:10:4:6::4 FF33::1
LSM ID : 7   Type: P2MP   Uptime : 01:22:50
FEC Root      : 213.6.6.6
Opaque decoded : [ipv6 2001:10:4:6::4 FF33::1]
Opaque length  : 32 bytes
Opaque value   : 04 0020
2001001000040006000000000000000004FF3300000000000000000000000001
Upstream client(s) :
  213.13.13.13:0   [Active]
    Expires        : Never          Path Set ID   : 7
    Out Label (U)   : None          Interface     : GigabitEthernet2.573*
    Local Label (D) : 7005          Next Hop      : 213.7.13.13
Replication client(s):
  MRIBv6(0)
    Uptime          : 01:22:50      Path Set ID   : None
    Interface       : Lspvif4
```

```
R8#show mpls mldp database opaque_type ipv6 2001:10:4:6::4 FF33::1
LSM ID : 4   Type: P2MP   Uptime : 00:09:53
FEC Root      : 213.6.6.6
Opaque decoded : [ipv6 2001:10:4:6::4 FF33::1]
Opaque length  : 32 bytes
Opaque value   : 04 0020
2001001000040006000000000000000004FF3300000000000000000000000001
Upstream client(s) :
  213.14.14.14:0   [Active]
    Expires        : Never          Path Set ID   : 4
    Out Label (U)   : None          Interface     : GigabitEthernet2.584*
    Local Label (D) : 8009          Next Hop      : 213.8.14.14
Replication client(s):
  MRIBv6(0)
    Uptime          : 00:09:53      Path Set ID   : None
    Interface       : Lspvif2
```

Tracing the full mLDP signaling path, XRv4 will receive the label mapping from CSR8 specifying the FEC root, so XRv4 sends a label mapping message upstream to XRv3. The mLDP database tells us that XRv4 allocated downstream label 94005 to XRv3 for incoming traffic.

```

RP/0/0/CPU0:XRv4#show mpls mldp database opaque type ipv6 2001:10:4:6::4
FF33::1
mLDP database
LSM-ID: 0x00032   Type: P2MP   Uptime: 00:12:11
  FEC Root       : 213.6.6.6
  Opaque decoded  : [ipv6 2001:10:4:6::4 ff33::1]
  Upstream neighbor(s) :
    213.13.13.13:0 [Active] Uptime: 00:12:11
    Local Label (D) : 94005
  Downstream client(s):
    LDP 213.8.8.8:0   Uptime: 00:12:11
      Next Hop        : 213.8.14.8
      Interface       : GigabitEthernet0/0/0/0.584
      Remote label (D) : 8009

```

At the branch point, XRv3 replicates the traffic to CSR7 using label 7005 and to XRv4 using label 94005. The LFIB confirms the replication also. XRv3 allocates downstream label 93000 to CSR6, which is the root (upstream).

```

RP/0/0/CPU0:XRv3#show mpls mldp database opaque type ipv6 2001:10:4:6::4
FF33::1
mLDP database
LSM-ID: 0x0002E   Type: P2MP   Uptime: 00:59:46
  FEC Root       : 213.6.6.6
  Opaque decoded  : [ipv6 2001:10:4:6::4 ff33::1]
  Upstream neighbor(s) :
    213.6.6.6:0 [Active] Uptime: 00:59:46
    Local Label (D) : 93000
  Downstream client(s):
    LDP 213.7.7.7:0   Uptime: 00:59:46
      Next Hop        : 213.7.13.7
      Interface       : GigabitEthernet0/0/0/0.573
      Remote label (D) : 7005
    LDP 213.14.14.14:0 Uptime: 00:14:23
      Next Hop        : 213.13.14.14
      Interface       : GigabitEthernet0/0/0/0.534
      Remote label (D) : 94005

```

```

RP/0/0/CPU0:XRv3#show mpls forwarding labels 93000

```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
93000	7005	MLDP: 0x0002e	Gi0/0/0/0.573	213.7.13.7	0
	94005	MLDP: 0x0002e	Gi0/0/0/0.534	213.13.14.14	0

CSR6 sees the label mapping from XRv3 and uses this for all downstream traffic in this P2MP tree. No upstream label is allocated by CSR6. CSR6 is the root of the tree, so the signaling path ends.

When CSR4 sends data, no additional signaling should take place. The C-MRIB packet counters on CSR6 shows packet exiting to the PMSI, while CSR7/CSR8 show them arriving from the PMSI.

```
R8#show ipv6 mroute FF33::1 2001:10:4:6::4 count | include HW
HW Forwarding: 38/1/122/0, Other: 0/0/0
```

```
R7#show mpls forwarding-table label 7005
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
7005 [T]   No Label    [ipv6 2001:10:4:6::4 FF33::1]
                                           16104
                                           aggregate
```

```
R8#show monitor capture CAP buffer detailed
  9 122      1.345035  00:50:56:A9:DE:77 -> 00:50:56:A9:FB:1C MPLS unicast
0000:  005056A9 FB1C0050 56A9DE77 81000E00      .PV....PV..w....
0010:  884701F4 913D6000 0000003C 3A3F2001      .G...='....<:?.
```

```

0020: 00100004 00060000 00000000 0004FF33 .....3
0030: 00000000 00000000 00000000 00018000 .....

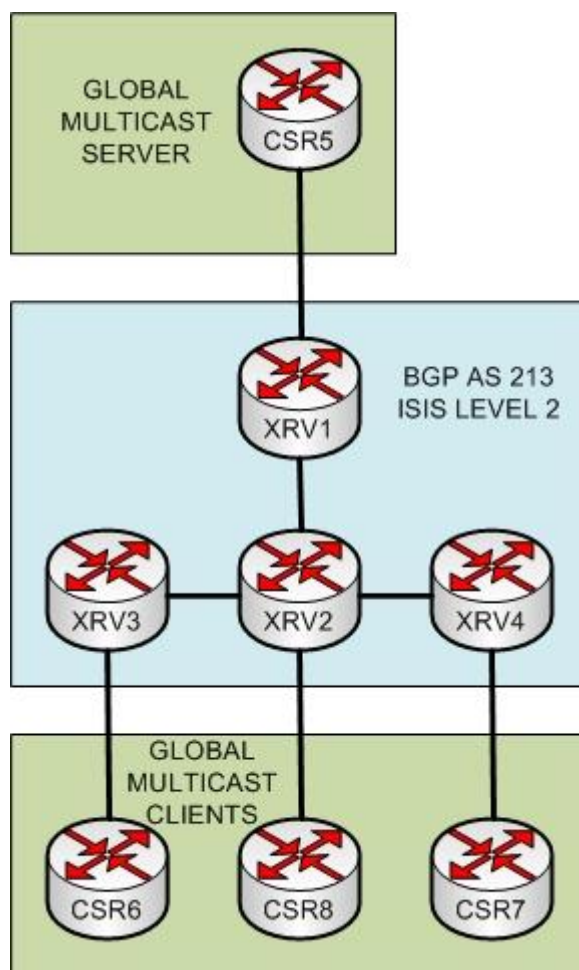
10 118 1.345035 2001:::0004 -> FF33:::0001 IPv6-ICMP
0000: 33330000 00010050 56A9FB1C 8100DD2 33.....PV.....
0010: 86DD6000 0000003C 3A3C2001 00100004 ..`....<:< .....
0020: 00060000 00000000 0004FF33 00000000 .....3....
0030: 00000000 00000000 00018000 71ED1B9F .....q...

```

Additional Reading – Reference configurations “mvpn-7”

28.6 Profile 8: Global Static – P2MP-TE

P2MP RSVP-TE is another technique used to transport LSM across an MPLS network. The benefits of this approach were discussed during the general MVPN comparison, but overall the feature is used in few-to-many scenarios where bandwidth reservations and/or FRR are required. **No other MVPN solutions can provide bandwidth reservations.** FRR is also inherent in this design without any special modifications, provided the sub-LSPs request FRR treatment. The feature is not supported on the CSR (or any XE platform) at all even though RSVP-TE was extended further to support it. The MPLS network consists only of XRV devices with CSRs as CE devices. Even though there are no VRFs anywhere in this test, it is still considered an MVPN profile, much like global mLDP in-band. The diagram is shown below.



The CSRs run RIPv2 with the XrV PEs at the pseudo PE-CE connections. Regular IPv4 BGP transports the IP prefixes while 6PE is used to transport IPv6. Only the connected transit PE-CE links are redistributed into BGP and each CSR has an IPv6 default route pointing towards the provider. The CSRs have joined two SSM (S,G) pairs on their loopbacks: (10.5.5.5, 232.1.1.1) and (FF33::1, 2001:10:5:11::5). The tunnel configuration for the IPv4 tunnel is shown below. A single P2MP tunnel could support both IPv4 and IPv6, but I demonstrate wanting them to be separate, with different tail-ends.

```
! Xrv1
interface tunnel-mte134
  ipv4 unnumbered Loopback0
  destination 213.12.12.12
  path-option 10 dynamic

  destination 213.13.13.13
  path-option 10 dynamic

  destination 213.14.14.14
  path-option 10 dynamic
```

```
signalled-bandwidth 20000
```

The normal TE process begins whether unicast/multicast routing is set up or not. This relies on the TED as any TE tunnel would. The basic MPLS configurations in the core, to include IGP, MPLS-TE, and BGP, are not discussed in detail here. For each sub-LSP, a separate RSVP PATH message is created. Between both tunnels (MTE134 and 136), we have 5 sub-LSPs. Looking at XRv2, it receives those sub-LSPs.

```
RP/0/0/CPU0:XRv2#show rsvp sender
```

Destination Add	P2MPID	Source Add	SPort	Input IF	Rate	Burst	Prot
213.12.12.12	134	213.11.11.11	10002	Gi0/0/0/0.512	20M	1K	Off
213.13.13.13	134	213.11.11.11	10002	Gi0/0/0/0.512	20M	1K	Off
213.14.14.14	134	213.11.11.11	10002	Gi0/0/0/0.512	20M	1K	Off

It appears as if all of them are requesting bandwidth individually. This is technically true, but RSVP-TE was extended to account for this by using the P2MP-ID. Below I show all of the sub-LSP path messages and their requests for bandwidth. Bandwidth is not summed when PATH messages contain requests for bandwidth if the P2MP-ID matches between them. We confirm that XRv2 has only allocated 20 Mbps (not 60) on its interfaces facing downstream.

```
RP/0/0/CPU0:XRv2#show rsvp sender p2mp-id 134 detail | utility egrep
'Destination|Rate'
```

Destination addr: 213.12.12.12. Subgroup: 1/213.11.11.11.
Rate: 20M bits/sec. Burst: 1K bytes. Peak: 20M bits/sec.
Destination addr: 213.13.13.13. Subgroup: 2/213.11.11.11.
Rate: 20M bits/sec. Burst: 1K bytes. Peak: 20M bits/sec.
Destination addr: 213.14.14.14. Subgroup: 3/213.11.11.11.
Rate: 20M bits/sec. Burst: 1K bytes. Peak: 20M bits/sec.

```
RP/0/0/CPU0:XRv2#show rsvp interface
```

*: RDM: Default I/F B/W % : 75% [default] (max resv/bc0), 0% [default] (bc1)

Interface	MaxBW (bps)	MaxFlow (bps)	Allocated (bps)	MaxSub (bps)
Gi0/0/0/0.512	100M	100M	0 (0%)	0
Gi0/0/0/0.523	100M	100M	20M (35%)	0
Gi0/0/0/0.524	100M	100M	20M (35%)	0

We can also ask XRv2 about its label bindings for these LSPs. Because XRv2 is downstream from the root (XRv1), the incoming local label is considered downstream. Notice the label is the same for all sub-LSPs; this implies replication occurs when label 92008 arrives inbound. The first sub-LSP terminates locally on XRv2 so there is no label swap operation. The other two are replicated down the tree to XRv3 and XRv4 using labels 93000 and 94001 respectively. The LFIB confirms this.

```
RP/0/0/CPU0:XRv2#show rsvp reservation p2mp-id 134 detail | utility egrep
'Destination|Label'
```

Destination addr: 213.12.12.12. Subgroup: 1/213.11.11.11.

Labels: Local downstream: 92008.
 Destination addr: 213.13.13.13. Subgroup: 2/213.11.11.11.
 Labels: Local downstream: 92008, Outgoing downstream: 93000.
 Destination addr: 213.14.14.14. Subgroup: 3/213.11.11.11.
 Labels: Local downstream: 92008, Outgoing downstream: 94001.

```
RP/0/0/CPU0:XRv2#show mpls forwarding labels 92008
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
92008	93000	P2MP TE: 134	Gi0/0/0/0.523	213.12.13.13	0
	94001	P2MP TE: 134	Gi0/0/0/0.524	213.12.14.14	0

Rather than trace the signaling path on every single router, we will check to see if XRv1 receives the RSVP RESV messages inbound, which would bring the tunnels up.

```
RP/0/0/CPU0:XRv1#show mpls traffic-eng tunnels p2mp 134 brief
```

TUNNEL NAME	DESTINATION	STATUS	STATE
tunnel-mte134	213.12.12.12	up	up
tunnel-mte134	213.13.13.13	up	up
tunnel-mte134	213.14.14.14	up	up

Displayed 1 (of 2) heads, 0 (of 0) midpoints, 0 (of 0) tails
 Displayed 1 up, 0 down, 0 recovering, 0 recovered heads

In terms of multicast, we must enable PIM on the MTE interfaces on the head-end, as well as statically join SSM groups to the tunnel. Below are configuration snippets for both head and tail ends. I do not show enabling multicast on interfaces because it's very basic. The tail ends must be told how to perform RPF lookups for multicast sources by referencing the LSP somehow. This is accomplished one of two ways: the old way involves using "static-rpf" and specifying a host route and BGP next-hop, and the new way involves just specifying the BGP next-hop inside an RPL. I use the old way on XRv2 and XRv3, while deploying the new way on XRv4. The concept of a "core-tree" is a way for PIM to have an idea of how to perform RPF lookups. In this case, we use RSVP-TE. In other profiles, we can select other options such as mLDP, in-band signaling, etc.

```
! XRv1
router igmp
 interface tunnel-mte134
   static-group 232.1.1.1 10.5.5.5

router mld
 interface tunnel-mte136
   static-group ff33::1 2001:10:5:11::5

! XRv2 and XRv3 (old method)
multicast-routing
 address-family ipv4
   core-tree-protocol rsvp-te
```

```

static-rpf 10.5.5.5 32 mpls 213.11.11.11
address-family ipv6
  core-tree-protocol rsvp-te
static-rpf 2001:10:5:11::5 128 mpls 213.11.11.11

! XRv4 (new method)
multicast-routing
  address-family ipv4
    core-tree-protocol rsvp-te
  address-family ipv6
    core-tree-protocol rsvp-te

route-policy RPL_LSM_ROOT
  set lsm-root 213.11.11.11
end-policy

router pim
  address-family ipv4
    rpf topology route-policy RPL_LSM_ROOT
  address-family ipv6
    rpf topology route-policy RPL_LSM_ROOT

```

Note: You must use an IPv4/32 or IPv6/128 mask with static RPF. It scales poorly as a result, and which is why the LSM-root method covers all multicast for this MVPN.

```

! Erroneous example
multicast-routing
  address-family ipv6
    static-rpf 2001:10:5:11:: 64 mpls 213.11.11.11
!!% Invalid prefix length: Prefix length should be host mask length.

```

Let's verify that XRv1 accepts traffic coming from its LAN facing CSR5 and forwards it out the MTE tunnel for both IPv4 and IPv6. Both of these MTEs are in the OIL for the (S,G) pairs because of the static IGMP/MLD joins we applied on the head-end. This is the only way to force traffic into the MTE tunnel.

```

RP/0/0/CPU0:XRv1#show pim topology 232.1.1.1 | begin 232
(10.5.5.5,232.1.1.1)SPT SSM Up: 00:57:27
JP: Join(00:00:38) RPF: GigabitEthernet0/0/0/0.551,10.5.11.5 Flags:
tunnel-mte134 00:57:27 fwd LI LH

RP/0/0/CPU0:XRv1#show pim ipv6 topology FF33::1 | begin ff33
(2001:10:5:11::5,ff33::1) SPT
SSM Up: 00:06:38 JP: Join(00:00:32) Flags:
RPF: GigabitEthernet0/0/0/0.551,fe80::5*
tunnel-mte136 00:06:38 fwd LI LH

```

Next, we verify the three tail-ends all have valid RPF lookups for the multicast sources. We will verify XRv2 without touching XRv3 since their configuration is identical. I cannot explain why all of these RPF

lookups are resulting in “null” despite first recursing to a valid MPLS-related entry. I have to assume this is an XRv issue.

```
RP/0/0/CPU0:XRv2#show pim rpf 10.5.5.5
Table: IPv4-Unicast-default
* 10.5.5.5/32 [200/10]
    via MPLS with SPE 213.11.11.11
    via Null with rpf neighbor 0.0.0.0

RP/0/0/CPU0:XRv2#show pim ipv6 rpf 2001:10:5:11::5
Table: IPv6-Unicast-default
2001:10:5:11::5/128 [200/0]
    via SPE ::ffff:213.11.11.11, MPLS
    via ::ffff:213.11.11.11, Null
```

XRv4’s output doesn’t look much better, but it’s a little different with the LSM-root syntax.

```
RP/0/0/CPU0:XRv4#show pim rpf 10.5.5.5
Table: IPv4-Unicast-default
* 10.5.5.5/32 [200/20]
    via MPLS with rpf neighbor 0.0.0.0

RP/0/0/CPU0:XRv4#show pim ipv6 rpf 2001:10:5:11::5
Table: IPv6-Unicast-default
2001:10:5:11::5/128 [200/0]
    via ::ffff:213.11.11.11MPLS, Null
```

When we check the PIM topology, all of the entries show what appears to be a valid RPF interface. I am not sure if this is another XRv issue, but it appears inconsistent.

```
RP/0/0/CPU0:XRv2#show pim topology 232.1.1.1 | begin 232
(10.5.5.5,232.1.1.1)SPT SSM Up: 00:57:31
JP: Join(never) RPF: MPLS(SPE),213.11.11.11 Flags:
GigabitEthernet0/0/0/0.582 00:57:31 fwd Join(00:02:52)

RP/0/0/CPU0:XRv2#show pim ipv6 topology FF33::1 | begin ff33
(2001:10:5:11::5,ff33::1) SPT
SSM Up: 00:42:12 JP: Join(never) Flags:
RPF: MPLS(SPE),::ffff:213.11.11.11
GigabitEthernet0/0/0/0.582 00:42:12 fwd Join(00:03:21)

RP/0/0/CPU0:XRv4#show pim topology 232.1.1.1 | begin 232
(10.5.5.5,232.1.1.1)SPT SSM Up: 00:58:19
JP: Join(never) RPF: MPLS(SPE),213.11.11.11 Flags:
GigabitEthernet0/0/0/0.574 00:58:19 fwd Join(00:02:48)

RP/0/0/CPU0:XRv4#show pim ipv6 topology FF33::1 | begin ff33
(2001:10:5:11::5,ff33::1) SPT
```

```
SSM Up: 00:42:54 JP: Join(never) Flags:
RPF: MPLS (SPE), ::ffff:213.11.11.11
GigabitEthernet0/0/0/0.574 00:42:54 fwd Join(00:02:35)
```

We will use CSR5 to send traffic from its loopback (10.5.5.5) to the proper SSM group (232.1.1.1). Using debugging on any of the tail end routers, we see packets being dropped by the MFIB process due to not being able to determine the egress interface. I suspect there is an RPF problem still. Debugging on XRv2 at least reveals that packets are making it into the MTE and being label-switched correctly. Packets arriving from XRv1 with label 92008 are being replicated correctly down to XRv3 and XRv4. This indicates that the MPLS encapsulation process is working correctly at the head-end, and the midpoints are able to swap labels.

```
! Any tail end router
RP/0/0/CPU0:XRv4#debug mfib ipv4 netio drop
netio[309]: MFIB : TID 0xe0000000 Pkt (10.5.5.5, 232.1.1.1) dropped [Cannot
get input IF handle in packet 0xb0c22e84]: Null input I/F
```

```
! XRv2
mpls_switch: FINT0_0_CPU0, mpls eos 1, ttl 254, len 122, inlabel 92008,
tbl_id=0x0, vrf_id=0x0 in=0x500
mpls_rewrite
mpls_rewrite: tos 0 eos 1 ttl 254, out_label 93000, #labels 1, RA - 0,
out_intf GigabitEthernet0_0_0_0.523
mpls_rewrite: tos 0 (0x0) eos 1 ttl 254, out_label 93000, top_of_stack
0x16b481fe
mpls_p2mp_switch_mpi_egress:
mpls_rewrite
mpls_rewrite: tos 0 eos 1 ttl 254, out_label 94001, #labels 1, RA - 0,
out_intf GigabitEthernet0_0_0_0.524
```

This is a tricky pitfall. The router describes multicast-routing being enabled on some specific interfaces, but “interface all enable” is thrown in there quietly. If you enable multicast-routing on the upstream facing MPLS interface, the RPF check “looks” better. On XRv2, XRv3, and XRv4, I just enabled multicast on all interfaces in an effort to see this work (not shown). With that in place, the RPF show commands appear complete for IPv4 only. I theorize that IPv6 RPF is still broken since the core is not IPv6 capable, and 6PE cannot help mask this.

```
RP/0/0/CPU0:XRv2#show pim rpf 10.5.5.5
Table: IPv4-Unicast-default
* 10.5.5.5/32 [200/10]
    via MPLS with SPE 213.11.11.11
    via GigabitEthernet0/0/0/0.512 with rpf neighbor 213.11.12.11

RP/0/0/CPU0:XRv2#show pim ipv6 rpf 2001:10:5:11::5
Table: IPv6-Unicast-default
2001:10:5:11::5/128 [200/0]
```

```

via SPE ::ffff:213.11.11.11, MPLS
via ::ffff:213.11.11.11, Null

RP/0/0/CPU0:XRv4#show pim rpf 10.5.5.5
Table: IPv4-Unicast-default
* 10.5.5.5/32 [200/20]
  via MPLS GigabitEthernet0/0/0/0.524 with rpf neighbor 213.12.14.12

RP/0/0/CPU0:XRv4#show pim ipv6 rpf 2001:10:5:11::5
Table: IPv6-Unicast-default
2001:10:5:11::5/128 [200/0]
  via ::ffff:213.11.11.11MPLS, Null

```

Testing reveals that it still doesn't matter, as the debug still shows dropped packets and the MFIB counters do not change on any tail-end router. The head-end shows them being sent into the tunnel, and debugs from XRv2 show it being label-switched. I believe this is an XRv data-plane limitation.

```

RP/0/0/CPU0:XRv1#show mfib route 10.5.5.5 | include Forwarding
IP Multicast Forwarding Information Base
Forwarding/Replication Counts: Packets in/Packets out/Bytes out
SW Forwarding Counts: 1906/1906/190600

RP/0/0/CPU0:XRv2#show mfib route 10.5.5.5 | include Forwarding
IP Multicast Forwarding Information Base
Forwarding/Replication Counts: Packets in/Packets out/Bytes out
SW Forwarding Counts: 0/0/0

RP/0/0/CPU0:XRv4#show mfib route 10.5.5.5 | include Forwarding
IP Multicast Forwarding Information Base
Forwarding/Replication Counts: Packets in/Packets out/Bytes out
SW Forwarding Counts: 0/0/0

```

Additional Reading – Reference configurations “mvpn-8”

28.7 Profile 9: Default MDT – MLDP – MP2MP – BGP-AD – PIM C-mcast Signaling

For this test, begin with the MVPN Profile 1 configuration where all of the MVPN BGP is set up already, plus the MP2MP/P2MP trees are built. XRv4 is the MP2MP root and there are some other P2MP roots related to SSM joins. Enabling BGP AD is very easy, and the snippets are below.

```

! BGP AD on XE
vrf definition MC
  address-family ipv4
    mdt auto-discovery mldp
  address-family ipv6
    mdt auto-discovery mldp

! BGP AD on XR

```

```
multicast-routing
vrf MC
  address-family ipv4
    bgp auto-discovery mldp
  address-family ipv6
    bgp auto-discovery mldp
```

I admit, the use-case of using BGP AD with mLDP MP2MP trees and PIM c-mcast signaling is very limited. It does not appear capable of dynamically distributing MP2MP root information, so all nodes still need to have roots configured. Each PE generates a Type 1 MVPN route, which is Intra-AS I-PMSI AD. This route contains the currently selected root and lists its tunnel type. Looking at CSR5, we see a Type 1 router for every PE, in the VPN. This membership is controlled by the traditional unicast route-target.

```
R5#show bgp ipv4 mvpn vrf MC | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 213:5 (default for vrf MC)
*>i [1] [213:5] [213.1.1.1]/12
                213.1.1.1                0      100      0 ?
*> [1] [213:5] [213.5.5.5]/12
                0.0.0.0                32768 ?
*>i [1] [213:5] [213.6.6.6]/12
                213.6.6.6                0      100      0 ?
*>i [1] [213:5] [213.7.7.7]/12
                213.7.7.7                0      100      0 ?
*>i [1] [213:5] [213.8.8.8]/12
                213.8.8.8                0      100      0 ?
*>i [1] [213:5] [213.12.12.12]/12
                213.12.12.12              100      0 i
*>i [3] [213:5] [10.2.7.2] [232.0.0.5] [213.7.7.7]/22
                213.7.7.7                0      100      0 ?
```

Looking into CSR5's specific route, we can see the root information encoded in hex. Tunnel-type 7 is an mLDP MP2MP tunnel, and the root address is 213.14.14.14. Also notice that the no-export community is set because we did not configure inter-AS AD. The importance of this route is to signal membership in an I-PMSI; that is to say, others who receive this route know that this PE is reachable via an MP2MP tunnel rooted at 213.14.14.14 for this MVPN instance.

```
R5#show bgp ipv4 mvpn vrf MC route-type 1 213.5.5.5
BGP routing table entry for [1][213:5][213.5.5.5]/12, version 18
Paths: (1 available, best #1, table MVPNv4-BGP-Table, not advertised to EGBP
peer)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (213.5.5.5)
```



```

Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
Community: no-export
Extended Community: RT:213:5
PMSI Attribute: Flags: 0x0, Tunnel type: 7, length 24, label: exp-null,
tunnel parameters: 0800 0104 D50E 0E0E 000E 0200 0B00 0213 0000 0213 0000
0000
rx pathid: 0, tx pathid: 0x0

```

There is also a single Type 3 BGP route, which is an S-PMSI AD route. This is originated by a PE that has a source behind it and someone else has expressed interest. This entry is permanent because the C(S,G) shown in the group is from an IGMPv3 SSM join from CSR3 and CSR4. It only takes one node to request multicast, and then the ingress PE (CSR7) announces that there is interest for a C(S,G). Notice the show command requires the route type, C-S, C-G, and BGP next-hop. You cannot shortcut this. The tunnel type is 2 to signal an mLDP P2MP tree and the root is 213.7.7.7 inside the tunnel message.

```

R5#show bgp ipv4 mvpn vrf MC route-type 3 10.2.7.2 232.0.0.5 213.7.7.7
BGP routing table entry for [3][213:5][10.2.7.2][232.0.0.5][213.7.7.7]/22,
version 12
Paths: (1 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGP
peer)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from [3][213:7][10.2.7.2][232.0.0.5][213.7.7.7]/22
(global)
    213.7.7.7 (metric 20) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Community: no-export
      Extended Community: RT:213:7
      Originator: 213.7.14.7, Cluster list: 213.12.12.12
      PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 24, label: exp-null,
tunnel parameters: 0600 0104 D507 0707 000E 0200 0B00 0213 0000 0213 0000
0001
      rx pathid: 0, tx pathid: 0x0

```

Testing IPv6 is a little more interesting. There are no Type 3 routes by default, because in order to have a router originate one, it needs to have received a C(S,G) join to indicate there is a source behind it, and someone is interested.

```

R8#show bgp ipv6 mvpn vrf MC | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 213:8 (default for vrf MC)
*>i [1][213:8][213.6.6.6]/12
      213.6.6.6      0      100      0 ?
*>i [1][213:8][213.7.7.7]/12
      213.7.7.7      0      100      0 ?
*> [1][213:8][213.8.8.8]/12

```

::

32768 ?

In this case, CSR6 originates a Type 3 route for the traffic sourced by CSR4 to the embedded RP group. This route was not generated until CSR4 started sending traffic, because CSR3 did not know the source ahead of time. The route was not generated when CSR4 started sending, but when CSR6 received the PIM join from CSR8.

```
R4#ping FF7E:240:2001:10:2:7::1 repeat 10000 time 1
Output Interface: GigabitEthernet2.546
```

```
R6#debug bgp ipv6 mvpn updates
BGP(16): nettable_walker
[3] [213:6] [2001:10:4:6::4] [FF7E:240:2001:10:2:7:0:1] [213.6.6.6]/46 route
sourced locally
```

```
BGP(16): (base) 213.12.12.12 send UPDATE (format)
[3] [213:6] [2001:10:4:6::4] [FF7E:240:2001:10:2:7:0:1] [213.6.6.6]/46, next
213.6.6.6, metric 0, path Local, extended community RT:213:6
```

Debugging CSR8 shows similar results. The update is received from CSR6 and added to the MVPN RIB. Notice the tunnel type is 2, which is an mLDP P2MP tunnel using S-PMSI, versus 7 which was used in the Type 1 route to detail the I-PMSI information (MP2MP tree).

```
R8#debug bgp ipv6 mvpn updates
BGP(16): 213.12.12.12 rcvd UPDATE w/ attr: nexthop 213.6.6.6, origin ?,
localpref 100, metric 0, originator 213.6.6.6, clusterlist 213.12.12.12,
community no-export, extended community RT:213:6
. BGP(16): 213.12.12.12 rcvd
[3] [213:6] [2001:10:4:6::4] [FF7E:240:2001:10:2:7:0:1] [213.6.6.6]/46

BGP(16): add RIB route
[3] [213:8] [2001:10:4:6::4] [FF7E:240:2001:10:2:7:0:1] [213.6.6.6]/46
Flags: 0x0, Tunnel type: 2, label: exp-null, tunnel parameters: 0600 0104
FFFFFFFFFFFFFFFFD506 0606 000E 0200 0B00 0213 0000 0213 FFFFFFFFFFFFFFFFFF8000 0001
```

We can clearly see the route in the table with the proper next hop. Looking at the route details confirms the debug, as we can see the root encoded in the tunnel data. The show command is rather long, so it is highlighted for emphasis.

```
R8#show bgp ipv6 mvpn vrf MC | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 213:8 (default for vrf MC)
*>i [1] [213:8] [213.6.6.6]/12
                        213.6.6.6                0      100      0 ?
*>i [1] [213:8] [213.7.7.7]/12
                        213.7.7.7                0      100      0 ?
*>  [1] [213:8] [213.8.8.8]/12
```

```

:: 32768 ?
*>i [3][213:8][2001:10:4:6::4][FF7E:240:2001:10:2:7:0:1][213.6.6.6]/46
213.6.6.6 0 100 0 ?

R8#show bgp ipv6 mvpn vrf MC route-type 3 2001:10:4:6::4
FF7E:240:2001:10:2:7:0:1 213.6.6.6
BGP routing table entry for
[3][213:8][2001:10:4:6::4][FF7E:240:2001:10:2:7:0:1][213.6.6.6]/46, version
12
Paths: (1 available, best #1, table MVPNV6-BGP-Table, not advertised to EBGp
peer)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from
[3][213:6][2001:10:4:6::4][FF7E:240:2001:10:2:7:0:1][213.6.6.6]/46 (global)
  213.6.6.6 (metric 30) from 213.12.12.12 (213.12.12.12)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    Community: no-export
    Extended Community: RT:213:6
    Originator: 213.6.6.6, Cluster list: 213.12.12.12
    PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 24, label: exp-null,
tunnel parameters: 0600 0104 D506 0606 000E 0200 0B00 0213 0000 0213 8000
0001
    rx pathid: 0, tx pathid: 0x0

```

Because the unicast architecture is split in half for IPv6 VPN, BGP AD routes from one half do not leak into the other. Again, this is controlled by the route-target. When we looked at the bottom half routers, we did not see I-PMSI AD messages (Type 1 routes) from any top half routers. Likewise, checking a top router shows nothing from the bottom half, and the dynamic Type 3 announcement is likewise not seen.

```

R5#show bgp ipv6 mvpn vrf MC | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 213:5 (default for vrf MC)
*>i [1][213:5][213.1.1.1]/12
      213.1.1.1      0      100      0 ?
*> [1][213:5][213.5.5.5]/12
      ::      32768 ?
*>i [1][213:5][213.12.12.12]/12
      213.12.12.12      100      0 i

```

Last, we will examine OAM for MP2MP and P2MP trees in this profile. Unlike Profile 12 where we were limited only to unidirectional P2MP trees (the root specified had to be local), with MP2MP pings we can test from any endpoint; the tree is bidirectional. We specify the FEC and opaque type and can quickly discover all the endpoints in the mVPN on this MDT.

```

R6# show mpls mldp database summary
LSM ID      Type      Root      Decoded Opaque Value      Client Cnt.
16          P2MP      213.7.7.7 [mdt 213:213 1]          1

```

```

13          MP2MP      213.14.14.14      [mdt 213:213 0]          1

R6#ping mpls mldp mp2mp 213.14.14.14 mdt 213:213 0
mp2mp Root node addr 213.14.14.14
Opaque type MDT, oui:index 0x213:0213, mdtnum 0
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
    timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
[snip]

```

```

Request #1
! reply addr 213.8.14.8
! reply addr 213.12.13.12
! reply addr 213.5.14.5
! reply addr 213.1.14.1
! reply addr 213.7.13.7

```

```

Round-trip min/avg/max = 27/93/170 ms

```

```

R7#show mpls mldp database summary
LSM ID      Type      Root          Decoded Opaque Value      Client
Cnt.
16          P2MP      213.7.7.7      [mdt 213:213 1]          3
13          MP2MP      213.14.14.14   [mdt 213:213 0]          2

```

```

R7#ping mpls mldp p2mp 213.7.7.7 mdt 213:213 1
p2mp Root node addr 213.7.7.7
Opaque type MDT, oui:index 0x213:0213, mdtnum 1
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
    timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
[snip]

```

```

Type escape sequence to abort.

```

```

Request #1
! reply addr 213.6.13.6
! reply addr 213.8.14.8

```

```

Round-trip min/avg/max = 87/143/200 ms

```

Quickly testing XRv, we see the command syntax and outputs are very similar. I believe this to be a data-plane limitation since MVPN is not supported, the OAM components also fail.

```

RP/0/0/CPU0:XRv2#ping mpls mldp mp2mp 213.14.14.14 mdt 213:213 0
Sending 1, 100-byte MPLS Echos to mldp mp2mp 213.14.14.14 mdt (vpn-
id:213:213, mdt:0),
    timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]

```

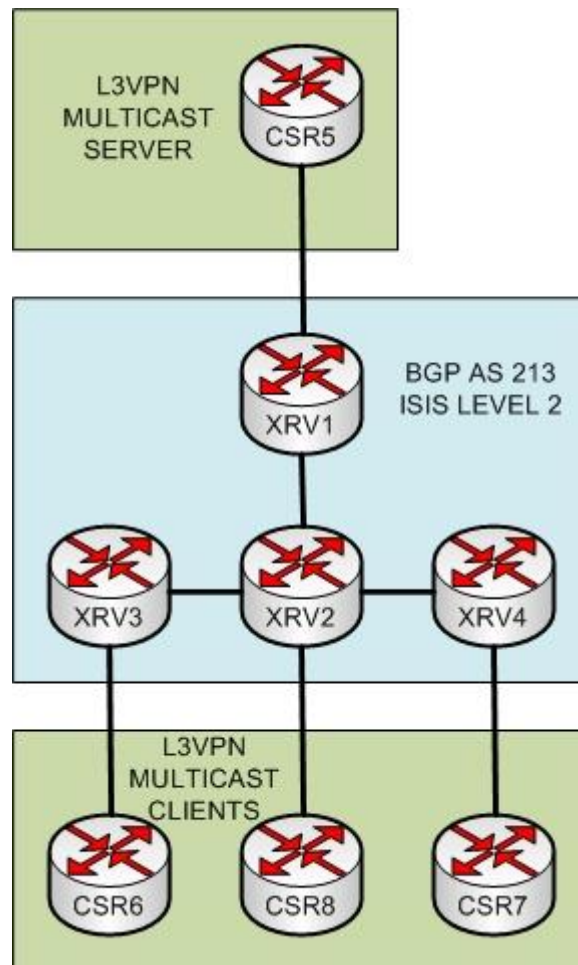
Request #1

.

Additional Reading – Reference configurations “mvpn-9”

28.8 Profile 10: VRF Static – P2MP TE - BGP-AD

This is very similar to profile 8 except now the CSRs are MPLS L3VPN customers, so their routing is carried by VPNV4/v6. Because XRv only supports MVPN from the headend, the tunnel tail ends cannot deliver multicast to the end clients, but we can verify the control-plane and some parts of the data-plane. **Note: This profile does not appear to be supported for IPv6 since the “mdt static” command is not available under that AF. The remaining relevant IPv6 configuration is left in the configuration files (as close as I could get) for reference.**



The configuration for a static multipoint TE (MTE) tunnel is very simple. Each destination gets its own set of path options, etc.

! XRV1

```

interface tunnel-mte134
  ipv4 unnumbered Loopback0
  destination 213.12.12.12
  path-option 10 dynamic

  destination 213.13.13.13
  path-option 10 dynamic

  destination 213.14.14.14
  path-option 10 dynamic

  signalled-bandwidth 20000

```

In terms of signaling, P2MP RSVP-TE requires a lot of “core state”. This is because each sub-LSP (tunnel destination within the overall multipoint tunnel) is signaled independently. That is, for N destinations, N RSVP PATH messages are generated from the head-end, and N RSVP RESV messages are generated from the tail-ends. In this case, XRv1 generates three separate PATH messages, one for each destination. Just like ordinary P2P RSVP TE, each PATH message has a label request object (LRO).

```

RP/0/0/CPU0:XRv1#show rsvp sender

```

Destination Add	P2MPId	Source Add	SPort	Input IF	Rate	Burst	Prot
213.13.13.13	134	213.11.11.11	10003	No	20M	1K	Off
213.14.14.14	134	213.11.11.11	10003	No	20M	1K	Off
213.12.12.12	134	213.11.11.11	10003	No	20M	1K	Off

The difference comes at the branch points, such as XRv2; the router can determine if multiple sub-LSPs as part of the same LSP (same P2MP-ID) share an upstream interface. In this case, XRv1 is upstream from XRv2 for all three of the sub-LSPs that terminate on XRv2 (local), XRv3, and XRv4. XRv2 will still issue (N) RSVP RESV messages to XRv1, but they contain the same label object. Even though XRv2 is one hop away, PHP cannot be used for LSM for several reasons. Since there is only one label in the stack to begin with, the core is not multicast enabled, and even if it were, the C-multicast information is unknown to the P routers. Besides, XRv2 still has to account for the other downstream neighbors also, so PHP doesn’t make sense for P2MP TE.

```

RP/0/0/CPU0:XRv1#show rsvp reservation

```

Destination Add	P2MPId	Source Add	SPort	Input IF	Sty	Serv	Rate	Burst
213.13.13.13	134	213.11.11.11	10003	Gi0/0/0/0.512	SE	LOAD	20000K	1K
213.14.14.14	134	213.11.11.11	10003	Gi0/0/0/0.512	SE	LOAD	20000K	1K
213.12.12.12	134	213.11.11.11	10003	Gi0/0/0/0.512	SE	LOAD	20000K	1K

From XRv1’s perspective, it will use this label to send traffic out (it’s XRv2’s local label) and downstream (away from the root). From XRv2’s perspective, the label is oriented the other way, incoming and upstream-facing.

```
RP/0/0/CPU0:XRv1#show rsvp reservation detail | utility egrep 'Destin|Label'
Destination addr: 213.13.13.13. Subgroup: 1/213.11.11.11.
Labels: Outgoing downstream: 92006.
Destination addr: 213.14.14.14. Subgroup: 2/213.11.11.11.
Labels: Outgoing downstream: 92006.
Destination addr: 213.12.12.12. Subgroup: 3/213.11.11.11.
Labels: Outgoing downstream: 92006.
```

On XRv2, traffic arriving with label 92006 is replicated down the individual trees using the RSVP-bound labels from the next downstream router. Notice that the path terminating on XRv2 does not have an outgoing downstream label since it is the tail end of the LSP. The LFIB shows the replication for leaves past the branch point. Be careful not to confuse this with load sharing. Again, no PHP is allowed here because XRv3 and XRv4 need to know in which VPN this traffic belongs. I am not sure why the unlabelled path to CSR8 is not shown in the LFIB.

```
RP/0/0/CPU0:XRv2#show rsvp reservation detail | utility egrep 'Destin|Label'
Destination addr: 213.13.13.13. Subgroup: 1/213.11.11.11.
Labels: Local downstream: 92006, Outgoing downstream: 93005.
Destination addr: 213.14.14.14. Subgroup: 2/213.11.11.11.
Labels: Local downstream: 92006, Outgoing downstream: 94005.
Destination addr: 213.12.12.12. Subgroup: 3/213.11.11.11.
Labels: Local downstream: 92006.
```

```
RP/0/0/CPU0:XRv2#show mpls forwarding labels 92006
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
92006	93005	P2MP TE: 134	Gi0/0/0/0.523	213.12.13.13	0
	94005	P2MP TE: 134	Gi0/0/0/0.524	213.12.14.14	0

The tail-end routers LFIB output does not look promising, much like XRv2's (lack of) customer LFIB entry, but I believe this is correct. Traffic arrives via the P2MP tunnel and is delivered to the customer unlabelled. Looking at XRv4, we see the LFIB then check the VRF-aware PIM topology (mroutetable).

```
RP/0/0/CPU0:XRv4#show mpls forwarding labels 94005
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94005	Unlabelled	P2MP TE: 134			

```
RP/0/0/CPU0:XRv4#show pim vrf RIP topology 232.1.1.1
(10.5.5.5,232.1.1.1)SPT SSM Up: 00:32:29
JP: Join(never) RPF: MPLS(SPE),213.11.11.11 Flags:
GigabitEthernet0/0/0/0.574 00:32:29 fwd Join(00:02:32)
```

P2MP RSVP-TE only supports SSM, which is why I used 232.1.1.1. The tail-end devices need to instruct the multicast process to use a different “core-tree” for RPF lookups, specifically using the TE tunnel. We also enable BGP AD for signaling of S-PMSI messaging specifically.

```
! All tail end routers
multicast-routing
vrf RIP
  address-family ipv4
    bgp auto-discovery p2mp-te
    core-tree-protocol rsvp-te
```

We can verify that RPF is correct another way also. The PIM topology said MPLS(SPE) followed by XRv1’s loopback which wasn’t very descriptive. We expect the RPF next-hop to be the BGP connector attribute, which is XRv1’s loopback0 213.11.11.11/32. The connector attribute is useful in Inter-AS scenarios.

```
RP/0/0/CPU0:XRv4#show pim vrf RIP rpf 10.5.5.5
Table: IPv4-Unicast-default
* 10.5.5.5/32 [200/1]
  via MPLS with rpf neighbor 213.11.11.11
  Connector: 213:11:213.11.11.11, Nexthop: 213.11.11.11
```

```
RP/0/0/CPU0:XRv4#show bgp vpnv4 unicast rd 213:11 10.5.5.5 | include
Connector
  Connector: type: 1, Value:213:11:213.11.11.11
```

The headend configuration is also awkward and took me a long time to understand. The MTE tunnel itself cannot be inside a VRF (the parser does not let you enable PIM on it), but we must configure a VRF-aware IGMP static-group for the SSM traffic we want to force downstream. This then creates a PIM join towards the CE (in the VRF), which is upstream towards the c-multicast source. Be sure to enable PIM on the global loopback0 (same address used for BGP peerings) and set the MDT source to that interface for all AFs; remember only IPv4 appears to work in v5.3.0. In the VRF, the only interface is the PE-CE link, but the MTE is statically assigned within that VRF. Interestingly, configuring IGMP parameters on an interface not enabled for multicast and not in the same VRF still works because of the “mdt static” assignment which maps an MTE tunnel to a VRF.

```
! XRv1
vrf RIP
  address-family ipv4
    bgp auto-discovery p2mp-te
    mdt static p2mp-te tunnel-mte134

router igmp
vrf RIP
  interface tunnel-mte134
    static-group 232.1.1.1 10.5.5.5
```


When we check the multicast state, we see the VRF on the PE shows the proper information, and issues an SSM (S,G) join towards CSR5. **Pitfall: I cannot get it working on the head-end unless I use “interface all enable” under the VRF.** PIM must be enabled on the MTE but you cannot do this explicitly, and if you only enable PIM on the PE-CE link, it simply does not work. Fortunately, you don’t have to enable PIM on the PE-P link, because the core is PIM-free. XRv PIM output accounts for the PIM DR as a neighbor, so if PIM is enabled on a multi-access link and the neighbor count is 1, don’t assume another router is there. Obviously there are no PIM neighbors on loopback0 or the MTE tunnel.

```
RP/0/0/CPU0:XRv1#show pim vrf RIP topology 232.1.1.1
Entry flags: MT - Crossed Data MDT threshold
(10.5.5.5,232.1.1.1)SPT SSM Up: 00:13:01
JP: Join(00:00:44) RPF: GigabitEthernet0/0/0/0.551,10.5.11.5 Flags: MT
tunnel-mte134 00:13:01 fwd LI LH
```

```
R5#show ip mroute 232.1.1.1 | begin \
(10.5.5.5, 232.1.1.1), 00:03:09/00:03:27, flags: sT
Incoming interface: Loopback0, RPF nbr 0.0.0.0
Outgoing interface list:
GigabitEthernet2.551, Forward/Sparse, 00:03:09/00:03:27
```

```
RP/0/0/CPU0:XRv1#show pim interface
PIM interfaces in VRF default
Address      Interface
PIM Nbr      Hello DR      DR
      Count Intvl Prior
213.11.11.11 Loopback0    on  1      30     1      this system
213.11.12.11 GigabitEthernet0/0/0/0.512 off 0      30     1      not elected
```

```
RP/0/0/CPU0:XRv1#show pim vrf RIP interface
PIM interfaces in VRF RIP
Address      Interface
PIM Nbr      Hello DR      DR
      Count Intvl Prior
213.11.11.11 tunnel-mte134 on  1      30     1      this system
10.5.11.11   GigabitEthernet0/0/0/0.551 on  2      30     1      this system
```

I enabled a 20 Mbps bandwidth reservation on this tunnel since this capability is the biggest benefit of RSVP-TE P2MP tunnels. When we checked the RSVP reservations on XRv1, we saw three reservations with 20 Mbps each. If we check the RSVP interface parameters, we see only 20 Mbps was allocated, and this is because it’s a single flow, not N different flows. This is desirable, and leads us to believe that the head-end router will not be replicating traffic.

```
RP/0/0/CPU0:XRv1#show rsvp interface
*: RDM: Default I/F B/W % : 75% [default] (max resv/bc0), 0% [default] (bc1)
Interface  MaxBW (bps)  MaxFlow (bps) Allocated (bps)  MaxSub (bps)
-----
Gi0/0/0/0.512 100M        100M          20M ( 20%)      0
```

From CSR5, I will generate some traffic and enable MPLS debugging on the XRv routers to watch it transit the network. Although the tunnel tail-ends won't deliver it to the final customer (XRv data-plane limitation), we can verify the LSP. This at least proves that MPLS imposition for MVPN works, or else XRv2 would have never seen LSM. MPLS packet debugging on XRv2 shows that when a packet along the P2MP LSP arrives, it is replicated to both routers as the LFIB directs. No messages are shown about its delivery to CSR8, presumably because it's broken. Even with these debugs and proper forwarding, the LFIB counters do not increment in the LFIB on XRv2.

```
! XRv2
netio[309]: mpls_switch: FINT0_0_CPU0, mpls eos 1, ttl 254, len 122, inlabel
92006, tbl_id=0x0, vrf_id=0x0 in=0xa00
netio[309]: mpls_p2mp_switch: dir: egress
netio[309]: mpls_p2mp_switch_egress:
netio[309]: mpls_p2mp_switch_mpi_egress:
netio[309]: mpls_rewrite
netio[309]: mpls_rewrite: tos 0 eos 1 ttl 254, out_label 93005, #labels 1, RA
- 0, out_intf GigabitEthernet0_0_0_0.523
netio[309]: mpls_p2mp_switch_mpi_egress:
netio[309]: mpls_rewrite
netio[309]: mpls_rewrite: tos 0 eos 1 ttl 254, out_label 94005, #labels 1, RA
- 0, out_intf GigabitEthernet0_0_0_0.524

! XRv3
netio[309]: mpls_switch: GigabitEthernet0_0_0_0.523, mpls eos 1, ttl 254, len
122, inlabel 93005, tbl_id=0x0, vrf_id=0x0 in=0xb00
netio[309]: mpls_p2mp_switch: dir: ingress

! XRv4
netio[309]: mpls_switch: GigabitEthernet0_0_0_0.524, mpls eos 1, ttl 253, len
122, inlabel 94005, tbl_id=0x0, vrf_id=0x0 in=0xb00
netio[309]: mpls_p2mp_switch: dir: ingress
```

At this point, we will perform some additional control-plane verifications, specifically examining the Type-4 Leaf AD message. We haven't used this in many profiles. This message is used by leaves (tail ends) to tell the originator of a Type-3 S-PMSI route (root) that they are interested in a particular C(S,G). Leaves will only send this if the Type-3 message has the "leaf information requested" flag inside the route. Without the command, we have no Leaf AD messages, because XRv1 did not signal it.

```
RP/0/0/CPU0:XRv1#show bgp ipv4 mvpn route-type 4
[no output]
```

```
RP/0/0/CPU0:XRv1#show bgp ipv4 mvpn vrf RIP
[3][32][10.5.5.5][32][232.1.1.1][213.11.11.11]/120 | include PMSI
PMSI: flags 0x00, type 1, label 0, ID 0x00000008600000086d50b0b0b
```

We can enable this flag per VRF, per address-family under the BGP AD stanza. Nothing else within the PMSI attributes changes; it is still an RSVP P2MP tunnel (type 1) and still has root 213.11.11.11. When we make the configuration change below and verify, we see that the other three XRv routers all signal to XRv1 their interest in the particular S-PMSI.

```
! XRv1
multicast-routing
vrf RIP
address-family ipv4
  bgp auto-discovery p2mp-te
  leaf-info-required

RP/0/0/CPU0:XRv1#show bgp ipv4 mvpn route-type 4 | begin Network
Network                Next Hop                Metric LocPrf Weight Path
Route Distinguisher: 213:11 (default for vrf RIP)
*>i[4][3][213:11][32][10.5.5.5][32][232.1.1.1][213.11.11.11][213.12.12.12]/224
                213.12.12.12                100          0 i
*>i[4][3][213:11][32][10.5.5.5][32][232.1.1.1][213.11.11.11][213.13.13.13]/224
                213.13.13.13                100          0 i
*>i[4][3][213:11][32][10.5.5.5][32][232.1.1.1][213.11.11.11][213.14.14.14]/224
                213.14.14.14                100          0 i

RP/0/0/CPU0:XRv1#show bgp ipv4 mvpn vrf RIP
[3][32][10.5.5.5][32][232.1.1.1][213.11.11.11]/120 | include PMSI
PMSI: flags 0x01, type 1, label 0, ID 0x0000008600000086d50b0b0b
```

Looking into the details of the Leaf AD message, we see that it contains a lot of information; the information needs to specify all the attributes of the S-PMSI to which the leaf has signaled interest, plus additional information. The first part of the Leaf AD NLRI is the “route key” which contains the exact Type-3 route. The last piece is the originating router’s IP address, which is XRv2’s loopback. The route otherwise looks identical to the Type-3 on the surface. It also contains two extended communities. The dynamic RT (aka VRF-import) is set to the root’s IP address followed by a random number. Because none of XRv1’s VPNv4 routes match this dynamic RT (yet it was imported), I am assuming that the “0” on the end of it means that it doesn’t have to map directly to a VPNv4 route as the Type-6/7 c-mcast signaling messages do. I show the output from the two local VPNv4 routes on XRv1 to prove that the numbers don’t match. I also cannot find any documentation on what exactly “SEG-NH” means, but my guess is “S-PMSI Egress Next-Hop”, which would make sense considering it is XRv2’s loopback (tail end).

```
RP/0/0/CPU0:XRv1#show bgp ipv4 mvpn vrf RIP
[4][3][213:11][32][10.5.5.5][32][232.1.1.1][213.11.11.11][213.12.12.12]/224
[snip]
Local, (Received from a RR-client)
  213.12.12.12 (metric 10) from 213.12.12.12 (213.12.12.12)
  Origin IGP, localpref 100, valid, internal, best, group-best, import-
candidate, imported
  Received Path ID 0, Local Path ID 1, version 34
```

```
Extended community: SEG-NH:213.12.12.12:0 RT:213.11.11.11:0
Source VRF: RIP, Source Route Distinguisher: 213:11
```

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast vrf RIP 10.5.5.5 | include VRF
Extended community: VRF Route Import:213.11.11.11:17 Source AS:213:0
RT:213:5
```

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast vrf RIP 10.5.11.0 | include VRF
Extended community: VRF Route Import:213.11.11.11:17 Source AS:213:0
RT:213:5
```

Another interesting behavior in this profile is that only the head-end seems to originate a Type-1 I-PMSI. We were led to believe all routers do this so they can establish some kind of “default MDT” to communicate. There is no such concept here since all the multicast joins and RPF information has been pre-configured and BGP is not terribly involved. The Type-1 originating head-end router doesn’t even have a PMSI attribute, so only the S-PMSI route (Type-3) has it, and it specified P2MP TE. This makes sense since only SSM is supported.

```
RP/0/0/CPU0:XRv1#show bgp ipv4 mvpn vrf RIP [1][213.11.11.11]/40 | begin
Local
  Local
    0.0.0.0 from 0.0.0.0 (213.11.11.11)
      Origin IGP, localpref 100, valid, redistributed, best, group-best,
import-candidate
      Received Path ID 0, Local Path ID 1, version 3
      Community: no-export
      Extended community: RT:213:5
```

Additional Reading – Reference configurations “mvpn-10”

28.9 Profile 11: Default MDT – GRE – BGP-AD – BGP C-mcast Signaling

For this test, begin with the MVPN Profile 3 configuration where PIM is set up already for both the customer and provider networks. The default MDT uses PIM-Bidir while the data MDTs use PIM-SSM. This will allow us to see more variety in the P-tunnel types. The only significant difference here is enabling BGP c-mcast signaling since BGP AD is already enabled per profile 3. This modification is similar to jumping from profile 9 to profile 13 (using mLDP, not GRE). The main benefit/use-case of this model is to gradually transition to next-generation MVPN. An SP can still use the existing PIM/GRE core for forwarding, along with existing QoS and MoFRR, while taking advantage of the newer BGP control-plane in the overlay for additional scalability.

```
! Enable BGP c-mcast signaling in XE
vrf definition MC
  address-family ipv4
    mdt overlay use-bgp
  address-family ipv6
```

```

mdt overlay use-bgp

! Enable BGP c-mcast signaling in XR
router pim
vrf MC
  address-family ipv4
    mdt c-multicast-routing bgp
  address-family ipv6
    mdt c-multicast-routing bgp

```

Because I cannot find a way to disable PIM on the MDT interface on XRv2, we will check for PIM neighbors on the segment and ensure there are none, except XRv2. XRv2 should see no neighbors, meaning that the XE routers are not sending PIM hellos. This means that PIM signaling is not being used over the MDT interfaces.

```

RP/0/0/CPU0:XRv2#show pim vrf MC neighbor | include mdt
213.12.12.12*          mdtMC          04:02:06  00:01:25  1 (DR)  P

```

```

R1#show ip pim vrf MC neighbor | begin ^Neighbor
Neighbor      Interface      Uptime/Expires      Ver      DR
Address
213.12.12.12  Tunnel2          04:02:29/00:01:30  v2       1 / DR B P G

```

```

R8#show ip pim vrf MC neighbor | begin ^Neighbor
Neighbor      Interface      Uptime/Expires      Ver      DR
Address
213.12.12.12  Tunnel3          03:54:42/00:01:19  v2       1 / DR B P G

```

As a quick check, and without even thinking about the topology, we can check the RR (XRv2) for any Type 6 or Type 7 BGP routes. These represents the Shared Tree Join and Source Tree Join (basically (*,G) and (S,G) messages) within the MVPN. If we see any of these routes, we know BGP c-mcast signaling is configured correctly. Specifically, we see three Type-6 joins which represent C(*,G) joins for a PIM-ASM group 225.0.0.1 and RP 10.5.10.10. We also see two Type-7 SPT joins for the C(S,G) PIM-SSM group 232.0.0.5 with source 10.2.7.2. CSR6 and CSR8 are requesting both the ASM and SSM flows, while CSR1 is only requesting the ASM flow.

```

RP/0/0/CPU0:XRv2#show bgp ipv4 mvpn route-type 6 | begin Network
Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 213:5
* i [6] [213:5] [213] [32] [10.5.10.10] [32] [225.0.0.1] /184
      213.1.1.1          0      100      0 ?
*>i      213.6.6.6          0      100      0 ?
* i      213.8.8.8          0      100      0 ?

```

```

Processed 1 prefixes, 3 paths
RP/0/0/CPU0:XRv2#show bgp ipv4 mvpn route-type 7 | begin Network
Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 213:7
*>i [7] [213:7] [213] [32] [10.2.7.2] [32] [232.0.0.5] /184

```

	213.6.6.6	0	100	0 ?
* i	213.8.8.8	0	100	0 ?

We will perform the same drill for IPv6. There is a single receiver, CSR8, requesting PIM-ASM traffic for the embedded-RP group FF7E:240:2001:10:2:7:0:1 with C-RP 2001:10:2:7::2. There are two routers, CSR7 and CSR8, requesting PIM-SSM traffic for the group FF33::1 from source 2001:10:4:6::4. Because the VPNv6 network is segmented horizontally, many IPv6 MVPN functions will be broken. This was intended to highlight unicast/multicast coordination issues as discussed in previous profiles.

```
RP/0/0/CPU0:XRv2#show bgp ipv6 mvpn route-type 6 | begin Network
  Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 213:7
*>i [6] [213:7] [213] [128] [2001:10:2:7::2] [128] [ff7e:240:2001:10:2:7:0:1]/376
      213.8.8.8          0          100          0 ?
```

```
RP/0/0/CPU0:XRv2#show bgp ipv6 mvpn route-type 7 | begin Network
  Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 213:6
*>i [7] [213:6] [213] [128] [2001:10:4:6::4] [128] [ff33::1]/376
      213.7.7.7          0          100          0 ?
* i      213.8.8.8          0          100          0 ?
```

First, we will test IPv4 SSM with a C(S,G) of (10.2.7.2, 232.0.0.5). CSR6 and CSR8 receive IGMPv3 joins for these C(S,G) pairs, then add C-MRIB entries. For brevity, we will focus on CSR8 only. The IGMPv3 join carries the source in “INCLUDE” mode, which is effectively a request for SSM multicast. The C-MRIB entry references a specific data MDT. The reason for this is because a data MDT was signaled for this group as a result of it being candidate for MDT optimization in the MDT data ACL for IPv4. This is signaled by the big ‘Y’.

```
R8#show ip igmp vrf MC groups 232.0.0.5 detail | begin Interface
Interface:      GigabitEthernet2.538
Group:          232.0.0.5
Flags:          SSM
Uptime:         1d01h
Group mode:     INCLUDE
Last reporter:  10.3.8.3
Group source list: (C - Cisco Src Report, U - URD, R - Remote, S - Static,
                  V - Virtual, M - SSM Mapping, L - Local,
                  Ac - Channel accounted towards access control limit)
Source Address  Uptime    v3 Exp   CSR Exp   Fwd  Flags
10.2.7.2        1d01h     00:02:26 stopped   Yes   R
```

```
R8#show ip mroute vrf MC 232.0.0.5 10.2.7.2 | begin \
(10.2.7.2, 232.0.0.5), 21:22:35/00:02:19, flags: sTIYg
  Incoming interface: Tunnel3, RPF nbr 213.7.7.7,
MDT:[213.7.7.7,232.4.7.0]/never
  Outgoing interface list:
```

GigabitEthernet2.538, Forward/Sparse, 21:22:35/00:02:19

```
R8#show ip pim mdt receive
Joined MDT-data [group/mdt number : source]  uptime/expires for VRF: MC
[232.4.7.0 : 213.7.7.7]  21:24:12/stopped
```

It seems impossible that this MDT information could be exchanged between remote PEs within PIM running between them. As seen in profile 3, this is a function of BGP AD. The Type-3 S-PMSI route originated from CSR7 that describes this C(S,G) was received by CSR6 and CSR8. So long as the egress routers are importing the unicast RT of the ingress router (RT:213:7 in this case), the Type-3 is imported to the MVPN. We can see by the tunnel-type that this is a PIM-SSM tunnel inside GRE, rooted at CSR7, using group address 232.4.7.0. The summary view is shown on CSR6 and the details are shown on CSR8.

```
R6#show bgp ipv4 mvpn vrf MC | include \[3\]
*>i [3] [213:6] [10.2.7.2] [232.0.0.5] [213.7.7.7]/22

R8#show bgp ipv4 mvpn vrf MC route-type 3 10.2.7.2 232.0.0.5 213.7.7.7
BGP routing table entry for [3] [213:8] [10.2.7.2] [232.0.0.5] [213.7.7.7]/22,
version 79
Paths: (1 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGP
peer)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from [3] [213:7] [10.2.7.2] [232.0.0.5] [213.7.7.7]/22
(global)
    213.7.7.7 (metric 20) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Community: no-export
      Extended Community: RT:213:7
      Originator: 213.7.7.7, Cluster list: 213.12.12.12
      PMSI Attribute: Flags: 0x0, Tunnel type: 3, length 8, label: exp-null,
tunnel parameters: D507 0707 E804 0700
      rx pathid: 0, tx pathid: 0x0
```

Once the C-MRIB entry is added in the egress router's C-MRIB, a BGP Type-7 SPT join is sent back towards the source. Because this is a BGP route send to a RR, other routers might try to import it when only CSR7 should. This is solved by the "VRF Import" or "dynamic RT"; CSR7 affixes a new extended-community to its unicast VPNv4 routes that effectively says "If you have BGP joins for this source/RP, use this RT". On CSR8, we can see the dynamic RT signaled by CSR7 for the customer prefix 10.2.7.0/24 is "RT:213.7.7.7:2". Note that while this has the same characteristics as a normal RT, it is technically a separate community field when attached to the unicast VPN routes. When originating the Type-7 SPT join to CSR7 (originated by CSR6 and CSR8), this is the export RT that is applied, allowing only CSR7 to import it.

```
R8#show bgp vpnv4 unicast vrf MC 10.2.7.0/24
BGP routing table entry for 213:8:10.2.7.0/24, version 123
```

```

Paths: (1 available, best #1, table MC)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from 213:7:10.2.7.0/24 (global)
    213.7.7.7 (metric 20) (via default) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:213:7 MVPN AS:213:0.0.0.0 MVPN VRF:213.7.7.7:2
      Originator: 213.7.7.7, Cluster list: 213.12.12.12
      Connector Attribute: count=1
        type 1 len 12 value 213:7:213.7.7.7
      mpls labels in/out nlabel/7006
      rx pathid: 0, tx pathid: 0x0

R8#show bgp ipv4 mvpn vrf MC route-type 7 213:7 213 10.2.7.2 232.0.0.5
BGP routing table entry for [7][213:7][213][10.2.7.2/32][232.0.0.5/32]/22,
version 77
Paths: (1 available, best #1, table MVPNv4-BGP-Table)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (213.8.8.8)
      Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
      Extended Community: RT:213.7.7.7:2
      rx pathid: 0, tx pathid: 0x0

```

CSR8 also shows that it received an MDT advertisement (signaled via the BGP Type-3 S-PMSI route) for this specific C(S,G). It is mapped to a P-tunnel MDT data group for transport across the core.

```

R8#show ip pim mdt receive
Joined MDT-data [group/mdt number : source]  uptime/expires for VRF: MC
[232.4.7.0 : 213.7.7.7]  21:38:18/stopped

```

A quick debug on CSR1, for example, shows this BGP Type-7 route being rejected as the RT was not imported on CSR1. CSR1 has allocated a dynamic RT for its customer prefix 10.1.9.0/24, which is "213.1.1.1:1", which does not match "213.7.7.7:2".

```

R1#show bgp vpnv4 unicast vrf MC 10.1.9.0
BGP routing table entry for 213:1:10.1.9.0/24, version 664
Paths: (1 available, best #1, table MC)
  Advertised to update-groups:
    25
  Refresh Epoch 1
  Local
    0.0.0.0 (via vrf MC) from 0.0.0.0 (213.1.1.1)
      Origin incomplete, metric 0, localpref 100, weight 32768, valid,
sourced, best

```



```
Extended Community: RT:213:1 MVPN AS:213:0.0.0.0 MVPN VRF:213.1.1.1:1
mpls labels in/out 1013/nolabel(MC)
rx pathid: 0, tx pathid: 0x0
```

```
! CSR1
```

```
BGP(15): 213.12.12.12 rcvd [7][213:7][213][10.2.7.2/32][232.0.0.5/32]/22 --
DENIED due to: extended community not supported;
```

At this point, CSR7 should have added a C-MRIB entry directing traffic for (10.2.7.2, 232.0.0.5) out towards the PMSI with RPF facing towards its customer LAN. This traffic has the small 'y' as it is sending traffic to an MDT data group and a big 'G' as the C-MRIB entry was derived from a BGP Type 7 SPT join. A quick verification of the "send" MDT shows that the C(S,G) was bound to a P-tunnel MDT data group.

```
R7#show ip mroute vrf MC 232.0.0.5 10.2.7.2 | begin \
(10.2.7.2, 232.0.0.5), 21:37:00/stopped, flags: sTyG
Incoming interface: GigabitEthernet2.527, RPF nbr 0.0.0.0
Outgoing interface list:
Tunnel2, Forward/Sparse, 21:37:00/stopped
```

```
R7#show ip pim mdt send
MDT-data send list for VRF: MC
  (source, group)                                MDT-data group/num    ref_count
(10.2.7.2, 232.0.0.5)                            232.4.7.0             1
```

With all of the c-mcast signaling complete, we now verify that the P-tunnel has been built. Because the data MDTs rely on SSM, CSR6 and CSR8 should have issued P(S,G) joins for (213.7.7.7, 232.4.7.0) towards CSR7. Checking all routers, we can see the P-PIM delivery tree is built properly. The big 'Z' means the MRIB entry represents a multicast tunnel while the little 'z' represents sending data on behalf of a customer MDT data group. The egress PEs forward this traffic to the "MVRF", which is essentially a decapsulation software process.

```
R7#show ip mroute 232.4.7.0 213.7.7.7 | begin \
(213.7.7.7, 232.4.7.0), 1d01h/00:02:44, flags: sTz
Incoming interface: Loopback0, RPF nbr 0.0.0.0
Outgoing interface list:
GigabitEthernet2.574, Forward/Sparse, 21:43:47/00:02:44
GigabitEthernet2.573, Forward/Sparse, 21:43:47/00:02:44
```

```
R6#show ip mroute 232.4.7.0 213.7.7.7 | begin \
(213.7.7.7, 232.4.7.0), 1d01h/stopped, flags: sTIz
Incoming interface: GigabitEthernet2.563, RPF nbr 213.6.13.13
Outgoing interface list:
MVRF MC, Forward/Sparse, 21:43:57/00:01:01
```

```
R8#show ip mroute 232.4.7.0 213.7.7.7 | begin \
(213.7.7.7, 232.4.7.0), 1d01h/stopped, flags: sTIz
Incoming interface: GigabitEthernet2.584, RPF nbr 213.8.14.14
```

Outgoing interface list:

MVRF MC, Forward/Sparse, 21:44:00/00:00:59

We will begin sending traffic on CSR2. Tracing hop by hop, CSR7 should see packet counter increases on its C-MRIB and P-MRIB entries. This indicates traffic is being properly encapsulated and forwarded to the core. On CSR7, we check the C(S,G) and P(S,G) entries at the same time. The packet counts are identical and only a 24-byte (GRE) packet difference is observed.

```
R7#show ip mroute vrf MC 232.0.0.5 10.2.7.2 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 45, Packets received:
45
Source: 10.2.7.2/32, Forwarding: 45/1/118/0, Other: 45/0/0
```

```
R7#show ip mroute 232.4.7.0 213.7.7.7 count | begin ^Group
Group: 232.4.7.0, Source count: 1, Packets forwarded: 45, Packets received:
45
Source: 213.7.7.7/32, Forwarding: 45/1/124/0, Other: 45/0/0
```

The core routers only see the P-tunnel traffic, but their counters indicate a working flow. The 'A' flag means accept (inbound) and the 'EG' flag means egress (outbound). We can tell the flow is moving in the right direction.

```
RP/0/0/CPU0:XRv3#show mfib route 232.4.7.0 213.7.7.7 | begin 213
(213.7.7.7,232.4.7.0), Flags:
Up: 21:49:54
Last Used: 00:00:00
SW Forwarding Counts: 135/135/16740
SW Replication Counts: 135/135/16740
SW Failure Counts: 0/0/0/0/0
GigabitEthernet0/0/0/0.563 Flags: NS EG, Up:21:49:54
GigabitEthernet0/0/0/0.573 Flags: A, Up:21:49:54
```

```
RP/0/0/CPU0:XRv4#show mfib route 232.4.7.0 213.7.7.7 | begin 213
(213.7.7.7,232.4.7.0), Flags:
Up: 21:50:40
Last Used: 00:00:00
SW Forwarding Counts: 180/180/22320
SW Replication Counts: 180/180/22320
SW Failure Counts: 0/0/0/0/0
GigabitEthernet0/0/0/0.574 Flags: A, Up:21:50:40
GigabitEthernet0/0/0/0.584 Flags: NS EG, Up:21:50:40
```

The egress routers, CSR6 and CSR8, both show counters on their P(S,G) and C(S,G) entries, indicating a working multicast flow. The C-MRIB performs this counter before removing the provider-imposed GRE encapsulation since the packet sizes are the same.

```
R6#show ip mroute vrf MC 232.0.0.5 10.2.7.2 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 343, Packets received:
343
Source: 10.2.7.2/32, Forwarding: 343/1/142/1, Other: 343/0/0
```

```
R6#show ip mroute 232.4.7.0 213.7.7.7 count | begin ^Group
Group: 232.4.7.0, Source count: 1, Packets forwarded: 343, Packets received:
343
Source: 213.7.7.7/32, Forwarding: 343/1/142/1, Other: 343/0/0
```

```
R8#show ip mroute vrf MC 232.0.0.5 10.2.7.2 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 341, Packets received:
341
Source: 10.2.7.2/32, Forwarding: 341/1/142/1, Other: 341/0/0
```

```
R8#show ip mroute 232.4.7.0 213.7.7.7 count | begin ^Group
Group: 232.4.7.0, Source count: 1, Packets forwarded: 341, Packets received:
341
Source: 213.7.7.7/32, Forwarding: 341/1/142/1, Other: 341/0/0
```

A quick EPC verification on CSR8, capturing inbound facing the core and outbound facing the customer, shows the encapsulation of the packet. We can clearly see the GRE multicast arriving with the ICMP multicast being forwarded to the customer. The difference in length is exactly 24 bytes and the timestamp is identical.

```
R8#show monitor capture CAP buffer detailed
 1  142    0.167990    213.7.7.7    -> 232.4.7.0    GRE
0000:  01005E04 07000050 56A9DE77 8100CE00  ..^....PV..w....
0010:  08004500 007C0E6A 0000FE2F E2D5D507  ..E...|.j.../....
0020:  0707E804 07000000 08004500 006428CF  .....E..d(.
0030:  0000FE01 9AC00A02 0702E800 00050800  .....

 2  118    0.167990    10.2.7.2    -> 232.0.0.5    ICMP
0000:  01005E00 00050050 56A9FB1C 8100DD2  ..^....PV.....
0010:  08004500 006428CF 0000FD01 9BC00A02  ..E..d(.....
0020:  0702E800 00050800 64CA000E 017F0000  .....d.....
0030:  0000299E EE54ABCD ABCDABCD ABCDABCD  ..)..T.....
```

Last, we will examine IPv6 SSM. ASM is covered in detail in several other profiles and the complexity in the customer network isn't the primary focus of MVPN. This is interesting because despite being SSM, there is no corresponding S-PMSI mapping. This is because the C(S,G) is not matched by the IPv6 data MDT ACL. We can check the ingress PE for (2001:10:4:6::4, FF33::1), which is CSR6, for any kind of data MDT information regarding this group. We do not see any Type-3 routes and subsequently no sent MDTs.

```
R6#show bgp ipv6 mvpn vrf MC | begin Network
Network                Next Hop                Metric LocPrf Weight Path
```

```

Route Distinguisher: 213:6 (default for vrf MC)
*> [1][213:6][213.6.6.6]/12
      ::                                32768 ?
*>i [1][213:6][213.7.7.7]/12
      213.7.7.7                        0    100    0 ?
*>i [1][213:6][213.8.8.8]/12
      213.8.8.8                        0    100    0 ?
*>i [7][213:6][213][2001:10:4:6::4][FF33::1]/46
      213.7.7.7                        0    100    0 ?

```

```

R6#show ipv6 pim mdt send
[no output]

```

On the egress PEs (CSR8 for example), the only real match we have for this C(S,G) is the I-PMSI, signaled by the BGP Type-1 route. Looking at the details, we see this is a type 5 tunnel, which indicates PIM-Bidir. The root of the tree is XRv4, and the group address is 238.255.0.1.

```

R8#show bgp ipv6 mvpn vrf MC route-type 1 213.7.7.7
BGP routing table entry for [1][213:8][213.7.7.7]/12, version 31
Paths: (1 available, best #1, table MVPNV6-BGP-Table, not advertised to EBGp
peer)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from [1][213:7][213.7.7.7]/12 (global)
    213.7.7.7 (metric 20) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Community: no-export
      Extended Community: RT:213:7
      Originator: 213.7.7.7, Cluster list: 213.12.12.12
      PMSI Attribute: Flags: 0x0, Tunnel type: 5, length 8, label: exp-null,
tunnel parameters: D507 0707 EEEF 0001
      rx pathid: 0, tx pathid: 0x0

```

The corresponding C-MRIB entry doesn't show a big 'Y' flag to indicate it is receiving traffic for a data MDT, but the little 'g' indicates that it sent a BGP c-mcast route (Type 6 or 7, but in this case, 7) to signal interest in this group. We can confirm these flags by checking the data MDTs received (none) and the originated BGP Type-7 route.

```

R8#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 | begin \
(2001:10:4:6::4, FF33::1), 1d02h/never, flags: sTIg
  Incoming interface: Tunnel3
  RPF nbr: ::FFFF:213.6.6.6
  Immediate Outgoing interface list:
    GigabitEthernet2.538, Forward, 1d02h/never

```

```

R8#show ipv6 pim mdt receive
[no output]

```

```

R8#show bgp ipv6 mvpn rd 213:6 neighbors 213.12.12.12 advertised-routes |
begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 213:6
*> [7] [213:6] [213] [2001:10:4:6::4] [FF33::1]/46
      ::
      32768 ?

```

A quick check on CSR6 confirms that this Type-7 route was imported. Again, this is because of the dynamic RT that CSR8 attached to the route targeting CSR6. This is derived from the VPNv6 route originated by CSR6 for the prefix 2001:10:4:6::/64. We confirm the dynamic RT added by CSR7 matches what CSR6 advertised in its VPNv6 route.

```

R6#show bgp ipv6 mvpn vrf MC route-type 7 213:6 213 2001:10:4:6::4 FF33::1
BGP routing table entry for [7] [213:6] [213] [2001:10:4:6::4] [FF33::1]/46,
version 39
Paths: (1 available, best #1, table MVPNV6-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    213.7.7.7 (metric 20) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:213.6.6.6:2
      Originator: 213.7.7.7, Cluster list: 213.12.12.12
      rx pathid: 0, tx pathid: 0x0

```

```

R6#show bgp vpnv6 unicast vrf MC 2001:10:4:6::/64
BGP routing table entry for [213:6]2001:10:4:6::/64, version 83
Paths: (1 available, best #1, table MC)
  Advertised to update-groups:
    5
  Refresh Epoch 1
  Local
    :: (via vrf MC) from 0.0.0.0 (213.6.6.6)
      Origin incomplete, metric 0, localpref 100, weight 32768, valid,
sourced, best
      Extended Community: RT:213:6 MVPN AS:213:0.0.0.0 MVPN VRF:213.6.6.6:2
      mpls labels in/out 6007/nolabel(MC)
      rx pathid: 0, tx pathid: 0x0

```

Knowing that the default MDT is built, we will not verify that. When CSR4 starts sending traffic, we verify that CSR6 is encapsulating those packets using the P-tunnel default MDT group. This shows that the C-MRIB is forwarding the traffic towards the PMSI correctly.

```

R6#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count
Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second
Other counts:      Total/RPF failed/Other drops(OIF-null, rate-limit etc)
VRF MC

```

```

53 routes, 6 (*,G)s, 46 (*,G/m)s
Group: FF33::1
Source: 2001:10:4:6::4,
SW Forwarding: 0/0/0/0, Other: 0/0/0
HW Forwarding: 119/1/118/0, Other: 0/0/0
Totals - Source count: 1, Packet count: 119

```

We can see this traffic enters CSR1 by virtue of C(S,G) presence there. The packet counters show that all packets have been discarded due to RPF failure. This is because CSR1 has no route to the source as the unicast RT was never imported in the VPNv6 topology. In an optimized network, this traffic would not be transported via the I-PMSI in the first place, which means CSR1 would not have this C-mcast state.

```

R1#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count
Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second
Other counts:      Total/RPF failed/Other drops(OIF-null, rate-limit etc)
VRF MC
53 routes, 6 (*,G)s, 46 (*,G/m)s
Group: FF33::1
Source: 2001:10:4:6::4,
SW Forwarding: 0/0/0/0, Other: 0/0/0
HW Forwarding: 0/0/0/0, Other: 291/291/0
Totals - Source count: 1, Packet count: 0

```

```

R1#show ipv6 rpf vrf MC 2001:10:4:6::4
RPF information for 2001:10:4:6::4
Lookup failed, no route exists

```

Performing the same check on CSR8, we see packets being successfully forwarded since RPF is valid.

```

R8#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count
Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second
Other counts:      Total/RPF failed/Other drops(OIF-null, rate-limit etc)
VRF MC
55 routes, 7 (*,G)s, 47 (*,G/m)s
Group: FF33::1
Source: 2001:10:4:6::4,
SW Forwarding: 0/0/0/0, Other: 0/0/0
HW Forwarding: 636/1/142/1, Other: 0/0/0
Totals - Source count: 1, Packet count: 636

```

```

Groups: 1, 1.00 average sources per group
R8#show ipv6 rpf vrf MC 2001:10:4:6::4
RPF information for 2001:10:4:6::4
RPF interface: Tunnel3
RPF neighbor: ::FFFF:213.6.6.6
RPF route/mask: 2001:10:4:6::/64
RPF type: Unicast
RPF recursion count: 0

```

```
Metric preference: 200
Metric: 0
```

A final check involves EPC on CSR8 to capture incoming P-tunnel traffic along the default MDT and customer IPv6 traffic after decapsulation. For brevity, we can see the headers and be reasonably sure the design is working. The IPv4 GRE packets along the default MDT are decapsulated (24 bytes removed) and the IPv6-ICMP packets are forwarded to the customer.

```
R8#show monitor capture CAP buffer detailed
```

```
0 142 0.000000 213.6.6.6 -> 238.255.0.1 GRE
0000: 01005E7F 00010050 56A9DE77 8100CE00 ..^....PV..w....
0010: 08004500 007C2396 0000FD2F CFAFD506 ..E...|#..../....
0020: 0606EEFF 00010000 86DD6000 0000003C .....`.....<
0030: 3A3F2001 00100004 00060000 00000000 :? .....

1 118 0.000000 2001::*:0004 -> FF33::*:0001 IPv6-ICMP
0000: 33330000 00010050 56A9FB1C 8100DD2 33.....PV.....
0010: 86DD6000 0000003C 3A3F2001 00100004 ..`.....<:? .....
0020: 00060000 00000000 0004FF33 00000000 .....3.....
0030: 00000000 00000000 00018000 F1081E0B .....

```

Additional Reading – Reference configurations “mvpn-11”

28.10 Profile 12: Default MDT – MLDP – P2MP – BGP-AD – BGP C-mcast Signaling

For this test, begin with the MVPN Profile 9 configuration where all of the MVPN BGP AD is set up already, plus the MP2MP/P2MP trees are built. We need to do two things: remove the MP2MP tree, and enable BGP C-multicast signaling. Both are very straightforward. **In XE, be sure to remove all of the MP2MP tree configuration for both IPv4/IPv6 before enabling P2MP. When you remove the MDT default, all the MDT data configuration also disappears.** We need to reapply our ACLs on XE only.

```
! Enable BGP c-mcast signaling and mLDP P2MP default trees in XE PEs
vrf definition MC
address-family ipv4
no mdt default mpls mldp 213.14.14.14
mdt default mpls mldp p2mp
mdt overlay use-bgp
mdt data mpls mldp 10 list ACL_DATA_MDT immediate-switch
address-family ipv6
no mdt default mpls mldp 213.14.14.14
mdt default mpls mldp p2mp
mdt overlay use-bgp
mdt data mpls mldp 10 list ACL_DATA_MDT_IPV6 immediate-switch

! Enable BGP c-mcast signaling and mLDP P2MP default trees in XRv2
multicast-routing
vrf MC
```

```

address-family ipv4
  no mdt default mldp ipv4 213.14.14.14
  mdt default mldp p2mp
address-family ipv6
  no mdt default mldp ipv4 213.14.14.14
  mdt default mldp p2mp

router pim
vrf MC
  address-family ipv4
    mdt c-multicast-routing bgp
  address-family ipv6
    mdt c-multicast-routing bgp

```

This profile helps demonstrate the value of using BGP-AD for an entirely dynamic solution, as well as using BGP for mcast signaling.. Because we did not specify a bidirectional MP2MP root, some replacement for a “default MDT” still needs to be built. This would carry key information like C-PIM BSR messages, for example. Without BGP-AD, this would not be possible, since the endpoints of the particular MVPN instance are unknown. In this configuration, every PE is the root of its own P2MP tree, and all other nodes within the MVPN instance join as leaves to every other tree. Compared to MP2MP trees, this introduces significantly more state into all routers (P and PE) within the SP network. The benefit is a fully dynamic construct with semi-optimal transmission paths by default. This has the same effect as the MP2MP “emulated LAN” in terms of C-signaling but does so with better SP core forwarding.

Taking a quick look at CSR6, we see several P2MP trees. Without actually proving/verifying anything, we can make some educated guesses based on this output. The first 6 entries relate to the IPv4 MDTs since they include all of the PEs within the MVPN (yellow). The last 3 entries are related to the IPv6 MDTs since they only include the “south-side” routers (green). A quick look at CSR1 helps confirm this estimate as CSR1 sees all of the IPv4 MDTs but only the “north-side” routers for IPv6 MDT.

```
R6#show mpls mldp database summary
```

LSM ID	Type	Root	Decoded Opaque Value	Client
Cnt.				
F	P2MP	213.12.12.12	[gid 1 (0x00000001)]	1
E	P2MP	213.1.1.1	[gid 65536 (0x00010000)]	1
D	P2MP	213.5.5.5	[gid 65536 (0x00010000)]	1
8	P2MP	213.6.6.6	[gid 65536 (0x00010000)]	2
9	P2MP	213.7.7.7	[gid 65536 (0x00010000)]	1
B	P2MP	213.8.8.8	[gid 65536 (0x00010000)]	1
7	P2MP	213.6.6.6	[gid 131072 (0x00020000)]	2
A	P2MP	213.7.7.7	[gid 131072 (0x00020000)]	1
C	P2MP	213.8.8.8	[gid 131072 (0x00020000)]	1

```
R1#show mpls mldp database summary
```

LSM ID	Type	Root	Decoded Opaque Value	Client
Cnt.				

A	P2MP	213.12.12.12	[gid 1 (0x00000001)]	1
3	P2MP	213.1.1.1	[gid 65536 (0x00010000)]	2
4	P2MP	213.5.5.5	[gid 65536 (0x00010000)]	1
5	P2MP	213.6.6.6	[gid 65536 (0x00010000)]	1
6	P2MP	213.7.7.7	[gid 65536 (0x00010000)]	1
7	P2MP	213.8.8.8	[gid 65536 (0x00010000)]	1
8	P2MP	213.1.1.1	[gid 131072 (0x00020000)]	2
9	P2MP	213.5.5.5	[gid 131072 (0x00020000)]	1
B	P2MP	213.12.12.12	[gid 262145 (0x00040001)]	1

One interesting thing to notice is that opaque-type is no longer MDT; it is global-ID, or GID. This is a different type of opaque value (specifically type 0x01 as we will see in OAM), but mLDP doesn't care about this opaque value. It just gets transported. Notice that the GIDs are the same for the entire IPv4 and IPv6 MVPNs, with the exception of XRv2. Its GID numbering scheme starts from 1 while the others to do. It would be valuable to understand where this GID came from, and how is it possible that all the routers agreed on it. I tried to find a way to set this on IOS XE and XR, but was unable to. This value is encoded in the BGP Type-1 I-PMSI message in hexadecimal. Looking at the IPv4 and IPv6 Type-1 routes on CSR1, we can see the values of 65536 (0x10000) and 131072 (0x20000) in the tunnel information for IPv4 and IPv6 Type-1 AD routes, respectively.

```
R6#show bgp ipv4 mvpn vrf MC route-type 1 213.8.8.8
BGP routing table entry for [1][213:6][213.8.8.8]/12, version 117
Paths: (1 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGp
peer)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from [1][213:8][213.8.8.8]/12 (global)
    213.8.8.8 (metric 30) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Community: no-export
      Extended Community: RT:213:8
      Originator: 213.8.8.8, Cluster list: 213.12.12.12
      PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 17, label: exp-null,
tunnel parameters: 0600 0104 D508 0808 0007 0100 0400 0100 00
      rx pathid: 0, tx pathid: 0x0
```

```
R6#show bgp ipv6 mvpn vrf MC route-type 1 213.8.8.8
BGP routing table entry for [1][213:6][213.8.8.8]/12, version 109
Paths: (1 available, best #1, table MVPNV6-BGP-Table, not advertised to EBGp
peer)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from [1][213:8][213.8.8.8]/12 (global)
    213.8.8.8 (metric 30) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Community: no-export
      Extended Community: RT:213:8
      Originator: 213.8.8.8, Cluster list: 213.12.12.12
```

```

PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 17, label: exp-null,
tunnel parameters: 0600 0104 D508 0808 0007 0100 0400 0200 00
rx pathid: 0, tx pathid: 0x0

```

Let's trace a few mLDP P2MP trees to make sure things look correct. As an example, all 6 PEs should be joined to CSR6's delivery tree. Since mLDP P2MP trees are receiver-driven, we will trace it back from the leaves first. Rather than use the GID for the show commands (which shows N-1 trees, generally), I will use the LSP ID. CSR6 instructs XRv4 to use label 1011 when delivering traffic downstream.

```

R1#show mpls mldp database summary | include 213.6.6.6
5          P2MP      213.6.6.6          [gid 65536 (0x00010000)]      1

```

```

R1#show mpls mldp database id 5 | section Upstream
Upstream client(s) :
  213.14.14.14:0    [Active]
    Expires         : Never          Path Set ID   : 5
    Out Label (U)   : None           Interface     : GigabitEthernet2.514*
    Local Label (D) : 1011           Next Hop      : 213.1.14.14

```

Because we expect CSR8 to route through XRv4 also, we will check its downstream label mapping. Notice that CSR8 has two trees rooted at CSR6, but the second is IPv6. This is not present on CSR1 because the north/south routers do not have IPv6 connectivity over the VPN. CSR8 uses label 8003.

```

R8#show mpls mldp database summary | include 213.6.6.6
8          P2MP      213.6.6.6          [gid 65536 (0x00010000)]      1
B          P2MP      213.6.6.6          [gid 131072 (0x00020000)]      1

```

```

R8#show mpls mldp database id 8 | section Upstream
Upstream client(s) :
  213.14.14.14:0    [Active]
    Expires         : Never          Path Set ID   : 8
    Out Label (U)   : None           Interface     : GigabitEthernet2.584*
    Local Label (D) : 8003           Next Hop      : 213.8.14.14

```

XRv4 also sees two tunnels rooted in CSR6. We check the IPv4 one and see both CSR1 and CSR8 as downstream clients with the correct label mappings (1011 and 8003). XRv4 advertises label 94013 towards XRv3 who is upstream towards CSR6. The numbers "1 2" highlighted in the summary output shows the number of upstream and downstream clients, respectively.

```

RP/0/0/CPU0:XRv4#show mpls mldp database brief | include 213.6.6.6
0x000004  P2MP      213.6.6.6          1 2    [global-id 65536]
0x000006  P2MP      213.6.6.6          1 1    [global-id 131072]

```

```

RP/0/0/CPU0:XRv4#show mpls mldp database 0x000004
mLDP database
LSM-ID: 0x000004   Type: P2MP   Uptime: 00:43:23

```

```

FEC Root          : 213.6.6.6
Opaque decoded    : [global-id 65536]
Upstream neighbor(s) :
  213.13.13.13:0 [Active] Uptime: 00:43:23
    Local Label (D) : 94013
Downstream client(s):
  LDP 213.1.1.1:0    Uptime: 00:43:17
    Next Hop        : 213.1.14.1
    Interface       : GigabitEthernet0/0/0/0.514
    Remote label (D) : 1011
  LDP 213.8.8.8:0    Uptime: 00:43:23
    Next Hop        : 213.8.14.8
    Interface       : GigabitEthernet0/0/0/0.584
    Remote label (D) : 8003

```

Before checking XRv3, we will check CSR5, CSR7, and XRv2. We expect their shortest paths to be via XRv3, so they should all advertise downstream labels to XRv3, similar to how CSR1 and CSR8 did so with XRv4. We see labels 5010, 7006, and 92017 advertised to XRv3. These should be downstream from XRv3's perspective, along with XRv4.

```

R5#show mpls mldp database summary | include 213.6.6.6
4          P2MP      213.6.6.6          [gid 65536 (0x00010000)]      1

```

```

R5#show mpls mldp database id 4 | section Upstream
Upstream client(s) :
  213.13.13.13:0    [Active]
    Expires          : Never          Path Set ID   : 4
    Out Label (U)    : None           Interface     : GigabitEthernet2.553*
    Local Label (D) : 5010           Next Hop      : 213.5.13.13

```

```

R7#show mpls mldp database summary | include 213.6.6.6
8          P2MP      213.6.6.6          [gid 65536 (0x00010000)]      1
A          P2MP      213.6.6.6          [gid 131072 (0x00020000)]      1

```

```

R7#show mpls mldp database id 8 | section Upstream
Upstream client(s) :
  213.13.13.13:0    [Active]
    Expires          : Never          Path Set ID   : 8
    Out Label (U)    : None           Interface     : GigabitEthernet2.573*
    Local Label (D) : 7006           Next Hop      : 213.7.13.13

```

```

RP/0/0/CPU0:XRv2#show mpls mldp database brief | include 213.6.6.6
0x00008      P2MP      213.6.6.6          1 1      [global-id 65536]

```

```

RP/0/0/CPU0:XRv2#show mpls mldp database 0x00008 | section Upstream
RP/0/0/CPU0:XRv2#show mpls mldp database 0x00008 | begin Upstream
Upstream neighbor(s) :
  213.13.13.13:0 [Active] Uptime: 00:50:07

```

```
Local Label (D) : 92017
[snip]
```

As expected, XRv3 has 4 downstream clients for this P2MP tree and one upstream client towards CSR6. Notice the “1 4” indicates this in brief form. A quick validation of the remote downstream labels shows our trace is correct so far. XRv3 advertises label 93000 towards CSR6.

```
RP/0/0/CPU0:XRv3#show mpls mldp database brief | in 213.6.6.6
0x00004  P2MP      213.6.6.6      1 4      [global-id 65536]
0x00005  P2MP      213.6.6.6      1 2      [global-id 131072]
```

```
RP/0/0/CPU0:XRv3#show mpls mldp database 0x00004
mLDP database
LSM-ID: 0x00004   Type: P2MP   Uptime: 00:49:40
FEC Root          : 213.6.6.6
Opaque decoded    : [global-id 65536]
Upstream neighbor(s) :
  213.6.6.6:0 [Active] Uptime: 00:49:40
    Local Label (D) : 93000
Downstream client(s):
  LDP 213.5.5.5:0   Uptime: 00:49:31
    Next Hop        : 213.5.13.5
    Interface       : GigabitEthernet0/0/0/0.553
    Remote label (D) : 5010
  LDP 213.7.7.7:0   Uptime: 00:49:40
    Next Hop        : 213.7.13.7
    Interface       : GigabitEthernet0/0/0/0.573
    Remote label (D) : 7006
  LDP 213.12.12.12:0 Uptime: 00:49:21
    Next Hop        : 213.12.13.12
    Interface       : GigabitEthernet0/0/0/0.523
    Remote label (D) : 92017
  LDP 213.14.14.14:0 Uptime: 00:49:35
    Next Hop        : 213.13.14.14
    Interface       : GigabitEthernet0/0/0/0.534
    Remote label (D) : 94013
```

On CSR6, we see the label 93000 from XRv3 with no upstream (local) label. P2MP trees are always unidirectional from the root towards the leaves, which is why there is a “full mesh” of P2MP trees in this profile.

```
R6#show mpls mldp database summary | include 213.6.6.
6
8          P2MP      213.6.6.6      [gid 65536 (0x00010000)]      2
7          P2MP      213.6.6.6      [gid 131072 (0x00020000)]      2
```

```
R6#show mpls mldp database id 8 | section Replic
Replication client(s):
```

```
MDT (VRF MC)
  Uptime      : 00:57:04      Path Set ID : None
  Interface    : Lspvif0
213.13.13.13:0
  Uptime      : 00:30:43      Path Set ID : None
  Out label (D) : 93000        Interface    : GigabitEthernet2.563*
  Local label (U) : None       Next Hop     : 213.6.13.13
```

At this point, we will begin testing multicast traffic. The details of BGP c-mcast signaling are described in detail in Profile 13. It is tested here but documented in less detail. First, we ensure the Type-6 C(*,G) join made its way to the C-RP. Each router will have originated this provided there was a client in the customer network interested in a C-group. Checking XRv2 first, we can see three interested PEs: CSR1, CSR6, and CSR8. CSR5 sends a C(*,G) PIM join towards the C-RP (CSR10) having installed it by receiving a BGP Type-6 route (big 'G').

```
RP/0/0/CPU0:XRv2#show bgp ipv4 mvpn rd 213:5 route-type 6 | begin Network
  Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 213:5
* i [6] [213:5] [213] [32] [10.5.10.10] [32] [225.0.0.1] /184
      213.1.1.1              0      100      0 ?
*>i      213.6.6.6              0      100      0 ?
* i      213.8.8.8              0      100      0 ?
```

```
R5#show ip mroute vrf MC 225.0.0.1 | section ^\(\*
(*, 225.0.0.1), 1d00h/stopped, RP 10.5.10.10, flags: SG
Incoming interface: GigabitEthernet2.550, RPF nbr 10.5.10.10
Outgoing interface list:
  Lspvif0, Forward/Sparse, 01:39:07/stopped
```

CSR2 will begin sending IPv4 ASM traffic to group 225.0.0.1. CSR7 should send a register message to the C-RP, CSR10 who will create C(S,G), state and start the C(S,G) join process towards the C-PIM DR, CSR7.

```
R2#ping 225.0.0.1 repeat 10000
```

```
! CSR7
PIM(1): Check RP 10.5.10.10 into the (*, 225.0.0.1) entry
PIM(1): Building Triggered (*,G) Join / (S,G,RP-bit) Prune message for
225.0.0.1
PIM(1): Adding register encap tunnel (Tunnel2) as forwarding interface of
(10.2.7.2, 225.0.0.1).
```

```
! CSR10
PIM(0): Received v2 Register on GigabitEthernet2.550 from 10.2.7.7
for 10.2.7.2, group 225.0.0.1
PIM(0): Adding register decap tunnel (Tunnel1) as accepting interface of
(10.2.7.2, 225.0.0.1).
PIM(0): Insert (10.2.7.2,225.0.0.1) join in nbr 10.5.10.5's queue
```

```
PIM(0): Building Join/Prune packet for nbr 10.5.10.5
PIM(0): Adding v2 (10.2.7.2/32, 225.0.0.1), S-bit Join
PIM(0): Send v2 join/prune to 10.5.10.5 (GigabitEthernet2.550)
```

All of the PEs with interested clients (egress PEs) generate BGP Type-7 SPT joins at this point. CSR5 does it because it actually received a C(S,G) prune from CSR10. The others do it because they received traffic down the C(*,G) shared tree and assume that they should join this SPT. The debugs from CSR5 and CSR6 both show local generation of the Type-7 route. Checking XRv2, we can see 4 copies of it; one from the PE connected to the RP and three from remote clients. If XRv were working, we would have seen one from XRv2. In the interest of brevity, the highlighted RD at the top of this output varies between routes, so this exact output is from CSR5. The RD of the route originated is CSR7, which is the PE towards the C-source.

```
! CSR5, CSR6, CSR1, CSR8
PIM(1): Upstream mode for (10.2.7.2, 225.0.0.1) changed from 1 to 2
BGP[15] MVPN: add c-route, type 7, bs len 0 asn=0, rd=213:5,
  source=10.2.7.2/4,
  group=225.0.0.1/4,
  nexthop=213.7.7.7,
  len left = 0
BGP[14] MVPN umh lookup: vrfid 1, source 10.2.7.2
BGP[4] MVPN umh lookup: vrfid 1, source 10.2.7.2, net 213:5:10.2.7.0/24,
213:5:10.2.7.0/24 with matching nexthop 213.7.7.7, remote-rd [213:7]:
0x9:213:0.0.0.0, 0x10B:213.7.7.7:1,
BGP: MVPN(15) create local route
[7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22
BGP(15): nettable_walker [7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22 route
sourced locally
BGP(15): 213.12.12.12 NEXT_HOP self is set for sourced RT Filter for net
[7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22
BGP(15): (base) 213.12.12.12 send UPDATE (format)
[7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22, next 213.5.5.5, metric 0, path
Local, extended community RT:213.7.7.7:1
```

```
RP/0/0/CPU0:XRv2# show bgp ipv4 mvpn rd 213:7
[7][213:7][213][32][10.2.7.2][32][225.0.0.1]/184 brief | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 213:7
* i [7][213:7][213][32][10.2.7.2][32][225.0.0.1]/184
                                213.1.1.1          0      100          0 ?
*>i                                213.5.5.5          0      100          0 ?
* i                                213.6.6.6          0      100          0 ?
* i                                213.8.8.8          0      100          0 ?
```

I've also highlighted the dynamic RT of 213.7.7.7:1. This was derived from the extended community carried within CSR7's VPNv4 route to the C-source. We can verify that below, and all routers use this in their Type-7 joins to target CSR7.

```

R5#show bgp vpnv4 unicast vrf MC 10.2.7.0/24
BGP routing table entry for 213:5:10.2.7.0/24, version 69
Paths: (1 available, best #1, table MC)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from 213:7:10.2.7.0/24 (global)
    213.7.7.7 (metric 20) (via default) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:213:7 MVPN AS:213:0.0.0.0 MVPN VRF:213.7.7.7:1
      Originator: 213.7.7.7, Cluster list: 213.12.12.12
      mpls labels in/out nolabel/7000
      rx pathid: 0, tx pathid: 0x0

```

CSR7 also originates a Type-5 Source Active AD message to notify others of an active multicast sender. Notice that the RT is a normal RT since this is an AD route, not a BGP c-mcast signaling route.

```

! CSR7
PIM(1): Add Lspvif0/0.0.0.0 to (10.2.7.2, 225.0.0.1), Forward state, by BGP
SG Join
BGP(15): nettable_walker [5][213:7][10.2.7.2][225.0.0.1]/18 route sourced
locally
BGP(15): delete RIB route [5][213:7][10.2.7.2][225.0.0.1]/18
BGP(15): 213.12.12.12 NEXT_HOP self is set for sourced RT Filter for net
[5][213:7][10.2.7.2][225.0.0.1]/18
BGP(15): (base) 213.12.12.12 send UPDATE (format)
[5][213:7][10.2.7.2][225.0.0.1]/18, next 213.7.7.7, metric 0, path Local,
extended community RT:213:7

```

CSR7 also receives the Type-7 SPT join, which is technically the best-path from XRv2. They all contain the same information so it doesn't matter which one the RR picks as best. In this case, CSR5's route was best. Upon receipt, once the BGP route is processed and installed, PIM takes over and enters the C(S,G) state into the PIM topology (MRIB). This means the C-SPT has been built back to the source so multicast can now travel our mLD P2MP tree. Technically this is not S-PMSI since no Type-3 S-PMSI AD routes were generated, and we did not configure "Data MDT" support for this group. The I-PMSI is still used, but the C-SPT is still invoked on the ingress PE.

```

! CSR7
BGP(15): 213.12.12.12 rcvd UPDATE w/ attr: nexthop 213.5.5.5, origin ?,
localpref 100, metric 0, originator 213.5.5.5, clusterlist 213.12.12.12,
extended community RT:213.7.7.7:1
BGP(15): 213.12.12.12 rcvd [7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22
BGP(15): skip vrf default table RIB route
[7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22
BGP(15): add RIB route [7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22
PIM(1): Join-list: (10.2.7.2/32, 225.0.0.1), S-bit set, BGP C-Route
PIM(1): MDT next_hop change from: 0 to 7 for (10.2.7.2, 225.0.0.1) Lspvif0

```

```
PIM(1): Add Lspvif0/0.0.0.0 to (10.2.7.2, 225.0.0.1), Forward state, by BGP  
SG Join
```

CSR10 sends a register-stop and prunes itself from the C-SPT since it is not in the transit path. CSR7 receives the register stop and tears down the encapsulation tunnel used for sending the register messages.

```
! CSR10  
PIM(0): Received v2 Register on GigabitEthernet2.550 from 10.2.7.7  
      for 10.2.7.2, group 225.0.0.1  
PIM(0): Send v2 Register-Stop to 10.2.7.7 for 10.2.7.2, group 225.0.0.1  
PIM(0): Insert (10.2.7.2,225.0.0.1) prune in nbr 10.5.10.5's queue  
PIM(0): Building Join/Prune packet for nbr 10.5.10.5  
PIM(0): Adding v2 (10.2.7.2/32, 225.0.0.1), S-bit Prune  
PIM(0): Send v2 join/prune to 10.5.10.5 (GigabitEthernet2.550)  
  
! CSR7  
PIM(1): Received v2 Register-Stop on GigabitEthernet2.574 from 10.5.10.10  
PIM(1): for source 10.2.7.2, group 225.0.0.1  
PIM(1): Removing register encap tunnel (Tunnel2) as forwarding interface of  
(10.2.7.2, 225.0.0.1).  
PIM(1): Clear Registering flag to 10.5.10.10 for (10.2.7.2/32, 225.0.0.1)
```

Before verifying mLDp operation, we check the MRIB state and counters on the ingress PE. CSR7 shows traffic entering from the customer LAN and being forwarded out the PMSI. The big 'G' means the entry was installed due to receiving a BGP Type-7 SPT join, and the little 'q' means that router originated a Type-5 SA message for this C(S,G).

```
R7#show ip mroute vrf MC 225.0.0.1 10.2.7.2 | begin \  
(10.2.7.2, 225.0.0.1), 00:28:50/00:02:32, flags: FTGq  
  Incoming interface: GigabitEthernet2.527, RPF nbr 0.0.0.0  
  Outgoing interface list:  
    Lspvif0, Forward/Sparse, 00:28:49/stopped  
  
R7#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | section ^Group  
Group: 225.0.0.1, Source count: 1, Packets forwarded: 861, Packets received:  
862  
  Source: 10.2.7.2/32, Forwarding: 861/0/117/0, Other: 862/1/0
```

The egress PEs should show incoming traffic on the PMSI and outgoing on their customer LANs. Their flags show a little 'g' indicating they sourced a BGP Type-7 SPT join and a big 'Q' indicating they received a Type-5 Source Active.

```
R8#show ip mroute vrf MC 225.0.0.1 10.2.7.2 | begin \  
(10.2.7.2, 225.0.0.1), 00:31:04/00:01:43, flags: JTgQ  
  Incoming interface: Lspvif0, RPF nbr 213.7.7.7  
  Outgoing interface list:
```


GigabitEthernet2.538, Forward/Sparse, 00:31:04/00:02:52

```
R8#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | section ^Group
Group: 225.0.0.1, Source count: 1, Packets forwarded: 928, Packets received:
928
```

Source: 10.2.7.2/32, Forwarding: 927/0/122/0, Other: 927/0/0

```
R1#show ip mroute vrf MC 225.0.0.1 10.2.7.2 | begin \
(10.2.7.2, 225.0.0.1), 00:30:59/00:02:42, flags: JTgQ
Incoming interface: Lspvif0, RPF nbr 213.7.7.7
Outgoing interface list:
GigabitEthernet2.519, Forward/Sparse, 00:30:59/00:02:44
```

```
R1#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | section ^Group
Group: 225.0.0.1, Source count: 1, Packets forwarded: 929, Packets received:
929
```

Source: 10.2.7.2/32, Forwarding: 928/0/122/0, Other: 928/0/0

```
R6#show ip mroute vrf MC 225.0.0.1 10.2.7.2 | begin \
(10.2.7.2, 225.0.0.1), 00:30:48/00:01:56, flags: JTgQ
Incoming interface: Lspvif0, RPF nbr 213.7.7.7
Outgoing interface list:
GigabitEthernet2.546, Forward/Sparse, 00:30:48/00:02:23
```

```
R6#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | section ^Group
Group: 225.0.0.1, Source count: 1, Packets forwarded: 921, Packets received:
921
```

Source: 10.2.7.2/32, Forwarding: 920/0/122/0, Other: 920/0/0

The PE facing the C-RP has interesting output. Since only two packets were sent in native multicast to the RP, the counter stops incrementing as the C(S,G) is pruned. This is the correct operation. The route was still installed from having sent a BGP Type-7 route (little 'g'), but the Type-5 SA is not bound to this C(S,G) since CSR5 did not join the C-SPT. The "other drops" column is incrementing as the OIL is null.

```
R5#show ip mroute vrf MC 225.0.0.1 10.2.7.2 | begin \
(10.2.7.2, 225.0.0.1), 00:30:55/00:02:08, flags: PTXg
Incoming interface: Lspvif0, RPF nbr 213.7.7.7
Outgoing interface list: Null
```

```
R5#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | section ^Group
Group: 225.0.0.1, Source count: 1, Packets forwarded: 3, Packets received:
928
```

Source: 10.2.7.2/32, Forwarding: 2/0/122/0, Other: 927/1/924

Since the source is behind CSR7, we know that CSR7 is the root of the mLDP P2MP tree. I perform a quick EPC on CSR7 outbound towards XRv3 to ensure the proper label bindings are used. CSR7 should

use label 93011 (0x16B53) to send MVPN traffic for VRF MC downstream towards XRv3. The packet capture shows this, along with the source/destination IP addresses highlighted.

```
R7#show mpls mldp database summary | include 213.7.7.7
7          P2MP      213.7.7.7      [gid 65536 (0x00010000)]      3
9          P2MP      213.7.7.7      [gid 131072 (0x00020000)]     3

R7#show mpls mldp database id 7 | section Replic
Replication client(s):
MDT (VRF MC)
  Uptime      : 01:42:45      Path Set ID   : None
  Interface   : Lspvif0
213.13.13.13:0
  Uptime      : 01:42:45      Path Set ID   : None
  Out label (D) : 93011      Interface     : GigabitEthernet2.573*
  Local label (U): None      Next Hop      : 213.7.13.13
213.14.14.14:0
  Uptime      : 01:42:40      Path Set ID   : None
  Out label (D) : 94014      Interface     : GigabitEthernet2.574*
  Local label (U): None      Next Hop      : 213.7.14.14
```

```
R7#show monitor capture CAP buffer detailed
 5 122 83460.123986 00:50:56:A9:EA:77 -> 00:50:56:A9:DB:37 MPLS unicast
0000: 005056A9 DB370050 56A9EA77 81000DF5 .PV..7.PV..w....
0010: 884716B5 31FE4500 006404EE 0000FE01 .G..1.E..d.....
0020: C5A50A02 0702E100 00010800 080A0000 .....}k.....
0030: 04EE0000 0000057D 6BD5ABCD ABCDABCD .....}k.....
```

Since testing ASM is much more involved and the concepts are identical between IPv4 and IPv6, we will quickly test IPv6 SSM only. Our baseline configuration does not signal S-PMSI for our IPv6 SSM group FF33::1, but we will modify the ACLs on our PEs to account for it. This deviation allows us to add some variety to the testing.

```
! Modification to XE/XR PE routers (same syntax)
ipv6 access-list ACL_DATA_MDT_IPV6
 permit ipv6 any host FF33::1
```

Adding this to ACL entry CSR6 (ingress PE), with BGP IPv6 MVPN debugging enabled, shows the immediate creation of the Type-3 S-PMSI route. Technically, the data MDT configurations only need to be added to the ingress PE connected to sources, but I add it everywhere for completeness.

```
! CSR6
BGP(16): nettable_walker [3][213:6][2001:10:4:6::4][FF33::1][213.6.6.6]/46
route sourced locally
BGP(16): (base) 213.12.12.12 send UPDATE (format)
[3][213:6][2001:10:4:6::4][FF33::1][213.6.6.6]/46, next 213.6.6.6, metric 0,
path Local, extended community RT:213:6
```

As an interesting test, if you delete the ACL entry, the route is immediately withdraw and downstream PEs will switch over to using the default MDT (I-PMSI) again. The lack of S-PMSI advertisement by the “root” of the P2MP, in the case of mLDP, means that there is no selective behavior for this C(S,G). The ACL is corrected and the entry is re-added before continuing.

```
R6(config-ipv6-acl)#no permit ipv6 any host FF33::1
```

```
BGP: MVPN(16) deleting the local route
[3][213:6][2001:10:4:6::4][FF33::1][213.6.6.6]/46
BGP(16): no valid path for [3][213:6][2001:10:4:6::4][FF33::1][213.6.6.6]/46
BGP(16): nettable_walker [3][213:6][2001:10:4:6::4][FF33::1][213.6.6.6]/46 no
best path
BGP(16): delete RIB route [3][213:6][2001:10:4:6::4][FF33::1][213.6.6.6]/46
BGP(16): (base) 213.12.12.12 send unreachable (format)
[3][213:6][2001:10:4:6::4][FF33::1][213.6.6.6]/46
BGP(16): 213.12.12.12 rcv UPDATE about
[3][213:6][2001:10:4:6::4][FF33::1][213.6.6.6]/46 -- withdrawn
```

Looking at the route from XRv2’s perspective, it was received from CSR6 and advertised to the other PEs in the update-group. With an RT of 213:6, we know that CSR1 and CSR5 will not import this Type-3 route, but CSR7 and CSR8 should. There are two other key components of this Type-3 route worth discussing. The first is the tunnel-type, which is 2, just like the P2MP default MDTs. It’s the same delivery mechanism in the data plane, but a more “selective” tunnel as not all nodes in the MVPN instance are required to join it. The 0x20001 is the global-id (GID) which is 131073 in decimal.

```
RP/0/0/CPU0:XRv2#show bgp ipv6 mvpn rd 213:6
[3][128][2001:10:4:6::4][128][ff33::1][213.6.6.6]/312 | begin Paths
Paths: (1 available, best #1, not advertised to EBGp peer)
  Advertised to update-groups (with more than one peer):
    0.2
  Path #1: Received by speaker 0
  Advertised to update-groups (with more than one peer):
    0.2
  Local, (Received from a RR-client)
    213.6.6.6 (metric 20) from 213.6.6.6 (213.6.6.6)
    Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, not-in-vrf
    Received Path ID 0, Local Path ID 1, version 75
    Community: no-export
    Extended community: RT:213:6
    PMSI: flags 0x00, type 2, label 0, ID
0x06000104d5060606000701000400020001
```

```
RP/0/0/CPU0:XRv2#show bgp ipv6 mvpn update-group 0.2 summary | begin Neighbor
Neighbor      Spk    AS MsgRcvd MsgSent  TblVer  InQ OutQ  Up/Down  St/PfxRcd
213.1.1.1      0    213   1894   1788     75    0    0    1d03h      1
```

213.5.5.5	0	213	1900	1797	75	0	0	1d03h	1
213.6.6.6	0	213	1909	1797	75	0	0	1d03h	2
213.7.7.7	0	213	1933	1788	75	0	0	1d03h	2
213.8.8.8	0	213	1970	1788	75	0	0	1d03h	3

A quick check on CSR1, CSR5, CSR7, and CSR8 confirms the assertion that only the last two PEs listed import and utilize this Type-3 S-PMSI route.

```
R1#show bgp ipv6 mvpn vrf MC | include \[3\]
[no output]
```

```
R5#show bgp ipv6 mvpn vrf MC | include \[3\]
[no output]
```

```
R7#show bgp ipv6 mvpn vrf MC | include \[3\]
*>i [3] [213:7] [2001:10:4:6::4] [FF33::1] [213.6.6.6]/46
```

```
R7#show bgp ipv6 mvpn vrf MC | include \[3\]
*>i [3] [213:7] [2001:10:4:6::4] [FF33::1] [213.6.6.6]/46
```

Even without the Type-3 route, CSR7 and CSR8 originated Type-7 SPT joins to CSR6 to request traffic for this C(S,G). The Type-3 route just allows the core to transport it more efficiently. The way to tell if the C-traffic is actually utilizing the data MDT is to check the C-MRIB entry. On CSR7, we can see the MDT with GID 131073 which we saw in the Type-3 route. We also see the big ‘Y’ flag which means this entry receives traffic from a data MDT (learned via Type-3 S-PMSI route). The little ‘g’ means it was installed by having sent a Type-7 SPT join. Because the GID is carried in the Type-3 route, it will be common for all subscribers to the S-PMSI. A quick check of the incoming interface and OIL shows us that the PMSI is upstream and the customer LAN is downstream, which is correct.

```
R7#show ipv6 mroute vrf MC FF33::1 | begin \
(2001:10:4:6::4, FF33::1), 1d04h/never, flags: sTIYg
Incoming interface: Lspvif0, MDT: [131073] (0x00020001), 213.6.6.6]/never
RPF nbr: ::FFFF:213.6.6.6
Immediate Outgoing interface list:
GigabitEthernet2.527, Forward, 1d04h/never
```

```
R8#show ipv6 mroute vrf MC FF33::1 | begin \
(2001:10:4:6::4, FF33::1), 1d05h/never, flags: sTIYg
Incoming interface: Lspvif0, MDT: [131073] (0x00020001), 213.6.6.6]/never
RPF nbr: ::FFFF:213.6.6.6
Immediate Outgoing interface list:
GigabitEthernet2.538, Forward, 1d05h/never
```

CSR6’s output is very similar but reversed in many ways. The flag cases have been inverted; little ‘y’ means the router originated a Type-3 S-PMSI route for this C(S,G) and is sending to an MDT data group. The big ‘G’ means it received a Type-7 SPT join for this C(S,G). Traffic enters from the customer LAN and

is sent to the PMSI using MDT 131073 (global-id). The MDT is “TX” only, where the others are receive-only; mLDP P2MP trees are always unidirectional.

```
R6#show ipv6 mroute vrf MC FF33::1 | begin \  
(2001:10:4:6::4, FF33::1), 04:36:09/never, flags: sTyG  
Incoming interface: GigabitEthernet2.546  
RPF nbr: 2001:10:4:6::4  
MDT TX nr: 131073 (0x00020001) LSM-ID: 0x12  
Immediate Outgoing interface list:  
Lspvif0, Forward, 04:36:09/never
```

Unlike ASM, there is no more BGP/C-PIM signaling that has to occur for traffic to flow. Once CSR4 starts sending traffic, CSR6 pushes the proper labels and sends it down the delivery tree. We will quickly trace the delivery tree to see if there is any obvious benefit. Remember that we already have several mLDP P2MP trees in the network, and we have just added another for S-PMSI supporting a C(S,G). Since P2MP trees are receiver-driven, we start with CSR7 and CSR8. Since we know the GID, we will use that as a filter to find the P2MP tree we want to trace. We find downstream label mappings of 7016 and 8016, one of which goes to XRv3 and one of which goes to XRv4.

```
R7#show mpls mldp database summary | include 131073  
12 P2MP 213.6.6.6 [gid 131073 (0x00020001)] 1
```

```
R7#show mpls mldp database id 12 | section Upstream  
Upstream client(s) :  
213.13.13.13:0 [Active]  
Expires : Never Path Set ID : 12  
Out Label (U) : None Interface : GigabitEthernet2.573*  
Local Label (D): 7016 Next Hop : 213.7.13.13
```

```
R8#show mpls mldp database summary | include 131073  
12 P2MP 213.6.6.6 [gid 131073 (0x00020001)] 1
```

```
R8#show mpls mldp database id 12 | section Upstream  
Upstream client(s) :  
213.14.14.14:0 [Active]  
Expires : Never Path Set ID : 12  
Out Label (U) : None Interface : GigabitEthernet2.584*  
Local Label (D): 8016 Next Hop : 213.8.14.14
```

Checking those two routers, we see that XRv4 uses label 8016 to send traffic to CSR8 and tells XRv3 to use label 94000 to send traffic towards itself. XRv3 acknowledges that label from XRv4 as well as the one from CSR7, and tells CSR6 to use label 93001 to send traffic towards it. Notice that XRv4 has one upstream and one downstream client, while XRv3 has one upstream and two downstream clients.

```
RP/0/0/CPU0:XRv4#show mpls mldp database brief | include 131073  
0x00010 P2MP 213.6.6.6 1 1 [global-id 131073]
```

```

RP/0/0/CPU0:XRv4#show mpls mldp database 0x00010
mLDP database
LSM-ID: 0x00010    Type: P2MP    Uptime: 00:34:09
  FEC Root          : 213.6.6.6
  Opaque decoded    : [global-id 131073]
  Upstream neighbor(s) :
    213.13.13.13:0 [Active] Uptime: 00:34:09
    Local Label (D) : 94000
  Downstream client(s):
    LDP 213.8.8.8:0    Uptime: 00:34:09
      Next Hop          : 213.8.14.8
      Interface          : GigabitEthernet0/0/0/0.584
      Remote label (D) : 8016

RP/0/0/CPU0:XRv3#show mpls mldp database brief | include 131073
0x00010    P2MP    213.6.6.6    1 2    [global-id 131073]

RP/0/0/CPU0:XRv3#show mpls mldp database 0x00010
mLDP database
LSM-ID: 0x00010    Type: P2MP    Uptime: 00:35:26
  FEC Root          : 213.6.6.6
  Opaque decoded    : [global-id 131073]
  Upstream neighbor(s) :
    213.6.6.6:0 [Active] Uptime: 00:35:26
    Local Label (D) : 93001
  Downstream client(s):
    LDP 213.7.7.7:0    Uptime: 00:35:26
      Next Hop          : 213.7.13.7
      Interface          : GigabitEthernet0/0/0/0.573
      Remote label (D) : 7016
    LDP 213.14.14.14:0 Uptime: 00:35:26
      Next Hop          : 213.13.14.14
      Interface          : GigabitEthernet0/0/0/0.534
      Remote label (D) : 94000

```

CSR6 is the root, using label 93001 to send traffic towards XRv3 for further distribution. Because all of the clients in the south-side of the network are requesting this data, the default MDT is identical to the data MDT. This is not an effective use of the data MDT since all PE's are requesting traffic. From a design perspective, it would not make sense to signal S-PMSI for this flow. We will test it quickly to verify it works. We just recently checked the C-MRIB, and knowing that the C(S,G) state won't change, we will just check the forwarding counters.

```

R6#show ipv6 mroute vrf MC FF33::1 count | include HW
  HW Forwarding: 37/0/118/0, Other: 0/0/0

R7#show ipv6 mroute vrf MC FF33::1 count | include HW
  HW Forwarding: 38/0/126/0, Other: 0/0/0

```

```
R8#show ipv6 mroute vrf MC FF33::1 count | include HW
HW Forwarding: 60/0/126/0, Other: 0/0/0
```

To show a more efficient design, we temporarily remove CSR3 as a client of this C-multicast feed. Upon doing this, CSR8 immediately withdraws the Type-7 SPT join, which makes sense since it no longer wants the traffic. There is no significant change on CSR6 since CSR7's route was the best-path and there is still at least one client.

```
! CSR8
BGP[16] MVPN: withdraw c-route, type 7, bs len 0 asn=213, remote-rd=213:6,
source=2001:10:4:6::4/16,
group=FF33::1/16,
nexthop=::FFFF:213.6.6.6,
len left = 0
BGP: MVPN(16) deleting the local route
[7] [213:6] [213] [2001:10:4:6::4] [FF33::1]/46
BGP(16): no valid path for [7] [213:6] [213] [2001:10:4:6::4] [FF33::1]/46
BGP(16): nettable_walker [7] [213:6] [213] [2001:10:4:6::4] [FF33::1]/46 no best
path
BGP(16): delete RIB route [7] [213:6] [213] [2001:10:4:6::4] [FF33::1]/46
BGP(16): (base) 213.12.12.12 send unreachable (format)
[7] [213:6] [213] [2001:10:4:6::4] [FF33::1]/46
```

XRv4 also generates syslog saying that it has deleted a branch to CSR8. This is because CSR8 is no longer part of this P2MP S-PMSI tree. Because XRv4 has no clients, XRv3 prunes XRv4 from the tree also.

```
RP/0/0/CPU0:XRv4#mpls_ldp[1042]: %ROUTING-MLDP-5-BRANCH_DELETE : 0x00010
[global-id 131073] P2MP 213.6.6.6, Delete LDP 213.8.8.8:0 branch remote label
8016

RP/0/0/CPU0:XRv3#mpls_ldp[1042]: %ROUTING-MLDP-5-BRANCH_DELETE : 0x00010
[global-id 131073] P2MP 213.6.6.6, Delete LDP 213.14.14.14:0 branch remote
label 94000
```

A quick check of XRv3's LFIB shows that the traffic is only going to CSR7, as expected. Now the S-PMSI has some benefit since using the I-PMSI from CSR6 would send traffic to CSR8 also, only to be dropped. Assuming there was full IPv6 connectivity in the VPN, like IPv4, the I-PMSI would span all PE's. We also check the mLDP database to verify the LFIB is correct. By removing this one client from the S-PMSI, both XRv4 and CSR8 were removed from the S-PMSI delivery tree.

```
RP/0/0/CPU0:XRv3#show mpls forwarding labels 93001
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
93001	7016	MLDP: 0x00010	Gi0/0/0/0.573	213.7.13.7	0

```

RP/0/0/CPU0:XRv3#show mpls mldp database 0x00010
mLDP database
LSM-ID: 0x00010    Type: P2MP    Uptime: 00:47:29
  FEC Root          : 213.6.6.6
  Opaque decoded     : [global-id 131073]
  Upstream neighbor(s) :
    213.6.6.6:0 [Active] Uptime: 00:47:29
      Local Label (D) : 93001
  Downstream client(s):
    LDP 213.7.7.7:0    Uptime: 00:47:29
      Next Hop          : 213.7.13.7
      Interface         : GigabitEthernet0/0/0/0.573
      Remote label (D) : 7016

```

As always, we finish with looking at OAM. Because mLDP P2MP trees are unidirectional, we must test them from the root. This is awkward because it looks like we are pinging the root from the root, but in reality we are just identifying the mLDP FEC and opaque values. XE does not have a keyword for “global-id”. Referencing “draft-bishnoi-mpls-mldp-opaque-types-01.txt” indicates that this opaque type is 0x01, so we can use that. From CSR6, we can test one-way connectivity from CSR6 to all of its leaves on the I-PMSI. We see three trees rooted at CSR6. The first one is the I-PMSI for the IPv4 MVPN. We verify that the remaining 5 PEs are connected on this I-PMSI. The “ping” command requires the exact hex data of the GID since there isn’t a more graceful way to test this in XE currently.

```

R6#show mpls mldp database summary | include 213.6.6.6
8          P2MP      213.6.6.6      [gid 65536 (0x00010000)]      2
7          P2MP      213.6.6.6      [gid 131072 (0x00020000)]      2
12         P2MP      213.6.6.6      [gid 131073 (0x00020001)]      2

```

```

R6#ping mpls mldp p2mp 213.6.6.6 hex 0x01 00010000
p2mp Root node addr 213.6.6.6
Opaque type hex value (0x1), num hex digits 4
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
  timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
[snip]

```

```

Request #1
! reply addr 213.8.14.8
! reply addr 213.5.13.5
! reply addr 213.1.14.1
! reply addr 213.7.13.7
! reply addr 213.12.13.12

```

```

Round-trip min/avg/max = 22/105/158 ms
Received 5 replies

```


We can also quickly verify the IPv6 I-PMSI and IPv6 S-PMSI trees. We expect the former to have two clients (CSR7 and CSR8) while the latter only has one client since we removed the MLD join from CSR3 (only CSR7 is a client).

```
R6#ping mpls mldp p2mp 213.6.6.6 hex 0x01 00020000
p2mp Root node addr 213.6.6.6
Opaque type hex value (0x1), num hex digits 4
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
    timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
[snip]
```

```
Request #1
! reply addr 213.7.13.7
! reply addr 213.8.14.8
```

Round-trip min/avg/max = 58/96/134 ms

```
R6#ping mpls mldp p2mp 213.6.6.6 hex 0x01 00020001
p2mp Root node addr 213.6.6.6
Opaque type hex value (0x1), num hex digits 4
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
    timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
[snip]
```

```
Request #1
! reply addr 213.7.13.7
```

Round-trip min/avg/max = 103/103/103 ms

Quickly testing XRv2, we look at the IPv4 I-PMSI and IPv6 I-PMSI. XR has a nice keyword for “global-id”, but you can also do it the hard way using the hex data. I demonstrate both below. It doesn’t seem to work, though, possibly due to a GID mismatch or XRv LSM limitations. I do not troubleshoot this further.

```
RP/0/0/CPU0:XRv2#show mpls mldp database brief | include 213.12.12.12
0x0000B P2MP 213.12.12.12 0 2 [global-id 1]
0x00005 P2MP 213.12.12.12 0 2 [global-id 262145]
```

```
RP/0/0/CPU0:XRv2#ping mpls mldp p2mp 213.12.12.12 hex 0x1 00000001
Sending 1, 100-byte MPLS Echos to mldp p2mp 213.12.12.12 hex fec (0x1,
00000001),
    timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]
```

```
Request #1
.
```

```
RP/0/0/CPU0:XRv2#ping mpls mldp p2mp 213.12.12.12 global-id 262145
```

```
Sending 1, 100-byte MPLS Echos to mldp p2mp 213.12.12.12 global-id 262145,
      timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]
```

Request #1

.

Additional Reading – Reference configurations “mvpn-12”

28.11 Profile 13: Default MDT – MLDP – MP2MP – BGP-AD – BGP C-mcast Signaling

For this test, begin with the MVPN Profile 9 configuration where all of the MVPN BGP AD is set up already, plus the MP2MP/P2MP trees are built. XRv4 is the MP2MP root and there are some other P2MP roots related to SSM joins. Enabling BGP C-multicast signaling is very easy, and the snippets are below.

```
! Enable BGP c-mcast signaling in XE PEs
vrf definition MC
  address-family ipv4
    mdt overlay use-bgp
  address-family ipv6
    mdt overlay use-bgp

! Enable BGP c-mcast signaling in XRv2
router pim
  vrf MC
    address-family ipv4
      mdt c-multicast-routing bgp
    address-family ipv6
      mdt c-multicast-routing bgp
```

Pitfall: There does not appear to be a way to stop XRv from sending PIM hellos over the emulated LAN.

Every router will still see XRv2 as a PIM neighbor. The real power of using BGP AD and C-mcast signaling together is that now (with the exception of the XRv issue) there is no PIM anywhere in the carrier network, other than PE-CE links. First, let's ensure some of the remote PEs still know who the RP is. After all, there is no PIM adjacency on the LSPvif0 anymore, so there must be a mechanism for the BSR messages to flow. CSR6, for example, still learns the RP. The debug confirms it came from the LSPvif0 interface.

```
R6#show ip pim vrf MC rp mapping
PIM Group-to-RP Mappings
Group(s) 224.0.0.0/4
  RP 10.5.10.10 (?), v2
    Info source: 10.5.10.10 (?), via bootstrap, priority 0, holdtime 150
    Uptime: 00:53:56, expires: 00:02:14
```

```

R6#debug ip pim vrf MC
PIM(1): Received v2 Bootstrap on Lspvif0 from 213.5.5.5
(1): pim_add_prm:: 224.0.0.0/240.0.0.0, rp=10.5.10.10, repl = 0, ver =2,
is_neg =0, bidir = 0, crp = 0
PIM(1): Update
prm_rp->bidir_mode = 0 vs bidir = 0 (224.0.0.0/4, RP:10.5.10.10), PIMv2

```

Double checking RPF and PIM neighbors, we clearly see the correct RPF to CSR5, but no PIM neighbor with CSR5. This seems to violate some fundamental PIM rules.

```

R6#show ip rpf vrf MC 10.5.10.10
RPF information for ? (10.5.10.10)
RPF interface: Lspvif0
RPF neighbor: ? (213.5.5.5)
RPF route/mask: 10.5.10.0/24
RPF type: unicast (bgp 213)
Doing distance-preferred lookups across tables
RPF topology: ipv4 multicast base, originated from ipv4 unicast base

```

```

R6#show ip pim vrf MC neighbor lspvif 0 | begin ^Neighbor
Neighbor          Interface          Uptime/Expires    Ver    DR
Address
213.12.12.12      Lspvif0          01:43:02/00:01:34 v2      1 / DR P G

```

This is where the Type 1 AD route is key. Remember that because all the PEs have discovered one another via the I-PSMI BGP AD routes, they know to use the bidirectional MP2MP tree. The Type-1 route signaled this with the tunnel-type 7 option and including the root address in the tunnel parameters, as discussed in Profile 9. Thus, ASM can still work with this design even without PIM in the overlay. Using MP2MP root 213.14.14.14 (0xD50E0E0E), we can reach CSR5.

```

R5#show bgp ipv4 mvpn vrf MC route-type 1 213.5.5.5
BGP routing table entry for [1][213:5][213.5.5.5]/12, version 54
Paths: (1 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGp
peer)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (213.5.5.5)
      Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
      Community: no-export
      Extended Community: RT:213:5
      PMSI Attribute: Flags: 0x0, Tunnel type: 7, length 24, label: exp-null,
tunnel parameters: 0800 0104 D50E 0E0E 000E 0200 0B00 0213 0000 0213 0000
0000
      rx pathid: 0, tx pathid: 0x0

```

Also recall that CSR4 is issuing IGMP joins for an ASM group 225.0.0.1 and an SSM group 232.0.0.5. In order to send the C(*,G) join to the C-RP, CSR6 must know the C-RP address first, which was just discussed. Once it knows that, it generates a Type 6 Shared-Tree Join route which is directly comparable to a PIM (*,G) join. This route information contains many pieces of information, and the RP address is included as the “source”. In this way, it looks just like the Type 7 Source-Tree Join route which is examined later.

```
R6#show bgp ipv4 mvpn vrf MC route-type 6 213:5 213 10.5.10.10 225.0.0.1
BGP routing table entry for [6][213:5][213][10.5.10.10/32][225.0.0.1/32]/22,
version 56
Paths: (1 available, best #1, table MVPNV4-BGP-Table)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (213.6.6.6)
    Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
    Extended Community: RT:213.5.5.5:2
    rx pathid: 0, tx pathid: 0x0
```

Looking at this BGP Type-6 route on CSR5, we can see it was iBGP learned from CSR6 through the route-reflector, XRv2. The RT is the MDT source address of the PE who is closest to the RP, in this case, and the “2” at the end is a random number used to differentiate the RTs for different C(*,G) joins. This allows CSR6 to “target” CSR5; since XRv2 reflects this route to many peers, those other peers should not import it.

```
R5#show bgp ipv4 mvpn vrf MC route-type 6 213:5 213 10.5.10.10 225.0.0.1
BGP routing table entry for [6][213:5][213][10.5.10.10/32][225.0.0.1/32]/22,
version 66
Paths: (1 available, best #1, table MVPNV4-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    213.6.6.6 (metric 20) from 213.12.12.12 (213.12.12.12)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    Extended Community: RT:213.5.5.5:2
    Originator: 213.6.6.6, Cluster list: 213.12.12.12
    rx pathid: 0, tx pathid: 0x0
```

With this route in the BGP RIB, the MRIB process for this VPN also needs to update itself so multicast traffic can flow. The C(*,G) entry looks normal, with the exception of the big “G” flag, which means the entry was installed by a received BGP c-mcast route. Normally, MRIB entries are the results of PIM signaling (joins, prunes, etc). In this case, the MRIB entry was BGP-signaled.

```

R5#show ip mroute vrf MC sparse
(*, 225.0.0.1), 00:00:37/00:02:22, RP 10.5.10.10, flags: SG
  Incoming interface: GigabitEthernet2.550, RPF nbr 10.5.10.10
  Outgoing interface list:
    Lspvif0, Forward/Sparse, 00:00:37/00:02:22

```

This raises the question of why CSR5 only has one Type 6 route when CSR1 and CSR8 should have also generated these routes. The answer is because XRv2 is a RR, it only advertises its best route. CSR6 is best because the IGP metric to the BGP next-hop is lowest. I've highlighted the show command because it is very long. This "loss" of information does not negatively affect forwarding; the LSPvif is in the OIL of the C(*,G) if at least one BGP Type 6 route is received. It does not matter which one, and BGP can be used to help scale MVPNs in this way. XRv2 should have generated one too, but I don't think it is fully supported.

```

RP/0/0/CPU0:XRv2#show bgp ipv4 mvpn rd 213:5
[6] [213:5] [213] [32] [10.5.10.10] [32] [225.0.0.1]/184 bestpath-compare
BGP routing table entry for
[6] [213:5] [213] [32] [10.5.10.10] [32] [225.0.0.1]/184, Route Distinguisher:
213:5
Versions:
  Process          bRIB/RIB   SendTblVer
  Speaker          52         52
Paths: (3 available, best #2)
  Advertised to update-groups (with more than one peer):
    0.2
  Path #1: Received by speaker 0
  Not advertised to any peer
  Local, (Received from a RR-client)
    213.1.1.1 (metric 40) from 213.1.1.1 (213.1.1.1)
    Origin incomplete, metric 0, localpref 100, valid, internal, import-
candidate, not-in-vrf, import suspect
    Received Path ID 0, Local Path ID 0, version 0
    Extended community: RT:213.5.5.5:2
    Higher IGP metric than best path (path #2)
  Path #2: Received by speaker 0
  Advertised to update-groups (with more than one peer):
    0.2
  Local, (Received from a RR-client)
    213.6.6.6 (metric 20) from 213.6.6.6 (213.6.6.6)
    Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, not-in-vrf, import suspect
    Received Path ID 0, Local Path ID 1, version 52
    Extended community: RT:213.5.5.5:2
    best of local AS, Overall best
  Path #3: Received by speaker 0
  Not advertised to any peer
  Local, (Received from a RR-client)
    213.8.8.8 (metric 40) from 213.8.8.8 (213.8.8.8)

```

```

Origin incomplete, metric 0, localpref 100, valid, internal, import-
candidate, not-in-vrf, import suspect
Received Path ID 0, Local Path ID 0, version 0
Extended community: RT:213.5.5.5:2
Higher IGP metric than best path (path #2)

```

There is a lot of upcoming activity once CSR2 starts sending traffic. I have enabled IPv4 MVPN debugging on CSR7 (first hop router), CSR6 (last hop router), and CSR5 (PE towards RP). CSR7 sends a PIM register message, which is unicast, towards the RP. I enabled this debugging after the fact, so registration is complete, but we can see the registration process continuing so the RP is aware of the active source.

```

! CSR7
PIM(2): Send v2 Data-header Register to 10.5.10.10 for 10.2.7.2, group
225.0.0.1
PIM(2): Received v2 Register-Stop on GigabitEthernet2.574 from 10.5.10.10
PIM(2): for source 10.2.7.2, group 225.0.0.1
PIM(2): Clear Registering flag to 10.5.10.10 for (10.2.7.2/32, 225.0.0.1)

```

The RP tries to join to SPT and sends a C(S,G) join to CSR5. CSR5 then creates a BGP Type-7 route and sends it to the RR. RR rules still apply, and when XRv2 reflects it back, it gets rejected on ingress.

```

! CSR5
BGP[15] MVPN: add c-route, type 7, bs len 0 asn=0, rd=213:5,
source=10.2.7.2/4,
group=225.0.0.1/4,
nexthop=213.7.7.7,
len left = 0
BGP: MVPN(15) create local route
[7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22
BGP(15): nettable_walker [7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22 route
sourced locally
BGP(15): 213.12.12.12 NEXT_HOP self is set for sourced RT Filter for net
[7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22
BGP(15): (base) 213.12.12.12 send UPDATE (format)
[7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22, next 213.5.5.5, metric 0, path
Local, extended community RT:213.7.14.7:2
BGP(15): 213.12.12.12 rcv UPDATE w/ attr: nexthop 213.5.5.5, origin ?,
localpref 100, metric 0, originator 213.5.5.5, clusterlist 213.12.12.12,
merged path , AS_PATH , community , extended community RT:213.7.14.7:2, SSA
attribute
BGP(15): 213.12.12.12 rcv UPDATE about
[7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22 -- DENIED due to: ORIGINATOR is
us; MP_REACH NEXTHOP is our own address;

```

Upon receiving this Type-7 Source-Tree Join, CSR7 issues a Type-5 Source Active AD route. This announces that there is a source actively sending traffic behind the originating PE. CSR7 adds the Type-7 to its MVPN RIB, then generates the Type 5 route and sends it to the RR.

```

! CSR7
BGP(15): 213.12.12.12 rcvd UPDATE w/ attr: nexthop 213.5.5.5, origin ?,
localpref 100, metric 0, originator 213.5.5.5, clusterlist 213.12.12.12,
extended community RT:213.7.14.7:2
BGP(15): 213.12.12.12 rcvd [7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22
BGP(15): skip vrf default table RIB route
[7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22
BGP(15): add RIB route [7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22
BGP(15): nettable_walker [5][213:7][10.2.7.2][225.0.0.1]/18 route sourced
locally
BGP(15): delete RIB route [5][213:7][10.2.7.2][225.0.0.1]/18
BGP(15): 213.12.12.12 NEXT_HOP self is set for sourced RT Filter for net
[5][213:7][10.2.7.2][225.0.0.1]/18
BGP(15): (base) 213.12.12.12 send UPDATE (format)
[5][213:7][10.2.7.2][225.0.0.1]/18, next 213.7.7.7, metric 0, path Local,
extended community RT:213:7

```

When CSR5 receives this Source Active message, it installs it in the RIB.

```

! CSR5
BGP(15): 213.12.12.12 rcvd UPDATE w/ attr: nexthop 213.7.7.7, origin ?,
localpref 100, metric 0, originator 213.7.14.7, clusterlist 213.12.12.12,
community no-export, extended community RT:213:7
BGP(15): 213.12.12.12 rcvd [5][213:7][10.2.7.2][225.0.0.1]/18
BGP(15): skip vrf default table RIB route [5][213:7][10.2.7.2][225.0.0.1]/18
BGP(15): add RIB route [5][213:5][10.2.7.2][225.0.0.1]/18

```

Like CSR10, when CSR4 saw multicast traffic, it immediately tries to switch to the SPT and issues a PIM (S,G) join to CSR6. CSR6 then generates a Type 7 BGP route to indicate interest in this C(S,G). This goes to the RR, along with the Type 7 routes from CSR5, CSR1, and CSR8. All of their CEs do the same thing.

```

! CSR6
BGP[15] MVPN: add c-route, type 7, bs len 0 asn=0, rd=213:6,
source=10.2.7.2/4,
group=225.0.0.1/4,
nexthop=213.7.7.7,
len left = 0
BGP: MVPN(15) create local route
[7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22
BGP(15): nettable_walker [7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22 route
sourced locally
BGP(15): 213.12.12.12 NEXT_HOP self is set for sourced RT Filter for net
[7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22
BGP(15): (base) 213.12.12.12 send UPDATE (format)
[7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22, next 213.6.6.6, metric 0, path
Local, extended community RT:213.7.14.7:2

```

CSR6 receives a Type 7 route from CSR5 for (10.2.7.2, 225.0.0.1). Clearly this source is NOT behind CSR6, and CSR6 should not accept this route. This is the purpose of the dynamic RTs; CSR6 is not importing RT:213.7.14.7:2, only CSR7 is, because the source is behind CSR7 only. The route is rejected. However, the Source Active AD route (Type-5) is accepted from the RR and installed. The RT on the Type 5 route, since this is an AD route (not C-mcast signaling, technically) is the standard unicast RT.

```
! CSR6
BGP(15): 213.12.12.12 rcvd UPDATE w/ attr: nexthop 213.5.5.5, origin ?,
localpref 100, metric 0, originator 213.5.5.5, clusterlist 213.12.12.12,
extended community RT:213.7.14.7:2
BGP(15): 213.12.12.12 rcvd [7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22 --
DENIED due to: extended community not supported;
BGP(15): 213.12.12.12 rcvd UPDATE w/ attr: nexthop 213.7.7.7, origin ?,
localpref 100, metric 0, originator 213.7.14.7, clusterlist 213.12.12.12,
community no-export, extended community RT:213:7
BGP(15): 213.12.12.12 rcvd [5][213:7][10.2.7.2][225.0.0.1]/18
BGP(15): skip vrf default table RIB route [5][213:7][10.2.7.2][225.0.0.1]/18
BGP(15): add RIB route [5][213:6][10.2.7.2][225.0.0.1]/18
```

On CSR7, the Type-7 route is received and accepted; the dynamic RT is imported because the C-source 10.2.7.2 is behind CSR7. CSR7 originated the Type-5 Source Active message which indicated so. This is all logical.

```
! CSR7
BGP(15): 213.12.12.12 rcvd UPDATE w/ attr: nexthop 213.5.5.5, origin ?,
localpref 100, metric 0, originator 213.5.5.5, clusterlist 213.12.12.12,
extended community RT:213.7.14.7:2
BGP(15): 213.12.12.12 rcvd [7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22
BGP(15): add RIB route [7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22
```

As a quick check, let's identify why CSR7 only saw, and subsequently installed, the Type-7 route from CSR5. This is because XRv2 only advertised the best path, which won against CSR6 due to lowest RID. As long as the first-hop router receives one Type 7, that is all that matters.

```
! CSR7
RP/0/0/CPU0:XRv2#show bgp ipv4 mvpn rd 213:7
[7][213:7][213][32][10.2.7.2][32][225.0.0.1]/184 bestpath-compare
BGP routing table entry for [7][213:7][213][32][10.2.7.2][32][225.0.0.1]/184,
Route Distinguisher: 213:7
Versions:
  Process          bRIB/RIB  SendTblVer
  Speaker          57        57
Paths: (4 available, best #2)
  Advertised to update-groups (with more than one peer):
    0.2
  Path #1: Received by speaker 0
  Not advertised to any peer
```



```

Local, (Received from a RR-client)
  213.1.1.1 (metric 40) from 213.1.1.1 (213.1.1.1)
    Origin incomplete, metric 0, localpref 100, valid, internal, import-
candidate, not-in-vrf
    Received Path ID 0, Local Path ID 0, version 0
    Extended community: RT:213.7.14.7:2
    Higher IGP metric than best path (path #2)
Path #2: Received by speaker 0
Advertised to update-groups (with more than one peer):
  0.2
Local, (Received from a RR-client)
  213.5.5.5 (metric 20) from 213.5.5.5 (213.5.5.5)
    Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, not-in-vrf
    Received Path ID 0, Local Path ID 1, version 57
    Extended community: RT:213.7.14.7:2
    best of local AS, Overall best
Path #3: Received by speaker 0
Not advertised to any peer
Local, (Received from a RR-client)
  213.6.6.6 (metric 20) from 213.6.6.6 (213.6.6.6)
    Origin incomplete, metric 0, localpref 100, valid, internal, import-
candidate, not-in-vrf
    Received Path ID 0, Local Path ID 0, version 0
    Extended community: RT:213.7.14.7:2
    Higher router ID than best path (path #2)
Path #4: Received by speaker 0
Not advertised to any peer
Local, (Received from a RR-client)
  213.8.8.8 (metric 40) from 213.8.8.8 (213.8.8.8)
    Origin incomplete, metric 0, localpref 100, valid, internal, import-
candidate, not-in-vrf
    Received Path ID 0, Local Path ID 0, version 0
    Extended community: RT:213.7.14.7:2
    Higher IGP metric than best path (path #2)

```

Let's verify the Type-7 installation on CSR7. These parameters should match up with the debugs. The route is from CSR5 (doesn't really matter) and describes C(10.2.7.2, 225.0.0.1). CSR7 also originates the Type-5 route. Again, notice the difference between the dynamic RT on the Type-7 and the standard RT on the Type-5.

```

R7#show bgp ipv4 mvpn vrf MC route-type 7 213:7 213 10.2.7.2 225.0.0.1
BGP routing table entry for [7] [213:7] [213] [10.2.7.2/32] [225.0.0.1/32]/22,
version 59
Paths: (1 available, best #1, table MVPNv4-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    213.5.5.5 (metric 20) from 213.12.12.12 (213.12.12.12)

```

```
Origin incomplete, metric 0, localpref 100, valid, internal, best
Extended Community: RT:213.7.14.7:2
Originator: 213.5.5.5, Cluster list: 213.12.12.12
rx pathid: 0, tx pathid: 0x0
```

```
R7#show bgp ipv4 mvpn vrf MC route-type 5 10.2.7.2 225.0.0.1
BGP routing table entry for [5][213:7][10.2.7.2][225.0.0.1]/18, version 58
Paths: (1 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGp
peer)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (213.7.14.7)
      Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
      Community: no-export
      Extended Community: RT:213:7
      rx pathid: 0, tx pathid: 0x0
```

Checking the same information on CSR5, we see the Type-5 was learned from CSR7 and installed, while the Type-7 was locally originated.

```
R5#show bgp ipv4 mvpn vrf MC route-type 5 10.2.7.2 225.0.0.1
BGP routing table entry for [5][213:5][10.2.7.2][225.0.0.1]/18, version 69
Paths: (1 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGp
peer)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from [5][213:7][10.2.7.2][225.0.0.1]/18 (global)
    213.7.7.7 (metric 20) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Community: no-export
      Extended Community: RT:213:7
      Originator: 213.7.14.7, Cluster list: 213.12.12.12
      rx pathid: 0, tx pathid: 0x0
```

```
R5#show bgp ipv4 mvpn vrf MC route-type 7 213:7 213 10.2.7.2 225.0.0.1
BGP routing table entry for [7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22,
version 67
Paths: (1 available, best #1, table MVPNV4-BGP-Table)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (213.5.5.5)
      Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
      Extended Community: RT:213.7.14.7:2
```

```
rx pathid: 0, tx pathid: 0x0
```

CSR6, as the last hop router, acknowledges the Type-5 and originates a Type-7. CSR6's Type-7 loses best-path on XRv2 and is not advertised further, but it doesn't matter since an identical C(S,G) route from CSR5 was advertised on.

```
R6#show bgp ipv4 mvpn vrf MC route-type 5 10.2.7.2 225.0.0.1
BGP routing table entry for [5][213:6][10.2.7.2][225.0.0.1]/18, version 59
Paths: (1 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGP
peer)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from [5][213:7][10.2.7.2][225.0.0.1]/18 (global)
    213.7.7.7 (metric 20) from 213.12.12.12 (213.12.12.12)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    Community: no-export
    Extended Community: RT:213:7
    Originator: 213.7.14.7, Cluster list: 213.12.12.12
    rx pathid: 0, tx pathid: 0x0
```

```
R6#show bgp ipv4 mvpn vrf MC route-type 7 213:7 213 10.2.7.2 225.0.0.1
BGP routing table entry for [7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22,
version 57
Paths: (1 available, best #1, table MVPNV4-BGP-Table)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (213.6.6.6)
    Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
    Extended Community: RT:213.7.14.7:2
    rx pathid: 0, tx pathid: 0x0
```

For completeness, we take a quick look at CSR1 and CSR8 also. Their process was identical to CSR6. The Type-7 routes (the C(S,G) joins) were sourced locally and the Type-5 route (Source Active) was learned via the RR with next-hop CSR7, the first-hop router.

```
R1#show bgp ipv4 mvpn vrf MC route-type 5 10.2.7.2 225.0.0.1 | include
^BGP|from
BGP routing table entry for [5][213:1][10.2.7.2][225.0.0.1]/18, version 25
  Local, imported path from [5][213:7][10.2.7.2][225.0.0.1]/18 (global)
    213.7.7.7 (metric 20) from 213.12.12.12 (213.12.12.12)

R1#show bgp ipv4 mvpn vrf MC route-type 7 213:7 213 10.2.7.2 225.0.0.1 |
include ^BGP|from
BGP routing table entry for [7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22,
version 23
```

```
0.0.0.0 from 0.0.0.0 (213.1.1.1)
```

```
R8#show bgp ipv4 mvpn vrf MC route-type 5 10.2.7.2 225.0.0.1 | include  
^BGP|from  
BGP routing table entry for [5][213:8][10.2.7.2][225.0.0.1]/18, version 61  
Local, imported path from [5][213:7][10.2.7.2][225.0.0.1]/18 (global)  
213.7.7.7 (metric 20) from 213.12.12.12 (213.12.12.12)
```

```
R8#show bgp ipv4 mvpn vrf MC route-type 7 213:7 213 10.2.7.2 225.0.0.1 |  
include ^BGP|from  
BGP routing table entry for [7][213:7][213][10.2.7.2/32][225.0.0.1/32]/22,  
version 59  
0.0.0.0 from 0.0.0.0 (213.8.8.8)
```

Now that all the signaling was verified, we can check the multicast routing. First, we check CSR7 to ensure traffic is arriving from the VPN customer and going towards the core via LSPvif0. We see some new flags here. The big “G” means the entry was installed from a received BGP c-mcast route, in this case, a Type-7 Source-Tree Join from CSR5. The little ‘q’ means that this router issues a Type-5 Source Active route for this C(S,G). The packet counters for CSR7 continue to increase, indicating a functional data-plane.

```
R7#show ip mroute vrf MC 225.0.0.1 10.2.7.2 | begin \  
(10.2.7.2, 225.0.0.1), 00:01:25/00:01:37, flags: FTGq  
Incoming interface: GigabitEthernet2.527, RPF nbr 0.0.0.0  
Outgoing interface list:  
Lspvif0, Forward/Sparse, 00:01:25/00:01:34
```

```
R7#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | begin ^Group  
Group: 225.0.0.1, Source count: 1, Packets forwarded: 87, Packets received:  
88  
Source: 10.2.7.2/32, Forwarding: 87/1/117/0, Other: 88/1/0
```

CSR6, CSR8, and CSR1 should have identical outputs with the exception of minor differences like packet counters, since I checked their outputs at different times. The flags are “inverted” on these entries: the little ‘g’ signifies that a BGP c-mcast route was sent representing this C(S,G), which is true because CSR6 did originate a Type-7 route for it. The big ‘Q’ means that a Type-5 Source Active route was received for this C(S,G). This is also true because CSR7 originated that message. Notice the counters are increasing.

```
R6#show ip mroute vrf MC 225.0.0.1 10.2.7.2 | begin \  
(10.2.7.2, 225.0.0.1), 00:01:02/00:01:57, flags: JTgQ  
Incoming interface: Lspvif0, RPF nbr 213.7.7.7  
Outgoing interface list:  
GigabitEthernet2.546, Forward/Sparse, 00:01:02/00:02:06
```

```
R6#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | begin ^Group  
Group: 225.0.0.1, Source count: 1, Packets forwarded: 58, Packets received:  
58
```

Source: 10.2.7.2/32, Forwarding: 57/1/122/0, Other: 57/0/0

```
R8#show ip mroute vrf MC 225.0.0.1 10.2.7.2 | begin \  
(10.2.7.2, 225.0.0.1), 00:02:14/00:00:45, flags: JTgQ  
Incoming interface: Lspvif0, RPF nbr 213.7.7.7  
Outgoing interface list:  
GigabitEthernet2.538, Forward/Sparse, 00:02:14/00:02:54
```

```
R8#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | begin ^Group  
Group: 225.0.0.1, Source count: 1, Packets forwarded: 131, Packets received:  
131  
Source: 10.2.7.2/32, Forwarding: 130/1/122/0, Other: 130/0/0
```

```
R1#show ip mroute vrf MC 225.0.0.1 10.2.7.2 | begin \  
(10.2.7.2, 225.0.0.1), 00:02:04/00:00:54, flags: JTgQ  
Incoming interface: Lspvif0, RPF nbr 213.7.7.7  
Outgoing interface list:  
GigabitEthernet2.519, Forward/Sparse, 00:02:04/00:02:32
```

```
R1#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | begin ^Group  
Group: 225.0.0.1, Source count: 1, Packets forwarded: 123, Packets received:  
123  
Source: 10.2.7.2/32, Forwarding: 122/1/122/0, Other: 122/0/0
```

Last, let's check CSR5. Very few multicast packets should be shown here, since the C-RP will prune itself from the SPT when it realizes it is not in the transit path anymore. The big 'P' indicates a prune and the little 'g' indicates the entry is the result of a sent BGP c-mcast message, specifically a Type-7 SPT join. The LSPvif is still a "LAN" in terms of forwarding, because this traffic is using the MP2MP mLDP tree.

```
R5#show ip mroute vrf MC 225.0.0.1 10.2.7.2 | begin \  
(10.2.7.2, 225.0.0.1), 00:02:42/00:00:17, flags: PTXg  
Incoming interface: Lspvif0, RPF nbr 213.7.7.7  
Outgoing interface list: Null
```

```
R5#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | begin ^Group  
Group: 225.0.0.1, Source count: 1, Packets forwarded: 5, Packets received:  
154  
Source: 10.2.7.2/32, Forwarding: 4/0/122/0, Other: 153/1/148
```

Earlier examples showed us the MDT information inside the MRIB entries, but no longer. One way we know this traffic is using the MP2MP mLDP tree is the lack of an S-PMSI AD route (Type-3). When this route is present, as it is currently for SSM traffic, the P2MP root is encoded in the same manner as the MP2MP tunnel, except the tunnel type is changed to 2, which is mLDP P2MP. We explore this next. First, we will do an inbound capture on CSR6 to see if the downstream label 6001 is used. We select CSR6 because it participates in both a P2MP tree (SSM) and the MP2MP tree. Assuming we were confused as to which one was in use, we check the incoming label. If the label is 6013, that means the P2MP tree is used.

```
R6#show mpls mldp database summary
```

LSM ID	Type	Root	Decoded Opaque Value	Client
Cnt.				
1C	P2MP	213.7.7.7	[mdt 213:213 1]	1
19	MP2MP	213.14.14.14	[mdt 213:213 0]	1

```
R6#show mpls mldp database opaque_type mdt 213:213 0 | section Upstream
```

```
Upstream client(s) :
```

213.13.13.13:0	[Active]		
Expires	: Never	Path Set ID	: 1F
Out Label (U)	: 93004	Interface	: GigabitEthernet2.563*
Local Label (D)	: 6001	Next Hop	: 213.6.13.13

```
R6#show mpls mldp database opaque_type mdt 213:213 1 | section Upstream
```

```
Upstream client(s) :
```

213.13.13.13:0	[Active]		
Expires	: Never	Path Set ID	: 23
Out Label (U)	: None	Interface	: GigabitEthernet2.563*
Local Label (D)	: 6013	Next Hop	: 213.6.13.13

We use EPC to verify the packet, and also check the LFIB for sanity. Label 0x1771 is 6001. The counters for this LSP constantly increase.

```
R6#show monitor capture CAP buffer detailed
```

```
0 122 0.000000 00:50:56:A9:DB:37 -> 00:50:56:A9:DE:0D MPLS unicast
0000: 005056A9 DE0D0050 56A9DB37 81000DEB .PV....PV..7....
0010: 88470177 11FC4500 00641576 0000FE01 .G.w..E..d.v....
0020: B51D0A02 0702E100 00010800 906C001B .....l..
0030: 120A0000 00004E95 8D23ABCD ABCDABCD .....N..#.....
```

```
R6#show mpls forwarding-table labels 6001
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Outgoing interface	Next Hop
6001	[T] No Label	[mdt 213:213 0] [V]	\		
			570724	aggregate/MC	

The process for SSM is a little more straightforward since there is no RP, and therefore no registration. Looking deeper at the Type-7 BGP SPT join, we see that for SSM, it is originated by both CSR6 and CSR8. Both of them have SSM clients (CSR3 and CSR4) that are requesting traffic for 232.0.0.5. On XRv2, we see both routes have an RD of 213:7. Note that CSR6 is the best path.

```
RP/0/0/CPU0:XRv2#show bgp ipv4 mvpn rd 213:7 route-type 7 | begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 213:7					
*>i[7][213:7][213][32][10.2.7.2][32][232.0.0.5]/184					
	213.6.6.6	0	100	0	?

```
* i                213.8.8.8                0    100    0 ?
```

Normally, the RD of a VPN route is equal to that of the originator. In this case, it is set to the RD of the destination. The only way CSR6 and CSR8 can determine this is by using the BGP AD messaging. This is where the Type 1 I-PMSI message is used. CSR8 reports learning this Type 1 from CSR7 which simply carries the RD, VPN next-hop for BGP, and tunnel information (seen earlier). The BGP next-hop determines what the dynamic RT will be (shown shortly).

```
R8#show bgp ipv4 mvpn rd 213:7 | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 213:7
*>i [1] [213:7] [213.7.7.7]/12
      213.7.7.7                0    100    0 ?
*>i [3] [213:7] [10.2.7.2] [232.0.0.5] [213.7.7.7]/22
      213.7.7.7                0    100    0 ?
*> [7] [213:7] [213] [10.2.7.2/32] [232.0.0.5/32]/22
      0.0.0.0                32768 ?
```

Upon receiving it, CSR7 creates a Type 3 S-PMSI route that states it has a C(S,G) of (10.2.7.2, 232.0.0.5) reachable via PE 213.7.7.7. The tunnel type 2 indicates mLDP P2MP tree and the root is 213.7.7.7, which is not the same as the MP2MP root of 213.14.14.14 seen in the Type-1 I-PMSI messages. **The only reason CSR7 (ingress PE) generated this was because the ACL specified that this group was candidate for data MDT construction.** The Type 3 just informs egress PEs know about the existing P2MP tree.

```
R7#show bgp ipv4 mvpn vrf MC route-type 3 10.2.7.2 232.0.0.5 213.7.7.7
BGP routing table entry for [3] [213:7] [10.2.7.2] [232.0.0.5] [213.7.7.7]/22,
version 54
Paths: (1 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGp
peer)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (213.7.14.7)
      Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
      Community: no-export
      Extended Community: RT:213:7
      PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 24, label: exp-null,
tunnel parameters: 0600 0104 D507 0707 000E 0200 0B00 0213 0000 0213 0000
0001
      rx pathid: 0, tx pathid: 0x0
```

CSR8 then assigns the RD of CSR7 along with a dynamic RT so that only CSR7 imports it. Notice the RD on the Type 7 route below, along with the RT. This RT was learned via the regular VPNv4 unicast route as an

extended community. This is also true for ASM but is detailed here instead. No extra configuration is needed to VPNv4/v6 to enable this, as long as extended communities are supported (they must be).

```
R8#show bgp ipv4 mvpn rd 213:7 route-type 7 213:7 213 10.2.7.2 232.0.0.5
BGP routing table entry for [7][213:7][213][10.2.7.2/32][232.0.0.5/32]/22,
version 56
Paths: (1 available, best #1, table MVPNV4-BGP-Table)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (213.8.8.8)
      Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
      Extended Community: RT:213.7.14.7:2
      rx pathid: 0, tx pathid: 0x0
```

```
R8#show bgp vpnv4 unicast rd 213:7 10.2.7.2
BGP routing table entry for 213:7:10.2.7.0/24, version 27
Paths: (1 available, best #1, no table)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    213.7.7.7 (metric 20) (via default) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:213:7 MVPN AS:213:0.0.0.0 MVPN VRF:213.7.14.7:2
      Originator: 213.7.14.7, Cluster list: 213.12.12.12
      mpls labels in/out nlabel/7019
      rx pathid: 0, tx pathid: 0x0
```

When the join reaches CSR7, it is imported into the MVPN because it contained the proper RT. Only CSR6's route makes it to CSR7 because XRv2 selected it as the best-path (seen earlier).

```
R7#show bgp ipv4 mvpn vrf MC route-type 7 213:7 213 10.2.7.2 232.0.0.5
BGP routing table entry for [7][213:7][213][10.2.7.2/32][232.0.0.5/32]/22,
version 55
Paths: (1 available, best #1, table MVPNV4-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    213.6.6.6 (metric 20) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:213.7.14.7:2
      Originator: 213.6.6.6, Cluster list: 213.12.12.12
      rx pathid: 0, tx pathid: 0x0
```

A quick check of the VPN MRIBs on CSR8 and CSR7 confirm our understanding. CSR6 claims the C(S,G) entry was installed by having sent a BGP route (Type 7), and CSR7 indicates the entry was installed by

having received it from an interested receiver. This is indicated by the little ‘g’ and big ‘G’, respectively. CSR6 has joined a “data MDT”, meaning a specific tree built for this flow, and CSR7 is sending to this MDT. The data MDT directions are indicated by the big ‘Y’ and little ‘y’, respectively. CSR6’s RPF faces the PMSI interface while CSR7’s RPF faces the source. We know this is definitely not using the default MDT since the command shows an MDT ID of 1. Everything appears correct.

```
R6#show ip mroute vrf MC ssm
(10.2.7.2, 232.0.0.5), 1d01h/00:02:49, flags: sTIYg
  Incoming interface: Lspvif0, RPF nbr 213.7.7.7, MDT: [1, 213.7.7.7]/never
  Outgoing interface list:
    GigabitEthernet2.546, Forward/Sparse, 1d01h/00:02:49
```

```
R7#show ip mroute vrf MC ssm
(10.2.7.2, 232.0.0.5), 1d01h/stopped, flags: sTyG
  Incoming interface: GigabitEthernet2.527, RPF nbr 0.0.0.0
  Outgoing interface list:
    Lspvif0, Forward/Sparse, 1d01h/stopped
```

When CSR2 starts sending, there is no more BGP AD or c-mcast signaling that must occur, since the trees are already fully built. CSR7 reports sending traffic successfully out to the LSPvif.

```
R7#show ip mroute vrf MC 232.0.0.5 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 96, Packets received:
96
  Source: 10.2.7.2/32, Forwarding: 96/1/118/0, Other: 96/0/0
```

CSR6 and CSR8 also show it being sent towards their respective customers. Just for extra verification, I used EPC on egress on CSR6 to show the IP data being sent. Because it’s not encapsulated, we can see the nicely formatted IP addresses in the header without digging through the hex. I highlighted the IP protocol, source, and destination IP addresses as a sanity check.

```
R6#show ip mroute vrf MC 232.0.0.5 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 76, Packets received:
76
  Source: 10.2.7.2/32, Forwarding: 76/1/122/0, Other: 76/0/0
```

```
R8#show ip mroute vrf MC 232.0.0.5 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 69, Packets received:
69
  Source: 10.2.7.2/32, Forwarding: 69/1/122/0, Other: 69/0/0
```

```
R6#show monitor capture CAP buffer detailed
-----
#   size  timestamp      source                destination  protocol
-----
  0   118    0.000000    10.2.7.2             -> 232.0.0.5      ICMP
```

```

0000: 01005E00 00050050 56A9DE0D 81000DDA ..^....PV.....
0010: 08004500 006416FA 0000FC01 AE950A02 ..E..d.....
0020: 0702E800 00050800 F9F4001C 00FF0000 .....
0030: 000053B3 2F87ABCD ABCDABCD ABCDABCD ..S./.....

```

Next, we will test IPv6. The concepts are identical to IPv4 but for completeness, we will test it. First, I recommend checking to ensure there are no PIM neighbors over the PMSI. XRv appears stubborn here, so we will ignore that.

```

R1#show ipv6 pim vrf MC neighbor | begin ^Neighbor
Neighbor Address      Interface      Uptime      Expires      Mode DR pri
::FFFF:213.12.12.12   Lspvif0       1d03h       00:01:36 B G   DR 1

```

```

R5#show ipv6 pim vrf MC neighbor | begin ^Neighbor
Neighbor Address      Interface      Uptime      Expires      Mode DR pri
::FFFF:213.12.12.12   Lspvif0       1d03h       00:01:18 B G   DR 1

```

```

R6#show ipv6 pim vrf MC neighbor | begin ^Neighbor
Neighbor Address      Interface      Uptime      Expires      Mode DR pri
::FFFF:213.12.12.12   Lspvif0       1d03h       00:01:17 B G   DR 1

```

```

R7#show ipv6 pim vrf MC neighbor | begin ^Neighbor
Neighbor Address      Interface      Uptime      Expires      Mode DR pri
FE80::2               Gi2.527       1d09h       00:01:37 B G     1
::FFFF:213.12.12.12   Lspvif0       1d08h       00:01:15 B G   DR 1

```

```

R8#show ipv6 pim vrf MC neighbor | begin ^Neighbor
Neighbor Address      Interface      Uptime      Expires      Mode DR pri
::FFFF:213.12.12.12   Lspvif0       1d08h       00:01:41 B G   DR 1

```

Checking XRv2 is a faster way to ensure none of the CSRs are sending PIM hellos onto the PMSI.

```

RP/0/0/CPU0:XRv2#show pim vrf MC ipv6 neighbor | begin Lmdt
LmdtMC
Neighbor Address      Uptime      Expires      DR pri      DR
Flags
::ffff:213.12.12.12*  1d08h       00:01:17 1          (DR)   P

```

A quick snapshot of the BGP IPv6 MVPN table is warranted. We check from CSR6; immediately we see far less information than we did for IPv4. The answer is twofold. First, the network is segmented with the top three PEs not having reachability to the bottom three PEs. Recall that these Type-1 I-PMSI routes have a standard unicast RT in accordance with the RT-export policies of the VRF. Normal RT rules dictate a router cannot accept VPN routes that are not imported locally. The debugs confirm this basic fact. Other dynamic RT's not destined to CSR6 are also rejected, but that is the normal operation for BGP c-mcast signaling.

```

R6#show bgp ipv6 mvpn vrf MC | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 213:6 (default for vrf MC)
*> [1][213:6][213.6.6.6]/12
      ::                                32768 ?
*>i [1][213:6][213.7.7.7]/12
      213.7.7.7                        0      100      0 ?
*>i [1][213:6][213.8.8.8]/12
      213.8.8.8                        0      100      0 ?
*>i [7][213:6][213][2001:10:4:6::4][FF33::1]/46
      213.7.7.7                        0      100      0 ?

```

```

R6#debug bgp ipv6 mvpn updates in
R6#clear bgp ipv6 mvpn * soft

```

```

BGP(16): 213.12.12.12 rcvd [1][213:1][213.1.1.1]/12 -- DENIED due to:
extended community not supported;
BGP(16): 213.12.12.12 rcvd [1][213:5][213.5.5.5]/12 -- DENIED due to:
extended community not supported;
BGP(16): 213.12.12.12 rcvd [1][213:12][213.12.12.12]/12 -- DENIED due to:
extended community not supported;

```

The table is also missing a Type-3 S-PMSI AD route for the SSM group FF33::1. This is potentially confusing, because for IPv4, once CSR7 received a Type-7 SPT join for the SSM group we tested, it immediately issued a Type-3 S-PMSI announcement. In this case, CSR6 receives a Type-7 indicating interest in C(S,G) pair (2001:10:4:6::4, FF33::1) but does not originate a Type-3 S-PMSI AD route. Again, this is because this particular IPv6 C(S,G) is not candidate for data MDT optimization per the ACL rules in the VRF. As a quick test, if we add the C-G to the ACL, even without clearing anything, CSR6 immediately creates a Type-3 S-PMSI route to announce the presence of an S-PMSI (specifically, a tunnel-type 2 mLDP P2MP transport) for this group.

```

R6(config-ipv6-acl)#permit ipv6 any ff33::1/128

```

```

BGP(16): nettable_walker [3][213:6][2001:10:4:6::4][FF33::1][213.6.6.6]/46
route sourced locally
BGP(16): (base) 213.12.12.12 send UPDATE (format)
[3][213:6][2001:10:4:6::4][FF33::1][213.6.6.6]/46, next 213.6.6.6, metric 0,
path Local, extended community RT:213:6
BGP: 213.12.12.12 Next hop is our own address 213.6.6.6

```

Before continuing, the ACL entry is removed (not shown). A quick look on CSR1 confirms that PEs CSR6, CSR7, and CSR8 are outside of the MVPN, but PEs XRv2 and CSR5 are within it.

```

R1#show bgp ipv6 mvpn vrf MC | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 213:1 (default for vrf MC)
*> [1][213:1][213.1.1.1]/12

```

```

:: 32768 ?
*>i [1] [213:1] [213.5.5.5]/12
213.5.5.5 0 100 0 ?
*>i [1] [213:1] [213.12.12.12]/12
213.12.12.12 100 0 i

```

Looking specifically at IPv6 SSM, both CSR2 and CSR3 originated C(S,G) joins towards their PEs requesting traffic for (2001:10:4:6::4, FF33::1). CSR7 and CSR8 both originated Type-7 SPT joins for this C(S,G) and sent it to XRv2, the RR. The RR picks CSR7 because it has a lower IGP cost to the BGP next-hop.

```

RP/0/0/CPU0:XRv2#show bgp ipv6 mvpn rd 213:6 route-type 7 | begin Network
Network Next Hop Metric LocPrf Weight Path
Route Distinguisher: 213:6
*>i [7] [213:6] [213] [128] [2001:10:4:6::4] [128] [ff33::1]/376
213.7.7.7 0 100 0 ?
* i 213.8.8.8 0 100 0 ?

```

Looking specifically at this best-path on CSR6, we can see why CSR6 imported it. The dynamic RT assigned to the C-S is what CSR7 used to set the dynamic RT on the Type-7 route it advertised towards CSR6. This way, only CSR6 imports it. This behavior is protocol agnostic and is true for IPv4 and IPv6.

```

R6#show bgp ipv6 mvpn vrf MC route-type 7 213:6 213 2001:10:4:6::4 FF33::1
BGP routing table entry for [7] [213:6] [213] [2001:10:4:6::4] [FF33::1]/46,
version 34
Paths: (1 available, best #1, table MVPNV6-BGP-Table)
Not advertised to any peer
Refresh Epoch 1
Local
213.7.7.7 (metric 20) from 213.12.12.12 (213.12.12.12)
Origin incomplete, metric 0, localpref 100, valid, internal, best
Extended Community: RT:213.6.6.6:1
Originator: 213.7.7.7, Cluster list: 213.12.12.12
rx pathid: 0, tx pathid: 0x0

```

```

R6#show bgp vpnv6 unicast vrf MC 2001:10:4:6::/64
BGP routing table entry for [213:6]2001:10:4:6::/64, version 25
Paths: (1 available, best #1, table MC)
Advertised to update-groups:
2
Refresh Epoch 1
Local
:: (via vrf MC) from 0.0.0.0 (213.6.6.6)
Origin incomplete, metric 0, localpref 100, weight 32768, valid,
sourced, best
Extended Community: RT:213:6 MVPN AS:213:0.0.0.0 MVPN VRF:213.6.6.6:1
mpls labels in/out 6001/nolabel(MC)

```

```
rx pathid: 0, tx pathid: 0x0
```

Before sending traffic, we can verify that the C-PIM control-plane is correct. The group is mapped to the I-PMSI, or the mLDP MP2MP tree, which explains the absence of any 'Y' or 'y' flags on these groups. The "data MDT" is not in use here. Again, the big 'G' means the entry was installed from having received a BGP Type 6 or 7 route. The little 'g' means the entry was installed from having sent one. Packets exit to the PMSI on CSR6 and enter from it on CSR7 and CSR8.

```
R6#show ipv6 mroute vrf MC FF33::1 | begin \  
(2001:10:4:6::4, FF33::1), 00:14:12/never, flags: sTG  
  Incoming interface: GigabitEthernet2.546  
  RPF nbr: 2001:10:4:6::4  
  Immediate Outgoing interface list:  
    Lspvif0, Forward, 00:14:12/never
```

```
R7#show ipv6 mroute vrf MC FF33::1 | begin \  
(2001:10:4:6::4, FF33::1), 00:47:32/never, flags: sTIg  
  Incoming interface: Lspvif0  
  RPF nbr: ::FFFF:213.6.6.6  
  Immediate Outgoing interface list:  
    GigabitEthernet2.527, Forward, 00:47:32/never
```

```
R8#show ipv6 mroute vrf MC FF33::1 | begin \  
(2001:10:4:6::4, FF33::1), 01:55:23/never, flags: sTIg  
  Incoming interface: Lspvif0  
  RPF nbr: ::FFFF:213.6.6.6  
  Immediate Outgoing interface list:  
    GigabitEthernet2.538, Forward, 01:55:23/never
```

When CSR4 starts sending traffic, there is no Type-5 Source Active AD message. This is because the group is SSM and CSR6 has already received a Type-7 join for it. We can quickly verify if the traffic is working by checking the counters on CSR6, CSR7, and CSR8.

```
R4#ping FF33::1 rep 10000  
Output Interface: GigabitEthernet2.546
```

```
R6#show ipv6 mroute vrf MC FF33::1 count  
Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second  
Other counts:      Total/RPF failed/Other drops(OIF-null, rate-limit etc)  
VRF MC  
  53 routes, 6 (*,G)s, 46 (*,G/m)s  
Group: FF33::1  
  Source: 2001:10:4:6::4,  
    SW Forwarding: 0/0/0/0, Other: 0/0/0  
    HW Forwarding: 87/0/118/0, Other: 0/0/0  
Totals - Source count: 1, Packet count: 87
```

```
R7#show ipv6 mroute vrf MC FF33::1 count
Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second
Other counts:      Total/RPF failed/Other drops(OIF-null, rate-limit etc)
VRF MC
  55 routes, 7 (*,G)s, 47 (*,G/m)s
Group: FF33::1
  Source: 2001:10:4:6::4,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 94/0/126/0, Other: 0/0/0
  Totals - Source count: 1, Packet count: 94
```

```
R8#show ipv6 mroute vrf MC FF33::1 count
Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second
Other counts:      Total/RPF failed/Other drops(OIF-null, rate-limit etc)
VRF MC
  55 routes, 7 (*,G)s, 47 (*,G/m)s
Group: FF33::1
  Source: 2001:10:4:6::4,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 95/0/126/0, Other: 0/0/0
  Totals - Source count: 1, Packet count: 95
```

For variety, we will capture inbound on CSR7 facing the mLDP MP2MP root of XRv4. Even though this isn't the shortest path back to CSR6, we know that we are not using S-PMSI since no Type 3 routes exist within this MVPN instance anywhere currently. The incoming label is 0x1B5E (7006) which is the correct MDT 0 local downstream label allocated by CSR7 for the I-PMSI. I also highlighted the IPv6 source/destination addresses as proof.

```
R7#show monitor capture CAP buffer dump
2
0000: 005056A9 EA770050 56A9DE77 81000DF6 .PV..w.PV..w....
0010: 884701B5 E03D0000 213F6000 0000003C .G...=...!?'`....<
0020: 3A3F2001 00100004 00060000 00000000 :? .....
0030: 0004FF33 00000000 00000000 00000000 ...3.....
0040: 00018000 9DF5263C 014F4F50 51525354 .....&<.OOPQRST
0050: 55565758 595A5B5C 5D5E5F60 61626364 UVWXYZ[\]^_`abcd
0060: 65666768 696A6B6C 6D6E6F70 71727374 efghijklmnopqrst
0070: 75767778 797A7B7C 7D7E7F80 8182 uvwxyz{|}~....
```

```
R7#show mpls mldp database opaque_type mdt 213:213 0 | section Upstream
Upstream client(s) :
  213.14.14.14:0    [Active]
    Expires        : Never          Path Set ID   : 1
    Out Label (U)   : 94003          Interface     : GigabitEthernet2.574*
    Local Label (D) : 7006           Next Hop      : 213.7.14.14
```

Last, we will test IPv6 ASM. Since the ACL we defined under the VRF accounts for our ASM groups which use Embedded RP, at some point we expect to see some additional route types, like Type-3 and Type-5

floating around the network. Since CSR3 is the only router requesting multicast for this group, let's trace the signaling path quickly. CSR3 sends a C(*,G) join to CSR8 who advertises a Type 6 Shared Tree Join towards the C-RP, which is CSR2. Notice the little 'g' to signify the C(*,G) entry was installed due to having sent a BGP c-mcast signaling route. Also note that the "show" command for a Type 6 route asks for the "multicast source"; this is the C-RP address. The "RT-import" process is also shown for completeness.

```
R8#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 | begin \(
(*, FF7E:240:2001:10:2:7:0:1), 02:18:24/never, RP 2001:10:2:7::2, flags: SCJg
  Incoming interface: Lspvif0
  RPF nbr: ::FFFF:213.7.7.7
  Immediate Outgoing interface list:
    GigabitEthernet2.538, Forward, 02:18:24/never
```

```
R8#show bgp ipv6 mvpn vrf MC route-type 6 213:7 213 2001:10:2:7::2
FF7E:240:2001:10:2:7:0:1
BGP routing table entry for
[6][213:7][213][2001:10:2:7::2][FF7E:240:2001:10:2:7:0:1]/46, version 40
Paths: (1 available, best #1, table MVPNV6-BGP-Table)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  Local
    :: from 0.0.0.0 (213.8.8.8)
    Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
    Extended Community: RT:213.7.7.7:1
    rx pathid: 0, tx pathid: 0x0
```

```
R8#show bgp vpnv6 unicast vrf MC 2001:10:2:7::2/64
BGP routing table entry for [213:8]2001:10:2:7::/64, version 25
Paths: (1 available, best #1, table MC)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from [213:7]2001:10:2:7::/64 (global)
    ::FFFF:213.7.7.7 (metric 20) (via default) from 213.12.12.12
(213.12.12.12)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    Extended Community: RT:213:7 MVPN AS:213:0.0.0.0 MVPN VRF:213.7.7.7:1
    Originator: 213.7.7.7, Cluster list: 213.12.12.12
    mpls labels in/out nolabel/7001
    rx pathid: 0, tx pathid: 0x0
```

XRv2 reflects this to CSR7, who imports it based on the dynamic RT that was set by CSR8, highlighted above. Again, CSR8 knows this because the ordinary VPNv6 unicast route to the RP contained this dynamic RT as an extended community when IPv6 MVPN AF was activated. The big 'G' on the C(*,G) state indicates this entry was installed due to having received a BGP Type-6 route.

```

R8#show bgp ipv6 mvpn vrf MC route-type 6 213:7 213 2001:10:2:7::2
BGP routing table entry for
[6][213:7][213][2001:10:2:7::2][FF7E:240:2001:10:2:7:0:1]/46, version 40
Paths: (1 available, best #1, table MVPNV6-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    213.8.8.8 (metric 20) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:213.7.7.7:1
      Originator: 213.8.8.8, Cluster list: 213.12.12.12
      rx pathid: 0, tx pathid: 0x0

```

```

R7#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 | begin \
(*, FF7E:240:2001:10:2:7:0:1), 00:43:53/never, RP 2001:10:2:7::2, flags: SG
  Incoming interface: GigabitEthernet2.527
  RPF nbr: FE80::2
  Immediate Outgoing interface list:
    Lspvif0, Forward, 00:43:53/never

```

CSR4 starts sending traffic to this group. The first thing that happens is that CSR6 sends a C-PIM register message to the C-RP, which is CSR2. Many of the routes are running “debug bgp ipv6 mvpn updates” and “debug ipv6 pim group FF7E:240:2001:10:2:7:0:1” for the upcoming verification.

```

R4#ping FF7E:240:2001:10:2:7:0:1 repeat 10000
Output Interface: GigabitEthernet2.546

```

```

! CSR6
IPv6 PIM: [MC] (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) Start registering to
2001:10:2:7::2

```

Upon receipt, CSR2 creates C(S,G) state and forwards the decapsulated register message down the shared tree towards CSR8. At present, CSR7’s path to CSR8 for multicast traffic using the I-PMSI (mLDP MP2MP tree), so these initial packets use this as transport. After some more register messages, CSR2 sends the register stop once the trees are built. CSR2 also sends a C(S,G) prune to CSR7 since CSR2 is not in the SPT transit path.

```

! CSR2
IPv6 PIM: (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) Received Register from
2001:10:4:6::6
IPv6 PIM: (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) Create entry
IPv6 PIM: (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) RPF changed from ::/- to
FE80::7/GigabitEthernet2.527
IPv6 PIM: (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) J/P adding Join on
GigabitEthernet2.527

```



```
IPv6 PIM: (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) Received Register from
2001:10:4:6::6
IPv6 PIM: (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) Set alive timer to 210
sec
IPv6 PIM: J/P entry: Prune root: 2001:10:4:6::4 group:
FF7E:240:2001:10:2:7:0:1 Sender: FE80::7 flags: RPT S
IPv6 PIM: (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) Received Register from
2001:10:4:6::6
IPv6 PIM: (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) Set alive timer to 210
sec
IPv6 PIM: (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) Send Register-Stop to
2001:10:4:6::6
```

CSR7 adds C(S,G) state receiving the join from CSR2, and then sends a Type-7 SPT join to into the BGP network targeting CSR6. The entry uses the S-PMSI (data MDT).

```
R7#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 2001:10:4:6::4 | begin \
(2001:10:4:6::4, FF7E:240:2001:10:2:7:0:1), 00:04:15/00:02:48, flags: STYg
Incoming interface: Lspvif0, MDT: [2147483649 (0x80000001),
213.6.6.6]/never
RPF nbr: ::FFFF:213.6.6.6
Outgoing interface list: Null
```

CSR8 does the exact same thing, except it has a legitimate OIL. We will examine the debugging on CSR8 specifically. The traffic is detected on the I-PMSI, which was multicast coming down the shared tree which was originally encapsulated in the C-PIM register messages. CSR8 responds with a BGP Type-7 SPT join towards CSR6.

```
R8#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 2001:10:4:6::4 | begin \
(2001:10:4:6::4, FF7E:240:2001:10:2:7:0:1), 00:26:56/00:01:14, flags: SJTYgQ
Incoming interface: Lspvif0, MDT: [2147483649 (0x80000001),
213.6.6.6]/never
RPF nbr: ::FFFF:213.6.6.6
Inherited Outgoing interface list:
GigabitEthernet2.538, Forward, 02:55:05/never
```

```
! CSR8
IPv6 PIM: [MC] (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) Signal present on
Lspvif0
IPv6 PIM: [MC] (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) Create entry
IPv6 PIM: [MC] (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1/128) Lspvif0 MRIB
modify NS
IPv6 PIM: [MC] (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) RPF changed from
::-/- to ::FFFF:213.6.6.6/Lspvif0

BGP[16] MVPN: add c-route, type 7, bs len 0 asn=0, rd=213:8,
source=2001:10:4:6::4/16,
group=FF7E:240:2001:10:2:7:0:1/16,
```

```
nexthop>::FFFF:213.6.6.6,  
len left = 0
```

The debugs show that PIM and BGP are cooperating on CSR8. The Type -7 is sent up the RPF towards the source as expected. The RT applied to this Type-7 was copied from the VPNv6 route to the C-source.

```
!CSR8  
BGP: MVPN(16) create local route  
[7][213:6][213][2001:10:4:6::4][FF7E:240:2001:10:2:7:0:1]/46  
IPv6 PIM: [MC] (2001:10:4:6::4,FF7E:240:2001:10:2:7:0:1) Originate BGP C-  
route to ::FFFF:213.6.6.6  
  
BGP(16): nettable_walker  
[7][213:6][213][2001:10:4:6::4][FF7E:240:2001:10:2:7:0:1]/46 route sourced  
locally  
BGP(16): (base) 213.12.12.12 send UPDATE (format)  
[7][213:6][213][2001:10:4:6::4][FF7E:240:2001:10:2:7:0:1]/46, next 213.8.8.8,  
metric 0, path Local, extended community RT:213.6.6.6:1  
  
R8#show bgp vpnv6 unicast vrf MC 2001:10:4:6::/64  
BGP routing table entry for [213:8]2001:10:4:6::/64, version 27  
Paths: (1 available, best #1, table MC)  
  Not advertised to any peer  
  Refresh Epoch 1  
  Local, imported path from [213:6]2001:10:4:6::/64 (global)  
    ::FFFF:213.6.6.6 (metric 30) (via default) from 213.12.12.12  
(213.12.12.12)  
    Origin incomplete, metric 0, localpref 100, valid, internal, best  
    Extended Community: RT:213:6 MVPN AS:213:0.0.0.0 MVPN VRF:213.6.6.6:1  
    Originator: 213.6.6.6, Cluster list: 213.12.12.12  
    mpls labels in/out nlabel/6001  
    rx pathid: 0, tx pathid: 0x0
```

This details what happens when a Type 7 is received and the router is not importing the dynamic RT. This is the Type-7 generated by CSR7 to join the SPT, yet it was denied since CSR8 is not importing RT:213.6.6.6:1.

```
! CSR8  
BGP(16): 213.12.12.12 rcvd UPDATE w/ attr: nexthop 213.7.7.7, origin ?,  
localpref 100, metric 0, originator 213.7.7.7, clusterlist 213.12.12.12,  
extended community RT:213.6.6.6:1  
BGP(16): 213.12.12.12 rcvd  
[7][213:6][213][2001:10:4:6::4][FF7E:240:2001:10:2:7:0:1]/46 -- DENIED due  
to: extended community not supported;
```

CSR6 receives one of the joins (XRv2 only forwards one, which is fine), creates a Type-5 Source Active message, under the assumption that a source behind it has started sending traffic. CSR7 and CSR8 will both install this in their BGP RIBs, but only CSR8 will bind it to the C(S,G) state entry.

```
! CSR5
BGP(16): 213.12.12.12 rcvd
[7] [213:6] [213] [2001:10:4:6::4] [FF7E:240:2001:10:2:7:0:1]/46
BGP(16): skip vrf default table RIB route
[7] [213:6] [213] [2001:10:4:6::4] [FF7E:240:2001:10:2:7:0:1]/46
BGP(16): add RIB route
[7] [213:6] [213] [2001:10:4:6::4] [FF7E:240:2001:10:2:7:0:1]/46

BGP(16): nettable_walker
[5] [213:6] [2001:10:4:6::4] [FF7E:240:2001:10:2:7:0:1]/42 route sourced locally
BGP(16): delete RIB route
[5] [213:6] [2001:10:4:6::4] [FF7E:240:2001:10:2:7:0:1]/42
BGP(16): (base) 213.12.12.12 send UPDATE (format)
[5] [213:6] [2001:10:4:6::4] [FF7E:240:2001:10:2:7:0:1]/42, next 213.6.6.6,
metric 0, path Local, extended community RT:213:6
```

Once registration is complete, CSR6 notifies the receivers about the availability of S-PMSI so they (CSR7) can signal the P2MP tree back to CSR6. This is the result of specifying the “mdt data” ACL under the VRF.

```
! CSR6
BGP(16): nettable_walker
[3] [213:6] [2001:10:4:6::4] [FF7E:240:2001:10:2:7:0:1] [213.6.6.6]/46 route
sourced locally
BGP(16): (base) 213.12.12.12 send UPDATE (format)
[3] [213:6] [2001:10:4:6::4] [FF7E:240:2001:10:2:7:0:1] [213.6.6.6]/46, next
213.6.6.6, metric 0, path Local, extended community RT:213:6
```

CSR8 is informed about an available S-PMSI for this multicast flow by CSR6. The debugs reveal the contents of the S-PMSI route, which shows a tunnel-type 2 (mLDP P2MP) and a root of 213.6.6.6 in hex. CSR8 sends mLDP label mapping messages, not shown, to advertise its downstream label mappings to XRv4 and ultimately towards CSR6. When this process is complete, mLDP reports it via syslog.

```
! CSR6
BGP(16): 213.12.12.12 rcvd UPDATE w/ attr: nexthop 213.6.6.6, origin ?,
localpref 100, metric 0, originator 213.6. 6.6, clusterlist 213.12.12.12,
community no-export, extended community RT:213:6
BGP(16): 213.12.12.12 rcvd
[3] [213:6] [2001:10:4:6::4] [FF7E:240:2001:10:2:7:0:1] [213.6.6.6]/46
BGP(16): skip vrf default table RIB route
[3] [213:6] [2001:10:4:6::4] [FF7E:240:2001:10:2:7:0:1] [213.6.6.6]/46
BGP(16): add RIB route
[3] [213:7] [2001:10:4:6::4] [FF7E:240:2001:10:2:7:0:1] [213.6.6.6]/46
```

```
Flags: 0x0, Tunnel type: 2, label: exp-null, tunnel parameters: 0600 0104
FFFFFFFFFFFFFFD506 0606 000E 0200 0B00 0 213 0000 0213 FFFFFFFFFFFFFFFF8000
0001
%MLDP-5-ADD_BRANCH: [mdt 213:213 2147483649] Root: 213.6.6.6, Add P2MP branch
MDT remote label
```

A quick check of the C-MRIB state information on CSR6, CSR7, and CSR8 is warranted to ensure the proper flags are set. CSR6 is the root of the mLDP P2MP tree, which is why it specifies “MDT TX” in the C-MRIB state. The MDT ID is 2147483649, which is non-zero, indicating a P2MP tree. The ‘F’ flag means this router was the C-PIM DR responsible for registering sources with the C-RP. The little ‘y’ indicates that this router is sending (TX) to a data MDT tree. The big ‘G’ indicates this route was installed from having received a BGP Type 6/7 route (7 in this case). The little ‘q’ indicates that this router originated a Type 5 Source Active message for this C(S,G). The RPF information appears correct also; the device closest to the source, CSR6, points upstream towards the source, and the remote PEs point to the BGP next-hop towards the source over the S-PMSI that was built as a result of a Type-3 S-PMSI BGP message.

```
R6#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 2001:10:4:6::4 | begin \
(2001:10:4:6::4, FF7E:240:2001:10:2:7:0:1), 00:43:22/00:02:27, flags: SFTyGq
Incoming interface: GigabitEthernet2.546
RPF nbr: 2001:10:4:6::4
MDT TX nr: 2147483649 (0x80000001) LSM-ID: 0x5
Immediate Outgoing interface list:
Lspvif0, Forward, 00:43:22/never
```

CSR7 shows the MDT as part of the incoming interface, which makes sense. The big ‘Y’ is set to indicate having joined a Data MDT, and the little ‘g’ is set to indicate having installed this entry by sending a BGP Type 7 route.

```
R7#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 2001:10:4:6::4 | begin \
(2001:10:4:6::4, FF7E:240:2001:10:2:7:0:1), 00:43:40/00:02:12, flags: STYg
Incoming interface: Lspvif0, MDT: [2147483649 (0x80000001),
213.6.6.6]/never
RPF nbr: ::FFFF:213.6.6.6
Outgoing interface list: Null
```

CSR8 has the “Yg” flags like CSR7 but with an additional big ‘Q’, signifying that this entry is mapped to a Type 5 Source Active route for the same C(S,G). Because CSR7 doesn’t have an interested receiver, the Type-5 isn’t bound to the C(S,G) state. CSR7 remains joined on the SPT since the RP is behind it.

```
R8#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 2001:10:4:6::4 | begin \
(2001:10:4:6::4, FF7E:240:2001:10:2:7:0:1), 00:43:42/00:02:08, flags: SJTYgQ
Incoming interface: Lspvif0, MDT: [2147483649 (0x80000001),
213.6.6.6]/never
RPF nbr: ::FFFF:213.6.6.6
Inherited Outgoing interface list:
GigabitEthernet2.538, Forward, 03:11:51/never
```

Validating the packet counters shows us that forwarding appears to be working.

```
R6#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 2001:10:4:6::4 count
Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second
Other counts:      Total/RPF failed/Other drops(OIF-null, rate-limit etc)
VRF MC
  54 routes, 6 (*,G)s, 46 (*,G/m)s
Group: FF7E:240:2001:10:2:7:0:1
  Source: 2001:10:4:6::4,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 1600/0/117/0, Other: 2/2/0
  Totals - Source count: 1, Packet count: 1600
```

```
R7#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 2001:10:4:6::4 count
Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second
Other counts:      Total/RPF failed/Other drops(OIF-null, rate-limit etc)
VRF MC
  56 routes, 7 (*,G)s, 47 (*,G/m)s
Group: FF7E:240:2001:10:2:7:0:1
  Source: 2001:10:4:6::4,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 1603/0/125/0, Other: 1/1/0
  Totals - Source count: 1, Packet count: 1603
```

```
R8#show ipv6 mroute vrf MC FF7E:240:2001:10:2:7:0:1 2001:10:4:6::4 count
Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second
Other counts:      Total/RPF failed/Other drops(OIF-null, rate-limit etc)
VRF MC
  56 routes, 7 (*,G)s, 47 (*,G/m)s
Group: FF7E:240:2001:10:2:7:0:1
  Source: 2001:10:4:6::4,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 1606/0/126/0, Other: 0/0/0
  Totals - Source count: 1, Packet count: 1606
```

Because CSR7 joined the P2MP tree also, we expect it to be receiving traffic directly from XRv3 now, not XRv4, since the shortest path to the P2MP root is via XRv3. This is more optimal for bandwidth, but not for state. A quick capture inbound on CSR7 from XRv3 confirms this. We also verify the proper downstream label is used, which is 7010 (0x1B62). The source/destination addresses are also highlighted.

```
R7#show monitor capture CAP buffer dump
0
0000: 005056A9 EA770050 56A9DB37 81000DF5 .PV..w.PV..7....
0010: 884701B6 203E0000 213F6000 0000003C .G.. >...!?'`....<
0020: 3A3F2001 00100004 00060000 00000000 :? .....
0030: 0004FF7E 02402001 00100002 00070000 ...~.@ .....
```

```

0040: 00018000 C7101640 06CCCCCD CECFD0D1 .....@.....
0050: D2D3D4D5 D6D7D8D9 DADBDCDD DEDFE0E1 .....
0060: E2E3E4E5 E6E7E8E9 EAEBECED EEEFF0F1 .....
0070: F2F3F4F5 F6F7F8F9 FAFBFCFD FEFF .....

```

```

R7#show mpls mldp database opaque_type mdt 213:213 2147483649 | section
Upstream

```

```

Upstream client(s) :
  213.13.13.13:0    [Active]
    Expires        : Never          Path Set ID   : 5
    Out Label (U)  : None           Interface     : GigabitEthernet2.573*
    Local Label (D): 7010           Next Hop      : 213.7.13.13

```

We also expect CSR8 to be receiving this flow. We will capture outbound on CSR8 within the VRF so we can see the raw IPv6 packets being forwarded to the customer without MPLS encapsulation. We clearly see the 0x86DD ethertype to represent IPv6. The IP protocol number is 0x3A which is 58 for IPv6 ICMP. The parser gives you some summarized information also.

```

R8#show monitor capture CAP buffer detailed
  1  118      1.352979      2001:::0004      ->      FF7E:::0001      IPv6-ICMP
0000: 33330000 00010050 56A9FB1C 8100DD2  33.....PV.....
0010: 86DD6000 0000003C 3A3C2001 00100004  ..`....<:< .....
0020: 00060000 00000000 0004FF7E 02402001  .....~.@ .
0030: 00100002 00070000 00018000 36FB1640  .....6...@

```

Just to verify that the top routers have broken connectivity (and for extra review), we can clear BGP updates for IPv6 MVPN inbound on CSR5 and look at the debugs. The RT values on BGP AD routes (Types 1-5) are regular VPN RTs (213:6, etc) that CSR5 is not importing. The dynamic RT's apply only to BGP c-mcast signaling (Type 6-7) which are also not imported by CSR5 in this topology. I've grouped the BGP routes in order from 1 to 7 for easier reading. CSR5 cannot establish I-PMSI with CSR6, CSR7, or CSR8 due to the Type-1 routes not being imported.

```

! CSR5
BGP(16): 213.12.12.12 rcvd UPDATE w/ attr: nexthop 213.7.7.7, origin ?,
localpref 100, metric 0, originator 213.7.7.7, clusterlist 213.12.12.12,
community no-export, extended community RT:213:7
BGP(16): 213.12.12.12 rcvd [1][213:7][213.7.7.7]/12 -- DENIED due to:
extended community not supported;

```

```

BGP(16): 213.12.12.12 rcvd UPDATE w/ attr: nexthop 213.6.6.6, origin ?,
localpref 100, metric 0, originator 213.6.6.6, clusterlist 213.12.12.12,
community no-export, extended community RT:213:6
BGP(16): 213.12.12.12 rcvd [1][213:6][213.6.6.6]/12 -- DENIED due to:
extended community not supported;

```

```
BGP(16): 213.12.12.12 rcvd UPDATE w/ attr: nexthop 213.8.8.8, origin ?,
localpref 100, metric 0, originator 213.8.8.8, clusterlist 213.12.12.12,
community no-export, extended community RT:213:8
BGP(16): 213.12.12.12 rcvd [1][213:8][213.8.8.8]/12 -- DENIED due to:
extended community not supported;
```

The Type-3 S-PMSI route and Type-5 Source Active are also not imported since CSR5 does not have unicast reachability to CSR6, so it doesn't make sense to import it. The RT is the same as the Type-1 routes shown above.

```
! CSR5
BGP(16): 213.12.12.12 rcvd UPDATE w/ attr: nexthop 213.6.6.6, origin ?,
localpref 100, metric 0, originator 213.6.6.6, clusterlist 213.12.12.12,
community no-export, extended community RT:213:6
BGP(16): 213.12.12.12 rcvd
[3][213:6][2001:10:4:6::4][FF7E:240:2001:10:2:7:0:1][213.6.6.6]/46 -- DENIED
due to: extended community not supported;

BGP(16): 213.12.12.12 rcvd UPDATE w/ attr: nexthop 213.6.6.6, origin ?,
localpref 100, metric 0, originator 213.6.6.6, clusterlist 213.12.12.12,
community no-export, extended community RT:213:6
BGP(16): 213.12.12.12 rcvd
[5][213:6][2001:10:4:6::4][FF7E:240:2001:10:2:7:0:1]/42 -- DENIED due to:
extended community not supported;
```

The BGP c-mcast signaling routes use the dynamic RTs. Seeing these rejected can be normal in many cases since the RT specifically targets a device for a C(*,G) or C(S,G) Join. The prune is expressed by withdrawing these messages.

```
! CSR5
BGP(16): 213.12.12.12 rcvd UPDATE w/ attr: nexthop 213.8.8.8, origin ?,
localpref 100, metric 0, originator 213.8.8.8, clusterlist 213.12.12.12,
extended community RT:213.7.7.7:1
BGP(16): 213.12.12.12 rcvd
[6][213:7][213][2001:10:2:7::2][FF7E:240:2001:10:2:7:0:1]/46 -- DENIED due
to: extended community not supported;

BGP(16): 213.12.12.12 rcvd UPDATE w/ attr: nexthop 213.7.7.7, origin ?,
localpref 100, metric 0, originator 213.7.7.7, clusterlist 213.12.12.12,
extended community RT:213.6.6.6:1
BGP(16): 213.12.12.12 rcvd
[7][213:6][213][2001:10:4:6::4][FF7E:240:2001:10:2:7:0:1]/46 -- DENIED due
to: extended community not supported;
BGP(16): 213.12.12.12 rcvd [7][213:6][213][2001:10:4:6::4][FF33::1]/46 --
DENIED due to: extended community not supported;
```

Last, we will examine OAM for MP2MP and P2MP trees in this profile. Unlike Profile 12 where we were limited only to unidirectional P2MP trees (the root specified had to be local), with MP2MP pings we can test from any endpoint; the tree is bidirectional. We specify the FEC and opaque type and can quickly discover all the endpoints in the MVPN on this MDT.

```
R6#show mpls mldp database summary
```

LSM ID	Type	Root	Decoded Opaque Value	Client
Cnt.				
15	P2MP	213.7.7.7	[mdt 213:213 1]	1
13	MP2MP	213.14.14.14	[mdt 213:213 0]	1

```
R6#ping mpls mldp mp2mp 213.14.14.14 mdt 213:213 0
```

```
mp2mp Root node addr 213.14.14.14
```

```
Opaque type MDT, oui:index 0x213:0213, mdtnum 0
```

```
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,  
timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200  
msec:
```

```
[snip]
```

```
Request #1
```

```
! reply addr 213.12.13.12
```

```
! reply addr 213.7.14.7
```

```
! reply addr 213.1.14.1
```

```
! reply addr 213.8.14.8
```

```
! reply addr 213.5.14.5
```

```
Round-trip min/avg/max = 20/107/173 ms
```

We can also check the S-PMSI P2MP tree to CSR7, but to do so, we have to test it from CSR7. We should be able to verify connectivity to CSR6 and CSR8 since they have joined this P2MP tree. CSR6 is also the root of another P2MP tree but for brevity, we do not verify that.

```
R7#show mpls mldp database summary
```

LSM ID	Type	Root	Decoded Opaque Value	Client
Cnt.				
15	P2MP	213.7.7.7	[mdt 213:213 1]	3
13	MP2MP	213.14.14.14	[mdt 213:213 0]	1

```
R7#ping mpls mldp p2mp 213.7.7.7 mdt 213:213 1
```

```
p2mp Root node addr 213.7.7.7
```

```
Opaque type MDT, oui:index 0x213:0213, mdtnum 1
```

```
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,  
timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200  
msec:
```

```
[snip]
```

```
Request #1
```

```
! reply addr 213.6.13.6
```



```
! reply addr 213.8.14.8
```

```
Round-trip min/avg/max = 122/155/189 ms
```

Additional Reading – Reference configurations “mvpn-13”

28.12 Profile 14: Partitioned MDT – MLDP P2MP – BGP-AD – BGP C-mcast signaling

For this test, begin with the MVPN Profile 12 configuration where all of the MVPN BGP AD is set up already, plus the P2MP trees are built (no MP2MP trees anywhere). We need to make sure the MDT trees are “partitioned” (described later) versus “default”. We also need to enable a “strict” RPF check (described later). Removing the default MDT also removes the data MDT configuration, so it must be added again.

```
! Basic XE configuration to enable partitioned MDT
vrf definition MC
  address-family ipv4
    no mdt default mpls mldp p2mp
    mdt partitioned mldp p2mp
    mdt data mpls mldp 10 list ACL_DATA_MDT immediate-switch
    mdt strict-rpf interface
  address-family ipv6
    no mdt default mpls mldp p2mp
    mdt partitioned mldp p2mp
    mdt data mpls mldp 10 list ACL_DATA_MDT_IPV6 immediate-switch
    mdt strict-rpf interface

! Basic XR configuration to enable partitioned MDT
multicast-routing
  vrf MC
    address-family ipv4
      no mdt default mldp p2mp
      mdt partitioned mldp ipv4 p2mp
      mdt data mldp 10 ACL_DATA_MDT immediate-switch
      bgp auto-discovery mldp
    address-family ipv6
      no mdt default mldp p2mp
      mdt partitioned mldp ipv4 p2mp
      mdt data mldp 10 ACL_DATA_MDT_IPV6 immediate-switch
      bgp auto-discovery mldp

route-policy RPL_MLDP
  set core-tree mldp-partitioned-p2mp
end-policy
```

BGP-AD is used to more aggressively shape what traffic is allowed to traverse the core so as to minimize flooding multicast to PEs with no interest. Specifically, there are many Type-3 S-PMSI routes that use wildcards now. A Type-3 can specify C(*,G) or C(*,*) wildcards. This is accomplished by setting the length

field to 0 in the S-PMSI route. A quick look at CSR6 shows some of these new routes. Notice that all six PEs originated a C(*,*) route saying that they are willing to construct S-PMSI trees for any multicast flow.

```
R6#show bgp ipv4 mvpn vrf MC | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
*>i  [3] [213:6] [*] [*] [213.1.1.1]/14
      213.1.1.1      0      100      0 ?
*>i  [3] [213:6] [*] [*] [213.5.5.5]/14
      213.5.5.5      0      100      0 ?
*>   [3] [213:6] [*] [*] [213.6.6.6]/14
      0.0.0.0      32768 ?
*>i  [3] [213:6] [*] [*] [213.7.7.7]/14
      213.7.7.7      0      100      0 ?
*>i  [3] [213:6] [*] [*] [213.8.8.8]/14
      213.8.8.8      0      100      0 ?
*>i  [3] [213:6] [*] [*] [213.12.12.12]/14
      213.12.12.12      100      0 i
*>i  [3] [213:6] [*] [224.0.0.13] [213.5.5.5]/18
      213.5.5.5      0      100      0 ?
*>i  [3] [213:6] [10.2.7.2] [232.0.0.5] [213.7.7.7]/22
      213.7.7.7      0      100      0 ?
```

This output shows us all three possible combinations: C(S,G), C(*,G), C(*,*). The C(S,G) is something we have seen before and this shows a router (CSR7 in this case) advertising that it is capable of delivering traffic to this C(S,G) via an S-PMSI. The same logic is true for the remaining two variations; all of them are candidates for S-PMSI tree construction. The partitioned nature of this MVPN profile means that trees are built-on demand, which means the concept of I-PMSI isn't appropriate. We can even see that CSR5 has originated an entry for group 224.0.0.13, the All-PIM-Routers group. This exists so CSR5 can distribute the BSR messaging from CSR10. The lack of an I-PMSI in general implies that we will see much more S-PMSI state as a result.

Because BSR is being distributed to all nodes (presumably), CSR5 should have an S-PMSI tree rooted at itself to serve this control traffic. We can check the mLDP database summary, but this doesn't tell us which tunnel is used for distributing this information. The BGP Type-3 C(*,224.0.0.13) route should tell us the tunnel information. Anyone importing RT 213:5 should show interest in this. We see it is a type 2 mLDP P2MP tunnel with GID value 0x1FFFF rooted at CSR5. A quick OAM check shows us that everyone else has joined this delivery tree.

```
R5#show mpls mldp database summary
LSM ID      Type      Root      Decoded Opaque Value      Client
Cnt.
E           P2MP      213.5.5.5      [gid 65536 (0x00010000)]      3
18          P2MP      213.5.5.5      [gid 131071 (0x0001FFFF)]      3
10          P2MP      213.5.5.5      [gid 131072 (0x00020000)]      1
```

```
R5#show bgp ipv4 mvpn vrf MC route-type 3 * 224.0.0.13 213.5.5.5
```

BGP routing table entry for [3][213:5][*][224.0.0.13][213.5.5.5]/18, version 275
 Paths: (1 available, best #1, table MVPNv4-BGP-Table, not advertised to EBGPeer)

```

    Advertised to update-groups:
      3
    Refresh Epoch 1
    Local
      0.0.0.0 from 0.0.0.0 (213.5.5.5)
        Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
        Community: no-export
        Extended Community: RT:213:5
        PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 17, label: exp-null,
tunnel parameters: 0600 0104 D505 0505 0007 0100 0400 01FF FF

```

```

R5#ping mpls mldp p2mp 213.5.5.5 hex 0x01 0001FFFF
p2mp Root node addr 213.5.5.5
Opaque type hex value (0x1), num hex digits 4
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
  timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]

```

```

Request #1
! reply addr 213.12.13.12
! reply addr 213.7.13.7
! reply addr 213.6.13.6
! reply addr 213.1.14.1
! reply addr 213.8.14.8

```

Round-trip min/avg/max = 32/80/125 ms

First, we will test IPv4 SSM, since it already has a Type3 C(S,G) S-PMSI route in the network. CSR7 originated this because CSR2 is the source. CSR6 and CSR8 originated Type-7 SPT joins towards CSR7, and XRv2 picked CSR6 as best-path due to the IGP metric being lower. As long as CSR7 receives one of them, that is enough. CSR7 imports the route based on its dynamic RT matching the VPNv4 MVPN RT assigned locally. This is how CSR7 knows that the Type-7 join is destined for it.

```

RP/0/0/CPU0:XRv2#show bgp ipv4 mvpn rd 213:7 route-type 7 | begin Network
  Network                Next Hop                Metric LocPrf Weight Path
Route Distinguisher: 213:7
*>i[7][213:7][213][32][10.2.7.2][32][232.0.0.5]/184
                213.6.6.6                0      100      0 ?
* i                213.8.8.8                0      100      0 ?

```

```

R7#show bgp ipv4 mvpn vrf MC route-type 7 213:7 213 10.2.7.2 232.0.0.5

```

BGP routing table entry for [7][213:7][213][10.2.7.2/32][232.0.0.5/32]/22, version 226

Paths: (1 available, best #1, table MVPNv4-BGP-Table)

Not advertised to any peer

Refresh Epoch 1

Local

213.6.6.6 (metric 20) from 213.12.12.12 (213.12.12.12)

Origin incomplete, metric 0, localpref 100, valid, internal, best

Extended Community: RT:213.7.7.7:1

Originator: 213.6.6.6, Cluster list: 213.12.12.12

rx pathid: 0, tx pathid: 0x0

R7#show bgp vpnv4 unicast rd 213:7 10.2.7.0/24

BGP routing table entry for 213:7:10.2.7.0/24, version 112

Paths: (1 available, best #1, table MC)

Advertised to update-groups:

3

Refresh Epoch 1

Local

0.0.0.0 (via vrf MC) from 0.0.0.0 (213.7.7.7)

Origin incomplete, metric 0, localpref 100, weight 32768, valid, sourced, best

Extended Community: RT:213:7 MVPN AS:213:0.0.0.0 MVPN VRF:213.7.7.7:1

mpls labels in/out 7000/nolabel(MC)

rx pathid: 0, tx pathid: 0x0

We can verify the PMSI binding in two ways. We can check the BGP Type-3 C(S,G) route to look at the tunnel opaque information, or we can look at the C-MRIB details. Both are shown below. The MDT is 0x10001 (65537) which is carried in the BGP route. The C-MRIB uses this for delivering the packet to the PMSI. The little 'y' and big 'G' mean the entry is sending to a data MDT (this group is permitted by the ACL) and was installed upon receipt of a BGP Type-7 route, respectively.

R7#show bgp ipv4 mvpn vrf MC route-type 3 10.2.7.2 232.0.0.5 213.7.7.7

BGP routing table entry for [3][213:7][10.2.7.2][232.0.0.5][213.7.7.7]/22, version 220

Paths: (1 available, best #1, table MVPNv4-BGP-Table, not advertised to EBGP peer)

Advertised to update-groups:

3

Refresh Epoch 1

Local

0.0.0.0 from 0.0.0.0 (213.7.7.7)

Origin incomplete, localpref 100, weight 32768, valid, sourced, local, best

Community: no-export

Extended Community: RT:213:7

PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 17, label: exp-null, tunnel parameters: 0600 0104 D507 0707 0007 0100 0400 0100 01

rx pathid: 0, tx pathid: 0x0

```
R7#show ip mroute vrf MC 232.0.0.5 10.2.7.2 verbose | begin \((
(10.2.7.2, 232.0.0.5), 05:31:45/stopped, flags: sTyG
  Incoming interface: GigabitEthernet2.527, RPF nbr 0.0.0.0
  MDT TX nr: 65537 LSM-ID: 0x28
  Outgoing interface list:
    Lspvif0, LSM MDT: 28 (data), Forward/Sparse, 03:32:11/stopped
```

Now that we know the tunnel being used, we will send traffic from CSR2. There should be no change in the BGP control-plane at this point as the S-PMSI and C-PIM delivery trees are complete. We will verify the C-MRIB state and packet counters concurrently on the relevant PEs. Looking at the egress PEs, they claim they are receiving traffic from a data MDT (big 'Y') and had the C-MRIB entry installed by having sent a BGP Type-7 route (little 'g'). Traffic arrives from the PMSI on the egress devices and is sent to the PMSI from the ingress device.

```
R6#show ip mroute vrf MC 232.0.0.5 10.2.7.2 | begin \((
(10.2.7.2, 232.0.0.5), 1d04h/00:02:11, flags: sTIYg
  Incoming interface: Lspvif0, RPF nbr 213.7.7.7, MDT: [65537,
213.7.7.7]/never
  Outgoing interface list:
    GigabitEthernet2.546, Forward/Sparse, 1d04h/00:02:11
```

```
R6#show ip mroute vrf MC 232.0.0.5 10.2.7.2 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 954, Packets received:
954
  Source: 10.2.7.2/32, Forwarding: 954/0/122/0, Other: 954/0/0
```

```
R7#show ip mroute vrf MC 232.0.0.5 10.2.7.2 | begin \((
(10.2.7.2, 232.0.0.5), 00:01:54/stopped, flags: sTyG
  Incoming interface: GigabitEthernet2.527, RPF nbr 0.0.0.0
  Outgoing interface list:
    Lspvif0, Forward/Sparse, 03:39:08/stopped
```

```
R7#show ip mroute vrf MC 232.0.0.5 10.2.7.2 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 969, Packets received:
969
  Source: 10.2.7.2/32, Forwarding: 969/0/118/0, Other: 969/0/0
```

```
R8#show ip mroute vrf MC 232.0.0.5 10.2.7.2 | begin \((
(10.2.7.2, 232.0.0.5), 1d04h/00:02:36, flags: sTIYg
  Incoming interface: Lspvif0, RPF nbr 213.7.7.7, MDT: [65537,
213.7.7.7]/never
  Outgoing interface list:
    GigabitEthernet2.538, Forward/Sparse, 1d04h/00:02:36
```

```
R8#show ip mroute vrf MC 232.0.0.5 10.2.7.2 count | begin ^Group
```

```
Group: 232.0.0.5, Source count: 1, Packets forwarded: 966, Packets received: 966
Source: 10.2.7.2/32, Forwarding: 966/0/122/0, Other: 966/0/0
```

A quick OAM ping can tell us who else is attached to the S-PMSI. It should only be CSR7 and CSR8. Additionally, we look at the downstream label mappings and expect to see replication towards XRv3 and XRv4. EPC proves this; the decimal label mappings in the show command match what EPC captured.

```
R7#ping mpls mldp p2mp 213.7.7.7 hex 0x01 00010001
p2mp Root node addr 213.7.7.7
Opaque type hex value (0x1), num hex digits 4
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
    timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]
```

```
Request #1
! reply addr 213.6.13.6
! reply addr 213.8.14.8
```

```
Round-trip min/avg/max = 9/86/164 ms
```

```
R7#show mpls mldp database opaque_type gid 65537 | section Replic
Replication client(s):
MDT (VRF MC)
    Uptime           : 00:11:03      Path Set ID   : None
    Interface        : Lspvif0
213.13.13.13:0
    Uptime           : 00:11:03      Path Set ID   : None
    Out label (D)    : 93015         Interface    : GigabitEthernet2.573*
    Local label (U)  : None          Next Hop     : 213.7.13.13
213.14.14.14:0
    Uptime           : 00:11:03      Path Set ID   : None
    Out label (D)    : 94020         Interface    : GigabitEthernet2.574*
    Local label (U)  : None          Next Hop     : 213.7.14.14
```

```
R7#show monitor capture CAP buffer detailed
2 122 1.820957 00:50:56:A9:EA:77 -> 00:50:56:A9:DB:37 MPLS unicast
0000: 005056A9 DB370050 56A9EA77 81000DF5 .PV..7.PV..w....
0010: 884716B5 71FE4500 00640F34 0000FE01 .G..q.E..d.4....
0020: B45B0A02 0702E800 00050800 CBFC0003 .[.....
0030: 00E50000 000010AF A0B6ABCD ABCDABCD .....

3 122 1.820957 00:50:56:A9:EA:77 -> 00:50:56:A9:DE:77 MPLS unicast
0000: 005056A9 DE770050 56A9EA77 81000DF6 .PV..w.PV..w....
0010: 884716F4 41FE4500 00640F34 0000FE01 .G..A.E..d.4....
0020: B45B0A02 0702E800 00050800 CBFC0003 .[.....
0030: 00E50000 000010AF A0B6ABCD ABCDABCD .....
```

We progress to IPv6 SSM. The pair C(2001:10:4:6::4, FF33::1) is not candidate for data MDT treatment but still technically travels along a P2MP S-PMSI path. CSR2 and CSR3 are receivers, so CSR7 and CSR8 both originate Type-7 SPT joins towards the ingress PE, CSR6. XRv2 (RR) sees both routes. When we look at the paths, we can see the dynamic RT applied. This value was derived from CSR7's VPNv6 route for the C-S. XR calls this the "VRF Route import", but it's the same as the MVPN RT shown in XE. This is the mechanism by which a receiver PE targets a sender PE.

```
RP/0/0/CPU0:XRv2#show bgp ipv6 mvpn rd 213:6
[7][213:6][213][128][2001:10:4:6:$
BGP routing table entry for
[7][213:6][213][128][2001:10:4:6::4][128][ff33::1]/376, Route Distinguisher:
213:6
[snip]
```

```
  Path #1: Received by speaker 0
  Advertised to update-groups (with more than one peer):
    0.2
  Local, (Received from a RR-client)
    213.7.7.7 (metric 20) from 213.7.7.7 (213.7.7.7)
      Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, not-in-vrf
      Received Path ID 0, Local Path ID 1, version 167
      Extended community: RT:213.6.6.6:1
  Path #2: Received by speaker 0
  Not advertised to any peer
  Local, (Received from a RR-client)
    213.8.8.8 (metric 30) from 213.8.8.8 (213.8.8.8)
      Origin incomplete, metric 0, localpref 100, valid, internal, import-
candidate, not-in-vrf
      Received Path ID 0, Local Path ID 0, version 0
      Extended community: RT:213.6.6.6:1
```

```
RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast rd 213:6 2001:10:4:6::/64
BGP routing table entry for 2001:10:4:6::/64, Route Distinguisher: 213:6
Versions:
```

Process	bRIB/RIB	SendTblVer
Speaker	92	92

```
Paths: (1 available, best #1)
  Advertised to update-groups (with more than one peer):
    0.2
  Path #1: Received by speaker 0
  Advertised to update-groups (with more than one peer):
    0.2
  Local, (Received from a RR-client)
    213.6.6.6 (metric 20) from 213.6.6.6 (213.6.6.6)
      Received Label 6001
      Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, not-in-vrf
```

```
Received Path ID 0, Local Path ID 1, version 92
Extended community: Source AS:213:0 VRF Route Import:213.6.6.6:1
RT:213:6
```

Notice that CSR6 is not originating a specific Type-3 S-PMSI route for this specific C(S,G). The group is not candidate for data MDT optimization and thus will match the next most specific Type-3 route for tunnel binding, which is the C(*,*). CSR6 originates this and identifies the GID as 0x20000 (131072 in decimal). The tunnel is mLDP P2MP (type 2) and it is rooted at CSR6, as expected.

```
R6#show bgp ipv6 mvpn vrf MC route-type 3 * * 213.6.6.6
BGP routing table entry for [3][213:6][*][*][213.6.6.6]/14, version 362
Paths: (1 available, best #1, table MVPNV6-BGP-Table, not advertised to EBGp
peer)
  Advertised to update-groups:
    3
  Refresh Epoch 1
  Local
    :: from 0.0.0.0 (213.6.6.6)
      Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
      Community: no-export
      Extended Community: RT:213:6
      PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 17, label: exp-null,
tunnel parameters: 0600 0104 D506 0606 0007 0100 0400 0200 00
      rx pathid: 0, tx pathid: 0x0
```

Using OAM, we can quickly test for leaves in this delivery tree, despite the “double wildcard” nature of this tunnel. Only CSR7 and CSR8 respond, and this is technically still S-PMSI operation since I-PMSI tunnels are not in use. In fact, the BGP Type-1 I-PMSI routes aren’t even carrying PMSI attributes at all.

```
R6#ping mpls mldp p2mp 213.6.6.6 hex 0x01 00020000
p2mp Root node addr 213.6.6.6
Opaque type hex value (0x1), num hex digits 4
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
  timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]

Request #1
! reply addr 213.8.14.8
! reply addr 213.7.13.7
```

```
Round-trip min/avg/max = 28/70/113 ms
```

```
R6#show bgp ipv6 mvpn vrf MC route-type 1 213.6.6.6
BGP routing table entry for [1][213:6][213.6.6.6]/12, version 361
```



```

Paths: (1 available, best #1, table MVPNV6-BGP-Table, not advertised to EBGp
peer)
  Advertised to update-groups:
    3
  Refresh Epoch 1
  Local
    :: from 0.0.0.0 (213.6.6.6)
      Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
      Community: no-export
      Extended Community: RT:213:6
      rx pathid: 0, tx pathid: 0x0

```

Before sending traffic, we quickly validate the C-PIM signaling at the ingress/egress PEs. We don't see any 'y' or 'Y' flags since this group is not using data MDTs. We do see 'g' flags on the egress PE's to signify that these C-MRIB entries were installed as a result of having sent a BGP Type-7 SPT join. Conversely, the ingress PE shows a big 'G' to signify that the entry was installed by receiving the route. Forwarding-wise, the ingress PE forwards towards the PMSI while the egress PEs accept traffic incoming from the PMSI.

```

R6#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 | begin \
(2001:10:4:6::4, FF33::1), 12:43:12/never, flags: sTG
  Incoming interface: GigabitEthernet2.546
  RPF nbr: 2001:10:4:6::4
  Immediate Outgoing interface list:
    Lspvif0, Forward, 12:43:12/never

```

```

R7#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 | begin \
(2001:10:4:6::4, FF33::1), 3d14h/never, flags: sTIg
  Incoming interface: Lspvif1
  RPF nbr: ::FFFF:213.6.6.6
  Immediate Outgoing interface list:
    GigabitEthernet2.527, Forward, 3d14h/never

```

```

R8#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 | begin \
(2001:10:4:6::4, FF33::1), 00:13:46/never, flags: sTIg
  Incoming interface: Lspvif2
  RPF nbr: ::FFFF:213.6.6.6
  Immediate Outgoing interface list:
    GigabitEthernet2.538, Forward, 00:13:46/never

```

CSR4 starts sending traffic. There is no change in the IPv4 MVPN signaling. A quick check of the C(S,G) packet counters shows us that the traffic flow is working as expected.

```

R4#ping FF33::1 repeat 10000
Output Interface: GigabitEthernet2.546

```

```
R6#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count | include HW
HW Forwarding: 92/0/118/0, Other: 0/0/0
```

```
R7#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count | include HW
HW Forwarding: 105/0/126/0, Other: 0/0/0
```

```
R8#show ipv6 mroute vrf MC FF33::1 2001:10:4:6::4 count | include HW
HW Forwarding: 99/0/126/0, Other: 0/0/0
```

Next, we will check the PMSI tunnel binding and mLDP P2MP tree. Both CSR7 and CSR8 import the Type-3 S-PMSI C(*,*) route from CSR6. We saw the GID earlier as being 131072 and we know the root is CSR6; we craft a regex to match this specific P2MP tree and view it based on the LSM-ID. As such, LSM packets should be arriving on CSR7 and CSR8 using the downstream labels allocated by those egress PEs.

```
R7#show mpls mldp database summary | include 213.6.6.6.*131072
29          P2MP      213.6.6.6          [gid 131072 (0x00020000)]      1
```

```
R7#show mpls mldp database id 29 | section Upstream
Upstream client(s) :
  213.13.13.13:0    [Active]
    Expires         : Never           Path Set ID   : 2A
    Out Label (U)   : None            Interface    : GigabitEthernet2.573*
    Local Label (D) : 7016            Next Hop     : 213.7.13.13
```

```
R8#show mpls mldp database summary | include 213.6.6.6.*131072
29          P2MP      213.6.6.6          [gid 131072 (0x00020000)]      1
```

```
R8#show mpls mldp database id 29 | section Upstream
Upstream client(s) :
  213.14.14.14:0    [Active]
    Expires         : Never           Path Set ID   : 29
    Out Label (U)   : None            Interface    : GigabitEthernet2.584*
    Local Label (D) : 8017            Next Hop     : 213.8.14.14
```

We can reasonably assume traffic is entering by using the LFIB counters. A more thorough examination would use EPC, capturing incoming traffic on those upstream facing links for each PE (CSR7 captures facing XRv3 and CSR8 captures facing XRv4). CSR7 shows label 0x1B68 (7016) and CSR8 shows label 0x1F51 (8017) entering from the SP core.

```
R7#show mpls forwarding-table labels 7016
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label       or Tunnel Id    Switched     interface
7016      [T]      No Label       [gid 131072 (0x00020000)] [V]         \
                                         24192      aggregate/MC
```

```
R8#show mpls forwarding-table label 8017
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
```

Label	Label	or Tunnel Id	Switched	interface
8017	[T] No Label	[gid 131072 (0x00020000)][V]	25074	\ aggregate/MC

R7#show monitor capture CAP buffer detailed

```

2 126      2.000000  00:50:56:A9:DB:37 -> 00:50:56:A9:EA:77 MPLS unicast
0000:  005056A9 EA770050 56A9DB37 81000DF5 .PV..w.PV..7....
0010:  884701B6 803E0000 213F6000 0000003C .G...>...!?'`....<
0020:  3A3F2001 00100004 00060000 00000000 :? .....
0030:  0004FF33 00000000 00000000 00000000 ...3.....

```

R8#show monitor capture CAP buffer detailed

```

0 126      0.000000  00:50:56:A9:DE:77 -> 00:50:56:A9:FB:1C MPLS unicast
0000:  005056A9 FB1C0050 56A9DE77 81000E00 .PV....PV..w....
0010:  884701F5 103D0000 213F6000 0000003C .G...=...!?'`....<
0020:  3A3F2001 00100004 00060000 00000000 :? .....
0030:  0004FF33 00000000 00000000 00000000 ...3.....

```

Next, we examine IPv4 ASM. As always, this model of multicast (in general) is much more complex than SSM both in terms of signaling and forwarding. First, we verify that the C-RP has the C(*,G) join having derived it from the Type 6 Shared tree join received by the PE in front of the C-RP (CSR5). The big 'G' on CSR5's C(*,G) entry means that the route was installed from having received a Type-6 shared tree join. I also show the dynamic RT matching between the unicast VPNv4 route and the BGP Type-6 route.

R5#show bgp ipv4 mvpn vrf MC route-type 6 213:5 213 10.5.10.10 225.0.0.1
BGP routing table entry for [6][213:5][213][10.5.10.10/32][225.0.0.1/32]/22,
version 306

Paths: (1 available, best #1, table MVPNV4-BGP-Table)

Not advertised to any peer

Refresh Epoch 1

Local

213.6.6.6 (metric 20) from 213.12.12.12 (213.12.12.12)

Origin incomplete, metric 0, localpref 100, valid, internal, best

Extended Community: RT:213.5.5.5:1

Originator: 213.6.6.6, Cluster list: 213.12.12.12

rx pathid: 0, tx pathid: 0x0

R5#show bgp vpnv4 unicast vrf MC 10.5.10.0/24

BGP routing table entry for 213:5:10.5.10.0/24, version 108

Paths: (1 available, best #1, table MC)

Advertised to update-groups:

3

Refresh Epoch 1

Local

0.0.0.0 (via vrf MC) from 0.0.0.0 (213.5.5.5)

Origin incomplete, metric 0, localpref 100, weight 32768, valid,
sourced, best

Extended Community: RT:213:5 MVPN AS:213:0.0.0.0 MVPN VRF:213.5.5.5:1

```
mpls labels in/out 5000/nolabel(MC)
rx pathid: 0, tx pathid: 0x0
```

```
R5#show ip mroute vrf MC 225.0.0.1 | begin \
(*, 225.0.0.1), 1d15h/stopped, RP 10.5.10.10, flags: SG
Incoming interface: GigabitEthernet2.550, RPF nbr 10.5.10.10
Outgoing interface list:
Lspvif0, Forward/Sparse, 14:31:12/stopped
```

```
R10#show ip mroute 225.0.0.1 | begin \
(*, 225.0.0.1), 1d15h/00:02:34, RP 10.5.10.10, flags: S
Incoming interface: Null, RPF nbr 0.0.0.0
Outgoing interface list:
GigabitEthernet2.550, Forward/Sparse, 14:31:27/00:02:34
```

When CSR2 starts sending, CSR7 registers the C(S,G) with the RP. It also sends a C(*,G) prune for the specific C-S back towards the RP. The C-RP will send a C(S,G) join towards CSR5. Upon receipt, CSR5 updates its C-MRIB and translates this C-PIM join into a BGP Type-7 SPT join towards CSR7. We also see the MDT was changed from null to E, so this is the LSM-ID that the flow currently uses on CSR5.

```
! CSR7
PIM(1): Check RP 10.5.10.10 into the (*, 225.0.0.1) entry
PIM(1): Building Triggered (*,G) Join / (S,G,RP-bit) Prune message for
225.0.0.1
PIM(1): Adding register encap tunnel (Tunnel2) as forwarding interface of
(10.2.7.2, 225.0.0.1).
PIM(1): Building Triggered (*,G) Join / (S,G,RP-bit) Prune message for
225.0.0.1

! CSR5
PIM(1): Received v2 Join/Prune on GigabitEthernet2.550 from 10.5.10.10, to us
PIM(1): Join-list: (10.2.7.2/32, 225.0.0.1), S-bit set
PIM(1): MDT next_hop change from: 0 to E for (10.2.7.2, 225.0.0.1) Lspvif0
PIM(1): Add GigabitEthernet2.550/10.5.10.10 to (10.2.7.2, 225.0.0.1), Forward
state, by PIM SG Join
PIM(1): Upstream mode for (10.2.7.2, 225.0.0.1) changed from 1 to 2
BGP[15] MVPN: add c-route, type 7, bs len 0 asn=0, rd=213:5,
source=10.2.7.2/4,
group=225.0.0.1/4,
nexthop=213.7.7.7,
len left = 0
BGP[14] MVPN umh lookup: vrfid 1, source 10.2.7.2
BGP[4] MVPN umh lookup: vrfid 1, source 10.2.7.2, net 213:5:10.2.7.0/24,
213:5:10.2.7.0/24 with matching nexthop 213.7.7.7, remote-rd [213:7]:
0x9:213:0.0.0.0, 0x10B:213.7.7.7:1,
BGP: MVPN(15) create local route
[7] [213:7] [213] [10.2.7.2/32] [225.0.0.1/32]/22
```

At the same time, egress PE's such as CSR1, CSR6, and CSR8 all generate Type 7 SPT joins just as CSR5 did once they receive traffic down the shared tree. The only different is the RD of each route varies between the originating router's RD. Only CSR1 is shown for brevity.

```
! CSR1
BGP[15] MVPN: add c-route, type 7, bs len 0 asn=0, rd=213:1,
source=10.2.7.2/4,
group=225.0.0.1/4,
nexthop=213.7.7.7,
len left = 0
```

XRv2 collects all of these routes, selects the best-path (doesn't matter which one), and reflects it to CSR7. Since they all have the same "RT import", CSR7 can import any of them, as it is the target of the C(S,G) join. XRv2 selected CSR5's join as best.

```
RP/0/0/CPU0:XRv2#show bgp ipv4 mvpn route-type 7 | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 213:7
* i [7] [213:7] [213] [32] [10.2.7.2] [32] [225.0.0.1] /184
      213.1.1.1          0      100      0 ?
*>i      213.5.5.5          0      100      0 ?
* i      213.6.6.6          0      100      0 ?
* i      213.8.8.8          0      100      0 ?
[snip]
```

CSR7 receives this best-path Type-7 route from XRv2 (originated from CSR5) and imports it to the VRF since the "RT import" (dynamic RT) is targeting CSR7. The PIM process interprets the C(S,G) information and begins forwarding native multicast towards the PMSI. Finally, since CSR7 is now aware that there must be a source behind it, it originates a Type-5 Source Active message to notify other PE's of an active source. PE's that come online and request multicast later can use this, since the C(S,G) entry will be in the C-MRIB when a Type-5 is received, even without receivers.

```
! CSR7
BGP(15): 213.12.12.12 rcvd UPDATE w/ attr: nexthop 213.5.5.5, origin ?,
localpref 100, metric 0, originator 213.5.5.5, clusterlist 213.12.12.12,
extended community RT:213.7.7.7:1
BGP(15): 213.12.12.12 rcvd [7] [213:7] [213] [10.2.7.2/32] [225.0.0.1/32] /22
BGP(15): skip vrf default table RIB route
[7] [213:7] [213] [10.2.7.2/32] [225.0.0.1/32] /22
BGP(15): add RIB route [7] [213:7] [213] [10.2.7.2/32] [225.0.0.1/32] /22
PIM(1): Join-list: (10.2.7.2/32, 225.0.0.1), S-bit set, BGP C-Route
PIM(1): MDT next_hop change from: 0 to 27 for (10.2.7.2, 225.0.0.1) Lspvif0
PIM(1): Add Lspvif0/0.0.0.0 to (10.2.7.2, 225.0.0.1), Forward state, by BGP
SG Join
BGP(15): nettable_walker [5] [213:7] [10.2.7.2] [225.0.0.1] /18 route sourced
locally
```

```

BGP(15): delete RIB route [5][213:7][10.2.7.2][225.0.0.1]/18
BGP(15): 213.12.12.12 NEXT_HOP self is set for sourced RT Filter for net
[5][213:7][10.2.7.2][225.0.0.1]/18
BGP(15): (base) 213.12.12.12 send UPDATE (format)
[5][213:7][10.2.7.2][225.0.0.1]/18, next 213.7.7.7, metric 0, path Local,
extended community RT:213:7

```

Egress PEs all receive this route and import it to their VRFs, since they all import RT 213:7. This specific output is from CSR8.

```

! CSR8
BGP(15): 213.12.12.12 rcvd UPDATE w/ attr: nexthop 213.7.7.7, origin ?,
localpref 100, metric 0, originator 213.7.7.7, clusterlist 213.12.12.12,
community no-export, extended community RT:213:7
BGP(15): 213.12.12.12 rcvd [5][213:7][10.2.7.2][225.0.0.1]/18
BGP(15): skip vrf default table RIB route [5][213:7][10.2.7.2][225.0.0.1]/18

```

At this point, egress PEs can receive traffic. As an optimization, the C-RP prunes itself from the shared tree when it realizes it is not in the transit path. CSR5 receives this prune message and removes the customer LAN from the forwarding tree. The C-RP only does this after telling the DR (CSR7) to stop registering the C(S,G). CSR7 also notices that the C(S,G) is not candidate for data MDT treatment.

```

! CSR7
PIM(1): Received v2 Register-Stop on GigabitEthernet2.574 from 10.5.10.10
PIM(1): for source 10.2.7.2, group 225.0.0.1
PIM(1): Removing register encap tunnel (Tunnel2) as forwarding interface of
(10.2.7.2, 225.0.0.1).
PIM(1): Clear Registering flag to 10.5.10.10 for (10.2.7.2/32, 225.0.0.1)
PIM(1): MDT ACL not matched for (10.2.7.2,225.0.0.1)

! CSR5
PIM(1): Received v2 Join/Prune on GigabitEthernet2.550 from 10.5.10.10, to us
PIM(1): Prune-list: (10.2.7.2/32, 225.0.0.1)
PIM(1): Prune GigabitEthernet2.550/225.0.0.1 from (10.2.7.2/32, 225.0.0.1) -
deleted

```

Quickly verifying the C-MRIB entries on all relevant PEs, we see the signaling is working. There is some awkwardness in this output since we now have many LSPvif interfaces (enumerations are numbers greater than 0 on some routers). This is due to the partitioned nature of the MVPN profile; it is also the reason for needing the “strict” RPF check because now there are multiple interfaces representing the PMSI. The ingress PE (CSR7) shows the familiar ‘G’ flags to indicate receiving a BGP Type-7 route and originator a Type-5 Source Active. The egress PEs (CSR1, CSR7, CSR8) show ‘gQ’ to indicate the inverse of what was just stated; BGP joins were advertised while Source Actives were received. CSR5’s output shows traffic entering and exiting the PMSI in hairpin fashion; we will look at this shortly.

```

R1#show ip mroute vrf MC 225.0.0.1 10.2.7.2 | begin \

```

```
(10.2.7.2, 225.0.0.1), 00:37:03/00:01:04, flags: JTgQ
  Incoming interface: Lspvif2, RPF nbr 213.7.7.7
  Outgoing interface list:
    GigabitEthernet2.519, Forward/Sparse, 00:37:03/00:02:35
```

```
R5#show ip mroute vrf MC 225.0.0.1 10.2.7.2 | begin \
(10.2.7.2, 225.0.0.1), 00:37:16/00:02:43, flags: Tg
  Incoming interface: Lspvif1, RPF nbr 213.7.7.7
  Outgoing interface list:
    Lspvif0, Forward/Sparse, 00:37:16/stopped
```

```
R6#show ip mroute vrf MC 225.0.0.1 10.2.7.2 | begin \
(10.2.7.2, 225.0.0.1), 00:13:08/00:02:11, flags: TgQ
  Incoming interface: Lspvif2, RPF nbr 213.7.7.7
  Outgoing interface list:
    GigabitEthernet2.546, Forward/Sparse, 00:13:08/00:02:45
```

```
R7#show ip mroute vrf MC 225.0.0.1 10.2.7.2 | begin \
(10.2.7.2, 225.0.0.1), 00:37:19/00:01:46, flags: FTGq
  Incoming interface: GigabitEthernet2.527, RPF nbr 0.0.0.0
  Outgoing interface list:
    Lspvif0, Forward/Sparse, 00:37:18/stopped
```

```
R8#show ip mroute vrf MC 225.0.0.1 10.2.7.2 | begin \
(10.2.7.2, 225.0.0.1), 00:37:21/00:01:42, flags: TgQ
  Incoming interface: Lspvif1, RPF nbr 213.7.7.7
  Outgoing interface list:
    GigabitEthernet2.538, Forward/Sparse, 00:37:21/00:02:27
```

A quick check of packet counters shows that the traffic appears to be working somewhat poorly. We see tons of RPF failures (50% of the traffic) and many more packets arriving than we should on most of the egress PEs. CSR8 appears to be the exception. I suspect this has something to do with CSR5 hairpinning traffic back into the network.

```
R1#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | begin ^Group
Group: 225.0.0.1, Source count: 1, Packets forwarded: 1300, Packets received:
2599
  Source: 10.2.7.2/32, Forwarding: 1298/0/122/0, Other: 2597/1299/0
```

```
R5#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | begin ^Group
Group: 225.0.0.1, Source count: 1, Packets forwarded: 1304, Packets received:
1305
  Source: 10.2.7.2/32, Forwarding: 1302/0/122/0, Other: 1303/1/0
```

```
R6#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | begin ^Group
Group: 225.0.0.1, Source count: 1, Packets forwarded: 577, Packets received:
1154
  Source: 10.2.7.2/32, Forwarding: 577/0/122/0, Other: 1154/577/0
```

```
R7#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | begin ^Group
Group: 225.0.0.1, Source count: 1, Packets forwarded: 1307, Packets received:
2613
```

```
Source: 10.2.7.2/32, Forwarding: 1307/0/117/0, Other: 2613/1306/0
```

```
R8#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | begin ^Group
Group: 225.0.0.1, Source count: 1, Packets forwarded: 1363, Packets received:
1364
```

```
Source: 10.2.7.2/32, Forwarding: 1362/0/122/0, Other: 1363/1/0
```

Strange situations like this warrant more detailed verification. We begin at CSR7 to trace the S-PMSI tree used for delivering this flow. Through a quick process of elimination, we don't see a C(S,G) or C(*,G) Type-3 route, so the C(*,*) matches. The tunnel GID is 0x1000 (65536), is rooted at CSR7, and is an mLDP P2MP tunnel.

```
R7#show bgp ipv4 mvpn vrf MC route-type 3 10.2.7.2 225.0.0.1 213.7.7.7
% Network not in table
```

```
R7#show bgp ipv4 mvpn vrf MC route-type 3 * 225.0.0.1 213.7.7.7
% Network not in table
```

```
R7#show bgp ipv4 mvpn vrf MC route-type 3 * * 213.7.7.7
BGP routing table entry for [3][213:7][*][*][213.7.7.7]/14, version 219
Paths: (1 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGp
peer)
```

```
Advertised to update-groups:
```

```
3
```

```
Refresh Epoch 1
```

```
Local
```

```
0.0.0.0 from 0.0.0.0 (213.7.7.7)
```

```
Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
```

```
Community: no-export
```

```
Extended Community: RT:213:7
```

```
PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 17, label: exp-null,
tunnel parameters: 0600 0104 D507 0707 0007 0100 0400 0100 00
```

```
rx pathid: 0, tx pathid: 0x0
```

Now we can look at the downstream label bindings and trace the activity in the core. CSR7 replicates the traffic down the tree to 93011 and 94001 (XRv3 and XRv4, respectively).

```
R7#show mpls mldp database summary | include 213.7.7.7.*65536
27 P2MP 213.7.7.7 [gid 65536 (0x00010000)] 3
```

```
R7#show mpls mldp database id 27 | section Replic
```

```
Replication client(s):
```

```
MDT (VRF MC)
```



```

Uptime           : 14:03:11      Path Set ID      : None
Interface        : Lspvif0
213.13.13.13:0
Uptime           : 14:02:40      Path Set ID      : None
Out label (D)    : 93011         Interface        : GigabitEthernet2.573*
Local label (U)  : None          Next Hop         : 213.7.13.13
213.14.14.14:0
Uptime           : 00:51:14      Path Set ID      : None
Out label (D)    : 94001         Interface        : GigabitEthernet2.574*
Local label (U)  : None          Next Hop         : 213.7.14.14

```

XRv3 replicates the traffic to CSR5, CSR6, and XRv2. It is interesting to note that suddenly XRv2 wants the traffic; this is technically correct since XRv1 is joining the group, but BGP c-mcast signaling did not actually make a Type-7 join. I believe this is an XRv-specific behavior. XRv4 replicates the traffic to CSR1 and CSR8. Because we know CSR1, CSR6, CSR8, and XRv2 are legitimate endpoints, we won't trace the tree farther in those directions. I am more curious about what is happening at CSR5.

```

RP/0/0/CPU0:XRv3#show mpls forwarding labels 93011
Local  Outgoing  Prefix      Outgoing    Next Hop    Bytes
Label  Label      or ID       Interface   Hop         Switched
-----
93011  5012        MLDP: 0x00018  Gi0/0/0/0.553 213.5.13.5    0
        6014        MLDP: 0x00018  Gi0/0/0/0.563 213.6.13.6    0
        92011      MLDP: 0x00018  Gi0/0/0/0.523 213.12.13.12  0

RP/0/0/CPU0:XRv4#show mpls forwarding labels 94001
Local  Outgoing  Prefix      Outgoing    Next Hop    Bytes
Label  Label      or ID       Interface   Hop         Switched
-----
94001  1011      MLDP: 0x00025  Gi0/0/0/0.514 213.1.14.1    0
        8014      MLDP: 0x00025  Gi0/0/0/0.584 213.8.14.8    0

```

CSR5's LFIB claims that the packet is being decapsulated and send to the customer. I suspect that the traffic is actually being re-encapsulated and sent down CSR5's S-PMSI C(*,*) tree to other spokes. Earlier debug output told us that LSM-ID "E" was used by CSR5 to send traffic downstream initially while the SPT was under construction. This reveals labels 93000 and 94013 used for downstream traffic.

```

R5#show mpls forwarding-table labels 5012 det
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id  Switched    interface
5012      No Label   [gid 65536 (0x00010000)][V]  \
                                         406870    aggregate/MC

```

```

R5#show mpls mldp database id E | section Replic
Replication client(s):
MDT  (VRF MC)
Uptime           : 14:09:43      Path Set ID      : None

```

```

Interface      : Lspvif0
213.14.14.14:0
Uptime         : 14:09:42      Path Set ID  : None
Out label (D)  : 94013         Interface    : GigabitEthernet2.554*
Local label (U): None          Next Hop     : 213.5.14.14
213.13.13.13:0
Uptime         : 10:55:01      Path Set ID  : None
Out label (D)  : 93000         Interface    : GigabitEthernet2.553*
Local label (U): None          Next Hop     : 213.5.13.13

```

Using EPC outbound on CSR5's links, we can see if CSR5 is sending LSM originally sourced at CSR2 back down into the network. We can clearly see that is the case; notice the IP header information shows traffic from 10.2.7.2 to 225.0.0.1. The labels 0x16F3B and 0x16B48 are 94013 and 93000, respectively. This is a highly undesirable situation.

```

R5#show monitor capture CAP buffer detailed
 2 122      1.417978 00:50:56:A9:DC:63 -> 00:50:56:A9:DE:77 MPLS unicast
0000: 005056A9 DE770050 56A9DC63 81000DE2 .PV..w.PV..c....
0010: 884716F3 D1FC4500 00641717 0000FC01 .G....E..d.....
0020: B57C0A02 0702E100 00010800 CD820005 .|.....
0030: 071E0000 000012EE 96B6ABCD ABCDABCD .....

 3 122      1.418970 00:50:56:A9:DC:63 -> 00:50:56:A9:DB:37 MPLS unicast
0000: 005056A9 DB370050 56A9DC63 81000DE1 .PV..7.PV..c....
0010: 884716B4 81FC4500 00641717 0000FC01 .G....E..d.....
0020: B57C0A02 0702E100 00010800 CD820005 .|.....
0030: 071E0000 000012EE 96B6ABCD ABCDABCD .....

```

A quick OAM check will reveal all the endpoints of this tree. This explains why we saw RPF failures on every node except CSR8; every other node is joined to CSR5's tree which means duplicate packets are being received. Considering CSR8 has an identical configuration as CSR6, I cannot explain this behavior, but at least the result of the behavior is predictable.

```

R5#ping mpls mldp p2mp 213.5.5.5 hex 0x01 00010000
p2mp Root node addr 213.5.5.5
Opaque type hex value (0x1), num hex digits 4
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
  timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]

```

Type escape sequence to abort.

```

Request #1
! reply addr 213.12.13.12
! reply addr 213.1.14.1
! reply addr 213.7.13.7

```

```
! reply addr 213.6.13.6
```

```
Round-trip min/avg/max = 33/126/200 ms
```

My speculation is that because a partitioned MDT is in use, traffic received by CSR5 is send back into the network because CSR5 still has a Type-6 Shared Tree join towards the C-RP. That never gets pruned unless the PE's have no more end hosts that send IGMP joins for the ASM groups in question. The traffic is not following the C(*,G) entry because the C(S,G) exists, but I don't believe the MDT is smart enough to assume anything, even though the active source is behind CSR7 (CSR5 knows it from the Type-5). The logic still doesn't explain why CSR8 is not part of this delivery tree.

```
R5#show ip mroute vrf MC | section 225.0.0.1
(*, 225.0.0.1), 1d15h/stopped, RP 10.5.10.10, flags: SG
  Incoming interface: GigabitEthernet2.550, RPF nbr 10.5.10.10
  Outgoing interface list:
    Lspvif0, Forward/Sparse, 14:26:39/stopped

(10.2.7.2, 225.0.0.1), 01:15:28/00:02:32, flags: Tg
  Incoming interface: Lspvif1, RPF nbr 213.7.7.7
  Outgoing interface list:
    Lspvif0, Forward/Sparse, 01:15:28/stopped
```

A hard-clear of BGP IPv4 MVPN makes CSR8 join the tree, so I am assuming are some race conditions behind the scenes. We can now see the RPF failures on CSR8 as well as the OAM results from CSR5.

```
R8#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | begin ^Group
Group: 225.0.0.1, Source count: 1, Packets forwarded: 2182, Packets received:
2218
  Source: 10.2.7.2/32, Forwarding: 2181/0/122/0, Other: 2217/36/0
```

```
R5#ping mpls mldp p2mp 213.5.5.5 hex 0x01 00010000
p2mp Root node addr 213.5.5.5
Opaque type hex value (0x1), num hex digits 4
Sending 1, 72-byte MPLS Echost to Target FEC Stack TLV descriptor,
  timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]
```

```
Request #1
! reply addr 213.1.14.1
! reply addr 213.8.14.8
! reply addr 213.6.13.6
! reply addr 213.7.13.7
! reply addr 213.12.13.12
```

```
Round-trip min/avg/max = 18/72/121 ms
```

Many of the routers (ingress and egress PE) are also complaining about coordination issues between the PIM and MRIB processes. I don't pretend to understand this, but I would say this feature is still somewhat tricky.

```
%PIM-4-PIM_INVALID_SB: Unexpected access to PIM interface subblock field -
Process= "MFIB_mrib_read", ipl= 0, pid= 586
-Traceback= 1#bc52344720f9eab7e85bf9762016d644 :7F032EE8D000+73D1382
:7F032EE8D000+73CF17F :7F032EE8D000+73CEF37 :7F032EE8D000+73CD75F
:7F032EE8D000+68AA708 :7F032EE8D000+68A565F :7F032EE8D000+8A1872B
:7F032EE8D000+8A166E0 :7F032EE8D000+89AA8D6 :7F032EE8D000+89A76B3
:7F032EE8D000+8A19CE0 :7F032EE8D000+8A198C5 :7F032EE8D000+896CB6B
:7F032EE8D000+896C04A :7F032EE8D000+896BA3C :7F032EE8D000+8A1D807
```

In summary, partitioned MDTs seem best deployed when paired with SSM. Only S-PMSI is used (XE only supports P2MP trees while XR supports MP2MP and P2MP trees) with wildcards in the BGP Type-3 routes to direct traffic into certain P-tunnels. If the C-RP is not in the same site as the sources when using ASM, you can end up with duplicated traffic in the network as I have illustrated above.

Additional Reading – Reference configurations “mvpn-14”

28.13 Profile 17: Default MDT – MLDP P2MP – BGP-AD – PIM C-mcast signaling

For this test, begin with the MVPN Profile 12 configuration where all of the MVPN BGP AD is set up already, plus the P2MP trees are built (no MP2MP trees anywhere). All we need to do is remove BGP c-multicast signaling and use PIM instead. The I-PMSI P2MP trees will still exist due to BGP AD, but we will never see Type 6 or 7 routes in this test.

```
! Basic XE configuration to disable BGP (enable PIM) c-mcast signaling
vrf definition MC
  address-family ipv4
    no mdt overlay use-bgp
  address-family ipv6
    no mdt overlay use-bgp

! Basic XR configuration to disable BGP (enable PIM) c-mcast signaling
router pim
  vrf MC
    address-family ipv4
      no mdt c-multicast-routing
    address-family ipv6
      no mdt c-multicast-routing
```

We will verify CSR1 as a sanity check. It still has all the appropriate I-PMSI P2MP trees, as well as the appropriate BGP AD routes. Specifically, it has Type-1 I-PMSI routes from all PEs for IPv4, and only the “north-side” routers for IPv6. The Type-3 S-PMSI route is an AD message originated from CSR7 to allow PEs with interested receivers for this C(S,G) to join this S-PMSI P2MP tree.

R1#show mpls mldp database summary

LSM ID	Type	Root	Decoded Opaque Value	Client
Cnt.				
15	P2MP	213.12.12.12	[gid 2 (0x00000002)]	1
E	P2MP	213.1.1.1	[gid 65536 (0x00010000)]	2
11	P2MP	213.5.5.5	[gid 65536 (0x00010000)]	1
12	P2MP	213.6.6.6	[gid 65536 (0x00010000)]	1
13	P2MP	213.7.7.7	[gid 65536 (0x00010000)]	1
14	P2MP	213.8.8.8	[gid 65536 (0x00010000)]	1
F	P2MP	213.1.1.1	[gid 131072 (0x00020000)]	2
10	P2MP	213.5.5.5	[gid 131072 (0x00020000)]	1
16	P2MP	213.12.12.12	[gid 262146 (0x00040002)]	1

R1#show bgp ipv4 mvpn vrf MC | begin Network

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 213:1 (default for vrf MC)					
*> [1] [213:1] [213.1.1.1]/12	0.0.0.0			32768	?
*>i [1] [213:1] [213.5.5.5]/12	213.5.5.5	0	100	0	?
*>i [1] [213:1] [213.6.6.6]/12	213.6.6.6	0	100	0	?
*>i [1] [213:1] [213.7.7.7]/12	213.7.7.7	0	100	0	?
*>i [1] [213:1] [213.8.8.8]/12	213.8.8.8	0	100	0	?
*>i [1] [213:1] [213.12.12.12]/12	213.12.12.12		100	0	i
*>i [3] [213:1] [10.2.7.2] [232.0.0.5] [213.7.7.7]/22	213.7.7.7	0	100	0	?

R1#show bgp ipv6 mvpn vrf MC | begin Network

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 213:1 (default for vrf MC)					
*> [1] [213:1] [213.1.1.1]/12	::			32768	?
*>i [1] [213:1] [213.5.5.5]/12	213.5.5.5	0	100	0	?
*>i [1] [213:1] [213.12.12.12]/12	213.12.12.12		100	0	i

We also expect to see a “full-mesh” of C-PIM neighbors over the I-PMSI. In the interest of brevity, we will check CSR6 and XRv2 for extra validation, then continue with the test.

R1#show ip pim vrf MC neighbor | begin ^Neighbor

Neighbor Address	Interface	Uptime/Expires	Ver	DR Prio/Mode

213.8.8.8	Lspvif0	00:03:49/00:01:22	v2	1 / S P G
213.5.5.5	Lspvif0	00:03:49/00:01:22	v2	1 / S P G
213.7.7.7	Lspvif0	00:03:49/00:01:22	v2	1 / S P G
213.6.6.6	Lspvif0	00:03:49/00:01:21	v2	1 / S P G
213.12.12.12	Lspvif0	00:15:08/00:01:21	v2	1 / DR P G

R6#show ip pim vrf MC neighbor | begin ^Neighbor

Neighbor Address	Interface	Uptime/Expires	Ver	DR Prio/Mode
213.8.8.8	Lspvif0	00:03:06/00:01:35	v2	1 / S P G
213.5.5.5	Lspvif0	00:03:06/00:01:35	v2	1 / S P G
213.7.7.7	Lspvif0	00:03:06/00:01:35	v2	1 / S P G
213.1.1.1	Lspvif0	00:04:04/00:01:35	v2	1 / S P G
213.12.12.12	Lspvif0	00:14:25/00:01:34	v2	1 / DR P G

RP/0/0/CPU0:XRv2#show pim vrf MC neighbor | begin ^Neighbor

Neighbor Address	Interface	Uptime	Expires	DR pri	Flags
10.11.12.11	GigabitEthernet0/0/0/0.512	1d07h	00:01:21	1	B P
10.11.12.12*	GigabitEthernet0/0/0/0.512	2d00h	00:01:35	1	(DR) B P E
213.1.1.1	LmdtMC	00:05:09	00:01:29	1	P
213.5.5.5	LmdtMC	00:04:11	00:01:29	1	P
213.6.6.6	LmdtMC	00:04:11	00:01:28	1	P
213.7.7.7	LmdtMC	00:04:11	00:01:30	1	P
213.8.8.8	LmdtMC	00:04:11	00:01:29	1	P
213.12.12.12*	LmdtMC	00:15:37	00:01:29	1	(DR) P

Because C-PIM signaling is used, we expect to see C(*,G) joins making their way through the PMSI towards the C-RP, with C(S,G) SSM joins making their way towards their respective C-S. In this case, CSR10 is the R-CP which should have the (*, 225.0.0.1) join. CSR2 is an SSM source, which means CSR7 should have the (10.2.7.2, 232.0.0.5) join.

R10#show ip mroute 225.0.0.1 | begin \

(*, 225.0.0.1), 22:23:06/00:03:16, RP 10.5.10.10, flags: S

Incoming interface: Null, RPF nbr 0.0.0.0

Outgoing interface list:

GigabitEthernet2.550, Forward/Sparse, 22:23:06/00:03:16

R7#show ip mroute vrf MC 232.0.0.5 10.2.7.2 | begin \

(10.2.7.2, 232.0.0.5), 22:22:54/00:02:56, flags: sTy

Incoming interface: GigabitEthernet2.527, RPF nbr 0.0.0.0

Outgoing interface list:

Lspvif0, Forward/Sparse, 22:22:54/00:02:56

The output on CSR7 indicates that this group is “sending” to an MDT data group (little ‘y’ flag). We can find out the S-PMSI tunnel binding by checking the BGP-AD messages for a matching Type-3 associated with this C(S,G). CSR7 originated this Type-3 S-PMSI route to tell other PEs about this C(S,G). The tunnel-type 2 field means this is an mLDP P2MP tunnel and the root is 213.7.7.7 (0xD5070707). The MDT number is 65537 (0x10001).

```

R7#show bgp ipv4 mvpn vrf MC route-type 3 10.2.7.2 232.0.0.5 213.7.7.7
BGP routing table entry for [3][213:7][10.2.7.2][232.0.0.5][213.7.7.7]/22,
version 200
Paths: (1 available, best #1, table MVPNv4-BGP-Table, not advertised to EBGp
peer)
  Advertised to update-groups:
    3
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (213.7.7.7)
      Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
      Community: no-export
      Extended Community: RT:213:7
      PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 17, label: exp-null,
tunnel parameters: 0600 0104 D507 0707 0007 0100 0400 0100 01
      rx pathid: 0, tx pathid: 0x0

```

Using the “verbose” modifier, we can see the binding between the S-PMSI and the C(S,G) MRIB entry. The little ‘p’ means that it was signaled from a C-PIM join, which is expected.

```

R7#show ip mroute vrf MC 232.0.0.5 10.2.7.2 verbose | begin \((
(10.2.7.2, 232.0.0.5), 22:37:25/00:03:08, flags: sTyp
  Incoming interface: GigabitEthernet2.527, RPF nbr 0.0.0.0
  MDT TX nr: 65537 LSM-ID: 0x23
  Outgoing interface list:
    Lspvif0, LSM MDT: 23 (data), Forward/Sparse, 22:37:25/00:03:08, p

```

When CSR2 starts sending traffic, it will be delivered to all endpoints of the P2MP tree. We could trace the tree to verify, but since that has been demonstrated many times, I will use show commands on CSR6 and CSR8. If the GID is 65537 is it highly likely this is part of the same mLDP P2MP tree.

```

R6#show mpls mldp database summary | include 65537
23          P2MP      213.7.7.7          [gid 65537 (0x00010001)]      1

R8#show mpls mldp database summary | include 65537
22          P2MP      213.7.7.7          [gid 65537 (0x00010001)]      1

```

We quickly verify the C-MRIB entries on CSR6 and CSR8. The RPF interface is the PMSI (MDT is shown without the verbose keyword here) and OIL includes the customer LAN. The big ‘Y’ indicates that these routers have “joined” a Data MDT.

```

R6#show ip mroute vrf MC 232.0.0.5 10.2.7.2 | begin \((
(10.2.7.2, 232.0.0.5), 22:42:35/00:02:26, flags: sTIY
  Incoming interface: Lspvif0, RPF nbr 213.7.7.7, MDT: [65537,
213.7.7.7]/never

```

Outgoing interface list:

GigabitEthernet2.546, Forward/Sparse, 22:42:35/00:02:26

```
R8#show ip mroute vrf MC 232.0.0.5 10.2.7.2 | begin \((
```

```
(10.2.7.2, 232.0.0.5), 22:42:57/00:02:05, flags: sTIY
```

```
Incoming interface: Lspvif0, RPF nbr 213.7.7.7, MDT: [65537, 213.7.7.7]/never
```

Outgoing interface list:

GigabitEthernet2.538, Forward/Sparse, 22:42:57/00:02:05

CSR2 begins sending traffic. Because the S-PMSI is signaled and the group is SSM, no more signaling will take place. We can check the packet counters on all relevant PEs to ensure packets are being forwarded.

```
R6#show ip mroute vrf MC 232.0.0.5 10.2.7.2 count | begin ^Group
```

```
Group: 232.0.0.5, Source count: 1, Packets forwarded: 68, Packets received: 68
```

```
Source: 10.2.7.2/32, Forwarding: 68/0/122/0, Other: 68/0/0
```

```
R7#show ip mroute vrf MC 232.0.0.5 10.2.7.2 count | begin ^Group
```

```
Group: 232.0.0.5, Source count: 1, Packets forwarded: 67, Packets received: 67
```

```
Source: 10.2.7.2/32, Forwarding: 67/0/118/0, Other: 67/0/0
```

```
R8#show ip mroute vrf MC 232.0.0.5 10.2.7.2 count | begin ^Group
```

```
Group: 232.0.0.5, Source count: 1, Packets forwarded: 69, Packets received: 69
```

```
Source: 10.2.7.2/32, Forwarding: 69/0/122/0, Other: 69/0/0
```

We will quickly trace the LSPs that make up the P2MP tree used for this flow. We will use a different method this time; we know that P2MP trees are unidirectional (downstream only) and are receiver-driven, but we can still verify it from the root downwards. On CSR7, we will check the downstream labels learned from XRv3 and XRv4. CSR7 should use labels 93015 and 94020 for XRv3 and XRv4 respectively when sending traffic down this tree.

```
R7#show mpls mldp database opaque_type gid 65537 | section Replic
```

Replication client(s):

MDT (VRF MC)

Uptime : 22:48:55 Path Set ID : None

Interface : Lspvif0

213.14.14.14:0

Uptime : 22:48:55 Path Set ID : None

Out label (D) : 94020 Interface : GigabitEthernet2.574*

Local label (U): None Next Hop : 213.7.14.14

213.13.13.13:0

Uptime : 22:48:42 Path Set ID : None

Out label (D) : 93015 Interface : GigabitEthernet2.573*

Local label (U): None Next Hop : 213.7.13.13

Using EPC on CSR7 outbound concurrently on both core links, we can verify the packets are actually using the labels we just traced. Assuming the root node pushes the right MVPN label onto the packet, the core routers will perform swap operations until it gets to the leaves. Notice the identical timestamps, source MAC address, and IP header information. This is the router performing replication as expected. The labels are highlighted (93015 and 94020) which are the downstream labels for XRv3 and XRv4.

```
R7#show monitor capture CAP buffer detailed
12 122 251158.141991 00:50:56:A9:EA:77 -> 00:50:56:A9:DB:37 MPLS unicast
0000: 005056A9 DB370050 56A9EA77 81000DF5 .PV..7.PV..w....
0010: 884716B5 71FE4500 0064067E 0000FE01 .G..q.E..d~....
0020: BD110A02 0702E800 00050800 28CC0001 .....(....
0030: 01050000 00000F7C 44FCABCD ABCDABCD .....|D.....

13 122 251160.141991 00:50:56:A9:EA:77 -> 00:50:56:A9:DE:77 MPLS unicast
0000: 005056A9 DE770050 56A9EA77 81000DF6 .PV..w.PV..w....
0010: 884716F4 41FE4500 0064067F 0000FE01 .G..A.E..d.....
0020: BD100A02 0702E800 00050800 20FB0001 .....
0030: 01060000 00000F7C 4CCCABCD ABCDABCD .....|L.....
```

XRv3 and XRv4 perform label swap operations and forward down their trees to CSR6 and CSR8 respectively.

```
RP/0/0/CPU0:XRv3#show mpls forwarding labels 93015
Local   Outgoing   Prefix      Outgoing   Next Hop    Bytes
Label   Label      or ID       Interface  Hop         Switched
-----
93015   6013       MLDP: 0x0001c  Gi0/0/0/0.563 213.6.13.6    0

RP/0/0/CPU0:XRv4#show mpls forwarding labels 94020
Local   Outgoing   Prefix      Outgoing   Next Hop    Bytes
Label   Label      or ID       Interface  Hop         Switched
-----
94020   8002       MLDP: 0x0001f  Gi0/0/0/0.584 213.8.14.8    0
```

Checking CSR6 and CSR8, we see the packets arrive with the proper labels. The LFIB also shows the GID for verification.

```
R6#show mpls forwarding-table labels 6013
Local   Outgoing   Prefix      Bytes Label   Outgoing   Next Hop
Label   Label      or Tunnel Id  Switched      interface
6013    [T] No Label  [gid 65537 (0x00010001)] [V] \
                                         51484      aggregate/MC

R8#show mpls forwarding-table labels 8002
Local   Outgoing   Prefix      Bytes Label   Outgoing   Next Hop
```

Label	Label	or Tunnel Id	Switched	interface
8002	[T] No Label	[gid 65537 (0x00010001)][V]	53558	\ aggregate/MC

Next, we will examine the more complex ASM setup. CSR10 is the R-CP, so when CSR2 starts sending traffic, CSR7 will register this C(S,G) with the C-RP. CSR10 decapsulates the payload and sends this traffic down the C(*,G) tree towards CSR6 and CSR8.

```
! CSR7
PIM(1): Check RP 10.5.10.10 into the (*, 225.0.0.1) entry
PIM(1): Building Triggered (*,G) Join / (S,G,RP-bit) Prune message for
225.0.0.1
PIM(1): Adding register encap tunnel (Tunnel2) as forwarding interface of
(10.2.7.2, 225.0.0.1).
```

```
! CSR10
PIM(0): Received v2 Register on GigabitEthernet2.550 from 10.2.7.7
for 10.2.7.2, group 225.0.0.1
PIM(0): Adding register decap tunnel (Tunnel1) as accepting interface of
(10.2.7.2, 225.0.0.1).
PIM(0): Insert (10.2.7.2,225.0.0.1) join in nbr 10.5.10.5's queue
PIM(0): Building Join/Prune packet for nbr 10.5.10.5
PIM(0): Adding v2 (10.2.7.2/32, 225.0.0.1), S-bit Join
PIM(0): Send v2 join/prune to 10.5.10.5 (GigabitEthernet2.550)
```

CSR5 receives that join from CSR10 and sends it towards the C(S,G), CSR7. CSR7 then updates the MDT in use from 0 (null in this case, since there is no MP2MP default MDT) to LSM-ID 0x17. We can see this is the I-PMSI P2MP tunnel rooted at CSR7, which is correct.

```
! CSR5
PIM(1): Received v2 Join/Prune on GigabitEthernet2.550 from 10.5.10.10, to us
PIM(1): Join-list: (10.2.7.2/32, 225.0.0.1), S-bit set
PIM(1): Add GigabitEthernet2.550/10.5.10.10 to (10.2.7.2, 225.0.0.1), Forward
state, by PIM SG Join
PIM(1): Insert (10.2.7.2,225.0.0.1) join in nbr 213.7.7.7's queue
PIM(1): Building Join/Prune packet for nbr 213.7.7.7
PIM(1): Adding v2 (10.2.7.2/32, 225.0.0.1), S-bit Join
PIM(1): Send v2 join/prune to 213.7.7.7 (Lspvif0)
```

```
! CSR7
Received v2 Join/Prune on Lspvif0 from 213.5.5.5, to us
Join-list: (10.2.7.2/32, 225.0.0.1), S-bit set
MDT next hop change from: 0 to 17 for (10.2.7.2, 225.0.0.1) Lspvif0
Add Lspvif0/213.5.5.5 to (10.2.7.2, 225.0.0.1), Forward state, by PIM SG Join
```

```
R7#show mpls mldp database summary | include ^17
17          P2MP          213.7.7.7          [gid 65536 (0x00010000)]          3
```

Now that the debug revealed the P2MP tree in use, we can verify the GID by checking the BGP-AS Type-1 I-PMSI message to confirm our findings. The tunnel-type 2 indicates an mLDP P2MP tree and the GID is 0x10000, which is 65536.

```
R7#show bgp ipv4 mvpn vrf MC route-type 1 213.7.7.7
BGP routing table entry for [1][213:7][213.7.7.7]/12, version 185
Paths: (1 available, best #1, table MVPNv4-BGP-Table, not advertised to EBGp
peer)
  Advertised to update-groups:
    3
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (213.7.7.7)
      Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
      Community: no-export
      Extended Community: RT:213:7
      PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 17, label: exp-null,
tunnel parameters: 0600 0104 D507 0707 0007 0100 0400 0100 00
      rx pathid: 0, tx pathid: 0x0
```

At this point CSR7 can start sending native multicast traffic down the C(S,G) tree. The C-RP will want to see traffic on the SPT before telling the DR to stop registering. Once it does, we see the big 'T' flag to indicate having seeing traffic along the SPT. The C-RP then issues a prune for this C(S,G) towards CSR5 as it no longer wants the traffic. It also sends a register-stop to the DR (CSR7) so that router stops sending both unicast-encapsulated traffic inside register messages and native multicast concurrently. Notice that CSR7 also tells us that this group will continue to use the default MDT since there was no match in the data MDT ACL.

```
R10#show ip mroute 225.0.0.1 10.2.7.2 | begin \
(10.2.7.2, 225.0.0.1), 00:13:13/00:02:20, flags: PT
  Incoming interface: GigabitEthernet2.550, RPF nbr 10.5.10.5
  Outgoing interface list: Null

! CSR10
PIM(0): Received v2 Register on GigabitEthernet2.550 from 10.2.7.7
      for 10.2.7.2, group 225.0.0.1
PIM(0): Send v2 Register-Stop to 10.2.7.7 for 10.2.7.2, group 225.0.0.1
PIM(0): Insert (10.2.7.2,225.0.0.1) prune in nbr 10.5.10.5's queue
PIM(0): Building Join/Prune packet for nbr 10.5.10.5
PIM(0): Adding v2 (10.2.7.2/32, 225.0.0.1), S-bit Prune

! CSR7
PIM(1): Received v2 Register-Stop on GigabitEthernet2.574 from 10.5.10.10
PIM(1): for source 10.2.7.2, group 225.0.0.1
```

```
PIM(1): Removing register encap tunnel (Tunnel2) as forwarding interface of
(10.2.7.2, 225.0.0.1).
PIM(1): Clear Registering flag to 10.5.10.10 for (10.2.7.2/32, 225.0.0.1)
PIM(1): MDT ACL not matched for (10.2.7.2,225.0.0.1)
```

All of the interested PEs also send C(S,G) joins towards CSR7. This allows CSR7 to leave the interface in a forwarding state.

```
! CSR7
PIM(1): Received v2 Join/Prune on Lspvif0 from 213.8.8.8, to us
PIM(1): Join-list: (10.2.7.2/32, 225.0.0.1), S-bit set
PIM(1): Update Lspvif0/213.8.8.8 to (10.2.7.2, 225.0.0.1), Forward state, by
PIM SG Join

PIM(1): Received v2 Join/Prune on Lspvif0 from 213.6.6.6, to us
PIM(1): Join-list: (10.2.7.2/32, 225.0.0.1), S-bit set
PIM(1): Update Lspvif0/213.6.6.6 to (10.2.7.2, 225.0.0.1), Forward state, by
PIM SG Join

PIM(1): Received v2 Join/Prune on Lspvif0 from 213.1.1.1, to us
PIM(1): Join-list: (10.2.7.2/32, 225.0.0.1), S-bit set
PIM(1): Update Lspvif0/213.1.1.1 to (10.2.7.2, 225.0.0.1), Forward state, by
PIM SG Join
```

Unlike some other examples, no Type-5 Source Active route is created. Although technically an AD route, the Type 5 is mostly to inform other PEs of an active source when BGP c-mcast signaling is in use. There has been no change in BGP AD signaling since CSR2 started sending traffic. No Type-3 was generated by CSR7 since this group is not candidate for S-PMSI treatment.

We can verify the C-MRIB entries for this C(S,G) on all receiver PEs. Because these all use the I-PMSI, there is no special MDT for them, and thus they do not reveal any MDT details, even with the “verbose” command. Only CSR7, the ingress PE, reveals some MDT information. It’s also clear that the ingress PE sends traffic out the PMSI while the egress PEs receive it from the PMSI.

```
R1#show ip mroute vrf MC 225.0.0.1 10.2.7.2 verbose | begin \(
(10.2.7.2, 225.0.0.1), 00:25:03/00:02:47, flags: JT
Incoming interface: Lspvif0, RPF nbr 213.7.7.7
Outgoing interface list:
GigabitEthernet2.519, Forward/Sparse, 00:25:03/00:02:04

R6#show ip mroute vrf MC 225.0.0.1 10.2.7.2 verbose | begin \(
(10.2.7.2, 225.0.0.1), 00:24:15/00:02:38, flags: JT
Incoming interface: Lspvif0, RPF nbr 213.7.7.7
Outgoing interface list:
GigabitEthernet2.546, Forward/Sparse, 00:24:15/00:02:19

R8#show ip mroute vrf MC 225.0.0.1 10.2.7.2 verbose | begin \(
```

```
(10.2.7.2, 225.0.0.1), 00:24:59/00:01:53, flags: JT
Incoming interface: Lspvif0, RPF nbr 213.7.7.7
Outgoing interface list:
GigabitEthernet2.538, Forward/Sparse, 00:24:59/00:02:19

R7#show ip mroute vrf MC 225.0.0.1 10.2.7.2 verbose | begin \((
(10.2.7.2, 225.0.0.1), 00:25:43/00:02:59, flags: FTp
Incoming interface: GigabitEthernet2.527, RPF nbr 0.0.0.0
Outgoing interface list:
Lspvif0, LSM MDT: 17 (default), Forward/Sparse, 00:25:43/00:03:05, p
```

Now that the control-plane is verified, we will verify the data-plane. First we will check basic multicast packet counts on each device.

```
R1#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | begin ^Group
Group: 225.0.0.1, Source count: 1, Packets forwarded: 818, Packets received:
818
Source: 10.2.7.2/32, Forwarding: 817/0/122/0, Other: 817/0/0

R6#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | begin ^Group
Group: 225.0.0.1, Source count: 1, Packets forwarded: 865, Packets received:
865
Source: 10.2.7.2/32, Forwarding: 864/0/122/0, Other: 864/0/0

R7#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | begin ^Group
Group: 225.0.0.1, Source count: 1, Packets forwarded: 865, Packets received:
866
Source: 10.2.7.2/32, Forwarding: 865/0/117/0, Other: 866/1/0

R8#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | begin ^Group
Group: 225.0.0.1, Source count: 1, Packets forwarded: 867, Packets received:
867
Source: 10.2.7.2/32, Forwarding: 866/0/122/0, Other: 866/0/0
```

Note that the PE facing the R-CP also forwarded a few multicast packets before it was pruned. This is why the 'T' flag is set on the C(S,G). Another 818 packets were dropped due to having a null OIL; since I-PMSI is used, all the PEs will receive this traffic, including CSR5 and XRv2.

```
R5#show ip mroute vrf MC 225.0.0.1 10.2.7.2 verbose | begin \((
(10.2.7.2, 225.0.0.1), 00:32:29/00:02:54, flags: PTX
Incoming interface: Lspvif0, RPF nbr 213.7.7.7
Outgoing interface list: Null

R5#show ip mroute vrf MC 225.0.0.1 10.2.7.2 count | begin ^Group
Group: 225.0.0.1, Source count: 1, Packets forwarded: 3, Packets received:
822
Source: 10.2.7.2/32, Forwarding: 2/0/122/0, Other: 821/1/818
```

We already determined that the LSM-ID of the I-PMSI for this MVPN is 17 (on CSR7, at least). We will use that to look at the mLDP bindings to see the downstream labels. Traffic towards XRv3 uses label 93011 while traffic to XRv4 uses label 94015.

```
R7#show mpls mldp database id 17 | section Replic
Replication client(s):
MDT (VRF MC)
  Uptime      : 23:58:18      Path Set ID   : None
  Interface   : Lspvif0
213.13.13.13:0
  Uptime      : 23:58:18      Path Set ID   : None
  Out label (D) : 93011      Interface     : GigabitEthernet2.573*
  Local label (U) : None      Next Hop      : 213.7.13.13
213.14.14.14:0
  Uptime      : 23:58:18      Path Set ID   : None
  Out label (D) : 94015      Interface     : GigabitEthernet2.574*
  Local label (U) : None      Next Hop      : 213.7.14.14
```

Using the same EPC configuration on CSR7, we verify the LSM leaving CSR7's upstream interfaces. We can see the proper labels encoded in hex below. Again, the timestamps, source MAC address, and IP payload are identical as CSR7 is replicating this traffic. These labels are different than the ones used for the IPv4 SSM flow (10.2.7.2, 232.0.0.5) so we know we are using a different delivery tree.

```
R7#show monitor capture CAP buffer detailed
2 122 0.123986 00:50:56:A9:EA:77 -> 00:50:56:A9:DB:37 MPLS unicast
0000: 005056A9 DB370050 56A9EA77 81000DF5 .PV..7.PV..w....
0010: 884716B5 31FE4500 00640D15 0000FE01 .G..1.E..d.....
0020: BD7E0A02 0702E100 00010800 97620002 .~.....b..
0030: 04130000 00000FAF D323ABCD ABCDABCD .....#.....

3 122 0.123986 00:50:56:A9:EA:77 -> 00:50:56:A9:DE:77 MPLS unicast
0000: 005056A9 DE770050 56A9EA77 81000DF6 .PV..w.PV..w....
0010: 884716F3 F1FE4500 00640D15 0000FE01 .G....E..d.....
0020: BD7E0A02 0702E100 00010800 97620002 .~.....b..
0030: 04130000 00000FAF D323ABCD ABCDABCD .....#.....
```

For brevity, we will check the LFIBs of XRv3 and XRv4 and move on. Because this is the I-PMSI tree, the traffic is delivered to all PEs. We can also see that CSR5 selected XRv3 as the preferred path to CSR7 (ECMP). This is where it sent its label mapping message.

```
RP/0/0/CPU0:XRv3#show mpls forwarding labels 93011
Local  Outgoing  Prefix      Outgoing  Next Hop      Bytes
Label  Label      or ID       Interface
-----
93011  5011       MLDP: 0x00018 Gi0/0/0/0.553 213.5.13.5    0
        6003       MLDP: 0x00018 Gi0/0/0/0.563 213.6.13.6    0
        92018       MLDP: 0x00018 Gi0/0/0/0.523 213.12.13.12  0
```

```
RP/0/0/CPU0:XRv4#show mpls forwarding labels 94015
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94015	1011	MLDP: 0x0001a	Gi0/0/0/0.514	213.1.14.1	0
	8017	MLDP: 0x0001a	Gi0/0/0/0.584	213.8.14.8	0

Last, we will test IPv6 SSM. This does not have an S-PMSI associated with it, so we expect it to use the I-PMSI tunnel rooted at CSR6. We can verify this by checking the BGP AD information. There are no Type-3 routes anywhere in the “south-side” of the IPv6 MVPN network.

```
R6#show bgp ipv6 mvpn vrf MC | begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 213:6 (default for vrf MC)					
*> [1] [213:6] [213.6.6.6]/12					
	::			32768	?
*>i [1] [213:6] [213.7.7.7]/12					
	213.7.7.7	0	100	0	?
*>i [1] [213:6] [213.8.8.8]/12					
	213.8.8.8	0	100	0	?

Verifying the current C-MRIB information, we can see the C(S,G) tree has already been built within an RP for (2001:10:4:6::4, FF33::1). Traffic enters CSR6 from the customer (CSR4) and goes to the PMSI, and the receivers accept traffic from the PMSI and deliver to remote customers.

```
R6#show ipv6 mroute vrf MC FF33::1 | begin \((
(2001:10:4:6::4, FF33::1), 23:59:45/00:02:57, flags: sT
Incoming interface: GigabitEthernet2.546
RPF nbr: 2001:10:4:6::4
Immediate Outgoing interface list:
Lspvif0, Forward, 23:59:45/00:02:57
```

```
R7#show ipv6 mroute vrf MC FF33::1 | begin \((
(2001:10:4:6::4, FF33::1), 3d01h/never, flags: sTI
Incoming interface: Lspvif0
RPF nbr: ::FFFF:213.6.6.6
Immediate Outgoing interface list:
GigabitEthernet2.527, Forward, 3d01h/never
```

```
R8#show ipv6 mroute vrf MC FF33::1 | begin \((
(2001:10:4:6::4, FF33::1), 00:00:04/never, flags: sTI
Incoming interface: Lspvif0
RPF nbr: ::FFFF:213.6.6.6
Immediate Outgoing interface list:
GigabitEthernet2.538, Forward, 00:00:04/never
```

CSR4 begins sending traffic and we check the packet counters.

```
R4#ping FF33::1 repeat 10000
Output Interface: GigabitEthernet2.546

R6#show ipv6 mroute vrf MC FF33::1 count | include HW
  HW Forwarding: 22/0/118/0, Other: 0/0/0

R7#show ipv6 mroute vrf MC FF33::1 count | include HW
  HW Forwarding: 30/0/126/0, Other: 0/0/0

R8#show ipv6 mroute vrf MC FF33::1 count | include HW
  HW Forwarding: 24/0/126/0, Other: 0/0/0
```

To determine the outgoing label, we can look at the tunnels rooted at CSR6 with a GID of 65536. We know that is the I-PMSI from this MVPN as it was shown many times earlier as the opaque information inside the BGP Type-1 I-PMSI route. We will check the database to extract the LSM-ID. CSR6 sends LSM to XRv3 using label 93002.

```
R6#show mpls mldp database summary | include 213.6.6.6.*65536
17      P2MP      213.6.6.6      [gid 65536 (0x00010000)]      2

R6#show mpls mldp database id 17 | section Replic
Replication client(s):
  MDT   (VRF MC)
    Uptime      : 1d00h          Path Set ID   : None
    Interface   : Lspvif0
    213.13.13.13:0
    Uptime      : 1d00h          Path Set ID   : None
    Out label (D) : 93002        Interface     : GigabitEthernet2.563*
    Local label (U): None        Next Hop      : 213.6.13.13
```

We can see that XRv3 replicates this along the I-PMSI towards XRv2, CSR5, XRv4, and CSR7. CSR5 and CSR7 are receivers and do not forward it on, while XRv4 replicates it towards CSR1 and CSR8. Only CSR7 and CSR8 really needed the traffic, so S-PMSI could have optimized the bandwidth usage here (at the expense of state). Because there are few receivers but many PEs, using the I-PMSI for this flow is highly inefficient.

```
RP/0/0/CPU0:XRv3#show mpls forwarding labels 93002
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
93002	5010	MLDP: 0x00016	Gi0/0/0/0.553	213.5.13.5	0
	7016	MLDP: 0x00016	Gi0/0/0/0.573	213.7.13.7	0
	92017	MLDP: 0x00016	Gi0/0/0/0.523	213.12.13.12	0
	94014	MLDP: 0x00016	Gi0/0/0/0.534	213.13.14.14	0


```
RP/0/0/CPU0:XRv4#show mpls forwarding labels 94014
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94014	1010	MLDP: 0x00019	Gi0/0/0/0.514	213.1.14.1	0
	8016	MLDP: 0x00019	Gi0/0/0/0.584	213.8.14.8	0

Last, we test OAM. I described earlier how there is no easy keyword for GID in XE, so we have to manually specify the opaque type of 0x01 in hexadecimal. By specifying the root of the LSP, and also testing this from the root (tree is unidirectional and downstream), we can quickly see all of the endpoints. This tests the LSPs for any oddities that may occur in the path. From CSR1, we will test the IPv4 and IPv6 I-PMSI tunnels. Notice that the IPv4 one has 5 peers while the IPv6 only has 2, due to VPNv4/v6 topology differences. I also quickly test the IPv4 S-PMSI for the SSM group from the root, CSR7. Notice that the opaque value changes to reflect the different GID.

```
R1#ping mpls mldp p2mp 213.1.1.1 hex 0x01 00010000
p2mp Root node addr 213.1.1.1
Opaque type hex value (0x1), num hex digits 4
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
    timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]
```

```
Request #1
! reply addr 213.6.13.6
! reply addr 213.12.13.12
! reply addr 213.7.14.7
! reply addr 213.8.14.8
! reply addr 213.5.14.5
```

```
Round-trip min/avg/max = 9/74/177 ms
```

```
R1#ping mpls mldp p2mp 213.1.1.1 hex 0x01 00020000
p2mp Root node addr 213.1.1.1
Opaque type hex value (0x1), num hex digits 4
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
    timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]
```

```
Request #1
! reply addr 213.12.13.12
! reply addr 213.5.14.5
```

```
Round-trip min/avg/max = 63/127/191 ms
```

```
R7#ping mpls mldp p2mp 213.7.7.7 hex 0x01 00010001
p2mp Root node addr 213.7.7.7
```

```
Opaque type hex value (0x1), num hex digits 4
```

```
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,  
    timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200  
[snip]
```

```
Type escape sequence to abort.
```

```
Request #1
```

```
! reply addr 213.6.13.6
```

```
! reply addr 213.8.14.8
```

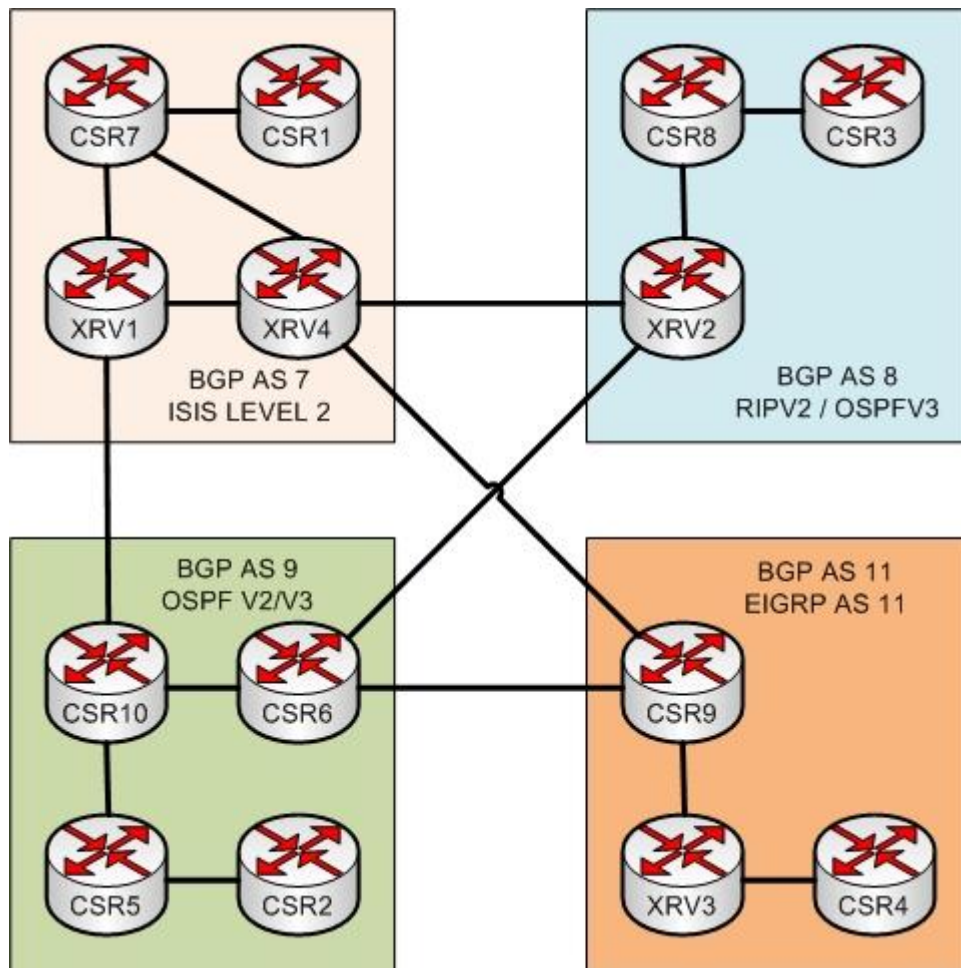
```
Round-trip min/avg/max = 10/95/181 ms
```

Additional Reading – Reference configurations “mvpn-17”

29. Describe and optimize multicast scale and performance

29.1 Inter-AS Multicast and Multicast Source Discovery Protocol (MSDP)

This section uses a large BGP topology with multiple ASes to demonstrate how multicast can flow across Internet boundaries. The basic topology is shown below. AS 7 and AS 9 use route-reflection to reduce the number of iBGP peerings, while AS 8 and AS 11 do not because they contain only 2 routers each. Every BGP peering negotiates four AFIs: IPv4 unicast, IPv4 multicast, IPv6 unicast, and IPV6 multicast. This allows us to quickly adjust the path attribute, next-hops, and other features without having to change the BGP topology mid-stride. Each AS also uses its own IGP or set of IGPs to achieve reachability for IPv4 and IPv6 as shown in the diagram.



Before doing any advanced MSDP or multicast configuration, we will verify the configurations. The basic configuration is very long only because there are many protocols running, but there is nothing complex yet. As an example, I show the BGP configurations on CSR10 and XRV1 only. I create a pair of peer groups (XR neighbor-groups) for IPv4 and IPv6 peers; an alternative approach would be peer-session and peer-policy templates. IPv4 unicast and IPv4 multicast are enabled on the "IBGP" group with IPv6 unicast and IPv6 multicast enabled on the "IBGPV6" group. I chose to show CSR10 and XRV1 because they are the route-reflectors within their respective AS boundaries and would have the most interesting configuration. Notice there are no advanced features configured, and I do not show the IGP configuration because it is very simple.

```
! CSR10
route-map RM_CONN_TO_BGP permit 10
  match interface Loopback0

router bgp 9
  bgp log-neighbor-changes
  no bgp default ipv4-unicast
  neighbor IBGP peer-group
  neighbor IBGP remote-as 9
```

```
neighbor IBGP update-source Loopback0
neighbor IBGP timers 10 40
neighbor IBGPV6 peer-group
neighbor IBGPV6 remote-as 9
neighbor IBGPV6 update-source Loopback0
neighbor IBGPV6 timers 10 40
neighbor 9.0.0.5 peer-group IBGP
neighbor 9.0.0.6 peer-group IBGP
neighbor 10.10.11.11 remote-as 7
neighbor 2009::5 peer-group IBGPV6
neighbor 2009::6 peer-group IBGPV6
neighbor FD00:10:10:11::11 remote-as 7
```

```
address-family ipv4
  redistribute connected route-map RM_CONN_TO_BGP
  neighbor IBGP send-community
  neighbor IBGP route-reflector-client
  neighbor IBGP next-hop-self
  neighbor 9.0.0.5 activate
  neighbor 9.0.0.6 activate
  neighbor 10.10.11.11 activate
```

```
address-family ipv4 multicast
  neighbor IBGP send-community
  neighbor IBGP route-reflector-client
  neighbor IBGP next-hop-self
  neighbor 9.0.0.5 activate
  neighbor 9.0.0.6 activate
  neighbor 10.10.11.11 activate
```

```
address-family ipv6
  redistribute connected route-map RM_CONN_TO_BGP
  neighbor IBGPV6 send-community
  neighbor IBGPV6 route-reflector-client
  neighbor IBGPV6 next-hop-self
  neighbor 2009::5 activate
  neighbor 2009::6 activate
  neighbor FD00:10:10:11::11 activate
```

```
address-family ipv6 multicast
  neighbor IBGPV6 send-community
  neighbor IBGPV6 route-reflector-client
  neighbor IBGPV6 next-hop-self
  neighbor 2009::5 activate
  neighbor 2009::6 activate
  neighbor FD00:10:10:11::11 activate
```

```
! XRv1
route-policy PASS
```

```

    pass
end-policy

router bgp 7
  bgp cluster-id 7.0.0.11
  address-family ipv4 unicast
    network 7.0.0.11/32

  address-family ipv4 multicast

  address-family ipv6 unicast
    network 2007::b/128

  address-family ipv6 multicast

  neighbor-group IBGP
    remote-as 7
    timers 10 40
    update-source Loopback0
    address-family ipv4 unicast
      route-reflector-client
      next-hop-self

    address-family ipv4 multicast
      route-reflector-client
      next-hop-self

  neighbor-group IBGPV6
    remote-as 7
    timers 10 40
    update-source Loopback0
    address-family ipv6 unicast
      route-reflector-client
      next-hop-self

    address-family ipv6 multicast
      route-reflector-client
      next-hop-self

  neighbor 2007::7
    use neighbor-group IBGPV6

  neighbor 7.0.0.7
    use neighbor-group IBGP

  neighbor 2007::e
    use neighbor-group IBGPV6

  neighbor 7.0.0.14

```

```

use neighbor-group IBGP

neighbor 10.10.11.10
  remote-as 9
  address-family ipv4 unicast
    route-policy PASS in
    route-policy PASS out

  address-family ipv4 multicast
    route-policy PASS in
    route-policy PASS out

neighbor fd00:10:10:11::10
  remote-as 9
  address-family ipv6 unicast
    route-policy PASS in
    route-policy PASS out

  address-family ipv6 multicast
    route-policy PASS in
    route-policy PASS out

```

Because all of the IPv4 BGP sessions use IPv4 peer loopbacks, and all of the IPv6 BGP sessions use IPv6 peer loopbacks, we can simply verifying BGP sessions being up rather than verify IGP reachability. The BGP sessions wouldn't be up if IGP was dysfunctional within any AS for IPv4 or IPv6. For brevity, I copy/paste the commands below on all BGP speakers. I only show some of the output (XRv1 and CSR10) for brevity. The benefit of the speeding verification approach is that the commands are valid on XE and XR platforms.

```

! All BGP speakers
show bgp ipv4 unicast summary | begin ^Neigh
show bgp ipv4 multicast summary | begin ^Neigh
show bgp ipv6 unicast summary | begin ^Neigh
show bgp ipv6 multicast summary | begin ^Neigh

```

```

R10#show bgp ipv4 unicast summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State/PfxRcd
9.0.0.5       4       9   5686   5660     6     0   0 14:57:00      1
9.0.0.6       4       9   5654   5666     6     0   0 14:56:52      2
10.10.11.11   4       7    880    966     6     0   0 14:31:06      3

```

```

R10#show bgp ipv4 multicast summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State/PfxRcd
9.0.0.5       4       9   5686   5660     6     0   0 14:57:00      1
9.0.0.6       4       9   5654   5666     6     0   0 14:56:52      2
10.10.11.11   4       7    880    966     6     0   0 14:31:06      3

```

```

R10#show bgp ipv6 unicast summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State/PfxRcd

```

```

2009::5          4          9      5680      5766          61      0      0 14:57:05      1
2009::6          4          9      5731      5770          61      0      0 14:56:50      2
FD00:10:10:11::11
                  4          7       880       966          61      0      0 14:30:59      3
R10#show bgp ipv6 multicast summary | begin ^Neigh
Neighbor        V          AS MsgRcvd MsgSent      TblVer  InQ  OutQ  Up/Down  State/PfxRcd
2009::5          4          9      5680      5766          61      0      0 14:57:05      1
2009::6          4          9      5731      5770          61      0      0 14:56:50      2
FD00:10:10:11::11
                  4          7       880       966          61      0      0 14:30:59      3

RP/0/0/CPU0:XRv1#show bgp ipv4 unicast summary | begin ^Neigh
Neighbor        Spk          AS MsgRcvd MsgSent      TblVer  InQ  OutQ  Up/Down  St/PfxRcd
7.0.0.7          0          7      5571      5267          6       0      0 14:37:05      1
7.0.0.14         0          7      5255      5266          6       0      0 14:35:14      2
10.10.11.10      0          9       966       880          6       0      0 14:31:00      1

RP/0/0/CPU0:XRv1#show bgp ipv4 multicast summary | begin ^Neigh
Neighbor        Spk          AS MsgRcvd MsgSent      TblVer  InQ  OutQ  Up/Down  St/PfxRcd
7.0.0.7          0          7      5571      5267          7       0      0 14:37:05      1
7.0.0.14         0          7      5255      5266          7       0      0 14:35:14      2
10.10.11.10      0          9       966       880          7       0      0 14:31:00      1

RP/0/0/CPU0:XRv1#show bgp ipv6 unicast summary | begin ^Neigh
Neighbor        Spk          AS MsgRcvd MsgSent      TblVer  InQ  OutQ  Up/Down  St/PfxRcd
2007::7          0          7      5560      5265          6       0      0 14:37:01      1
2007::14         0          7      5253      5265          6       0      0 14:35:14      2
fd00:10:10:11::10
                  0          9       966       880          6       0      0 14:30:53      1

RP/0/0/CPU0:XRv1#show bgp ipv6 multicast summary | begin ^Neigh
Neighbor        Spk          AS MsgRcvd MsgSent      TblVer  InQ  OutQ  Up/Down  St/PfxRcd
2007::7          0          7      5560      5265          6       0      0 14:37:01      1
2007::14         0          7      5253      5265          6       0      0 14:35:14      2
fd00:10:10:11::10
                  0          9       966       880          6       0      0 14:30:53      1

```

At this point, all test host routers (CSR1 through CSR4) should be able to ping one another. Using a TCL script, I test this quickly on these four routers (ping results not shown for brevity).

```

! CSR1 through CSR4
tclsh
foreach i {
9.2.5.2
7.1.7.1
8.3.8.3
11.4.13.4
} { ping $i timeout 1 repeat 3 }

```

Additionally, all BGP loopbacks should have reachability between one another. This will be important for the MSDP testing later. We will initiate this from the XE BGP speakers only, but that will give us a good picture as to which loopbacks are reachable since we can probe the XR loopbacks as well.

```

! All XE BGP speakers
tclsh
foreach i {
7.0.0.7
7.0.0.11
7.0.0.14
8.0.0.12
8.0.0.8
11.0.0.9
11.0.0.13
9.0.0.5
9.0.0.10
9.0.0.6
} { ping $i timeout 1 repeat 3 source loopback 0 }

```

PIMv2 is enabled on all interfaces in the topology except for on the four test hosts. We can quickly test this by looking at the IPv4 and IPv6 PIM neighbors by using copy/paste again. This is critical for multicast transport, and PIM must also be enabled on all inter-AS transit links for multicast to flow between AS boundaries.

```

! All PIM XE speakers
show ip pim neighbor | begin ^Neigh
show ipv6 pim neighbor | begin ^Neigh

```

```

! All PIM XR speakers
show pim neighbor
show pim ipv6 neighbor

```

```

R10#show ip pim neighbor | begin ^Neigh
Neighbor          Interface          Uptime/Expires    Ver    DR
Address
9.6.10.6           GigabitEthernet2.560  16:28:04/00:01:21 v2      1 / S P G
9.5.10.5           GigabitEthernet2.550  16:25:53/00:01:33 v2      1 / S P G
10.10.11.11        GigabitEthernet2.501  16:19:57/00:01:24 v2      1 / DR P G

```

```

R10#show ipv6 pim neighbor | begin ^Neigh
Neighbor Address      Interface          Uptime    Expires    Mode DR pri
FE80::5             Gi2.550           16:54:42   00:01:23 B G      1
FE80::11            Gi2.501           16:08:59   00:01:32 B G      DR 1
FE80::6             Gi2.560           16:52:11   00:01:25 B G      1

```

```

RP/0/0/CPU0:XRv1#show pim neighbor
[snip]
Neighbor Address      Interface          Uptime    Expires    DR pri    Flags
10.10.11.10          GigabitEth0/0/0/0.501  16:20:07   00:01:30 1          P
10.10.11.11*         GigabitEth0/0/0/0.501  16:20:14   00:01:15 1 (DR) B P E

```


7.11.14.11*	GigabitEth0/0/0/0.514	16:20:14	00:01:19	1		B	P	E
7.11.14.14	GigabitEth0/0/0/0.514	16:17:02	00:01:28	1	(DR)	B	P	
7.7.11.7	GigabitEth0/0/0/0.571	16:20:09	00:01:41	1		P		
7.7.11.11*	GigabitEth0/0/0/0.571	16:20:14	00:01:35	1	(DR)	B	P	E
7.0.0.11*	Loopback0	16:20:14	00:01:22	1	(DR)	B	P	E

```
RP/0/0/CPU0:XRv1#show pim ipv6 neighbor
```

```
[snip]
```

```
GigabitEthernet0/0/0/0.501
```

Neighbor Address	Uptime	Expires	DR	pri	DR	Flags
fe80::10	16:09:09	00:01:24	1			B
fe80::11*	16:09:15	00:01:22	1		(DR)	B P

```
GigabitEthernet0/0/0/0.514
```

Neighbor Address	Uptime	Expires	DR	pri	DR	Flags
fe80::11*	16:09:15	00:01:35	1			B P
fe80::14	16:08:56	00:01:39	1		(DR)	B P

```
GigabitEthernet0/0/0/0.571
```

Neighbor Address	Uptime	Expires	DR	pri	DR	Flags
fe80::7	16:09:12	00:01:23	1			B
fe80::11*	16:09:15	00:01:33	1		(DR)	B P

Next, we will configure the rendezvous points (RP). Each AS has at least one RP to support PIM sparse mode (PIM-SM). Routers not contained with the AS boundaries are used to simulate hosts that can send and receive multicast traffic. Rather than simulate this using router loopbacks, this lab will try to make the testing more realistic by having PIM designed routers (DR) perform their natural roles on the LAN segments they serve. AS 7 uses PIM bootstrap router (BSR) which is built into PIMv2; XRv1 is the RP for that AS. AS 9 uses Cisco's AutoRP which was used in PIMv1 for RP discovery and dissemination; CSR10 is the RP for that AS. AS 8 and AS 11 use the static RP method given their small size. CSR9 is the RP for AS 11 and XRv2 is the RP for AS 8. Because the focus of this lab is inter-AS multicast and MSDP, we will not be examining all of the advanced PIM features here. I configure the RPs for IPv4 only; IPv6 inter-AS multicast is a very different topic and is examined later.

```
! XRv1
router pim
 address-family ipv4
   bsr candidate-bsr 7.0.0.11 hash-mask-len 30 priority 1
   bsr candidate-rp 7.0.0.11 priority 192 interval 60
```

```
! CSR10
ip pim autorp listener
ip pim send-rp-announce Loopback0 scope 1
ip pim send-rp-discovery Loopback0 scope 2
```

```
! CSR6 and CSR5
ip pim autorp listener
```

```

! CSR9
ip pim rp-address 11.0.0.9

! XRv3
router pim
 address-family ipv4
   rp-address 11.0.0.9

! XRv2
router pim
 address-family ipv4
   rp-address 8.0.0.12

! CSR8
ip pim rp-address 8.0.0.12

```

First, we will verify the routers in AS 7 have all learned the RP through BSR. In this case, the RP and BSR address is 7.0.0.11; I highlight the RP address in yellow and the BSR address in green. This is straightforward with no surprises.

```

R7#show ip pim rp mapping
PIM Group-to-RP Mappings
Group(s) 224.0.0.0/4
  RP 7.0.0.11 (?), v2
    Info source: 7.0.0.11 (?), via bootstrap, priority 192, holdtime 150
    Uptime: 00:03:00, expires: 00:01:40

```

```

RP/0/0/CPU0:XRv1#show pim rp mapping
PIM Group-to-RP Mappings
Group(s) 224.0.0.0/4
  RP 7.0.0.11 (?), v2
    Info source: 7.0.0.11 (?), elected via bsr, priority 192, holdtime 150
    Uptime: 00:03:48, expires: 00:01:56

```

```

RP/0/0/CPU0:XRv4#show pim rp mapping
PIM Group-to-RP Mappings
Group(s) 224.0.0.0/4
  RP 7.0.0.11 (?), v2
    Info source: 7.11.14.11 (?), elected via bsr, priority 192, holdtime 150
    Uptime: 00:04:16, expires: 00:02:28

```

Next, we verify the RP mappings in AS 9. The immediate problem we find with PIM and AutoRP is that this dynamic RP signaling leaks beyond AS boundaries. This is generally undesirable unless the ASes share an RP, which is highly unlikely in real life and certainly not the case in this lab. The most obvious solution would be to ensure ASes did not have reachability between RPs at all, but that would break inter-AS multicast entirely. Since the loopbacks are advertised into BGP, reachability exists and RPF

passes, which is why RP information is currently allowed to leak. On CSR10, we can see two RP mappings, both from BSR and AutoRP.

```
R10#show ip pim rp mapping
PIM Group-to-RP Mappings
This system is an RP (Auto-RP)
This system is an RP-mapping agent (Loopback0)

Group(s) 224.0.0.0/4
  RP 9.0.0.10 (?), v2v1
    Info source: 9.0.0.10 (?), elected via Auto-RP
    Uptime: 00:06:30, expires: 00:02:28
  RP 7.0.0.11 (?), v2
    Info source: 7.0.0.11 (?), via bootstrap, priority 192, holdtime 150
    Uptime: 00:05:23, expires: 00:02:18
```

To see which one is actually preferred, we can test it with the RP-hash command. We can see that the AutoRP is preferred, which would technically work for intra-AS multicast in AS 9, but it is very sloppy. The reason AS 7 did not learn the AutoRP mappings is because they were not configured to listen for it. Additionally, those mappings are flooded in dense-mode (which is what the listen command facilitates) which isn't operational on the XR boxes presently.

```
R10#show ip pim rp-hash 224.1.1.1
  RP 9.0.0.10 (?), v2v1
    Info source: 9.0.0.10 (?), elected via Auto-RP
    Uptime: 00:09:43, expires: 00:02:14
PIMv2 Hash Value (mask 255.255.255.252)
  RP 9.0.0.10, via Auto-RP
  RP 7.0.0.11, via bootstrap, priority 192, hash value 930306691
```

We can look at CSR9 to see that it learns XRv1 as the RP, which overrides the static configuration. This is a bigger deal since their intra-AS multicast would now be broken as the local RP would not be used. We could use the "override" keyword on CSR9 so that the static mapping (of equal prefix length) overrides any dynamic mappings. This is also a sloppy solution and is not preferable in this situation.

```
R9#show ip pim rp mapping
PIM Group-to-RP Mappings
Group(s) 224.0.0.0/4
  RP 7.0.0.11 (?), v2
    Info source: 7.0.0.11 (?), via bootstrap, priority 192, holdtime 150
    Uptime: 00:11:59, expires: 00:01:44
Group(s): 224.0.0.0/4, Static
  RP: 11.0.0.9 (?)
```

The better solution is to filter the BSR messages bi-directionally on all inter-AS links where BSR is enabled. We only need to do this on XRv1 and XRv4, but for extra protection, it makes sense to

configure it on all transit interfaces. This way, it protects the non-BSR AS from accidentally learning BSR information. We do not have the granularity to selectively filter group/RP mappings from the BSR message; we can only turn the filter off or on. I show the configuration on XRv4 and CSR6 for brevity, but I apply the “bsr-border” command on all transit links.

```
! XRv1
router pim
  address-family ipv4
    interface GigabitEthernet0/0/0/0.524
      bsr-border

    interface GigabitEthernet0/0/0/0.594
      bsr-border

! CSR6
interface GigabitEthernet2.562
  ip pim bsr-border

interface GigabitEthernet2.569
  ip pim bsr-border
```

On XE, we can verify it by checking the PIM interface summary to find an asterisk after the PIM operating mode on each interface. Alternatively, we can look at the interface details to see it confirmed more explicitly. There does not appear to be an XR show command to verify this.

```
R6#show ip pim interface gig2.569
Address      Interface          Ver/   Nbr    Query  DR      DR
              Interface          Mode   Count  Intvl  Prior
10.6.9.6     GigabitEthernet2.569 v2/S * 1      30     1       10.6.9.9

R6#show ip pim interface gig2.569 detail
GigabitEthernet2.569 is up, line protocol is up
  Internet address is 10.6.9.6/24
  Multicast switching: fast
  Multicast packets in/out: 0/26
  Multicast TTL threshold: 0
  PIM: enabled
    PIM version: 2, mode: sparse
    PIM DR: 10.6.9.9
    PIM neighbor count: 1
    PIM Hello/Query interval: 30 seconds
    PIM Hello packets in/out: 2140/2152
    PIM J/P interval: 60 seconds
    PIM State-Refresh processing: enabled
    PIM State-Refresh origination: disabled
    PIM NBMA mode: disabled
    PIM ATM multipoint signalling: disabled
    PIM domain border: enabled
```

```
PIM neighbors rpf proxy capable: TRUE
PIM BFD: disabled
PIM Non-DR-Join: FALSE
Multicast Tagswitching: disabled
```

Quickly checking CSR10 and CSR9, we can see their RP mapping tables look correct now. The RP addresses are shown in yellow and the information sources are shown in green. They no longer learn the BSR information from AS 7.

```
R10#show ip pim rp mapping
PIM Group-to-RP Mappings
This system is an RP (Auto-RP)
This system is an RP-mapping agent (Loopback0)
Group(s) 224.0.0.0/4
  RP 9.0.0.10 (?), v2v1
    Info source: 9.0.0.10 (?), elected via Auto-RP
      Uptime: 00:29:07, expires: 00:02:53
```

```
R9#show ip pim rp mapping
PIM Group-to-RP Mappings
Group(s): 224.0.0.0/4, Static
  RP: 11.0.0.9 (?)
```

Although less of a problem, the AutoRP announcements are still being sent out of AS 9. Looking at CSR6, we can confirm this, as these groups are secretly operating in dense mode. Group 224.0.1.39 is used by candidate-RPs to notify the mapping agent (MA) about their candidacy, and group 224.0.1.40 is used to disseminate the RP information to clients. CSR6 sees traffic from 9.0.0.10 towards 224.0.1.40 which are the mapping advertisements and floods these out of the AS.

```
R6#show ip mroute 224.0.1.40 9.0.0.10 | begin \((
(9.0.0.10, 224.0.1.40), 00:30:09/00:02:31, flags: LT
  Incoming interface: GigabitEthernet2.560, RPF nbr 9.6.10.10
  Outgoing interface list:
    Loopback0, Forward/Sparse, 00:30:09/stopped
    GigabitEthernet2.562, Forward/Sparse, 00:30:09/stopped
    GigabitEthernet2.569, Forward/Sparse, 00:30:09/stopped
```

These isn't an easy button to solve this, so we can manually remove groups 224.0.1.39 and 224.0.1.40 from the mroute OIL using a multicast boundary. We apply this to all transit links on all routers, but like BSR, it is most important to do it on the AS 9 ASBRs. I show the configuration on CSR6 and XRv1.

```
! CSR6
ip access-list extended ACL_DENY_AUTORP
deny ip any host 224.0.1.39
deny ip any host 224.0.1.40
permit ip any any
```

```

interface GigabitEthernet2.562
 ip multicast boundary ACL_DENY_AUTORP

interface GigabitEthernet2.569
 ip multicast boundary ACL_DENY_AUTORP

! XRv1
ipv4 access-list ACL_DENY_AUTORP
 10 deny ipv4 any host 224.0.1.39
 20 deny ipv4 any host 224.0.1.40
 30 permit ipv4 any any

multicast-routing
 address-family ipv4
   interface GigabitEthernet0/0/0/0.501
     boundary ACL_DENY_AUTORP

```

First, we check the configuration to make sure the list was applied properly. We can see the filter as a multicast boundary inbound and outbound, and we see hits against the ACL entries matching those groups. This is a good indication that the filter is working.

```

R6#show ip multicast interface gig2.569
GigabitEthernet2.569 is up, line protocol is up
 Internet address is 10.6.9.6/24
 Multicast routing: enabled
 Multicast switching: fast
 Multicast packets in/out: 0/36
 Multicast boundary: ACL_DENY_AUTORP (in/out)
 Multicast TTL threshold: 0
 Multicast Tagswitching: disabled

```

```

R6#show access-l ACL_DENY_AUTORP
Extended IP access list ACL_DENY_AUTORP
 10 deny ip any host 224.0.1.39 (2 matches)
 20 deny ip any host 224.0.1.40 (4 matches)
 30 permit ip any any

```

This is also true for XRv1, though the commands are slightly different. Looking at the MRIB before the AutoRP group times out, we can see that XRv1 did see traffic from 9.0.0.10 for 224.0.1.40. This times out soon after the filters are applied and the entry is removed.

```

RP/0/0/CPU0:XRv1#show mrib route 224.0.1.40 9.0.0.10
[snip]
(9.0.0.10,224.0.1.40) RPF nbr: 0.0.0.0 Flags: RPF
 Up: 00:04:18
 Outgoing Interface List
   Loopback0 Flags: F IC, Up: 00:04:18

```

```
GigabitEthernet0/0/0/0.514 Flags: F, Up: 00:04:18
GigabitEthernet0/0/0/0.571 Flags: F, Up: 00:04:18
```

```
RP/0/0/CPU0:XRv1#show mrib route 224.0.1.40 9.0.0.10
```

```
No matching route in MRIB route-DB
```

We can further confirm it worked by checking the (S,G) state for the AutoRP mapping group to ensure it is only being locally consumed (forwarded to loopback0 only) or forwarded within the AS. This ensures that AS 9, the AS using AutoRP, doesn't leak information out of its AS in the first place.

```
R6#show ip mroute 224.0.1.40 9.0.0.10 | begin \((
(9.0.0.10, 224.0.1.40), 00:35:15/00:02:22, flags: LT
  Incoming interface: GigabitEthernet2.560, RPF nbr 9.6.10.10
  Outgoing interface list:
    Loopback0, Forward/Sparse, 00:35:15/stopped
```

With multiple disparate RPs servicing overlapping ASM groups, we must use MSDP to build a topology in which they can be tied together. MSDP is very similar to BGP as it is TCP based (port 639) and is designed to notify neighboring ASes about active sources within the local AS. These source active (SA) messages are like "long-range" PIM register messages. The general value of this is that any RPs with existing (*,G) trees in a remote AS will be notified of active sources in the local AS. These remote RPs can issue (S,G) joins back to the source in another AS, which may not even transit the MSDP peers (doesn't matter; MSDP is used for signaling only). The original intent of MSDP is that it would mirror the BGP topology exactly, at least with respect to RPs. Though MSDP can be configured on non-RPs, this only makes sense in hierarchical architectures where some aggregation nodes might reduce the overall amount of MSDP peers required. For now, we will configure MSDP peers between the four ASes in a near-full mesh except AS 8 and AS 11 will not have a direct peering. The basic MSDP peer configuration is simple, and the remote-AS field is optional. If not configured, it can be derived from BGP or set to zero, depending on software versions. I personally prefer to set the value for completeness.

```
! CSR10
ip msdp peer 7.0.0.11 connect-source Loopback0 remote-as 7
ip msdp peer 8.0.0.12 connect-source Loopback0 remote-as 8
ip msdp peer 11.0.0.9 connect-source Loopback0 remote-as 11

! CSR9
ip msdp peer 7.0.0.11 connect-source Loopback0 remote-as 7
ip msdp peer 9.0.0.10 connect-source Loopback0 remote-as 9

! XRv1
router msdp
  peer 8.0.0.12
    connect-source Loopback0
    remote-as 8

  peer 9.0.0.10
```

```

connect-source Loopback0
remote-as 9

peer 11.0.0.9
connect-source Loopback0
remote-as 11

! XRv2
router msdp
peer 7.0.0.11
connect-source Loopback0
remote-as 7

peer 9.0.0.10
connect-source Loopback0
remote-as 9

```

When the peers start to come up, MSDP prints syslog messages to inform us. Below are the log messages on CSR10 and XRv1 from their mutual peering being established.

```

! CSR10
%MSDP-5-PEER_UPDOWN: Session to peer 7.0.0.11 going up

! XRv1
msdp[1045]: %ROUTING-MSDP-5-INIT_PEER_UP_DOWN : MSDP peer up: 9.0.0.10

```

To see the status of all peers, the MSDP summary table displays it clearly on both XE and XR. Since AS 8 and AS 11 don't peer directly, we can conclude that all peer are up by checking CSR10 and XRv1. The "peer name" is the DNS-resolved name for the peer, and the question mark means that resolution was not performed. This is not an error condition and has no effect on MSDP operations.

```
R10#show ip msdp summary
```

```
MSDP Peer Status Summary
```

Peer Address	AS	State	Uptime/ Downtime	Reset Count	SA Count	Peer Name
7.0.0.11	7	Up	00:01:43	0	0	?
8.0.0.12	8	Up	00:01:46	0	0	?
11.0.0.9	11	Up	00:03:51	0	0	?

```
RP/0/0/CPU0:XRv1#show msdp summary
```

```
Out of Resource Handling Enabled
```

```
Maximum External SA's Global : 20000
```

```
Current External Active SAs : 0
```

```
MSDP Peer Status Summary
```

Peer Address	AS	State	Uptime/ Downtime	Reset Count	Peer Name	Active SA Cnt	Cfg.Max Ext.SAs	TLV recv/sent
8.0.0.12	8	Up	00:02:46	0	?	0	0	6/6
9.0.0.10	9	Up	00:02:46	0	?	0	0	3/6


```
11.0.0.9      11      Up      00:00:15  0      ?      0      0      1/1
```

We can look at the details on each peer, much like we would with the “show bgp neighbor” command. There is not much interesting traffic at present, but we learn some new things about MSDP. We can see that the SA filtering options are very flexible as we can apply route-maps in/out to filter RP information and (S,G) information. The outputs on both routers also show us detailed statistics for messages exchanged. One of the most important message counters is the MSDP RPF failure, which is discussed in detail later. Last, MSDP is also capable of MD5 authentication, much like BGP. Notice that XR shows the keep-alive timer at 30 seconds with a dead timer of 2.5 times that (75 seconds).

```
R10#show ip msdp peer 7.0.0.11
MSDP Peer 7.0.0.11 (?), AS 7 (configured AS)
Connection status:
  State: Up, Resets: 0, Connection source: Loopback0 (9.0.0.10)
  Uptime(Downtime): 00:05:31, Messages sent/received: 6/12
  Output messages discarded: 0
  Connection and counters cleared 00:09:14 ago
SA Filtering:
  Input (S,G) filter: none, route-map: none
  Input RP filter: none, route-map: none
  Output (S,G) filter: none, route-map: none
  Output RP filter: none, route-map: none
SA-Requests:
  Input filter: none
Peer ttl threshold: 0
SAs learned from this peer: 0
Number of connection transitions to Established state: 1
  Input queue size: 0, Output queue size: 0
MD5 signature protection on MSDP TCP connection: not enabled
Message counters:
  RPF Failure count: 0
  SA Messages in/out: 0/0
  SA Requests in: 0
  SA Responses out: 0
  Data Packets in/out: 0/0
```

```
RP/0/0/CPU0:XRv1#show msdp peer 9.0.0.10
MSDP Peer 9.0.0.10 (?), AS 9
Description:
Connection status:
  State: Up, Resets: 0, Connection Source: 7.0.0.11
  Uptime(Downtime): 00:06:04, SA messages received: 0
  TLV messages sent/received: 13/7
  Output messages discarded: 0
  Connection and counters cleared 00:06:04 ago
SA Filtering:
  Input (S,G) filter: none
  Input RP filter: none
```

```

Output (S,G) filter: none
Output RP filter: none
SA-Requests:
Input filter: none
Sending SA-Requests to peer: disabled
Password: None
Peer ttl threshold: 2
Input queue size: 0, Output queue size: 0
KeepAlive timer period: 30
Peer Timeout timer period: 75
NSR:
State: Unknown, Oper-Downs: 0
NSR-Uptime(NSR-Downtime): 2w0d

```

To confirm that MSDP uses TCP port 639, we can check the TCP details to see the sessions established. Both CSR10 and XRv1 show 3 MSDP sessions, each to the proper peer. We also note that the lower IP address is responsible for opening the MSDP session (that is, the lower peer IP is the TCP client).

```

R10#show tcp brief | include 639
7FBC42BF0C40  9.0.0.10.639      8.0.0.12.65296      ESTAB
7FBC42BF7E20  9.0.0.10.30326      11.0.0.9.639        ESTAB
7FBC42BF15C0  9.0.0.10.639      7.0.0.11.30561      ESTAB

RP/0/0/CPU0:XRv1#show tcp brief | include :639
0x10151770 0x60000000  0  0  7.0.0.11:24200  8.0.0.12:639      ESTAB
0x101b1638 0x60000000  0  0  7.0.0.11:30561  9.0.0.10:639      ESTAB
0x101c71fc 0x60000000  0  0  7.0.0.11:23252  11.0.0.9:639      ESTAB
0x101b32a8 0x60000000  0  0  0.0.0.0:639     0.0.0.0:0         LISTEN
0x101b1a84 0x00000000  0  0  0.0.0.0:639     0.0.0.0:0         LISTEN

```

We will configure MD5 authentication between CSR10 and XRv1. Normally, one would configure this everywhere, but for brevity I limit it to one connection. Key-chains are not supported. We can verify this by checking the peer detailed and filtering for the important output.

```

! CSR10
ip msdp password peer 7.0.0.11 AS7AS9_MSDP

! XRv1
router msdp
peer 9.0.0.10
password clear AS7AS9_MSDP

R10#show ip msdp peer 7.0.0.11 | include MD5
MD5 signature protection on MSDP TCP connection: enabled

RP/0/0/CPU0:XRv1#show msdp peer 9.0.0.10 | include Password
Password: Configured, set on active socket

```

We can also verify this with low-level TCP commands. Recovering the TCB from the TCP brief for a particular TCP session, we look at the option detailed to find MD5 negotiated. This does not appear visible on XR.

```
R10#show tcp brief | include 7.0.0.11
7FBC42BF6068 9.0.0.10.639 7.0.0.11.45268 ESTAB
```

```
R10#show tcp tcb 7FBC42BF6068 | begin ^Option
Option Flags: md5, Retrans timeout
[snip]
```

Next, we will adjust the session timers between CSR10 and XRv1. For faster dead-peer detection, I will reduce the keep-alive timer to 15 seconds and the maximum time to wait for a message (dead-timer, basically) to 35 seconds.

```
! CSR10
ip msdp keepalive 7.0.0.11 15 35

! XRv1
router msdp
 peer 9.0.0.10
  keepalive 15 35
```

This information does not appear visible on XE, so we will verify it on XR only.

```
RP/0/0/CPU0:XRv1#show msdp peer 9.0.0.10 | include timer period
KeepAlive timer period: 15
Peer Timeout timer period: 35
```

Next, we will actually send some multicast data between sites. CSR3 will request traffic for group 225.0.1.3, which will trigger a (*,G) join from the PIM-DR (CSR8) to the local RP (XRv2). XRv2 is the root of the shared tree and is waiting to hear about active sources via PIM register messages (intra-AS) or MSDP SA messages (inter-AS).

```
! CSR3
interface GigabitEthernet2.538
 ip igmp join-group 225.0.1.3
```

```
R8#show ip igmp groups 225.0.1.3
IGMP Connected Group Membership
Group Address      Interface          Uptime    Expires    Last Reporter
Group Accounted
225.0.1.3          GigabitEthernet2.538 00:00:19  00:02:40  8.3.8.3
```

```
R8#show ip mroute 225.0.1.3 | begin \((
```

```
(*, 225.0.1.3), 00:00:31/00:02:28, RP 8.0.0.12, flags: SJC
Incoming interface: GigabitEthernet2.582, RPF nbr 8.8.12.12
Outgoing interface list:
GigabitEthernet2.538, Forward/Sparse, 00:00:31/00:02:28

RP/0/0/CPU0:XRv2#show pim topology 225.0.1.3 | begin 225
(*, 225.0.1.3) SM Up: 00:00:46 RP: 8.0.0.12*
JP: Join(never) RPF: Decapstunnel0, 8.0.0.12 Flags:
GigabitEthernet0/0/0/0.582 00:00:46 fwd Join(00:02:43)
```

CSR2 will be the first sender for this group. Before starting, we enable some PIM and MSDP debugging on various to understand the process.

```
! CSR5
debug ip pim 225.0.1.3

! CSR10
debug ip pim 225.0.1.3
debug ip msdp 8.0.0.12 peer
debug ip msdp 8.0.0.12 detail

! XRv2
debug msdp peer 9.0.0.10
debug mrrib route

! CSR2
R2#ping 225.0.1.3 repeat 10000
```

As the PIM DR, CSR5 senses traffic on the LAN and originates a PIM register message towards the local RP, which is CSR10. CSR10 is not aware of any receivers for this group and would normally reply immediately with a PIM register-stop message. This time, CSR10 first originates an MSDP source-active (SA) message to notify XRv2 that there is an active source. After that, it continues with its normal PIM operation of sending a register-stop. We can see that, generally speaking, the MSDP actions happen before the PIM ones as indicated by the debug.

```
! CSR5
PIM(0): Check RP 9.0.0.10 into the (*, 225.0.1.3) entry
PIM(0): Building Triggered (*,G) Join / (S,G,RP-bit) Prune message for
225.0.1.3
PIM(0): Adding register encap tunnel (Tunnel1) as forwarding interface of
(9.2.5.2, 225.0.1.3).

! CSR10
MSDP(0): 8.0.0.12: Building SA message from SA cache
MSDP(0): start_index = 0, sa_cache_index = 0, Qlen = 0
MSDP(0): Sent entire sa-cache, sa_cache_index = 0, Qlen = 0
PIM(0): Received v2 Register on GigabitEthernet2.550 from 9.5.10.5
```

```

    for 9.2.5.2, group 225.0.1.3
PIM(0): Check RP 9.0.0.10 into the (*, 225.0.1.3) entry
PIM(0): Adding register decap tunnel (Tunnel2) as accepting interface of (*,
225.0.1.3).
PIM(0): Adding register decap tunnel (Tunnel2) as accepting interface of
(9.2.5.2, 225.0.1.3).
MSDP(0): 8.0.0.12: Send 100-byte SA encapsulated data for (9.2.5.2,
225.0.1.3), RP 9.0.0.10

```

Multiple things are happening concurrently now, so we will look at CSR10 first. After sending the MSDP SA message, it doesn't wait to receive an (S,G) join from the remote AS and immediately sends a PIM register-stop to CSR5. CSR5 receives this and, as far as it knows, there are no receivers for this group, so the DR stops sending register messages. As seen above, the SA message carries that 100-byte ICMP ping as a payload, much like a register message does.

```

! CSR10
PIM(0): Send v2 Register-Stop to 9.5.10.5 for 9.2.5.2, group 225.0.1.3

! CSR5
PIM(0): Received v2 Register-Stop on GigabitEthernet2.550 from 9.0.0.10
PIM(0):   for source 9.2.5.2, group 225.0.1.3
PIM(0): Removing register encap tunnel (Tunnel1) as forwarding interface of
(9.2.5.2, 225.0.1.3).
PIM(0): Clear Registering flag to 9.0.0.10 for (9.2.5.2/32, 225.0.1.3)

```

On the other side of the network, XRv2 receives the MSDP SA message from CSR10. The MSDP debugging doesn't reveal much human-readable output, but we can see that a message was received from MSDP and an MRIB (S,G) entry of (9.2.5.2, 225.0.1.3) was created. This is a good indication that it successfully processed the SA message from CSR10.

```

! XRv2
msdp[1045]: [1] default 9.0.0.10 peer: peer 9.0.0.10 timeout timer started.
Expiry in 75 secs
rib[1146]:   [ 27] TID: 0xe0000000 (9.2.5.2,225.0.1.3) Added E*, #A=0, #F=0,
#MDT_A=0, Route Ver = 0x2c30
rib[1146]:   [ 12] TID: 0xe0000000 (9.2.5.2,225.0.1.3) Updated E C* RPF*,
#A=1, #F=1, #MDT_A=0, RPF=8.0.0.12 [Gi0/0/0/0.582 F* NS*] [De0 A*], Route Ver
= 0x2c33
msdp[1045]: [1] default 7.0.0.11 peer: peer 7.0.0.11 timeout timer started.
Expiry in 75 secs
mrrib[1146]:   [ 27] TID: 0xe0000000 (9.2.5.2,225.0.1.3) Unchanged
mrrib[1146]:   [ 12] TID: 0xe0000000 (9.2.5.2,225.0.1.3) Updated E C RPF*,
#A=1, #F=1, #MDT_A=0, RPF=10.6.12.6 [Gi0/0/0/0.562 NS*], Route Ver = 0x2c34

```

In lieu of cryptic debug, we can verify the SA-cache on XRv2 to ensure it sees the entry. We can see the (S,G), followed the RPF source (more on this later) and the remote AS. The encapsulated data received is

a byte counter for the amount of traffic carried inside of the SA message; like PIM register messages, the payload is transferred to MSDP SA messages when they are originated. The 100-byte ICMP echo was tunneled across while the inter-AS SPT was first built.

```
RP/0/0/CPU0:XRv2#show msdp sa-cache 225.0.1.3
MSDP Flags:
E - set MRIB E flag , L - domain local source is active,
EA - externally active source, PI - PIM is interested in the group,
DE - SAs have been denied. Timers age/expiration,
Cache Entry:
(9.2.5.2, 225.0.1.3), RP 9.0.0.10, MBGP/AS 9, 00:08:55/00:01:50
  Learned from peer 9.0.0.10, RPF peer 9.0.0.10
  SAs recvd 10, Encapsulated data received: 100
  grp flags: PI, src flags: E, EA, PI
```

It is important to note that the SA cache shows received SA messages. As such, CSR10 has no entries, despite originating one. Like BGP, we can see the advertised SAs by invoking the peer command with a special option. This shows the SAs advertised to a certain peer. SA messages are categorized by whether they were advertised by their presence in the MRIB (local origination) or forwarded on from the SA cache (like a string of MSDP peers).

```
R10#show ip msdp sa-cache 225.0.1.3
MSDP Source-Active Cache
Group not found

R10#show ip msdp peer 8.0.0.12 advertised-SAs
MSDP SA advertised to peer 8.0.0.12 (?) from mroute table

225.0.1.3      9.2.5.2 (?)
MSDP SA advertised to peer 8.0.0.12 (?) from SA cache
```

We can further verify the MSDP statistics on XRv2. We can see that 30 SAs have been received (periodic updates) while none of have been sent. The output is similar on CSR10 which shows 30 outbound SAs and none received. CSR10 also shows a single “data” packet out, which is the 100-byte ICMP echo that was MSDP-encapsulated as a carry-over from the original PIM register message from CSR5.

```
RP/0/0/CPU0:XRv2#show msdp statistics peer 9.0.0.10
MSDP Peer Statistics :- default
Peer 9.0.0.10 : AS is 9, State is Up, 1 active SAs
  TLV Rcvd : 33 total
              3 keepalives, 0 notifications
              30 SAs, 0 SA Requests
              0 SA responses, 0 unknowns
  TLV Sent : 59 total
              59 keepalives, 0 notifications
              0 SAs, 0 SA Requests
```

```
0 SA responses
SA msgs : 30 received, 0 sent
```

```
R10#show ip msdp peer 8.0.0.12 | begin Message_count
Message counters:
  RPF Failure count: 0
  SA Messages in/out: 0/30
  SA Requests in: 0
  SA Responses out: 0
  Data Packets in/out: 0/1
```

The more significant effect of having received this SA message is that now XRv2 can originate an (S,G) join back to 9.2.5.2 since it is aware of a new source. We will casually glaze over the MSDP RPF rules for now, but we see that the RPF peer for the originating RP is CSR6, and MSDP instructs the router to consult the RIB for this information.

```
P/0/0/CPU0:XRv2#show msdp rpf 9.0.0.10
RPF peer for 9.0.0.10 is 9.0.0.10 AS 9, rule: 1
  bgp/rib lookup: nexthop: 9.0.0.10, asnum: 9
```

```
RP/0/0/CPU0:XRv2#show route 9.0.0.10
Routing entry for 9.0.0.10/32
  Known via "bgp 8", distance 20, metric 0
  Tag 9, type external
  Routing Descriptor Blocks
    10.6.12.6, from 10.6.12.6, BGP external
    Route metric is 0
  No advertising protos.
```

With this information, XRv2 sends the (S,G) join to CSR6. The MRIB entry also has the 'E' flag set which means MSDP external, equivalent to the 'M' flag in XE.

```
RP/0/0/CPU0:XRv2#show pim topology 225.0.1.3 9.2.5.2 | begin 9.2
(9.2.5.2,225.0.1.3)SPT SM Up: 00:19:10
JP: Join(00:00:39) RPF: GigabitEthernet0/0/0/0.562,10.6.12.6 Flags:
KAT(00:02:04) E RA
  GigabitEthernet0/0/0/0.582 00:19:09 fwd Join(00:03:02)
```

CSR6 and CSR10 continue building the SPT by sending the (S,G) join towards CSR5. Notice that CSR10 has the "A" flag set to indicate that this was a candidate for MSDP advertisement, implying that an SA may have been advertised for it. CSR6 and CSR5 have no idea MSDP is even in use, and they do not need to know.

```
R6#show ip mroute 225.0.1.3 9.2.5.2 | begin \
(9.2.5.2, 225.0.1.3), 00:19:38/00:01:35, flags: T
  Incoming interface: GigabitEthernet2.560, RPF nbr 9.6.10.10
```

Outgoing interface list:

GigabitEthernet2.562, Forward/Sparse, 00:19:38/00:02:51

```
R10#show ip mroute 225.0.1.3 9.2.5.2 | begin \((
```

```
(9.2.5.2, 225.0.1.3), 00:19:47/00:02:00, flags: TA
```

Incoming interface: GigabitEthernet2.550, RPF nbr 9.5.10.5

Outgoing interface list:

GigabitEthernet2.560, Forward/Sparse, 00:19:47/00:03:24

```
R5#show ip mroute 225.0.1.3 9.2.5.2 | begin \((
```

```
(9.2.5.2, 225.0.1.3), 00:22:16/00:03:20, flags: FT
```

Incoming interface: GigabitEthernet2.525, RPF nbr 0.0.0.0

Outgoing interface list:

GigabitEthernet2.550, Forward/Sparse, 00:22:16/00:02:48

As a final check, we look at CSR8. I do not trace the SPT switchover process, which also occurs, but suffice it to say that CSR8 has the (S,G) entry as well due to SPT switchover. Counting packets forwarding to CSR3, we can see the feature is working as expected. I issue the command a few times to ensure the packet counters continue to increment.

```
R8#show ip mroute 225.0.1.3 9.2.5.2 count | begin ^Group
```

```
Group: 225.0.1.3, Source count: 1, Packets forwarded: 1372, Packets received: 1372
```

Source: 9.2.5.2/32, Forwarding: 1371/1/118/0, Other: 1371/0/0

```
R8#show ip mroute 225.0.1.3 9.2.5.2 count | begin ^Group
```

```
Group: 225.0.1.3, Source count: 1, Packets forwarded: 1422, Packets received: 1422
```

Source: 9.2.5.2/32, Forwarding: 1421/1/118/0, Other: 1421/0/0

To test a more complex example, we will originate traffic from CSR4 for group 225.0.1.3 at the same time. Since AS 11 and AS 8 do not have a direct MSDP connection, AS 7 and AS 9 will need to forward the SA generated by CSR9, the local RP. This behavior is the MSDP default; this is different than iBGP which doesn't reflect iBGP-learned routes by default. MSDP is very liberally with SA advertisements and relies on RPF for loop prevention. To test, we can simply start the ping and after enabling some debug. I use different MSDP debugs on XRv2 to try and get more valuable information. I also disable the PIM debugging since that process is the same, essentially, and we are trying to learn MSDP.

```
! CSR9, CSR10
```

```
debug ip msdp detail
```

```
debug ip msdp peer
```

```
! XRv2, XRv1
```

```
debug msdp cache
```

```
debug msdp events
```

```
! CSR4
```



```
ping 225.0.1.3 repeat 10000 timeout 1
```

Upon received the PIM register message from XRv3, CSR9 originates the MSDP SA for (11.4.13.4, 225.0.1.3). CSR9 sends it to both XRv1 and CSR10 as we can see from peer advertised-SA output.

```
! CSR9
```

```
MSDP(0): 9.0.0.10: Originating SA message
```

```
MSDP(0): start_index = 0, mroute_cache_index = 0, Qlen = 0
```

```
MSDP(0): Sent entire mroute table, mroute_cache_index = 0, Qlen = 0
```

```
MSDP(0): 9.0.0.10: No entries found
```

```
MSDP(0): 9.0.0.10: Building SA message from SA cache
```

```
MSDP(0): start_index = 0, sa_cache_index = 0, Qlen = 0
```

```
MSDP(0): Sent entire sa-cache, sa_cache_index = 0, Qlen = 0
```

```
R9#show ip msdp peer 7.0.0.11 advertised-SAs
```

```
MSDP SA advertised to peer 7.0.0.11 (?) from mroute table
```

```
225.0.1.3      11.4.13.4 (?)
```

```
MSDP SA advertised to peer 7.0.0.11 (?) from SA cache
```

```
225.0.1.3      9.2.5.2 (?) RP: 9.0.0.10
```

```
R9#show ip msdp peer 9.0.0.10 advertised-SAs
```

```
MSDP SA advertised to peer 9.0.0.10 (?) from mroute table
```

```
225.0.1.3      11.4.13.4 (?)
```

```
MSDP SA advertised to peer 9.0.0.10 (?) from SA cache
```

First, we will look at CSR10. It receives the message from CSR9 with encapsulation data inside. This passes the MSDP RPF check (discussed in detail later), which means it is added to the SA cache and forwarded to other MSDP peers, except the one from which the SA was received. Because XRv1 also received this SA from CSR9 directly, it performs the same action; install the SA into the cache and advertise it onward.

```
! CSR10
```

```
MSDP(0): 11.0.0.9: Received 120-byte msg 506 from peer
```

```
MSDP(0): 11.0.0.9: SA TLV, len: 120, ec: 1, RP: 11.0.0.9, with data
```

```
MSDP(0): 11.0.0.9: Peer RPF check passed for 11.0.0.9, peer is RP
```

```
MSDP(0): WAVL Insert SA Source 11.4.13.4 Group 225.0.1.3 RP 11.0.0.9  
Successful
```

```
MSDP(0): 7.0.0.11: Forward 120-byte SA (11.4.13.4, 225.0.1.3) from 11.0.0.9  
to 7.0.0.11
```

```
MSDP(0): 8.0.0.12: Forward 120-byte SA (11.4.13.4, 225.0.1.3) from 11.0.0.9  
to 8.0.0.12
```

```
! XRv1
```

```
msdp[1045]: [1] default cache: adding remote RP 11.0.0.9
```

```
msdp[1045]: [1] default (11.4.13.4, 225.0.1.3) cache: new entry - rpf: 11.0.0.9
```

This means that CSR10 and XRv1 are both going to advertise the SA for (11.4.13.4, 225.0.1.3) to one another as well. When CSR10 receives it, this packet is rejected due to failing the MSDP RPF check. The same is true on XRv1 for the SA received from CSR10. This is perfectly logical and is loosely analogous to the BGP AS-path loop-prevention mechanism. Even without understanding the details of MSDP RPF yet, we can look at the diagram and generally understand its purpose. CSR10 and XRv1 both received the SA directly from CSR9, so accepting the same SA from a second-hand peer is less desirable.

```
! CSR10
MSDP(0): Received 120-byte TCP segment from 7.0.0.11
MSDP(0): Append 120 bytes to 0-byte msg 507 from 7.0.0.11, qs 1
MSDP(0): 7.0.0.11: Received 120-byte msg 507 from peer
MSDP(0): 7.0.0.11: SA TLV, len: 120, ec: 1, RP: 11.0.0.9, with data
MSDP(0): 7.0.0.11: Peer RPF check failed for 11.0.0.9, EBGp route/peer in AS 7/0
```

```
! XRv1
msdp[1045]: [1] default cache: rejecting SA from peer 9.0.0.10 for RP 11.0.0.9
```

To prove that each router installed the proper SA entry, we can check the caches. Both of them show 11.0.0.9 as the RPF peer (as opposed to one another) which is correct given the BGP topology.

```
RP/0/0/CPU0:XRv1#show msdp sa-cache 225.0.1.3
MSDP Flags:
E - set MRIB E flag , L - domain local source is active,
EA - externally active source, PI - PIM is interested in the group,
DE - SAs have been denied. Timers age/expiration,
Cache Entry:
(9.2.5.2, 225.0.1.3), RP 9.0.0.10, MBGP/AS 9, 00:48:42/00:02:07
  Learned from peer 9.0.0.10, RPF peer 9.0.0.10
  SAs recvd 53, Encapsulated data received: 100
  grp flags: none, src flags: EA
(11.4.13.4, 225.0.1.3), RP 11.0.0.9, MBGP/AS 11, 00:09:50/00:02:17
  Learned from peer 11.0.0.9, RPF peer 11.0.0.9
  SAs recvd 12, Encapsulated data received: 100
  grp flags: none, src flags: EA

R10#show ip msdp sa-cache 225.0.1.3
MSDP Source-Active Cache - 1 entries for 225.0.1.3
(11.4.13.4, 225.0.1.3), RP 11.0.0.9, BGP/AS 11, 00:10:02/00:05:49, Peer 11.0.0.9
```

There is a problem on XRv2, however. Despite both XRv1 and CSR10 advertising their SA messages to XRv2, XRv2 rejects them both. The debug reveals this and the SA cache has no entries originating in AS

11. The only entry is the first one we tested earlier from AS 9. XRv2 is claiming that both of these SAs are failing RPF.

```
! XRv2
msdp[1045]: [1] default cache: rejecting SA from peer 7.0.0.11 for RP
11.0.0.9
msdp[1045]: [1] default cache: rejecting SA from peer 9.0.0.10 for RP
11.0.0.9
```

```
RP/0/0/CPU0:XRv2#show msdp sa-cache 225.0.1.3
MSDP Flags:
E - set MRIB E flag , L - domain local source is active,
EA - externally active source, PI - PIM is interested in the group,
DE - SAs have been denied. Timers age/expiration,
Cache Entry:
(9.2.5.2, 225.0.1.3), RP 9.0.0.10, MBGP/AS 9, 00:51:05/00:01:57
  Learned from peer 9.0.0.10, RPF peer 9.0.0.10
  SAs recvd 55, Encapsulated data received: 100
  grp flags: PI, src flags: E, EA, PI
```

XR has a special debug command for troubleshooting MSDP RPF issues. I enable it on XRv2 to get more details, along with MSDP TLV debugging which will reveal the contents of the SA TLVs. The router is indicating that there is an RPF route towards 11.0.0.9 (via BGP) but there isn't an MSDP peer. This is by design as we never configured a peer between those ASes.

```
RP/0/0/CPU0:XRv2#debug msdp rpf
RP/0/0/CPU0:XRv2#debug msdp tlv
```

```
msdp[1045]: [1] default 7.0.0.11 tlv: ReadQ: 20 bytes in buffer
msdp[1045]: [1] default 7.0.0.11 tlv: ReadQ: TLV parsed length = 20, type = 1
msdp[1045]: [1] default 7.0.0.11 tlv: 7.0.0.11: recvd <SourceActive,20>
msdp[1045]: [1] default 7.0.0.11 tlv: RP 11.0.0.9 has 1 entries
msdp[1045]: [1] default 7.0.0.11 tlv: -- (11.4.13.4/32 225.0.1.3)
msdp[1045]: [1] default 7.0.0.11 tlv:
msdp[1045]: [1] 11.0.0.9 RPF: Found RPF but no rpf peer for 11.0.0.9 - Failed
msdp[1045]: [1] 11.0.0.9 RPF: RPF lookup for 11.0.0.9 failed for peer
7.0.0.11: No Peer preferred
```

```
msdp[1045]: [1] default 9.0.0.10 tlv: ReadQ: 20 bytes in buffer
msdp[1045]: [1] default 9.0.0.10 tlv: ReadQ: TLV parsed length = 20, type = 1
msdp[1045]: [1] default 9.0.0.10 tlv: 9.0.0.10: recvd <SourceActive,20>
msdp[1045]: [1] default 9.0.0.10 tlv: RP 11.0.0.9 has 1 entries
msdp[1045]: [1] default 9.0.0.10 tlv: -- (11.4.13.4/32 225.0.1.3)
msdp[1045]: [1] default 9.0.0.10 tlv:
msdp[1045]: [1] 11.0.0.9 RPF: Found RPF but no rpf peer for 11.0.0.9 - Failed
msdp[1045]: [1] 11.0.0.9 RPF: RPF lookup for 11.0.0.9 failed for peer
9.0.0.10: No Peer preferred
```

The debug also reveals a semi-frightening error message that isn't exactly intuitive. Clearly there are BGP routes for this remote RP, but the debug seems to indicate there is an error.

```
! XRv2
msdp[1045]: [2] 11.0.0.9 RPF: BGP route not found for 11.0.0.9: 'sysdb'
detected the 'warning' condition 'A SysDB client tried to access a
nonexistent item or list an empty directory'
```

To better understand MSDP RPF, we will discuss its operation and purpose. There are three rules to MSDP and they are applied depending on the BGP topology. First, the MSDP process tries to find a BGP neighbor that has the same IP address as the MSDP peer.

1. If the MSDP peer is also an iBGP peer, apply Rule 1.
2. If the MSDP peer is also an eBGP peer, apply Rule 2.
3. If the MSDP peer is not a BGP peer at all, apply Rule 3.

The rules are all very similar in nature. I describe them in detail below, and then summarize them afterwards. To identify some terms initially, the “originating RP” is the RP that originated the SA message. In our case, this would be 11.0.0.9. The “advertising peer” is the MSDP peer that advertised the SA message in question, which may or not be the same as the originating RP. In this case, 9.0.0.10 and 7.0.0.11 are the advertising peers.

Rule 1 (iBGP peer): Check the BGP MRIB for the best path to the originating RP. If no match exists, check the BGP URIB instead. If no match exists, RPF fails. If a path is found, the receiving RP compares the BGP neighbor address (not the next-hop) to the MSDP peer address; they must match. In summary, this rule means that the BGP and MSDP topologies should match within an AS.

Rule 2 (eBGP peer): Check the BGP MRIB for the best path to the originating RP. If no match exists, check the BGP URIB instead. If no match exists, RPF fails. If a path is found, the first AS in the path towards the originating RP must match the peer AS of the eBGP peer. If it doesn't, RPF fails. In summary, the BGP and MSDP topologies should match between ASes. Compared to rule 1, the MSDP peer address and BGP peer address do not have to match, but the general AS advertisement path must match.

Rule 3 (no BGP peer): Check the BGP MRIB for the best path to the originating RP. If no match exists, check the BGP URIB instead. If no match exists, RPF fails. If a path is found, the BGP MRIB is searched for the best path to the advertising peer, followed by the BGP URIB if a path isn't found. If a path is found, the first AS in the path towards the originating RP must match the AS of the MSDP peer. In summary, this rule is similar to rule 2 but performs two different lookups; one for the originating RP and one for the advertising peer.

Summary of rules: All of them check the BGP MRIB for a path to the originating RP, followed by the BGP URIB if a path is not found. For all lookups, this is the order of operations. The difference is in the follow-on activities which may introduce additional checks depending on the peer type.

Troubleshooting XRv2, we check the BGP neighbor tables for IPv4 unicast and multicast. 11.0.0.9 is not a BGP peer of any kind, so we assume MSDP rule 3 is invoked.

```
RP/0/0/CPU0:XRv2#show bgp ipv4 unicast summary | begin ^Neigh
Neighbor      Spk    AS  MsgRcvd  MsgSent    TblVer    InQ  OutQ  Up/Down  St/PfxRcd
8.0.0.8        0     8    170     160        20     0     0  00:24:11      2
10.6.12.6       0     9     46     43         20     0     0  00:24:49     11
10.12.14.14     0     7     45     43         20     0     0  00:24:49     11

RP/0/0/CPU0:XRv2#show bgp ipv4 multicast summary | begin ^Neigh
Neighbor      Spk    AS  MsgRcvd  MsgSent    TblVer    InQ  OutQ  Up/Down  St/PfxRcd
8.0.0.8        0     8    170     161         13     0     0  00:24:20      1
10.6.12.6       0     9     46     43         13     0     0  00:24:58      3
10.12.14.14     0     7     45     43         13     0     0  00:24:58      3
```

First, we check the BGP MRIB (IPv4 multicast) for a path to the originating RP (11.0.0.9). We find no match there, but we do in the BGP URIB (IPv4 unicast). The best path is via XRv4 due to being the oldest eBGP route. The first AS in the best path is AS 7.

```
RP/0/0/CPU0:XRv2#show bgp ipv4 multicast 11.0.0.9
% Network not in table

RP/0/0/CPU0:XRv2#show bgp ipv4 unicast 11.0.0.9
[snip]
  Path #1: Received by speaker 0
  Not advertised to any peer
  9 11
    10.6.12.6 from 10.6.12.6 (9.0.0.6)
      Origin incomplete, localpref 100, valid, external, group-best
      Received Path ID 0, Local Path ID 0, version 0
      Origin-AS validity: not-found
  Path #2: Received by speaker 0
  Advertised to update-groups (with more than one peer):
    0.3
  Advertised to peers (in unique update groups):
    8.0.0.8
    7 11
      10.12.14.14 from 10.12.14.14 (7.0.0.14)
        Origin incomplete, localpref 100, valid, external, best, group-best
        Received Path ID 0, Local Path ID 1, version 14
        Origin-AS validity: not-found
```

Since we found a path, XRv2 progresses to run the same checks against the advertising peer (7.0.0.11). XRv4 is the best path due to the shortest AS-path length.

```

RP/0/0/CPU0:XRv2#show bgp ipv4 multicast 7.0.0.11
% Network not in table

RP/0/0/CPU0:XRv2#show bgp ipv4 unicast 7.0.0.11
[snip]
  Path #1: Received by speaker 0
  Not advertised to any peer
  9 7
    10.6.12.6 from 10.6.12.6 (9.0.0.6)
      Origin IGP, localpref 100, valid, external, group-best
      Received Path ID 0, Local Path ID 0, version 0
      Origin-AS validity: not-found
  Path #2: Received by speaker 0
  Advertised to update-groups (with more than one peer):
    0.3
  Advertised to peers (in unique update groups):
    8.0.0.8
  7
    10.12.14.14 from 10.12.14.14 (7.0.0.14)
      Origin IGP, localpref 100, valid, external, best, group-best
      Received Path ID 0, Local Path ID 1, version 6
      Origin-AS validity: not-found

```

Last, the router verifies that the first AS in the best-path to the originating RP (11.0.0.9) is the same as the advertising MSDP peer (7.0.0.11). Both of them are 7, so RPF should pass according to the rules above. We saw earlier that it did not, which leads us to question the aforementioned RPF rule logic. In an attempt to repair this, I will make the MSDP RPF check “more correct” by advertising CSR9’s loopback (the originating RP address) into the BGP MRIB.

```

! CSR9
router bgp 11
  address-family ipv4 multicast
    redistribute connected route-map RM_CONN_TO_BGP

```

This means that XRv2 will learn BGP MRIB routes for the originating RP now instead of relying on the BGP URIB. This shouldn’t be required per the rules described earlier, but perhaps XR assumes that because IPv4 multicast is running, that is the only AFI it checks. Now, it prefers CSR6 as the best-path due to being the oldest eBGP route.

```

RP/0/0/CPU0:XRv2#show bgp ipv4 multicast 11.0.0.9/32
[snip]
  9 11
    10.6.12.6 from 10.6.12.6 (9.0.0.6)
      Origin incomplete, localpref 100, valid, external, best, group-best
      Received Path ID 0, Local Path ID 1, version 14
  Path #2: Received by speaker 0

```

```

Not advertised to any peer
7 11
10.12.14.14 from 10.12.14.14 (7.0.0.14)
Origin incomplete, localpref 100, valid, external, group-best
Received Path ID 0, Local Path ID 0, version 0

```

Suddenly, RPF passes on XRv2. Everything else we verified above is still correct, but the route to the originating RP changed from being unicast to multicast. Examining the debug on XRv2, we no longer see the “scary” BGP messages about missing routes. Based on this, I assert that the XR debug message was referring to the lack of an IPv4 multicast BGP route specifically, but didn’t make that entirely clear. We will later prove that **when XR BGP is running multicast and unicast AFIs, the RPF process expects to always have a routing entry in the multicast BGP RIB for RPF**. Without it, RPF will fail, as BGP unicast routes are not considered in this case.

```

! XRv2
msdp[1045]: [1] default 9.0.0.10 tlv: ReadQ: 20 bytes in buffer
msdp[1045]: [1] default 9.0.0.10 tlv: ReadQ: TLV parsed length = 20, type = 1
msdp[1045]: [1] default 9.0.0.10 tlv: 9.0.0.10: recvd <SourceActive,20>
msdp[1045]: [1] default 9.0.0.10 tlv: RP 11.0.0.9 has 1 entries
msdp[1045]: [1] default 9.0.0.10 tlv: -- (11.4.13.4/32 225.0.1.3)
msdp[1045]: [1] default 9.0.0.10 tlv:
msdp[1045]: [1] 11.0.0.9 RPF: Found RPF peer 9.0.0.10 for RP 11.0.0.9 -
Matched Highest in AS (iv)
msdp[1045]: [1] 11.0.0.9 RPF: Accepting SA from peer 9.0.0.10 for RP 11.0.0.9
due to Matched Highest in AS (iv)

msdp[1045]: [1] default 7.0.0.11 tlv: ReadQ: 20 bytes in buffer
msdp[1045]: [1] default 7.0.0.11 tlv: ReadQ: TLV parsed length = 20, type = 1
msdp[1045]: [1] default 7.0.0.11 tlv: 7.0.0.11: recvd <SourceActive,20>
msdp[1045]: [1] default 7.0.0.11 tlv: RP 11.0.0.9 has 1 entries
msdp[1045]: [1] default 7.0.0.11 tlv: -- (11.4.13.4/32 225.0.1.3)
msdp[1045]: [1] default 7.0.0.11 tlv:
msdp[1045]: [1] 11.0.0.9 RPF: Found RPF peer 9.0.0.10 for RP 11.0.0.9 -
Matched Highest in AS (iv)
msdp[1045]: [1] 11.0.0.9 RPF: RPF lookup for 11.0.0.9 failed for peer
7.0.0.11: 9.0.0.10 preferred

```

When we check the RPF details on XRv2, we see that MSDP rule 4 is used. This appears very confusing and the word “rule” in XR not does correlate with the MSDP “rules” we just discussed. We can at least see that 9.0.0.10 is the next-hop which has a valid BGP URIB route (the BGP peer and MSDP peer), which allows RPF to work.

```

RP/0/0/CPU0:XRv2#show msdp rpf 11.0.0.9
RPF peer for 11.0.0.9 is 9.0.0.10 AS 9, rule: 4
bgp/rib lookup: nexthop: 9.0.0.10, asnum: 11

```

The XR “rules” are specific to RFC3618 which describe conditions for RPF checks but this is not made clear in the documentation. This is enabled on XR and cannot be disabled, yet the default on IOS and XE platforms is to use the Cisco “rules” discussed earlier. Unfortunately, both RFC3618 and Cisco call them “MSDP RPF rules” so there isn’t an easy way to distinguish them in the show commands or debug outputs. One would have to simply know the difference, the rule descriptions, and the defaults per platform. XE platforms can optionally use RFC3618 rules which we will investigate later. The RFC3618 rules are also processed in top-down fashion with lower number rules being the more preferred RPF mechanisms. RFC3618, and the debug output on XR, list the rules in Roman numeral format.

- i. The advertising MSDP peer is the same address as the originating RP.
- ii. The advertising MSDP peer is the eBGP next-hop for the RPF route.
- iii. The advertising MSDP peer is the BGP peer that advertised the route towards the originating RP, but it isn’t the same BGP peer address as the next-hop.
- iv. The advertising MSDP peer resides in the closest AS towards the originating RP.
- v. The advertising MSDP peer RPF was statically configured.

This raises the question of how RPF passed on the original SA when the originating RP (9.0.0.10) wasn’t in the MRIB either. This was from our first test where CSR2 was the sender and CSR10 was the originating RP. We can examine XRv2 MSDP RPF debugs to see which RFC3618 rule was used to accept this SA from CSR10. This debug can be confusing because it suggests “rule 1” was used, when clearly the routes to CSR10 are eBGP, not iBGP as the “Cisco MSDP RPF rules” specify. Again, this is because XR is following “RFC3618 RPF rule 1”; when the originating RP and advertising peer are the same address, RPF passes immediately. Under Cisco’s rules, RPF isn’t even checked in this case, as this is a special case clearly documented. The debug clearly shows that the advertising peer and originating RP are the same.

```
! XRv2
msdp[1045]: [1] default 9.0.0.10 tlv: ReadQ: 40 bytes in buffer
msdp[1045]: [1] default 9.0.0.10 tlv: ReadQ: TLV parsed length = 20, type = 1
msdp[1045]: [1] default 9.0.0.10 tlv: 9.0.0.10: recvd <SourceActive,20>
msdp[1045]: [1] default 9.0.0.10 tlv: RP 9.0.0.10 has 1 entries
msdp[1045]: [1] default 9.0.0.10 tlv: -- (9.2.5.2/32 225.0.1.3)
msdp[1045]: [1] default 9.0.0.10 tlv:
msdp[1045]: [1] 9.0.0.10 RPF: Found RPF peer 9.0.0.10 for RP 9.0.0.10 -
Matched RP Address (i)
msdp[1045]: [1] 9.0.0.10 RPF: Accepting SA from peer 9.0.0.10 for RP 9.0.0.10
due to Matched RP Address (i)
```

Going back to the second test with CSR4 as a sender, we can verify the SA cache on XRv2 to see the new entry. The AS is 11 with the remote RP being 11.0.0.9. Although there is no MSDP or BGP peering with this address, RPF passes due to the IPv4 multicast route for the originating RP advertised by CSR6 and XRv4 (best path dependent).

```
RP/0/0/CPU0:XRv2#show msdp sa-cache
MSDP Flags:
```


E - set MRIB E flag , L - domain local source is active,
 EA - externally active source, PI - PIM is interested in the group,
 DE - SAs have been denied. Timers age/expiration,
 Cache Entry:
 (9.2.5.2, 225.0.1.3), RP 9.0.0.10, MBGP/AS 9, 00:48:35/00:01:54
 Learned from peer 9.0.0.10, RPF peer 9.0.0.10
 SAs recvd 52, Encapsulated data received: 0
 grp flags: PI, src flags: E, EA, PI
 (11.4.13.4, 225.0.1.3), RP 11.0.0.9, MBGP/AS 11, 00:17:24/00:01:54
 Learned from peer 9.0.0.10, RPF peer 9.0.0.10
 SAs recvd 19, Encapsulated data received: 0
 grp flags: PI, src flags: E, EA, PI

A quick check of multicast counters on CSR8 shows this working. CSR2 has many more packets than CSR4 because of the RPF problems we had to resolve. Both CSR2 and CSR4 have been sending the whole time which is why the packet counts are high.

```
R8#show ip mroute 225.0.1.3 count | begin ^Group
Group: 225.0.1.3, Source count: 2, Packets forwarded: 3945, Packets received:
3945
  RP-tree: Forwarding: 3/0/118/0, Other: 3/0/0
  Source: 11.4.13.4/32, Forwarding: 1095/1/118/0, Other: 1095/0/0
  Source: 9.2.5.2/32, Forwarding: 2847/1/118/0, Other: 2847/0/0

R8#show ip mroute 225.0.1.3 count | begin ^Group
Group: 225.0.1.3, Source count: 2, Packets forwarded: 3965, Packets received:
3965
  RP-tree: Forwarding: 3/0/118/0, Other: 3/0/0
  Source: 11.4.13.4/32, Forwarding: 1105/1/118/0, Other: 1105/0/0
  Source: 9.2.5.2/32, Forwarding: 2857/1/118/0, Other: 2857/0/0
```

Because the eBGP peers have MSDP connections between their loopbacks (which are also RPs), we bypass many of the complex RPF issues and rules. On XE, the special case of not checking RPF applies, and on XR, RFC3618 rule 1 is always applied. We can introduce some of this complexity for additional practice. On XRv1 and CSR10, I reconfigure the MSDP session slightly. XRv1 will peer with CSR10's direct connected transit link while CSR10 continues to peer with XRv1's loopback. This is analogous with modifying the BGP update-source, except we modify the MSDP connect-source. From XRv1's perspective, this means that SA messages from CSR10 will be sourced from 10.10.11.10 (transit link) with an RP of 9.0.0.10 (loopback). Since the MSDP peer address (10.10.11.10) and RP (9.0.0.10) differ, this does not qualify for RFC3618 rule 1. The topology doesn't change much, so the general design is the same, but XR will have to select another RFC3618 RPF rule to accept SA messages from 9.0.0.10.

```
! CSR10
ip msdp peer 7.0.0.11 connect-source gig2.501 remote-as 7

! XRv1
```

```

router msdp
  no peer 9.0.0.10
  peer 10.10.11.10
  password clear AS7AS9_MSDP
  connect-source Loopback0
  remote-as 9
  keepalive 15 35

```

Since CSR2 is still sending traffic, CSR10 is still originating periodic SA messages. XRv1 will receive this whether there are clients in AS 7 or not. On XRv1, the MSDP RPF rule used in this case is RFC3618 rule 3. Despite the route to 9.0.0.10/32 being eBGP-learned, rule 3 is used. At a glance, the logic for RFC3618 rule 2 holds true since the eBGP next-hop is 10.10.11.10 which is the advertising peer as well. The debug shows RFC3618 rule 3 being matched for that particular SA received from 9.0.0.10. The reason for failing to match RFC3618 rule 2 is similar to the error on XRv2. The RFC3618 definitions, and XR in general, assume that the routes in question are BGP IPv4 multicast. Since XRv1 does not have an IPv4 multicast route to 9.0.0.10, it “falls back” to the unicast route which qualifies it for RFC3618 rule 3 RPF. We confirm this with the show command also.

```

! XRv1
msdp[1045]: [1] default 10.10.11.10 tlv: ReadQ: TLV parsed length = 20, type
= 1
msdp[1045]: [1] default 10.10.11.10 tlv: 10.10.11.10: recvd <SourceActive,20>
msdp[1045]: [1] default 10.10.11.10 tlv: RP 9.0.0.10 has 1 entries
msdp[1045]: [1] default 10.10.11.10 tlv: -- (9.2.5.2/32 225.0.1.3)
msdp[1045]: [1] default 10.10.11.10 tlv:
msdp[1045]: [1] 9.0.0.10 RPF: Found RPF peer 10.10.11.10 for RP 9.0.0.10 -
Matched Next Hop (iii)
msdp[1045]: [1] 9.0.0.10 RPF: Accepting SA from peer 10.10.11.10 for RP
9.0.0.10 due to Matched Next Hop (iii)

RP/0/0/CPU0:XRv1#show msdp rpf 9.0.0.10
RPF peer for 9.0.0.10 is 10.10.11.10 AS 9, rule: 3
  bgp/rib lookup: nexthop: 10.10.11.10, asnum: 0

```

If we want XRv1 to see this as candidate for matching RFC3618 rule 2, we would need to advertise 9.0.0.10 inside BGP IPv4 multicast. We can re-use the route-map from IPv4 unicast for this.

```

! CSR10
router bgp 9
  address-family ipv4 multicast
  redistribute connected route-map RM_CONN_TO_BGP

```

With debugging still enabled on XRv1, we can see the new match. This debug clearly shows that the next-hop was the eBGP peer which is expected. The SA is exactly the same, but since XRv1 now has an IPv4 multicast route towards the originating RP, rule 2 can be used. I also verify with the show command.

```

! XRv1
msdp[1045]: [1] default 10.10.11.10 tlv: ReadQ: 20 bytes in buffer
msdp[1045]: [1] default 10.10.11.10 tlv: ReadQ: TLV parsed length = 20, type
= 1
msdp[1045]: [1] default 10.10.11.10 tlv: 10.10.11.10: recvd <SourceActive,20>
msdp[1045]: [1] default 10.10.11.10 tlv: RP 9.0.0.10 has 1 entries
msdp[1045]: [1] default 10.10.11.10 tlv: -- (9.2.5.2/32 225.0.1.3)
msdp[1045]: [1] default 10.10.11.10 tlv:
msdp[1045]: [1] 9.0.0.10 RPF: Found RPF peer 10.10.11.10 for RP 9.0.0.10 -
Matched External BGP Peer (ii)
msdp[1045]: [1] 9.0.0.10 RPF: Accepting SA from peer 10.10.11.10 for RP
9.0.0.10 due to Matched External BGP Peer (ii)

RP/0/0/CPU0:XRv1#show msdp rpf 9.0.0.10
RPF peer for 9.0.0.10 is 10.10.11.10 AS 9, rule: 2
  bgp/rib lookup: nexthop: 10.10.11.10, asnum: 9

```

By now, it should be clear that the MSDP RPF rules can be confusing and often inconsistent between platforms. Fortunately, the debug messaging is helpful, especially on XR where the matched rules are shown. To demonstrate more differences between Cisco's version and RFC3618, we can use CSR9. It is currently using Cisco's method, which means our verifications are limited to debugs. The equivalent show command only works when RFC3618 is enabled.

```

R9#show ip msdp rpf-peer 8.0.0.12
This command requires "ip msdp rpf rfc3618" to be configured.

```

To test it, I will originate traffic for a bogus group on CSR3. Even if AS 11 doesn't have receivers, CSR9 should accept the SA message from XRv2. We saw that, in the same topology but in the reverse order, XRv2 rejected these messages under the RFC3618 construct. CSR9, however, accepts them, based on Cisco's MSDP RPF rule 3.

```

R9#debug ip msdp peer
MSDP Peer debugging is on

```

```

MSDP(0): 7.0.0.11: Received 120-byte msg 1017 from peer
MSDP(0): 7.0.0.11: SA TLV, len: 120, ec: 1, RP: 8.0.0.12, with data
MSDP(0): 7.0.0.11: Peer RPF check failed for 8.0.0.12, EBGp route/peer in AS
7/0
MSDP(0): 9.0.0.10: Received 120-byte msg 1018 from peer
MSDP(0): 9.0.0.10: SA TLV, len: 120, ec: 1, RP: 8.0.0.12, with data
MSDP(0): 9.0.0.10: Peer RPF check passed for 8.0.0.12, used EBGp peer
MSDP(0): WAVL Insert SA Source 8.3.8.3 Group 225.9.9.9 RP 8.0.0.12 Successful
MSDP(0): 7.0.0.11: Forward 120-byte SA (8.3.8.3, 225.9.9.9) from 9.0.0.10 to
7.0.0.11

```

```

R9#show ip msdp sa-cache

```

```

MSDP Source-Active Cache - 2 entries
(9.2.5.2, 225.0.1.3), RP 9.0.0.10, MBGP/AS 9, 04:43:52/00:05:35, Peer
9.0.0.10
Learned from peer 9.0.0.10, RPF peer 9.0.0.10,
SAs received: 304, Encapsulated data received: 2
(8.3.8.3, 225.9.9.9), RP 8.0.0.12, BGP/AS 8, 00:00:06/00:05:53, Peer 9.0.0.10
Learned from peer 9.0.0.10, RPF peer 9.0.0.10,
SAs received: 1, Encapsulated data received: 1

```

The path via CSR6 was chosen as the BGP best-path due to being the oldest route. Notice that CSR9 doesn't care about the lack of an IPv4 multicast route for the originating RP, as XRv2 did.

```

R9#show bgp ipv4 multicast 8.0.0.12
% Network not in table

R9#show bgp ipv4 unicast 8.0.0.12
BGP routing table entry for 8.0.0.12/32, version 47
Paths: (2 available, best #2, table default)
  Advertised to update-groups:
    3          4
  Refresh Epoch 1
  7 8
    10.9.14.14 from 10.9.14.14 (7.0.0.14)
      Origin IGP, localpref 100, valid, external
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  9 8
    10.6.9.6 from 10.6.9.6 (9.0.0.6)
      Origin IGP, localpref 100, valid, external, best
      rx pathid: 0, tx pathid: 0x0

```

Per Cisco's rule 3, the router then checks the path to the advertising peer to ensure the AS of that peer matches the first AS of the best path, which is 9. They match, so the SA is installed as we saw above. CSR9 actually has an IPv4 multicast route for this since we advertised it on CSR10 earlier, but this is irrelevant.

```

R9#show bgp ipv4 multicast 9.0.0.10
BGP routing table entry for 9.0.0.10/32, version 28
Paths: (2 available, best #2, table 8000)
  Advertised to update-groups:
    3          4
  Refresh Epoch 1
  7 9
    10.9.14.14 from 10.9.14.14 (7.0.0.14)
      Origin incomplete, localpref 100, valid, external
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  9

```

```
10.6.9.6 from 10.6.9.6 (9.0.0.6)
  Origin incomplete, localpref 100, valid, external, best
  rx pathid: 0, tx pathid: 0x0
```

Personally, I would prefer to have all MSDP speakers playing by the same “rules” in a real network design. We enable RFC3618 for RPF rules on CSR9 and observe the differences. We now expect more XR-like behavior with respect to the debug outputs and show commands. The XE platforms don’t tell you rules 1 through 5, but instead give you a brief description of why it was selected. For example, if we look at the MSDP RPF towards 9.0.0.10, we can see that the advertising peer equals the originating RP (“Peer is RP”), which is RFC3618 rule 1. Under Cisco’s rules, this would also “pass” by not even triggering the RPF check. If we repeat the show command for XRv2’s loopback on CSR9, the RFC3618 rule is 4 (“Peer is best in the closest AS”), which is consistent with how XRv2 views CSR9 (reciprocity, in a way). There is no BGP or MSDP peer with this address, and the path to 8.0.0.12 is consistent with the SA received from the MSDP peer in that AS.

```
! CSR9
ip msdp rpf rfc3618

R9#show ip msdp rpf-peer 9.0.0.10
RPF peer information for ? (9.0.0.10)
  RPF peer: ? (9.0.0.10)
  RPF route/mask: 9.0.0.10/32
  RPF rule: Peer is RP
  RPF type: mbgp

R9#show ip msdp rpf-peer 8.0.0.12
RPF peer information for ? (8.0.0.12)
  RPF peer: ? (9.0.0.10)
  RPF route/mask: 9.0.0.10/32
  RPF rule: Peer is best in the closest AS
  RPF type: mbgp
```

Examining the debugs, we can confirm these findings. From CSR10, SA messages originating from XRv2 are accepted since this is the best-path back to the originating RP. When CSR9 receives SAs from XRv1 from CSR10, these are rejected, since CSR9 is expecting to learn these directly from CSR10. This makes sense because CSR10 and CSR9 have a direct MSDP peering, so RFC3618 rule 4 is used as shown above. In order for the XE debugging to be particularly useful, one would have to memorize the five RFC3618 rules since it doesn’t give you the Roman numeral as XR does. Then again, if RPF passes and everything is functional, an administrator may have less of a reason to reveal these details.

```
! CSR9
MSDP(0): 9.0.0.10: Received 20-byte msg 1047 from peer
MSDP(0): 9.0.0.10: SA TLV, len: 20, ec: 1, RP: 8.0.0.12
MSDP(0): 9.0.0.10: RPF check passed for 8.0.0.12, Peer is best in the closest
AS
```

```

MSDP(0): 7.0.0.11: Received 20-byte msg 1044 from peer
MSDP(0): 7.0.0.11: SA TLV, len: 20, ec: 1, RP: 9.0.0.10
MSDP(0): 7.0.0.11: Peer RPF check failed for 9.0.0.10, EMBGP route/peer in AS
7/0

```

I view MSDP as being a combination of PIM and BGP in some ways. It has RPF rules for control-plane SAs just like PIM has RPF rules for register messages. It is like BGP based on its ability to relay information across ASes, along with its TCP transport and rich filtering options. We notice that MSDP peers, upon receipt of a valid SA (one that passes RPF), will automatically re-send that SA to all other MSDP peers except the peer from which it was received. While this doesn't cause any loops due to RPF rules, it can be very inefficient, especially in a meshed network. For example, when CSR9 receives SAs from CSR10, there is no reason for it to re-send them to XRv1, since XRv1 and CSR10 have a direct MSDP session. The same is true on XRv2's reception of SAs from CSR10. In BGP, the iBGP "split-horizon" rule effectively prevents this re-advertisement of routes within an AS; we can relax this rule with route-reflectors. Note: The MSDP AS information is not relevant for MSDP advertisement in terms of "split-horizon" like BGP. In MSDP, the opposite logic is used; SAs are flooded liberally by default and we can restrict their advertisement using mesh-groups. Mesh-groups are generally used for intra-AS flood reduction but we can use it for inter-AS as well. The concept is simple; SAs received from a peer in a given mesh-group can never be advertised to a peer in the same mesh-group. The assumption is that the sending MSDP peer always has a direct connection to all other peers in the mesh-group. Enabling this feature can reduce SA flooding, saving routing resources (less RPF lookups and guaranteed failures) and bandwidth (less flooding). We will configure this on all MSDP peers in accordance with the example I gave above.

```

! CSR9
ip msdp mesh-group MG9 7.0.0.11
ip msdp mesh-group MG9 9.0.0.10

! XRv2
router msdp
 peer 7.0.0.11
   mesh-group MG12

 peer 9.0.0.10
   mesh-group MG12

```

Before observing the operational effect of this feature, we will verify the configuration by checking the peer details on each router. Mesh groups are only locally significant and are not signaled in MSDP. They can be any text string and every peer can belong to a maximum of one mesh group. The way I've configured it, it is similar to a BGP route-reflector design where XRv2 and CSR9 should be clients only; that is, not route-reflectors. They should not advertise anything other than logically originated SA messages.

```

R9#show ip msdp peer | include ^MSDP_Peer|mesh

```

```
MSDP Peer 7.0.0.11 (?), AS 7 (configured AS)
  Peer is member of mesh-group MG9
MSDP Peer 9.0.0.10 (?), AS 9 (configured AS)
  Peer is member of mesh-group MG9
```

```
RP/0/0/CPU0:XRv2#show msdp peer | utility egrep '^MSDP Peer|mesh'
MSDP Peer 7.0.0.11 (?), AS 7
  Peer is a member of mesh group MG12
MSDP Peer 9.0.0.10 (?), AS 9
  Peer is a member of mesh group MG12
```

First, we will verify the behavior on CSR9. With CSR2 sending traffic, CSR10 (9.0.0.10 as the RP address and MSDP peer) will advertise SAs to all MSDP peers since there are no mesh groups configured there. When CSR9 receives it, it can only advertise it to peers in a different mesh group than CSR10. Since all peers are in the same mesh group as CSR10, CSR9 does nothing beyond process it locally. It is interesting to note that enabling mesh groups for a peer totally bypasses the MSDP RPF checks as the debug reveals. This could be a workaround for RPF issues as well. Since both messages bypass RPF, CSR9 installs the one it receives first, which came directly from CSR10. It doesn't really matter which one gets installed since the (S,G) join will traverse the shortest path to the originating RP anyway.

```
! CSR9
debug ip msdp peer
debug ip msdp detail
```

```
MSDP(0): Received 120-byte TCP segment from 9.0.0.10
MSDP(0): Append 120 bytes to 0-byte msg 3916 from 9.0.0.10, qs 1
MSDP(0): 9.0.0.10: Received 120-byte msg 3916 from peer
MSDP(0): 9.0.0.10: SA TLV, len: 120, ec: 1, RP: 9.0.0.10, with data
MSDP(0): 9.0.0.10: Peer RPF check bypassed, peer 9.0.0.10 in mesh-group MG9
MSDP(0): WAVL Insert SA Source 9.2.5.2 Group 225.0.1.3 RP 9.0.0.10 Successful
MSDP(0): Received 120-byte TCP segment from 7.0.0.11
MSDP(0): Append 120 bytes to 0-byte msg 3917 from 7.0.0.11, qs 1
MSDP(0): 7.0.0.11: Received 120-byte msg 3917 from peer
MSDP(0): 7.0.0.11: SA TLV, len: 120, ec: 1, RP: 9.0.0.10, with data
MSDP(0): 7.0.0.11: Peer RPF check bypassed, peer 7.0.0.11 in mesh-group MG9
```

```
R9#show ip msdp sa-cache 225.0.1.3
MSDP Source-Active Cache - 1 entries for 225.0.1.3
(9.2.5.2, 225.0.1.3), RP 9.0.0.10, MBGP/AS 9, 00:03:14/00:05:35, Peer
9.0.0.10
```

We will perform the same verification on XRv2, which should behave identically. It receives and accepts for SAs, but installs the first one received into the SA cache. This came from CSR10 which is indicated by the advertising peer address in the SA cache.

```
! XRv2
```

```
debug msdp tlv
debug msdp rpf
```

```
msdp[1045]: [1] default 9.0.0.10 tlv: ReadQ: 20 bytes in buffer
msdp[1045]: [1] default 9.0.0.10 tlv: ReadQ: TLV parsed length = 20, type = 1
msdp[1045]: [1] default 9.0.0.10 tlv: 9.0.0.10: recvd <SourceActive,20>
msdp[1045]: [1] default 9.0.0.10 tlv: RP 9.0.0.10 has 1 entries
msdp[1045]: [1] default 9.0.0.10 tlv: -- (9.2.5.2/32 225.0.1.3)
msdp[1045]: [1] default 9.0.0.10 tlv:
msdp[1045]: [1] 9.0.0.10 RPF: Peer 9.0.0.10 is in mesh group. Bypassing RPF
checks
msdp[1045]: [1] 9.0.0.10 RPF: Accepting SA from peer 9.0.0.10 for RP 9.0.0.10
due to Matched a mesh group member
```

```
msdp[1045]: [1] default 7.0.0.11 tlv: ReadQ: 20 bytes in buffer
msdp[1045]: [1] default 7.0.0.11 tlv: ReadQ: TLV parsed length = 20, type = 1
msdp[1045]: [1] default 7.0.0.11 tlv: 7.0.0.11: recvd <SourceActive,20>
msdp[1045]: [1] default 7.0.0.11 tlv: RP 9.0.0.10 has 1 entries
msdp[1045]: [1] default 7.0.0.11 tlv: -- (9.2.5.2/32 225.0.1.3)
msdp[1045]: [1] default 7.0.0.11 tlv:
msdp[1045]: [1] 9.0.0.10 RPF: Peer 7.0.0.11 is in mesh group. Bypassing RPF
checks
msdp[1045]: [1] 9.0.0.10 RPF: Accepting SA from peer 7.0.0.11 for RP 9.0.0.10
due to Matched a mesh group member
```

```
RP/0/0/CPU0:XRv2#show msdp sa-cache 225.0.1.3
```

```
MSDP Flags:
```

```
E - set MRIB E flag , L - domain local source is active,
EA - externally active source, PI - PIM is interested in the group,
DE - SAs have been denied. Timers age/expiration,
```

```
Cache Entry:
```

```
(9.2.5.2, 225.0.1.3), RP 9.0.0.10, MBGP/AS 9, 00:06:50/00:02:18
  Learned from peer 9.0.0.10, RPF peer 9.0.0.10
  SAs recvd 16, Encapsulated data received: 200
  grp flags: PI, src flags: E, EA, PI
```

As further proof, we can check CSR9 to ensure it is not advertising the SA or (9.2.5.2, 225.0.1.3) towards XRv1 (7.0.0.11). Being in the same mesh group, this should never happen. It is, however, advertising its locally originated SA which is a result of CSR4 sending traffic into the network. This is the desired result, which effectively allows CSR9 to be a “non-transit” MSDP node.

```
R9#show ip msdp peer 7.0.0.11 advertised-SAs
```

```
MSDP SA advertised to peer 7.0.0.11 (?) from mroute table
```

```
225.0.1.3      11.4.13.4 (?)
```

```
MSDP SA advertised to peer 7.0.0.11 (?) from SA cache
[no output]
```


There is a third way to bypass RPF in MSDP as well; this is the default-peer feature. This is valuable for environments where SAs should be accepted from a peer no matter what. You can specify multiple default-peers in a router, but they are processed top-down assuming the peers are up and a prefix-list is not used. We can optionally apply a prefix list to each default peer to specify which sources apply to which peer. For example, on CSR9 we could configure CSR10 and XRv1 as default peers given the sources in each of their ASes. Although this will remain in the configuration for demonstration purposes, this feature only makes sense when MSDP and BGP are not deployed together. This can be used for stub ASes that may not be running BGP, or perhaps a multi-homed customer to two ISPs with two static default routes. The current network does not fit this design without significant rework, so we don't test this further.

```
! CSR9
ip msdp default-peer 9.0.0.10 prefix-list PL_AS9
ip msdp default-peer 7.0.0.11 prefix-list PL_AS7
ip prefix-list PL_AS7 seq 5 permit 7.0.0.0/8 le 32
ip prefix-list PL_AS9 seq 5 permit 9.0.0.0/8 le 32
```

XR is less flexible than XE with respect to this feature, so I show a snippet below but do not commit it to the configuration. XR only allows for one default peer (good in a single-homed PE-CE environment without BGP) and thus no prefix-list support is needed. The snippet shows XRv1 as an example.

```
! XRv2 (example only)
router msdp
  default-peer 7.0.0.11
```

There are other minor options used for fine-tuning SA advertising. One of them is setting a basic SA-limit, which is loosely analogous to OSPF's "max-lsa" command. Only a certain amount of SAs can be received from a peer, regardless of what (S,G) pairs they represent, and this is a security mechanism to prevent DoS attacks against the MSDP control-plane. The syntax differs slightly between XE and XR, and we configure this between CSR10 and XRv1 for demonstration with a limit of 500 SAs. XE clearly shows this new limit in the peer output but XR does not.

```
! XRv1
router msdp
  peer 10.10.11.10
    maximum external-sa 500
```

```
! CSR10
ip msdp sa-limit 7.0.0.11 500
```

```
R10#show ip msdp peer 7.0.0.11 | include limit
  SAs learned from this peer: 0, SAs limit: 500
```

We can also filter SAs based on sources, groups, or both together. On CSR9 and XRv1, we configure them to reject sending SAs sourced from their own major networks to any administratively-scoped

addresses towards CSR9 and XRv2. In a real deployment, a feature like this would make sense on all external peers, but I demonstrate it towards CSR9 and XRv2. The filters on CSR9 use a combination of route-maps and ACLs while XRv1 only uses ACLs. XR RPLs are not supported for SA-filtering in MSDP.

```
! CSR10
ip access-list extended ACL_SA_FILTER
  deny ip 9.0.0.0 0.255.255.255 239.0.0.0 0.255.255.255
  permit ip any any

route-map RM_FILTER_SA permit 10
  match ip address ACL_SA_FILTER

ip msdp sa-filter in 8.0.0.12 list ACL_SA_FILTER
ip msdp sa-filter out 8.0.0.12 list ACL_SA_FILTER
ip msdp sa-filter in 11.0.0.9 route-map RM_FILTER_SA
ip msdp sa-filter out 11.0.0.9 route-map RM_FILTER_SA

! XRv1
ipv4 access-list ACL_SA_FILTER
  10 deny ipv4 9.0.0.0 0.255.255.255 239.0.0.0 0.255.255.255
  20 permit ipv4 any any

router msdp
  peer 11.0.0.9
    sa-filter in list ACL_SA_FILTER
    sa-filter out list ACL_SA_FILTER

  peer 8.0.0.12
    sa-filter in list ACL_SA_FILTER
    sa-filter out list ACL_SA_FILTER
```

We quickly verify that the configurations were applied successfully on both routers for both peers. On CSR10, the output changes slightly based on whether a route-map or ACL is used, since XE gives us the option.

```
R10#show ip msdp peer 8.0.0.12 | section Filtering
SA Filtering:
  Input (S,G) filter: ACL_SA_FILTER, route-map: none
  Input RP filter: none, route-map: none
  Output (S,G) filter: ACL_SA_FILTER, route-map: none
  Output RP filter: none, route-map: none

R10#show ip msdp peer 11.0.0.9 | section Filtering
SA Filtering:
  Input (S,G) filter: none, route-map: RM_FILTER_SA
  Input RP filter: none, route-map: none
  Output (S,G) filter: none, route-map: RM_FILTER_SA
  Output RP filter: none, route-map: none
```

```
RP/0/0/CPU0:XRv1#show msdp peer 11.0.0.9 | begin Filtering
SA Filtering:
```

```
Input (S,G) filter: ACL_SA_FILTER
```

```
Input RP filter: none
```

```
Output (S,G) filter: ACL_SA_FILTER
```

```
Output RP filter: none
```

```
RP/0/0/CPU0:XRv1#show msdp peer 8.0.0.12 | begin Filtering
SA Filtering:
```

```
Input (S,G) filter: ACL_SA_FILTER
```

```
Input RP filter: none
```

```
Output (S,G) filter: ACL_SA_FILTER
```

```
Output RP filter: none
```

To demonstrate this in action, I will source traffic from CSR2 to group 239.1.1.1, which will be (9.2.5.2, 239.1.1.1). This will match the outgoing filter on CSR10 and the SA will not be advertised to XRv2 or CSR9 at all, but XRv1 will receive it. XRv1 is likewise not allowed to send it to XRv2 or CSR9, so they shouldn't have the SA via any means. We enable debugging on CSR10 and XRv1 to confirm this. Of note, it seems like the XE route-map doesn't work since the message was filtered to XRv2 (ACL) but not to CSR9 (route-map).

```
R2#ping 239.1.1.1 repeat 1000000 timeout 1
```

```
! CSR10
```

```
MSDP(0): 7.0.0.11: Building SA message from SA cache
```

```
MSDP(0): start_index = 0, sa_cache_index = 0, Qlen = 0
```

```
MSDP(0): Sent entire sa-cache, sa_cache_index = 0, Qlen = 0
```

```
MSDP(0): Received 3-byte TCP segment from 7.0.0.11
```

```
MSDP(0): Append 3 bytes to 0-byte msg 9190 from 7.0.0.11, qs 1
```

```
MSDP(0): 7.0.0.11: Received 3-byte msg 9190 from peer
```

```
MSDP(0): 7.0.0.11: Keepalive TLV
```

```
MSDP(0): 7.0.0.11: Sending Keepalive message to peer
```

```
MSDP(0): 8.0.0.12: Filtered encapsulated message to peer
```

```
MSDP(0): 11.0.0.9: Send 100-byte SA encapsulated data for (9.2.5.2, 239.1.1.1), RP 9.0.0.10
```

```
MSDP(0): 7.0.0.11: Send 100-byte SA encapsulated data for (9.2.5.2, 239.1.1.1), RP 9.0.0.10
```

The XR debug isn't nearly as valuable since it doesn't show the filtering in real-time, but we do see the SA arriving from 9.0.0.10.

```
! XRv1
```

```
msdp[1045]: [1] default 10.10.11.10 tlv: ReadQ: 120 bytes in buffer
```

```
msdp[1045]: [1] default 10.10.11.10 tlv: ReadQ: TLV parsed length = 120, type = 1
```

```
msdp[1045]: [1] default 10.10.11.10 tlv: 10.10.11.10: rcvd
<SourceActive,120>
msdp[1045]: [1] default 10.10.11.10 tlv: RP 9.0.0.10 has 1 entries
msdp[1045]: [1] default 10.10.11.10 tlv: -- (9.2.5.2/32 239.1.1.1)
msdp[1045]: [1] default 10.10.11.10 tlv: -- 100 bytes of encap'd data
remain
```

We confirm the error by checking CSR10 to see if it actually advertised SAs. We also check XRv2 and CSR9 for further confirmation. We can conclude the route-map mechanism isn't working in XE in version 3.13.2S. I would recommend to use the ACL only, primarily because it always works and because it is consistent with XR.

```
R10#show ip msdp peer 8.0.0.12 advertised-SAs | include 239
[no output]
```

```
R10#show ip msdp peer 11.0.0.9 advertised-SAs | include 239
239.1.1.1          9.2.5.2 (?)
```

```
RP/0/0/CPU0:XRv2#show msdp sa-cache 239.1.1.1
[no output]
```

```
R9#show ip msdp sa-cache 239.1.1.1
MSDP Source-Active Cache - 1 entries for 239.1.1.1
(9.2.5.2, 239.1.1.1), RP 9.0.0.10, MBGP/AS 9, 00:03:49/00:05:14, Peer
9.0.0.10
```

Continuing on, XRv1 correctly processes this SA from CSR10 and adds it to the SA cache. XRv1 is allowed to learn this SA as it was not filtered outbound on CSR10 or inbound on XRv1.

```
RP/0/0/CPU0:XRv1#show msdp sa-cache 239.1.1.1
MSDP Flags:
E - set MRIB E flag , L - domain local source is active,
EA - externally active source, PI - PIM is interested in the group,
DE - SAs have been denied. Timers age/expiration,
Cache Entry:
(9.2.5.2, 239.1.1.1), RP 9.0.0.10, MBGP/AS 9, 00:01:16/00:01:54
  Learned from peer 10.10.11.10, RPF peer 10.10.11.10
  SAs rcvd 2, Encapsulated data received: 100
  grp flags: none, src flags: EA
```

Both XE and XR show ACL hits which are revealed when the SAs are matched against a filter. Of course, on CSR10, these hits only increment when advertising SAs to XRv2 since the route-map applied to CSR9 is dysfunctional and never matches.

```
RP/0/0/CPU0:XRv1#show access-lists ACL_SA_FILTER
ipv4 access-list ACL_SA_FILTER
10 deny ipv4 9.0.0.0 0.255.255.255 239.0.0.0 0.255.255.255 (28 matches)
```

```
20 permit ipv4 any any (24 matches)
```

```
R10#show access-list ACL_SA_FILTER  
Extended IP access list ACL_SA_FILTER
```

```
10 deny ip 9.0.0.0 0.255.255.255 239.0.0.0 0.255.255.255 (4 matches)  
20 permit ip any any (43 matches)
```

TTL scoping is a simple mechanism to help confine multicast traffic within a certain AS. This becomes very significant in IPv6 since it is one of the main mechanisms for performing inter-AS ASM; there is no MSDP for IPv6. MSDP can also identify TTL-thresholds per peer which will cause it not to encapsulate data traffic into SA messages to peers if the TTL of the multicast data packet is less than or equal to the configured value. For example, XE has no default here, but XR has a default threshold value of 2. This makes sense because in terms of inter-AS multicast, it's highly likely that the source and destination and less than 2 hops away. Building an inter-AS multicast tree would be useless since the data-plane traffic would expire before reaching the receivers. We quickly verify the defaults before adjusting the values.

```
R10#show ip msdp peer 7.0.0.11 | include ttl  
Peer ttl threshold: 0
```

```
RP/0/0/CPU0:XRv1#show msdp peer 10.10.11.10 | include ttl  
Peer ttl threshold: 2
```

On XRv1 and CSR10, on the peering they share, I adjust the TTL-threshold value to 8. This means that multicast packets with TTL <= 8 cannot be encapsulated into SA messages and sent to the peer in question. A higher threshold implies a more strict filtering policy. After configuring the feature, we quickly verify the change, although testing this is difficult given our inability to adjust ping TTLs.

```
! CSR10  
ip msdp ttl-threshold 7.0.0.11 8
```

```
! XRv1  
router msdp  
peer 10.10.11.10  
ttl-threshold 8
```

```
R10#show ip msdp peer 7.0.0.11 | include ttl  
Peer ttl threshold: 8
```

```
RP/0/0/CPU0:XRv1#show msdp peer 10.10.11.10 | include ttl  
Peer ttl threshold: 8
```

MSDP also has a concept of redistribution. This is totally unrelated to route redistribution used in unicast routing protocols. The word “redistribution” in this context relates to the redistribution of sources within the MSDP topology. In other words, it is a first-stage filter that is used by an MSDP speaker to determine for which (S,G) an SA is created at all. For example, local multicast traffic

(239.0.0.0/8 groups, RFC1918 sources, etc) shouldn't even have an SA built at all in many cases; there is little reason to announce this traffic outside of an AS boundary. MSDP speakers that aren't RPs have no reason to originate SAs at all, only forward them. MSDP redistribution can achieve both goals. This feature is only supported in XE, and we will configure it on CSR9. CSR9 has no other filtering configured, so when CSR4 sends traffic to group 239.1.1.2, CSR9 should not even create an SA message. The feature accepts a route-map, ACL, or AS-path ACL. For simplicity, I use an ordinary ACL below.

```
! CSR9
ip access-list extended ACL_MSDP_REDISTRIBUTE
deny ip 11.0.0.0 0.255.255.255 239.0.0.0 0.255.255.255
permit ip any any

ip msdp redistribute list ACL_MSDP_REDISTRIBUTE
```

To test it, I will use two test groups. One is a global group of 225.0.1.3, the group CSR4 is using currently. I also send traffic to 239.1.1.2; both of these groups are successfully registered with the RP. The 'A' flag means these are candidates for MSDP advertisement and the SAs will be redistributed unless we manually filter them. The reason 239.1.1.2 is pruned is because there are no receivers, but 225.0.1.3 actually has real receivers from earlier.

```
! CSR9
ping 225.0.1.3 repeat 1
ping 239.1.1.2 repeat 1
```

```
R9#show ip mroute 239.1.1.2 11.4.13.4 | begin \
(11.4.13.4, 239.1.1.2), 00:03:22/00:02:57, flags: PA
Incoming interface: GigabitEthernet2.593, RPF nbr 11.9.13.13
Outgoing interface list: Null
```

```
R9#show ip mroute 225.0.1.3 11.4.13.4 | begin \
(11.4.13.4, 225.0.1.3), 00:02:04/00:03:29, flags: A
Incoming interface: GigabitEthernet2.593, RPF nbr 11.9.13.13
Outgoing interface list:
GigabitEthernet2.594, Forward/Sparse, 00:02:04/00:03:25
```

Checking CSR9's advertised SAs to CSR10 as an example, we only see group 225.0.1.3 and not 239.1.1.2. Like the SA filter ACLs, we can also see matches for (S,G) entries that match a particular ACL entry. We cannot confirm this with MSDP debug on CSR9 because it displays no output. The filtering likely occurs before the MSDP debug processes see it. This concludes the MSDP section of this book.

```
R9#show ip msdp peer 9.0.0.10 advertised-SAs
MSDP SA advertised to peer 9.0.0.10 (?) from mroute table

225.0.1.3 11.4.13.4 (?)
```

```
R9#show access-lists ACL_MSDP_REDISTRIBUTE
```

Extended IP access list ACL_MSDP_REDISTRIBUTE

```
10 deny ip 11.0.0.0 0.255.255.255 239.0.0.0 0.255.255.255 (17 matches)
20 permit ip any any (14 matches)
```

As mentioned earlier, setting multicast TTL-thresholds on administrative boundaries can help scope multicast as well. It is cruder than ACL-based filtering and not often used with IPv4; XE no longer supports the command and it is hidden in the parser. XR does support the command, and I enable it on the link to CSR10 for demonstration purposes. Earlier we used a multicast boundary to filter AutoRP advertisements, so we don't demonstrate that again, but that is a preferred method for inter-AS filtering.

```
R10(config-subif)#ip multicast ttl 8
```

This command is no longer supported.

```
! XRv1
multicast-routing
 address-family ipv4
   interface GigabitEthernet0/0/0/0.501
     ttl-threshold 8
```

One of the simplest methods to do inter-AS multicast is to use SSM. This obviates the need for PIM RPs and MSDP entirely, relying only on RPF back to the sources directly. We quickly enable SSM for a custom range 238.0.0.0/8 on all multicast routers in the network. By default, XR uses 232.0.0.0/8 and XE doesn't have SSM enabled at all. We have tested SSM in other chapters using 232.0.0.0/8 but using a custom group is good for variety.

```
! All XE multicast routers
ip pim ssm range ACL_SSM
ip access-list standard ACL_SSM
 permit 238.0.0.0 0.255.255.255
```

```
! All XR multicast routers
ipv4 access-list ACL_SSM
 10 permit ipv4 any 238.0.0.0/8
```

```
multicast-routing
 address-family ipv4
   ssm range ACL_SSM
```

For SSM to work, the PIM routers connecting to receivers must support IGMPv3, which is not the XE default. We enable it on CSR5, CSR7, and CSR8 under their appropriate LAN interfaces. The receivers could potentially be CSR1 through CSR4, so they must also support IGMPv3. Only CSR5 is shown for brevity.

```
! CSR5
```

```
interface GigabitEthernet2.525
 ip igmp version 3
```

To test this, CSR3 will request traffic for 238.3.3.3 from 9.2.5.2 which resides in AS 9. We ensure that CSR8 receives the IGMPv3 membership report in INCLUDE mode, which “includes” specific sources, namely 9.2.5.2.

```
R8#show ip igmp groups 238.3.3.3 detail
```

```
Flags: L - Local, U - User, SG - Static Group, VG - Virtual Group,
       SS - Static Source, VS - Virtual Source,
       Ac - Group accounted towards access control limit
```

```
Interface:      GigabitEthernet2.538
```

```
Group:          238.3.3.3
```

```
Flags:          SSM
```

```
Uptime:         00:00:29
```

```
Group mode:     INCLUDE
```

```
Last reporter:  8.3.8.3
```

```
Group source list: (C - Cisco Src Report, U - URD, R - Remote, S - Static,
                   V - Virtual, M - SSM Mapping, L - Local,
                   Ac - Channel accounted towards access control limit)
```

Source Address	Uptime	v3 Exp	CSR Exp	Fwd	Flags
9.2.5.2	00:00:29	00:02:31	stopped	Yes	R

CSR8 creates (S,G) state for this SSM group. Interestingly, the RPF check to 9.2.5.2 is via multicast BGP because we advertised that LAN route into BGP for MSDP testing earlier. The ‘s’ flag implies SSM is in use and the ‘l’ flag implies that a host-specific join was received.

```
R8#show ip mroute ssm
```

```
(9.2.5.2, 238.3.3.3), 00:01:28/00:02:01, flags: sTI
```

```
Incoming interface: GigabitEthernet2.582, RPF nbr 8.8.12.12, Mbgp
```

```
Outgoing interface list:
```

```
GigabitEthernet2.538, Forward/Sparse, 00:01:28/00:02:01
```

```
R8#show ip rpf 9.2.5.2
```

```
RPF information for ? (9.2.5.2)
```

```
RPF interface: GigabitEthernet2.582
```

```
RPF neighbor: ? (8.8.12.12)
```

```
RPF route/mask: 9.2.5.0/24
```

```
RPF type: multicast (bgp 8)
```

```
Doing distance-preferred lookups across tables
```

```
RPF topology: ipv4 multicast base, originated from ipv4 unicast base
```

We will quickly trace the (S,G) tree back to the source using a variety of show commands. The SPT is built as such: CSR8 > XRv2 > CSR6 > CSR10 > CSR5. CSR5 is the root of the SPT as the RPF neighbor is connected.


```
RP/0/0/CPU0:XRv2#show pim topology 238.3.3.3 | begin 238
(9.2.5.2,238.3.3.3)SPT SSM Up: 00:04:54
JP: Join(now) RPF: GigabitEthernet0/0/0/0.562,10.6.12.6 Flags:
GigabitEthernet0/0/0/0.582 00:04:54 fwd Join(00:03:29)
```

```
R6#show ip mroute ssm
(9.2.5.2, 238.3.3.3), 00:05:33/00:02:53, flags: sT
Incoming interface: GigabitEthernet2.560, RPF nbr 9.6.10.10
Outgoing interface list:
GigabitEthernet2.562, Forward/Sparse, 00:05:33/00:02:53
```

```
R10#show ip mroute ssm
(9.2.5.2, 238.3.3.3), 00:08:20/00:03:02, flags: sT
Incoming interface: GigabitEthernet2.550, RPF nbr 9.5.10.5
Outgoing interface list:
GigabitEthernet2.560, Forward/Sparse, 00:08:20/00:03:02
```

```
R5#show ip mroute ssm
(9.2.5.2, 238.3.3.3), 00:09:02/00:03:20, flags: sT
Incoming interface: GigabitEthernet2.525, RPF nbr 0.0.0.0
Outgoing interface list:
GigabitEthernet2.550, Forward/Sparse, 00:09:02/00:03:20
```

Checking the packet counters on CSR8 for brevity, we can see traffic is successfully being delivered to CSR3. There was no additional complexity with PIM RPs or MSDP in this design, and once the SPT is built, there is no additional signaling required once the source starts sending traffic.

```
R2#ping 238.3.3.3 repeat 10000 timeout 1
```

```
R8#show ip mroute 238.3.3.3 9.2.5.2 count | begin ^Group
Group: 238.3.3.3, Source count: 1, Packets forwarded: 18, Packets received:
18
Source: 9.2.5.2/32, Forwarding: 18/1/118/0, Other: 18/0/0
```

```
R8#show ip mroute 238.3.3.3 9.2.5.2 count | begin ^Group
Group: 238.3.3.3, Source count: 1, Packets forwarded: 28, Packets received:
28
Source: 9.2.5.2/32, Forwarding: 28/1/118/0, Other: 28/0/0
```

Next, we will examine IPv6 inter-AS multicast routing. Because there is no concept of MSDP, we can achieve intelligent scoping in one of three ways. I list them in order of my personal preference, and probably from most straightforward to most difficult.

1. Use SSM as we did for IPv4. The signaling logic is identical for IPv4 SSM and IPv6 SSM.
2. Use embedded RP for ASM, which does not apply in IPv4. The IPv6 data packet embeds the IPv6 RP information and provided there is a common RP for this group, the proper trees can be built.

3. Use the scope bits inside the IPv6 group address for ASM. These were discussed in the IPv6 general architecture section and allow the administrator to define arbitrary boundaries. We could also use these to scope SSM traffic as they are independent from the multicast transport type.

First, we will use IPv6 SSM similar to how we did for IPv4. We will use the default IPv6 SSM range of FF33::/32 since XE does not appear capable of specifying a custom range. Technically, FF3x::/32 where 3 <= x <= F are valid IPv6 ranges, but we will use FF33::/32 for simplicity (scope bits of link-local or node-local don't make sense for SSM). CSR4 will be a receiver for group FF33::4 and CSR1 will be the source (2007:7:1:7::1). Just like IPv4 SSM requires IGMPv3 at the receiver, IPv6 SSM requires MLDv2 at the receiver. Both XE and XR have this enabled by default, so we don't have to modify this. CSR4 issues the INCLUDE mode membership report to XRv3 who processes it locally.

```
! CSR4
interface GigabitEthernet2.543
  ipv6 mld join-group FF33::4 2007:7:1:7::1

RP/0/0/CPU0:XRv3#show mld groups ff33::4 gig0/0/0/0.543 detail
Interface:      GigabitEthernet0/0/0/0.543
Group:          ff33::4
Uptime:         00:01:52
Router mode:    INCLUDE
Host mode:      INCLUDE
Last reporter:  fe80::4
Group source list:
  Source Address      Uptime    Expires    Fwd  Flags
  2007:7:1:7::1      00:01:52  00:03:50  Yes  Remote 4
```

XRv3 now creates (S,G) create within the IPv6 MRIB and PIM topology. The SPT is built as follows: XRv3 > CSR9 > XRv4 > CSR7. Just like with IPv4, the little 's' flag on XE indicates this is an SSM group, where XR actually uses the string "SSM".

```
RP/0/0/CPU0:XRv3#show pim ipv6 topology ff33::4 | begin ff33
(2007:7:1:7::1, ff33::4) SPT
SSM Up: 00:04:27 JP: Join(00:00:21) Flags:
RPF: GigabitEthernet0/0/0/0.593, fe80::9
  GigabitEthernet0/0/0/0.543 00:04:27 fwd LI LH

R9#show ipv6 mroute ff33::4 | begin \
(2007:7:1:7::1, FF33::4), 00:06:10/00:02:10, flags: sT
Incoming interface: GigabitEthernet2.594
RPF nbr: FE80::14
Immediate Outgoing interface list:
  GigabitEthernet2.593, Forward, 00:06:10/00:02:10

RP/0/0/CPU0:XRv4#show pim ipv6 topology ff33::4 | begin ff33
```

```

(2007:7:1:7::1, ff33::4) SPT
SSM Up: 00:06:14 JP: Join(00:00:16) Flags:
RPF: GigabitEthernet0/0/0/0.574, fe80::7
    GigabitEthernet0/0/0/0.594 00:06:14 fwd Join(00:03:21)

R7#show ipv6 mroute ff33::4 | begin \
(2007:7:1:7::1, FF33::4), 00:07:05/00:02:57, flags: sT
    Incoming interface: GigabitEthernet2.517
    RPF nbr: 2007:7:1:7::1
    Immediate Outgoing interface list:
        GigabitEthernet2.574, Forward, 00:07:05/00:02:57

```

We begin pinging on CSR1 to test reachability. We can check the counters on XRv3 to ensure traffic is being delivered to CSR4. We can see packets in/packets out on XRv3, and the numbers are equal which means there is only one OIL interface (no replication). I issue the command a few seconds apart so we can see the counters increment. The 'A' flag on Gig0/0/0/0.593 shows that packets were "accepted" from this interface (valid RPF) and the 'EG' flag on Gig0/0/0/0.543 shows that packets were "egressing" towards CSR4.

```

R1#ping ff33::4 repeat 10000 timeout 1
Output Interface: GigabitEthernet2.517

```

```

RP/0/0/CPU0:XRv3#show mfib ipv6 route ff33::4 | begin ff33
(2007:7:1:7::1, ff33::4)
Flags:
Up: 00:09:31
Last Used: 00:00:00
SW Forwarding Counts: 11/11/1100
SW Replication Counts: 11/11/1100
SW Failure Counts: 0/0/0/0/0
GigabitEthernet0/0/0/0.543 Flags: NS EG, Up:00:09:31
GigabitEthernet0/0/0/0.593 Flags: A, Up:00:09:31

```

```

RP/0/0/CPU0:XRv3#show mfib ipv6 route ff33::4 | begin ff33
(2007:7:1:7::1, ff33::4)
Flags:
Up: 00:09:33
Last Used: 00:00:00
SW Forwarding Counts: 13/13/1300
SW Replication Counts: 13/13/1300
SW Failure Counts: 0/0/0/0/0
GigabitEthernet0/0/0/0.543 Flags: NS EG, Up:00:09:33
GigabitEthernet0/0/0/0.593 Flags: A, Up:00:09:33

```

Next, we will test embedded RP in an inter-AS environment. We first need to identify the RPs and statically configure them (and only them) as RPs. Because this information is contained within the IPv6 data traffic, we don't necessarily need to be fancy with per-group RP assignments. The RP will look like a

static RP on these nodes (because it is). We begin with CSR10 and CSR9, then verify the RP configuration afterwards.

```
! CSR10
ipv6 pim rp-address 2009::a
```

```
! CSR9
ipv6 pim rp-address 200b::9
```

```
R10#show ipv6 pim range-list 2009::a
Static SM RP: 2009::A Exp: never Learnt from : ::
FF00::/8 Up: 00:00:26
```

```
R9#show ipv6 pim range-list 200b::9
Static SM RP: 200B::9 Exp: never Learnt from : ::
FF00::/8 Up: 00:00:12
```

We configure the same feature on XRv1 and XRv2. The configuration is similar and we verify the RP mappings before continuing. The difference is that in XR, you cannot simply specify a static RP; you must use the specified embedded-RP command. Additionally, you must specify a group-list, but I permit all globally-scoped groups for each RP. The ACL is commonly used for scoping since we can permit/deny certain ranges within a given scope. The breakdown of the group addresses is discussed next. At this point, every AS has an RP.

```
! XRv1
ipv6 access-list ACL_EMBEDDED_RP
10 permit ipv6 any ff7e:b40:2007::/96
```

```
router pim
address-family ipv6
embedded-rp 2007::b ACL_EMBEDDED_RP
```

```
! XRv2
ipv6 access-list ACL_EMBEDDED_RP
10 permit ipv6 any ff7e:c40:2008::/96
```

```
router pim
address-family ipv6
embedded-rp 2008::c ACL_EMBEDDED_RP
```

```
RP/0/0/CPU0:XRv1#show pim ipv6 group-map ff7e:b40:2007::
IP PIM Group Mapping Table
(* indicates group mappings being used)
(+ indicates BSR group mappings active in MRIB)
```

Group Range	Proto	Client	Groups
ff7e:b40:2007::/96*	SM	embd-cfg	0

```
RP: 2007::b
RPF: De6tunnel2, 2007::b (us)
```

```
RP/0/0/CPU0:XRv2#show pim ipv6 group-map ff7e:c40:2008::
IP PIM Group Mapping Table
(* indicates group mappings being used)
(+ indicates BSR group mappings active in MRIB)
```

Group Range	Proto	Client	Groups
ff7e:c40:2008::/96*	SM	embd-cfg	0
RP: 2008::c			
RPF: De6tunnel2, 2008::c (us)			

Before continuing, we will specify the appropriate embedded RP group-ranges supported by each embedded RP. For simplicity I will use global scope (0xE) for now. The RP host address is limited to 4 bits, which is why I used hexadecimal characters for the router loopback low-order bits rather than decimal numbers. This is relevant particularly on CSR10 and the XR routers. In the examples below, yellow highlights the scope, green highlights the RP host ID, pink highlights the prefix length (0x40 = 64), grey highlights the prefix, and cyan highlights the available groups.

```
CSR9:  FF7E:0940:200B:0:0:0:0:0/96
CSR10: FF7E:0A40:2009:0:0:0:0:0/96
XRv1:  FF7E:0B40:2007:0:0:0:0:0/96
XRv2:  FF7E:0C40:2008:0:0:0:0:0/96
```

A very quick way to verify this is to ping an IPv6 multicast group in each of these ranges from any of our test clients provided embedded RP is supported (it is enabled by default on XE and XR). The PIM DR that registers these sources will be smart enough to know the RP is embedded based on the flags in the group address header (these flags were discussed in the IPv6 architecture section). I will conduct this test on CSR2 using CSR5 as the PIM DR. We enable debugging on CSR5 to watch it create range-list (RP mapping) entries for the proper RPs, then confirm it creates (S,G) state for each entry.

```
! CSR9 is RP
R2#ping FF7E:0940:200B:0:0:0:0:0
Output Interface: GigabitEthernet2.525
```

```
! CSR10 is RP
R2#ping FF7E:0A40:2009:0:0:0:0:0
Output Interface: GigabitEthernet2.525
```

```
! XRv1 is RP
R2#ping FF7E:0B40:2007:0:0:0:0:0
Output Interface: GigabitEthernet2.525
```

```
! XRv2 is RP
R2#ping FF7E:0C40:2008:0:0:0:0:0
```

Output Interface: GigabitEthernet2.525

```
! CSR5
IPv6 PIM: (2009:9:2:5::2,FF7E:940:200B::/128) MRIB update (t=1)
IPv6 PIM: Create range list 200B::9 Sparse
IPv6 PIM: Create range FF7E:940:200B::/96
IPv6 PIM: Adding monitor for 200B::9
IPv6 PIM: RPF change for root 200B::9: nbr FE80::10, GigabitEthernet2.550 via
the rib

IPv6 PIM: (2009:9:2:5::2,FF7E:A40:2009::/128) MRIB update (t=1)
IPv6 PIM: Create range list 2009::A Sparse
IPv6 PIM: Create range FF7E:A40:2009::/96
IPv6 PIM: Adding monitor for 2009::A
IPv6 PIM: RPF change for root 2009::A: nbr FE80::10, GigabitEthernet2.550 via
the rib

IPv6 PIM: (2009:9:2:5::2,FF7E:B40:2007::/128) MRIB update (t=1)
IPv6 PIM: Create range list 2007::B Sparse
IPv6 PIM: Create range FF7E:B40:2007::/96
IPv6 PIM: Adding monitor for 2007::B
IPv6 PIM: RPF change for root 2007::B: nbr FE80::10, GigabitEthernet2.550 via
the rib

IPv6 PIM: (2009:9:2:5::2,FF7E:C40:2008::/128) MRIB update (t=1)
IPv6 PIM: Create range list 2008::C Sparse
IPv6 PIM: Create range FF7E:C40:2008::/96
IPv6 PIM: Adding monitor for 2008::C
IPv6 PIM: RPF change for root 2008::C: nbr FE80::10, GigabitEthernet2.550 via
the rib
```

We can confirm the MRIB to ensure the (S,G) state exists for each one. Since CSR5 should be registering each (S,G) to each remote RP, the entries should exist for each (S,G) that CSR2 used.

```
R5#show ipv6 mroute | begin FF7
(2009:9:2:5::2, FF7E:940:200B::), 00:01:40/00:01:49, flags: SFT
  Incoming interface: GigabitEthernet2.525
  RPF nbr: 2009:9:2:5::2, Registering
  Immediate Outgoing interface list:
    Tunnel0, Forward, 00:01:40/never

(2009:9:2:5::2, FF7E:A40:2009::), 00:01:11/00:02:18, flags: SFT
  Incoming interface: GigabitEthernet2.525
  RPF nbr: 2009:9:2:5::2, Registering
  Immediate Outgoing interface list:
    Tunnel0, Forward, 00:01:11/never

(2009:9:2:5::2, FF7E:B40:2007::), 00:00:55/00:02:34, flags: SFT
  Incoming interface: GigabitEthernet2.525
```

```
RPF nbr: 2009:9:2:5::2, Registering
Immediate Outgoing interface list:
  Tunnel0, Forward, 00:00:55/never
```

```
(2009:9:2:5::2, FF7E:C40:2008::), 00:00:40/00:02:49, flags: SFT
Incoming interface: GigabitEthernet2.525
RPF nbr: 2009:9:2:5::2, Registering
Immediate Outgoing interface list:
  Tunnel0, Forward, 00:00:40/never
```

Last, we can check the range-list mappings beginning with the first embedded RP. We see all four entries in the table, which proves our embedded RP group construction from earlier was correct.

```
R5#show ipv6 pim range-list | begin ^Embed
Embedded SM RP: 2007::B Exp: never Learnt from : ::
  FF7E:B40:2007::/96 Up: 00:01:38
Embedded SM RP: 2008::C Exp: never Learnt from : ::
  FF7E:C40:2008::/96 Up: 00:01:23
Embedded SM RP: 2009::A Exp: never Learnt from : ::
  FF7E:A40:2009::/96 Up: 00:01:54
Embedded SM RP: 200B::9 Exp: never Learnt from : ::
  FF7E:940:200B::/96 Up: 00:02:23
```

It appears that register messages are never being sent. None of the remote RPs show any indication of receiving a register message. To prove it, below I show two PIM tunnels. One is the register tunnel for the IPv4 RP within AS 7 and the other is for the embedded RP in AS 8. This is the same behavior seen in the MVPN section with embedded RP as well.

```
R5#show ip pim tunnel
Tunnel1
  Type      : PIM Encap
  RP        : 9.0.0.10
  Source    : 9.5.10.5
  State     : UP
  Last event : Created (1d02h)
```

```
R5#show ipv6 pim tunnel
Tunnel0*
  Type : PIM Encap
  RP   : Embedded RP Tunnel
  Source: 2009::5
```

Despite the embedded RP destination being known, the tunnel destination is unspecified. One may think this is just a cosmetic issue because the RP isn't learned ahead of time and the tunnel is built dynamically. Having testing embedded RP before, this is actually a normal case.

```
R5#show derived-config interface tunnel1
```

```

interface Tunnel1
  description Pim Register Tunnel (Encap) for RP 9.0.0.10
  ip unnumbered GigabitEthernet2.550
  tunnel source GigabitEthernet2.550
  tunnel destination 9.0.0.10
  tunnel tos 192

```

```

R5#show derived-config interface tunnel0
interface Tunnel0
  description Pim Register Tunnel (Encap) for Embedded RP
  no ip address
  ipv6 unnumbered Loopback0
  ipv6 enable
  tunnel source Loopback0
  tunnel destination ::
  tunnel tos 224
  tunnel ttl 65

```

Furthermore, tunnel0 shows no outbound packets while tunnel1 does. Tunnel0 doesn't even show output drops or any sign of activity. Debugging CEF drops and IPv6 packets also reveals no indication of PIMv6 register messages being sent. Unfortunately, this is also the case in a working embedded RP design, making the problem difficult to troubleshoot.

```

R5#show interfaces tunnel1 | include output
Last input never, output never, output hang never
Input queue: 0/375/0/0 (size/max/drops/flushes); Total output drops: 0
5 minute output rate 0 bits/sec, 0 packets/sec
  10 packets output, 1280 bytes, 0 underruns
  0 output errors, 0 collisions, 0 interface resets
  0 output buffer failures, 0 output buffers swapped out

```

```

R5#show interfaces tunnel0 | include output
Last input never, output never, output hang never
Input queue: 0/375/0/0 (size/max/drops/flushes); Total output drops: 0
5 minute output rate 0 bits/sec, 0 packets/sec
  0 packets output, 0 bytes, 0 underruns
  0 output errors, 0 collisions, 0 interface resets
  0 output buffer failures, 0 output buffers swapped out

```

The feature actually works if traffic is sourced from CSR5's loopback and sent out of the interface as well. I assume that because this local traffic is processed-switched, the registration process happens successfully. I quickly test a group for each of the 4 embedded RPs to make sure the registers are working now. All of the OILs are null since there are no receivers in the network for any of these groups.

```

R5#ping FF7E:A40:2009:: source loopback0
Output Interface: loopback0

```



```
R10#show ipv6 mroute FF7E:A40:2009:: | begin \  
(2009::5, FF7E:A40:2009::), 00:00:25/00:03:04, flags: SP  
    Incoming interface: GigabitEthernet2.550  
    RPF nbr: FE80::5  
    Outgoing interface list: Null
```

```
R5#ping FF7E:940:200B:: source loopback0  
Output Interface: loopback0
```

```
R9#show ipv6 mroute FF7E:940:200B:: | begin \  
(2009::5, FF7E:940:200B::), 00:00:12/00:03:16, flags: SP  
    Incoming interface: GigabitEthernet2.569  
    RPF nbr: FE80::6  
    Outgoing interface list: Null
```

For additional verification, IPv6 PIM debugs on CSR9 (one of the RPs) shows that a register was received. We never saw this before when trying to make CSR5 register sources on its LAN interface to CSR2.

```
! CSR9  
IPv6 PIM: (2009::5,FF7E:940:200B::) Received Register from 2009::5  
IPv6 PIM: (2009::5,FF7E:940:200B::) Create entry  
IPv6 PIM: (2009::5,FF7E:940:200B::) RPF changed from ::/- to  
FE80::6/GigabitEthernet2.569
```

When testing this with XRv1 and XRv2, they receive the registers, but RPF appears to be failing despite both routers have valid unicast BGP routes.

```
R5#ping FF7E:B40:2007:: source loopback0  
Output Interface: loopback0
```

```
RP/0/0/CPU0:XRv1#show pim ipv6 topology ff7e:b40:2007:: | begin 2009  
(2009::5,ff7e:b40:2007::)  
SM Up: 00:00:14 JP: Null(never) Flags: KAT(00:03:15) RA RR (00:04:20)  
RPF: Null,::  
    No interfaces in immediate olist
```

```
R5#ping FF7E:C40:2008:: source loopback0  
Output Interface: loopback0
```

```
RP/0/0/CPU0:XRv2#show pim ipv6 topology ff7e:c40:2008:: | begin 2009  
(2009::5,ff7e:c40:2008::)  
SM Up: 00:00:16 JP: Null(never) Flags: KAT(00:03:15) RA RR (00:04:18)  
RPF: Null,::  
    No interfaces in immediate olist
```

```
RP/0/0/CPU0:XRv1#show route ipv6 2009::5  
Routing entry for 2009::5/128  
    Known via "bgp 7", distance 20, metric 0
```

```

Tag 9, type external
Routing Descriptor Blocks
  fe80::10, from fd00:10:10:11::10, via GigabitEthernet0/0/0/0.501, BGP
external
  Route metric is 0
  No advertising protos.

```

```

RP/0/0/CPU0:XRv2#show route ipv6 2009::5
Routing entry for 2009::5/128
  Known via "bgp 8", distance 20, metric 0
  Tag 9, type external
  Routing Descriptor Blocks
    fe80::6, from fd00:10:6:12::6, via GigabitEthernet0/0/0/0.562, BGP
external
  Route metric is 0
  No advertising protos.

```

This is because whenever XR is running IPv4/v6 multicast in BGP, the IPv4/v6 unicast routes are not allowed to satisfy RPF. This is stricter than XE where any route can satisfy RPF if the multicast BGP entries do not exist. This is part of the reason why the MSDP RPF rules were so difficult earlier; XR assumes that if BGP multicast is available it must be used; unicast BGP routes are ignored. We will quickly advertise CSR5's IPv6 loopback into IPv6 BGP multicast, and then start the pings again.

```

! CSR5
router bgp 9
 address-family ipv6 multicast
   network 2009::5/128

```

```

R5#ping FF7E:B40:2007:: source loopback0
Output Interface: loopback0

```

```

RP/0/0/CPU0:XRv1#show pim ipv6 topology ff7e:b40:2007:: | begin 2009
(2009::5,ff7e:b40:2007::)
SM Up: 00:00:22 JP: Null(never) Flags: KAT(00:03:09) RA RR (00:04:12)
RPF: GigabitEthernet0/0/0/0.501,fe80::10
  No interfaces in immediate olist

```

```

R5#ping FF7E:C40:2008:: source loopback0
Output Interface: loopback0

```

```

RP/0/0/CPU0:XRv2#show pim ipv6 topology FF7E:C40:2008:: | begin 2009
(2009::5,ff7e:c40:2008::)
SM Up: 00:00:05 JP: Null(never) Flags: KAT(00:03:27) RA RR (00:04:29)
RPF: GigabitEthernet0/0/0/0.562,fe80::6
  No interfaces in immediate olist

```

Despite the RPF next-hops being the same as the unicast routes, XR simply didn't accept the unicast BGP routes for RPF when multicast AFIs were available. A quick look at the PIM RPF information confirms the

valid entries. CSR9 and CSR10, also running BGP IPv6 multicast, accepted the RPF derived from the BGP IPv6 unicast route to 2009::5.

```
RP/0/0/CPU0:XRv1#show pim ipv6 rpf 2009::5
Table: IPv6-Multicast-default
2009::5/128 [20/0]
  via fe80::10, GigabitEthernet0/0/0/0.501
```

```
RP/0/0/CPU0:XRv2#show pim ipv6 rpf 2009::5
Table: IPv6-Multicast-default
2009::5/128 [20/0]
  via fe80::6, GigabitEthernet0/0/0/0.562
```

To actually test a real flow, I will join a group in AS8 from CSR3. This is an ASM join so no sources are included, as CSR8 shows.

```
! CSR3
interface GigabitEthernet2.538
  ipv6 mld join-group FF7E:C40:2008::3

R8#show ipv6 mld groups FF7E:C40:2008::3 detail
Interface:      GigabitEthernet2.538
Group:          FF7E:C40:2008::3
Uptime:         03:51:31
Router mode:    EXCLUDE (Expires: 00:03:23)
Host mode:      INCLUDE
Last reporter:  FE80::3
Source list is empty
```

CSR8 originates the (*,G) join towards the RP, which is 2008::C (XRv2). It gleans this from the group itself just like the first-hop routers did during the source registration process. XRv2 receives the join, sees itself as the RP, and becomes the root of the (*,G) tree. It is standing by to receive register messages on its decapsulation PIM tunnel.

```
R8#show ipv6 mroute ff7e:c40:2008::3 | begin \
(*, FF7E:C40:2008::3), 03:52:45/never, RP 2008::C, flags: SCJ
Incoming interface: GigabitEthernet2.582
RPF nbr: FE80::12
Immediate Outgoing interface list:
  GigabitEthernet2.538, Forward, 03:52:45/never
```

```
RP/0/0/CPU0:XRv2#show pim ipv6 topology ff7e:c40:2008::3 | begin ::3
(*,ff7e:c40:2008::3)
SM Up: 03:50:33 JP: Join(never) Flags:
RP: 2008::c*
RPF: Decaps6tunnel2,2008::c
  GigabitEthernet0/0/0/0.582 03:50:33 fwd Join(00:02:58)
```

We will send traffic from CSR5 (sourcing from the loopback so it works reliably) towards this new group. When XRv2 receives the register message, it does nothing special, and the signaling follows the same PIM ASM process we have reviewed many times.

```
R5#ping FF7E:C40:2008::3 source loopback0 repeat 100000 timeout 1
Output Interface: loopback0
```

XRv2 sends the (S,G) back towards 2009::5, which goes through CSR6 and CSR10 on the way. This is based on the BGP IPv6 multicast route which describes the RPF back to the source in question.

```
RP/0/0/CPU0:XRv2#show pim ipv6 topology | begin 2009
(2009::5, ff7e:c40:2008::3) SPT
SM Up: 00:09:56 JP: Join(00:00:10) Flags: KAT(00:02:59) RA RR (00:04:01)
RPF: GigabitEthernet0/0/0/0.562, fe80::6
    GigabitEthernet0/0/0/0.582 00:00:34 fwd Join(00:02:55)
```

```
R6#show ipv6 mroute FF7E:C40:2008::3 | begin \
(2009::5, FF7E:C40:2008::3), 00:04:16/00:03:02, flags: ST
Incoming interface: GigabitEthernet2.560
RPF nbr: FE80::10
Immediate Outgoing interface list:
    GigabitEthernet2.562, Forward, 00:04:16/00:03:02
```

```
R10#show ipv6 mroute FF7E:C40:2008::3 | begin \
(2009::5, FF7E:C40:2008::3), 00:04:26/00:03:03, flags: ST
Incoming interface: GigabitEthernet2.550
RPF nbr: FE80::5
Immediate Outgoing interface list:
    GigabitEthernet2.560, Forward, 00:04:26/00:03:03
```

CSR8 switches over to the SPT (not terribly relevant in this design or for embedded RP) and is actively receiving packets along this tree. I issue the command a few seconds apart to ensure the counters are incrementing, which shows packets being forwarded towards the client, CSR3.

```
R8#show ipv6 mroute FF7E:C40:2008::3 2009::5 count | begin ^Group
Group: FF7E:C40:2008::3
Source: 2009::5,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 107/1/118/0, Other: 0/0/0
Totals - Source count: 1, Packet count: 107
```

```
R8#show ipv6 mroute FF7E:C40:2008::3 2009::5 count | begin ^Group
Group: FF7E:C40:2008::3
Source: 2009::5,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 137/1/118/0, Other: 0/0/0
```

Totals - Source count: 1, Packet count: 137

The last technique for inter-AS multicast is scoping. XE makes it easy with a custom command that automatically blocks all traffic at the configuration scope or less. XR requires manual multicast boundaries, which is similar to the IPv4 multicast boundary method. Let's assume that AS7 and AS9 are part of the same organization but are different geographical sites. They should be able to share organization-local traffic, but not site-local traffic. Their links to ASes 8 and 11 should block organizational traffic since those ASes are not within the organization. First, we will configure the XE routers in AS 9 since they are simpler. CSR10 limits scope 5 (site-local) and below within the AS, while CSR6 limits scope 8 (organization-local) and below within the AS. This means that global traffic can use any of these three links and organization-local traffic can use Gig2.501 on CSR10. I also show the context-sensitive help; we can use numbers or textual macros (which get converted into numbers).

```
R10(config-subif)#ipv6 multicast boundary scope ?
<3-15>                               Scope identifier for this zone
admin-local                          Admin-local(4)
organization-local                   Organization-local(8)
site-local                           Site-local(5)
subnet-local                         Subnet-local(3)
vpn                                  Virtual Routing/Forwarding(14)

! CSR10
interface GigabitEthernet2.501
  ipv6 multicast boundary scope 5

! CSR6
interface GigabitEthernet2.562
  ipv6 multicast boundary scope 8

interface GigabitEthernet2.569
  ipv6 multicast boundary scope 8
```

On XR, we have to create exhaustive ACLs. Since wildcards are not supported for IPv6 ACLs on either XE or XR, we enumerate all of the possible group ranges. Since the 'R', 'P', and 'T' bits have some dependencies upon one another, we don't need to enumerate all 16 combinations for the third hex digit, but we do need to account for the combinations shown below (there are four). We also should account for lesser, routable scopes, such as scopes 3 (subnet-local) and 4 (admin-local). The ACL on XRv4 is even longer since it has to account for all routable scopes at value 8 or less.

```
! XRv1
ipv6 access-list ACL_SITE_BOUNDARY
 10 deny ipv6 any ff05::/32
 20 deny ipv6 any ff15::/32
 30 deny ipv6 any ff35::/32
 40 deny ipv6 any ff75::/32
 50 deny ipv6 any ff04::/32
```

```
60 deny ipv6 any ff14::/32
70 deny ipv6 any ff34::/32
80 deny ipv6 any ff54::/32
90 deny ipv6 any ff03::/32
100 deny ipv6 any ff13::/32
110 deny ipv6 any ff33::/32
120 deny ipv6 any ff73::/32
130 permit ipv6 any any
```

```
multicast-routing
address-family ipv6
  interface GigabitEthernet0/0/0/0.501
    boundary ACL_SITE_BOUNDARY
```

```
! XRv4
```

```
ipv6 access-list ACL_ORG_BOUNDARY
```

```
10 deny ipv6 any ff08::/32
20 deny ipv6 any ff18::/32
30 deny ipv6 any ff38::/32
40 deny ipv6 any ff78::/32
50 deny ipv6 any ff07::/32
60 deny ipv6 any ff17::/32
70 deny ipv6 any ff37::/32
80 deny ipv6 any ff57::/32
90 deny ipv6 any ff06::/32
100 deny ipv6 any ff16::/32
110 deny ipv6 any ff36::/32
120 deny ipv6 any ff76::/32
130 deny ipv6 any ff05::/32
140 deny ipv6 any ff15::/32
150 deny ipv6 any ff35::/32
160 deny ipv6 any ff75::/32
170 deny ipv6 any ff04::/32
180 deny ipv6 any ff14::/32
190 deny ipv6 any ff34::/32
200 deny ipv6 any ff54::/32
210 deny ipv6 any ff03::/32
220 deny ipv6 any ff13::/32
230 deny ipv6 any ff33::/32
240 deny ipv6 any ff73::/32
250 permit ipv6 any any
```

```
multicast-routing
address-family ipv6
  interface GigabitEthernet0/0/0/0.524
    boundary ACL_ORG_BOUNDARY

  interface GigabitEthernet0/0/0/0.594
    boundary ACL_ORG_BOUNDARY
```

We can see the effect of this within our existing topology. Earlier, when we tested IPv6 SSM, we used group FF33::4. This group is subnet-local with a scope value of 3, which is relatively low. XRv3 and CSR9 both have the (S,G) join for this as they are trying to build the tree back towards CSR1.

```
RP/0/0/CPU0:XRv3#show pim ipv6 topology | begin ff33
(2007:7:1:7::1,ff33::4) SPT
SSM Up: 08:18:21 JP: Join(00:00:15) Flags:
RPF: GigabitEthernet0/0/0/0.593,fe80::9
GigabitEthernet0/0/0/0.543 08:18:21 fwd LI LH

R9#show ipv6 mroute ff33::4 | begin \
(2007:7:1:7::1, FF33::4), 07:47:26/00:03:17, flags: sT
Incoming interface: GigabitEthernet2.594
RPF nbr: FE80::14
Immediate Outgoing interface list:
GigabitEthernet2.593, Forward, 07:47:26/00:03:17
```

However, XRv4 should not have this (S,G) due to the multicast-boundary applied. Only traffic scoped 9 or higher can be exchanged with AS 11. We see no multicast state for this group, and furthermore the boundary shows hits against the FF33::/32 entry. This proves that the boundary is functioning as expected, and this breaks connectivity from CSR1 to CSR4 using this group.

```
RP/0/0/CPU0:XRv4#show pim ipv6 topology ff33::4
No PIM topology table entries found.
```

```
RP/0/0/CPU0:XRv4#show access-list ipv6
ipv6 access-list ACL_ORG_BOUNDARY
10 deny ipv6 any ff08::/32
[snip, other entries are not interesting]
210 deny ipv6 any ff03::/32
220 deny ipv6 any ff13::/32
230 deny ipv6 any ff33::/32 (8 matches)
240 deny ipv6 any ff73::/32
250 permit ipv6 any any
```

To test the XE scope-based boundaries, we will try to join a new group on CSR4 via CSR2 as the source. The group is FF38::4 which is exactly equal to the scope configured on the AS 9 border routers; we expect the (S,G) join from CSR9 to be rejected by CSR6.

```
! CSR4
interface GigabitEthernet2.543
ipv6 mld join-group FF38::4 2009:9:2:5::2

RP/0/0/CPU0:XRv3#show pim ipv6 topology | begin ff38
(2009:9:2:5::2,ff38::4) SPT
SSM Up: 00:00:18 JP: Join(00:00:57) Flags:
```

```

RPF: GigabitEthernet0/0/0/0.593, fe80::9
    GigabitEthernet0/0/0/0.543  00:00:18  fwd LI LH

R9#show ipv6 mroute ff38::4 | begin \
(2009:9:2:5::2, FF38::4), 00:00:30/00:02:59, flags: sT
    Incoming interface: GigabitEthernet2.569
    RPF nbr: FE80::6
    Immediate Outgoing interface list:
        GigabitEthernet2.593, Forward, 00:00:30/00:02:59

```

Looking at CSR6, it has no entry for this group. There aren't many positive-logic show commands for this feature. The best way to verify is by confirming the absence of the groups in question on a given router. IPv6 PIM debugging on XE does reveal that the (S,G) join from CSR9 is rejected due to multicast scope, but there does not appear to be a comparable show command.

```

R6#show ipv6 mroute ff38::4
No mroute entries found.

! CSR6
IPv6 PIM: Received J/P on GigabitEthernet2.569 from FE80::9 target: FE80::6
(to us)
IPv6 PIM: J/P Group FF38::4 is a scoped boundary or denied, skipping

```

Configuring another SSM join on CSR1 with a group of similar scope towards CSR2, we can confirm that this works. CSR7 receives the INCLUDE-mode join which contains the source of CSR2. This is because AS 7 and AS 9 are allowed to share organization-scope IPv6 multicast.

```

! CSR1
interface GigabitEthernet2.517
    ipv6 mld join-group FF38::1 2009:9:2:5::2

R7#show ipv6 mld groups FF38::1 detail
Interface:      GigabitEthernet2.517
Group:          FF38::1
Uptime:         00:00:23
Router mode:    INCLUDE
Host mode:      INCLUDE
Last reporter:  FE80::1
Group source list:
Source Address              Uptime    Expires    Fwd  Flags
2009:9:2:5::2              00:00:23  00:03:56  Yes  Remote 4

```

CSR7 originates an (S,G) join towards CSR2, which is via XRv1. XRv1 then sends the (S,G) join towards CSR10 which is the RPF neighbor for the source. It passes the filter outbound on XRv1 since the scope is organization-local. If the multicast boundary denied this group, the link to CSR10 would not be in the OIL.


```

R7#show ipv6 mroute FF38::1 | begin \
(2009:9:2:5::2, FF38::1), 00:02:02/never, flags: sTI
Incoming interface: GigabitEthernet2.571
RPF nbr: FE80::11
Immediate Outgoing interface list:
GigabitEthernet2.517, Forward, 00:02:02/never

RP/0/0/CPU0:XRv1#show pim ipv6 topology | begin ff38
(2009:9:2:5::2, ff38::1) SPT
SSM Up: 00:02:20 JP: Join(00:00:14) Flags:
RPF: GigabitEthernet0/0/0/0.501, fe80::10
GigabitEthernet0/0/0/0.571 00:02:20 fwd Join(00:03:16)

```

The received (S,G) join likewise passes the filter inbound on CSR10 since the groups scope of 8 is greater than the boundary scope of 5. CSR10 passes the join to CSR5 who is the root of the SPT.

```

R10#show ipv6 mroute FF38::1 | begin \
(2009:9:2:5::2, FF38::1), 00:03:27/00:02:38, flags: sT
Incoming interface: GigabitEthernet2.550
RPF nbr: FE80::5
Immediate Outgoing interface list:
GigabitEthernet2.501, Forward, 00:03:27/00:02:38

R5#show ipv6 mroute FF38::1 | begin \
(2009:9:2:5::2, FF38::1), 00:03:42/00:02:50, flags: sT
Incoming interface: GigabitEthernet2.525
RPF nbr: 2009:9:2:5::2
Immediate Outgoing interface list:
GigabitEthernet2.550, Forward, 00:03:42/00:02:50

```

To confirm that organization-local multicast works, we ping from CSR2 and check the counters on CSR7 to ensure they increment.

```

R7#show ipv6 mroute FF38::1 count | begin ^Group
Group: FF38::1
Source: 2009:9:2:5::2,
SW Forwarding: 0/0/0/0, Other: 0/0/0
HW Forwarding: 12/1/118/0, Other: 0/0/0
Totals - Source count: 1, Packet count: 12

R7#show ipv6 mroute FF38::1 count | begin ^Group
Group: FF38::1
Source: 2009:9:2:5::2,
SW Forwarding: 0/0/0/0, Other: 0/0/0
HW Forwarding: 22/1/118/0, Other: 0/0/0
Totals - Source count: 1, Packet count: 22

```

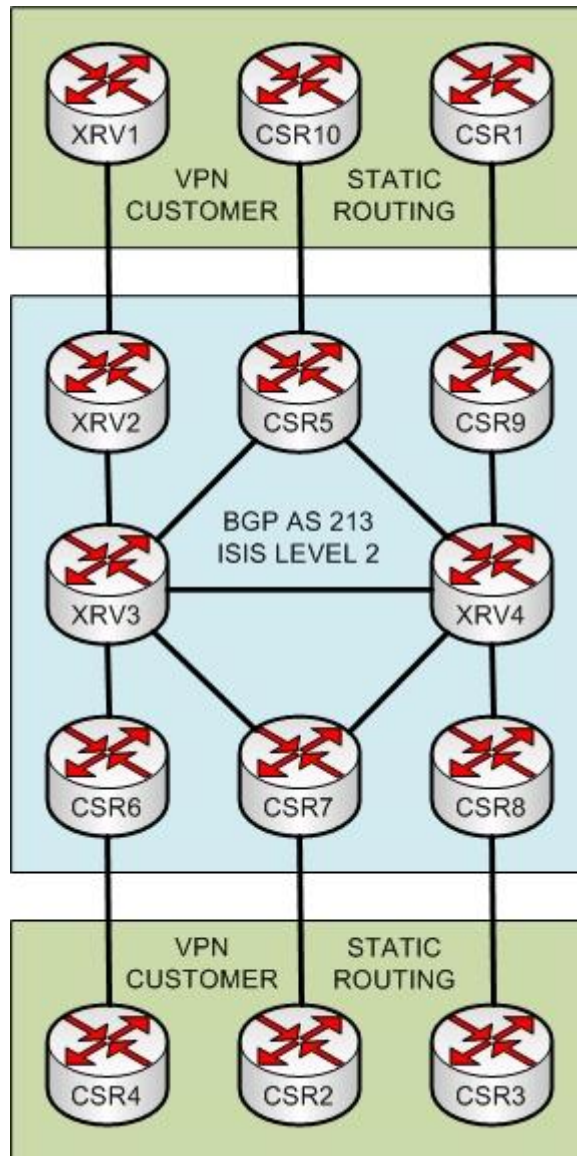
In summary, there are several key lessons from this chapter:

- It is bad practice to let local RP information (BSR, AutoRP) leak in/out of a multicast domain/AS
 1. Use BSR border at the boundaries
 2. Filter AutoRP groups manually at the boundaries
 3. Last ditch effort: Use static RP “overrides”
- Cisco and RFC3618 MSDP RPF rules are different. XE uses Cisco’s rules by default while XR uses RFC3618. I recommend enabling RFC3618 RPF rules on XE routers because they are simpler to understand and consistent with XR platforms. This also enables the “show ip msdp rpf-peer” command within XE which returns the RFC3618 RPF rule used to match the RP input.
- When XR has BGP multicast AFIs configured, the RPF lookup always uses this table over the corresponding BGP unicast AFI. Existing BGP unicast routes, however valid they may be, cannot be used for RPF unless the corresponding multicast AFI is totally disabled on the router.
- For IPv4, inter-AS ASM almost always requires MSDP. SSM is much simpler as it removes the need for RPs and MSDP entirely.
- By default, MSDP is very liberal with its SA advertisement and relies heavily on RPF to prevent loops. A well-tuned MSDP network makes use of mesh-groups, TTL-thresholds, redistribution filters, and SA filters to minimize SA flooding in the first place.
- For IPv6, there are three main design options for enabling inter-AS multicast service:
 1. SSM: Simplest and most reliable
 2. Embedded RP ASM: Works well when used in conjunction with RP ACLs and scopes
 3. Scoping with globally-reachable RPs: Difficult to manage but possible

Additional Reading – Reference configurations “inter-as-mcast”

29.2 Multicast Only Fast Re-Reroute (MoFRR)

MoFRR is a feature used to ensure high availability for multicast traffic. It does not introduce any new protocols but adjusts how RPF works with traditional PIM. It is also not specific to MVPN or MPLS in general. When the PIM last hop router (LHR) receives an IGMP join, whether ASM or SSM, it will originate some kind of PIM join. ASM joins will be (*,G) PIM joins sent to the RP while SSM joins will be (S,G) PIM joins sent to the first hop router (FHR) connected to the source. In either case, the LHR and all intermediate routers towards the root of the given tree select exactly one RPF interface. This will normally follow the shortest IGP path to the root, and in the case of ECMP, select the PIM neighbor with the highest IP address. MoFRR allows a router to send multiple PIM joins towards the root of the tree for diversity; this technique works best when the two ECMP paths are separate networks or “planes” that are totally independent, but this is not a requirement. We can test it using an existing topology. Specifically, this section uses the MVPN profile 0 with PIM-SSM basic configuration as a starting point. The diagram is shown again for reference.



The only initial modification is that the direct link between XRv3 and XRv4 has been adjusted to an IS-IS metric of 20. In this way, there are 3 ECMPs between XRv3 and XRv4; MoFRR requires ECMP to operate. We can check the FIB on each core router towards one another to verify this configuration change.

```
! XRv3 and XRv4
router isis 1
interface GigabitEthernet0/0/0/0.534
point-to-point
address-family ipv4 unicast
metric 20

address-family ipv6 unicast
metric 20
```

```
RP/0/0/CPU0:XRv3#show cef ipv4 213.14.14.14 | include via
  via 213.13.14.14, GigabitEthernet0/0/0/0.534, 5 dependencies, weight 0,
class 0 [flags 0x0]
  via 213.5.13.5, GigabitEthernet0/0/0/0.553, 8 dependencies, weight 0,
class 0 [flags 0x0]
  via 213.7.13.7, GigabitEthernet0/0/0/0.573, 8 dependencies, weight 0,
class 0 [flags 0x0]
```

```
RP/0/0/CPU0:XRv4#show cef ipv4 213.13.13.13 | include via
  via 213.13.14.13, GigabitEthernet0/0/0/0.534, 5 dependencies, weight 0,
class 0 [flags 0x0]
  via 213.5.14.5, GigabitEthernet0/0/0/0.554, 8 dependencies, weight 0,
class 0 [flags 0x0]
  via 213.7.14.7, GigabitEthernet0/0/0/0.574, 8 dependencies, weight 0,
class 0 [flags 0x0]
```

Next, we will configure some new SSM groups within the customer VPN. Since the core of the network is purely using SSM, that is what will be tested. Using SSM in the customer VPN will simplify the design and allow us to focus on MoFRR, not basic PIM-ASM processes. To support data MDTs for sources behind it, CSR1's data MDT ACL will be extended to the entire SSM range, since CSR9 will be an IPv4 multicast source for this test. We will join a new SSM group on CSR4 as this will be the receiver.

```
! CSR1
ip access-list extended ACL_DATA_MDT
  no 10
  10 permit ip any 232.0.0.0 0.255.255.255

! CSR4
interface GigabitEthernet2.546
  ip igmp join-group 232.42.51.8 source 10.1.9.9
```

Before continuing, we will verify that the proper MDT has been built. CSR6 receives the IGMPv2 join operating in INCLUDE mode; we verify the proper source is carried in the membership report. A corresponding C(S,G) entry is created for this source/group pair.

```
R6#show ip igmp vrf MC groups 232.42.51.8 detail
[snip]
Interface:      GigabitEthernet2.546
Group:         232.42.51.8
Flags:         SSM
Uptime:        00:19:39
Group mode:    INCLUDE
Last reporter: 10.4.6.4
[snip]
```

Source Address	Uptime	v3 Exp	CSR Exp	Fwd	Flags
10.1.9.9	00:19:39	00:02:30	stopped	Yes	R

CSR6 translates this into a C-PIM join and sends it to CSR1. The join is sent within the default MDT which is used for C-PIM signaling.

```
R6#debug ip pim vrf MC
PIM(1): Insert (10.1.9.9,232.42.51.8) join in nbr 213.1.1.1's queue
PIM(1): Building Join/Prune packet for nbr 213.1.1.1
PIM(1): Adding v2 (10.1.9.9/32, 232.42.51.8), S-bit Join
PIM(1): Send v2 join/prune to 213.1.1.1 (Tunnel3)
```

Upon receipt, CSR1 adds the C(S,G) state to its C-MRIB and notifies CSR6, via PIM MDT TLV, that a data MDT is available for this C(S,G).

```
R1#debug ip pim vrf MC
PIM(17): Received v2 Join/Prune on Tunnel3 from 213.6.6.6, to us
PIM(17): Join-list: (10.1.9.9/32, 232.42.51.8), S-bit set
PIM(17): Add Tunnel3/213.6.6.6 to (10.1.9.9, 232.42.51.8), Forward state, by
PIM SG Join
PIM(17): MDT next_hop change from: 232.255.0.1 to 232.4.1.0 for (10.1.9.9,
232.42.51.8) Tunnel3
```

```
R1#show ip pim mdt send
MDT-data send list for VRF: MC
      (source, group)                MDT-data group/num  ref_count
(10.1.9.9, 232.42.51.8)            232.4.1.0          1
```

CSR6 receives this update, updates its C-MRIB entry with the MDT data to bind it to the proper P(S,G) entry. Without this data MDT, the traffic would be forwarded within the default MDT. The big 'Y' implies this entry is joined to a data MDT, and the information is embedded within the C-RIB entry. So far, this is all review, and nothing specific to MoFRR has occurred.

```
R6#debug ip pim vrf MC
PIM(1): Receive MDT Packet (11649) from 213.1.1.1 (Tunnel3), length (ip: 44,
udp: 24), ttl: 1
PIM(1): TLV type: 1 length: 16 MDT Packet length: 16
```

```
R6#show ip pim mdt receive
Joined MDT-data [group/mdt number : source]  uptime/expires for VRF: MC
[232.4.1.0 : 213.1.1.1]  00:24:13/00:02:45
[232.4.7.0 : 213.7.7.7]  03:55:34/00:02:25
```

```
R6#show ip mroute vrf MC 232.42.51.8 10.1.9.9 | begin \
(10.1.9.9, 232.42.51.8), 00:26:17/00:02:50, flags: sTIY
  Incoming interface: Tunnel3, RPF nbr 213.1.1.1,
MDT: [213.1.1.1,232.4.1.0]/00:02:11
  Outgoing interface list:
    GigabitEthernet2.546, Forward/Sparse, 00:07:08/00:02:50
```

As a final verification, we will check the P(S,G) entry on CSR6 to ensure it has been created. The RPF interface points towards XRv3 as expected. The big 'Z' identifies this as a multicast tunnel, and the OIL includes the multicast VRF being serviced.

```
R6#show ip mroute 232.4.1.0 213.1.1.1 | begin \  
(213.1.1.1, 232.4.1.0), 00:28:08/stopped, flags: sTIZ  
Incoming interface: GigabitEthernet2.563, RPF nbr 213.6.13.13  
Outgoing interface list:  
MVRF MC, Forward/Sparse, 00:10:28/00:01:31
```

We know that this join must transit XRv3 since that is the only path CSR6 has to reach CSR1. XRv3 has three options to reach 213.1.1.1, which is the result of our IS-IS metric adjustment earlier. Based on this, XRv3 will select the PIM neighbor with the highest IP address, which is XRv4.

```
RP/0/0/CPU0:XRv3#show route ipv4 213.1.1.1 | include via  
Known via "isis 1", distance 115, metric 30, type level-2  
213.13.14.14, from 213.1.1.1, via GigabitEthernet0/0/0/0.534  
213.5.13.5, from 213.1.1.1, via GigabitEthernet0/0/0/0.553  
213.7.13.7, from 213.1.1.1, via GigabitEthernet0/0/0/0.573
```

```
RP/0/0/CPU0:XRv3#show pim ipv4 rpf 213.1.1.1  
Table: IPv4-Unicast-default  
* 213.1.1.1/32 [115/30]  
via GigabitEthernet0/0/0/0.534 with rpf neighbor 213.13.14.14
```

Although both CSR7 and CSR5 are viable alternative paths, they do not receive the P(S,G) join from XRv3 since they are not in the reverse path towards the root of the tree. We can confirm this issue by checking the P-MRIB entry on CSR5 and CSR7; this group is not found. XRv3 also makes no mention of any attempt to involve those routers in the multicast delivery path, either. So far, this is all expected PIM behavior.

```
R5#show ip mroute 232.4.1.0 213.1.1.1  
Group 232.4.1.0 not found
```

```
R7#show ip mroute 232.4.1.0 213.1.1.1  
Group 232.4.1.0 not found
```

```
RP/0/0/CPU0:XRv3#show pim ipv4 topology 232.4.1.0 213.1.1.1 | begin 232  
(213.1.1.1,232.4.1.0)SPT SSM Up: 00:15:09  
JP: Join(00:00:35) RPF: GigabitEthernet0/0/0/0.534,213.13.14.14 Flags:  
GigabitEthernet0/0/0/0.563 00:15:09 fwd Join(00:03:08)
```

To enable MoFRR, we must have ECMP on the router on which MoFRR will be configured. In this case, XRv3 has ECMP to 213.1.1.1, and we will configure MoFRR there. The configuration is very straightforward, and we can use an ACL to select which groups to enable for MoFRR. Since MoFRR

essentially duplicates the multicast flow along alternative paths, it can be wasteful of bandwidth, and should be used sparingly for critical flows.

```
! XRv3
ipv4 access-list ACL_MOFRR
 10 permit ipv4 any 232.0.0.0 0.255.255.255

router pim
 address-family ipv4
   mofrr
     flow ACL_MOFRR
```

To verify this, we can look at the detailed P-MRIB output on XRv3. While the summary P-MRIB information does identify this as MoFRR capable, it doesn't tell us anything about the alternate path. Instead, towards the bottom of this command we can see a secondary RPF neighbor identified as CSR7. This router was selected over CSR5 since it has the next highest IP address. This means that XRv3 issues a P(S,G) join to CSR7 and XRv4 at the same time.

```
RP/0/0/CPU0:XRv3#show pim ipv4 topology 232.4.1.0 213.1.1.1 detail | begin
232
(213.1.1.1,232.4.1.0)SPT SSM Up: 00:19:06
JP: Join(00:00:08) RPF: GigabitEthernet0/0/0/0.534,213.13.14.14 MoFRR, Flags:
Up: MT clr (00:00:00) MDT: JoinSend N, Cache N/N, Misc (0x0,0/0)
Cache: Add 00:00:00, Rem 00:00:00. MT Cnt: Set 0, Unset 0. Joins sent 0
MDT-ifh 0x0/0x0, MT Slot none/ none
RPF-redirect BW usage: 0, Flags: 0x0, ObjID: 0x0
c-multicast-routing: PIM* BGPJP: 1d10h
RPF Table: IPv4-Unicast-default
RPF Secondary: GigabitEthernet0/0/0/0.573,213.7.13.7
GigabitEthernet0/0/0/0.563 00:19:06 fwd Join(00:03:07)
```

We check the P-MRIB tables on CSR5 and CSR7 again. CSR5 still doesn't have the entry, but CSR7 does, and nothing looks fancy about it. CSR7 is totally unaware that MoFRR is in effect. CSR7 identifies XRv4 as its RPF neighbor to CSR1, which is correct.

```
R5#show ip mroute 232.4.1.0 213.1.1.1
Group 232.4.1.0 not found
```

```
R7#show ip mroute 232.4.1.0 213.1.1.1 | begin \
(213.1.1.1, 232.4.1.0), 00:36:58/00:03:28, flags: sTZ
Incoming interface: GigabitEthernet2.574, RPF nbr 213.7.14.14
Outgoing interface list:
GigabitEthernet2.573, Forward/Sparse, 00:03:00/00:03:28
```

Because both XRv3 and CSR7 identified XRv4 as their RPF neighbor towards 213.1.1.1, that means XRv4 is a "merge point", and a single point of failure, for this flow. More importantly, it means that XRv4 should have both links to CSR7 and XRv3 in its OIL for this P(S,G). Like CSR7, and every other router

besides XRv3, XRv4 is totally unaware that MoFRR is enabled in the network. XRv4 will duplicate multicast traffic towards XRv3 and CSR7 concurrently. When XRv3 receives multiple copies, it will discard the ones from CSR7 (the alternate path) until the main path fails. In XR, failure is determined by a loss of traffic for 30 ms; after this time, switchover occurs, and the entire process takes about 50 ms on real hardware routers.

```
RP/0/0/CPU0:XRv4#show pim ipv4 topology 232.4.1.0 213.1.1.1 | begin 232
(213.1.1.1,232.4.1.0)SPT SSM Up: 00:23:10
JP: Join(00:00:43) RPF: GigabitEthernet0/0/0/0.514,213.1.14.1 Flags:
GigabitEthernet0/0/0/0.534 00:23:10 fwd Join(00:02:45)
GigabitEthernet0/0/0/0.574 00:04:44 fwd Join(00:03:06)
```

We begin sending traffic on CSR9 to test this. Since all PIM signaling is complete, CSR4 should be receiving this traffic. For brevity, we will verify that traffic is arriving on CSR6 within the P(S,G) entry, and being transferred to the C(S,G) group for forwarding. We can see the packet counts are identical between the two, as expected.

```
R9#ping 232.42.51.8 repeat 10000 time 1
```

```
R6#show ip mroute 232.4.1.0 213.1.1.1 count | begin Group
Group: 232.4.1.0, Source count: 1, Packets forwarded: 85, Packets received:
85
Source: 213.1.1.1/32, Forwarding: 85/1/142/1, Other: 85/0/0
```

```
R6#show ip mroute vrf MC 232.42.51.8 10.1.9.9 count | begin Group
Group: 232.42.51.8, Source count: 1, Packets forwarded: 85, Packets received:
85
Source: 10.1.9.9/32, Forwarding: 85/1/142/1, Other: 85/0/0
```

XRv3 has very interesting output. I include the forwarding legend for clarity since the counters are different than a normal setup. We can see that 446 packets have arrived, but exactly half of them have been forwarded. The other half are identified as RPF failures.

```
RP/0/0/CPU0:XRv3#show mfib ipv4 route 232.4.1.0 213.1.1.1 | begin ^Forwarding
Forwarding/Replication Counts: Packets in/Packets out/Bytes out
Failure Counts: RPF / TTL / Empty Olist / Encap RL / Other

(213.1.1.1,232.4.1.0), Flags: MoFE MoFS
Up: 00:29:12
Last Used: 00:00:00
SW Forwarding Counts: 446/223/27652
SW Replication Counts: 446/223/27652
SW Failure Counts: 223/0/0/0/0
GigabitEthernet0/0/0/0.534 Flags: A, Up:00:29:12
GigabitEthernet0/0/0/0.563 Flags: NS EG, Up:00:29:12
GigabitEthernet0/0/0/0.573 Flags: A2, Up:00:10:46
```


Looking at the P-MRIB, which shows slightly different information than the P-PIM topology, we see the interface towards XRv4 is identified as 'A' for accept. This is the RPF interface. The interface towards CSR7 is identified as 'A2' because this is the secondary RPF, and this is where the packets are being dropped. We can see the MoFRR is reported as "inactive", which is expected in a stable network.

```
RP/0/0/CPU0:XRv3#show mrib ipv4 route 232.4.1.0 | begin 232
(213.1.1.1,232.4.1.0) RPF nbr: 213.13.14.14 Flags: RPF MoFE MoFS
Up: 00:31:08
MOFRR State: Inactive Sequence No 1
Incoming Interface List
  GigabitEthernet0/0/0/0.534 Flags: A, Up: 00:31:08
  GigabitEthernet0/0/0/0.573 Flags: A2, Up: 00:12:42
Outgoing Interface List
  GigabitEthernet0/0/0/0.563 Flags: F NS, Up: 00:31:08
```

We can further confirm that both XRv4 and CSR7 are forwarding equal quantities of packets to XRv3, which would explain the exact 50% drop rate. The CSR only updates its counters every 10 seconds which explains the tiny discrepancy in the packet counts. Notice that on XRv4, the software replication process has twice as many packets going out as coming in; this proves that XRv4 is replicating packets properly.

```
R7#show ip mroute 232.4.1.0 213.1.1.1 count | begin Group
Group: 232.4.1.0, Source count: 1, Packets forwarded: 489, Packets received:
489
Source: 213.1.1.1/32, Forwarding: 489/1/142/1, Other: 489/0/0
```

```
RP/0/0/CPU0:XRv4#show mfib ipv4 route 232.4.1.0 213.1.1.1 | begin 232
(213.1.1.1,232.4.1.0), Flags:
Up: 00:33:41
Last Used: 00:00:00
SW Forwarding Counts: 492/492/61008
SW Replication Counts: 492/984/122016
SW Failure Counts: 0/0/0/0/0
GigabitEthernet0/0/0/0.514 Flags: A, Up:00:33:41
GigabitEthernet0/0/0/0.534 Flags: NS EG, Up:00:33:41
GigabitEthernet0/0/0/0.574 Flags: NS EG, Up:00:15:15
```

To test MoFRR, we will shutdown XRv4's interface to XRv3 (not shown). MoFRR determines that the link has failed when the PIM neighbor fails (in the graceful shutdown I did, it happens immediately). MoFRR is still receiving packets from CSR7 and makes this the primary RPF interface, marked with an 'A' below. Because a third path exists, XRv3 signals a P(S,G) join to CSR5 to use it for a backup path now, marked 'A2'. Also note that while MoFRR is still technically inactive, the sequence number is 2, which counts the number of MoFRR triggers per (S,G). MoFRR is only active for the time it takes PIM to reconverge its forwarding state in the network. This is similar to TE-FRR except RSVP is used, not PIM.

```
RP/0/0/CPU0:XRv3#show mrib ipv4 route 232.4.1.0 | begin 232
(213.1.1.1,232.4.1.0) RPF nbr: 213.7.13.7 Flags: RPF MoFE MoFS
```

```

Up: 00:37:42
MOFRR State: Inactive Sequence No 2
Incoming Interface List
  GigabitEthernet0/0/0/0.553 Flags: A2, Up: 00:00:46
  GigabitEthernet0/0/0/0.573 Flags: A, Up: 00:19:16
Outgoing Interface List
  GigabitEthernet0/0/0/0.563 Flags: F NS, Up: 00:37:42

```

We verify that CSR5 is now part of the P-PIM control plane for this multicast tunnel, and that XRv4 has added CSR5 to its OIL. The OIL now contains CSR5 and CSR7 only.

```

R5#show ip mroute 232.4.1.0 213.1.1.1 | begin \
(213.1.1.1, 232.4.1.0), 00:01:11/00:03:17, flags: sTZ
  Incoming interface: GigabitEthernet2.554, RPF nbr 213.5.14.14
  Outgoing interface list:
    GigabitEthernet2.553, Forward/Sparse, 00:01:11/00:03:17

RP/0/0/CPU0:XRv4#show pim ipv4 topology 232.4.1.0 213.1.1.1 | begin 232
(213.1.1.1,232.4.1.0)SPT SSM Up: 00:41:07
JP: Join(00:00:46) RPF: GigabitEthernet0/0/0/0.514,213.1.14.1 Flags:
  GigabitEthernet0/0/0/0.554 00:04:11 fwd Join(00:03:14)
  GigabitEthernet0/0/0/0.574 00:22:41 fwd Join(00:02:50)

```

Finally, we ensure CSR6 is still receiving packets. We check the counters 10 seconds apart to verify this.

```

R6#show ip mroute vrf MC 232.42.51.8 10.1.9.9 count | begin Group
Group: 232.42.51.8, Source count: 1, Packets forwarded: 705, Packets
received: 705
  Source: 10.1.9.9/32, Forwarding: 705/1/142/1, Other: 705/0/0

R6#show ip mroute vrf MC 232.42.51.8 10.1.9.9 count | begin Group
Group: 232.42.51.8, Source count: 1, Packets forwarded: 715, Packets
received: 715
  Source: 10.1.9.9/32, Forwarding: 715/1/142/1, Other: 715/0/0

```

Before continuing, we will bring the link between XRv3 and XRv4 back. We will also stop the pings from CSR9 to reset the network to an idle state. We quickly verify that XRv3 re-establishes its proper RPF interface via XRv4, which triggers a prune to CSR5. Notice that MoFRR sequence number is 3, which indicates another change occurred when the link came back up.

```

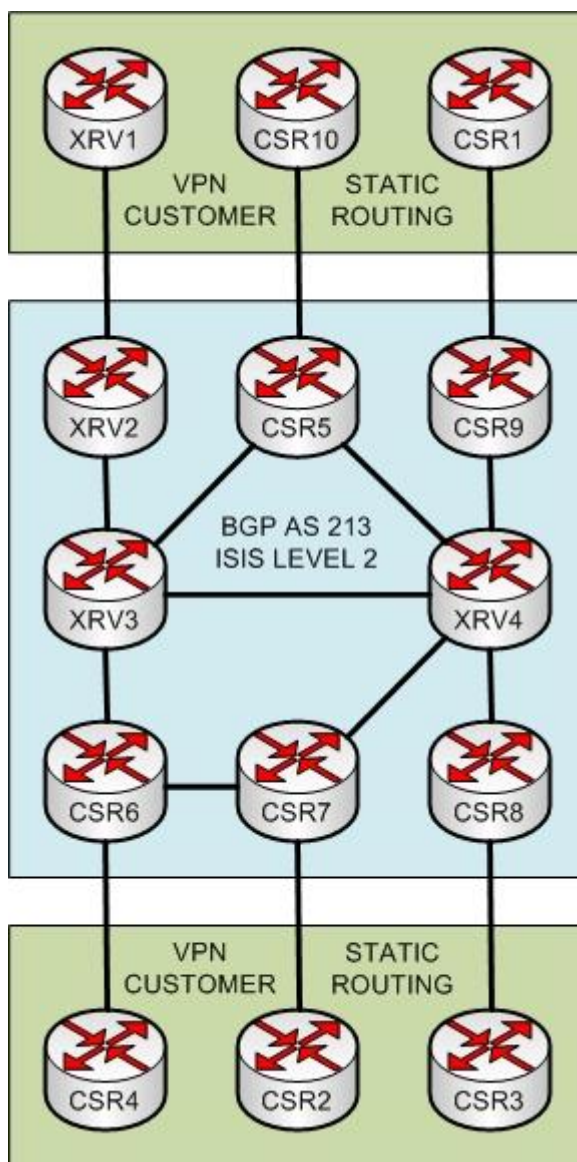
RP/0/0/CPU0:XRv3#show mrib ipv4 route 232.4.1.0 | begin 232
(213.1.1.1,232.4.1.0) RPF nbr: 213.13.14.14 Flags: RPF MoFE MoFS
Up: 00:43:40
MOFRR State: Inactive Sequence No 3
Incoming Interface List
  GigabitEthernet0/0/0/0.534 Flags: A, Up: 00:00:31
  GigabitEthernet0/0/0/0.573 Flags: A2, Up: 00:25:15
Outgoing Interface List

```

```
GigabitEthernet0/0/0/0.563 Flags: F NS, Up: 00:43:40
```

```
R5#show ip mroute 232.4.1.0 213.1.1.1 | begin \  
(213.1.1.1, 232.4.1.0), 00:08:30/00:02:59, flags: sPTZ  
Incoming interface: GigabitEthernet2.554, RPF nbr 213.5.14.14  
Outgoing interface list: Null
```

To test MoFRR on XE, we will add a new link between CSR6 and CSR7, while concurrently shutting down the link between XRv3 and CSR7. This gives CSR6 two options for its original P(S,G), and still leaves XRv3 with two options as well. In order for ECMP to work from CSR6 to CSR1, the IS-IS cost of this new link must be 20. Because CSR7 and XRv3 no longer have direct reachability, we have additional path diversity. The path from CSR6 to XRv4 is totally independent of links and nodes via CSR7 and XRv3, which is ideal for HA design. The updated diagram is shown below.



The interface configurations are basic and are not shown, but we will verify the ECMP behavior as I just described. CSR6 has two paths via XRv3 and CSR7, and XRv3 has two paths via XRv4 and CSR5.

```
R6#show ip cef 213.1.1.1
213.1.1.1/32
  nexthop 213.6.7.7 GigabitEthernet2.567 label 7005
  nexthop 213.6.13.13 GigabitEthernet2.563 label 93005

RP/0/0/CPU0:XRv3#show cef ipv4 213.1.1.1 | include via
  via 213.13.14.14, GigabitEthernet0/0/0/0.534, 7 dependencies, weight 0,
class 0 [flags 0x0]
  via 213.5.13.5, GigabitEthernet0/0/0/0.553, 10 dependencies, weight 0,
class 0 [flags 0x0]
```

In this way, we can have MoFRR enabled both at the edge and core routers for additional redundancy. In a purely planar network, this wouldn't be necessary, since the two planes would be "enough" redundancy for the vast majority of cases. I demonstrate it here because it is possible; there is nothing that says MoFRR is limited to a certain type or quantity of nodes. The configuration on XE is very similar to XR, except we will enable the "sticky" option. This ensures that the primary RPF doesn't change even if a better one shows up later. This leads to better stability in the network, and unlike with XRv3, we would not prune stable links to choose a link that just came back up. This is also supported in XR and the option is "non-revertive", which is a little more descriptive. XE MoFRR can only accept an extended ACL, so we will identify the specific tunnel endpoint of CSR1, along with all SSM groups for variety.

```
! CSR1
ip multicast rpf mofrr ACL_MOFRR sticky
ip access-list extended ACL_MOFRR
 permit ip host 213.1.1.1 232.0.0.0 0.255.255.255
```

To verify it, we can look at the P(S,G) entry for the multicast tunnel. The output is comparable to XR. We can see RIB-based MoFRR (different than flow-based MoFRR described earlier) is enabled and the secondary RPF information is made very clear. CSR6 selects XRv3 as the primary RPF due to having a low IP address. Otherwise, nothing is different.

```
R6#show ip mroute 232.4.1.0 | begin \
(213.1.1.1, 232.4.1.0), 01:19:06/stopped, flags: sTIZ
  Incoming interface: GigabitEthernet2.563, RPF nbr 213.6.13.13, RIB based
MoFRR
  Secondary RPF interface: GigabitEthernet2.567, Secondary RPF nbr 213.6.7.7
  Outgoing interface list:
    MVRP MC, Forward/Sparse, 01:01:26/00:01:33
```

We verify that CSR7 received the P(S,G) join for this group and adds CSR6 to its OIL. XRv3 is still running MoFRR and has added both XRv4 and CSR5 as RPF interfaces, which means both of them should have the entry, too.

```
R7#show ip mroute 232.4.1.0 213.1.1.1 | begin \  
(213.1.1.1, 232.4.1.0), 01:18:54/00:03:10, flags: sTZ  
  Incoming interface: GigabitEthernet2.574, RPF nbr 213.7.14.14  
  Outgoing interface list:  
    GigabitEthernet2.567, Forward/Sparse, 00:02:58/00:03:10  
  
RP/0/0/CPU0:XRv3#show mrib ipv4 route 232.4.1.0 | begin 232  
(213.1.1.1,232.4.1.0) RPF nbr: 213.13.14.14 Flags: RPF MoFE MoFS  
Up: 00:11:08  
MOFRR State: Inactive Sequence No 1  
Incoming Interface List  
  GigabitEthernet0/0/0/0.534 Flags: A, Up: 00:11:08  
  GigabitEthernet0/0/0/0.553 Flags: A2, Up: 00:11:08  
Outgoing Interface List  
  GigabitEthernet0/0/0/0.563 Flags: F NS, Up: 00:11:08
```

We quickly verify CSR5 and XRv4 as well. CSR5's output is very basic and not exciting, but now XRv4 will be very busy. It now has three interfaces in its OIL for this group having received P(S,G) joins from CSR5, CSR7, and XRv3. There is only one receiver yet the traffic is being replicated three times, which is highly inefficient.

```
R5#show ip mroute 232.4.1.0 213.1.1.1 | begin \  
(213.1.1.1, 232.4.1.0), 00:27:36/00:02:42, flags: sTZ  
  Incoming interface: GigabitEthernet2.554, RPF nbr 213.5.14.14  
  Outgoing interface list:  
    GigabitEthernet2.553, Forward/Sparse, 00:11:52/00:02:42  
  
RP/0/0/CPU0:XRv4#show pim ipv4 topology 232.4.1.0 213.1.1.1 | begin 232  
(213.1.1.1,232.4.1.0)SPT SSM Up: 00:12:57  
JP: Join(now) RPF: GigabitEthernet0/0/0/0.514,213.1.14.1 Flags:  
  GigabitEthernet0/0/0/0.534 00:12:57 fwd Join(00:02:37)  
  GigabitEthernet0/0/0/0.554 00:12:57 fwd Join(00:03:23)  
  GigabitEthernet0/0/0/0.574 00:05:14 fwd Join(00:02:54)
```

To test this, we will send pings from CSR9 again. This time we will move from XRv4 down to CSR6. As expected, XRv4 replicates the traffic three times, as we can see by the 30 packets received and 90 transmitted. Packets are accepted from CSR1 ("A" flag) and egress out of three other interface ("EG" flag).

```
R9#ping 232.42.51.8 repeat 10000 time 1  
  
RP/0/0/CPU0:XRv4#show mrib ipv4 route 232.4.1.0 213.1.1.1 | begin 232  
(213.1.1.1,232.4.1.0), Flags:
```

```

Up: 00:16:59
Last Used: 00:00:00
SW Forwarding Counts: 30/30/3720
SW Replication Counts: 30/90/11160
SW Failure Counts: 0/0/0/0/0
GigabitEthernet0/0/0/0.514 Flags: A, Up:00:16:59
GigabitEthernet0/0/0/0.534 Flags: NS EG, Up:00:16:59
GigabitEthernet0/0/0/0.554 Flags: NS EG, Up:00:16:59
GigabitEthernet0/0/0/0.574 Flags: NS EG, Up:00:09:15

```

There is nothing new on XRv3, since the duplicated packets arriving from CSR5 are dropped due to RPF failure. 180 packets have arrived, but only 90 are transmitted. XRv3 still forwards traffic to CSR6, which is imported because CSR6 has selected XRv3 as the primary RPF neighbor.

```

RP/0/0/CPU0:XRv3#show mfib ipv4 route 232.4.1.0 213.1.1.1 | begin 232
(213.1.1.1,232.4.1.0), Flags: MoFE MoFS
Up: 00:17:59
Last Used: 00:00:00
SW Forwarding Counts: 180/90/11160
SW Replication Counts: 180/90/11160
SW Failure Counts: 90/0/0/0/0
GigabitEthernet0/0/0/0.534 Flags: A, Up:00:17:59
GigabitEthernet0/0/0/0.553 Flags: A2, Up:00:17:59
GigabitEthernet0/0/0/0.563 Flags: NS EG, Up:00:17:59

```

CSR5 and CSR7 are very straightforward and their outputs are not shown, since they receive a single flow of traffic and replicate it once downstream. CSR6 has interesting output as we expected to see several RPF failures for this P(S,G). The reason the forwarding packets are higher is because I did not clear the counters before this test, but it isn't necessary to see the difference.

```

R6#show ip mroute 232.4.1.0 213.1.1.1 count | begin Forwarding
Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second
Other counts: Total/RPF failed/Other drops(OIF-null, rate-limit etc)

Group: 232.4.1.0, Source count: 1, Packets forwarded: 1289, Packets received:
1553
Source: 213.1.1.1/32, Forwarding: 1289/1/142/1, Other: 1553/264/0

```

Using EPC, we will conduct a quick verification to see two packets arriving at approximately the same time from XRv3 and CSR7, but only one being forwarded to the customer. In this case, the first packet that arrived was from XRv3 and was immediately forwarded to the customer. The backup flow from CSR7 came a short time later and was discarded. I show the MAC addresses for CSR7 and XRv3 for confirmation.

```

R6#show ip arp 213.6.7.7
Protocol Address Age (min) Hardware Addr Type Interface

```

```

Internet 213.6.7.7 25 0050.56a9.ea77 ARPA Gig2.567

R6#show ip arp 213.6.13.13
Protocol Address Age (min) Hardware Addr Type Interface
Internet 213.6.13.13 91 0050.56a9.ea54 ARPA Gig2.563

R6#show monitor capture CAP buffer detailed
 0 142 0.000000 213.1.1.1 -> 232.4.1.0 GRE
0000: 01005E04 01000050 56A9EA54 8100CDEB ..^....PV..T....
0010: 08004500 007C05DE 0000FD2F F86DD501 ..E..|...../.m..
0020: 0101E804 01000000 08004500 006405EB .....E..d..
0030: 0000FE01 88710A01 0909E82A 33080800 .....q.....*3...

 1 118 0.000000 10.1.9.9 -> 232.42.51.8 ICMP
0000: 01005E2A 33080050 56A9DE0D 8100DDA ..^*3..PV.....
0010: 08004500 006405EB 0000FD01 89710A01 ..E..d.....q..
0020: 0909E82A 33080800 A2360006 01DD0000 ...*3....6.....
0030: 0000307A A9B6ABCD ABCDABCD ABCDABCD ..0z.....

 2 142 0.000992 213.1.1.1 -> 232.4.1.0 GRE
0000: 01005E04 01000050 56A9EA77 8100DEF ..^....PV..w....
0010: 08004500 007C05DE 0000FD2F F86DD501 ..E..|...../.m..
0020: 0101E804 01000000 08004500 006405EB .....E..d..
0030: 0000FE01 88710A01 0909E82A 33080800 .....q.....*3...

```

To quickly test the sticky feature, we will shutdown the link to XRv3 on CSR6. CSR6 immediately switches to CSR7 as the primary RPF interface, and since there are no alternate paths, the secondary RPF does not exist.

```

R6#show ip mroute 232.4.1.0 213.1.1.1 | begin \
(213.1.1.1, 232.4.1.0), 01:39:16/stopped, flags: sTIZ
Incoming interface: GigabitEthernet2.567, RPF nbr 213.6.7.7, RIB based
MoFRR
Secondary RPF interface: Null, Secondary RPF nbr 0.0.0.0
Outgoing interface list:
MVRF MC, Forward/Sparse, 01:21:36/00:02:47

```

Of paramount importance is CSR6's ability to continue receiving multicast traffic. Checking the packet counters, we can see the forwarded packets continue to increase with no new RPF drops. Because there isn't a second path, there isn't a backup flow to discard.

```

R6#show ip mroute 232.4.1.0 213.1.1.1 count | begin Group
Group: 232.4.1.0, Source count: 1, Packets forwarded: 1808, Packets received:
2541
Source: 213.1.1.1/32, Forwarding: 1808/1/142/1, Other: 2541/733/0

R6#show ip mroute 232.4.1.0 213.1.1.1 count | begin Group

```

```
Group: 232.4.1.0, Source count: 1, Packets forwarded: 1818, Packets received: 2551
```

```
Source: 213.1.1.1/32, Forwarding: 1818/1/142/1, Other: 2551/733/0
```

We will bring the link to XRv3 back up and see how CSR6 reacts. Ideally, CSR6 should continue using CSR7 as the primary RPF which will reduce P-PIM control-plane churn in the network core. When ECMP is in play, it may not matter which path is chosen, so constantly changing may be undesirable. As expected, XRv3 is now the secondary RPF path. When performing an unrelated RPF lookup for 213.1.1.1, we see that XRv3 is still shown, but for this particular MoFRR P(S,G) this default RPF mechanism does not take effect, thanks to the “sticky” option.

```
R6#show ip mroute 232.4.1.0 213.1.1.1 | begin \  
(213.1.1.1, 232.4.1.0), 01:42:25/stopped, flags: sTIZ  
Incoming interface: GigabitEthernet2.567, RPF nbr 213.6.7.7, RIB based  
MoFRR  
Secondary RPF interface: GigabitEthernet2.563, Secondary RPF nbr  
213.6.13.13  
Outgoing interface list:  
MVRP MC, Forward/Sparse, 01:24:45/00:02:37
```

```
R6#show ip rpf 213.1.1.1  
RPF information for ? (213.1.1.1)  
RPF interface: GigabitEthernet2.563  
RPF neighbor: ? (213.6.13.13)  
RPF route/mask: 213.1.1.1/32  
RPF type: unicast (isis 213)  
Doing distance-preferred lookups across tables  
RPF topology: ipv4 multicast base, originated from ipv4 unicast base
```

Checking the packet counters, CSR6 is still receiving traffic, but is now incurring RPF failures as well, this time discarding traffic from XRv3.

```
R6#show ip mroute 232.4.1.0 213.1.1.1 count | begin Group  
Group: 232.4.1.0, Source count: 1, Packets forwarded: 2058, Packets received: 2937  
Source: 213.1.1.1/32, Forwarding: 2058/1/142/1, Other: 2937/879/0
```

```
R6#show ip mroute 232.4.1.0 213.1.1.1 count | begin Group  
Group: 232.4.1.0, Source count: 1, Packets forwarded: 2068, Packets received: 2957  
Source: 213.1.1.1/32, Forwarding: 2068/1/142/1, Other: 2957/889/0
```

We can prove this further with EPC again. This time, the packets from CSR7 and XRv3 arrived at exactly the same time, but the one from CSR7 was forwarded to the customer.

```
R6#show monitor capture CAP buffer detailed  
18 142 607.249987 213.1.1.1 -> 232.4.1.0 GRE  
0000: 01005E04 01000050 56A9EA77 81000DEF ..^....PV..w....
```



```

0010: 08004500 007C083D 0000FD2F F60ED501  ..E...|. =.../....
0020: 0101E804 01000000 08004500 0064084A  .....E..d.J
0030: 0000FE01 86120A01 0909E82A 33080800  .....*3...

19  118  607.249987  10.1.9.9      -> 232.42.51.8      ICMP
0000: 01005E2A 33080050 56A9DE0D 8100DDA  ..^*3..PV.....
0010: 08004500 0064084A 0000FD01 87120A01  ..E..d.J.....
0020: 0909E82A 33080800 5BBC0006 043C0000  ...*3...[....<..
0030: 00003083 EDC8ABCD ABCDABCD ABCDABCD  ..0.....

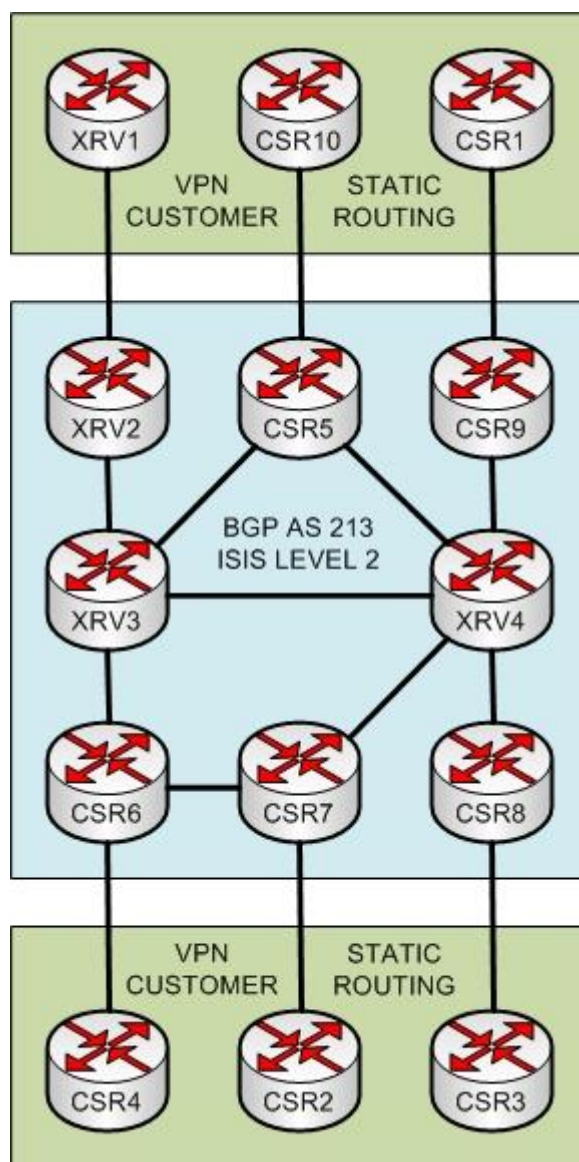
20  142  607.249987  213.1.1.1     -> 232.4.1.0      GRE
0000: 01005E04 01000050 56A9EA54 8100CDEB  ..^....PV..T....
0010: 08004500 007C083D 0000FD2F F60ED501  ..E...|. =.../....
0020: 0101E804 01000000 08004500 0064084A  .....E..d.J
0030: 0000FE01 86120A01 0909E82A 33080800  .....*3...

```

Additional Reading – Reference configurations “mvpn-mofrr”

29.3 Protecting mLDP LSPs with Fast Re-Route (FRR)

This section details how to provide FRR for mLDP traffic. For testing, we will use the same alternative MVPN network used in the MoFRR section which adds an additional link between CSR6 and CSR7 for redundancy. The base configuration is MVPN profile 1, which uses a MP2MP default tree, statically configured, with PIM C-multicast signaling.



Since mLDP does not have any inherent ability to be protected by TE (unlike P2MP RSVP-TE), it relies on existing TE-FRR mechanisms. Specifically, we can provide link protection only for mLDP LSM traffic in the network core using a combination of existing technologies. As examined in the TE section, we can use NHOP tunnels for link protection coupled with primary one-hop tunnels to reach each next-hop. The primary tunnels use auto-route announce and FRR, and the NHOP backup tunnels can protect these tunnels. mLDP can be configured to use these primary one-hop TE tunnels for recursive forwarding which are inherently protected by NHOP backups. This protects every link in the topology if configured everywhere. This can be done manually or automatically, but the combination of auto-primary and auto-backup tunnels suits this purpose perfectly. Those topics are covered in detail in the TE section so we configure them quickly on all LSRs in the topology. **Note: The feature does not appear to work well on XR. There is no obvious way to have auto-backups protect auto-primaries that request FRR, nor is there a way to force mLDP traffic into a TE tunnel.** We will limit the test to CSR6 and CSR7 instead.

```

! All XE LSRs
mpls mldp path traffic-eng
mpls ldp discovery targeted-hello accept
mpls traffic-eng auto-tunnel backup nhop-only
mpls traffic-eng auto-tunnel backup tunnel-num min 5000 max 5999
mpls traffic-eng auto-tunnel primary onehop
mpls traffic-eng auto-tunnel primary config mpls ip
mpls traffic-eng auto-tunnel primary tunnel-num min 6000 max 6999

! All XR LSRs
mpls ldp
  address-family ipv4
    discovery targeted-hello accept

```

The configuration isn't exactly straightforward. Enabling auto-backup and auto-primary tunnels is basic, with the exception that because these primary tunnels are one-hop, not PE-PE, we must enable LDP on them as well. This stitches the LSP together end to end. All routers are able to accept targeted sessions to support this, even the XR core routers. I explicitly used different number ranges for the backup and primary tunnels to make verification easier. The "magic" command is "mpls mldp path traffic-eng" on XE which allows the mLDP LSPs to use TE tunnels. A quick check on CSR6 shows us that the tunnels are up; there will be 2 primary tunnels to XRv3 and CSR7, and the backup tunnels routing around those protected links. This magic command also assumes the tunnels are LDP-enabled with autoroute announce configured on them, which is fitting for auto-primary tunnels.

```

R6#show mpls traffic-eng tunnels summary | begin auto
auto-tunnel:
  backup Enabled (2 ), id-range:5000-5999
  onehop Enabled (2 ), id-range:6000-6999
  mesh Disabled (0 ), id-range:64336-65335

```

To verify this further, we can look at the RSVP RESV messages to see the tunnels originating on CSR6. We can clearly see the direction each tunnel is going. The backup tunnels are highlighted in yellow with the primary tunnels in green.

```

R6#show ip rsvp reservation filter session-type 7 sender 213.6.6.6

```

Destination	Tun Sender	TunID	LSPID	Next Hop	I/F	Fi	Serv	BPS
213.7.7.7	213.6.6.6	5000	1	213.6.13.13	Gi2.563	SE	LOAD	0
213.7.7.7	213.6.6.6	6001	8053	213.6.7.7	Gi2.567	SE	LOAD	0
213.13.13.13	213.6.6.6	5001	1	213.6.7.7	Gi2.567	SE	LOAD	0
213.13.13.13	213.6.6.6	6000	928	213.6.13.13	Gi2.563	SE	LOAD	0

We can determine which primary tunnel is protected by which backup tunnel by checking the FRR database. Unlike previous examples, the FRR database on the PLR has no in-label since the PLR is the tunnel head. Tunnel6000 goes to XRv3 and is backed up by Tunnel5001 which also terminates there. We

can tell this is an NHOP tunnel as a result; we confirm this by looking at the RSVP RESV messages above to see the same tunnel tail-end. The same is true about the relationship between Tunnel6001 and Tunnel5000.

```
R6#show mpls traffic-eng fast-reroute database
P2P Headend FRR information:
Protected tunnel      In-label Out intf/label    FRR intf/label    Status
-----
Tunnel6000           Tun hd   Gi2.563:implicit  Tu5001:implicit-  ready
Tunnel6001           Tun hd   Gi2.567:implicit  Tu5000:implicit-  ready
```

Now that we have verified FRR was enabled on the primary tunnels, we can verify that autoroute announce was as well. We can see each tunnel with autoroute announce enabled with its specified tail-end.

```
R6#show mpls traffic-eng autoroute
MPLS TE autorouting enabled
  destination 0000.0000.0007.00, area isis level-2, has 1 tunnels
    Tunnel6001 (load balancing metric 0, nexthop 213.7.7.7)
      (flags: Announce)
  destination 0000.0000.0013.00, area isis level-2, has 1 tunnels
    Tunnel6000 (load balancing metric 0, nexthop 213.13.13.13)
      (flags: Announce)
```

Next, we will verify LDP is enabled on the tunnel and has established targeted sessions. This will allow the LDP label exchange to occur between the head and tail ends, which is important for tunnel stitching and described in the TE section. The reason CSR7's entry says active/passive is because CSR7 has also established a session to CSR6, but XRv3 has not established one to CSR6. The "xmit/recv" is what really matters for label exchange.

```
R6#show mpls ldp discovery | begin Targeted Hello
Targeted Hellos:
  213.6.6.6 -> 213.13.13.13 (ldp): active, xmit/recv
    LDP Id: 213.13.13.13:0
  213.6.6.6 -> 213.7.7.7 (ldp): active/passive, xmit/recv
    LDP Id: 213.7.7.7:0
```

This implies CSR6 can learn the labels towards any node beyond these primary tunnels, such as XRv14. We confirm this by checking the LDP bindings and ultimately, the FIB. These labels are stacked before the TE label (which is implicit-null on the primary tunnels) when sending traffic outbound. This is just a basic unicast check and isn't terribly relevant to mLDP.

```
R6#show mpls ldp bindings 213.14.14.14 32
lib entry: 213.14.14.14/32, rev 15
local binding: label: 6002
```

```
remote binding: lsr: 213.13.13.13:0, label: 93000
remote binding: lsr: 213.7.7.7:0, label: 7000
```

```
R6#show ip cef 213.14.14.14
213.14.14.14/32
  nexthop 213.7.7.7 Tunnel6001 label 7000
  nexthop 213.13.13.13 Tunnel6000 label 93000
```

Of greater importance is mLDP. We verify that CSR6 has joined the MP2MP default MDT, which is the result of having sent a label mapping message upstream. CSR6 informs XRv3 that it should receive traffic using label 6012, and XRv3 responds with an upstream mapping of 93009 which is used to send traffic towards the root. The next-hop interface is a TE tunnel which is not possible unless the router is explicitly told to consider TE tunnels as paths for mLDP (configured earlier).

```
R6#show mpls mldp database opaque_type mdt 213:213 0
LSM ID : 1 (RNR LSM ID: 2)   Type: MP2MP   Uptime : 01:20:04
  FEC Root      : 213.14.14.14
  Opaque decoded : [mdt 213:213 0]
  Opaque length  : 11 bytes
  Opaque value   : 02 000B 0002130000021300000000
  RNR active LSP : (this entry)
  Upstream client(s) :
    213.13.13.13:0   [Active]
      Expires        : Never          Path Set ID   : 1
      Out Label (U)   : 93009          Interface    : Tunnel6000*
      Local Label (D) : 6012           Next Hop      : 213.13.13.13
  Replication client(s):
    MDT (VRF MC)
      Uptime          : 01:20:04       Path Set ID   : 2
      Interface       : Lspvif0
```

A quick OAM check verifies that all of the other XE PE routers have configured this properly as well. XRv2 is not running TE with mLDP but can still communicate; we also check the PIM neighbors inside the VPN to prove that the C-PIM signaling works.

```
R6#ping mpls mldp mp2mp 213.14.14.14 mdt 213:213 0
mp2mp Root node addr 213.14.14.14
Opaque type MDT, oui:index 0x213:0213, mdtnum 0
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
  timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]

Request #1
! reply addr 213.8.14.8
! reply addr 213.5.14.5
! reply addr 213.12.13.12
```

```
! reply addr 213.1.14.1
! reply addr 213.7.14.7
```

Round-trip min/avg/max = 48/411/1676 ms

Received 5 replies

```
R6#show ip pim vrf MC neighbor | begin ^Neighbor
```

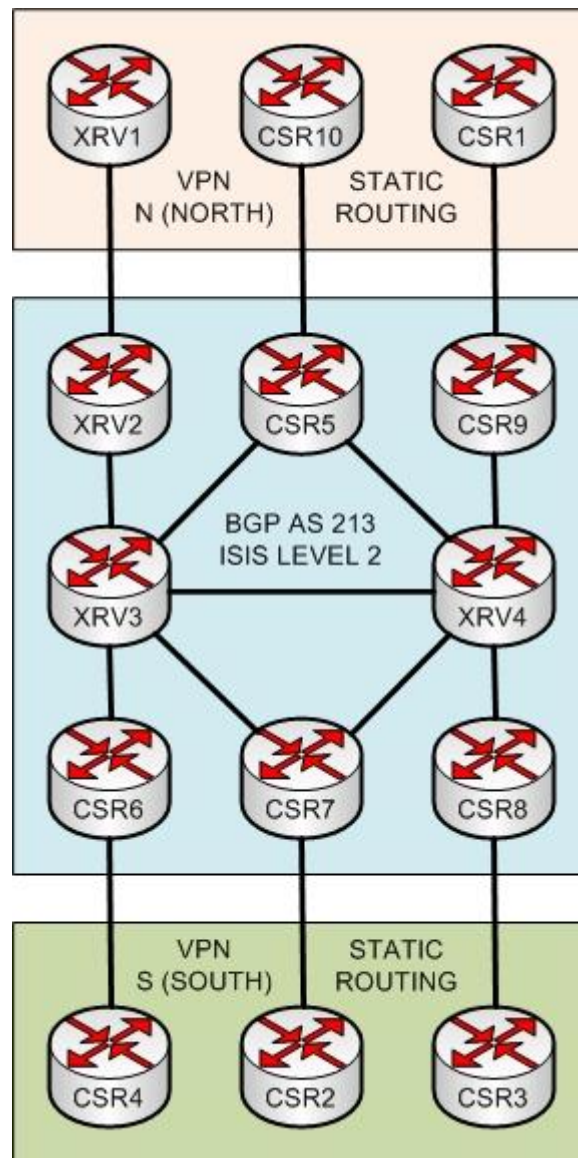
Neighbor Address	Interface	Uptime/Expires	Ver	DR Prio/Mode
213.7.7.7	Lspvif0	00:21:09/00:01:18	v2	1 / S P G
213.5.5.5	Lspvif0	00:21:09/00:01:44	v2	1 / S P G
213.1.1.1	Lspvif0	00:21:09/00:01:15	v2	1 / S P G
213.8.8.8	Lspvif0	00:21:09/00:01:43	v2	1 / S P G
213.12.12.12	Lspvif0	01:14:37/00:01:38	v2	1 / DR P G

At this point, all links are protected by NHOP tunnels using TE-FRR (assuming we configure auto-primary and auto-backup everywhere, which we didn't). Actually verifying this feature in the data-plane is difficult in this compressed topology, so it is skipped.

Additional Reading – Reference configurations “mvpn-mldp-frr”

29.4 MVPN Extranet

Extranets are defined as networks that have been extended outside of an intranet to provide reachability to external organizations. This could be a business partnership of sorts. According to senior engineers in Cisco, extranets are much more common in enterprise MPLS networks than in SP ones, but the concepts are the same in all environments. Building extranets for unicast connectivity with MPLS L3VPN is simple and is controlled entirely by the RT import/export policies. There are some additional considerations for multicast traffic. Specifically, an MVPN extranet exists when sources are in one VPN and receivers are in another VPN. Some draft RFCs (draft-rosen-l3vpn-mvpn-extranet-03) regarding MVPN extranet refer to these as “red” (sources) and “blue” (receivers) VPNs for simplicity. Those terms are used here also. The network diagram is shown below and is very similar to the previous MVPN examples except that the VPNs have been split in half.



The top half of the network (XRV2, CSR5, and CSR1) are in VPN “N”, meaning North. The bottom half (CSR6, CSR7, CSR8) are in VRF “S” for South. XRV1, CSR9, CSR4, and CSR3 are all requesting the same IPv4 and IPv6 SSM groups with sources spread across both VRFs. This configuration is described in detail later. The groups in use are as follows:

IPv4 SSM with North source: (10.5.10.10, 232.0.0.7)
 IPv4 SSM with South source: (10.2.7.2, 232.0.0.5)
 IPv6 SSM with North source: (2001:10:5:10::10, FF33::2)
 IPv6 SSM with South source: (2001:10:2:7::2, FF33::1)

29.4.1 PIM/GRE

Multicast extranets are achievable using the classic Draft-Rosen family of MDTs. However, newer MVPN profiles can take advantage of BGP-AD to simplify the extranet mechanisms, so this would be similar to profile 3. The draft specified earlier uses terms “red” and “blue” to specify the VPNs in which the sources and receivers lie, respectively. For our first test, CSR10 is the source and CSR6/CSR8 are the extranet receivers for (10.5.10.10, 232.0.0.7). Therefore, VRF N is the “red” VPN and VRF S is the “blue” VPN for this test. The colors of red and blue will swap as we test connectivity in multiple directions for multiple protocols, which is why the VRF names are not tied to colors. Before actually doing any extranet connectivity, we will verify that intranet multicast works within VRF N. XRv11 and CSR9 are both requesting traffic for the aforementioned flow, so they should be able to receive intranet multicast. We won’t verify every component of MVPN profile 3 again, but we will begin by checking the C-MRIB on all PEs. They all claim that a data MDT is in use, which means there should be an associated BGP Type-3 S-PMSI route for the C(S,G). The XE boxes show the ‘Y’ and ‘y’ flag to indicate reception from and transmission to a data MDT, respectively. We can also see the PMSI to C(S,G) binding.

```
R5#show ip mroute vrf N 232.0.0.7 10.5.10.10 verbose | begin \
(10.5.10.10, 232.0.0.7), 00:32:34/00:02:37, flags: sTYp
  Incoming interface: GigabitEthernet2.550, RPF nbr 10.5.10.10
  Outgoing interface list:
    Tunnel1, GRE MDT: 232.4.5.0 (data), Forward/Sparse, 00:43:52/00:02:37, p
```

```
R1#show ip mroute vrf N 232.0.0.7 10.5.10.10 | begin \
(10.5.10.10, 232.0.0.7), 13:55:34/00:02:50, flags: sTIY
  Incoming interface: Tunnel0, RPF nbr 213.5.5.5,
MDT: [213.5.5.5,232.4.5.0]/never
  Outgoing interface list:
    GigabitEthernet2.519, Forward/Sparse, 13:55:34/00:02:50
```

```
RP/0/0/CPU0:XRv2#show pim vrf N topology 232.0.0.7 10.5.10.10 | begin 232
(10.5.10.10,232.0.0.7)SPT SSM Up: 14:01:43
JP: Join(00:00:24) RPF: mdtN,213.5.5.5 Flags:
  GigabitEthernet0/0/0/0.512 14:01:43 fwd LI LH
```

Checking the MDT caches on each router, we can see that CSR5 advertises the data MDT (implicit from the BGP-AD routes), and the egress PEs install it.

```
R5#show ip pim vrf N mdt send
MDT-data send list for VRF: N
  (source, group)                                MDT-data group/num    ref_count
  (10.5.10.10, 232.0.0.7)                        232.4.5.0            1
```

```
R1#show ip pim vrf N mdt receive
Joined MDT-data [group/mdt number : source]    uptime/expires for VRF: N
[232.4.5.0 : 213.5.5.5] 00:47:23/stopped
```

```
RP/0/0/CPU0:XRv2#show pim vrf N mdt cache
Core Source      Cust (Source, Group)                                Core Data      Expires
```


213.5.5.5 (10.5.10.10, 232.0.0.7) 232.4.5.0 never

Checking the BGP MVPN information, we see that CSR5 has originated the Type-3 route to describe the availability of an S-PMSI. As long as the other PEs are importing RT:213:5, they will see this S-PMSI availability. Tunnel-type 3 means this is a PIM-SSM GRE, the root is 213.5.5.5, and the core group is 232.4.5.0. This is how others know about the data MDT when BGP-AD is used.

```
R5#show bgp ipv4 mvpn vrf N route-type 3 10.5.10.10 232.0.0.7 213.5.5.5
BGP routing table entry for [3][213:5][10.5.10.10][232.0.0.7][213.5.5.5]/22,
version 189
Paths: (1 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGp
peer)
  Advertised to update-groups:
    6
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (213.5.5.5)
      Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
      Community: no-export
      Extended Community: RT:213:5
      PMSI Attribute: Flags: 0x0, Tunnel type: 3, length 8, label: exp-null,
tunnel parameters: D505 0505 E804 0500
      rx pathid: 0, tx pathid: 0x0
```

When CSR10 starts sending traffic, we can see the packet counters increasing on all three PEs, indicating that traffic is being sent. The XRv only shows packets out, but it is working.

```
R5#show ip mroute vrf N 232.0.0.7 10.5.10.10 count | begin ^Group
Group: 232.0.0.7, Source count: 1, Packets forwarded: 187, Packets received:
187
  Source: 10.5.10.10/32, Forwarding: 187/1/118/0, Other: 187/0/0
```

```
R1#show ip mroute vrf N 232.0.0.7 10.5.10.10 count | begin ^Group
Group: 232.0.0.7, Source count: 1, Packets forwarded: 199, Packets received:
199
  Source: 10.5.10.10/32, Forwarding: 199/1/142/1, Other: 199/0/0
```

```
RP/0/0/CPU0:XRv2#show mfib vrf N route 232.0.0.7 10.5.10.10 | begin 232
(10.5.10.10,232.0.0.7),   Flags:
Up: 14:10:10
Last Used: never
SW Forwarding Counts: 0/228/22800
SW Replication Counts: 0/228/22800
SW Failure Counts: 0/0/0/0/0
mdtN Flags:  A MI, Up:00:53:21
GigabitEthernet0/0/0/0.512 Flags:  NS EG, Up:14:10:10
```

To implement the “red method” as defined by the draft, the blue PEs must import the RT’s for the red multicast sources and BGP-AD routes. The draft specifically states you must use a new RT for this, called the “violet” RT. This is a good practice because only a few VPN routes behind a red PE would be multicast sources, so it makes sense not to have all the blue PEs with receivers import all red RTs. On CSR5, we will solve this in two steps. First, an export-map is applied to CSR5 for IPv4 that matches 10.5.10.0/24 and adds a RT:213:500. CSR10’s loopback, 10.10.10.10/32, is not a multicast source and only has the basic blue RT:213:5.

```
! CSR5
ip prefix-list PL_EXTRANET_SOURCES seq 5 permit 10.5.10.0/24

route-map RM_EXTRANET_EXPORT permit 10
 match ip address prefix-list PL_EXTRANET_SOURCES
 set extcommunity rt 213:500 additive

vrf definition N
 address-family ipv4
  export map RM_EXTRANET_EXPORT
```

Assuming CSR8 imports this RT (configuration is applied for VRF S but not shown), we will have one-way unicast connectivity from CSR8 to CSR5. This is OK because we only need this routing information for RPF; we are not attempting to establish bidirectional unicast connectivity.

```
R8#show bgp vpnv4 unicast vrf S 10.5.10.0/24
BGP routing table entry for 213:8:10.5.10.0/24, version 136
Paths: (1 available, best #1, table S)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from 213:5:10.5.10.0/24 (global)
    213.5.5.5 (metric 20) (via default) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:213:5 RT:213:500
      Originator: 213.5.5.5, Cluster list: 213.12.12.12
      Connector Attribute: count=1
        type 1 len 12 value 213:5:213.5.5.5
      mpls labels in/out nolaabel/5002
      rx pathid: 0, tx pathid: 0x0
```

The problem is that the BGP-AD routes did not get this “violet” RT applied, so CSR8 does not know to import it. We can check XRv2, the RR, to confirm that the route only carries the blue RT:213:5.

```
RP/0/0/CPU0:XRv2#show bgp ipv4 mvpn rd 213:5 [1][213.5.5.5]/40 | include
Exten
      Extended community: RT:213:5
```

```
R8#show bgp ipv4 mvpn vrf S route-type 1 213.5.5.5
```

```
% Network not in table
```

CSR8 still has a semi-valid C(S,G) entry, but there is no big 'Y' to indicate reception from S-PMSI. This is going to lead to problems since the entry has a valid RPF interface but is not using the proper delivery mechanism. CSR8 does not report seeing any multicast traffic arriving for this C(S,G). Its MDT cache only has an entry for the south-intranet VPN flow.

```
R8#show ip mroute vrf S 232.0.0.7 10.5.10.10 count | begin ^Group
Group: 232.0.0.7, Source count: 1, Packets forwarded: 0, Packets received: 0
Source: 10.5.10.10/32, Forwarding: 0/0/0/0, Other: 0/0/0
```

```
R8#show ip pim vrf S mdt receive
Joined MDT-data [group/mdt number : source] uptime/expires for VRF: S
[232.4.7.0 : 213.7.7.7] 14:27:57/stopped
```

To fix this, we configure CSR5 to add the violet RT to its outgoing IPv4 MVPN routes. We can be indiscriminate with the route-map because if an extranet receiver wants multicast from a source behind CSR5, it needs to import the violet RT, so it need not be route/prefix specific.

```
! CSR5
route-map RM_MVPN_EXTRANET_RT permit 10
set extcommunity rt 213:500 additive

router bgp 213
address-family ipv4
neighbor 213.12.12.12 route-map RM_MVPN_EXTRANET_RT out
```

Next, we verify that CSR8 imports the BGP-AD routes from CSR5 and updates its C-MRIB entry via the MDT cache. Checking the packet counters, we can see traffic is flow successful.

```
R8#show bgp ipv4 mvpn vrf S route-type 3 10.5.10.10 232.0.0.7 213.5.5.5
BGP routing table entry for [3][213:8][10.5.10.10][232.0.0.7][213.5.5.5]/22,
version 195
Paths: (1 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGP
peer)
Not advertised to any peer
Refresh Epoch 1
Local, imported path from [3][213:5][10.5.10.10][232.0.0.7][213.5.5.5]/22
(global)
213.5.5.5 (metric 20) from 213.12.12.12 (213.12.12.12)
Origin incomplete, metric 0, localpref 100, valid, internal, best
Community: no-export
Extended Community: RT:213:5 RT:213:500
Originator: 213.5.5.5, Cluster list: 213.12.12.12
PMSI Attribute: Flags: 0x0, Tunnel type: 3, length 8, label: exp-null,
tunnel parameters: D505 0505 E804 0500
rx pathid: 0, tx pathid: 0x0
```

```

R8#show ip mroute vrf S 232.0.0.7 10.5.10.10 | begin \
(10.5.10.10, 232.0.0.7), 01:20:02/00:02:20, flags: sTIY
Incoming interface: Tunnell1, RPF nbr 213.5.5.5,
MDT:[213.5.5.5,232.4.5.0]/never
Outgoing interface list:
GigabitEthernet2.538, Forward/Sparse, 01:20:02/00:02:20

R8#show ip pim vrf S mdt receive
Joined MDT-data [group/mdt number : source] uptime/expires for VRF: S
[232.4.5.0 : 213.5.5.5] 00:01:06/stopped
[232.4.7.0 : 213.7.7.7] 14:30:41/stopped

R8#show ip mroute vrf S 232.0.0.7 10.5.10.10 count | begin ^Group
Group: 232.0.0.7, Source count: 1, Packets forwarded: 67, Packets received:
67
Source: 10.5.10.10/32, Forwarding: 67/1/142/1, Other: 67/0/0

```

As always, a quick EPC alleviates any doubt we have about forwarding. CSR8 shows GRE multicast arriving from CSR5's MDT source towards the data MDT group 232.4.5.0. CsR8 decapsulates it and forwards the ICMP multicast for (10.5.10.10, 232.0.0.7) towards the customer. The difference in size is exactly 24 (GRE header) and the timestamps are the same.

```

R8#show monitor capture CAP buffer detailed
 2  142    1.002991    213.5.5.5        -> 232.4.5.0        GRE
0000:  01005E04 05000050 56A9862A 8100CE00  ..^....PV...*....
0010:  08004500 007C02D1 0000FE2F F272D505  ..E..|...../.r..
0020:  0505E804 05000000 08004500 006402DF  .....E..d..
0030:  0000FE01 BDA30A05 0A0AE800 00070800  .....

 3  118    1.002991    10.5.10.10        -> 232.0.0.7        ICMP
0000:  01005E00 00070050 56A9FB1C 8100DD2  ..^....PV.....
0010:  08004500 006402DF 0000FD01 BEA30A05  ..E..d.....
0020:  0A0AE800 00070800 3FBA000A 01C40000  .....?.....
0030:  00003723 059FABCD ABCDABCD ABCDABCD  ..7#.....

```

Let's try to bend the rules by importing the non-violet RT on CSR6. That is, we will import RT:215:5 and not the violet target of RT:213:500. This will give CSR6 reachability to 10.10.10.10/32, which isn't really relevant, but it's sloppy. The draft clearly states: *"If the route matching S is a blue route (i.e., carries the blue RT but not the violet RT), then a Join is sent over the blue default PMSI. However, if the route matching S is a violet route (i.e., carries the violet RT), a Join is sent over the red default PMSI."* This would imply that an RPF route with only RT:213:5 would have its PIM join sent via the blue VPN. The keyword is "default MDT", which we are not using yet. We will test I-PMSI later. Once CSR6 imports RT213:5, everything literally just works. This RT captures both the unicast route to the source, 10.5.10.0/24, and the BGP-AD S-PMSI route for this C(S,G) in question. All of the same verifications we

did for CSR8 are shown below on CSR6 to confirm proper operation. This mechanism would not require the route-maps on CSR5 to add RT:213:500 to multiple prefixes.

```
R6#show bgp ipv4 mvpn vrf S route-type 3 10.5.10.10 232.0.0.7 213.5.5.5
BGP routing table entry for [3][213:6][10.5.10.10][232.0.0.7][213.5.5.5]/22,
version 199
Paths: (1 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGp
peer)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from [3][213:5][10.5.10.10][232.0.0.7][213.5.5.5]/22
(global)
    213.5.5.5 (metric 20) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Community: no-export
      Extended Community: RT:213:5 RT:213:500
      Originator: 213.5.5.5, Cluster list: 213.12.12.12
      PMSI Attribute: Flags: 0x0, Tunnel type: 3, length 8, label: exp-null,
tunnel parameters: D505 0505 E804 0500
      rx pathid: 0, tx pathid: 0x0
```

```
R6#show ip mroute vrf S 232.0.0.7 10.5.10.10 | begin \
(10.5.10.10, 232.0.0.7), 01:29:14/00:02:59, flags: sTIY
  Incoming interface: Tunnel0, RPF nbr 213.5.5.5,
MDT: [213.5.5.5,232.4.5.0]/never
  Outgoing interface list:
    GigabitEthernet2.546, Forward/Sparse, 01:29:14/00:02:59
```

```
R6#show ip pim vrf S mdt receive
Joined MDT-data [group/mdt number : source]  uptime/expires for VRF: S
[232.4.5.0 : 213.5.5.5]  01:07:24/stopped
[232.4.7.0 : 213.7.7.7]  14:40:22/stopped
```

```
R6#show ip mroute vrf S 232.0.0.7 10.5.10.10 count | begin ^Group
Group: 232.0.0.7, Source count: 1, Packets forwarded: 939, Packets received:
939
  Source: 10.5.10.10/32, Forwarding: 939/1/142/1, Other: 939/0/0
```

```
R6#show monitor capture CAP buffer detailed
  0  142    0.000000   213.5.5.5      -> 232.4.5.0      GRE
0000:  01005E04 05000050 56A9EA54 8100CDEB  ..^....PV..T....
0010:  08004500 007C049A 0000FE2F F0A9D505  ..E..|...../....
0020:  0505E804 05000000 08004500 006404A8  .....E..d..
0030:  0000FE01 BBDA0A05 0A0AE800 00070800  .....

  1  118    0.000000   10.5.10.10    -> 232.0.0.7      ICMP
0000:  01005E00 00070050 56A9DE0D 8100DDA  ..^....PV.....
0010:  08004500 006404A8 0000FD01 BCDA0A05  ..E..d.....
0020:  0A0AE800 00070800 42FC000A 038D0000  .....B.....
```

```
0030: 0000372A 008DABCD ABCDABCD ABCDABCD ..7*.....
```

To add complexity, we will remove this C(S,G) from being candidate for data MDT optimization. We do this by removing it from the IPv4 ACL on CSR5 so that it can no longer originate the Type-3 route. Because our ACL is only one line long, removing that one line turns the ACL into a “permit ip any any”. Instead, I will put a “deny ip any any” at the top. The BGP debugs clearly show this S-PMSI advertisement being withdrawn.

```
R5#show access-lists ACL_DATA_MDT
Extended IP access list ACL_DATA_MDT
    5 deny ip any any (3 matches)
    10 permit ip any host 232.0.0.7

! CSR5
BGP: MVPN(15) deleting the local route
[3][213:5][10.5.10.10][232.0.0.7][213.5.5.5]/22
BGP(15): no valid path for [3][213:5][10.5.10.10][232.0.0.7][213.5.5.5]/22
BGP(15): nettable_walker [3][213:5][10.5.10.10][232.0.0.7][213.5.5.5]/22 no
best path
BGP(15): delete RIB route [3][213:5][10.5.10.10][232.0.0.7][213.5.5.5]/22
BGP(15): (base) 213.12.12.12 send unreachable (format)
[3][213:5][10.5.10.10][232.0.0.7][213.5.5.5]/22
BGP(15): 213.12.12.12 rcv UPDATE about
[3][213:5][10.5.10.10][232.0.0.7][213.5.5.5]/22 -- withdrawn
```

A spot check on some interested PEs, intranet and extranet, shows the big ‘Y’ missing. Also, on the ingress PE CSR5, the little ‘y’ is missing. These groups will be using the I-PMSI as there is no selective P-tunnel available in the network for this C(S,G).

```
R5#show ip mroute vrf N 232.0.0.7 10.5.10.10 | begin \
(10.5.10.10, 232.0.0.7), 00:22:13/00:03:11, flags: sT
  Incoming interface: GigabitEthernet2.550, RPF nbr 10.5.10.10
  Outgoing interface list:
    Tunnell1, Forward/Sparse, 01:26:35/00:03:11
```

```
R1#show ip mroute vrf N 232.0.0.7 10.5.10.10 | begin \
(10.5.10.10, 232.0.0.7), 14:37:18/00:02:06, flags: sTI
  Incoming interface: Tunnel0, RPF nbr 213.5.5.5
  Outgoing interface list:
    GigabitEthernet2.519, Forward/Sparse, 14:37:18/00:02:06
```

```
R8#show ip mroute vrf S 232.0.0.7 10.5.10.10 | begin \
(10.5.10.10, 232.0.0.7), 01:36:44/00:02:38, flags: sTI
  Incoming interface: Tunnell1, RPF nbr 213.5.5.5
  Outgoing interface list:
    GigabitEthernet2.538, Forward/Sparse, 01:36:44/00:02:38
```

Another spot check on those same routers shows the MDT cache missing an entry for (10.5.10.10, 232.0.0.7). This is expected since that MDT cache information was gleaned from the Type-3 route that was just withdrawn. CSR8 still has its intranet SSM data MDT which is not being examined right now.

```
R1#show ip pim vrf N mdt receive
```

```
[no output]
```

```
R8#show ip pim vrf S mdt receive
```

```
Joined MDT-data [group/mdt number : source]  uptime/expires for VRF: S
[232.4.7.0 : 213.7.7.7]  14:50:03/stopped
```

```
R5#show ip pim vrf N mdt send
```

```
[no output]
```

Because we did not modify any RT policies, CSR6 is still importing RT:213:5 and CSR8 is still importing RT:213:500, which means both the BGP-AD and unicast extranet sources are inside VRF S (blue VPN). Looking at the I-PMSI BGP-AD routes, we can clearly see the default MDT groups are different. VRF S uses 232.255.0.2 while VRF N uses 232.255.0.1. There is no communication between VPNs over the default MDT, which means no PIM neighbors, etc. The draft clearly states: *"If MI-PMSIs are being used, the blue VRFs must immediately join the P-tunnels specified in the red I-PMSI A-D routes.* This is not happening. Despite CSR8 having imported CSR5's I-PMSI route, it did not join that MDT. I believe this could work with other MVPN profiles (like mLDP), but not GRE. We can immediately see that, when S-PMSI is not used, the complexity of MVPN extranet increases significantly.

```
R8#show bgp ipv4 mvpn vrf S route-type 1 213.5.5.5
```

```
BGP routing table entry for [1][213:8][213.5.5.5]/12, version 194
```

```
Paths: (1 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGp peer)
```

```
Not advertised to any peer
```

```
Refresh Epoch 1
```

```
Local, imported path from [1][213:5][213.5.5.5]/12 (global)
```

```
213.5.5.5 (metric 20) from 213.12.12.12 (213.12.12.12)
```

```
Origin incomplete, metric 0, localpref 100, valid, internal, best
```

```
Community: no-export
```

```
Extended Community: RT:213:5 RT:213:500
```

```
Originator: 213.5.5.5, Cluster list: 213.12.12.12
```

```
PMSI Attribute: Flags: 0x0, Tunnel type: 3, length 8, label: exp-null,
```

```
tunnel parameters: D505 0505 E8FF 0001
```

```
rx pathid: 0, tx pathid: 0x0
```

```
R8#show ip igmp groups
```

```
IGMP Connected Group Membership
```

Group Address	Interface	Uptime	Expires	Last Reporter
232.4.7.0	Loopback0	15:09:38	stopped	213.8.8.8
232.255.0.2	Loopback0	15:17:02	stopped	213.8.8.8
224.0.1.40	Loopback0	1d16h	00:02:04	213.8.8.8

```
R8#show ip mroute 232.255.0.1
Group 232.255.0.1 not found
```

There are two solutions for this which have been supported by Cisco routers since 12.2S.

- a. **One option is to configure the source VRF on the receiver PEs, which is known as receive side chaining (RSC).** This allows the “red” sources to have traffic follow the red default MDT all the way to the egress PE, because the egress PE is actually in that VPN now. This will allow the extranet receivers to join the red default MDT. This is best used when the flows are high bandwidth and/or there are few receivers. It is my personal favorite option.
- b. **The second option is to configure the receiver VRF on the source PEs, which is source side chaining (SSC).** This allows the ingress PE to perform ingress replication so that the red multicast traffic is sent as both red and blue. This means that the receiver PEs don’t need to add more VRF configurations. This is best used when the flows are low bandwidth (extra replication) or there are many receivers where RSC scales poorly, as everything egress PE would need to join the red default MDT.

First, I will use the RSC solution. It’s somewhat similar to the “red method” defined in the draft as we are modifying the blue VPN to receive multicast differently. This is quite configuration intensive compared to the S-PMSI solution and requires a lot of verification. The configuration modifications on CSR8 are shown below. We define the “red” VRF containing sources on the receiver PE, which is blue. It imports the routes from CSR5 using the violet RT. The configuration guide says that RT export should also be configured on this additional VRF, but it is not necessary. Another unnecessary step in the guide is importing the red sources into VRF S directly; since VRF S is explicitly using VRF N for RPF, there is no reason to import those routes. We use group-based VRF selection so that VRF S (blue) knows the look in VRF N (red) for sources.

```
! CSR8
vrf definition N
 rd 213:88
 route-target import 213:500
 address-family ipv4
  mdt auto-discovery pim
  mdt default 232.255.0.1

ip multicast-routing vrf N distributed
ip pim vrf N ssm default

ip access-list standard ACL_N_GROUPS
 permit 232.0.0.7

ip multicast vrf S rpf select vrf N group-list ACL_N_GROUPS
```


One interesting thing to note is that we have a unidirectional PIM neighbor over the red default MDT. CSR8 has imported CSR5's RTs but not vice versa, so the PIM hellos only go one way. This isn't terribly relevant, but an interesting note and a quick way to check if the receiver successfully joined the source's default MDT.

```
R5#show ip pim vrf N neighbor | begin ^Neighbor
Neighbor          Interface          Uptime/Expires      Ver   DR
Address
213.1.1.1          Tunnel1          00:06:15/00:01:24  v2    1 / B S P G
213.12.12.12       Tunnel1          16:41:47/00:01:27  v2    1 / DR B P G
```

```
R8#show ip pim vrf N neighbor | begin ^Neighbor
Neighbor          Interface          Uptime/Expires      Ver   DR
Address
213.5.5.5          Tunnel4          00:06:54/00:01:43  v2    1 / B S P G
```

We can verify the group-based VRF selection policy is configured correctly also.

```
R8#show ip rpf vrf S select
Multicast Group-to-Vrf Mappings
Group(s): 232.0.0.7/32, RPF vrf: N, Acl: ACL_N_GROUPS
```

Next, we step through the three major MRIB entries that build this extranet connectivity. First, we need to verify that the blue PE has joined the red default MDT properly. The 'Z' flag must be set so the router identifies this as a multicast tunnel. The OIL should be directing traffic into the red VRF.

```
R8#show ip mroute 232.255.0.1 213.5.5.5 | begin \
(213.5.5.5, 232.255.0.1), 00:05:30/stopped, flags: sTIZ
Incoming interface: GigabitEthernet2.584, RPF nbr 213.8.14.14
Outgoing interface list:
MVRF N, Forward/Sparse, 00:05:30/00:00:29
```

Next, the red VRF indicates that this entry is an "extranet" entry with the big 'E'. This is a proxy mechanism in the control plane that essentially says "the real receivers are in the blue VRF". Notice that the OIL is null since there are no real receivers in the red VPN from the perspective of the blue PE.

```
R8#show ip mroute vrf N 232.0.0.7 10.5.10.10 | begin \
(10.5.10.10, 232.0.0.7), 00:05:51/stopped, flags: sTE
Incoming interface: Tunnel4, RPF nbr 213.5.5.5
Outgoing interface list: Null
Extranet receivers in vrf S:
(10.5.10.10, 232.0.0.7), 00:07:19/00:02:39, OIF count: 1, flags: sTI
```

VRF S, the blue VPN, has a normal C-MRIB entry except for the RPF lookup uses the red VPN. This makes sense since the blue VPN has no unicast reachability to the red sources. This is a little different than the S-PMSI tests earlier where importing the violet RT actually provided one-way unicast reachability.

```
R8#show ip mroute vrf S 232.0.0.7 10.5.10.10 | begin \  
(10.5.10.10, 232.0.0.7), 00:07:31/00:02:27, flags: sTI  
Incoming interface: Tunnel4, RPF nbr 213.5.5.5, using vrf N  
Outgoing interface list:  
GigabitEthernet2.538, Forward/Sparse, 00:07:31/00:02:27
```

```
R8#show ip rpf vrf S 10.5.10.0  
failed, no route exists
```

```
R8#show ip rpf vrf N 10.5.10.0  
RPF information for ? (10.5.10.0)  
RPF interface: Tunnel4  
RPF neighbor: ? (213.5.5.5)  
RPF route/mask: 10.5.10.0/24  
RPF type: unicast (bgp 213)  
Doing distance-preferred lookups across tables  
BGP originator: 213.5.5.5  
RPF topology: ipv4 multicast base, originated from ipv4 unicast base
```

Checking the counters on all three MRIB entries shows that traffic is being piped from the global table (red default MDT) into the red VPN, then into the blue VPN. It's expected that the default MDT will have a few more packets since it carries additional control-plane traffic, and potential other data flows. The red and blue VPN counters should be identical or very close.

```
R8#show ip mroute 232.255.0.1 213.5.5.5 count | begin ^Group  
Group: 232.255.0.1, Source count: 1, Packets forwarded: 688, Packets  
received: 688  
Source: 213.5.5.5/32, Forwarding: 688/1/138/1, Other: 688/0/0
```

```
R8#show ip mroute vrf N 232.0.0.7 10.5.10.10 count | begin ^Group  
Group: 232.0.0.7, Source count: 1, Packets forwarded: 596, Packets received:  
596  
Source: 10.5.10.10/32, Forwarding: 596/1/142/1, Other: 596/0/0
```

```
R8#show ip mroute vrf S 232.0.0.7 10.5.10.10 count | begin ^Group  
Group: 232.0.0.7, Source count: 1, Packets forwarded: 596, Packets received:  
596  
Source: 10.5.10.10/32, Forwarding: 596/1/142/1, Other: 596/0/0
```

An EPC verification proves that traffic is being forwarded correctly. Traffic traverses the red default MDT.

```
R8#show monitor capture CAP buffer detailed
```

```

 6 142      1.593023    213.5.5.5      -> 232.255.0.1      GRE
0000: 01005E7F 00010050 56A9862A 8100CE00  ..^....PV...*....
0010: 08004500 007C1453 0000FE2F E4F4D505  ..E...|.S.../....
0020: 0505E8FF 00010000 08004500 00641461  .....E..d.a
0030: 0000FE01 AC210A05 0A0AE800 00070800  .....!.....

 7 118      1.593023    10.5.10.10     -> 232.0.0.7      ICMP
0000: 01005E00 00070050 56A9FB1C 8100DD2  ..^....PV.....
0010: 08004500 00641461 0000FD01 AD210A05  ..E..d.a.....!..
0020: 0A0AE800 00070800 CDB1000E 02950000  .....
0030: 00003784 7671ABCD ABCDABCD ABCDABCD  ..7.vq.....

```

We will quickly re-test this on CSR6 for completeness. The downside of the RSC mechanism is that you must configure it on all receivers, so we have to do twice the work. We will use RT:213:5, the real blue VPN RT, versus the violet RT:213:500. It should work the same way since draft RFC suggesting the use of “violet RT” was written long after this feature was introduced. The snippet for CSR6 is not shown here since it is the same as CSR8’s with the RD changing to 213:66 and the RT-import policy using RT:213:5. Remember that CSR6’s blue VPN (S) is not importing any new RTs. First, we verify the three MRIB entries: red default MDT, red VPN, and blue VPN. This shows the control-plane linkages between them.

```

R6#show ip mroute 232.255.0.1 213.5.5.5 | begin \
(213.5.5.5, 232.255.0.1), 00:01:32/00:01:58, flags: sTIZ
  Incoming interface: GigabitEthernet2.563, RPF nbr 213.6.13.13
  Outgoing interface list:
    MVRF N, Forward/Sparse, 00:01:32/00:01:58

R6#show ip mroute vrf N 232.0.0.7 10.5.10.10 | begin \
(10.5.10.10, 232.0.0.7), 00:01:19/00:01:40, flags: sTE
  Incoming interface: Tunnel3, RPF nbr 213.5.5.5
  Outgoing interface list: Null
  Extranet receivers in vrf S:
    (10.5.10.10, 232.0.0.7), 03:48:25/00:02:55, OIF count: 1, flags: sTI

R6#show ip mroute vrf S 232.0.0.7 10.5.10.10 | begin \
(10.5.10.10, 232.0.0.7), 03:48:29/00:02:52, flags: sTI
  Incoming interface: Tunnel3, RPF nbr 213.5.5.5, using vrf N
  Outgoing interface list:
    GigabitEthernet2.546, Forward/Sparse, 03:48:29/00:02:52

```

Next, we verify the packet counters for each of these three entries. Notice the red/blue VPNs have the same number of packets, since the blue VPN is just a proxy mechanism.

```

R6#show ip mroute 232.255.0.1 213.5.5.5 count | begin ^Group
Group: 232.255.0.1, Source count: 1, Packets forwarded: 422, Packets
received: 422
  Source: 213.5.5.5/32, Forwarding: 422/1/139/1, Other: 422/0/0

```

```
R6#show ip mroute vrf N 232.0.0.7 10.5.10.10 count | begin ^Group
Group: 232.0.0.7, Source count: 1, Packets forwarded: 38, Packets received:
38
```

```
Source: 10.5.10.10/32, Forwarding: 38/1/142/1, Other: 38/0/0
```

```
R6#show ip mroute vrf S 232.0.0.7 10.5.10.10 count | begin ^Group
Group: 232.0.0.7, Source count: 1, Packets forwarded: 38, Packets received:
38
```

```
Source: 10.5.10.10/32, Forwarding: 38/1/142/1, Other: 38/0/0
```

EPC verifies that the traffic was received from the core and sent to the customer successfully.

```
R6#show monitor capture CAP buffer detailed
```

```
1 142 0.005997 213.5.5.5 -> 232.255.0.1 GRE
0000: 01005E7F 00010050 56A9EA54 8100CDEB ..^....PV..T....
0010: 08004500 007C1DB7 0000FE2F DB90D505 ..E..|...../....
0020: 0505E8FF 00010000 08004500 00641DC5 .....E..d..
0030: 0000FE01 A2BD0A05 0A0AE800 00070800 .....
```

```
2 118 0.005997 10.5.10.10 -> 232.0.0.7 ICMP
0000: 01005E00 00070050 56A9DE0D 8100DDA ..^....PV.....
0010: 08004500 00641DC5 0000FD01 A3BD0A05 ..E..d.....
0020: 0A0AE800 00070800 13E90010 06F00000 .....
0030: 000037AD 2BB4ABCD ABCDABCD ABCDABCD ..7.+.....
```

We will quickly test the same scenario with IPv6. We can re-use the BGP route-map to add the violet RT (RT:213:500) to the IPv6 MVPN AD routes on CSR5. We will create a new route-map with an IPv6 prefix-list to match the extranet source (2001:10:5:10::/64). We can locally verify the unicast prefix's RT addition, but must check the RR for see the IPv6 MVPN RT addition. Both are correct.

```
R5#show bgp vpnv6 unicast vrf N 2001:10:5:10::/64
```

```
[snip]
```

```
Local
```

```
:: (via vrf N) from 0.0.0.0 (213.5.5.5)
```

```
Origin incomplete, metric 0, localpref 100, weight 32768, valid,
sourced, best
```

```
Extended Community: RT:213:5 RT:213:500
```

```
mpls labels in/out 5009/nolabel(N)
```

```
rx pathid: 0, tx pathid: 0x0
```

```
RP/0/0/CPU0:XRv2#show bgp ipv6 mvpn rd 213:5 [1][213.5.5.5]/40
```

```
[snip]
```

```
Local, (Received from a RR-client)
```

```
213.5.5.5 (metric 20) from 213.5.5.5 (213.5.5.5)
```

```
Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, not-in-vrf
```

```
Received Path ID 0, Local Path ID 1, version 149
```

```
Community: no-export
```

```
Extended community: RT:213:5 RT:213:500
PMSI: flags 0x00, type 3, label 0, ID 0xd5050505e8ff0001
```

We also configure CSR6 and CSR8 with the RPF selector to allow any source sending to group FF33::2 inside the blue VPN (S) to be RPF-checked using the red VPN (N). Remember that CSR6 is importing RT:213:5 (blue RT) and CSR8 is importing RT:213:500 (violet RT). Both should work as they did for IPv4.

```
! CSR6 and CSR8
ipv6 multicast vrf S rpf select vrf N group-range ACL_N_GROUPS_IPV6
ipv6 access-list ACL_N_GROUPS_IPV6
permit ipv6 any host FF33::2
```

First, we will verify CSR8's MRIB entries in sequence. Traffic is received from the core on the red VPN and passed to the red proxy VRF. This passes the traffic into the blue VPN for delivery to the customer.

```
R8#show ip mroute 232.255.0.1 213.5.5.5 | begin \
(213.5.5.5, 232.255.0.1), 00:45:48/stopped, flags: sTIZ
Incoming interface: GigabitEthernet2.584, RPF nbr 213.8.14.14
Outgoing interface list:
MVRF N, Forward/Sparse, 00:45:48/00:02:11
```

```
R8#show ipv6 mroute vrf N FF33::2 2001:10:5:10::10 | begin \
(2001:10:5:10::10, FF33::2), 00:10:04/never, flags: sTE
Incoming interface: Tunnel4
RPF nbr: ::FFFF:213.5.5.5
Outgoing interface list: Null
Extranet receivers in vrf S, OIF count 1
```

```
R8#show ipv6 mroute vrf S FF33::2 2001:10:5:10::10 | begin \
(2001:10:5:10::10, FF33::2), 03:38:03/never, flags: sTI
Incoming interface: Tunnel4
RPF nbr: ::FFFF:213.5.5.5, using vrf N
Immediate Outgoing interface list:
GigabitEthernet2.538, Forward, 03:38:03/never
```

Because the blue VPN has no unicast route back to the source, our group-based VRF selection policy can satisfy RPF. This mechanism is very clean since the routes remain within the red (source) VPN.

```
R8#show ipv6 route vrf S 2001:10:5:10::/64
% Route not found
```

```
R8#show ipv6 cef vrf S 2001:10:5:10::10
::/0
no route
```

```
R8#show ipv6 rpf vrf S select
Multicast Group-to-Vrf Mappings
```

Group(s): FF33::2/128, RPF vrf: N, Acl: ACL_N_GROUPS_IPV6

We send traffic from CSR10 and check the packet counters to see traffic being forwarded. EPC verifies that the IPv4 GRE traffic is decapsulated to reveal the IPv6-ICMP multicast inside.

```
R8#show ip mroute 232.255.0.1 213.5.5.5 count | begin ^Group
Group: 232.255.0.1, Source count: 1, Packets forwarded: 262, Packets
received: 262
Source: 213.5.5.5/32, Forwarding: 262/1/115/1, Other: 262/0/0
```

```
R8#show ipv6 mroute vrf N FF33::2 2001:10:5:10::10 count | begin ^Group
Group: FF33::2
Source: 2001:10:5:10::10,
SW Forwarding: 0/0/0/0, Other: 0/0/0
HW Forwarding: 39/1/142/1, Other: 0/0/0
Totals - Source count: 1, Packet count: 39
```

```
R8#show ipv6 mroute vrf S FF33::2 2001:10:5:10::10 count | begin ^Group
Group: FF33::2
Source: 2001:10:5:10::10,
SW Forwarding: 0/0/0/0, Other: 0/0/0
HW Forwarding: 39/1/142/1, Other: 0/0/0
Totals - Source count: 1, Packet count: 39
```

```
R8#show monitor cap CAP buffer detailed
 2 142    0.723006 213.5.5.5    -> 232.255.0.1    GRE
0000: 01005E7F 00010050 56A9862A 8100CE00    ..^....PV..*....
0010: 08004500 007C003A 0000FE2F F90DD505    ..E...|:.... /....
0020: 0505E8FF 00010000 86DD6000 0000003C    .....`....<
0030: 3A3F2001 00100005 00100000 00000000    :? .....

 3 118    0.724013 2001::*:0010    -> FF33::*:0002    IPv6-ICMP
0000: 33330000 00020050 56A9FB1C 8100DD2    33.....PV.....
0010: 86DD6000 0000003C 3A3F2001 00100005    ..`....<:? .....
0020: 00100000 00000000 0010FF33 00000000    .....3.....
0030: 00000000 00000000 00028000 C3B4259E    .....%.
```

To prove that the violet RT:213:500 isn't really needed when doing RSC in this way, we will verify CSR6 as well. First, we examine the three MRIB entries. Packets arrive from the core multicast tunnel (big 'Z'), then get passed to the red VPN (N) with extranet receivers (big 'E'), then passed to get blue VPN (S).

```
R6#show ip mroute 232.255.0.1 213.5.5.5 | begin \
(213.5.5.5, 232.255.0.1), 00:54:37/stopped, flags: sTIZ
Incoming interface: GigabitEthernet2.563, RPF nbr 213.6.13.13
Outgoing interface list:
MVRF N, Forward/Sparse, 00:54:37/00:02:22
```

```
R6#show ipv6 mroute vrf N FF33::2 2001:10:5:10::10 | begin \
```

```
(2001:10:5:10::10, FF33::2), 00:16:51/never, flags: sTE
Incoming interface: Tunnel3
RPF nbr: ::FFFF:213.5.5.5
Outgoing interface list: Null
Extranet receivers in vrf S, OIF count 1

R6#show ipv6 mroute vrf S FF33::2 2001:10:5:10::10 | begin \
(2001:10:5:10::10, FF33::2), 06:50:42/never, flags: sTI
Incoming interface: Tunnel3
RPF nbr: ::FFFF:213.5.5.5,using vrf N
Immediate Outgoing interface list:
GigabitEthernet2.546, Forward, 06:50:42/never
```

The blue VPN traffic passes RPF due to our group-to-VRF mappings. The red VPN is used for the RPF lookup.

```
R6#show ipv6 route vrf S 2001:10:5:10::/64
% Route not found

R6#show ipv6 cef vrf S 2001:10:5:10::10
::/0
no route

R6#show ipv6 rpf vrf S select
Multicast Group-to-Vrf Mappings
Group(s): FF33::2/128, RPF vrf: N, Acl: ACL_N_GROUPS_IPV6
```

With traffic still flowing from CSR10, we check the packet counters to ensure we see packets flowing from the core, through the proxy extranet VPN, into the customer VPN.

```
R6#show ip mroute 232.255.0.1 213.5.5.5 count | begin ^Group
Group: 232.255.0.1, Source count: 1, Packets forwarded: 1732, Packets
received: 1732
Source: 213.5.5.5/32, Forwarding: 1732/1/125/1, Other: 1732/0/0

R6#show ipv6 mroute vrf N FF33::2 2001:10:5:10::10 count | begin ^Group
Group: FF33::2
Source: 2001:10:5:10::10,
SW Forwarding: 0/0/0/0, Other: 0/0/0
HW Forwarding: 269/1/142/1, Other: 0/0/0
Totals - Source count: 1, Packet count: 269

R6#show ipv6 mroute vrf S FF33::2 2001:10:5:10::10 count | begin ^Group
Group: FF33::2
Source: 2001:10:5:10::10,
SW Forwarding: 0/0/0/0, Other: 0/0/0
HW Forwarding: 269/1/142/1, Other: 0/0/0
Totals - Source count: 1, Packet count: 269
```

Finally, EPC confirms proper operation on CSR6 also. We will use the brief output to verify.

```
R6#show mon cap CAP buffer brief
 1 142    0.498027 213.5.5.5      -> 232.255.0.1    GRE
 2 118    0.498027 2001:::0010      -> FF33:::0002    IPv6-ICMP
```

Next, we will test the SSC extranet delivery option. We will use VRF S as the red VPN where CSR2 is the source and XRv1/CSR9 are the receivers. We already know that if we use S-PMSI, things will “just work” as long as we import the proper RTs on the receiver PE's since the data MDTs aren't VPN-specific. Therefore, we will test using I-PMSI only using SSC. The benefit of this approach is that we only have to configure a new VRF on CSR7 (one ingress PE router). The drawback is that multicast will be replicated for each VPN, wasting bandwidth in the core. CSR7 is currently offering an S-PMSI P-tunnel for (10.2.7.2, 232.0.0.5).

```
R7#show bgp ipv4 mvpn vrf S route-type 3 10.2.7.2 232.0.0.5 213.7.7.7
BGP routing table entry for [3] [213:7] [10.2.7.2] [232.0.0.5] [213.7.7.7]/22,
version 200
Paths: (1 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGp
peer)
  Advertised to update-groups:
    5
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (213.7.7.7)
      Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
      Community: no-export
      Extended Community: RT:213:7
      PMSI Attribute: Flags: 0x0, Tunnel type: 3, length 8, label: exp-null,
tunnel parameters: D507 0707 E804 0700
      rx pathid: 0, tx pathid: 0x0
```

```
R7#show ip pim vrf S mdt send
MDT-data send list for VRF: S
  (source, group)                                MDT-data group/num  ref_count
  (10.2.7.2, 232.0.0.5)                          232.4.7.0          1
```

The first thing we do on CSR7 is insert a “deny ip any any” at the top of its IPv4 data MDT ACL. This will ensure only I-PMSI can be used and also allows us to revert back to S-PMSI quickly if needed. A quick verification on CSR7 shows that the S-PMSI advertisement is gone. Again, when S-PMSI is used, extranet becomes easy and “just works” since the default MDTs don't need to pseudo-merge.

```
R7#show bgp ipv4 mvpn vrf S route-type 3 10.2.7.2 222.0.0.5 213.7.7.7
% Network not in table
```



```
R7#show ip pim vrf S mdt send  
[no output]
```

Rather than have XRv2 and CSR1 join the red VPN (red is now S, since red means where the sources are), we will configure CSR7 to join the blue (N) VPN. The receiver also needs to import the remote RTs for each receiver so that BGP-AD can build the default MDT. We will use a “violet” RT so we can avoid any potential BGP-AD issues. CSR7 will export its extranet sources with RT:213:700 and interested receivers in the blue VPN will import this to satisfy RPF. I do not show this basic RT import configuration on XRv2 and CSR1. The proxy VRF N needs to be fully connected with the receivers; importing their RTs allows it to import the BGP-AD routes and exporting allows the opposite to happen on the other end. Failing to do this both ways results in a broken I-PMSI P-tunnel.

```
! CSR7  
vrf definition N  
  rd 213:77  
  route-target export 213:1  
  route-target export 213:12  
  route-target import 213:7  
  route-target import 213:12  
  route-target import 213:1  
  address-family ipv4  
    mdt auto-discovery pim  
    mdt default 232.255.0.1  
  address-family ipv6  
    mdt auto-discovery pim  
    mdt default 232.255.0.1  
  
vrf definition S  
  address-family ipv4  
    export map RM_EXTRANET_EXPORT  
  
ip prefix-list PL_EXTRANET_SOURCES seq 5 permit 10.2.7.0/24  
  
route-map RM_EXTRANET_EXPORT permit 10  
  match ip address prefix-list PL_EXTRANET_SOURCES  
  set extcommunity rt 213:700 additive  
  
ip multicast-routing vrf N distributed  
ipv6 multicast-routing vrf N  
ip pim vrf N ssm default
```

Before looking at the C-MRIB entries, let’s ensure the BGP-AD Type-1 I-PMSI routes were properly exchanged. CSR7 has the Type-1 routes for XRv12 and CSR1 because it imported RT:213:12 and RT:213:1, respectively. It has two local routes for its own I-PMSI advertisement, one for each MDT. This gets sloppy because we are essentially introducing the red MDT into the blue VPN, which is not

desirable. Fortunately, BGP picks the locally sourced route versus the imported one. This is exported with the RTs that CSR1 and XRv2 will import, and determines which multicast group to use.

```
R7#show bgp ipv4 mvpn vrf N | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 213:77 (default for vrf N)
*>i [1][213:77][213.1.1.1]/12
      213.1.1.1      0      100      0 ?
*> [1][213:77][213.7.7.7]/12
      0.0.0.0      32768 ?
*      0.0.0.0      32768 ?
*>i [1][213:77][213.12.12.12]/12
      213.12.12.12      100      0 i

R7#show bgp ipv4 mvpn vrf N route-type 1 213.7.7.7
BGP routing table entry for [1][213:77][213.7.7.7]/12, version 42
Paths: (2 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGp
peer)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (213.7.7.7)
      Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
      Community: no-export
      Extended Community: RT:213:1 RT:213:12
      PMSI Attribute: Flags: 0x0, Tunnel type: 3, length 8, label: exp-null,
tunnel parameters: D507 0707 E8FF 0001
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  Local, imported path from [1][213:7][213.7.7.7]/12 (S)
    0.0.0.0 from 0.0.0.0 (213.7.7.7)
      Origin incomplete, localpref 100, weight 32768, valid, external
      Community: no-export
      Extended Community: RT:213:7
      PMSI Attribute: Flags: 0x0, Tunnel type: 3, length 8, label: exp-null,
tunnel parameters: D507 0707 E8FF 0002
      rx pathid: 0, tx pathid: 0
```

If we didn't use a new "violet" RT, we would have issues on CSR1/XRv2. It will learn two different Type-1 I-PMSI routes; as they have different RDs, the RR sends both to the endpoints. One of them is the wrong MDT (red) with RT:213:7, and the other is the right MDT (blue) with RTs allowed within the blue VPN. Below is an error case where CSR1 selects the wrong MDT as bestpath and fails to join the blue MDT properly.

```
R1#show bgp ipv4 mvpn vrf N route-type 1 213.7.7.7
BGP routing table entry for [1][213:1][213.7.7.7]/12, version 230
```

```

Paths: (2 available, best #2, table MVPNV4-BGP-Table, not advertised to EBGp
peer)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from [1][213:77][213.7.7.7]/12 (global)
    213.7.7.7 (metric 20) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Community: no-export
      Extended Community: RT:213:1 RT:213:12
      Originator: 213.7.7.7, Cluster list: 213.12.12.12
      PMSI Attribute: Flags: 0x0, Tunnel type: 3, length 8, label: exp-null,
tunnel parameters: D507 0707 E8FF 0001
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  Local, imported path from [1][213:7][213.7.7.7]/12 (global)
    213.7.7.7 (metric 20) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Community: no-export
      Extended Community: RT:213:7
      Originator: 213.7.7.7, Cluster list: 213.12.12.12
      PMSI Attribute: Flags: 0x0, Tunnel type: 3, length 8, label: exp-null,
tunnel parameters: D507 0707 E8FF 0002
      rx pathid: 0, tx pathid: 0x0

```

By using RT:213:700 for the unicast routes, we can safely import that route to satisfy RPF without importing the bogus BGP-AD route seen above. CSR1 will now only receive, and select, the correct Type-1 BGP and properly bind the I-PMSI to the C-MRIB entries. Manipulating other BGP attributes/filters could also solve it, but that is sloppy and beyond the scope of good L3VPN design.

```

R1#show bgp ipv4 mvpn vrf N route-type 1 213.7.7.7
BGP routing table entry for [1][213:1][213.7.7.7]/12, version 264
Paths: (1 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGp
peer)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from [1][213:77][213.7.7.7]/12 (global)
    213.7.7.7 (metric 20) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Community: no-export
      Extended Community: RT:213:1 RT:213:12
      Originator: 213.7.7.7, Cluster list: 213.12.12.12
      PMSI Attribute: Flags: 0x0, Tunnel type: 3, length 8, label: exp-null,
tunnel parameters: D507 0707 E8FF 0001
      rx pathid: 0, tx pathid: 0x0

```

Verification for SSC happens in the reverse order than it did for RSC. We first verify traffic entering from VRF S (red), being passed to VRF N (blue), and then being encapsulated for transmission to the core. Checking the red VPN, we see a valid C-MRIB entry that differs slightly from the RSC equivalent. In RSC,

the OIL was null, and we only had extranet receivers. With SSC, the router still have to send multicast traffic down the red default MDT for other potential red clients, so this extranet VPN is an augment to the overall MVPN. The big 'E' signifies this is an extranet entry, with extranet receivers listed below.

```
R7#show ip mroute vrf S 232.0.0.5 10.2.7.2 | begin \((
(10.2.7.2, 232.0.0.5), 00:15:18/00:03:13, flags: sTE
  Incoming interface: GigabitEthernet2.527, RPF nbr 0.0.0.0
  Outgoing interface list:
    Tunnel3, Forward/Sparse, 00:15:18/00:03:13
  Extranet receivers in vrf N:
    (10.2.7.2, 232.0.0.5), 00:08:24/00:03:05, OIF count: 1, flags: sT
```

Once inside the blue VPN, the C-MRIB verifies RPF. The source is connected (0.0.0.0) and the VRF used for RPF lookups is S. With SSC, we did not have to explicitly configure this VRF-RPF lookup policy as we did with RSC, since the unicast routes were exchanged locally on the same PE. The blue VPN directs traffic out of its default MDT; we can see PIM neighbors have formed on this interface with each receiver (blue) PE. This signals the router to process the packet for encapsulation.

```
R7#show ip pim vrf N neighbor | begin ^Neighbor
Neighbor          Interface          Uptime/Expires    Ver    DR
Address
213.1.1.1         Tunnel1          01:27:46/00:01:39 v2      1 / B S P G
213.12.12.12      Tunnel1          02:18:38/00:01:16 v2      1 / DR B P G
```

```
R7#show ip mroute vrf N 232.0.0.5 10.2.7.2 | begin \((
(10.2.7.2, 232.0.0.5), 00:21:04/00:03:13, flags: sT
  Incoming interface: GigabitEthernet2.527, RPF nbr 0.0.0.0, using vrf S
  Outgoing interface list:
    Tunnel1, Forward/Sparse, 00:21:04/00:03:13
```

Because CSR7 is within the default MDT now for all receivers, it will have multiple P(S,G) entries in the blue default MDT. In this case, receivers exist behind XRv12 and CSR1, but we care more about the traffic being sent from this router, so we check the local P(S,G) entry. Packets being encapsulated by CSR7 are forwarded into core for transport to the blue receivers.

```
R7#show ip mroute 232.255.0.1 213.7.7.7 | begin \((
(213.7.7.7, 232.255.0.1), 00:00:29/00:03:00, flags: sT
  Incoming interface: Loopback0, RPF nbr 0.0.0.0
  Outgoing interface list:
    GigabitEthernet2.574, Forward/Sparse, 00:00:29/00:03:00
    GigabitEthernet2.573, Forward/Sparse, 00:00:29/00:03:00
```

We can begin sending traffic from CSR2 and checking packet counters. We see packets enter VRF S and get forwarded to VRF N. From there, they are sent down the blue MDT towards blue receivers.

```
R7#show ip mroute vrf S 232.0.0.5 10.2.7.2 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 10, Packets received:
10
Source: 10.2.7.2/32, Forwarding: 10/1/118/0, Other: 10/0/0
```

```
R7#show ip mroute vrf N 232.0.0.5 10.2.7.2 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 10, Packets received:
10
Source: 10.2.7.2/32, Forwarding: 10/1/118/0, Other: 10/0/0
```

```
R7#show ip mroute 232.255.0.1 213.7.7.7 count | begin ^Group
Group: 232.255.0.1, Source count: 3, Packets forwarded: 126, Packets
received: 126
Source: 213.7.7.7/32, Forwarding: 54/1/91/1, Other: 54/0/0
```

At the remote end, we see packets being received by the VRF N (blue) PEs.

```
RP/0/0/CPU0:XRv2#show mfib vrf N route 232.0.0.5 10.2.7.2 | begin 232
(10.2.7.2,232.0.0.5), Flags:
Up: 06:00:15
Last Used: never
SW Forwarding Counts: 0/23/2300
SW Replication Counts: 0/23/2300
SW Failure Counts: 0/0/0/0/0
mdtN Flags: A MI, Up:00:09:38
GigabitEthernet0/0/0/0.512 Flags: NS EG, Up:06:00:15
```

```
R1#show ip mroute vrf N 232.0.0.5 10.2.7.2 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 49, Packets received:
49
Source: 10.2.7.2/32, Forwarding: 49/1/142/1, Other: 49/0/0
```

An EPC verification on CSR1 shows packets arriving on the blue MDT from the core and being sent towards the extranet customers.

```
R1#show monitor capture CAP buffer detailed
0 142 0.000000 213.7.7.7 -> 232.255.0.1 GRE
0000: 01005E7F 00010050 56A9862A 8100CDBA ..^....PV...*....
0010: 08004500 007C0AE5 0000FE2F EC5ED507 ..E...|...../.^..
0020: 0707E8FF 00010000 08004500 00645E64 .....E..d^d
0030: 0000FE01 652B0A02 0702E800 00050800 ....e+.....

1 118 0.001007 10.2.7.2 -> 232.0.0.5 ICMP
0000: 01005E00 00050050 56A91AAA 81000DBF ..^....PV.....
0010: 08004500 00645E64 0000FD01 662B0A02 ..E..d^d....f+..
0020: 0702E800 00050800 2EF0001D 00BE0000 .....
0030: 000037E8 1697ABCD ABCDABCD ABCDABCD ..7.....
```

Last, we examine using the SSC method with IPv6. We won't intentionally break this network up front by failing to use a "violet RT" since that issue was already demonstrated. The vast majority of the configuration will be on CSR7 again, and the basic things like configuring the default MDT and BGP-AD under the VRF IPv6 AF is not shown. The configuration is identical to IPv4. Adding a new IPv6 prefix-list and route-map to account for the source behind CSR7 is also required, but is not shown because it is very basic. This sets the violet RT:213:700 so routers in the blue VPN (receivers like XRv2 and CSR1) can import the unicast VPNv6 routes for RPF validation, but not the BGP-AD routes related to the red VPN. It's worth verifying that those egress PEs only have a single BGP Type-1 I-PMSI route for the blue VPN. The default MDT for VRF N (the blue VPN in this test) is 232.255.0.1, which is shown below. Notice there is no violet RT attached to these BGP-AD routes.

```
R1#show bgp ipv6 mvpn vrf N route-type 1 213.7.7.7
[snip]
Local, imported path from [1][213:77][213.7.7.7]/12 (global)
  213.7.7.7 (metric 20) from 213.12.12.12 (213.12.12.12)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    Community: no-export
    Extended Community: RT:213:1 RT:213:12
    Originator: 213.7.7.7, Cluster list: 213.12.12.12
    PMSI Attribute: Flags: 0x0, Tunnel type: 3, length 8, label: exp-null,
tunnel parameters: D507 0707 E8FF 0001
    rx pathid: 0, tx pathid: 0x0
```

```
RP/0/0/CPU0:XRv2#show bgp ipv6 mvpn vrf N [1][213.7.7.7]/40
[snip]
Local, (Received from a RR-client)
  213.7.7.7 (metric 20) from 213.7.7.7 (213.7.7.7)
    Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, imported
    Received Path ID 0, Local Path ID 1, version 139
    Community: no-export
    Extended community: RT:213:1 RT:213:12
    PMSI: flags 0x00, type 3, label 0, ID 0xd5070707e8ff0001
    Source VRF: default, Source Route Distinguisher: 213:77
```

Next, we can validate two-way PIM neighbors over the blue MDT. CSR7 must be able to receive C(S,G) joins from the extranet receivers within the blue VPN.

```
R7#show ipv6 pim vrf N neighbor
PIM Neighbor Table
Mode: B - Bidir Capable, G - GenID Capable
Neighbor Address      Interface      Uptime        Expires       Mode DR pri
::FFFF:213.1.1.1      Tunnel1        00:24:50      00:01:35     B G      1
::FFFF:213.12.12.12   Tunnel1        01:15:44      00:01:22     B G      DR 1

R1#show ipv6 pim vrf N neighbor | include 7.7
::FFFF:213.7.7.7      Tunnel2        00:26:01      00:01:24     B G      1
```

```
RP/0/0/CPU0:XRv2#show pim vrf N ipv6 neighbor | include 7.7
::ffff:213.7.7.7                                01:16:45  00:01:34 1
```

Both XRv2 and CSR1 have valid RPF interfaces for these extranet sources, so they send C-PIM joins towards CSR7 over the blue MDT. The output doesn't show which protocol the RPF route was learned, but the admin distance of 200 hints at iBGP (VPNv6). These blue receivers need not have any red configuration on them as the source (CSR7) is performing the replication.

```
R1#show ipv6 mroute vrf N FF33::1 2001:10:2:7::2 | begin \
(2001:10:2:7::2, FF33::1), 00:28:37/never, flags: sTI
  Incoming interface: Tunnel2
  RPF nbr: ::FFFF:213.7.7.7
  Immediate Outgoing interface list:
    GigabitEthernet2.519, Forward, 00:28:37/never
```

```
R1#show ipv6 rpf vrf N 2001:10:2:7::2
RPF information for 2001:10:2:7::2
  RPF interface: Tunnel2
  RPF neighbor: ::FFFF:213.7.7.7
  RPF route/mask: 2001:10:2:7::/64
  RPF type: Unicast
  RPF recursion count: 0
  Metric preference: 200
  Metric: 0
```

```
RP/0/0/CPU0:XRv2#show pim vrf N ipv6 topology ff33::1 2001:10:2:7::2 | begin
ffff
(2001:10:2:7::2, ff33::1) SPT
SSM Up: 07:14:35 JP: Join(00:01:00) Flags:
RPF: mdtN, ::ffff:213.7.7.7
  GigabitEthernet0/0/0/0.512 07:14:35 fwd LI LH
```

```
RP/0/0/CPU0:XRv2#show pim vrf N ipv6 rpf 2001:10:2:7::2
Table: IPv6-Unicast-default
2001:10:2:7::2/128 [200/0]
  via ::ffff:213.7.7.7, mdtN
  Connector 213:7:213.7.7.7, Nexthop ::ffff:213.7.7.7
```

Performing the MRIB verification on CSR7, we do them in reverse order from RSC. Traffic entering the PE arrives in the red VPN (S) from the LAN first. The traffic is sent down the red MDT (not null, as it was in RSC) and also sent into the extranet VPN, which is blue. The blue C-MRIB sends it to the blue default MDT for encapsulation, and this blue VPN uses the red VPN for RPF verification.

```
R7#show ipv6 mroute vrf S FF33::1 2001:10:2:7::2 | begin \
(2001:10:2:7::2, FF33::1), 01:33:52/00:03:18, flags: sTE
  Incoming interface: GigabitEthernet2.527
```

```
RPF nbr: 2001:10:2:7::2
Immediate Outgoing interface list:
  Tunnel13, Forward, 01:33:46/00:03:18
```

Extranet receivers in vrf N, OIF count 1

```
R7#show ipv6 mroute vrf N FF33::1 2001:10:2:7::2 | begin \
(2001:10:2:7::2, FF33::1), 01:33:52/00:02:36, flags: sT
Incoming interface: GigabitEthernet2.527
RPF nbr: 2001:10:2:7::2, using vrf S
Immediate Outgoing interface list:
  Tunnel1, Forward, 01:33:52/00:02:36
```

```
R7#show ip mroute 232.255.0.1 213.7.7.7 | begin \
(213.7.7.7, 232.255.0.1), 01:36:36/00:03:10, flags: sT
Incoming interface: Loopback0, RPF nbr 0.0.0.0
Outgoing interface list:
  GigabitEthernet2.574, Forward/Sparse, 01:36:21/00:03:10
  GigabitEthernet2.573, Forward/Sparse, 01:36:36/00:02:52
```

CSR2 begins sending traffic; we validate counters for each one of the MRIB entries. Naturally, the IPv4 default MDT (blue) will have many more hits than the C(S,G) specific for this IPv6 SSM flow as it carries all traffic in the default MDT.

```
R7#show ipv6 mroute vrf S FF33::1 2001:10:2:7::2 count | begin ^Group
Group: FF33::1
Source: 2001:10:2:7::2,
SW Forwarding: 0/0/0/0, Other: 0/0/0
HW Forwarding: 14/1/118/0, Other: 0/0/0
Totals - Source count: 1, Packet count: 14
```

```
R7#show ipv6 mroute vrf N FF33::1 2001:10:2:7::2 count | begin ^Group
Group: FF33::1
Source: 2001:10:2:7::2,
SW Forwarding: 0/0/0/0, Other: 0/0/0
HW Forwarding: 14/1/118/0, Other: 0/0/0
Totals - Source count: 1, Packet count: 14
```

```
R7#show ip mroute 232.255.0.1 213.7.7.7 count | begin ^Group
Group: 232.255.0.1, Source count: 3, Packets forwarded: 2685, Packets
received: 2685
Source: 213.7.7.7/32, Forwarding: 1783/1/112/0, Other: 1783/0/0
```

At the remote end, we can see XRv2 and CSR1 accepting and forwarding packets. An EPC capture on CSR1 further proves it by showing the GRE encapsulated MDT tunnel arriving and the raw IPv6-ICMP exiting towards the extranet customer.


```

RP/0/0/CPU0:XRv2#show mfib vrf N ipv6 route ff33::1 2001:10:2:7::2 | begin
ff33
(2001:10:2:7::2,ff33::1)
Flags:
Up: 07:34:05
Last Used: never
SW Forwarding Counts: 0/172/17200
SW Replication Counts: 0/172/17200
SW Failure Counts: 0/0/0/0/0
mdtN Flags: A MI, Up:01:43:28
GigabitEthernet0/0/0/0.512 Flags: NS EG, Up:07:34:05

R1#show ipv6 mroute vrf N FF33::1 2001:10:2:7::2 count | begin ^Group
Group: FF33::1
Source: 2001:10:2:7::2,
SW Forwarding: 0/0/0/0, Other: 0/0/0
HW Forwarding: 191/1/142/1, Other: 0/0/0
Totals - Source count: 1, Packet count: 191

1#show monitor capture CAP buffer brief
0 142 0.000000 213.7.7.7 -> 232.255.0.1 GRE
1 118 0.000000 2001::*:0002 -> FF33::*:0001 IPv6-ICMP

```

Additional Reading – Reference configurations “mvpn-extranet-gre”

29.4.2 mLDP

Extranet services for mLDP is not very different in principle from the PIM/GRE examples. We are still using the concepts of default/data MDTs, along with Type-1 (I-PMSI) and Type-3 (S-PMSI) BGP-AD routes. The base MVPN profile in use this time is profile 9, so C-PIM signaling is still used in the overlay. VRF N covers the top three routers and VRF S covers the bottom three. VRF N uses XRv4 as its mLDP MP2MP root and VRF S uses XRv3. VRF N has a VPN ID of 213:1512 while VRF S had a VPN ID of 213:678. These must match on all routers in a VPN, along with the MP2MP root addresses.

Before getting started with extranet services, we will do very brief checks of the mLDP infrastructure for both VPNs to ensure the basic MP2MP trees are built. As the root for VRF N, XRv4 should have three downstream clients on its MP2MP tree: CSR1, CSR5, and XRv3 (towards XRv2). XRv3 likewise has one downstream (XRv2) and one upstream (XRv4) client. For VRF S, XRv3 has three downstream clients: CSR6, CSR7, and XRv4 (towards CSR8). XRv4 has one upstream client (XRv3) and one downstream client (CSR8). This is a good estimate that things are working properly.

```

RP/0/0/CPU0:XRv4#show mpls mldp database mp2mp brief
LSM ID   Type   Root           Up Down Decoded Opaque Value
0x00004  MP2MP  213.13.13.13   1  1    [mdt 213:678 0]
0x00001  MP2MP  213.14.14.14   0  3    [mdt 213:1512 0]

```

```

RP/0/0/CPU0:XRv3#show mpls mldp database mp2mp brief

```

LSM ID	Type	Root	Up	Down	Decoded Opaque Value
0x00002	MP2MP	213.13.13.13	0	3	[mdt 213:678 0]
0x00001	MP2MP	213.14.14.14	1	1	[mdt 213:1512 0]

Next, we validate C-PIM adjacencies in each overlay. For brevity, we will check CSR1 and CSR8 only. Each of them has a neighborhood with the other two VPN members. Realistically, one would want to verify all PEs in this way as PIM neighbors can be unidirectional.

```
R1#show ip pim vrf N neighbor | begin ^Neighbor
Neighbor          Interface          Uptime/Expires      Ver   DR
Address           Prio/Mode
213.12.12.12      Lspvif0          00:23:09/00:01:23  v2    1 / DR B P G
213.5.5.5         Lspvif0          00:25:10/00:01:31  v2    1 / B S P G
```

```
R8#show ip pim vrf S neighbor | begin ^Neighbor
Neighbor          Interface          Uptime/Expires      Ver   DR
Address           Prio/Mode
213.6.6.6         Lspvif0          00:03:17/00:01:24  v2    1 / B S P G
213.7.7.7         Lspvif0          00:03:17/00:01:23  v2    1 / B S P G
```

A quick test of intranet MVPN traffic will verify proper operation before progressing to extranets. This will not be a comprehensive review of mLDP profile 9, however. CSR10 starts sending traffic to group 232.0.0.7. We check the C-MRIB states to see how the routers are forwarding traffic. A data MDT (mLDP P2MP) is in use, which implies the presence of a Type-3 S-PMSI route. This is where XRv stops working; it claims null RPF when clearly that is not true. We don't expect it to work with LSM anyway.

```
R5#show ip mroute vrf N 232.0.0.7 10.5.10.10 verbose | begin \
(10.5.10.10, 232.0.0.7), 00:31:59/00:03:23, flags: sTyp
Incoming interface: GigabitEthernet2.550, RPF nbr 0.0.0.0
MDT TX nr: 1 LSM-ID: 0x3
Outgoing interface list:
Lspvif0, LSM MDT: 3 (data), Forward/Sparse, 00:31:59/00:03:23, p
```

```
R1#show ip mroute vrf N 232.0.0.7 10.5.10.10 | begin \
(10.5.10.10, 232.0.0.7), 06:20:31/00:02:34, flags: sTIY
Incoming interface: Lspvif0, RPF nbr 213.5.5.5, MDT: [1, 213.5.5.5]/never
Outgoing interface list:
GigabitEthernet2.519, Forward/Sparse, 06:20:31/00:02:34
```

```
RP/0/0/CPU0:XRv2#show pim vrf N topology 232.0.0.7 10.5.10.10 | begin 232
(10.5.10.10,232.0.0.7)SPT SSM Up: 00:03:14
JP: Join(00:00:34) RPF: Null,213.5.5.5 Flags:
GigabitEthernet0/0/0/0.512 00:03:14 fwd LI LH
```

```
RP/0/0/CPU0:XRv2#show pim vrf N rpf 10.5.10.10
Table: IPv4-Unicast-default
* 10.5.10.10/32 [200/0]
```

via LmdtN with rpf neighbor 213.5.5.5

We check the MDT data bindings on all routers. XRv2 and CSR1 receive MDT information via the Type-3 S-PMSI route, while CSR5 sends it. The S-PMSI specifies an mLDP P2MP tunnel (type 2) rooted at CSR5.

```
R5#show bgp ipv4 mvpn vrf N route-type 3 10.5.10.10 232.0.0.7 213.5.5.5
BGP routing table entry for [3] [213:5] [10.5.10.10] [232.0.0.7] [213.5.5.5]/22,
[snip]
  Local
    0.0.0.0 from 0.0.0.0 (213.5.5.5)
      Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
      Community: no-export
      Extended Community: RT:213:5
      PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 24, label: exp-null,
tunnel parameters: 0600 0104 D505 0505 000E 0200 0B00 0213 0000 1512 0000
0001
      rx pathid: 0, tx pathid: 0x0
```

```
R5#show ip pim vrf N mdt send
MDT-data send list for VRF: N
  (source, group)                                MDT-data group/num  ref_count
  (10.5.10.10, 232.0.0.7)                        1                   1
```

```
R1#show ip pim vrf N mdt receive
Joined MDT-data [group/mdt number : source]  uptime/expires for VRF: N
[1 : 213.5.5.5]  00:02:11/stopped
```

```
RP/0/0/CPU0:XRv2#show ip pim vrf N mdt cache
Core Source      Cust (Source, Group)                Core Data          Expires
213.5.5.5        (10.5.10.10, 232.0.0.7)            [mdt 213:1512 1]  never
```

Sending traffic from CSR10 shows packets being sent into the core by CSR5 and received by CSR1. EPC confirms the MPLS decapsulation and native IP multicast forwarding to CSR9. The difference in 4 bytes represents the popping of the MPLS shim header. A final validation of label 0x3F4 (1012) shows this is the proper local downstream label for CSR1 along that data MDT. The label is highlighted in yellow while the IP headers are in green.

```
R5#show ip mroute vrf N 232.0.0.7 10.5.10.10 count | begin ^Group
Group: 232.0.0.7, Source count: 1, Packets forwarded: 27, Packets received:
27
  Source: 10.5.10.10/32, Forwarding: 27/1/118/0, Other: 27/0/0
```

```
R1#show ip mroute vrf N 232.0.0.7 10.5.10.10 count | begin ^Group
Group: 232.0.0.7, Source count: 1, Packets forwarded: 28, Packets received:
28
  Source: 10.5.10.10/32, Forwarding: 28/1/122/0, Other: 28/0/0
```

```

R1#show monitor capture CAP buffer detail
 0 122      0.000000  00:50:56:A9:86:2A -> 00:50:56:A9:1A:AA MPLS unicast
0000:  005056A9 1AAA0050 56A9862A 81000DBA  .PV....PV...*....
0010:  8847003F 41FD4500 00641ECF 0000FE01  .G.?A.E..d.....
0020:  A1B30A05 0A0AE800 00070800 D7E20012  .....
0030:  00410000 000039B5 6C5FABCD ABCDABCD  .A....9.1_.....

 1 118      0.000000  10.5.10.10      -> 232.0.0.7      ICMP
0000:  01005E00 00070050 56A91AAA 81000DBF  ..^....PV.....
0010:  08004500 00641ECF 0000FC01 A3B30A05  ..E..d.....
0020:  0A0AE800 00070800 D7E20012 00410000  .....A..
0030:  000039B5 6C5FABCD ABCDABCD ABCDABCD  ..9.1_.....

```

```

R1#show mpls mldp database opaque_type mdt 213:1512 1 | section Upstream
Upstream client(s) :
 213.14.14.14:0      [Active]
    Expires          : Never          Path Set ID   : 5
    Out Label (U)    : None           Interface     : GigabitEthernet2.514*
    Local Label (D) : 1012           Next Hop      : 213.1.14.14

```

Because S-PMSI is offered for (10.5.10.10, 232.0.0.7), we theorize that importing the RT associated with the BGP-AD and VPNv4 routes from CSR5 will be sufficient to build connectivity to extranets. With GRE, this worked quite easily, since the entire core was PIM-enabled and there was nothing stopping PIM P(S,G) joins for data MDT groups transiting between any PEs. At this point, neither CSR6 nor CSR8 have the red VPN (N) configured locally. VRF N will be our “red VPN” at first, the one with the sources, so we will not verify intranet MVPN on VRF S yet. Instead, we will perform a similar test as we did with GRE. CSR5 will create a “violet” RT for its BGP-AD routes along with any extranet sources. CSR8 will import this “violet” RT as an extranet client. CSR6 will import the raw red RT:213:5 and attempt to draw extranet MVPN services. These imports are happening directly into the blue VPN (S), not a proxy red VPN (N). Fortunately, the entire “violet” RT constructs are identical to the PIM/GRE model and the configurations are not displayed again. We can verify that both the VPNv4 route for 10.5.100/24 and the BGP-AD Type1/3 routes from CSR5 both have the new violet RT:213:500.

```

RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf N 10.5.10.0/24 | include Extend
Extended community: RT:213:5 RT:213:500

```

```

RP/0/0/CPU0:XRv2#show bgp ipv4 mvpn vrf N [1][213.5.5.5]/40 | include Extend
Extended community: RT:213:5 RT:213:500

```

```

RP/0/0/CPU0:XRv2#show bgp ipv4 mvpn vrf N
[3][32][10.5.10.10][32][232.0.0.7][213.5.5.5]/120 | include Extend
Extended community: RT:213:5 RT:213:500

```

A quick check of the mLDP database on CSR6 shows that the P2MP tunnel rooted at CSR5 was not built. We can clearly see the Type-3 S-PMSI route has been imported. The issue is that the VPN ID is encoded in the variable-length opaque nested inside the FEC element. The PMSI attributes carry this value, which

is 213:1512. There are no VRFs locally configured that match this VPN ID, so the MVPN construction cannot occur. We must use another method.

```
R6#show mpls mldp database summary
```

LSM ID	Type	Root	Decoded Opaque Value	Client Cnt.
C	P2MP	213.7.7.7	[mdt 213:678 1]	1
9	MP2MP	213.13.13.13	[mdt 213:678 0]	1

```
R6#show bgp ipv4 mvpn vrf S route-type 3 10.5.10.10 232.0.0.7 213.5.5.5
```

```
BGP routing table entry for [3][213:6][10.5.10.10][232.0.0.7][213.5.5.5]/22,
version 130
```

```
Paths: (1 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGp
peer)
```

```
Not advertised to any peer
```

```
Refresh Epoch 1
```

```
Local, imported path from [3][213:5][10.5.10.10][232.0.0.7][213.5.5.5]/22
(global)
```

```
213.5.5.5 (metric 20) from 213.12.12.12 (213.12.12.12)
```

```
Origin incomplete, metric 0, localpref 100, valid, internal, best
```

```
Community: no-export
```

```
Extended Community: RT:213:5 RT:213:500
```

```
Originator: 213.5.5.5, Cluster list: 213.12.12.12
```

```
PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 24, label: exp-null,
tunnel parameters: 0600 0104 D505 0505 000E 0200 0B00 0213 0000 1512 0000
0001
```

```
rx pathid: 0, tx pathid: 0x0
```

We will create the red VPN locally on CSR6 and CSR8. This is the RSC method we used for PIM/GRE, which implies we configure the source VPN on the receiver PE. The VPN ID will be 213:1512 and for cleanliness, we will remove the red and violet RT import statements from VRF S on CSR6 and CSR8, respectively. We will import these into the red VPN (N) only. Between the violet and red RTs, both CSR6 and CSR8 should be part of this mLDP MP2MP tree. We can check for C-PIM neighbors in each router. CSR6 and CSR8 also form neighbors with one another, as expected, but of greatest importance are the neighborships with CSR5.

```
! CSR6
```

```
vrf definition N
```

```
rd 213:66 ! On CSR8, this is 213:88 instead
```

```
vpn id 213:1512
```

```
route-target import 213:5 ! On CSR8, this is 213:500 instead
```

```
address-family ipv4
```

```
mdt default mpls mldp 213.14.14.14
```

```
address-family ipv6
```

```
mdt default mpls mldp 213.14.14.14
```

```
vrf definition S
```

```
no route-target import 213:5 ! On CSR8, this is 213:500 instead
```

```
R6#show ip pim vrf N neighbor | begin ^Neighbor
Neighbor          Interface          Uptime/Expires    Ver    DR
Address
213.8.8.8          Lspvif10          00:04:16/00:01:27 v2      1 / B S P G
213.12.12.12       Lspvif10          00:06:24/00:01:31 v2      1 / DR B P G
213.1.1.1          Lspvif10          00:06:26/00:01:23 v2      1 / B S P G
213.5.5.5          Lspvif10          00:06:26/00:01:24 v2      1 / B S P G
```

```
R8#show ip pim vrf N neighbor | begin ^Neighbor
Neighbor          Interface          Uptime/Expires    Ver    DR
Address
213.12.12.12       Lspvif1           00:04:07/00:01:38 v2      1 / DR B P G
213.6.6.6          Lspvif1           00:04:09/00:01:31 v2      1 / B S P G
213.5.5.5          Lspvif1           00:04:09/00:01:31 v2      1 / B S P G
213.1.1.1          Lspvif1           00:04:09/00:01:31 v2      1 / B S P G
```

Examining the C-MRIB entries, we see that traffic arrives into the proxy red VPN (N) and is piped in software to the blue VPN (S). The red entry shows both the big 'Y' and big 'E' entries, indicating traffic was received from an MDT data group and also represents extranet capability. We never saw these two flags together in the PIM/GRE method because the traditional RSC/SSC extranet configuration was not needed when using S-PMSI. Also notice that, so far, it doesn't seem to matter whether we used the violet (CSR8) or red (CSR6) RTs for pulling in these BGP-AD and VPNv4 unicast routes.

```
R8#show ip mroute vrf N 232.0.0.7 10.5.10.10 | begin \
(10.5.10.10, 232.0.0.7), 00:03:43/stopped, flags: sTYE
Incoming interface: Lspvif1, RPF nbr 213.5.5.5, MDT: [1, 213.5.5.5]/never
Outgoing interface list: Null
```

```
Extranet receivers in vrf S:
(10.5.10.10, 232.0.0.7), 11:29:51/00:02:11, OIF count: 1, flags: sTI
```

```
R8#show ip mroute vrf S 232.0.0.7 10.5.10.10 | begin \
(10.5.10.10, 232.0.0.7), 11:29:55/00:02:07, flags: sTI
Incoming interface: Lspvif1, RPF nbr 213.5.5.5, using vrf N
Outgoing interface list:
GigabitEthernet2.538, Forward/Sparse, 11:29:55/00:02:07
```

```
R6#show ip mroute vrf N 232.0.0.7 10.5.10.10 | begin \
(10.5.10.10, 232.0.0.7), 00:01:35/00:01:24, flags: sTYE
Incoming interface: Lspvif10, RPF nbr 213.5.5.5, MDT: [1, 213.5.5.5]/never
Outgoing interface list: Null
```

```
Extranet receivers in vrf S:
(10.5.10.10, 232.0.0.7), 20:30:55/00:02:59, OIF count: 1, flags: sTI
```

```
R6#show ip mroute vrf S 232.0.0.7 10.5.10.10 | begin \
(10.5.10.10, 232.0.0.7), 20:30:49/00:02:04, flags: sTI
```

```
Incoming interface: Lspvif10, RPF nbr 213.5.5.5, using vrf N
Outgoing interface list:
  GigabitEthernet2.546, Forward/Sparse, 20:30:49/00:02:04
```

Since these routers claim to have received data MDT information, we check their caches to confirm. This would have been derived from the Type-3 S-PMSI route, which both routes have imported into VRF N.

```
R8#show ip pim vrf N mdt receive
Joined MDT-data [group/mdt number : source]  uptime/expires for VRF: N
[1 : 213.5.5.5]  10:54:30/stopped
```

```
R8#show bgp ipv4 mvpn vrf N | include \[3\]
*>i [3] [213:88] [10.5.10.10] [232.0.0.7] [213.5.5.5] /22
```

```
R6#show ip pim vrf N mdt receive
Joined MDT-data [group/mdt number : source]  uptime/expires for VRF: N
[1 : 213.5.5.5]  10:57:49/stopped
```

```
R6#show bgp ipv4 mvpn vrf N | include \[3\]
*>i [3] [213:66] [10.5.10.10] [232.0.0.7] [213.5.5.5] /22
```

The RPF mechanism for this traffic is identical to how it was for PIM/GRE. CSR6 and CSR8 have group-based RPF selection policies inside VRF S (blue) that specify certain groups can be RPF-checked in VRF N (red). The configuration is identical on CSR6 and CSR8, so only one is shown.

```
R6#show ip rpf vrf S select
Multicast Group-to-Vrf Mappings
Group(s): 232.0.0.7/32, RPF vrf: N, Acl: ACL_N_GROUPS
```

Knowing that the MDT number is 1, we can check their mLDP databases to determine the incoming label for each one. This will be used for the final data-plane verifications as well. CSR8 maps this entry to LSPvif1 while CSR6 selects LSPvif10; in both cases, this matches with the virtual interface assigned to this VPN as shown in the PIM neighbor show commands. CSR8 uses local downstream label 8013 while CSR6 uses 6013. As a sanity check, we also see the root is CSR5 and the type is P2MP, which is correct.

```
R8#show mpls mldp database opaque_type mdt 213:1512 1
LSM ID : E   Type: P2MP   Uptime : 00:08:29
FEC Root      : 213.5.5.5
Opaque decoded : [mdt 213:1512 1]
Opaque length  : 11 bytes
Opaque value   : 02 000B 0002130000151200000001
Upstream client(s) :
  213.14.14.14:0   [Active]
    Expires        : Never          Path Set ID   : E
    Out Label (U)  : None           Interface     : GigabitEthernet2.584*
    Local Label (D): 8013           Next Hop      : 213.8.14.14
Replication client(s):
```

```

MDT (VRF N)
  Uptime           : 00:08:29      Path Set ID   : None
  Interface        : Lspvif1
R6#show mpls mldp database opaque_type mdt 213:1512 1
LSM ID : 13      Type: P2MP      Uptime : 00:09:50
FEC Root      : 213.5.5.5
Opaque decoded : [mdt 213:1512 1]
Opaque length  : 11 bytes
Opaque value   : 02 000B 0002130000151200000001
Upstream client(s) :
  213.13.13.13:0 [Active]
    Expires      : Never          Path Set ID   : 13
    Out Label (U) : None          Interface     : GigabitEthernet2.563*
    Local Label (D) : 6013        Next Hop      : 213.6.13.13
Replication client(s):
  MDT (VRF N)
    Uptime           : 00:09:50      Path Set ID   : None
    Interface        : Lspvif10

```

It is worth mentioning that although these P2MP trees will be used for delivery, both CSR6 and CSR8 are part of the red MP2MP default MDT. Without it, there would be no C-PIM overlay signaling.

```

R6#show mpls mldp database summary | include MP2MP
9      MP2MP  213.13.13.13      [mdt 213:678 0]      1
10     MP2MP  213.14.14.14      [mdt 213:1512 0]    1

R8#show mpls mldp database summary | include MP2MP
1      MP2MP  213.13.13.13      [mdt 213:678 0]      1
B      MP2MP  213.14.14.14      [mdt 213:1512 0]      1

```

A quick verification on CSR5 shows nothing unusual; the C-MRIB entry sends traffic down a data MDT (little 'y'). The "verbose" keyword shows us the MDT details that are bound to this C(S,G).

```

R5#show ip mroute vrf N 232.0.0.7 10.5.10.10 verbose | begin \
(10.5.10.10, 232.0.0.7), 00:18:49/00:02:51, flags: sTyp
Incoming interface: GigabitEthernet2.550, RPF nbr 0.0.0.0
MDT TX nr: 1  LSM-ID: 0x7
Outgoing interface list:
  Lspvif0, LSM MDT: 7 (data), Forward/Sparse, 11:47:39/00:02:51, p

```

CSR10 begins sending traffic. CSR5 replicates the traffic to XRv3 and XRv4 using labels 93015 and 94009 respectively. Both of these core routers pass it down the tree to their receivers. Notice that label 1012 is in the replication list for XRv4; the intranet receivers expressing interest in this C(S,G) still receive the traffic flow.

```

R5#show mpls mldp database opaque_type mdt 213:1512 1 | section Replicat

```


Replication client(s):

```
MDT (VRF N)
  Uptime      : 11:21:41      Path Set ID : None
  Interface   : Lspvif0
213.14.14.14:0
  Uptime      : 11:21:41      Path Set ID : None
  Out label (D) : 94009       Interface   : GigabitEthernet2.554*
  Local label (U): None       Next Hop    : 213.5.14.14
213.13.13.13:0
  Uptime      : 00:16:14      Path Set ID : None
  Out label (D) : 93015       Interface   : GigabitEthernet2.553*
  Local label (U): None       Next Hop    : 213.5.13.13
```

RP/0/0/CPU0:XRv3#show mpls forwarding labels 93015

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
93015	6013	MLDP: 0x00006	Gi0/0/0/0.563	213.6.13.6	0

RP/0/0/CPU0:XRv4#show mpls forwarding labels 94009

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94009	1012	MLDP: 0x00002	Gi0/0/0/0.514	213.1.14.1	0
	8013	MLDP: 0x00002	Gi0/0/0/0.584	213.8.14.8	0

Next, we check the C-MRIB counters on CSR6 and CSR8 within both the red and blue VPNs. Traffic counters should be identical between the two if things are working properly. We also show the red VPN counters on CSR1 to prove that intranet connectivity is not broken.

```
R8#show ip mroute vrf N 232.0.0.7 10.5.10.10 count | begin ^Group
Group: 232.0.0.7, Source count: 1, Packets forwarded: 37, Packets received:
37
Source: 10.5.10.10/32, Forwarding: 37/1/122/0, Other: 37/0/0
```

```
R8#show ip mroute vrf S 232.0.0.7 10.5.10.10 count | begin ^Group
Group: 232.0.0.7, Source count: 1, Packets forwarded: 37, Packets received:
37
Source: 10.5.10.10/32, Forwarding: 37/1/122/0, Other: 37/0/0
```

```
R6#show ip mroute vrf N 232.0.0.7 10.5.10.10 count | begin ^Group
Group: 232.0.0.7, Source count: 1, Packets forwarded: 34, Packets received:
34
Source: 10.5.10.10/32, Forwarding: 34/1/122/0, Other: 34/0/0
```

```
R6#show ip mroute vrf S 232.0.0.7 10.5.10.10 count | begin ^Group
Group: 232.0.0.7, Source count: 1, Packets forwarded: 34, Packets received:
34
```

Source: 10.5.10.10/32, Forwarding: 34/1/122/0, Other: 34/0/0

```
R1#show ip mroute vrf N 232.0.0.7 10.5.10.10 count | begin ^Group
Group: 232.0.0.7, Source count: 1, Packets forwarded: 20, Packets received:
20
```

Source: 10.5.10.10/32, Forwarding: 20/1/122/0, Other: 20/0/0

Last, we conduct EPC verifications on CSR6 and CSR8 to ensure the proper labels are entering and the ICMP multicast is being decapsulated and sent to the customer. MPLS shim headers are shown in yellow and the IP multicast headers are shown in green. We see that CSR6 receives label 0x177D (6013) and CSR8 receives label 0x1F4D (8013), which is expected.

```
R6#show monitor capture CAP buffer detailed
```

```
0 122 0.000000 00:50:56:A9:EA:54 -> 00:50:56:A9:DE:0D MPLS unicast
0000: 005056A9 DE0D0050 56A9EA54 81000DEB .PV....PV..T....
0010: 88470177 D1FD4500 00642176 0000FE01 .G.w..E..d!v....
0020: 9F0C0A05 0A0AE800 00070800 57020015 .....W...
0030: 00A70000 00003C27 EA64ABCD ABCDABCD .....<'.d.....
```

```
1 118 0.000000 10.5.10.10 -> 232.0.0.7 ICMP
0000: 01005E00 00070050 56A9DE0D 81000DDA ..^....PV.....
0010: 08004500 00642176 0000FC01 A10C0A05 ..E..d!v.....
0020: 0A0AE800 00070800 57020015 00A70000 .....W.....
0030: 00003C27 EA64ABCD ABCDABCD ABCDABCD ..<'.d.....
```

```
R8#show monitor capture CAP buffer detailed
```

```
2 122 0.630024 00:50:56:A9:86:2A -> 00:50:56:A9:FB:1C MPLS unicast
0000: 005056A9 FB1C0050 56A9862A 81000E00 .PV....PV..*....
0010: 884701F4 D1FD4500 006421AE 0000FE01 .G....E..d!.....
0020: 9ED40A05 0A0AE800 00070800 7BE60015 .....{...
0030: 00DF0000 00003C28 C547ABCD ABCDABCD .....<(.G.....
```

```
3 118 0.630024 10.5.10.10 -> 232.0.0.7 ICMP
0000: 01005E00 00070050 56A9FB1C 81000DD2 ..^....PV.....
0010: 08004500 006421AE 0000FC01 A0D40A05 ..E..d!.....
0020: 0A0AE800 00070800 7BE60015 00DF0000 .....{.....
0030: 00003C28 C547ABCD ABCDABCD ABCDABCD ..<(.G.....
```

Next, we will configure CSR5 to stop offering data MDT service for this C(S,G). Using the “deny ip any any” technique in the data MDT ACL, we will force the traffic to use the default MDT. To prove it worked, the two lines below show the before/after results. CSR5 originally originated a Type-3 S-PMSI route for that C(S,G), but now has withdrawn it. CSR5 no longer displays the little ‘y’ flag on its C(S,G) entry and specifically says the MDT in use is the “default”.

```
R5#show bgp ipv4 mvpn vrf N | include \[3\]
*> [3] [213:5] [10.5.10.10] [232.0.0.7] [213.5.5.5]/22
```

!! ACL modification !!

```
R5#show bgp ipv4 mvpn vrf N | include \[3\  
[no output]
```

```
R5#show ip mroute vrf N 232.0.0.7 10.5.10.10 verbose | begin \  
(10.5.10.10, 232.0.0.7), 00:19:29/00:03:11, flags: sTp  
  Incoming interface: GigabitEthernet2.550, RPF nbr 0.0.0.0  
  Outgoing interface list:  
    Lspvif0, LSM MDT: 2 (default), Forward/Sparse, 11:48:18/00:03:11, p
```

CSR6 and CSR8 also lose their data MDT cache information and strip the 'Y' from the C(S,G). Otherwise, the verification is generally the same. The red VPN (N) entries are still marked as 'E' for extranet, and the blue VPN (S) entries still use the red VPN for RPF lookups per the group-based selection policy.

```
R6#show ip mroute vrf N 232.0.0.7 10.5.10.10 | begin \  
(10.5.10.10, 232.0.0.7), 00:48:39/stopped, flags: sTE  
  Incoming interface: Lspvif10, RPF nbr 213.5.5.5  
  Outgoing interface list: Null
```

```
  Extranet receivers in vrf S:  
(10.5.10.10, 232.0.0.7), 21:17:59/00:02:55, OIF count: 1, flags: sTI
```

```
R6#show ip mroute vrf S 232.0.0.7 10.5.10.10 | begin \  
(10.5.10.10, 232.0.0.7), 21:18:03/00:02:51, flags: sTI  
  Incoming interface: Lspvif10, RPF nbr 213.5.5.5, using vrf N  
  Outgoing interface list:  
    GigabitEthernet2.546, Forward/Sparse, 21:18:03/00:02:51
```

```
R6#show ip pim vrf N mdt receive  
[no output]
```

```
R8#show ip mroute vrf N 232.0.0.7 10.5.10.10 | begin \  
(10.5.10.10, 232.0.0.7), 00:49:04/stopped, flags: sTE  
  Incoming interface: Lspvif1, RPF nbr 213.5.5.5  
  Outgoing interface list: Null
```

```
  Extranet receivers in vrf S:  
(10.5.10.10, 232.0.0.7), 12:15:12/00:02:52, OIF count: 1, flags: sTI
```

```
R8#show ip mroute vrf S 232.0.0.7 10.5.10.10 | begin \  
(10.5.10.10, 232.0.0.7), 12:15:12/00:02:52, flags: sTI  
  Incoming interface: Lspvif1, RPF nbr 213.5.5.5, using vrf N  
  Outgoing interface list:  
    GigabitEthernet2.538, Forward/Sparse, 12:15:12/00:02:52
```

```
R8#show ip pim vrf N mdt receive  
[no output]
```

We know that an mLDP default MDT always has MDT ID 0, so we can check the mLDP database to determine the downstream local labels. Also note that there are upstream labels in the MP2MP tree, though we don't care about that for this test. CSR6 uses label 6012 and CSR8 uses label 8011.

```
R6#show mpls mldp database opaque_type mdt 213:1512 0 | section Upstream
Upstream client(s) :
  213.13.13.13:0    [Active]
    Expires        : Never          Path Set ID   : 10
    Out Label (U)   : 93013          Interface    : GigabitEthernet2.563*
    Local Label (D) : 6012           Next Hop     : 213.6.13.13
```

```
R8#show mpls mldp database opaque_type mdt 213:1512 0 | section Upstream
Upstream client(s) :
  213.14.14.14:0    [Active]
    Expires        : Never          Path Set ID   : B
    Out Label (U)   : 94014          Interface    : GigabitEthernet2.584*
    Local Label (D) : 8011           Next Hop     : 213.8.14.14
```

CSR10 begins sending traffic. We will check the red/blue VPN counters on CSR6 and CSR8 as we always have. Additionally, we check CSR1 again to ensure intranet MVPN is functional.

```
R6#show ip mroute vrf N 232.0.0.7 10.5.10.10 count | begin ^Group
Group: 232.0.0.7, Source count: 1, Packets forwarded: 86, Packets received:
86
Source: 10.5.10.10/32, Forwarding: 86/1/122/0, Other: 86/0/0
```

```
R6#show ip mroute vrf S 232.0.0.7 10.5.10.10 count | begin ^Group
Group: 232.0.0.7, Source count: 1, Packets forwarded: 86, Packets received:
86
Source: 10.5.10.10/32, Forwarding: 86/1/122/0, Other: 86/0/0
```

```
R8#show ip mroute vrf N 232.0.0.7 10.5.10.10 count | begin ^Group
Group: 232.0.0.7, Source count: 1, Packets forwarded: 10, Packets received:
10
Source: 10.5.10.10/32, Forwarding: 10/1/122/0, Other: 10/0/0
```

```
R8#show ip mroute vrf S 232.0.0.7 10.5.10.10 count | begin ^Group
Group: 232.0.0.7, Source count: 1, Packets forwarded: 10, Packets received:
10
Source: 10.5.10.10/32, Forwarding: 10/1/122/0, Other: 10/0/0
```

```
R1#show ip mroute vrf N 232.0.0.7 10.5.10.10 count | begin ^Group
Group: 232.0.0.7, Source count: 1, Packets forwarded: 126, Packets received:
126
Source: 10.5.10.10/32, Forwarding: 126/1/122/0, Other: 126/0/0
```

Finally, we confirm proper operation using EPC on CSR6 and CSR8. CSR6 receives label 0x177C (6012) and CSR8 receives label 0x1F4B (8011), which is correct for the default MDT in the red VPN. For brevity, we use the IOS parser to tell us what the payload is for the IP packet.

```
R6#show monitor cap CAP buffer detailed
  1 122      0.241992 00:50:56:A9:EA:54 -> 00:50:56:A9:DE:0D MPLS unicast
0000: 005056A9 DE0D0050 56A9EA54 81000DEB .PV....PV..T....
0010: 88470177 C1FC4500 0064259C 0000FE01 .G.w..E..d%.....
0020: 9AE60A05 0A0AE800 00070800 26050016 .....&....
0030: 00AD0000 00003C47 1B3BABCD ABCDABCD .....<G.;.....

  2 118      0.241992 10.5.10.10 -> 232.0.0.7 ICMP
0000: 01005E00 00070050 56A9DE0D 81000DDA ..^....PV.....
0010: 08004500 0064259C 0000FB01 9DE60A05 ..E..d%.....
0020: 0A0AE800 00070800 26050016 00AD0000 .....&.....
0030: 00003C47 1B3BABCD ABCDABCD ABCDABCD ..<G.;.....
```

```
R8#show monitor cap CAP buffer detailed
  1 122      0.103984 00:50:56:A9:86:2A -> 00:50:56:A9:FB:1C MPLS unicast
0000: 005056A9 FB1C0050 56A9862A 81000E00 .PV....PV..*....
0010: 884701F4 B1FD4500 00642601 0000FE01 .G....E..d&.....
0020: 9A810A05 0A0AE800 00070800 9AC90016 .....
0030: 01120000 00003C48 A610ABCD ABCDABCD .....<H.....

  2 118      0.103984 10.5.10.10 -> 232.0.0.7 ICMP
0000: 01005E00 00070050 56A9FB1C 81000DD2 ..^....PV.....
0010: 08004500 00642601 0000FC01 9C810A05 ..E..d&.....
0020: 0A0AE800 00070800 9AC90016 01120000 .....
0030: 00003C48 A610ABCD ABCDABCD ABCDABCD ..<H.....
```

Because we are using LSM (and not raw IP), we can use MPLS OAM to help verify tunnel connectivity. We are using the default red MDT for transport, so we can test the MP2MP tree rooted at XRv4. We see replies from all routers except CSR7, since the 4 shown below are all part of the red VPN in some way. Even when using RSC, CSR6 and CSR8 will respond to these probes.

```
R5#ping mpls mldp mp2mp 213.14.14.14 mdt 213:1512 0
mp2mp Root node addr 213.14.14.14
Opaque type MDT, oui:index 0x213:01512, mdtnum 0
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
  timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]
```

Type escape sequence to abort.

```
Request #1
! reply addr 213.6.13.6
! reply addr 213.12.13.12
```

```
! reply addr 213.8.14.8
! reply addr 213.1.14.1
```

Next, we will test SSC using VRF S as the red VPN and VRF N as the blue VPN. CSR2 will be the source for (10.2.7.2, 232.0.0.5) with XRv2 and CSR1 as receivers. SSC means that CSR7 will be in both red and blue VPNs. This method is significantly more complex than RSC as CSR7 must import the blue RTs from all interested receivers (RT:213:1 and RT:213:12 in this case), and those receivers must reciprocate. CSR7's red VPN also imports routes from its local blue VPN to stitch the extranet together. This means CSR7's VRF N imports RT:213:7 also. Because the sources are red (S), they need to be advertised in the blue VPN so the egress PEs can satisfy RPF. This means CSR7's blue VPN needs to export an RT that the receivers are importing (can re-use RT:213:1 and RT:213:12) to pass those routes, or potentially use a "violet" RT if desired. In the GRE test, we re-used RT:213:1 and RT:213:12 for the BGP-AD routes and RT:213:700 for the VPNv4 routes, which appeared to work well. Because this method is so much more complicated than RSC, the snippet is shown below. CSR1 and XRv2 also need to import the violet RT:213:700, but that is not shown.

```
! CSR7
vrf definition N
  rd 213:77
  vpn id 213:1512
  route-target export 213:700
  route-target import 213:7
  route-target import 213:12
  route-target import 213:1
  address-family ipv4
    mdt auto-discovery mldp
    mdt default mpls mldp 213.14.14.14
  address-family ipv6
    mdt auto-discovery mldp
    mdt default mpls mldp 213.14.14.14
```

Using RT:213:700 for both BGP-AD and VPNv4 routes was an option I did not consider in earlier tests. The original reason for using the violet RT was because having the remote PEs import multiple I-PMSI routes from the same router allows BGP to decide which tunnel to use (based on best-path). This can have unintentional results as we are not trying to get CSR1/XRv2 to join the red VPN with SSC. Checking CSR1, we only see a single Type-1 I-PMSI route from CSR7 using the violet RT. The root is identified as XRv14 for the mLDP MP2MP (type 7) tree within VPN 213:1512. These fields are highlighted in the PMSI attributes below. We know this is the default MDT since the MD ID is 0 (0x00000000).

```
R1#show bgp ipv4 mvpn vrf N route-type 1 213.7.7.7
BGP routing table entry for [1][213:1][213.7.7.7]/12, version 56
Paths: (1 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGP
peer)
  Not advertised to any peer
  Refresh Epoch 1
```

```

Local, imported path from [1][213:77][213.7.7.7]/12 (global)
 213.7.7.7 (metric 20) from 213.12.12.12 (213.12.12.12)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  Community: no-export
  Extended Community: RT:213:700
  Originator: 213.7.7.7, Cluster list: 213.12.12.12
  PMSI Attribute: Flags: 0x0, Tunnel type: 7, length 24, label: exp-null,
tunnel parameters: 0800 0104 D50E 0E0E 000E 0200 0B00 0213 0000 1512 0000
0000
  rx pathid: 0, tx pathid: 0x0

```

The default MDT for VRF N is huge now since the RSC configuration is still intact. Nonetheless, we can see CSR7 is a new neighbor from CSR1's and XRv2's perspective, which shows it was recently added. Technically, this is part of the I-PMSI setup, which wasn't really an issue since CSR7 generated its own I-PMSI route with the violet RT:213:700 from the beginning.

```

R1#show ip pim vrf N neighbor | begin ^Neighbor
Neighbor          Interface          Uptime/Expires    Ver    DR
Address                                     Prio/Mode
213.7.7.7          Lspvif0          00:08:09/00:01:27 v2      1 / B S P G
213.8.8.8          Lspvif0          00:55:51/00:01:28 v2      1 / B S P G
213.6.6.6          Lspvif0          00:55:51/00:01:26 v2      1 / B S P G
213.12.12.12       Lspvif0          13:37:01/00:01:17 v2      1 / DR B P G
213.5.5.5          Lspvif0          14:18:56/00:01:27 v2      1 / B S P G

RP/0/0/CPU0:XRv2#show pim vrf N neighbor | begin ^Neighbor
Neighbor Address          Interface          Uptime    Expires    DR pri
Flags
10.11.12.11          GigabitEthernet0/0/0/0.512 13:40:36  00:01:41  1          B P
10.11.12.12*          GigabitEthernet0/0/0/0.512 13:40:41  00:01:26  1 (DR)    B P E
213.1.1.1             LmdtN              13:37:33  00:01:23  1          P
213.5.5.5             LmdtN              13:37:33  00:01:25  1          P
213.6.6.6             LmdtN              00:56:23  00:01:24  1          P
213.7.7.7             LmdtN              00:08:41  00:01:25  1          P
213.8.8.8             LmdtN              00:56:23  00:01:25  1          P
213.12.12.12*         LmdtN              13:37:39  00:01:15  1 (DR)    P

```

Normally I don't document my mistakes in detail, but this configuration is broken in many ways. Most of these errors come from trying to use S-PMSI with SSC incorrectly. Looking at CSR7, the issue is clear. Within VRF N, CSR7 selects the locally originated I-PMSI route over the imported one; the best-path has RT:213:700 which allows blue receivers to import it. However, there is only one S-PMSI route, and it is the imported one. VRF N did not create one locally, so this RT is still 213:7. Remote receivers cannot import this per the current design.

```

R7#show bgp ipv4 mvpn vrf N route-type 1 213.7.7.7
[snip]
Local

```

```

0.0.0.0 from 0.0.0.0 (213.7.7.7)
  Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
  best
  Community: no-export
  Extended Community: RT:213:700
  PMSI Attribute: Flags: 0x0, Tunnel type: 7, length 24, label: exp-null,
  tunnel parameters: 0800 0104 D50E 0E0E 000E 0200 0B00 0213 0000 1512 0000
  0000
  rx pathid: 0, tx pathid: 0x0
Refresh Epoch 1
Local, imported path from [1][213:7][213.7.7.7]/12 (S)
  0.0.0.0 from 0.0.0.0 (213.7.7.7)
  Origin incomplete, localpref 100, weight 32768, valid, external
  Community: no-export
  Extended Community: RT:213:7
  PMSI Attribute: Flags: 0x0, Tunnel type: 7, length 24, label: exp-null,
  tunnel parameters: 0800 0104 D50D 0D0D 000E 0200 0B00 0213 0000 0678 0000
  0000
  rx pathid: 0, tx pathid: 0

R7#show bgp ipv4 mvpn vrf N route-type 3 10.2.7.2 232.0.0.5 213.7.7.7
[snip]
Local, imported path from [3][213:7][10.2.7.2][232.0.0.5][213.7.7.7]/22 (S)
  0.0.0.0 from 0.0.0.0 (213.7.7.7)
  Origin incomplete, localpref 100, weight 32768, valid, external, best
  Community: no-export
  Extended Community: RT:213:7
  PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 24, label: exp-null,
  tunnel parameters: 0600 0104 D507 0707 000E 0200 0B00 0213 0000 0678 0000
  0001
  rx pathid: 0, tx pathid: 0x0

```

Also, the unicast VPNv4 route used by the blue egress PEs for RPF has RT:213:7 also, as it was imported and not locally originated. This is not imported by CSR1 or XRv2 as a result.

```

R7#show bgp vpnv4 unicast vrf N 10.2.7.0/24
BGP routing table entry for 213:77:10.2.7.0/24, version 9
Paths: (1 available, best #1, table N)
  Not advertised to any peer
Refresh Epoch 1
Local, imported path from 213:7:10.2.7.0/24 (S)
  0.0.0.0 (via vrf S) (via S) from 0.0.0.0 (213.7.7.7)
  Origin incomplete, metric 0, localpref 100, weight 32768, valid,
  external, best
  Extended Community: RT:213:7
  mpls labels in/out nolabel/nolabel(N)
  rx pathid: 0, tx pathid: 0x0

```


This is why CSR1 does not have the S-PMSI route, the data MDT cache information, or valid RPF/C-MRIB information for this C(S,G).

```
R1#show bgp ipv4 mvpn vrf N route-type 3 10.2.7.2 232.0.0.5 213.7.7.7
```

```
% Network not in table
```

```
R1#show ip pim vrf N mdt receive
```

```
[no output]
```

```
R1#show ip route vrf N 10.2.7.0
```

```
Routing Table: N
```

```
% Subnet not in table
```

```
R1#show ip mroute vrf N 232.0.0.5 10.2.7.2 | begin \((
```

```
(10.2.7.2, 232.0.0.5), 20:26:46/00:02:15, flags: sTI
```

```
  Incoming interface: Null, RPF nbr 0.0.0.0
```

```
  Outgoing interface list:
```

```
    GigabitEthernet2.519, Forward/Sparse, 20:26:46/00:02:15
```

The GRE tests did not explicitly test S-PMSI with SSC, so we didn't see this issue earlier. The BGP Type-1 route is already being imported properly, but the Type-3 and VPNv4 routes are not. This explains why we have C-PIM neighbors but not much else. Using the technique from the GRE tests of adding RT:213:700 to the VPNv4/v6 routes will at least correct the RPF problem. I show a portion of the configuration below, since the prefix-lists inside just match the C-Source.

```
! CSR7
```

```
vrf definition S
```

```
  address-family ipv4
```

```
    export map RM_EXTRANET_EXPORT
```

```
  address-family ipv6
```

```
    export map RM_EXTRANET_EXPORT_IPV6
```

```
R1#show ip rpf vrf N 10.2.7.2
```

```
RPF information for ? (10.2.7.2)
```

```
  RPF interface: Lspvif0
```

```
  RPF neighbor: ? (213.7.7.7)
```

```
  RPF route/mask: 10.2.7.0/24
```

```
  RPF type: unicast (bgp 213)
```

```
  Doing distance-preferred lookups across tables
```

```
  RPF topology: ipv4 multicast base, originated from ipv4 unicast base
```

Within XE/XR, I cannot find a mechanism to selectively match PMSI routes. If I could, some BGP route-maps would be a tempting solution to this problem. Even by setting the RT of the C-source as shown above, this was not transitive to the Type-3 S-PMSI route.

```
R7#show bgp vpnv4 unicast vrf N 10.2.7.0/24 | include Extend
```

```
Extended Community: RT:213:7 RT:213:700
```

```
R7#show bgp ipv4 mvpn vrf N route-type 3 10.2.7.2 232.0.0.5 213.7.7.7 |  
include Extend
```

```
Extended Community: RT:213:7
```

Even if it were transitive, the S-PMSI route still includes the red VPN ID, which is incorrect. Even if we manipulated the RT somehow (which is possible on XRv ... match RD 213:77, set RT 213:700), it would not allow the blue endpoints to build the proper VPN. **The key point is that the VPN membership is determined by more than just the RT; other mLDP FEC elements matter also.** We saw this earlier when we tried to do RSC without configuring the red VPN on the blue PE. mLDP is more strict about VPN membership than GRE.

```
R7#show bgp ipv4 mvpn vrf N route-type 3 10.2.7.2 232.0.0.5 213.7.7.7 |  
include PMSI
```

```
PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 24, label: exp-null,  
tunnel parameters: 0600 0104 D507 0707 000E 0200 0B00 0213 0000 0678 0000  
0001
```

The key to this puzzle is configuring CSR7 to originate its own S-PMSI route. That guarantees that it will have the proper RT and VPN ID values. This is as simple as configuring the data MDTs inside the proper AF (IPv4/v6) under the blue VPN (N) on CSR7. Just like for the Type-1 I-PMSI route, the locally originated route was preferred over the imported one, so it can be advertised to the RR. Notice the correct RT and VPN ID.

```
! CSR7  
vrf definition N  
  address-family ipv4  
    mdt data mpls mldp 10 list ACL_DATA_MDT immediate-switch
```

```
R7#show bgp ipv4 mvpn vrf N route-type 3 10.2.7.2 232.0.0.5 213.7.7.7  
BGP routing table entry for [3][213:77][10.2.7.2][232.0.0.5][213.7.7.7]/22,  
version 14
```

```
Paths: (2 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGp  
peer)
```

```
  Advertised to update-groups:
```

```
    3
```

```
  Refresh Epoch 1
```

```
  Local
```

```
    0.0.0.0 from 0.0.0.0 (213.7.7.7)
```

```
      Origin incomplete, localpref 100, weight 32768, valid, sourced, local,  
best
```

```
      Community: no-export
```

```
      Extended Community: RT:213:700
```

```
      PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 24, label: exp-null,  
tunnel parameters: 0600 0104 D507 0707 000E 0200 0B00 0213 0000 1512 0000  
0001
```

```

    rx pathid: 0, tx pathid: 0x0
Refresh Epoch 1
Local, imported path from [3][213:7][10.2.7.2][232.0.0.5][213.7.7.7]/22 (S)
0.0.0.0 from 0.0.0.0 (213.7.7.7)
    Origin incomplete, localpref 100, weight 32768, valid, external
    Community: no-export
    Extended Community: RT:213:7
    PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 24, label: exp-null,
tunnel parameters: 0600 0104 D507 0707 000E 0200 0B00 0213 0000 0678 0000
0001
    rx pathid: 0, tx pathid: 0

```

Now, we can continue with our standard verification. CSR1 has a populated MDT cache and C-MRIB entry. The mLDP process allocates downstream local label 1012 for this data MDT. The FEC root (CSR7) and tree type (P2MP) are both correct. The big 'Y' shows that data is expected to enter on the data MDT and the MDT information is embedded in the C-MRIB entry.

```

R1#show ip pim vrf N mdt receive
Joined MDT-data [group/mdt number : source]  uptime/expires for VRF: N
[1 : 213.7.7.7]  00:05:01/stopped

R1#show ip mroute vrf N 232.0.0.5 10.2.7.2 | begin \
(10.2.7.2, 232.0.0.5), 21:54:33/00:02:32, flags: sTIY
    Incoming interface: Lspvif0, RPF nbr 213.7.7.7, MDT: [1, 213.7.7.7]/never
    Outgoing interface list:
        GigabitEthernet2.519, Forward/Sparse, 21:54:33/00:02:32

R1#show mpls mldp database opaque_type mdt 213:1512 1
LSM ID : 9   Type: P2MP   Uptime : 00:05:02
FEC Root      : 213.7.7.7
Opaque decoded : [mdt 213:1512 1]
Opaque length  : 11 bytes
Opaque value   : 02 000B 0002130000151200000001
Upstream client(s) :
    213.14.14.14:0   [Active]
        Expires      : Never           Path Set ID   : 9
        Out Label (U) : None           Interface    : GigabitEthernet2.514*
        Local Label (D) : 1012         Next Hop     : 213.1.14.14
Replication client(s):
    MDT (VRF N)
        Uptime       : 00:05:02       Path Set ID   : None
        Interface    : Lspvif0

```

The verification for XRv2 is similar, but since data-plane LSM MVPN does not appear supported, there are some software issues with resolving RPF, I am assuming. I go into some extra detail to verify RPF and unicast routing is correct; we can clearly see the control-plane for unicast VPNv6 is correct, to include a 2-label stack.

```
RP/0/0/CPU0:XRv2#show pim vrf N mdt cache
Core Source      Cust (Source, Group)      Core Data      Expires
213.7.7.7        (10.2.7.2, 232.0.0.5)      [mdt 213:1512 1]  never
```

```
RP/0/0/CPU0:XRv2#show pim vrf N topology 232.0.0.5 10.2.7.2 | begin 232
(10.2.7.2,232.0.0.5)SPT SSM Up: 15:30:02
JP: Join(00:00:49) RPF: Null,0.0.0.0 Flags:
GigabitEthernet0/0/0/0.512 15:30:02 fwd LI LH
```

```
RP/0/0/CPU0:XRv2#show pim vrf N rpf 10.2.7.2
Table: IPv4-Unicast-default
* 10.2.7.2/32 [200/0]
via Null with rpf neighbor 213.7.7.7
```

```
RP/0/0/CPU0:XRv2#show cef vrf N ipv4 10.2.7.2
10.2.7.0/24, version 56, internal 0x5000001 0x0 (ptr 0xa142dff4) [1], 0x0
(0x0), 0x208 (0xa156d438)
Prefix Len 24, traffic index 0, precedence n/a, priority 3
via 213.7.7.7, 5 dependencies, recursive [flags 0x6000]
path-idx 0 NHID 0x0 [0xa15d56f4 0x0]
recursion-via-/32
next hop VRF - 'default', table - 0xe0000000
next hop 213.7.7.7 via 92003/0/21
next hop 213.12.13.13/32 Gi0/0/0/0.523 labels imposed {93003 7007}
```

Next, we verify the head end where SSC is configured. The sequence of verification is reversed from RSC: first we check the red VPN (S), then the extranet proxy (N), then the core path (mLDP). Be careful of the two different LSM-IDs. 0x7 represents the ID within VRF S, which services the intranet VPN members. LSM-ID 0x8 is within VRF N which is for the extranet members. Remember that SSC replicates traffic for each extranet configured. That is why the OIL for the red VPN (S) entry is not null as it was for RSC.

```
R7#show ip mroute vrf S 232.0.0.5 10.2.7.2 verbose | begin \
(10.2.7.2, 232.0.0.5), 01:50:39/00:03:13, flags: sTyp
Incoming interface: GigabitEthernet2.527, RPF nbr 0.0.0.0
MDT TX nr: 1 LSM-ID: 0x7
Outgoing interface list:
Lspvif0, LSM MDT: 7 (data), Forward/Sparse, 01:11:27/00:03:13, p
Extranet receivers in vrf N:
(10.2.7.2, 232.0.0.5), 01:13:03/00:02:44, OIF count: 1, flags: sTyp
```

```
R7#show ip mroute vrf N 232.0.0.5 10.2.7.2 verbose | begin \
(10.2.7.2, 232.0.0.5), 01:13:02/00:02:45, flags: sTyp
Incoming interface: GigabitEthernet2.527, RPF nbr 0.0.0.0, using vrf S
MDT TX nr: 1 LSM-ID: 0x8
Outgoing interface list:
Lspvif1, LSM MDT: 8 (data), Forward/Sparse, 01:11:26/00:02:45, p
```

Below is a summary view of the LSM-IDs and their VPNs. This helps confirm the output seen in the C-MRIB entries. Looking at 0x8, the extranet delivery tree, we see the router is only sending traffic towards XRv4 in terms of LSM. I assume this is because XRv2 has RPF issues and cannot issue the C-PIM (S,G).

```
R7#show mpls mldp database summary | include 7.7.7
7          P2MP      213.7.7.7          [mdt 213:678 1]          3
8          P2MP      213.7.7.7          [mdt 213:1512 1]        2
```

```
R7#show mpls mldp database id 8 | section Replica
Replication client(s):
MDT (VRF N)
  Uptime          : 00:17:02          Path Set ID   : None
  Interface       : Lspvif1
213.14.14.14:0
  Uptime          : 00:17:02          Path Set ID   : None
  Out label (D)   : 94016             Interface    : GigabitEthernet2.574*
  Local label (U) : None              Next Hop     : 213.7.14.14
```

When XRv4 receives MPLS packets with label 94016, it forwards them to CSR1 only, as this is a data MDT within an extranet. Label 1012 is what CSR1 sent to XRv4 (in a label-mapping message) to “join” the extranet LSM tree.

```
RP/0/0/CPU0:XRv4#show mpls forwarding labels 94016
Local   Outgoing   Prefix           Outgoing         Next Hop         Bytes
Label   Label      or ID           Interface        Next Hop         Switched
-----
94016   1012          MLDP: 0x00007   Gi0/0/0/0.514   213.1.14.1       0
```

For completeness, we will look at the intranet labels also. This is within the red VPN and is not related to extranet. Because there are intranet customers on both sides of the network (CSR6 and CSR8), CSR7 replicates the traffic towards XRv3 and XRv4. The labels used are 93011 and 94015.

```
R7#show mpls mldp database id 7 | section Replica
Replication client(s):
MDT (VRF S)
  Uptime          : 01:20:58          Path Set ID   : None
  Interface       : Lspvif0
213.13.13.13:0
  Uptime          : 01:20:57          Path Set ID   : None
  Out label (D)   : 93011             Interface    : GigabitEthernet2.573*
  Local label (U) : None              Next Hop     : 213.7.13.13
213.14.14.14:0
  Uptime          : 01:20:57          Path Set ID   : None
  Out label (D)   : 94015             Interface    : GigabitEthernet2.574*
  Local label (U) : None              Next Hop     : 213.7.14.14
```

Next, CSR2 begins sending traffic. We expect that CSR1 will be able to receive it properly, as will the existing intranet customers CSR6 and CSR8. First, we verify that CSR7 is seeing identical packet counts on both the red and blue VPN C-MRIB entries.

```
R7#show ip mroute vrf S 232.0.0.5 10.2.7.2 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 28, Packets received:
28
Source: 10.2.7.2/32, Forwarding: 28/1/118/0, Other: 28/0/0
```

```
R7#show ip mroute vrf N 232.0.0.5 10.2.7.2 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 28, Packets received:
28
Source: 10.2.7.2/32, Forwarding: 28/1/118/0, Other: 28/0/0
```

An EPC capture on the outgoing interfaces shows the replication occurring. Two MPLS packets are sent towards XRv4 with labels 94015 (red color and VPN) and 94016 (blue color and VPN). One MPLS packet is sent towards XRv3 with label 93011 (yellow color, but is for the red VPN).

```
R7#show monitor capture CAP buffer detailed
 2 122    0.677974 00:50:56:A9:EA:77 -> 00:50:56:A9:EA:54 MPLS unicast
0000: 005056A9 EA540050 56A9EA77 81000DF5 .PV..T.PV..w....
0010: 884716B5 31FE4500 00646365 0000FE01 .G...1.E..dce....
0020: 602A0A02 0702E800 00050800 9BA1001E `*.....
0030: 00E10000 00003CD3 A4D6ABCD ABCDABCD .....<.....

 3 122    0.677974 00:50:56:A9:EA:77 -> 00:50:56:A9:86:2A MPLS unicast
0000: 005056A9 862A0050 56A9EA77 81000DF6 .PV...*.PV..w....
0010: 884716F3 F1FE4500 00646365 0000FE01 .G....E..dce....
0020: 602A0A02 0702E800 00050800 9BA1001E `*.....
0030: 00E10000 00003CD3 A4D6ABCD ABCDABCD .....<.....

 4 122    0.677974 00:50:56:A9:EA:77 -> 00:50:56:A9:86:2A MPLS unicast
0000: 005056A9 862A0050 56A9EA77 81000DF6 .PV...*.PV..w....
0010: 884716F4 01FE4500 00646365 0000FE01 .G....E..dce....
0020: 602A0A02 0702E800 00050800 9BA1001E `*.....
0030: 00E10000 00003CD3 A4D6ABCD ABCDABCD .....<.....
```

Next, we verify packet counters on the remote PEs, both extranet and intranet. We do not verify XRv2 since the control-plane was broken (no possibility of data-plane working).

```
R1#show ip mroute vrf N 232.0.0.5 10.2.7.2 count | begin ^Group
Group: 232.0.0.5, Source count: 1, Packets forwarded: 456, Packets received:
456
Source: 10.2.7.2/32, Forwarding: 456/1/122/0, Other: 456/0/0
```

```
R8#show ip mroute vrf S 232.0.0.5 10.2.7.2 count | begin ^Group
```

```
Group: 232.0.0.5, Source count: 1, Packets forwarded: 426, Packets received: 426
```

```
Source: 10.2.7.2/32, Forwarding: 426/1/122/0, Other: 426/0/0
```

```
R6#show ip mroute vrf S 232.0.0.5 10.2.7.2 count | begin ^Group
```

```
Group: 232.0.0.5, Source count: 1, Packets forwarded: 429, Packets received: 429
```

```
Source: 10.2.7.2/32, Forwarding: 429/1/122/0, Other: 429/0/0
```

Using EPC, we verify proper extranet connectivity on CSR1. MPLS packets arrive with label 0x3F4 (1012) and are immediately decapsulated and sent towards CSR9 inside the blue VPN (N).

```
R1#show monitor capture CAP buffer detailed
```

```
0 122 0.000000 00:50:56:A9:86:2A -> 00:50:56:A9:1A:AA MPLS unicast
0000: 005056A9 1AAA0050 56A9862A 81000DBA .PV....PV..*....
0010: 8847003F 41FD4500 0064646D 0000FE01 .G.?A.E..ddm....
0020: 5F220A02 0702E800 00050800 929B001E _".....
0030: 01E90000 00003CD7 ACD0ABCD ABCDABCD .....<.....
```

```
1 118 0.000000 10.2.7.2 -> 232.0.0.5 ICMP
0000: 01005E00 00050050 56A91AAA 81000DBF ..^....PV.....
0010: 08004500 0064646D 0000FC01 61220A02 ..E..ddm....a"..
0020: 0702E800 00050800 929B001E 01E90000 .....
0030: 00003CD7 ACD0ABCD ABCDABCD ABCDABCD ..<.....
```

A quick OAM check on CSR7 shows everyone connected in the S-PMSI for this group. The verification is a little different with SSC since there are technically two P2MP trees as we saw earlier. When testing the extranet tree (blue VPN), only CSR1 responds. When testing the intranet tree (red VPN), CSR6 and CSR8 respond. The trees have the same root and, in this case, the same MDT ID. The major differentiator is the VPN ID. This is the expected and correct behavior.

```
R7#ping mpls mldp p2mp 213.7.7.7 mdt 213:1512 1
```

```
p2mp Root node addr 213.7.7.7
```

```
Opaque type MDT, oui:index 0x213:01512, mdtnum 1
```

```
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
```

```
timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200 msec:
```

```
[snip]
```

```
Type escape sequence to abort.
```

```
Request #1
```

```
! reply addr 213.1.14.1
```

```
R7#ping mpls mldp p2mp 213.7.7.7 mdt 213:678 1
```

```
p2mp Root node addr 213.7.7.7
```

```
Opaque type MDT, oui:index 0x213:0678, mdtnum 1
```

```
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,  
    timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200  
msec:  
[snip]
```

Type escape sequence to abort.

```
Request #1  
! reply addr 213.6.13.6  
! reply addr 213.8.14.8
```

Because the delivery mechanism is identical for IPv6, we will quickly evaluate it. The software plumbing appears to be broken; in all other examples, including the PIM IPv6 examples, the red VPN had extranet entries to connect the LSP to the blue VPN. In this case, it doesn't, but the RPF resolution from our group-based selection process is working. This is based on an RSC model.

```
R6#show ipv6 mroute vrf N  
No mroute entries found.
```

```
R6#show ipv6 mroute vrf S ff33::2 2001:10:5:10::10 | begin \  
(2001:10:5:10::10, FF33::2), 00:11:07/never, flags: sTI  
Incoming interface: Lspvif0  
RPF nbr: ::FFFF:213.5.5.5  
Immediate Outgoing interface list:  
GigabitEthernet2.546, Forward, 00:11:07/never
```

RPF validation appears somewhat correct, except the LSPvif interface enumerations don't match. The data MDT information doesn't appear to have been transported, either.

```
R6#show ipv6 rpf vrf S select  
Multicast Group-to-Vrf Mappings  
Group(s): FF33::2/128, RPF vrf: N, Acl: ACL_N_GROUPS_IPV6
```

```
R6#show ipv6 rpf vrf N 2001:10:5:10::10  
RPF information for 2001:10:5:10::10  
RPF interface: Lspvif1  
RPF neighbor: ::FFFF:213.5.5.5  
RPF route/mask: 2001:10:5:10::/64  
RPF type: Unicast  
RPF recursion count: 0  
Metric preference: 200  
Metric: 0
```

```
R6#show ipv6 pim vrf N mdt receive  
[no output]
```


All of the necessary BGP-AD and VPNv6 routes are inside VRF N, as they should be. The C-PIM neighbors are also functioning, which seems to indicate an operational default MDT.

```
R6#show bgp ipv6 mvpn vrf N | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 213:66 (default for vrf N)
*>i [1] [213:66] [213.5.5.5]/12
      213.5.5.5      0      100      0 ?
*>i [3] [213:66] [2001:10:5:10::10] [FF33::2] [213.5.5.5]/46
      213.5.5.5      0      100      0 ?
```

```
R6#show bgp vpnv6 unicast vrf N | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 213:66 (default for vrf N)
*>i 2001:10:5:10::/64
      ::FFFF:213.5.5.5
      0      100      0 ?
```

```
R6#show ipv6 pim vrf N neighbor
PIM Neighbor Table
Mode: B - Bidir Capable, G - GenID Capable
Neighbor Address      Interface      Uptime      Expires      Mode DR pri
::FFFF:213.1.1.1      Lspvif1      00:14:45    00:01:19    B G      1
::FFFF:213.5.5.5      Lspvif1      00:14:45    00:01:20    B G      1
::FFFF:213.8.8.8      Lspvif1      00:14:46    00:01:19    B G      1
::FFFF:213.12.12.12   Lspvif1      00:14:45    00:01:35    B G    DR 1
```

There is no point in sending traffic from CSR10 since we know the P2MP trees won't be built. If it were, CSR6 would show a P2MP tree rooted at 213.5.5.5 with the VPN ID 213:1512 inside the opaque. Even a router reload on CSR6 and CSR8 (to potentially fix any order-of-operations issues with the configuration) did not resolve the issue.

```
R6#show mpls mldp database summary
LSM ID      Type      Root      Decoded Opaque Value      Client
Cnt.
5           P2MP      213.7.7.7      [mdt 213:678 1]      1
1           MP2MP     213.13.13.13   [mdt 213:678 0]      1
3           MP2MP     213.14.14.14   [mdt 213:1512 0]     1
```

However, IPv6 appears to work somewhat with SSC (doesn't actually work, but control-plane looks better). There is no S-PMSI offered for (2001:10:2:7::2, FF33::1) at present, but we can verify the extranet connectivity using the default MDT. CSR7 originates an IPv6 Type-1 I-PMSI route within the blue VPN (N) and advertises it to CSR1. We know this is for the blue VPN since the VPN ID is 213:1512. CSR1 is able to install this and join the default MDT. We can see the C-PIM neighbor with CSR7. We can

also see the tree is MP2MP (type 7) and is rooted at XRv4. Notice the correct “violet” RT:213:700; this is because the route was originated locally from CSR7 inside VRF N.

```
R1#show bgp ipv6 mvpn vrf N route-type 1 213.7.7.7
BGP routing table entry for [1][213:1][213.7.7.7]/12, version 100
Paths: (1 available, best #1, table MVPNV6-BGP-Table, not advertised to EBGp
peer)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from [1][213:77][213.7.7.7]/12 (global)
    213.7.7.7 (metric 20) from 213.12.12.12 (213.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Community: no-export
      Extended Community: RT:213:700
      Originator: 213.7.7.7, Cluster list: 213.12.12.12
      PMSI Attribute: Flags: 0x0, Tunnel type: 7, length 24, label: exp-null,
tunnel parameters: 0800 0104 D50E 0E0E 000E 0200 0B00 0213 0000 1512 0000
0000
      rx pathid: 0, tx pathid: 0x0
```

```
R1#show ipv6 pim vrf N neighbor | include 7.7.7
::FFFF:213.7.7.7          Lspvif0          00:38:11  00:01:37 B G      1
```

Checking the C-MRIB entries on CSR7, we see traffic arriving from VRF S, being forwarded to VRF N (big ‘E’ for extranet), then being encapsulated inside MPLS.

```
R7#show ipv6 mroute vrf S FF33::1 2001:10:2:7::2 | begin \
(2001:10:2:7::2, FF33::1), 02:24:20/00:03:11, flags: sTE
Incoming interface: GigabitEthernet2.527
RPF nbr: 2001:10:2:7::2
Immediate Outgoing interface list:
  Lspvif0, Forward, 02:24:20/00:03:11

Extranet receivers in vrf N, OIF count 1
```

```
R7#show ipv6 mroute vrf N FF33::1 2001:10:2:7::2 | begin \
(2001:10:2:7::2, FF33::1), 00:40:44/00:02:47, flags: sT
Incoming interface: GigabitEthernet2.527
RPF nbr: 2001:10:2:7::2, using vrf S
Immediate Outgoing interface list:
  Lspvif1, Forward, 00:40:44/00:02:47
```

Notice that the two entries have different LSPvif interfaces; the one in VRF S is for red receivers (intranet) and the one in vrf N is for blue receivers (extranet). Two outputs are shown below; the first is for intranet and the second is for extranet. These are the two default MDTs for the red and blue VPNs respectively. We note the upstream labels for each since CSR7 will be replicating traffic for each VPN in this way. Unlike for IPv4, we will later see two packets, not three, since we are using the MP2MP tree.

XRv4 will perform any downstream replication needed as CSR7's only job is to get the traffic to the MP2MP root.

```
R7#show mpls mldp database opaque_type mdt 213:678 0
LSM ID : 1 (RNR LSM ID: 2)   Type: MP2MP   Uptime : 04:04:11
FEC Root      : 213.13.13.13
Opaque decoded : [mdt 213:678 0]
Opaque length  : 11 bytes
Opaque value   : 02 000B 0002130000067800000000
RNR active LSP : (this entry)
Upstream client(s) :
  213.13.13.13:0 [Active]
    Expires      : Never           Path Set ID   : 1
    Out Label (U) : 93010          Interface    : GigabitEthernet2.573*
    Local Label (D) : 7009          Next Hop      : 213.7.13.13
Replication client(s):
  MDT (VRF S)
    Uptime       : 04:04:11       Path Set ID   : 2
    Interface    : Lspvif0
```

```
R7#show mpls mldp database opaque_type mdt 213:1512 0
LSM ID : C (RNR LSM ID: D)   Type: MP2MP   Uptime : 00:46:19
FEC Root      : 213.14.14.14
Opaque decoded : [mdt 213:1512 0]
Opaque length  : 11 bytes
Opaque value   : 02 000B 0002130000151200000000
RNR active LSP : (this entry)
Upstream client(s) :
  213.14.14.14:0 [Active]
    Expires      : Never           Path Set ID   : C
    Out Label (U) : 94009          Interface    : GigabitEthernet2.574*
    Local Label (D) : 7011          Next Hop      : 213.7.14.14
Replication client(s):
  MDT (VRF N)
    Uptime       : 00:46:19       Path Set ID   : D
    Interface    : Lspvif1
```

CSR1 also reports a valid C(S,G) entry for this flow. Notice there are no 'y' or 'Y' flags as this is using the I-PMSI P-tunnel, which is mLDP MP2MP. Because we know the characteristics of the blue default MDT, we can query the mLDP database to find the incoming local label. That value is 1009.

```
R1#show ipv6 mroute vrf N FF33::1 2001:10:2:7::2 | begin \
(2001:10:2:7::2, FF33::1), 23:15:33/never, flags: sTI
Incoming interface: Lspvif0
RPF nbr: ::FFFF:213.7.7.7
Immediate Outgoing interface list:
  GigabitEthernet2.519, Forward, 23:15:33/never
```

```
R1#show mpls mldp database opaque_type mdt 213:1512 0 | section Upstream
Upstream client(s) :
  213.14.14.14:0      [Active]
    Expires           : Never           Path Set ID   : 1
    Out Label (U)     : 94007           Interface    : GigabitEthernet2.514*
    Local Label (D)   : 1009           Next Hop     : 213.1.14.14
```

Next, we start sending traffic on CSR2. We check the C-MRIB counters on CSR7 to ensure traffic is being sent into the MPLS core as LSM.

```
R7#show ipv6 mroute vrf S FF33::1 2001:10:2:7::2 count | begin ^Group
Group: FF33::1
  Source: 2001:10:2:7::2,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 8/0/118/0, Other: 0/0/0
  Totals - Source count: 1, Packet count: 8
```

```
R7#show ipv6 mroute vrf N FF33::1 2001:10:2:7::2 count | begin ^Group
Group: FF33::1
  Source: 2001:10:2:7::2,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 8/0/118/0, Other: 0/0/0
  Totals - Source count: 1, Packet count: 8
```

Quickly checking the packet counters on all receivers, intranet and extranet, confirms broken extranet operation. Packets arrive as expected to the intranet receivers, but not to the extranet ones.

```
R6#show ipv6 mroute vrf S FF33::1 2001:10:2:7::2 count | begin ^Group
Group: FF33::1
  Source: 2001:10:2:7::2,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 401/1/126/0, Other: 0/0/0
  Totals - Source count: 1, Packet count: 401
```

```
R8#show ipv6 mroute vrf S FF33::1 2001:10:2:7::2 count | begin ^Group
Group: FF33::1
  Source: 2001:10:2:7::2,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 387/1/126/0, Other: 0/0/0
  Totals - Source count: 1, Packet count: 387
```

```
R1#show ipv6 mroute vrf N FF33::1 2001:10:2:7::2 count | begin ^Group
Group: FF33::1
  Source: 2001:10:2:7::2,
    SW Forwarding: 0/0/0/0, Other: 0/0/0
    HW Forwarding: 0/0/0/0, Other: 0/0/0
  Totals - Source count: 1, Packet count: 0
```

We already verified the control-plane on CSR7, but EPC clearly shows that CSR7 is only sending traffic upstream to XRv3 along the red default MDT. Three packets, paced one second apart, all have the same label 93010 (0x16B52) and the destination MAC of XRv3.

```
R7#show ip arp 213.7.13.13
```

Protocol	Address	Age (min)	Hardware Addr	Type	Interface
Internet	213.7.13.13	37	0050.56a9.ea54	ARPA	Giga2.573

```
R7#show monitor capture CAP buffer detailed
```

```

0 126 0.000000 00:50:56:A9:EA:77 -> 00:50:56:A9:EA:54 MPLS unicast
0000: 005056A9 EA540050 56A9EA77 81000DF5 .PV..T.PV..w....
0010: 884716B5 203F0000 213F6000 0000003C .G.. ?...!?'`....<
0020: 3A3F2001 00100002 00070000 00000000 :? .....
0030: 0002FF33 00000000 00000000 00000000 ...3.....

1 126 1.000000 00:50:56:A9:EA:77 -> 00:50:56:A9:EA:54 MPLS unicast
0000: 005056A9 EA540050 56A9EA77 81000DF5 .PV..T.PV..w....
0010: 884716B5 203F0000 213F6000 0000003C .G.. ?...!?'`....<
0020: 3A3F2001 00100002 00070000 00000000 :? .....
0030: 0002FF33 00000000 00000000 00000000 ...3.....

2 126 2.001007 00:50:56:A9:EA:77 -> 00:50:56:A9:EA:54 MPLS unicast
0000: 005056A9 EA540050 56A9EA77 81000DF5 .PV..T.PV..w....
0010: 884716B5 203F0000 213F6000 0000003C .G.. ?...!?'`....<
0020: 3A3F2001 00100002 00070000 00000000 :? .....
0030: 0002FF33 00000000 00000000 00000000 ...3.....

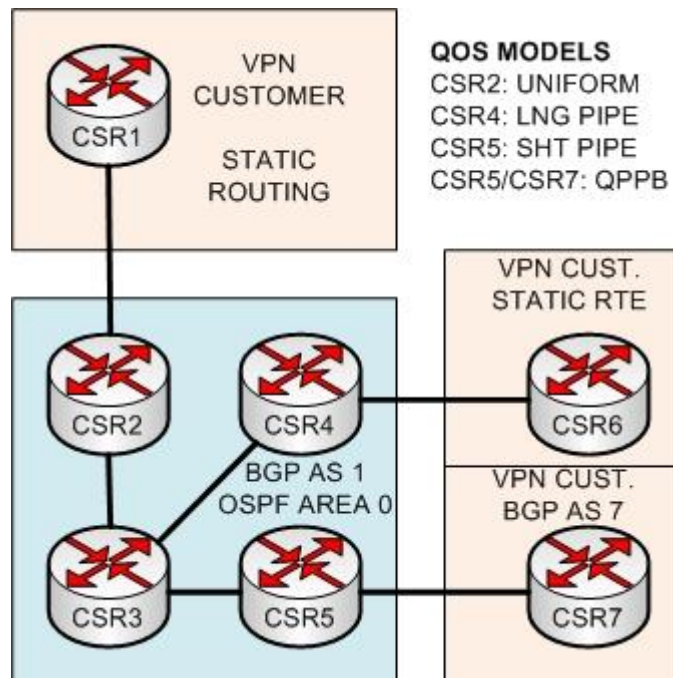
```

Clearing several software processes did not resolve the issue. A reload of CSR7 still did not resolve it. I believe the configuration is correct and all control-plane verifications are accurate.

Additional Reading – Reference configurations “mvpn-extranet-mldp”

30. Describe, implement, and troubleshoot MPLS QoS models and related features

The subsections on MPLS QoS all use the same set of diagrams. The diagram changes slightly for cosmetic reasons when demonstrating IOS XR QoS but the general network remains the same. The diagram is universal for the next three sections.



The configurations below include both IOS and IOS-XE configuration files for all 3 MPLS QoS models, plus QPPB.

Additional Reading – Reference configurations “mpls-qos” and “gns3-mpls-qos”

30.1 Uniform

Uniform is a QoS model whereby changes in the SP network are propagated to the customers. Using an MPLS example, if a customer sends traffic into the network as DSCP AF41, the SP will likely mark this as EXP4 for transport. Let's assume that this traffic was remarked to EXP3 in the core, likely due to policing. This value applies only to the topmost shim header as the rest of the label stack is generally not evaluated/seen by an LSR (the exception would be load sharing for non-IP traffic using the bottom label). Therefore, when the penultimate hop tries to remove the topmost label to expose the VPN label (assuming a basic 2-label VPN design) to the egress PE (CSR2), it will not carry this value. The egress PE will be ignorant of any change in the SP network. This is the primary use-case for enabling explicit-null on a PE; it effectively disables PHP to allow those EXP modifications to be maintained. After receiving an explicit-null labeled packet, the egress PE cannot automatically carry this value into an IP TOS mapping since the label stack is stripped when the routing decision is made. This happens before egress QoS in the order of operations; a new placeholder is needed to match the EXP on ingress then set the IPP/DSCP on egress. This is the “qos-group” construct. A brute-force method would be the following applied to the ingress (core-facing) links:

```
! PE routers
class-map match-all CMAP_EXP0
  match qos-group 0
class-map match-all CMAP_EXP1
```

```

    match qos-group 1
[snip]
class-map match-all CMAP_EXP7
    match qos-group 7

policy-map PMAP_IN_FROM_CORE
    class CMAP_EXP0
        set qos-group 0
    class CMAP_EXP1
        set qos-group 1
[snip]
    class CMAP_EXP7
        set qos-group 7

```

While this works, it is configuration intensive and redundant. A more elegant and dynamic mechanism exists; we can simply tell the router to perform a direct copy from EXP to QG. It would have an identical effect as the 8 class-maps above. The drawback of this approach is described later with regards to discard-class-based WRED. Note: The configurations suffixed “gns3” use this semi-dynamic method while the ones with no special suffix are CSR’s using the brute force method. The end result is same, it’s just a matter of configuration elegance and length.

```

! PE routers
policy-map PMAP_IN_FROM_CORE
    class class-default
        set qos-group mpls experimental topmost

```

Let’s pretend that QG numbers 0-7 were used for some other purpose in my network. I demonstrate a simple workaround on CSR2 using the table-map. This is NOT the same logic as the table-map used for QPPB (which is really a route-map). I can still avoid using 8 class maps and maintain some semblance of dynamic configuration while just selecting new values. On CSR2, I map EXP to QG + 50. The table-map is a generic entity just used to map numbers and is not specific to MPLS QoS. Unfortunately, Cisco clearly states in their IOS XE 3S documentation that the “table-map” is supported, but it actually isn’t (it is missing both from global configuration and from MQC constructs). You must use this “brute force” method on the CSR1000v and likely any other IOS XE 3S platform.

```

! PE routers
policy-map PMAP_IN_FROM_CORE
    class class-default
        set qos-group mpls experimental topmost table TM_EXP_TO_QG

table-map TM_EXP_TO_QG
    map from 0 to 50
    map from 1 to 51
[snip]
    map from 7 to 57

```

Going the other way (out to the customer), it would be nice if we could have similar semi-dynamic behavior. We can also configure IPP, DSCP, or EXP to be derived from the qos-group.

```
policy-map PMAP_OUT_TO_CUSTOMER
  class class-default
    set precedence qos-group table TM_QG_TO_IPP

table-map TM_QG_TO_IPP
  map from 50 to 0
  map from 51 to 1
  [snip]
  map from 57 to 7
  default 0
```

The downside to this approach is that we lose some granularity because our egress policy is essentially just copying EXP->IPP now. We aren't doing any queuing or traffic conditioning. If that is desired, you can break out a few individual EXPs (in my examples I use EXP1 and EXP5, using the default for the rest) for custom treatment. Unless, of course, you just want to apply generic shaping to the default class, in which case explicit break-outs would not be required.

An unfortunate lack-of-feature is that discard-class-based WRED cannot be dynamically mapped this way. In the configurations you will see examples of this feature where I statically set the discard-class to a number. It's effectively like IPP based WRED (lower numbers get worse treatment) except it is like the QoS-group; you can set it inbound and perform WRED outbound. If this was supported dynamically you would be able to provide EXP->IPP mapping and WRED with very little configuration.

A quick example of uniform mode: send traffic from CSR6 with DSCP CS2 and expect to see DSCP AF11 on CSR1.

```
R6#ping 12.0.0.1 tos 64 repeat 5
```

We can quickly check CSR2 to see it matching EXP1 on ingress, setting QG1, matching QG1 on egress, and setting DSCP AF11. CSR3 performs the EXP2->EXP1 remarking for all EXP2 transit traffic; this policy is applied to all core interfaces.

```
! CSR3
class-map match-all CMAP_EXP2
  match mpls experimental topmost 2
```

```
policy-map PMAP_REMARK_EXP
  class CMAP_EXP2
    set mpls experimental topmost 1
```

```
R2#show policy-map interface gig2.523 input | section EXP1
  Class-map: CMAP_EXP1 (match-all)
```



```

5 packets, 630 bytes
5 minute offered rate 0000 bps, drop rate 0000 bps
Match: mpls experimental topmost 1
QoS Set
  discard-class 1
    Marker statistics: Disabled
  qos-group 1
    Marker statistics: Disabled

```

```

R2#show policy-map interface gig2.512 output | section QG1
Class-map: CMAP_QG1 (match-all)
  5 packets, 590 bytes
  5 minute offered rate 0000 bps, drop rate 0000 bps
  Match: qos-group 1
  [snip]
  bandwidth 15% (150000 kbps)
  Exp-weight-constant: 4 (1/16)
  Mean queue depth: 0 packets
  discard-class      Transmitted      Random drop      Tail
drop                Minimum      Maximum      Mark
                    pkts/bytes      pkts/bytes      pkts/bytes
thresh              thresh      prob
16                  0          0/0          0/0          0/0
                    32  1/10
11                  1          5/590        0/0          0/0
                    17  1/7
  [snip]

```

CSR1 should report seeing DSCP AF11 now.

```

R1#show flow monitor FNF_MONITOR_IPV4 cache format table | begin IPV4
IPV4 SRC ADDR      IPV4 DST ADDR      INTF INPUT      IP TOS      pkts
=====
46.0.0.6           12.0.0.1           Gi2.512          0x28          5

```

Math conversion:

DSCP CS2 = 010 000 = 16, left shift x 2 = 64 or 0x40 (0100 0000 for TOS)

DSCP AF11 = 001 010 = 10, left shift x 2 = 40 or 0x28 (0010 1000 for TOS)

30.2 Short pipe

Short pipe is the default mode and requires no special configuration. The word “pipe” indicates that the customer QoS markings are not modified by the SP and are preserved across the VPN. The word “short” indicates that any markings made the provider are NOT used to determine the egress QoS treatment for the customer. Using the same example from above, if the flow is remarked to EXP3 in the core, this value applies to the topmost shim header only and this new marker has no influence on the egress PE’s queuing policy. Thus, PHP can be enabled for this model. The egress queuing on the PE (CSR5) will use

regular IPv4/IPv6 mechanisms. The configuration on CSR5 is very straightforward and I added a simple QoS policy just to illustrate the point. The remarking performed on CSR3 will not affect how CSR5 queues traffic outbound towards the customer. I would imagine this is the most common for service providers as it is the least complex and, since the traffic already arrived at the customer, it doesn't matter as much how you deliver their traffic. It was more important in the SP core to remark the shim headers but at the end of the LSP, it matters less.

A quick example of short-pipe mode: send traffic from CSR6 with DSCP CS2 and expect to see the same value on CSR5.

```
R6#ping 57.0.0.7 tos 64 repeat 5
```

```
R5#show policy-map interface gig2.557 output
[snip]
```

```

      Class-map: class-default (match-any)
        121 packets, 8544 bytes
      [snip]
      default      59/3776      0/0      0/0
52      104  1/10
      cs2          5/590      0/0      0/0
65      104  1/10
      cs6          57/4178     0/0      0/0
91      104  1/10

```

Above, notice that the traffic matches class-default because we did not define any custom actions for IPP2. We can see the counters under the WRED configuration, though. Below, notice that there is no change to the IP TOS byte, as expected, and the PE was NOT able to make queuing/shaping decisions based on SP-modified markings. Only IPP was visible in short-pipe mode.

```

R7#show flow monitor FNF_MONITOR_IPV4 cache format table | begin IPV4
IPV4 SRC ADDR      IPV4 DST ADDR      INTF INPUT          IP TOS              pkts
=====
46.0.0.6           57.0.0.7           Gi2.557              0x40                 5

```

Math conversion:

DSCP CS2 = 010 000 = 16, left shift x 2 = 64 or 0x40 (0100 0000 for TOS)

30.3 Pipe (AKA long pipe)

Long pipe is a combination of the uniform and short pipe modes. It still “pipes” the customer markings through the network, so the customer has no idea that traffic markings were modified along the path. Like uniform mode, the topmost MPLS marking must arrive at the egress PE (CSR4) because the egress QoS treatment will be based on the SP-modified markings. However, these egress policies will NOT remark packets as they are sent to the customer; this is the fundamental difference between uniform

and long pipe. Explicit-null is still required, as are qos-group placeholders. The QoS policies are nearly identical on CSR2 and CSR5, only that CSR5 is not setting new TOS byte values. One other note about this mechanism is that we do not HAVE to match every single value as we did in uniform mode. On CSR4's configuration, notice that I do not have class-maps for EXP2-4 or QG2-4, because I don't need to customize the behavior for those markings. Uniform mode MUST match every single value in order to perform a rewrite on egress.

A quick example of long-pipe mode: send traffic from CSR1 with DSCP CS2 and expect to see the same value on CSR6.

```
R1#ping 46.0.0.6 tos 64 repeat 5
```

Check the PE for matches. CSR4 sees EXP1 coming in as expected, and performs proper outbound actions based on QG1.

```
R4#show policy-map interface gig2.534 input | section EXP1
Class-map: CMAP_EXP1 (match-all)
  5 packets, 630 bytes
  5 minute offered rate 0000 bps, drop rate 0000 bps
  Match: mpls experimental topmost 1
  QoS Set
    discard-class 1
      Marker statistics: Disabled
    qos-group 1
      Marker statistics: Disabled
```

```
R4#show policy-map int gig2.546 output | section QG1
Class-map: CMAP_QG1 (match-all)
  5 packets, 590 bytes
[snip]
```

	discard-class	Transmitted	Random drop	Tail
drop	Minimum	Maximum	Mark	
thresh	pkts/bytes	pkts/bytes	pkts/bytes	
	thresh	prob		
	0	0/0	0/0	0/0
16	32 1/10			
	1	5/590	0/0	0/0
11	17 1/7			

Above, notice that the traffic matches the specific class for QG1. We can see the counters under the WRED configuration also. Below, notice that there is no change to the IP TOS byte, as expected, yet the PE was able to make queuing/shaping decisions based on SP-modified markings.

```
R6#show flow monitor FNF_MONITOR_IPV4 cache format table | begin IPV4
IPV4 SRC ADDR    IPV4 DST ADDR    INTF INPUT          IP TOS          pkts
```

=====	=====	=====	=====	=====
12.0.0.1	46.0.0.6	Gi2.546	0x40	5

Math conversion:

DSCP CS2 = 010 000 = 16, left shift x2 = 64 or 0x40 (0100 0000 for TOS)

30.4 QoS Policy Propagation through BGP (QPPB)

QoS Policy Propagation through BGP (QGGP) is an interesting feature where the BGP NLRI carries the QoS markings for specific routes in a roundabout way. This can simplify QoS configuration on many devices by encoding it within the prefix via communities. It isn't terribly granular or widely deployed but for simple policies it might be a better option than putting MQC configurations on thousands of network devices. It can be awkward because, in many cases, BGP is just transporting prefixes and is not involved in data plane operations. That is why this feature really only makes sense for IPv4/IPv6 unicast AFI/SAFI.

Notice that only CSR5 and CSR7 are running BGP as the PE-CE protocol; the other CEs just have a static default route for reachability and no other connected routes. CSR7 signals CSR5 to set IPP for any traffic that it sources from specific subnets. This essentially says "When I send traffic FROM this subnet, set this QoS marking on ingress". This is what the table-map does; this not related to the actual "table-map" command in global configuration that we used earlier, this is a route-map applied to BGP using the "table-map" command under an address-family. This could be handy to allow your customers to change their QoS markings per-prefix without the SP becoming involved, which would require some kind of remote signaling not available in MQC. There are a few reasons why wouldn't the customer just do the marking locally in the first place:

1. Customer router is very low end, either in feature set or performance, but does support basic BGP and communities. Classification/marketing can be a CPU intensive process.
2. Customer router is actually a firewall, again with limited feature set.
3. Customer is not network savvy and will accept imprecise QoS in return for minimal complexity
4. Inter-domain route exchange where the source provider can advertise what the QoS treatment should be. This can simplify the ASBR QoS configurations, specifically for Inter-AS MPLS Option A. Other non-MPLS related inter-AS exchanges can benefit as well.

Specifically, this mechanism of QPPB is enabled at the link level with the "bgp-policy source ip-prec-map". Substituting the word "destination" for "source" is also valid but the router uses the destination address for the QPPB lookup. Because QPPB is always processed inbound, the logic would become "When I send traffic TOWARDS this subnet, set this QoS marking on ingress". You can enable both simultaneously; in my example I have all QPPB features enabled at once on the customer facing link.

Once enabling this feature on an interface, a quick verification is recommended. Beware the verbiage; QPPB is an ingress-only feature, so the words "input" and "output" in the show command really mean "source" and "destination" in the configuration, respectively. The show command also confirms the assertion that QPPB is an input feature.

```

! CSR5
interface GigabitEthernet2.557
  bgp-policy source ip-prec-map
  bgp-policy destination ip-prec-map

R5#show ip interface gig2.557 | include BGP
  BGP Policy Mapping is enabled (output ip-prec-map) (input ip-prec-map)
  Input features: BGP Policy Map, QoS Classification, Access List, MCI Check

```

Only a single “table-map” can be applied under a BGP AF at one time. Changing the table-map will overwrite the existing value. Thus, your QPPB route-map should have widely spaced sequence numbers for easy insertions.

CSR7 has five networks advertised into BGP. Two of them want traffic to be marked EXP3 and EXP4 by the PE, while the remaining three have no preference according to the customer. CSR7 sets the communities when it originates the networks into BGP and advertises these to the PE (CSR5). CSR5 receives these communities, checks its table-map for any corresponding matches, and takes action. CSR5 is also configured to provide IPP5 treatment for 7.7.7.5/32 (using a prefix-list match) and IPP6 treatment for 7.7.7.6/32 (matching the BGP AS-path, which CSR7 prepended). This is meant to demonstrate that communities do not have to be signaled from another router and that any arbitrary route-map match statements are valid with a few exceptions; as an example, matching MED does not appear to work, and it is effectively a match-any. In most cases, communities make sense, since the customer can signal QoS through BGP in this way. The prefix 7.7.7.7/32 does not match any table-map clause and thus gets no modification. This is NOT the same as setting IPP0 as a default route-map entry and that would overwrite existing QoS markings.

```

! CSR5
ip bgp-community new-format
ip community-list standard COMMLIST_IPP3 permit 77:3
ip community-list standard COMMLIST_IPP4 permit 77:4
ip as-path access-list 7 permit ^7_.+
ip prefix-list PL_7775 seq 5 permit 7.7.7.5/32

route-map RM_QPPB_FOR_CUSTOMER permit 10
  match community COMMLIST_IPP3
  set ip precedence flash
route-map RM_QPPB_FOR_CUSTOMER permit 20
  match community COMMLIST_IPP4
  set ip precedence flash-override
route-map RM_QPPB_FOR_CUSTOMER permit 30
  match ip address prefix-list PL_7775
  set ip precedence critical
route-map RM_QPPB_FOR_CUSTOMER permit 40
  match as-path 7
  set ip precedence internet

```

```
router bgp 1
 address-family ipv4 vrf A
  table-map RM_QPPB_FOR_CUSTOMER
```

The verifications are shown below. The routing table clearly shows the mapping of each destination prefix on CSR7 to the appropriate QoS value identified in the QPPB table-map.

```
! Signaled by customer community
R5#show ip route vrf A 7.7.7.3 | include Routing_entry|Tag
Routing entry for 7.7.7.3/32
  Tag 7, precedence flash (3), type external

! Signaled by customer community
R5#show ip route vrf A 7.7.7.4 | include Routing_entry|Tag
Routing entry for 7.7.7.4/32
  Tag 7, precedence flash-override (4), type external

! Locally specified
R5#show ip route vrf A 7.7.7.5 | include Routing_entry|Tag
Routing entry for 7.7.7.5/32
  Tag 7, precedence critical (5), type external

! Locally specified
R5#show ip route vrf A 7.7.7.6 | include Routing_entry|Tag
Routing entry for 7.7.7.6/32
  Tag 7, precedence internet (6), type external

! Not specified at all; no treatment yet
R5#show ip route vrf A 7.7.7.7 | include Routing_entry|Tag
Routing entry for 7.7.7.7/32
  Tag 7, type external
```

To test it, we will send unmarked traffic from CSR7 to CSR1 source from all 5 of its addresses using a quick TCL script. Using FNF on CSR1 will allow us to capture the markings.

```
foreach x {
  7.7.7.3
  7.7.7.4
  7.7.7.5
  7.7.7.6
  7.7.7.7
} { ping 12.0.0.1 source $x }
```

Another nice feature of QPPB is that in the order of operations, it occurs before routing (and thus before MPLS imposition), so when the IPP is set by CSR5 on ingress it is automatically copied to EXP just as if we used ingress MQC to classify/mark the traffic. With FNF enabled on CSR3 we can see some of these EXP

markings being carried through. Of course, any SP-remark that occurs will overwrite what was written at imposition, so beware of the order of operations when mixing QPPB with uniform/long-pipe QoS models, or mixing it with MQC ingress classification/markings. Let's compare CSR1's captured IP packets with CSR3's captured MPLS packets.

```
R1#show flow monitor FNF_MONITOR_IPV4 cache format table | begin IPV4
```

IPV4 SRC ADDR	IPV4 DST ADDR	INTF INPUT	IP TOS	pkts
7.7.7.5	12.0.0.1	Gi2.512	0xB8	5
7.7.7.4	12.0.0.1	Gi2.512	0x80	5
7.7.7.7	12.0.0.1	Gi2.512	0x00	5
7.7.7.3	12.0.0.1	Gi2.512	0x60	5
7.7.7.6	12.0.0.1	Gi2.512	0xC0	5

I cannot explain it, but despite telling QPPB to set IPP5 for 7.7.7.5/32, it set DSCP EF (DSCP 46 or TOS 0xB8). This is true on both classic IOS and IOS XE. Perhaps the router assumes that this is a more appropriate value; from an MPLS perspective it makes no difference as EXP only captures the 3 leftmost TOS bits. Let's check CSR3 to see the EXP values in the shim headers. I include "20.." because I want to capture labels going to CSR2 only. The output is not totally authoritative but the correlation between CSR3's MPLS FNF cache and CSR1's IPv4 FNF cache is clear. Without any additional verification, we can safely assume that QPPB is marking traffic correctly.

```
R3#show flow monitor FNF_MPLS_MONITOR cache format table | include 20.. | INTF | =====
```

INTF INPUT	INTF OUTPUT	MPLS LABEL 1	MPLS LABEL 2	pkts
Gi2.535	Gi2.523	3000 / 6	2004 / 6	5
Gi2.535	Gi2.523	3000 / 0	2004 / 0	5
Gi2.535	Gi2.523	3000 / 5	2004 / 5	5
Gi2.535	Gi2.523	3000 / 4	2004 / 4	5
Gi2.535	Gi2.523	3000 / 3	2004 / 3	5

QPPB can also be used to set QoS-group flags for source or destination networks. It works the same way as IPP. The router can set QoS-group on ingress then perform some egress shaping or other mechanism without interrupting your existing MQC ingress classification/markings. This is a less intrusive and more flexible usage of QPPB. It can be enabled using the "ip-qos-map" command instead of "ip-prec-map". The two can be enabled concurrently, and in this example, source-based IPP and QG mapping are enabled together.

```
! CSR5
interface GigabitEthernet2.557
  bgp-policy source ip-prec-map
  bgp-policy source ip-qos-map
  bgp-policy destination ip-prec-map
```

```
bgp-policy destination ip-qos-map
```

```
R5#show ip interface gig2.557 | include BGP
```

```
BGP Policy Mapping is enabled (output ip-qos-map) (input ip-qos-map)
```

```
(output ip-prec-map) (input ip-prec-map)
```

```
Input features: BGP Policy Map, QoS Classification, Access List, MCI Check
```

In this case, we set qos-group 7 for all unmatched traffic in the QPPB route-map just to illustrate its use. CSR5 can now use this on different egress interface encapsulations, MPLS or otherwise. This route-map clause just extends the existing QPPB policy.

```
! CSR5
```

```
route-map RM_QPPB_FOR_CUSTOMER permit 1000
```

```
set ip qos-group 7
```

```
! Default match clause in QPPB route-map
```

```
R5#show ip route vrf A 7.7.7.7 | include Routing_entry|Tag
```

```
Routing entry for 7.7.7.7/32
```

```
Tag 7, qos-group 7, type external
```

Another quick example demonstrates the IPP and QG mapping using the “destination” method. In order for this to work, CSR2 advertises a few test loopbacks into BGP within the customer VRF so that CSR7 has something to which traffic can be destined; I didn’t want to run PE-CE BGP with CSR1. A QPPB route-map entry uses a locally-defined prefix-list to match those subnets (99.99.99.2/32 and 99.99.99.22/32) and sets both an IPP and QG mapping. Another option would be to set communities at CSR2 when the routes were redistributed, match them in the QPPB route-map, and take action. Notice that the new route-map clause comes before the default entry (index 1000 above).

```
! CSR5
```

```
ip prefix-list PL_LOOP99_TESTS seq 5 permit 99.99.99.0/24 le 32
```

```
route-map RM_QPPB_FOR_CUSTOMER permit 200
```

```
match ip address prefix-list PL_LOOP99_TESTS
```

```
set ip precedence network
```

```
set ip qos-group 99
```

Here are the routing entries within the VRF. Since this is a destination learned across the VPN, we must use destination-based QPPB (enabled in the beginning of this section).

```
! Locally specified
```

```
R5#show ip route vrf A 99.99.99.2 | include Routing_entry|bgp
```

```
Routing entry for 99.99.99.2/32
```

```
Known via "bgp 1", distance 200, metric 0, precedence network (7), qos-group 99, type internal
```

```
! Locally specified
```



```
R5#show ip route vrf A 99.99.99.22 | include Routing_entry|bgp
Routing entry for 99.99.99.22/32
  Known via "bgp 1", distance 200, metric 0, precedence network (7), qos-
  group 99, type internal
```

A quick FNF check on CSR3 proves that the IPP7 mapped to EXP7 as imposition. I sent 15 packets per route with TOS 0. Just to prove there is no magic, I also show the label stack per prefix from CSR5 (ingress LSR).

```
R3#show flow monitor FNF_MPLS_MONITOR cache format table | begin INTF
INTF INPUT          INTF OUTPUT      MPLS LABEL 1    MPLS LABEL 2    pkts
=====
Gi2.535             Gi2.523          3000 /7        2000 /7        15
Gi2.535             Gi2.523          3000 /7        2001 /7        15
```

```
R5#show ip cef vrf A 99.99.99.2
99.99.99.2/32
  nexthop 35.0.0.3 GigabitEthernet2.535 label 3000 2000
```

```
R5#show ip cef vrf A 99.99.99.22
99.99.99.22/32
  nexthop 35.0.0.3 GigabitEthernet2.535 label 3000 2001
```

Some Cisco documentation suggests that QPPB is processed before input ACLs and MQC in the order of operations. We will test it quickly in IOS (12.4T) by matching IPP3 traffic entering from the customer, then sourcing traffic from 7.7.7.3/32. So long as QPPB marks the traffic first, we expect to see matches in the MQC and ACL counters. In IOS, we see this is true as the ACL matches the traffic as IPP3 along with MQC.

```
! CSR5
class-map match-all CMAP_IPP3_ORDER_OF_OPS_TEST
  match precedence 3
```

```
policy-map PMAP_ORDER_OF_OPS_TEST
  class CMAP_IPP3_ORDER_OF_OPS_TEST
```

```
interface GigabitEthernet2.557
  ip access-group ACL_IPP3_ORDER_OF_OPS_TEST in
  service-policy input PMAP_ORDER_OF_OPS_TEST
```

```
R7#ping 12.0.0.1 source 7.7.7.3 repeat 15
```

```
Type escape sequence to abort.
Sending 15, 100-byte ICMP Echos to 12.0.0.1, timeout is 2 seconds:
Packet sent with a source address of 7.7.7.3
!!!!!!!!!!!!!!
Success rate is 100 percent (15/15), round-trip min/avg/max = 72/94/108 ms
```

```
R5#show policy-map interface gig2.557 input | include ORDER_OF_OPS|packets
Service-policy input: PMAP_ORDER_OF_OPS_TEST
Class-map: CMAP_IPP3_ORDER_OF_OPS_TEST (match-all)
  15 packets, 1710 bytes
```

```
R5#show access-lists ACL_IPP3_ORDER_OF_OPS_TEST
Extended IP access list ACL_IPP3_ORDER_OF_OPS_TEST
  10 permit ip any any precedence flash (15 matches)
  20 permit ip any any
```

IOS XE's operational flow is different than IOS with respect to QPPB. Specifically, QPPB is now processed after the input ACL but still before MQC. The above output was regular IOS 12.4T, the below output is IOS XE 3.13.2S.

```
R7#ping 12.0.0.1 source 7.7.7.3 repeat 15
```

```
Type escape sequence to abort.
Sending 15, 100-byte ICMP Echos to 12.0.0.1, timeout is 2 seconds:
Packet sent with a source address of 7.7.7.3
!!!!!!!!!!!!!!
Success rate is 100 percent (15/15), round-trip min/avg/max = 72/94/108 ms
```

```
R5#show policy-map interface gig2.557 input | include ORDER_OF_OPS|packets
Service-policy input: PMAP_ORDER_OF_OPS_TEST
Class-map: CMAP_IPP3_ORDER_OF_OPS_TEST (match-all)
  15 packets, 1770 bytes
```

```
R5#show access-lists ACL_IPP3_ORDER_OF_OPS_TEST
Extended IP access list ACL_IPP3_ORDER_OF_OPS_TEST
  10 permit ip any any precedence flash
  20 permit ip any any (15 matches)
```

30.5 QoS specifics on IOS XRv

XRv does not appear to support QoS data-plane operations at all. Although I provide the configurations to mirror what was done above, I cannot test it. I cannot even apply an inbound service-policy without the parser complaining.

```
! Any XRv router
interface GigabitEthernet0/0/0/0.523
 service-policy input PMAP_IN_FROM_CORE
!!% 'NetIO' detected the 'warning' condition 'Netio or a feature DLL
returned:': Invalid argument
```

Outbound policies can be applied but are not actually in effect.

```
RP/0/0/CPU0:XRv1#show run interface gig0/0/0/0.512
```

```

interface GigabitEthernet0/0/0/0.512
[snip]
service-policy output PMAP_SHAPE_TO_CUSTOMER

RP/0/0/CPU0:XRv1#show policy-map interface gig0/0/0/0.512
GigabitEthernet0/0/0/0.512 direction input: Service Policy not installed
GigabitEthernet0/0/0/0.512 direction output: Service Policy not installed

```

Even if it were supported, WRED drop probability doesn't look functional either, as a general comment. Considering XR's WRED configuration is highly granular compared to IOS, I assume this is specific to XRv because the feature is hardware dependent.

```

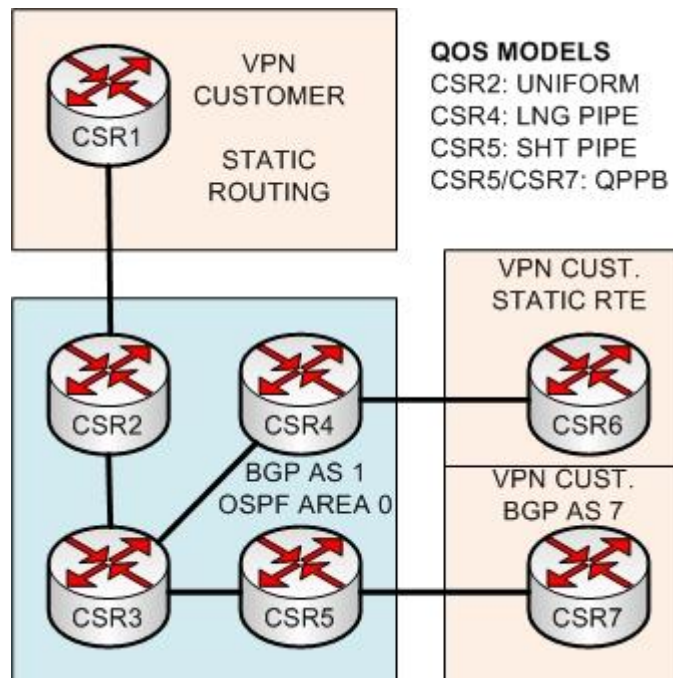
! Any XRv router
class CMAP_QG1
  set dscp af11
  bandwidth percent 15
  random-detect discard-class 1 11 packets 17 packets probability 7

class class-default
  set precedence 0

!!% Subsystem(8147), Code(5): cerrno 0xafe98a00: Policy-map
PMAP_QUEUE_TO_CUSTOMER class CMAP_QG1 failed: random-detect 'drop
probability' is not supported

```

Nonetheless, we will try to do some testing. The provided configurations are mirrored to the CSRs previously used because I feel they are mostly correct as is. I have replaced the core CSRs with XRv routers. Below is the replacement diagram.



When we try to enable IPP-based QPPB on the interface, XRv rejects it:

```
! Any XRv router
interface GigabitEthernet0/0/0/0.557
  ipv4 bgp policy propagation input ip-precedence source
!!% 'ipv4-ma' detected the 'warning' condition 'Platform doesn't support ip-
prec based QPPB in input direction on this card'
```

XR QPPB appears to only support one direction per type. For example, source-based IPP and destination-based QG is acceptable, but you cannot do both source and destination for IPP or QG concurrently. The QG command is accepted by XRv but, in our case, QG is useless because we cannot apply an MQC policy-map anywhere else. Even if QPPB was actually marking traffic, no follow-on actions could occur (unless there were IOS or IOS XE routers elsewhere in the SP network). I left the command in the configuration for reference.

```
! XRv3
interface GigabitEthernet0/0/0/0.557
[snip]
  ipv4 bgp policy propagation input qos-group source
```

Here is the RPL configuration from XRv3 since it is the most interesting part of this test. It's just like the route-map except more powerful. Like programming functions/methods, you can pass parameters. When you call the RPL, you pass it the actual value. Rather than hardcode the default QG to 7, I pass it in from BGP where the RPL is referenced. I always try to use variety with RPL to master the syntax as it is quite comprehensive.

```

RP/0/0/CPU0:XRv3#show rpl
[snip]
prefix-set PS_LOOP99_TESTS
  99.99.99.2/32,
  99.99.99.22/32
end-set
!
route-policy RPL_QPPB_FOR_CUSTOMER($DEFAULT_QG)
  if community matches-any (77:3) then
    set ip-precedence 3
  elseif community matches-any (77:4) then
    set ip-precedence 4
  elseif destination in (7.7.7.5/32) then
    set ip-precedence 5
  elseif as-path length ge 2 then
    set ip-precedence 6
  elseif destination in PS_LOOP99_TESTS then
    set qos-group 9
    set ip-precedence 7
  else
    set qos-group $DEFAULT_QG
  endif
end-policy

```

Applying the QPPB policy to BGP is almost identical to IOS and IOS XE.

```

RP/0/0/CPU0:XRv3#show run router bgp 1 vrf A address-family ipv4 unicast
router bgp 1
  vrf A
    address-family ipv4 unicast
      table-policy RPL_QPPB_FOR_CUSTOMER(7)
      redistribute connected

```

Despite all the limitations with XRv and QoS, with QPPB we can at least do some control-plane verifications. This can show us if our RPL was configured correctly. In XRv, the QPPB information is not visible from the RIB, but can be seen from the FIB. The quotes around the “A” are because XRv assumes that “A” by itself means “all”, due to ambiguity. Even though the FIB shows IPP values, we cannot enable it at the interface level, as seen above.

```

! Signaled by community
RP/0/0/CPU0:XRv3#show cef vrf "A" ipv4 7.7.7.3/32 | include Prec
  IP Precedence: 3

! Signaled by community
RP/0/0/CPU0:XRv3#show cef vrf "A" ipv4 7.7.7.4/32 | include Prec
  IP Precedence: 4

! Locally specified by inline prefix

```

```

RP/0/0/CPU0:XRv3#show cef vrf "A" ipv4 7.7.7.5/32 | include Prec
IP Precedence: 5

! Locally specified by inline AS path
RP/0/0/CPU0:XRv3#show cef vrf "A" ipv4 7.7.7.6/32 | include Prec
IP Precedence: 6

! Locally specified in default class using parameterization
RP/0/0/CPU0:XRv3#show cef vrf "A" ipv4 7.7.7.7/32 | utility egrep 'Prec|QoS'
QoS Group: 7

! Locally specified by prefix-set
RP/0/0/CPU0:XRv3#show cef vrf "A" ipv4 99.99.99.2/32 | include QoS
QoS Group: 9, IP Precedence: 7

! Locally specified by prefix-set
RP/0/0/CPU0:XRv3#show cef vrf "A" ipv4 99.99.99.22/32 | include QoS
QoS Group: 9, IP Precedence: 7

```

We know that our uniform and long-pipe QoS models cannot work on XRv since the input MQC policies from the MPLS core cannot be applied. We also cannot configure XRv4 (formerly CSR3) to perform EXP remarking in the core for the same reason. Short-pipe mode does work correctly because when the LSRs perform label swap operations, they don't touch the EXP bits. XRv basically has no mechanism to do anything else with the EXP bits once it copied the IPP into EXP at imposition. A quick test verifies this.

```

R1#ping 57.0.0.7 tos 64 repeat 5

R1#show flow monitor FNF_MONITOR_IPV4 cache format table | begin IPV4
IPV4 SRC ADDR      IPV4 DST ADDR      INTF INPUT          IP TOS              pkts
=====
12.0.0.1           57.0.0.7           Gi2.557              0x40                 5

```

We can reuse our TCL script for CSR7 to test the QPPB marking. Earlier we saw a variety of markings consistent with our QPPB policy. Now, we see TOS 0 for everything. XRv is not marking the traffic despite the FIB being correctly programmed, again because we did not enable the BGP policy feature at the interface level.

```

R1#show flow monitor FNF_MONITOR_IPV4 cache format table | begin IPV4
IPV4 SRC ADDR      IPV4 DST ADDR      INTF INPUT          IP TOS              pkts
=====
7.7.7.5            12.0.0.1           Gi2.512              0x00                 5
7.7.7.6            12.0.0.1           Gi2.512              0x00                 5
7.7.7.7            12.0.0.1           Gi2.512              0x00                 5
7.7.7.3            12.0.0.1           Gi2.512              0x00                 5
7.7.7.4            12.0.0.1           Gi2.512              0x00                 5

```

Netflow is totally unsupported in XRv, so we cannot use it for QoS verification. The command syntax

isn't even available (probably removed due to lack of supported hardware). The XR 5.2.0 documentation clearly states the parser tree begins with the word "flow". Fortunately, the CSRs as CE devices offer some visibility between FNF and EPC.

```
RP/0/0/CPU0:XRv1(config)#flow?
      ^
% Invalid input detected at '^' marker.
```

IOS XR supports QPPB for IPv6 too. I cannot find any documentation that claims IPv6 support for IOS or IOS XE. CSR7 advertises two IPv6 prefixes to XRv13; we will match one (::7:7:7:7/128) by setting the community to give it IPP3 treatment, and we let the other match the default QG setting (::7:7:7:77/128). We don't need to modify RPL because I call the policy from within the IPv6 unicast AFI and can pass in a new value of 6 (not 7) with parameterization. Communities and BGP attributes are not AFI specific so matching non-prefix related things in a QPPB RPL or route-map is preferable.

```
RP/0/0/CPU0:XRv3#show run router bgp 1 vrf A address-family ipv6 unicast
router bgp 1
  vrf A
    address-family ipv6 unicast
      table-policy RPL_QPPB_FOR_CUSTOMER(6)
      redistribute connected
```

```
! Signaled by community
RP/0/0/CPU0:XRv3#show cef vrf "A" ipv6 ::7:7:7:7/128 | utility egrep
'Prec|QoS'
  IP Precedence: 3
```

```
! Not specified; matches default RPL "else" statement
RP/0/0/CPU0:XRv3#show cef vrf "A" ipv6 ::7:7:7:77/128 | utility egrep
'Prec|QoS'
  QoS Group: 6
```

When we try to enable IPP-based QPPB on the interface, XRv also rejects it for IPv6. I just enabled sourced-based QoS group for reference as I did for IPv4.

```
interface GigabitEthernet0/0/0/0.557
  ipv6 bgp policy propagation input ip-precedence source
!!% 'ipv6-ma' detected the 'warning' condition 'Platform doesn't support ip-
prec based QPPB in input direction on this card'
```

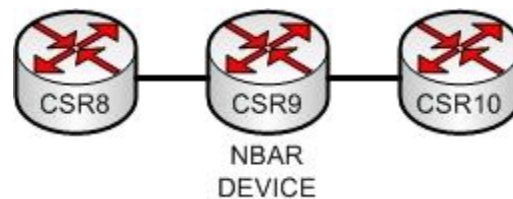
Additional Reading – Reference configurations "mpls-qos-xr"

30.6 Network Based Application Recognition (NBAR) summary and configurations

Network based application recognition allows MQC to identify applications/protocols directly without having to use other match criteria, such as ports/protocols inside an ACL. It is invoked using the "match protocol" keyword in IOS and IOS XE.

NBAR is not supported on IOS XR. The “match protocol” parser tree only lets you match IP protocols, like TCP and UDP. You can specify a raw IP protocol number also, but this is nowhere near the power of NBAR2 on IOS XE platforms. Given this restriction, NBAR is not evaluated on XR in this test.

The network is simple with CSR8, CSR9, and CSR10 arrayed in a line. NBAR is not specifically an SP technology and is unlikely to be tested, but since XE makes use of many advanced NBAR features, it is discussed. Most of the NBAR configurations will be on CSR9, applied inbound to the link facing CSR8. The diagram is shown below.



A simple and common NBAR deployment is shown by the PMAP_SIMPLE policy. We can identify some protocols based on the built-in NBAR2 protocol pack listing inside a class-map, then perform some action. In this case, I only care about matching packets, not performing QoS treatment, so the policy-map invokes the class and does nothing.

```
! CSR9
class-map match-any CMAP_TTY_MGMT
  match protocol telnet
  match protocol ssh

policy-map PMAP_SIMPLE
  class CMAP_TTY_MGMT
```

Below are a few quick tests where CSR9 is supported to match telnet or SSH traffic arriving from CSR8. CSR8 telnets and SSH'es to CSR10 to test it. The final test is to show that ping traffic does not match.

```
! telnet test
R8#telnet 90.0.0.10
Trying 90.0.0.10 ... Open

R9#show policy-map interface gig2.589 input class CMAP_TTY_MGMT
GigabitEthernet2.589

Service-policy input: PMAP_SIMPLE

Class-map: CMAP_TTY_MGMT (match-any)
  9 packets, 552 bytes
  5 minute offered rate 0000 bps
  Match: protocol telnet
```



```

Match: protocol ssh

R9#clear counter
Clear "show interface" counters on all interfaces [confirm]

! ssh test
R8#ssh -l fakeuser 90.0.0.10
Password:

R9#show policy-map interface gig2.589 input class CMAP_TTY_MGMT
GigabitEthernet2.589

Service-policy input: PMAP_SIMPLE

Class-map: CMAP_TTY_MGMT (match-any)
  82 packets, 4818 bytes
  5 minute offered rate 1000 bps
  Match: protocol telnet
  Match: protocol ssh

R8#ping 90.0.0.10 repeat 15

R9#show policy-map interface gig2.589 input
GigabitEthernet2.589

Service-policy input: PMAP_SIMPLE

Class-map: CMAP_TTY_MGMT (match-any)
  0 packets, 0 bytes
  5 minute offered rate 0000 bps
  Match: protocol telnet
  Match: protocol ssh

Class-map: class-default (match-any)
  15 packets, 1770 bytes
  5 minute offered rate 1000 bps, drop rate 0000 bps
  Match: any

```

Additional Reading – Reference configurations "nbar"

30.6.1 NBAR Custom Protocols

NBAR also allows the user to define custom protocols in a number of ways. I will not be discussing the deprecated "ip nbar port-map method" as it is no longer shown in the context-sensitive help:

```

R9(config)#ip nbar port-map radius udp 1645 1646
%NBAR-6-PORT_MAP_DEPRECATED: Port-map command will be deprecated soon.
    In future it will not be necessary to configure port-map on a Protocol
to create a new

```

Custom protocol onto the same well known port.

1. Simple TCP/UDP port mapping to a custom name. Before NBAR2, some common protocols like Windows RDP and RADIUS were not pre-defined. Using this technique for commonly used protocols is handy, as well as custom applications. PMAP_CUSTOM_RADIUS is an example of using this feature to match the old RADIUS authentication (1645) and accounting (1646) ports. NBAR2 only considers the newer RADIUS ports for classification by default.

```
R9#show ip nbar port-map | include radius
port-map radius                        udp 1813 1812

R9(config)#ip nbar custom UDP_RADIUS_OLD udp 1645 1646

R9#show ip nbar port-map | include RADIUS|radius
port-map UDP_RADIUS_OLD                udp 1645 1646
port-map radius                        udp 1813 1812
```

These are wrapped in the PMAP_CUSTOM_RADIUS policy-map and applied to the interface facing CSR8.

```
! CSR9
class-map match-any CMAP_RADIUS
  match protocol radius
  match protocol UDP_RADIUS_OLD

policy-map PMAP_CUSTOM_RADIUS
  class CMAP_RADIUS
```

On CSR8, we setup a bogus RADIUS server with IP addresses 90.0.0.100. AAA is described in the security chapter, but the commands are in the configuration file. Interestingly, even on this relatively new XE version, the default ports are the old ports.

```
R8(config-radius-server)#address ipv4 90.0.0.100 ?
  acct-port  UDP port for RADIUS accounting server (default is 1646)
  alias      1-8 aliases for this server (max. 8)
  auth-port  UDP port for RADIUS authentication server (default is 1645)
```

Once the server is configured, we will use CSR8 to telnet to itself. CSR8 should then try to reach the RADIUS server for authentication, which appears to be on the LAN behind CSR9. We should see matches under the class-map regardless of which set of ports we use.

```
R8#telnet 89.0.0.8
Trying 89.0.0.8 ... Open

User Access Verification

Username: fakeuser
```

Password:

```
R9#show policy-map interface gig2.589 input class CMAP_RADIUS
GigabitEthernet2.589
```

```
Service-policy input: PMAP_CUSTOM_RADIUS
```

```
Class-map: CMAP_RADIUS (match-any)
  4 packets, 472 bytes
  5 minute offered rate 0000 bps
  Match: protocol radius
  Match: protocol UDP_RADIUS_OLD
```

It works with the old ports, so we know our custom NBAR class is operational. Let's double-check the new ports by using 1812 and 1813 in the RADIUS server configuration (configuration not shown). It also works. The reason we see exactly 4 packets is because retransmit count was set to 3, so that means 3 retries after the first failure.

```
R9#clear counter
Clear "show interface" counters on all interfaces [confirm]
```

```
R8#telnet 89.0.0.8
Trying 89.0.0.8 ... Open
```

User Access Verification

```
Username: fakeuser
Password:
```

```
R9#show policy-map interface gig2.589 input class CMAP_RADIUS
GigabitEthernet2.589
```

```
Service-policy input: PMAP_CUSTOM_RADIUS
```

```
Class-map: CMAP_RADIUS (match-any)
  4 packets, 472 bytes
  5 minute offered rate 0000 bps
  Match: protocol radius
  Match: protocol UDP_RADIUS_OLD
```

A more complicated custom protocol involves measuring byte offsets in the packet to match specific strings in the payload. It is kind of like flexible packet matching (FPM), but is less powerful/easier to use. PMAP_CUSTOM_PAYLOAD demonstrates looking for the string "BEEF" inside a UDP packet using port 55555. CSR8 uses IP SLA (beyond the scope of this test) to generate these packets every 2 seconds towards CSR10. CSR10 will not respond, but it doesn't matter. The IP SLA has been configured to use the string BEEF, or more specifically, 0x0000BEEF since the data-pattern is measured as a 32-bit entity, and I

have only configured 16 bits. This is the packet dump from CSR8 that demonstrates this. Even more confusing is the “phantom” 4 bytes of 0x0010000 (shown in cyan) which is not part of the UDP header (shown in green) nor the payload (shown in yellow) that I specified.

```
! CSR9
IP: s=89.0.0.8 (local), d=90.0.0.10, len 44, local feature
7F1B9765C230: 4500002C 00000000 FF110000 59000008 E.,.....Y...
7F1B9765C240: 5A00000A CEDBD903 001867FC 00010000 Z...N[Y...g|....
7F1B9765C250: 0000BEEF 0000BEEF 0000BEEF ..>o..>o..>o
```

We must tell NBAR to offset 6 bytes into the payload of UDP to see the first occurrence of BEEF. This would generally be used to identify a virus or malicious packet entering the network. I would have normally expected a 2 byte offset, but as seen above, IP SLA is adding some extra detail into the payload.

```
! CSR9
ip nbar custom FIND_BEEF 6 hex BEEF udp 55555

class-map match-any CMAP_FIND_BEEF
  match protocol FIND_BEEF

policy-map PMAP_CUSTOM_PAYLOAD
  class CMAP_FIND_BEEF
```

We can verify the operation using the commands shown below.

```
R9#show policy-map interface gig2.589 input class CMAP_FIND_BEEF
GigabitEthernet2.589

Service-policy input: PMAP_CUSTOM_PAYLOAD

Class-map: CMAP_FIND_BEEF (match-any)
  7 packets, 434 bytes
  5 minute offered rate 0000 bps
  Match: protocol FIND_BEEF
```

Taking this a step further, let’s pretend we are trying to stop a virus with a generally unrecognizable hex string on a specific set of ports. At the same time, we need to ensure we don’t drop legitimate traffic. We can specify the port-range as we have before, but abstract a specific portion (up to 4 bytes) of the hex string for specification in a class-map. This adds some flexibility without having to define tons of custom NBAR protocols.

Specifically, packets with the hex strings ending in 0x####5100, 0x####570A, and 0x####5B0C running on UDP port 55556 are considered suspect. Anything else is acceptable. Because we are fabricating the packets with IP SLA, we need to account for the first 4 “phantom” bytes. We are matching the last 2

bytes of the pattern, which implies we still need to account for a 6 byte offset; the “virus” does not modify the first 2 bytes of the payload pattern. The configuration is very similar except instead of the “hex” keyword we use “variable” to denote that this sub-protocol variable will be defined in the class-map. We can call the variable whatever we want.

```
! CSR9
ip nbar custom STOP_VIRUS 6 variable LAST_2_BYTES 2 udp 55556

class-map match-any CMAP_VIRUS_570A
  match protocol STOP_VIRUS LAST_2_BYTES "0x570A"
class-map match-any CMAP_VIRUS_5100
  match protocol STOP_VIRUS LAST_2_BYTES "0x5100"
class-map match-any CMAP_VIRUS_5B0C
  match protocol STOP_VIRUS LAST_2_BYTES "0x5B0C"

policy-map PMAP_STOP_VIRUS
  class CMAP_VIRUS_5100
  class CMAP_VIRUS_570A
  class CMAP_VIRUS_5B0C
```

To test this, I have 4 IP SLA instances; three viruses and one benign flow (uses 0x6BA4570C, which is fine). We expect PMAP_STOP_VIRUS to match the first three, but not the last one. Normally it makes more sense to configure all of these matches in a single match-any class-map, but I used three separate class-maps just to show the counters without having to start/stop IP SLAs and clear counters. All three run at the same rate, so the packet counters should be nearly equal.

```
R9#show policy-map interface gig2.589 input
GigabitEthernet2.589

Service-policy input: PMAP_STOP_VIRUS

Class-map: CMAP_VIRUS_5100 (match-any)
  12 packets, 744 bytes
  5 minute offered rate 0000 bps
  Match: protocol STOP_VIRUS LAST_2_BYTES "0x5100"

Class-map: CMAP_VIRUS_570A (match-any)
  13 packets, 806 bytes
  5 minute offered rate 0000 bps
  Match: protocol STOP_VIRUS LAST_2_BYTES "0x570A"

Class-map: CMAP_VIRUS_5B0C (match-any)
  12 packets, 744 bytes
  5 minute offered rate 0000 bps
  Match: protocol STOP_VIRUS LAST_2_BYTES "0x5B0C"

Class-map: class-default (match-any)
```

```
12 packets, 744 bytes
5 minute offered rate 0000 bps, drop rate 0000 bps
Match: any
```

30.6.2 NBAR Attributes

Another useful feature of NBAR2 is the ability to apply specific attributes to a protocol (whether it's a built-in or custom protocol). These attributes can be used for monitoring, but also to simplify QoS policies. For example, I can classify SSH, telnet, and RADIUS together in a category of "net-admin", then match this attribute in a class-map. The trade-off for a simplified QoS policy is a potentially expanded NBAR configuration, unless the default attributes for a protocol are already acceptable. Attribute types are as follows:

application-group: Used to group things together based on similar application/use (jabber, flash, gnutella, AOL, etc)

category: First level of categorization (email, chat, gaming, etc)

encrypted: Determine if the flow is encrypted (yes, no, unassigned)

p2p-technology: Determine if the flow is point to point only (yes, no, unassigned)

sub-category: Second level of categorization (auth services, database, etc)

tunnel: Determine if the flow is inside a tunnel (yes, no, unassigned)

The process contains up to three steps as the first one is optional.

1. Define custom attributes if you like. If the built-in classifications are not appropriate, XE lets you create one custom attribute per type. Only application-group, category, and sub-category can be redefined since the other attributes are essentially yes/no questions. In this example, I created a custom application-group for RADIUS just to demonstrate. I like that you can specify the context-sensitive help string, too.

```
! CSR9
ip nbar attribute application-group custom RADIUS_GROUP "Custom group for
RADIUS services"
```

The below command shows that we only have a limit of one custom attribute per type, and that we already created an application-group attribute.

```
R9#show ip nbar attribute-custom

      Name :  application-group
      Help :  Application-group attribute
Custom Groups Limit :  1
Custom Groups Created :  RADIUS_GROUP

      Name :  category
      Help :  Category attribute
```

```
Custom Groups Limit : 1
```

```
Name : sub-category  
Help : Sub-category attribute
```

```
Custom Groups Limit : 1
```

If we try to create more application-group attributes, the router rejects it.

```
R9(config)#ip nbar attribute application-group custom NEW_ONE  
% NBAR Error: Custom groups limit reached for attribute "application-group"
```

2. Next, we create an attribute-map, which is essentially a list of attributes. We can specify as many or as few attributes as we like. This map will be attached to a protocol later on. Since we are attaching this to our RADIUS protocol (both the built-in and the custom one), we will select appropriate categories. See below for the snippet.

```
! CSR9  
ip nbar attribute-map RADIUS_ATT_MAP  
attribute application-group RADIUS_GROUP  
attribute category net-admin  
attribute encrypted encrypted-no  
attribute p2p-technology p2p-tech-no  
attribute sub-category authentication-services  
attribute tunnel tunnel-no
```

If we use the ? symbol under the attribute map when setting the application-group, we can see our help text.

```
R9(config)#ip nbar attribute-map RADIUS_ATT_MAP  
R9(config-attribute-map)#attribute application-group ?  
RADIUS_GROUP Custom group for RADIUS services
```

3. Apply the attribute-map to a protocol using the “attribute-set” command. In this case, we want to affix these attributes to our RADIUS definitions. However, before we do that, let’s verify the existing attributes of the protocols first.

```
R9#show ip nbar protocol-attribute radius  
Protocol Name : radius  
application-group : other  
category : net-admin  
encrypted : encrypted-no  
p2p-technology : p2p-tech-no  
sub-category : authentication-services  
tunnel : tunnel-no
```

```
R9#show ip nbar protocol-attribute UDP_RADIUS_OLD
```

```

        Protocol Name :   UDP_RADIUS_OLD
    application-group :   other
            category :   other
        encrypted :   encrypted-unassigned
    p2p-technology :   p2p-tech-unassigned
        sub-category :   other
            tunnel :   tunnel-unassigned

```

Now, we will overwrite those settings with our new attribute-map. Any unspecified attributes in the map are left alone in the original attribute listing. Notice how our policy basically mirrors what “radius” already had, plus our new custom group. The custom “UDP_RADIUS_OLD” protocol we defined had very generic attributes since the router has no idea what it is.

```

! CSR9
ip nbar attribute-set radius RADIUS_ATT_MAP
ip nbar attribute-set UDP_RADIUS_OLD RADIUS_ATT_MAP

```

```

R9#show ip nbar protocol-attribute radius
    Protocol Name :   radius
    application-group :   RADIUS_GROUP
            category :   net-admin
        encrypted :   encrypted-no
    p2p-technology :   p2p-tech-no
        sub-category :   authentication-services
            tunnel :   tunnel-no

```

```

R9#show ip nbar protocol-attribute UDP_RADIUS_OLD
    Protocol Name :   UDP_RADIUS_OLD
    application-group :   RADIUS_GROUP
            category :   net-admin
        encrypted :   encrypted-no
    p2p-technology :   p2p-tech-no
        sub-category :   authentication-services
            tunnel :   tunnel-no

```

For a given attribute, we can see all registered protocols. For example, here is the list of all protocols that are in application-group RADIUS_GROUP. I also show a truncated version of all of the category net-admin protocols to demonstrate the use. Notice we see our custom UDP_RADIUS_OLD in the list, too.

```

R9#show ip nbar attribute application-group RADIUS_GROUP
    UDP_RADIUS_OLD      User defined Protocol UDP_RADIUS_OLD
    radius              Remote Authentication Dial In User Service protocol

R9#show ip nbar attribute category net-admin
    914c/g              Texas Instruments 914 Terminal
    9pfs                Plan 9 file service
    UDP_RADIUS_OLD      User defined Protocol UDP_RADIUS_OLD

```



```

acap                ACAP
active-directory    Active Directory Traffic
agentx              AgentX
alpes                Alpes
[snip]

```

At this point, we can use the attributes any way we like. I built a basic MQC configuration to test it using the policy-map PMAP_RADIUS_ATTRIBUTES. Specifically, the class-map looks for both the category net-admin AND application-group RADIUS_GROUP. We will telnet from CSR8 to itself to test the RADIUS configuration. At this point, it does not matter which ports our RADIUS configuration on CSR8 is using since either set of ports will match.

```

! CSR9
class-map match-all CMAP_RADIUS_ATTRIBUTES
  match protocol attribute category net-admin
  match protocol attribute application-group RADIUS_GROUP

policy-map PMAP_RADIUS_ATTRIBUTES
  class CMAP_RADIUS_ATTRIBUTES

```

Below are some quick tests to ensure the policy is functional.

```

R8#telnet 89.0.0.8
Trying 89.0.0.8 ... Open

```

```

User Access Verification

```

```

Username: fakeuser
Password:

```

```

R9#show policy-map interface gig2.589
GigabitEthernet2.589

```

```

Service-policy input: PMAP_RADIUS_ATTRIBUTES

Class-map: CMAP_RADIUS_ATTRIBUTES (match-all)
  4 packets, 472 bytes
  5 minute offered rate 0000 bps
  Match: protocol attribute category net-admin
  Match: protocol attribute application-group RADIUS_GROUP

Class-map: class-default (match-any)
  0 packets, 0 bytes
  5 minute offered rate 0000 bps, drop rate 0000 bps
  Match: any

```

Just like before, we see exactly (4) RADIUS packets enter, except this time we classified them based on

two attributes. Let's telnet from CSR8 to CSR10 and ensure telnet traffic does not match the RADIUS class. Telnet is part of category net-admin but not part of application-group RADIUS_GROUP (we validate this using both lookup mechanisms). This test is just validating basic match-all class-map logic when using NBAR attributes.

```
R9#show ip nbar protocol-attribute telnet | include _category|application
      application-group : other
      category : net-admin

R9#show ip nbar attribute category net-admin | include ^__telnet
telnet                               Telnet - virtual text-oriented terminal over network

R9#show ip nbar attribute application-group RADIUS_GROUP | include telnet
[no output]

R8#telnet 90.0.0.10
Trying 90.0.0.10 ... Open

R9#show policy-map interface gig2.589
GigabitEthernet2.589

Service-policy input: PMAP_RADIUS_ATTRIBUTES

Class-map: CMAP_RADIUS_ATTRIBUTES (match-all)
  0 packets, 0 bytes
  5 minute offered rate 0000 bps
  Match: protocol attribute category net-admin
  Match: protocol attribute application-group RADIUS_GROUP

Class-map: class-default (match-any)
  19 packets, 1144 bytes
  5 minute offered rate 0000 bps, drop rate 0000 bps
  Match: any
```

30.6.3 NBAR Attributes with HTTP

Our next test will include using NBAR to classify HTTP based on the host/URL information. We can do this in two ways: use the built-in MQC syntax inside the class-map with "match protocol http" or generate a custom NBAR protocol of type HTTP. We will test both. I have enabled HTTP server on CSR10 (user/pass: http/http) and added files to flash called "route.txt" and "cef.txt". The files contain the results of CSR10's RIB and FIB respectively. Let's quickly make sure the HTTP download works before continuing; filtering is used to reduce output for verification.

```
R10#show ip route | redirect flash:route.txt
R10#show ip cef | redirect flash:cef.txt

R8#more http://http:http@90.0.0.10/route.txt | begin Gateway
Gateway of last resort is 90.0.0.9 to network 0.0.0.0
```

```

S*    0.0.0.0/0 [1/0] via 90.0.0.9
      90.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C      90.0.0.0/24 is directly connected, GigabitEthernet2.590
L      90.0.0.10/32 is directly connected, GigabitEthernet2.590

```

```

R8#more http://http:http@90.0.0.10/cef.txt | include Gigabit
0.0.0.0/0          90.0.0.9          GigabitEthernet2.590
90.0.0.0/24        attached          GigabitEthernet2.590
90.0.0.0/32        receive           GigabitEthernet2.590
90.0.0.9/32        attached          GigabitEthernet2.590
90.0.0.10/32       receive           GigabitEthernet2.590
90.0.0.255/32      receive           GigabitEthernet2.590

```

A key point with the “host” and “url” keywords (example: www.cisco.com/go/coolstuff/page.htm)

-Host: matches the host name only (www.cisco.com)

-URL: matches everything after the host name ([/go/coolstuff/page.htm](http://www.cisco.com/go/coolstuff/page.htm))

We will use the “url” feature with a simple regular expression on CSR9. This will ultimate feed into a policy-map called PMAP_HTTP. Let’s also try it with the NBAR custom protocol method to match the CEF file. The policy-map should meter both individually.

```

! CSR9
class-map match-all CMAP_HTTP
  match protocol http url "*route\.txt*"

ip nbar custom HTTP_CEF_FILE http url *cef\.txt*
class-map match-all CMAP_HTTP_CUSTOM
  match protocol HTTP_CEF_FILE

policy-map PMAP_HTTP
  class CMAP_HTTP

```

In case you didn’t know that the parser expected a regular expression, you can verify with the following show command. It will show all the sub-classification parameters for a given protocol.

```

R9#show ip nbar parameter subclassification http
Protocol      Parameter      Parameter type
-----
http          mime          regexp_url
http          content-encoding  regexp_url
http          location       regexp_url
http          server         regexp_url
http          from           regexp_url
http          referer        regexp_url
http          user-agent     regexp_url
http          host           regexp_url

```

```
http          url          regexp_url
```

Now, we can copy the files and look for matches.

```
R8#copy http://http:http@90.0.0.10/route.txt null:
Accessing http://*:~@90.0.0.10/route.txt...
Loading http://*:~@90.0.0.10/route.txt
866 bytes copied in 0.045 secs (19244 bytes/sec)
```

```
R8#copy http://http:http@90.0.0.10/cef.txt null:
Accessing http://*:~@90.0.0.10/cef.txt...
Loading http://*:~@90.0.0.10/cef.txt
664 bytes copied in 0.045 secs (14756 bytes/sec)
```

```
R9#show policy-map interface gig2.589
GigabitEthernet2.589
```

```
Service-policy input: PMAP_HTTP
```

```
Class-map: CMAP_HTTP (match-all)
  22 packets, 1858 bytes
  5 minute offered rate 0000 bps
  Match: protocol http url "*route\.txt*"
```

```
Class-map: CMAP_HTTP_CUSTOM (match-all)
  22 packets, 1850 bytes
  5 minute offered rate 0000 bps
  Match: protocol HTTP_CEF_FILE
```

```
Class-map: class-default (match-any)
  16 packets, 960 bytes
  5 minute offered rate 0000 bps, drop rate 0000 bps
  Match: any
```

The reason we see some class-default matches is because the underlying TCP setup/teardown process is not HTTP-specific and would not contain HTTP URL information. If we downloaded a bigger file this would be less significant as we should see about the same number of class-default packets. As always, let's prove it. We should see approximately 8 class-default packets (we saw 16 for two downloads), and many more data packets, when downloading a large file. I can't explain the MQC byte counters, but I estimate it is because NBAR is sampling only certain packets in the flow to conserve CPU/memory resources. Later versions of code allow you to select coarse or fine granularity when sampling; coarse is less accurate but conserves resources.

```
R10#show tech-support | append bootflash:route.txt
```

```
R10#dir bootflash: | include route
 39  -rw-          642828  MON 3 2015 00:14:18 +00:00  route.txt
```

```
R8#copy http://http:http@90.0.0.10/route.txt null:
Accessing http://*:~@90.0.0.10/route.txt...
Loading http://*:~@90.0.0.10/route.txt !!!
642828 bytes copied in 0.448 secs (1434884 bytes/sec)
```

```
R9#show policy-map interface gig2.589
GigabitEthernet2.589
```

```
Service-policy input: PMAP_HTTP
```

```
Class-map: CMAP_HTTP (match-all)
  2792 packets, 162518 bytes
  5 minute offered rate 0000 bps
  Match: protocol http url "*route\.txt*"
```

```
Class-map: class-default (match-any)
  8 packets, 480 bytes
  5 minute offered rate 0000 bps, drop rate 0000 bps
  Match: any
```

The design reason to use the NBAR option (cef.txt) over the MQC inline option (route.txt) for HTTP matching is NBAR attributes. I've added an attribute-map for the CEF inspection. Because the custom protocol was defined outside of a specific class-map, this protocol can be customized. Of course the FIB has nothing to do with "file-sharing", so the attributes below are for testing only.

```
! CSR9
ip nbar attribute-map HTTP_CEF_ATT_MAP
  attribute category file-sharing
  attribute encrypted encrypted-no
  attribute sub-category file-sharing

ip nbar attribute-set HTTP_CEF_FILE HTTP_CEF_ATT_MAP
```

Notice how some fields, like P2P and tunnel, remain unassigned because my attribute-map did not modify them. Other fields, such as category, sub-category, and encrypted were modified. The first show command shows the map as a stand-alone entity, and the second show command is used to validate that the attribute-set operation worked; that is, that the attribute-map was properly applied to the protocol in question.

```
R9#show ip nbar attribute-map HTTP_CEF_ATT_MAP
  Profile Name : HTTP_CEF_ATT_MAP
    category : file-sharing
    encrypted : encrypted-no
    sub-category : file-sharing
```

```
R9#show ip nbar protocol-attribute HTTP_CEF_FILE
```

```

    Protocol Name : HTTP_CEF_FILE
  application-group : other
    category : file-sharing
    encrypted : encrypted-no
  p2p-technology : p2p-tech-unassigned
    sub-category : file-sharing
    tunnel : tunnel-unassigned

```

Let's double check and ensure NBAR registered these attributes with our protocol using the reverse-lookup mechanism.

```

R9#show ip nbar attribute category file-sharing | include CEF
HTTP_CEF_FILE          User defined Protocol HTTP_CEF_FILE

R9#show ip nbar attribute sub-category file-sharing | include CEF
HTTP_CEF_FILE          User defined Protocol HTTP_CEF_FILE

R9#show ip nbar attribute encrypted encrypted-no | include CEF
HTTP_CEF_FILE          User defined Protocol HTTP_CEF_FILE

```

The policy-map PMAP_HTTP_ATT was created to match this traffic now based on attributes.

```

! CSR9
class-map match-all CMAP_HTTP_ATT
  match protocol attribute category file-sharing

policy-map PMAP_HTTP_ATT
  class CMAP_HTTP_ATT

```

We can see the matches below. Of course, this policy would also match anything else displayed by the “show ip nbar attribute category file-sharing” command, as all of these protocols share this category attribute.

```

R8#copy http://http:http@90.0.0.10/cef.txt null:
Accessing http://*:*@90.0.0.10/cef.txt...
Loading http://*:*@90.0.0.10/cef.txt
664 bytes copied in 0.041 secs (16195 bytes/sec)

R9#show policy-map interface gig2.589
GigabitEthernet2.589

Service-policy input: PMAP_HTTP_ATT

Class-map: CMAP_HTTP_ATT (match-all)
  22 packets, 1850 bytes
    5 minute offered rate 0000 bps
    Match: protocol attribute category file-sharing

```

```

Class-map: class-default (match-any)
  8 packets, 480 bytes
  5 minute offered rate 0000 bps, drop rate 0000 bps
Match: any

```

30.6.4 NBAR Protocol-ID

NBAR can be used to determine a specific “protocol-id” for a given parameter, along with the type. For common layer3/layer4 protocols, the IP protocol number or transport port number are used for the protocol-id, respectively. Here are some examples of layer-3 and layer-4 protocol-IDs. You can use this, along with the NBAR port-map, to determine port/protocol numbers. Some people do not know the IP protocol number for RSVP, for example, as it is only really used for MPLS-TE, call admission control, and a few other topics. TFTP is based on UDP and uses port 69, as shown below.

```

R9#show ip nbar protocol-id rsvp
Protocol Name      id      type
-----
rsvp               46      L3 IANA

R9#show ip nbar protocol-id tftp
Protocol Name      id      type
-----
tftp              69      L4 IANA

```

Some things are not IANA assigned but are still semi-common. Notice that the protocol-id is the same value for TFTP and bittorrent, but the type of protocol is different. I am assuming that “L7 STANDARD” is some commonly agreed-upon numbering scheme for applications not tracked or managed by IANA. I have not found any authoritative documentation on what that means for certain.

```

R9#show ip nbar protocol-id bittorrent

Protocol Name      id      type
-----
bittorrent         69      L7 STANDARD

```

Custom protocols get an internally allocated protocol-id, which are larger numbers.

```

R9#show ip nbar protocol-id FIND_BEEF
Protocol Name      id      type
-----
FIND_BEEF         242     Custom

```

You can specify this ID for tracking/collection purposes when you define a new protocol using the “id” suffix keyword. The relevance of this number for custom protocols is generally limited to Netflow exports because NBAR can be integrated with FNF (shown under the FNF section of this document).

```
R9(config)#ip nbar custom PROTO_ID destination tcp 55557 id 57
```

```
R9#show ip nbar protocol-id PROTO_ID
```

Protocol Name	id	type
PROTO_ID	57	Custom

30.6.5 NBAR Protocol Discovery

NBAR can also be used as a mechanism for protocol discovery. It's like Netflow except less granular, but also very easy and automatic. It can be enabled at the interface level for IPv4, IPv6, or both. We have applied it to the link facing CSR8. I've performed a file download via HTTP, a telnet session to CSR10, and some pings. The main show command by itself prints verbose output, showing several counters per protocol it learns.

```
! CSR9
interface GigabitEthernet2.589
 ip nbar protocol-discovery
```

```
R9#show ip nbar protocol-discovery
```

GigabitEthernet2.589

Last clearing of "show ip nbar protocol-discovery" counters 00:01:31

Protocol	Input	Output
	-----	-----
	Packet Count	Packet Count
	Byte Count	Byte Count
	5min Bit Rate (bps)	5min Bit Rate (bps)
	5min Max Bit Rate (bps)	5min Max Bit Rate (bps)
	-----	-----
HTTP_CEF_FILE	22	20
	1850	3380
	0	0
	0	1000
telnet	26	25
	1552	1517
	0	0
	0	0
ping	5	5
	590	590
	0	0
	0	0
unknown	10	5
	600	310
	0	0

	0	0
Total	63	55
	4592	5797
	0	0
	0	1000

We can narrow down this output by selecting only specific rows (packet count, byte count, etc).

```
R9#show ip nbar protocol-discovery stats packet-count
```

```
GigabitEthernet2.589
```

```
Last clearing of "show ip nbar protocol-discovery" counters 00:01:20
```

	Input	Output
	-----	-----
Protocol	Packet Count	Packet Count
-----	-----	-----
telnet	26	25
HTTP_CEF_FILE	22	20
ping	5	5
unknown	10	5
Total	63	55

We can narrow it even further by restricting it to a single protocol as well. When you do this, the “unknown” traffic is always displayed. In this particular case, the unknown traffic is the IP SLA traffic using bogus ports from earlier tests (some were still running).

```
R9#show ip nbar protocol-discovery stats packet-count protocol telnet
```

```
GigabitEthernet2.589
```

```
Last clearing of "show ip nbar protocol-discovery" counters 00:03:25
```

	Input	Output
	-----	-----
Protocol	Packet Count	Packet Count
-----	-----	-----
telnet	26	25
unknown	10	5
Total	63	55

The nice thing about this feature is that it honors your custom protocols at all times. Even without configuring MQC anywhere, you can see if your custom protocols are traversing the network. Earlier we looked at using NBAR to find a certain text string, specifically 0x0000BEEF. If we turn on that IP SLA again, NBAR should be able to see that protocol without MQC. Unfortunately, this feature doesn’t work well with our STOP_VIRUS protocol, because we abstracted the last 2 bytes to the MQC class-map, so

that protocol will essentially match anything on UDP 55556. That is one downside to the MQC abstraction method (with the “variable” keyword); it relies on MQC being configured and applied.

```
R9#show policy-map interface brief
[no output]
```

```
R9#show ip nbar protocol-discovery protocol FIND_BEEF
```

```
GigabitEthernet2.589
```

```
Last clearing of "show ip nbar protocol-discovery" counters 00:00:25
```

Protocol	Input	Output
	-----	-----
	Packet Count	Packet Count
	Byte Count	Byte Count
	5min Bit Rate (bps)	5min Bit Rate (bps)
	5min Max Bit Rate (bps)	5min Max Bit Rate (bps)

FIND_BEEF	5	0
	310	0
	0	0
	0	0

31. Describe, implement, and troubleshoot MPLS TE / QoS mechanisms

31.1 MPLS RSVP-TE (General)

The configurations for all MPLS TE features are contained in the files below (manual TE, FRR, CBTS/PBTS), with the exception of DS-TE and automatic tunnels. Each of those has an alternative architecture since it was not conducive to mix them in a test with the vast majority of existing TE features.

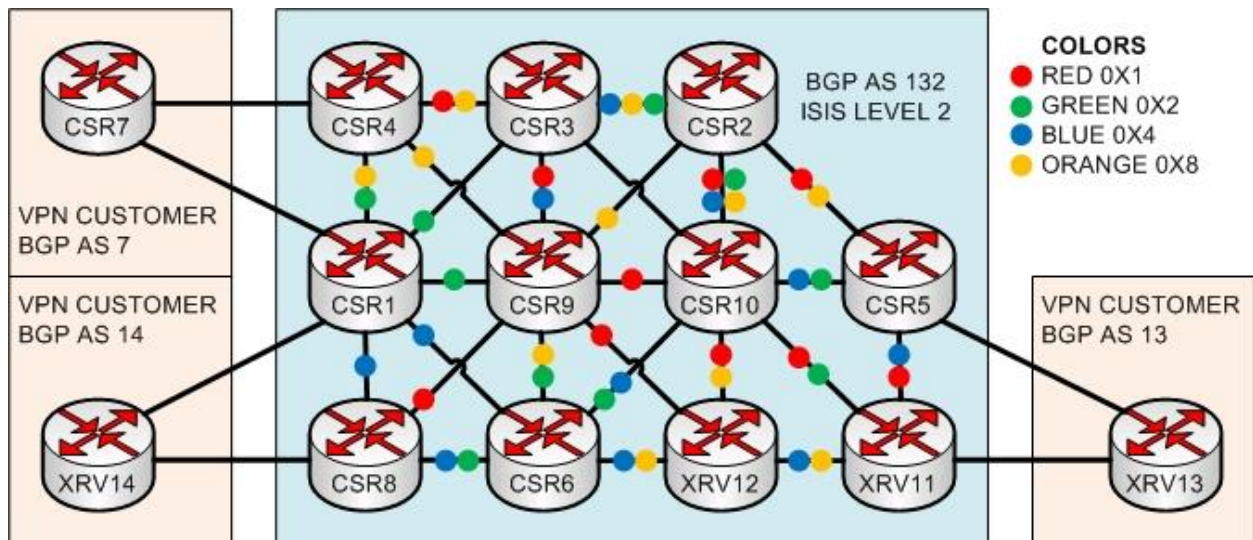
Additional Reading – Reference configurations “te-manual”

31.1.1 TE Topology (TED) construction and RSVP-TE signaling

MPLS traffic engineering is one of MPLS' most powerful features. It allows for completely arbitrary traffic steering from the source to destination based on a variety of attributes, such as available bandwidth, link colors, and other characteristics. OSPF and IS-IS were extended to carry these attributes. Also, RSVP was extended to RSVP-TE to support MPLS-TE, which includes some new objects.

First, we will examine how IS-IS exchanges TE information. The diagram is shown below. The topology is large and densely connected so as to create many routing paths. CSR7, XRv13, and XRv14 are all connected by both an L3VPN and L2VPN so we can demonstrate TE features for both (there isn't much difference, really). Most of the testing will use IS-IS but it isn't significant since both IS-IS and OSPF carry the same information, just in different ways. MPLS-TE attribute information is limited to an IS-IS level or

OSPF area, so the entire topology is IS-IS L2. OSPF will be examined later. Inter-area TE is examined in the Unified MPLS section while Inter-AS TE is examined in the Inter-AS MPLS sections.



Because everything is intra-level, every router can see every link characteristic in the topology. This means that we can fully validate a correct configuration only accessing one router. From the perspective of CSR1, we can check the TE topology and look specifically at the MPLS TE router-ID. This is normally a loopback on each router, and it must be configured under the IGP process. TE must also be enabled for an IS-IS level or OSPF area. In the case of IS-IS, wide metrics must be enabled to carry the TE attributes. A very cursory check of the topology at least shows we have all 11 LSRs running TE under IS-IS. We can see the IS-IS NETs and the TE-ID, which is the loopback0 IPv4 address.

```
! XE routers
router isis 132
metric-style wide
mpls traffic-eng router-id Loopback0
mpls traffic-eng level-2

! XR routers
router isis 132
address-family ipv4 unicast
metric-style wide
mpls traffic-eng router-id Loopback0
mpls traffic-eng level-2
```

```
CSR1#show mpls traffic-eng topology level-2 brief | include TE_Id
IGP Id: 0000.0000.0001.00, MPLS TE Id:1.1.1.1 Router Node (isis level-2)
IGP Id: 0000.0000.0002.00, MPLS TE Id:2.2.2.2 Router Node (isis level-2)
IGP Id: 0000.0000.0003.00, MPLS TE Id:3.3.3.3 Router Node (isis level-2)
IGP Id: 0000.0000.0004.00, MPLS TE Id:4.4.4.4 Router Node (isis level-2)
IGP Id: 0000.0000.0005.00, MPLS TE Id:5.5.5.5 Router Node (isis level-2)
IGP Id: 0000.0000.0006.00, MPLS TE Id:6.6.6.6 Router Node (isis level-2)
```

```
IGP Id: 0000.0000.0008.00, MPLS TE Id:8.8.8.8 Router Node (isis level-2)
IGP Id: 0000.0000.0009.00, MPLS TE Id:9.9.9.9 Router Node (isis level-2)
IGP Id: 0000.0000.0010.00, MPLS TE Id:10.10.10.10 Router Node (isis level-2)
IGP Id: 0000.0000.0011.00, MPLS TE Id:11.11.11.11 Router Node (isis level-2)
IGP Id: 0000.0000.0012.00, MPLS TE Id:12.12.12.12 Router Node (isis level-2)
```

If we forget to set an MPLS TE-ID on a router, this quick check will reveal that. It can be a simple mistake. We remove it under CSR3 as a test; it no longer shows up. CSR3 is now incapable of running MPLS-TE. We will see this in action later when we build TE tunnels.

```
CSR3(config-router)#no mpls traffic-eng router-id
```

```
CSR1#show mpls traffic-eng topology level-2 brief | include TE_Id
IGP Id: 0000.0000.0001.00, MPLS TE Id:1.1.1.1 Router Node (isis level-2)
IGP Id: 0000.0000.0002.00, MPLS TE Id:2.2.2.2 Router Node (isis level-2)
IGP Id: 0000.0000.0004.00, MPLS TE Id:4.4.4.4 Router Node (isis level-2)
IGP Id: 0000.0000.0005.00, MPLS TE Id:5.5.5.5 Router Node (isis level-2)
IGP Id: 0000.0000.0006.00, MPLS TE Id:6.6.6.6 Router Node (isis level-2)
IGP Id: 0000.0000.0008.00, MPLS TE Id:8.8.8.8 Router Node (isis level-2)
IGP Id: 0000.0000.0009.00, MPLS TE Id:9.9.9.9 Router Node (isis level-2)
IGP Id: 0000.0000.0010.00, MPLS TE Id:10.10.10.10 Router Node (isis level-2)
IGP Id: 0000.0000.0011.00, MPLS TE Id:11.11.11.11 Router Node (isis level-2)
IGP Id: 0000.0000.0012.00, MPLS TE Id:12.12.12.12 Router Node (isis level-2)
```

We will repair CSR3 by adding the TE-ID back. If we change the filter to be more generic and look at all of the "nodes" in the topology, we see a new node type at the bottom. The "network" node is the TE representation of an OSPF DR or IS-IS DIS. Every single link in the topology is OSPF/IS-IS point-to-point specifically to bypass the DR/DIS election, except for the link between CSR2-CSR3. This is just for demonstration purposes and has no operational benefit in this network. The point is that DR/DIS pseudonodes count as TE nodes and are included in the TE topology.

```
CSR1#show mpls traffic-eng topology level-2 brief | include ^IGP
IGP Id: 0000.0000.0001.00, MPLS TE Id:1.1.1.1 Router Node (isis level-2)
IGP Id: 0000.0000.0002.00, MPLS TE Id:2.2.2.2 Router Node (isis level-2)
IGP Id: 0000.0000.0003.00, MPLS TE Id:3.3.3.3 Router Node (isis level-2)
IGP Id: 0000.0000.0004.00, MPLS TE Id:4.4.4.4 Router Node (isis level-2)
IGP Id: 0000.0000.0005.00, MPLS TE Id:5.5.5.5 Router Node (isis level-2)
IGP Id: 0000.0000.0006.00, MPLS TE Id:6.6.6.6 Router Node (isis level-2)
IGP Id: 0000.0000.0008.00, MPLS TE Id:8.8.8.8 Router Node (isis level-2)
IGP Id: 0000.0000.0009.00, MPLS TE Id:9.9.9.9 Router Node (isis level-2)
IGP Id: 0000.0000.0010.00, MPLS TE Id:10.10.10.10 Router Node (isis level-2)
IGP Id: 0000.0000.0011.00, MPLS TE Id:11.11.11.11 Router Node (isis level-2)
IGP Id: 0000.0000.0012.00, MPLS TE Id:12.12.12.12 Router Node (isis level-2)
IGP Id: 0000.0000.0003.01, Network Node (isis level-2)
```

Tracing the entire topology is excellent practice for understanding how TE works, but we can validate a few select links for brevity. Looking at the router node for CSR1, we can see 5 attached links. This is consistent with the topology and it also links the neighbors by IS-IS NET. Provided the NETs are logical and easily understandable, we can see the neighbors are CSR8, CSR4, CSR6, CSR9, and CSR3, which is correct. The links are identified as point-to-point and this is very similar to reading OSPF LSA1s. Each link has specific attributes, including TE and IGP metrics, attribute flags (link colors discussed later), and shared risk link groups (SRLG). These detailed characters are validated in later sections.

```
CSR1#show mpls traffic-eng topology level-2 1.1.1.1 brief
```

```
IGP Id: 0000.0000.0001.00, MPLS TE Id:1.1.1.1 Router Node (isis level-2)
link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0008.00, nbr_node_id:14, gen:1838
frag_id: 0, Intf Address: 132.1.8.1, Nbr Intf Address: 132.1.8.8
TE metric: 1, IGP metric: 10, attribute flags: 0x4
SRLGs: None

link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0004.00, nbr_node_id:7, gen:1838
frag_id: 0, Intf Address: 132.1.4.1, Nbr Intf Address: 132.1.4.4
TE metric: 10, IGP metric: 10, attribute flags: 0xA
SRLGs: None

link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0006.00, nbr_node_id:12, gen:1838
frag_id: 0, Intf Address: 132.1.6.1, Nbr Intf Address: 132.1.6.6
TE metric: 1, IGP metric: 10, attribute flags: 0x4
SRLGs: None

link[3]: Point-to-Point, Nbr IGP Id: 0000.0000.0009.00, nbr_node_id:10, gen:1838
frag_id: 0, Intf Address: 132.1.9.1, Nbr Intf Address: 132.1.9.9
TE metric: 1, IGP metric: 10, attribute flags: 0x2
SRLGs: None

link[4]: Point-to-Point, Nbr IGP Id: 0000.0000.0003.00, nbr_node_id:8, gen:1838
frag_id: 0, Intf Address: 132.1.3.1, Nbr Intf Address: 132.1.3.3
TE metric: 10, IGP metric: 10, attribute flags: 0x2
SRLGs: None
```

Moving on to CSR3, we can see there is a link to CSR1, plus four more to CSR4, CSR9, CSR10, and a pseudonode identified as 0000.0000.0003.01. Notice that the show command references the IS-IS NET and not the loopback address as this is an alternative method of verification. Stepping through the network in this way is the best way to validate that the network has been configured properly.

```
CSR1#show mpls traffic-eng topology level-2 igp-id isis 0000.0000.0003.00 brief
```

```
IGP Id: 0000.0000.0003.00, MPLS TE Id:3.3.3.3 Router Node (isis level-2)
link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0001.00, nbr_node_id:6, gen:1847
frag_id: 0, Intf Address: 132.1.3.3, Nbr Intf Address: 132.1.3.1
TE metric: 10, IGP metric: 10, attribute flags: 0x2
SRLGs: None

link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0004.00, nbr_node_id:7, gen:1847
```

```
frag_id: 0, Intf Address: 132.3.4.3, Nbr Intf Address: 132.3.4.4
TE metric: 10, IGP metric: 10, attribute flags: 0x9
SRLGs: None
```

```
link[2]: Broadcast, DR: 0000.0000.0003.01, nbr_node_id:26, gen:1847
frag_id: 0, Intf Address: 132.2.3.3
TE metric: 10, IGP metric: 10, attribute flags: 0xE
SRLGs: None
```

```
link[3]: Point-to-Point, Nbr IGP Id: 0000.0000.0009.00, nbr_node_id:10, gen:1847
frag_id: 0, Intf Address: 132.3.9.3, Nbr Intf Address: 132.3.9.9
TE metric: 10, IGP metric: 10, attribute flags: 0x5
SRLGs: None
```

```
link[4]: Point-to-Point, Nbr IGP Id: 0000.0000.0010.00, nbr_node_id:18, gen:1847
frag_id: 0, Intf Address: 132.3.10.3, Nbr Intf Address: 132.3.10.10
TE metric: 10, IGP metric: 10, attribute flags: 0x0
SRLGs: None
```

Next, we examine the pseudonode, which is the only one in the topology. The IGP ID is similar to the IS-IS NET of the DIS, minus the area, and with a non-zero N-selector. CSR3 is the DIS and the pseudonode connects to CSR3 and CSR2. This node isn't a real physical router in the graph and does not have any specific MPLS-TE attributes. Just like an OSPF LSA2 or IS-IS DIS LSP, it just connects other routers; all costs are zero.

```
CSR1#show mpls traffic-eng topology level-2 igp-id isis 0000.0000.0003.01 brief
```

```
IGP Id: 0000.0000.0003.01, Network Node (isis level-2)
  link[0]: Broadcast, Nbr IGP Id: 0000.0000.0003.00, nbr_node_id:8, gen:1848
  link[1]: Broadcast, Nbr IGP Id: 0000.0000.0002.00, nbr_node_id:16, gen:1848
```

From this, we can look at CSR2. We will use the IP address format for variety in the show command. CSR2 is connected to the DIS (CSR3), CSR5, CSR9, and CSR10, which is correct per the diagram. This process should be repeated for all nodes in the topology to verify the TE database (TED) is correct.

```
CSR1#show mpls traffic-eng topology level-2 2.2.2.2 brief
```

```
IGP Id: 0000.0000.0002.00, MPLS TE Id:2.2.2.2 Router Node (isis level-2)
  link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0005.00, nbr_node_id:21, gen:1849
  frag_id: 0, Intf Address: 132.2.5.2, Nbr Intf Address: 132.2.5.5
  TE metric: 10, IGP metric: 10, attribute flags: 0x9
  SRLGs: None

  link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0009.00, nbr_node_id:10, gen:1849
  frag_id: 0, Intf Address: 132.2.9.2, Nbr Intf Address: 132.2.9.9
  TE metric: 10, IGP metric: 10, attribute flags: 0x1
  SRLGs: None

  link[2]: Broadcast, DR: 0000.0000.0003.01, nbr_node_id:26, gen:1849
  frag_id: 0, Intf Address: 132.2.3.2
```

TE metric: 10, IGP metric: 10, attribute flags: 0xE
SRLGs: None

link[3]: Point-to-Point, Nbr IGP Id: 0000.0000.0010.00, nbr_node_id:18, gen:1849
frag_id: 0, Intf Address: 132.2.10.2, Nbr Intf Address: 132.2.10.10
TE metric: 10, IGP metric: 10, attribute flags: 0xF
SRLGs: None

Omitting detailed "brief" keyword will also include all of the link bandwidth information for each of the eight available priorities (discussed later). This is how TE tracks how much bandwidth is available.

CSR1#show mpls traffic-eng topology level-2 2.2.2.2

IGP Id: 0000.0000.0002.00, MPLS TE Id:2.2.2.2 Router Node (isis level-2) id 16
link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0005.00, nbr_node_id:21, gen:1849
frag_id: 0, Intf Address: 132.2.5.2, Nbr Intf Address: 132.2.5.5
TE metric: 10, IGP metric: 10, attribute flags: 0x9
SRLGs: None
physical_bw: 1000000 (kbps), max_reservable_bw_global: 100000 (kbps)
max_reservable_bw_sub: 0 (kbps)

	Total Allocated BW (kbps)	Global Pool Reservable BW (kbps)	Sub Pool Reservable BW (kbps)
	-----	-----	-----
bw[0]:	0	100000	0
bw[1]:	0	100000	0
bw[2]:	0	100000	0
bw[3]:	0	100000	0
bw[4]:	0	100000	0
bw[5]:	0	100000	0
bw[6]:	0	100000	0
bw[7]:	0	100000	0

link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0009.00, nbr_node_id:10, gen:1849
frag_id: 0, Intf Address: 132.2.9.2, Nbr Intf Address: 132.2.9.9
TE metric: 10, IGP metric: 10, attribute flags: 0x1
SRLGs: None
physical_bw: 1000000 (kbps), max_reservable_bw_global: 100000 (kbps)
max_reservable_bw_sub: 0 (kbps)

	Total Allocated BW (kbps)	Global Pool Reservable BW (kbps)	Sub Pool Reservable BW (kbps)
	-----	-----	-----
bw[0]:	0	100000	0
bw[1]:	0	100000	0
bw[2]:	0	100000	0
bw[3]:	0	100000	0
bw[4]:	0	100000	0
bw[5]:	0	100000	0
bw[6]:	0	100000	0
bw[7]:	0	100000	0

[snip]

The parser syntax above is generic for all IGP's that support TE distribution, which makes it the preferred way to look the TED. The information itself is carried differently in different protocols. We can look at the raw information within IS-IS as well with the "verbose" option. From CSR1, we look at CSR8. As attributes of the individual links (the IS-Extended neighbors), all of the TE attributes are carried. The TE metric is only carried if it is modified, which we will see later. For low-level debugging, this is the best way to see the source of the information, especially if the TED information appears incorrect.

```
CSR1#show isis database level-2 CSR8.00-00 verbose
Tag 132:
IS-IS Level-2 LSP CSR8.00-00
LSPID                LSP Seq Num  LSP Checksum  LSP Holdtime  ATT/P/OL
CSR8.00-00           0x000002AC   0x8A57        853           0/0/0
  Auth:              Length: 17
  Area Address: 00
  NLPID:             0xCC 0x8E
  Topology:          IPv4 (0x0)
                   IPv6 (0x2)
  Router ID:         8.8.8.8
  Hostname: CSR8
  Metric: 10         IS (MT-IPv6) CSR9.00
  Metric: 10         IS (MT-IPv6) CSR6.00
  Metric: 10         IS (MT-IPv6) CSR1.00
  Metric: 10         IS-Extended CSR6.00
    Affinity: 0x00000006
    Interface IP Address: 132.6.8.8
    Neighbor IP Address: 132.6.8.6
    Physical BW: 1000000 kbits/sec
    Reservable Global Pool BW: 100000 kbits/sec
    Global Pool BW Unreserved:
      [0]: 100000 kbits/sec, [1]: 100000 kbits/sec
      [2]: 100000 kbits/sec, [3]: 100000 kbits/sec
      [4]: 100000 kbits/sec, [5]: 100000 kbits/sec
      [6]: 100000 kbits/sec, [7]: 100000 kbits/sec
  Metric: 10         IS-Extended CSR1.00
    Affinity: 0x00000004
    Interface IP Address: 132.1.8.8
    Neighbor IP Address: 132.1.8.1
    Physical BW: 1000000 kbits/sec
    Reservable Global Pool BW: 100000 kbits/sec
    Global Pool BW Unreserved:
      [0]: 100000 kbits/sec, [1]: 100000 kbits/sec
      [2]: 100000 kbits/sec, [3]: 100000 kbits/sec
      [4]: 100000 kbits/sec, [5]: 100000 kbits/sec
      [6]: 100000 kbits/sec, [7]: 100000 kbits/sec
  Metric: 10         IS-Extended CSR9.00
    Affinity: 0x00000001
    Interface IP Address: 132.8.9.8
```



```

Neighbor IP Address: 132.8.9.9
Physical BW: 1000000 kbits/sec
Reservable Global Pool BW: 100000 kbits/sec
Global Pool BW Unreserved:
  [0]: 100000 kbits/sec, [1]: 100000 kbits/sec
  [2]: 100000 kbits/sec, [3]: 100000 kbits/sec
  [4]: 100000 kbits/sec, [5]: 100000 kbits/sec
  [6]: 100000 kbits/sec, [7]: 100000 kbits/sec
IP Address: 8.8.8.8
Metric: 0 IP 8.8.8.8/32
IPv6 Address: ::8:8:8:8
Metric: 0 IPv6 (MT-IPv6) ::8:8:8:8/128

```

We can also look at topology information using some TE link-level commands. We can look at TE-IGP neighbors, link attributes, and other things. The commands are verbose and, for the most part, show the same information we've already seen, except with link-level granularity and in a different display format. Below we look at the TE neighbors as seen by IGP as well as the TE link summary. There isn't anything noteworthy about these commands by themselves, but I document them for completeness. Because this command details the IGP neighbor addresses as seen by TE, it might be valuable for explicit-path construction (discussed later).

```

CSR1#show mpls traffic-eng link-management igp-neighbors
Link ID:: Gi2.513
  Neighbor ID: 0000.0000.0003.00 (area: isis level-2, IP: 132.1.3.3)
              up, Sources: IGP
Link ID:: Gi2.514
  Neighbor ID: 0000.0000.0004.00 (area: isis level-2, IP: 132.1.4.4)
              up, Sources: IGP
Link ID:: Gi2.516
  Neighbor ID: 0000.0000.0006.00 (area: isis level-2, IP: 132.1.6.6)
              up, Sources: IGP
Link ID:: Gi2.518
  Neighbor ID: 0000.0000.0008.00 (area: isis level-2, IP: 132.1.8.8)
              up, Sources: IGP
Link ID:: Gi2.519
  Neighbor ID: 0000.0000.0009.00 (area: isis level-2, IP: 132.1.9.9)
              up, Sources: IGP

```

```

CSR1#show mpls traffic-eng link-management summary
System Information::
  Links Count: 5
  Flooding System: enabled
IGP Area ID:: isis level-2
  Flooding Protocol: ISIS
  Flooding Status: data flooded
  Periodic Flooding: enabled (every 60 seconds, next in 28 seconds)
  Flooded Links: 5
  IGP System ID: 0000.0000.0001.00

```

```

MPLS TE Router ID:    1.1.1.1
Neighbors:             5
Link ID:: Gi2.513 (132.1.3.1)
Local Intfc ID:       21
Link Status:
  SRLGs:               None
  Intfc Switching Capability Descriptors:
    Default:           Intfc Switching Cap psc1, Encoding ethernet
  Link Label Type:     Packet
  Physical Bandwidth:  1000000 kbits/sec
  Max Res Global BW:   100000 kbits/sec (reserved: 0% in, 3% out)
  Max Res Sub BW:      0 kbits/sec (reserved: 100% in, 100% out)
  MPLS TE Link State:  MPLS TE on, RSVP on, admin-up, flooded, allocated
  Inbound Admission:   reject-huge
  Outbound Admission:  allow-if-room
  Link MTU:            IP 1500, MPLS 1500
  Admin. Weight:       10 (IGP)
  IGP Neighbor Count:  1
[snip]

```

We can also look at LSP statistical information. This will show the number of setup/teardown requests, tunnels admitted, tunnels rejected, and other information. This can be helpful for troubleshooting tunnel flaps.

```

CSR1#show mpls traffic-eng link-management statistics
System Information::
  LSP Admission Statistics:
    Path:      140 setup requests, 139 admits, 1 rejects, 0 setup errors
               138 tear requests, 0 preempts, 0 tear errors
    Resv:      147 setup requests, 147 admits, 0 rejects, 0 setup errors
               133 tear requests, 0 preempts, 0 tear errors
Link ID:: Gi2.513 (132.1.3.1)
  Link Admission Statistics:
    Up Path:   38 setup requests, 38 admits, 0 rejects, 0 setup errors
               37 tear requests, 0 preempts, 0 tear errors
    Up Resv:   38 setup requests, 38 admits, 0 rejects, 0 setup errors
               71 tear requests, 0 preempts, 0 tear errors
    Down Path: 76 setup requests, 76 admits, 0 rejects, 0 setup errors
               75 tear requests, 0 preempts, 0 tear errors
    Down Resv: 84 setup requests, 84 admits, 0 rejects, 0 setup errors
               37 tear requests, 0 preempts, 0 tear errors
[snip]

```

The mechanism by which TE tunnels are signaled is RSVP-TE. Segment routing TE (SR-TE) is very new but not in scope for the lab exam, and is not documented in this chapter. There are three key messages used by RSVP to signal TE LSPs.

PATH: Message sent hop-by-hop from source to destination, or in MPLS terms, head to tail. It carries the previous hop (PHOP) so that the tail end knows how to reply along the same path. After RSVP-TE extensions, the PATH message carries new objects:

Label Request Object (LRO): Used to request TE labels along the path, but does not carry the label values directly as this information needs to flow from tail to head (RESV messages).

Explicit Route Object (ERO): This is the output of the path calculation (PCALC) algorithm which essentially tells the PATH messages which way to go. Since RSVP is used for signaling only, it is not expected to have direct visibility into the TED as this is PCALC's duty.

Record Route Object (RRO): Used to record the route taken by the PATH message, giving a "record" of the LSP to downstream nodes. This can be valuable when using loose-hop expansion to prevent signaling loops.

Session Attribute Object (SAO): Carries information about the session, such as the mode of operation (shared explicit, described later), node/bandwidth protection flags, and fast-reroute.

Sender Tspec: Carries bandwidth reservation information as an average rate.

RESV: Message sent hop-by-hop using the PHOPs learned from the PATH message. It is impossible for the RESV messages to follow a different path than the PATH messages. The next-hop (NHOP) is carried inside the RESV messages so that the upstream nodes know the proper direction of the LSP.

Label object: This message primarily carries the label object, which is the actual label value of the TE tunnel. Coupled with the NHOP value, this defines the LSP forwarding plane.

Record Route Object (RRO): Used to record the route taken by the RESV message, giving a "record" of the LSP to upstream nodes. This can be valuable when using loose-hop expansion to see the actual path that was signaled.

PATHERR: An error message to signal something went awry during RSVP signaling. This could be insufficient bandwidth or a link failure, as examples. Upon receipt of a PATHERR message, the headend will recompute the path.

First, we will examine PCALC operation in detail using a very basic TE tunnel from CSR1 to CSR5 [ID 1]. TE tunnel configuration in XE has some awkwardness; the tunnel is not running IP, yet requires an IP address in the form of the TE-ID (unnumbered). The tunnel source is unspecified, but the destination is the remote TE-ID (tail end). I also clear all affinity bits so link coloring is not a factor, and this is discussed later. The PCALC process is also called constrained shortest path first (CSPF), which takes all of the tunnel constraints into consideration, finds the shortest path that meets the constraints, and assembles the hop-by-hop path into an ERO. This gets fed into the RSVP process via the PATH message for

signaling. Before creating the tunnel, we enable debug on CSR1 so we can watch the PCALC process. The "lookup" debug is typically the more valuable one, but we can also check the "spf" debug occasionally for extra detail. The SPF output is extensive but we will analyze it once. The debug is broken into sections for clarity.

```
CSR1#debug mpls traffic-eng path lookup
CSR1#debug mpls traffic-eng path spf
```

```
! CSR1
interface Tunnel1
  description BASIC
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 5.5.5.5
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 dynamic
```

Towards the top of the debug, we see the "constraints" being fed into the algorithm as input. The process performs a lookup within IS-IS level-2 and finds a shortest path (there are many) to XRv11 via CSR9 and CSR10 based on these constraints; right now, there aren't any. The total path-cost is 30, which is based on the TE metrics contained within the TE topology. This metric is copied over from the IGP metric by default, which is 10 for IS-IS. The path lookup process can then begin to find a path from 1.1.1.1 to 5.5.5.5.

```
! CSR1
TE-PCALC-API: 1.1.1.1_1->5.5.5.5_1 {7}: P2P LSP Path Lookup called
TE-PCALC: 1.1.1.1_1->5.5.5.5_1 {7}: Path Request Info
  Flags: METRIC_TE
  IP explicit-path: None (dynamic)
  bw 0, min_bw 0, metric: 0
  setup_pri 7, hold_pri 7
  affinity_bits 0x0, affinity_mask 0x0
TE-PCALC-PATH: 1.1.1.1_1->5.5.5.5_1 {7}: Area (isis level-2) Path Lookup
begin
```

The first step of the SPF algorithm is to evaluate the neighbor links. First, the local node address is obviously in the path, so it is immediately move from the tentative (tent) list to the path list. The other 5 links to CSR1's neighbors are added to the tent list for evaluation. After this pass, the output shows the current tent list along with their administrative weights (also known as the TE metric, shown as "aw"), along with the minimum bandwidth and hop count. We can see the tentative list is actually a stack structure, since items added first are at the bottom of the tent list.

```
! CSR1
TE-PCALC-SPF: Begin SPF for tunnel1 to dest 0000.0000.0005.00
TE-PCALC-SPF: Added 0000.0000.0001.00 to tent list (aw 0, min_bw
18446744073709551615, prev_node(NULL))
```

```

TE-PCALC-SPF: Moved 0000.0000.0001.00 to path list
TE-PCALC-SPF: Evaluating link 132.1.8.8
TE-PCALC-SPF: Added 0000.0000.0008.00 to tent list (aw 10, min_bw 100000,
prev_node(0000.0000.0001.00))
TE-PCALC-SPF: Evaluating link 132.1.4.4
TE-PCALC-SPF: Added 0000.0000.0004.00 to tent list (aw 10, min_bw 100000,
prev_node(0000.0000.0001.00))
TE-PCALC-SPF: Evaluating link 132.1.3.3
TE-PCALC-SPF: Added 0000.0000.0003.00 to tent list (aw 10, min_bw 100000,
prev_node(0000.0000.0001.00))
TE-PCALC-SPF: Evaluating link 132.1.6.6
TE-PCALC-SPF: Added 0000.0000.0006.00 to tent list (aw 10, min_bw 100000,
prev_node(0000.0000.0001.00))
TE-PCALC-SPF: Evaluating link 132.1.9.9
TE-PCALC-SPF: Added 0000.0000.0009.00 to tent list (aw 10, min_bw 100000,
prev_node(0000.0000.0001.00))
TE-PCALC-SPF: rrr_pcalc_dump_tentative list:
    0000.0000.0009.00: (aw=10, min_bw=100000, hops=1)
    0000.0000.0006.00: (aw=10, min_bw=100000, hops=1)
    0000.0000.0003.00: (aw=10, min_bw=100000, hops=1)
    0000.0000.0004.00: (aw=10, min_bw=100000, hops=1)
    0000.0000.0008.00: (aw=10, min_bw=100000, hops=1)

```

At this point, all 5 options appear equal, so CSR1 must select one to continue running SPF. It selects the first option (the last one pushed onto the tent stack) and adds it to the path list. It then evaluates all of CSR9's links. Notice that only CSR10, CSR2, and XRv12 are considered tentative candidates and are added to the tent list; other paths are not due to being farther away from the destination. The tent list grows to include these new nodes, and CSR9 has been removed from the tent list (being added to the path list does this).

```

! CSR1
TE-PCALC-SPF: Moved 0000.0000.0009.00 to path list
TE-PCALC-SPF: Evaluating link 132.9.12.12
TE-PCALC-SPF: Added 0000.0000.0012.00 to tent list (aw 20, min_bw 100000,
prev_node(0000.0000.0009.00))
TE-PCALC-SPF: Evaluating link 132.6.9.6
TE-PCALC-SPF: Evaluating link 132.4.9.4
TE-PCALC-SPF: Evaluating link 132.8.9.8
TE-PCALC-SPF: Evaluating link 132.2.9.2
TE-PCALC-SPF: Added 0000.0000.0002.00 to tent list (aw 20, min_bw 100000,
prev_node(0000.0000.0009.00))
TE-PCALC-SPF: Evaluating link 132.3.9.3
TE-PCALC-SPF: Evaluating link 132.1.9.1
TE-PCALC-SPF: Evaluating link 132.9.10.10
TE-PCALC-SPF: Added 0000.0000.0010.00 to tent list (aw 20, min_bw 100000,
prev_node(0000.0000.0009.00))
TE-PCALC-SPF: rrr_pcalc_dump_tentative list:
    0000.0000.0006.00: (aw=10, min_bw=100000, hops=1)

```

```

0000.0000.0003.00: (aw=10, min_bw=100000, hops=1)
0000.0000.0004.00: (aw=10, min_bw=100000, hops=1)
0000.0000.0008.00: (aw=10, min_bw=100000, hops=1)
0000.0000.0010.00: (aw=20, min_bw=100000, hops=2)
0000.0000.0002.00: (aw=20, min_bw=100000, hops=2)
0000.0000.0012.00: (aw=20, min_bw=100000, hops=2)

```

The process continues for the next node in the tent stack, which is CSR6. All of its links are evaluated, but none are added to the tent list as all of them are suboptimal. CSR6 is added to the path list and thus removed from the tent list.

! CSR1

TE-PCALC-SPF: Moved 0000.0000.0006.00 to path list

TE-PCALC-SPF: Evaluating link 132.6.8.8

TE-PCALC-SPF: Evaluating link 132.1.6.1

TE-PCALC-SPF: Evaluating link 132.6.10.10

TE-PCALC-SPF: Evaluating link 132.6.12.12

TE-PCALC-SPF: Evaluating link 132.6.9.9

TE-PCALC-SPF: rrr_pcalc_dump_tentative list:

```

0000.0000.0003.00: (aw=10, min_bw=100000, hops=1)
0000.0000.0004.00: (aw=10, min_bw=100000, hops=1)
0000.0000.0008.00: (aw=10, min_bw=100000, hops=1)
0000.0000.0010.00: (aw=20, min_bw=100000, hops=2)
0000.0000.0002.00: (aw=20, min_bw=100000, hops=2)
0000.0000.0012.00: (aw=20, min_bw=100000, hops=2)

```

The process continues for other nodes, including CSR3, CSR4, CSR8, and the DIS pseudonode representing the CSR2-CSR3 link. At the end of the evaluation, the tent list has been reduced to CSR10, CSR2, and XRv12.

! CSR1

TE-PCALC-SPF: Moved 0000.0000.0003.00 to path list

TE-PCALC-SPF: Evaluating link 132.3.4.4

TE-PCALC-SPF: Evaluating link 132.1.3.1

TE-PCALC-SPF: Added 0000.0000.0003.01 to tent list (aw 20, min_bw 100000, prev_node(0000.0000.0003.00))

TE-PCALC-SPF: rrr_pcalc_dump_tentative list:

```

0000.0000.0004.00: (aw=10, min_bw=100000, hops=1)
0000.0000.0008.00: (aw=10, min_bw=100000, hops=1)
0000.0000.0003.01: (aw=20, min_bw=100000, hops=2)
0000.0000.0010.00: (aw=20, min_bw=100000, hops=2)
0000.0000.0002.00: (aw=20, min_bw=100000, hops=2)
0000.0000.0012.00: (aw=20, min_bw=100000, hops=2)

```

TE-PCALC-SPF: Moved 0000.0000.0004.00 to path list

TE-PCALC-SPF: Evaluating link 132.3.4.3

TE-PCALC-SPF: Evaluating link 132.1.4.1

TE-PCALC-SPF: rrr_pcalc_dump_tentative list:

```

0000.0000.0008.00: (aw=10, min_bw=100000, hops=1)

```

```

0000.0000.0003.01: (aw=20, min_bw=100000, hops=2)
0000.0000.0010.00: (aw=20, min_bw=100000, hops=2)
0000.0000.0002.00: (aw=20, min_bw=100000, hops=2)
0000.0000.0012.00: (aw=20, min_bw=100000, hops=2)
TE-PCALC-SPF: Moved 0000.0000.0008.00 to path list
TE-PCALC-SPF: Evaluating link 132.6.8.6
TE-PCALC-SPF: Evaluating link 132.1.8.1
TE-PCALC-SPF: Evaluating link 132.8.9.9
TE-PCALC-SPF: rrr_pcalc_dump_tentative list:
0000.0000.0003.01: (aw=20, min_bw=100000, hops=2)
0000.0000.0010.00: (aw=20, min_bw=100000, hops=2)
0000.0000.0002.00: (aw=20, min_bw=100000, hops=2)
0000.0000.0012.00: (aw=20, min_bw=100000, hops=2)
TE-PCALC-SPF: Moved 0000.0000.0003.01 to path list
TE-PCALC-SPF: rrr_pcalc_dump_tentative list:
0000.0000.0010.00: (aw=20, min_bw=100000, hops=2)
0000.0000.0002.00: (aw=20, min_bw=100000, hops=2)
0000.0000.0012.00: (aw=20, min_bw=100000, hops=2)

```

The evaluation continues as the tent list continues to shrink in length. When CSR2 finally gets added to the path list, evaluating the links yields the final destination. The final evaluation shows a path to CSR5.

```

! CSR1
TE-PCALC-SPF: Moved 0000.0000.0010.00 to path list
TE-PCALC-SPF: Evaluating link 132.10.12.12
TE-PCALC-SPF: Evaluating link 132.2.10.2
TE-PCALC-SPF: No nbr node and Nbr 132.2.10.2 not in static tree
TE-PCALC-SPF: Evaluating link 132.6.10.6
TE-PCALC-SPF: Evaluating link 132.10.11.11
TE-PCALC-SPF: Added 0000.0000.0011.00 to tent list (aw 30, min_bw 100000,
prev_node(0000.0000.0010.00))
TE-PCALC-SPF: Evaluating link 132.3.10.3
TE-PCALC-SPF: Evaluating link 132.5.10.5
TE-PCALC-SPF: rrr_pcalc_dijkstra_spf: No acceptable reverse link for
132.5.10.10 in unidirectional path
TE-PCALC-SPF: Evaluating link 132.9.10.9
TE-PCALC-SPF: rrr_pcalc_dump_tentative list:
0000.0000.0002.00: (aw=20, min_bw=100000, hops=2)
0000.0000.0012.00: (aw=20, min_bw=100000, hops=2)
0000.0000.0011.00: (aw=30, min_bw=100000, hops=3)
TE-PCALC-SPF: Moved 0000.0000.0002.00 to path list
TE-PCALC-SPF: Evaluating link 132.2.5.5
TE-PCALC-SPF: Added 0000.0000.0005.00 to tent list (aw 30, min_bw 100000,
prev_node(0000.0000.0002.00))
TE-PCALC-SPF: Evaluating link 132.2.9.9
TE-PCALC-SPF: rrr_pcalc_dump_tentative list:
0000.0000.0012.00: (aw=20, min_bw=100000, hops=2)
0000.0000.0005.00: (aw=30, min_bw=100000, hops=3)
0000.0000.0011.00: (aw=30, min_bw=100000, hops=3)

```

```

TE-PCALC-SPF: Moved 0000.0000.0012.00 to path list
TE-PCALC-SPF: Evaluating link 132.6.12.6
TE-PCALC-SPF: Evaluating link 132.10.12.10
TE-PCALC-SPF: Evaluating link 132.9.12.9
TE-PCALC-SPF: Evaluating link 132.11.12.11
TE-PCALC-SPF: rrr_pcalc_dump_tentative list:
    0000.0000.0005.00: (aw=30, min_bw=100000, hops=3)
    0000.0000.0011.00: (aw=30, min_bw=100000, hops=3)

```

The SPF process shows this final path in reverse order since the algorithm stacks the vertices from tail to head. The final path is CSR1 > CSR9 > CSR2 > CSR5 and has a cost of 30. The remaining messages state, in various ways, that the tunnel was built successfully.

```

! CSR1
TE-PCALC-PATH: Path from 0000.0000.0001.00 -> 0000.0000.0005.00:
    132.2.5.2->132.2.5.5 (admin_weight=30):
    132.2.9.9->132.2.9.2 (admin_weight=20):
    132.1.9.1->132.1.9.9 (admin_weight=10):
    num_hops 4, accumulated_aw 30, min_bw 100000
TE-PCALC-PATH: 1.1.1.1_1->5.5.5.5_1 {7}: Area (isis level-2) Path Lookup
end: path found
TE-PCALC-API: 1.1.1.1_1->5.5.5.5_1 {7}: P2P LSP Path Lookup result: success
%MPLS_TE-5-TUN: Tun1: installed LSP 1_1 (popt 10) for nil, got 1st feasible
path opt
%MPLS_TE-5-LSP: LSP 1.1.1.1 1_1: UP
%MPLS_TE-5-TUN: Tun1: LSP path change 1_1 for nil, normal
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnell1, changed state to up

```

The entire PCALC computation happens locally on CSR1. The output of PCALC is the ERO, which is essentially the path that RSVP should signal. Provided the TED is correct, PCALC can complete. We will flap the tunnel (shut/no shut) with RSVP debugging enabled so we can view the signaling process also. First, CSR1 sees an incoming PROXY_PATH, which is basically a PATH message generated locally on the router from some other process (PCALC, in this case). The key part of this message, at least for now, is the ERO. Other things like the sender Tspec (bandwidth reservation) will be discussed later.

```

! CSR1
Incoming PROXY_PATH:
  version:1 flags:0000 cksum:0000 ttl:255 reserved:0 length:208
SESSION          type 7 length 16:
  Tun Dest:      5.5.5.5  Tun ID: 1  Ext Tun ID: 1.1.1.1
HOP              type 1 length 12:
  Hop Addr: 127.0.0.1  LIH: 0x00000000
TIME_VALUES      type 1 length 8 :
  Refresh Period (msec): 30000
EXPLICIT_ROUTE   type 1 length 44:
  1.1.1.1 (Strict IPv4 Prefix, 8 bytes, /32)
  132.1.9.9 (Strict IPv4 Prefix, 8 bytes, /32)

```



```

132.2.9.2 (Strict IPv4 Prefix, 8 bytes, /32)
132.2.5.5 (Strict IPv4 Prefix, 8 bytes, /32)
5.5.5.5 (Strict IPv4 Prefix, 8 bytes, /32)
SESSION_ATTRIBUTE      type 7 length 16:
  Setup Prio: 7, Holding Prio: 7
  Flags: (0x4) SE Style
  Session Name: BASIC
SENDER_TEMPLATE        type 7 length 12:
  Tun Sender: 1.1.1.1  LSP ID: 5
SENDER_TSPEC           type 2 length 36:
  version=0, length in words=7
  Token bucket fragment (service_id=1, length=6 words
    parameter id=127, flags=0, parameter length=5
    average rate=0 bytes/sec, burst depth=1000 bytes
    peak rate    =0 bytes/sec
    min unit=0 bytes, max pkt size=2147483647 bytes
ADSPEC                 type 2 length 48:
  version=0  length in words=10
  General Parameters  break bit=0  service length=8
                                     IS Hops:0
                                     Minimum Path Bandwidth (bytes/sec):2147483647
                                     Path Latency (microseconds):0
                                     Path MTU:4294967295
Controlled Load Service  break bit=0  service length=0
LABEL_REQUEST          type 1 length 8 :
  Layer 3 protocol ID: 2048

```

CSR1 turns this into a "real" PATH message and sends it to CSR9, which is the next item in the explicit path. Notice the interface address is used since TE is selecting specific links, not just selecting nodes. Notice that the minimum bandwidth towards the bottom is 125 MBps, or 1 Gbps, which is the link speed as seen by RSVP. By looking at "debug ip rsvp dump-messages" we can see all the details of the RSVP signaling.

```

! CSR1
%MPLS_TE-5-TUN: Tun1: installed LSP 1_5 (popt 10) for nil, got 1st feasible
path opt

```

Outgoing Path:

```

  version:1 flags:0000 cksum:7A91 ttl:255 reserved:0 length:200
SESSION              type 7 length 16:
  Tun Dest: 5.5.5.5  Tun ID: 1  Ext Tun ID: 1.1.1.1
HOP                  type 1 length 12:
  Hop Addr: 132.1.9.1  LIH: 0x00000011
TIME_VALUES          type 1 length 8 :
  Refresh Period (msec): 30000
EXPLICIT_ROUTE       type 1 length 36:
  132.1.9.9 (Strict IPv4 Prefix, 8 bytes, /32)
  132.2.9.2 (Strict IPv4 Prefix, 8 bytes, /32)
  132.2.5.5 (Strict IPv4 Prefix, 8 bytes, /32)

```

```

5.5.5.5 (Strict IPv4 Prefix, 8 bytes, /32)
LABEL_REQUEST      type 1 length 8 :
  Layer 3 protocol ID: 2048
SESSION_ATTRIBUTE  type 7 length 16:
  Setup Prio: 7, Holding Prio: 7
  Flags: (0x4) SE Style
  Session Name: BASIC
SENDER_TEMPLATE    type 7 length 12:
  Tun Sender: 1.1.1.1 LSP ID: 5
SENDER_TSPEC       type 2 length 36:
  version=0, length in words=7
  Token bucket fragment (service_id=1, length=6 words
    parameter id=127, flags=0, parameter length=5
    average rate=0 bytes/sec, burst depth=1000 bytes
    peak rate    =0 bytes/sec
    min unit=0 bytes, max pkt size=2147483647 bytes
ADSPEC            type 2 length 48:
  version=0 length in words=10
  General Parameters break bit=0 service length=8
                                     IS Hops:1
                               Minimum Path Bandwidth (bytes/sec):125000000
                               Path Latency (microseconds):0
                               Path MTU:1500
Controlled Load Service break bit=0 service length=0

```

CSR9 receives this path message from CSR1; notice that the PHOP is included so CSR9 knows where to send the RESV message. The top of the ERO is a local address on CSR9, which is expected.

```

! CSR9
Incoming Path:
  version:1 flags:0000 cksum:7A91 ttl:255 reserved:0 length:200
SESSION      type 7 length 16:
  Tun Dest:   5.5.5.5 Tun ID: 1 Ext Tun ID: 1.1.1.1
HOP          type 1 length 12:
  Hop Addr: 132.1.9.1 LIH: 0x00000011
TIME_VALUES  type 1 length 8 :
  Refresh Period (msec): 30000
EXPLICIT_ROUTE type 1 length 36:
  132.1.9.9 (Strict IPv4 Prefix, 8 bytes, /32)
  132.2.9.2 (Strict IPv4 Prefix, 8 bytes, /32)
  132.2.5.5 (Strict IPv4 Prefix, 8 bytes, /32)
  5.5.5.5 (Strict IPv4 Prefix, 8 bytes, /32)
[snip]

```

CSR9 strips its own address out of the ERO, updates PHOP, and sends a PATH message to the next-hop in the ERO, which is CSR2.

```

! CSR9

```

Outgoing Path:

```
version:1 flags:0000 cksum:29AF ttl:254 reserved:0 length:192
SESSION          type 7 length 16:
  Tun Dest:      5.5.5.5  Tun ID: 1  Ext Tun ID: 1.1.1.1
HOP              type 1 length 12:
  Hop Addr: 132.2.9.9  LIH: 0x0000000C
TIME_VALUES      type 1 length 8 :
  Refresh Period (msec): 30000
EXPLICIT_ROUTE   type 1 length 28:
  132.2.9.2 (Strict IPv4 Prefix, 8 bytes, /32)
  132.2.5.5 (Strict IPv4 Prefix, 8 bytes, /32)
  5.5.5.5 (Strict IPv4 Prefix, 8 bytes, /32)
[snip]
```

CSR2 receives the PATH message from CSR9 and the process continues. CSR2 creates a new PATH message towards CSR5, updates PHOP, and removes its own address from the ERO.

! CSR2

Incoming Path:

```
version:1 flags:0000 cksum:29AF ttl:254 reserved:0 length:192
SESSION          type 7 length 16:
  Tun Dest:      5.5.5.5  Tun ID: 1  Ext Tun ID: 1.1.1.1
HOP              type 1 length 12:
  Hop Addr: 132.2.9.9  LIH: 0x0000000C
TIME_VALUES      type 1 length 8 :
  Refresh Period (msec): 30000
EXPLICIT_ROUTE   type 1 length 28:
  132.2.9.2 (Strict IPv4 Prefix, 8 bytes, /32)
  132.2.5.5 (Strict IPv4 Prefix, 8 bytes, /32)
  5.5.5.5 (Strict IPv4 Prefix, 8 bytes, /32)
[snip]
```

Outgoing Path:

```
version:1 flags:0000 cksum:DCD0 ttl:253 reserved:0 length:184
SESSION          type 7 length 16:
  Tun Dest:      5.5.5.5  Tun ID: 1  Ext Tun ID: 1.1.1.1
HOP              type 1 length 12:
  Hop Addr: 132.2.5.2  LIH: 0x0000000D
TIME_VALUES      type 1 length 8 :
  Refresh Period (msec): 30000
EXPLICIT_ROUTE   type 1 length 20:
  132.2.5.5 (Strict IPv4 Prefix, 8 bytes, /32)
  5.5.5.5 (Strict IPv4 Prefix, 8 bytes, /32)
[snip]
```

CSR5 receives the PATH message and knows that it is the tail end of the LSP (tunnel destination indicates this). Now that the PATH message signaling from head to tail is done, the RESV messages must be sent in the reverse direction. The PHOPs carried in the PATH messages are used to derive next-hops (NHOPs)

for the RESV messages on the way back. This ensures that the proper interfaces are used in the reserve path. The most significant part of the RESV message is the last line, which is the label. PHP is supported with RSVP-TE so the tail-end router typically allocates implicit-null.

! CSR5

Incoming Path:

```
version:1 flags:0000 cksum:DCD0 ttl:253 reserved:0 length:184
SESSION          type 7 length 16:
  Tun Dest:      5.5.5.5  Tun ID: 1  Ext Tun ID: 1.1.1.1
HOP              type 1 length 12:
  Hop Addr: 132.2.5.2 LIH: 0x0000000D
TIME_VALUES      type 1 length 8 :
  Refresh Period (msec): 30000
EXPLICIT_ROUTE   type 1 length 20:
  132.2.5.5 (Strict IPv4 Prefix, 8 bytes, /32)
  5.5.5.5 (Strict IPv4 Prefix, 8 bytes, /32)
[snip]
```

Outgoing Resv:

```
version:1 flags:0000 cksum:E141 ttl:255 reserved:0 length:108
SESSION          type 7 length 16:
  Tun Dest:      5.5.5.5  Tun ID: 1  Ext Tun ID: 1.1.1.1
HOP              type 1 length 12:
  Hop Addr: 132.2.5.5 LIH: 0x0000000D
TIME_VALUES      type 1 length 8 :
  Refresh Period (msec): 30000
STYLE            type 1 length 8 :
  Shared-Explicit (SE)
FLOWSPEC         type 2 length 36:
  version = 0 length in words = 7
  service id = 5, service length = 6
  tspec parameter id = 127, flags = 0, length = 5
  average rate = 0 bytes/sec, burst depth = 1000 bytes
  peak rate    = 0 bytes/sec
  min unit = 0 bytes, max pkt size = 1500 bytes
FILTER_SPEC      type 7 length 12:
  Tun Sender: 1.1.1.1, LSP ID: 5
LABEL            type 1 length 8 :
  Labels: 3
```

CSR2 receives this RESV message from CSR5, programs the label into its LFIB for this LSP, and sends another RESV back to CSR9. CSR2 allocates label 2008 for this LSP. Label 3 is implicit-null, which makes sense since CSR2 is the penultimate hop along the TE LSP. Just like LDP, having the next-hop plus label effectively creates a link in the FEC. This makes RSVP a good candidate for MPLS-TE since the RESV message nicely carries both, similar to LDP.

! CSR2

Incoming Resv:

```
version:1 flags:0000 cksum:E141 ttl:255 reserved:0 length:108
SESSION          type 7 length 16:
  Tun Dest:      5.5.5.5  Tun ID: 1  Ext Tun ID: 1.1.1.1
HOP              type 1 length 12:
  Hop Addr: 132.2.5.5  LIH: 0x0000000D
TIME_VALUES      type 1 length 8 :
  Refresh Period (msec): 30000
STYLE           type 1 length 8 :
  Shared-Explicit (SE)
FLOWSPEC         type 2 length 36:
  version = 0 length in words = 7
  service id = 5, service length = 6
  tspec parameter id = 127, flags = 0, length = 5
  average rate = 0 bytes/sec, burst depth = 1000 bytes
  peak rate      = 0 bytes/sec
  min unit = 0 bytes, max pkt size = 1500 bytes
FILTER_SPEC      type 7 length 12:
  Tun Sender: 1.1.1.1, LSP ID: 5
LABEL           type 1 length 8 :
  Labels: 3
```

Outgoing Resv:

```
version:1 flags:0000 cksum:D570 ttl:255 reserved:0 length:108
SESSION          type 7 length 16:
  Tun Dest:      5.5.5.5  Tun ID: 1  Ext Tun ID: 1.1.1.1
HOP              type 1 length 12:
  Hop Addr: 132.2.9.2  LIH: 0x0000000C
TIME_VALUES      type 1 length 8 :
  Refresh Period (msec): 30000
STYLE           type 1 length 8 :
  Shared-Explicit (SE)
FLOWSPEC         type 2 length 36:
  version = 0 length in words = 7
  service id = 5, service length = 6
  tspec parameter id = 127, flags = 0, length = 5
  average rate = 0 bytes/sec, burst depth = 1000 bytes
  peak rate      = 0 bytes/sec
  min unit = 0 bytes, max pkt size = 1500 bytes
FILTER_SPEC      type 7 length 12:
  Tun Sender: 1.1.1.1, LSP ID: 5
LABEL           type 1 length 8 :
  Labels: 2008
```

CSR9 receives the RESV message from CSR2, programs label 2008 into its LFIB for this LSP, and sends a new RESV with local label 9013 towards CSR1.

! CSR9

Incoming Resv:

```
version:1 flags:0000 cksum:D570 ttl:255 reserved:0 length:108
SESSION          type 7 length 16:
  Tun Dest:      5.5.5.5  Tun ID: 1  Ext Tun ID: 1.1.1.1
HOP              type 1 length 12:
  Hop Addr: 132.2.9.2  LIH: 0x0000000C
TIME_VALUES      type 1 length 8 :
  Refresh Period (msec): 30000
STYLE           type 1 length 8 :
  Shared-Explicit (SE)
FLOWSPEC         type 2 length 36:
  version = 0 length in words = 7
  service id = 5, service length = 6
  tspec parameter id = 127, flags = 0, length = 5
  average rate = 0 bytes/sec, burst depth = 1000 bytes
  peak rate      = 0 bytes/sec
  min unit = 0 bytes, max pkt size = 1500 bytes
FILTER_SPEC      type 7 length 12:
  Tun Sender: 1.1.1.1, LSP ID: 5
LABEL           type 1 length 8 :
  Labels: 2008
```

Outgoing Resv:

```
version:1 flags:0000 cksum:BA08 ttl:255 reserved:0 length:108
SESSION          type 7 length 16:
  Tun Dest:      5.5.5.5  Tun ID: 1  Ext Tun ID: 1.1.1.1
HOP              type 1 length 12:
  Hop Addr: 132.1.9.9  LIH: 0x00000011
TIME_VALUES      type 1 length 8 :
  Refresh Period (msec): 30000
STYLE           type 1 length 8 :
  Shared-Explicit (SE)
FLOWSPEC         type 2 length 36:
  version = 0 length in words = 7
  service id = 5, service length = 6
  tspec parameter id = 127, flags = 0, length = 5
  average rate = 0 bytes/sec, burst depth = 1000 bytes
  peak rate      = 0 bytes/sec
  min unit = 0 bytes, max pkt size = 1500 bytes
FILTER_SPEC      type 7 length 12:
  Tun Sender: 1.1.1.1, LSP ID: 5
LABEL           type 1 length 8 :
  Labels: 9013
```

The head-end, CSR1, receives the final RESV message and programs this label into the LFIB. The TE tunnel signaling is now complete.

! CSR1

Incoming Resv:

```

version:1 flags:0000 cksum:BA08 ttl:255 reserved:0 length:108
SESSION          type 7 length 16:
  Tun Dest:      5.5.5.5  Tun ID: 1  Ext Tun ID: 1.1.1.1
HOP              type 1 length 12:
  Hop Addr: 132.1.9.9  LIH: 0x00000011
TIME_VALUES      type 1 length 8 :
  Refresh Period (msec): 30000
STYLE           type 1 length 8 :
  Shared-Explicit (SE)
FLOWSPEC         type 2 length 36:
  version = 0 length in words = 7
  service id = 5, service length = 6
  tspec parameter id = 127, flags = 0, length = 5
  average rate = 0 bytes/sec, burst depth = 1000 bytes
  peak rate      = 0 bytes/sec
  min unit = 0 bytes, max pkt size = 1500 bytes
FILTER_SPEC      type 7 length 12:
  Tun Sender: 1.1.1.1, LSP ID: 5
LABEL           type 1 length 8 :
  Labels: 9013

```

We can also look at a terser version of the RSVP signaling with "debug ip rsvp signalling". Refreshing the tunnel again, we see CSR1 send a PATH message to CSR9 with a PHOP of its local link address facing CSR9. The ERO details are not revealed here, but the basic RSVP message type (PATH, RESV, etc) and critical hop information (PHOP, NHOP, etc), are shown.

```

! CSR1
RSVP: 1.1.1.1_5->5.5.5.5_1[Src] {7}: Path refresh, Event: none, State: stay
in normal
RSVP: 1.1.1.1_5->5.5.5.5_1[Src] {7}: Path refresh (msec), config: 30000 curr:
30000 xmit: 30000
RSVP: 1.1.1.1_5->5.5.5.5_1[Src] {7}: Sending Path message to 132.1.9.9
RSVP: 1.1.1.1_5->5.5.5.5_1[Src] {7}: building hop object with src addr:
132.1.9.1

```

CSR9 receives this PATH message, updates the PHOP, and sends it to CSR2. We won't trace the remainder of the path as it is the same at each hop. CSR9 receives a RESV back in from CSR2 and sends it back to CSR1.

```

! CSR9
RSVP: 1.1.1.1_5->5.5.5.5_1[Src] {7}: Received Path message from 132.1.9.1 (on
GigabitEthernet2.519)
  RSVP: 1.1.1.1_5->5.5.5.5_1[Src] {7}: [rsvp_examine_and_mark_md_events]
Existing PSB MD = Ignore
  RSVP: 1.1.1.1_5->5.5.5.5_1[Src] {7}: [rsvp_examine_and_mark_tspec_events]
Existing PSB TSpec = Ignore
RSVP: 1.1.1.1_5->5.5.5.5_1[Src] {7}: Incoming Path, No change

```

```

RSVP: 1.1.1.1_5->5.5.5.5_1[Src] {7}: Sending Resv message to 132.1.9.1 from
132.1.9.9
RSVP: 1.1.1.1_5->5.5.5.5_1[Src] {7}: building hop object with src addr:
132.1.9.9

```

CSR1 receives the RESV and sees no change.

```

! CSR1
RSVP: session 5.5.5.5_1[1.1.1.1] (7): Received Resv message from 132.1.9.9
(on GigabitEthernet2.519)
RSVP: 1.1.1.1_5->5.5.5.5_1[Src] {7}: Successfully parsed Resv message from
132.1.9.9 (on GigabitEthernet2.519)
RSVP: 1.1.1.1_5->5.5.5.5_1[Src] {7}: No change in reservation

```

We can verify the tunnel with several show commands also. The primary show command is below and details most of the critical tunnel components. The first line shows the status of the interface. "Path" determines whether PCALC was successful and "Signalling" determines whether the path has been actually set up. A broken RSVP configuration won't affect PCALC, so PCALC can be valid while RSVP signaling is broken. The path options are listed and the one being used is identified as "basis for setup", along with the cost of the path. The next several lines just show configuration details. Next, there is RSVP information, to include the labels being used. There is no incoming label since this is the tunnel head; traffic only exits, never enters. The ERO, RRO, Tspec, and Fspec attributes are shown under the PATH and RESV fields as well.

```

CSR1#show mpls traffic-eng tunnels tunnel 1

```

```

Name: BASIC                                     (Tunnell) Destination: 5.5.5.5
Status:
  Admin: up      Oper: up      Path: valid      Signalling: connected
  path option 10, type dynamic (Basis for Setup, path weight 30)

Config Parameters:
  Bandwidth: 0      kbps (Global)  Priority: 7 7  Affinity: 0x0/0x0
  Metric Type: TE (default)
  AutoRoute: disabled LockDown: disabled Loadshare: 0 [0] bw-based
  auto-bw: disabled
Active Path Option Parameters:
  State: dynamic path option 10 is active
  BandwidthOverride: disabled LockDown: disabled Verbatim: disabled

InLabel  : -
OutLabel  : GigabitEthernet2.519, 9013
Next Hop  : 132.1.9.9
RSVP Signalling Info:
  Src 1.1.1.1, Dst 5.5.5.5, Tun_Id 1, Tun_Instance 5
  RSVP Path Info:
    My Address: 132.1.9.1

```



```

Explicit Route: 132.1.9.9 132.2.9.2 132.2.5.5 5.5.5.5
Record Route: NONE
Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits
RSVP Resv Info:
Record Route: NONE
Fspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits
Shortest Unconstrained Path Info:
Path Weight: 30 (TE)
Explicit Route: 132.1.9.9 132.2.9.2 132.2.5.5 5.5.5.5
History:
Tunnel:
Time since created: 1 hours, 15 minutes
Time since path change: 57 minutes, 24 seconds
Number of LSP IDs (Tun_Instances) used: 5
Current LSP: [ID: 5]
Uptime: 57 minutes, 24 seconds
Prior LSP: [ID: 4]
ID: path option unknown
Removal Trigger: tunnel shutdown

```

You can verify the tunnel signaling by checking RSVP directly as well. We can verify the PATH messages in brief and verbose form. "session-type 7" refers to RSVP-TE tunnels as opposed to regular IPv4 RSVP. This original flavor of RSVP is not commonly used since the Integrated Services (IntServ) QoS model is unpopular. The tunnel head has no PHOP or incoming interface, but all other routers will. The detailed output is similar and shows in incoming and outgoing ERO (before and after the local address is removed), along with other traffic parameters, such as bandwidth and fast-reroute. The detailed output is also very similar to the "dump-messages" debugging in terms of format.

```

CSR1#show ip rsvp sender filter session-type 7 tunnel-id 1
Destination      Tun Sender      TunID LSPID Prev Hop      I/F      BPS
5.5.5.5          1.1.1.1         1     5     none          none      0

```

```

CSR1#show ip rsvp sender detail filter session-type 7 tunnel-id 1
PATH:

```

```

Tun Dest: 5.5.5.5 Tun ID: 1 Ext Tun ID: 1.1.1.1
Tun Sender: 1.1.1.1 LSP ID: 5
Path refreshes:
sent: to NHOP 132.1.9.9 on GigabitEthernet2.519
Session Attr:
Setup Prio: 7, Holding Prio: 7
Flags: (0x4) SE Style
Session Name: BASIC

```

```

ERO: (incoming)
1.1.1.1 (Strict IPv4 Prefix, 8 bytes, /32)
132.1.9.9 (Strict IPv4 Prefix, 8 bytes, /32)
132.2.9.2 (Strict IPv4 Prefix, 8 bytes, /32)
132.2.5.5 (Strict IPv4 Prefix, 8 bytes, /32)
5.5.5.5 (Strict IPv4 Prefix, 8 bytes, /32)

```

ERO: (outgoing)

```
132.1.9.9 (Strict IPv4 Prefix, 8 bytes, /32)
132.2.9.2 (Strict IPv4 Prefix, 8 bytes, /32)
132.2.5.5 (Strict IPv4 Prefix, 8 bytes, /32)
5.5.5.5 (Strict IPv4 Prefix, 8 bytes, /32)
Traffic params - Rate: 0 bits/sec, Max. burst: 1K bytes
Min Policed Unit: 0 bytes, Max Pkt Size 2147483647 bytes
Fast-Reroute Backup info:
  Inbound FRR: Not active
  Outbound FRR: No backup tunnel selected
Path ID handle: 2F00040A.
Incoming policy: Accepted. Policy source(s): MPLS/TE
Status: Proxied
Output on GigabitEthernet2.519. Policy status: Forwarding. Handle: 0A00040E
Policy source(s): MPLS/TE
```

Likewise, we can do the same for the RESV messages. This time, there is a NHOP and outgoing interface, but there won't be at the tail end. We can also see the label value of 9013 which we saw in the RSVP dump-messages and the MPLS-TE show commands.

```
CSR1#show ip rsvp reservation filter session-type 7 tunnel-id 1
Destination      Tun Sender      TunID LSPID Next Hop      I/F      Fi Serv BPS
5.5.5.5          1.1.1.1         1     5     132.1.9.9     Gi2.519  SE LOAD 0
```

```
CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 1
Reservation:
```

```
Tun Dest: 5.5.5.5 Tun ID: 1 Ext Tun ID: 1.1.1.1
Tun Sender: 1.1.1.1 LSP ID: 5
Next Hop: 132.1.9.9 on GigabitEthernet2.519
Label: 9013 (outgoing)
Reservation Style is Shared-Explicit, QoS Service is Controlled-Load
Resv ID handle: 0400040F.
Average Bitrate is 0 bits/sec, Maximum Burst is 1K bytes
Min Policed Unit: 0 bytes, Max Pkt Size: 1500 bytes
Status:
Policy: Accepted. Policy source(s): MPLS/TE
```

The same commands are valid on any router in the LSP. Moving to CSR9, we will issue the same commands. Since CSR9 has no concept of "tunnel 1", as an interface, we can specify that we want to see all tunnels for which CSR9 is a "midpoint". Other role options include "head" and "tail". We could specify the "tunnel ID" of 1, and Cisco's implementation populates this ID based on the tunnel interface number. We can see the local label sent to CSR1 is 9013 and the downstream (outbound) label is 2008 received from CSR2. A quick look at the LFIB clearly shows a swap operation. The [5] represents the LSP ID, which is the "Tun_Instance" in the detailed output. The "1" just before this LSP IP in the LFIB represents the tunnel ID.

```
CSR9#show mpls traffic-eng tunnels role middle
```

P2P TUNNELS/LSPs:

LSP Tunnel BASIC is signalled, connection is up

InLabel : GigabitEthernet2.519, 9013

Prev Hop : 132.1.9.1

OutLabel : GigabitEthernet2.529, 2008

Next Hop : 132.2.9.2

RSVP Signalling Info:

Src 1.1.1.1, Dst 5.5.5.5, Tun_Id 1, Tun_Instance 5

RSVP Path Info:

My Address: 132.2.9.9

Explicit Route: 132.2.9.2 132.2.5.5 5.5.5.5

Record Route: NONE

Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits

RSVP Resv Info:

Record Route: NONE

Fspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits

CSR9#show mpls forwarding-table labels 9013

Local	Outgoing	Prefix	Bytes Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched	interface	
9013	2008	1.1.1.1 1 [5]	0	Gi2.529	132.2.9.2

As a midpoint, CSR9 has both PHOPs and NHOPs in the RSVP signaling chain. Traffic arrives from CSR1 and is switched towards CSR2. Notice that the destination and sender addresses are the same in both outputs, since the LSP is unidirectional. The output will be similar on all midpoints in the LSP, so we will skip CSR10. CSR1 is the PHOP (upstream) and CSR2 is the NHOP (downstream).

CSR9#show ip rsvp sender filter session-type 7 tunnel-id 1

Destination	Tun Sender	TunID	LSPID	Prev Hop	I/F	BPS
5.5.5.5	1.1.1.1	1	5	132.1.9.1	Gi2.519	0

CSR9#show ip rsvp reservation filter session-type 7 tunnel-id 1

Destination	Tun Sender	TunID	LSPID	Next Hop	I/F	Fi	Serv	BPS
5.5.5.5	1.1.1.1	1	5	132.2.9.2	Gi2.529	SE	LOAD	0

On the tail end, the outputs are different. We can reference the tunnel by ID as well, or use "role tail", etc. Notice that the local label is implicit-null but there is no out label; this is the opposite of what we saw at the headend. This makes sense because the TE LSP ends at CSR5. Additionally, the RESV message has no NHOP or outgoing interface since this is the terminating point of the LSP, as mentioned earlier.

CSR5#show mpls traffic-eng tunnels source-id 1

P2P TUNNELS/LSPs:

LSP Tunnel BASIC is signalled, connection is up

InLabel : GigabitEthernet2.525, implicit-null

Prev Hop : 132.2.5.2

OutLabel : -

RSVP Signalling Info:

```

    Src 1.1.1.1, Dst 5.5.5.5, Tun_Id 1, Tun_Instance 5
RSVP Path Info:
  My Address: 5.5.5.5
  Explicit Route: NONE
  Record Route: NONE
  Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits
RSVP Resv Info:
  Record Route: NONE
  Fspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits

```

```

CSR5#show ip rsvp sender filter session-type 7 tunnel-id 1
Destination      Tun Sender      TunID LSPID Prev Hop      I/F      BPS
5.5.5.5          1.1.1.1         1     5     132.2.5.2      Gi2.525  0

```

```

CSR5#show ip rsvp reservation filter session-type 7 tunnel-id 1
Destination      Tun Sender      TunID LSPID Next Hop      I/F      Fi Serv BPS
5.5.5.5          1.1.1.1         1     5     none           none      SE LOAD 0

```

To detail an example of a valid PCALC but failed RSVP signalling, I will configure an ACL on all of CSR5's interfaces to block IP protocol 46, which is RSVP. If RSVP cannot receive PATH messages, the LSP signalling cannot complete.

```

! CSR5
ip access-list extended DENY_RSVP
  deny  46 any any
  permit ip any any

interface GigabitEthernet2.525
  ip access-group DENY_RSVP in

interface GigabitEthernet2.550
  ip access-group DENY_RSVP in

interface GigabitEthernet2.551
  ip access-group DENY_RSVP in

```

Because RSVP PATH messages sent to CSR5 are discarded, there is no RSVP PATHERR signaling since CSR1 keeps originating PATH messages. PCALC sees no issue with the network at all, though. In this case, it would not make sense to troubleshoot the TED or anything related to IGP as the issue is likely data-plane/RSVP related; we know this because the “path is valid. There are several other issues that can cause TE failures but most are related to specific constraints and will be examined later.

```

CSR1#show mpls traffic-eng tunnels tunnel 1 | section Status
Status:
  Admin: up      Oper: down  Path: valid  Signalling: RSVP
signalling proceeding
  path option 10, type dynamic (Basis for Setup, path weight 30)

```

```

CSR1#show ip rsvp sender
To          From          Pro DPort Sport Prev Hop          I/F          BPS
5.5.5.5      1.1.1.1      0   1       10    none         none         0

CSR1#show ip rsvp reservation
To          From          Pro DPort Sport Next Hop    I/F          Fi Serv BPS
[no output]

```

31.1.2 TE attributes

This section continues from the last with a functional topology (test ACLs removed). We will begin adding tunnels with various attributes to see how they are used. We can use MPLS OAM (detailed in a separate section) to verify the tunnels are functional in the data-plane also. The first and most straightforward tunnel attribute is the TE metric. This is a "second metric" that we assign to each link. The TE metric is commonly used for creating two topologies in a network; IGP metrics can be used for data flows while TE metrics can define a separate topology for voice flows, as an example. If it is not specified, it is copied from the IGP metric. Like any metric, it can be asymmetric, and is evaluated by the local node. A new tunnel [ID 2] will create an LSP to XRv11 but we want to force it through CSR6. Knowing that all other link costs are equal, we can lower the TE metric on CSR1 facing CSR6 to achieve this. We do not need to do it in the other direction, technically, since we don't expect return traffic to take the same path. This means the path will be lower cost than the other 2-hop paths through CSR3 or CSR9.

```

! CSR1
interface GigabitEthernet2.516
 mpls traffic-eng administrative-weight 5

```

We can verify this change is present in the TED. After all, all TE-aware nodes need to be aware of this change. Checking CSR6 also, we see that its local TE metric did not change, so we have introduced some asymmetry into the network.

```

CSR1#show mpls traffic-eng topology level-2 1.1.1.1 brief
IGP Id: 0000.0000.0001.00, MPLS TE Id:1.1.1.1 Router Node (isis level-2)
[snip]
link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0006.00, nbr_node_id:4,
gen:570
frag_id: 0, Intf Address: 132.1.6.1, Nbr Intf Address: 132.1.6.6
TE metric: 5, IGP metric: 10, attribute flags: 0x4
SRLGs: None
[snip]

CSR1#show mpls traffic-eng topology level-2 6.6.6.6 brief
IGP Id: 0000.0000.0006.00, MPLS TE Id:6.6.6.6 Router Node (isis level-2)
[snip]
link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0001.00, nbr_node_id:16,
gen:563

```

```

frag_id: 0, Intf Address: 132.1.6.6, Nbr Intf Address: 132.1.6.1
TE metric: 10, IGP metric: 10, attribute flags: 0x4
SRLGs: None
[snip]

```

Once the tunnel comes up, we can see the ERO specifies CSR6 as the next hop. The path weight is 25 which implies the TE metric of 5 was used (5+10+10 for total path). Recall that this ERO was the output of the PCALC operation and the input for the initial RSVP PATH message. The tunnel configuration specifies that the TE metric should be used explicitly, but this is the default. You can also specify IGP, but this is rare.

```

! CSR1
interface Tunnel2
  description TE METRIC
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 dynamic
  tunnel mpls traffic-eng path-selection metric te

CSR1#show mpls traffic-eng tunnels tunnel 2 | section Status|RSVP_Path
Status:
Admin: up      Oper: up      Path: valid      Signalling: connected
path option 10, type dynamic (Basis for Setup, path weight 25)

RSVP Path Info:
My Address: 132.1.6.1
Explicit Route: 132.1.6.6 132.6.12.12 132.11.12.11 11.11.11.11
Record Route:  NONE
Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits

```

A quick OAM check shows us the label stack and reveals the hops along the TE LSP.

```

CSR1#traceroute mpls traffic-eng tunnel 2
Tracing MPLS TE Label Switched Path on Tunnel2, timeout is 2 seconds
[snip]

Type escape sequence to abort.
 0 132.1.6.1 MRU 1500 [Labels: 6011 Exp: 0]
L 1 132.1.6.6 MRU 1500 [Labels: 92010 Exp: 0] 2 ms
L 2 132.6.12.12 MRU 1500 [Labels: implicit-null Exp: 0] 38 ms
! 3 132.11.12.11 31 ms

```

A similar example on XRv11 can be used to steer traffic to CSR8 away from XRv12. We increase the TE metric on the XRv11-XRv12 link so that it is less preferred. We make the configuration change then

check the TE topology. For variety, we will verify it from XRv12's perspective as all routers in the IS-IS level have this updated information.

```
! XRv11
mpls traffic-eng
 interface GigabitEthernet0/0/0/0.512
   admin-weight 300

RP/0/0/CPU0:XRv12#show mpls traffic-eng topology 11.11.11.11 brief | begin
0012
  Link[2]:Point-to-Point, Nbr IGP Id:0000.0000.0012.00, Nbr Node Id:1,
gen:69260
  Frag Id:0, Intf Address:132.11.12.11, Intf Id:0
  Nbr Intf Address:132.11.12.12, Nbr Intf Id:0
  TE Metric:300, IGP Metric:10
  Attribute Flags: 0xc
  Ext Admin Group:
    Length: 256 bits
    Value : 0x::c
  Attribute Names: BLUE(2) ORANGE(3)
  Switching Capability:None, Encoding:unassigned
  BC Model ID:RDM
  Physical BW:1000000 (kbps), Max Reservable BW Global:100000 (kbps)
  Max Reservable BW Sub:0 (kbps)
```

Next, we configure the tunnel. XR has a clean way of ignoring affinities with a self-explanatory command. XR also provides some verbose logging, which includes showing the ERO in the log message. We can also verify it with the traditional show commands; the output is almost identical to IOS.

```
! XRv11
interface tunnel-te2
 description TE METRIC
 ipv4 unnumbered Loopback0
 logging events all
 destination 8.8.8.8
 path-selection metric te
 affinity ignore
 path-option 10 dynamic

te_control[1044]: %ROUTING-MPLS_TE-5-LSP_EXPLICITROUTE : tunnel-te2
(signalled-name: XRv11_t2, LSP Id: 2) explicit-route, 132.10.11.10,
132.6.10.6, 132.6.8.8, 8.8
te_control[1044]: %ROUTING-MPLS_TE-5-LSP_RECORDROUTE : tunnel-te2 (signalled-
name: XRv11_t2, LSP Id: 2) record-route empty.
te_control[1044]: %ROUTING-MPLS_TE-5-LSP_UPDOWN : tunnel-te2 (signalled-name:
XRv11_t2, LSP Id: 2) state changed to up
```

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 2
```

```

Name: tunnel-te2 Destination: 8.8.8.8 Ifhandle:0x480
Signalled-Name: XRv11_t2
Status:
  Admin:      up Oper:      up Path:  valid Signalling: connected

  path option 10, type dynamic (Basis for Setup, path weight 30)
  G-PID: 0x0800 (derived from egress interface properties)
  Bandwidth Requested: 0 kbps CT0
  Creation Time: DAY MON 9 12:31:56 2015 (00:03:45 ago)
Config Parameters:
  Bandwidth:      0 kbps (CT0) Priority:  7  7
  Number of affinity constraints: 1
  Ignore all

  Metric Type: TE (interface)
  Hop-limit: disabled
  Cost-limit: disabled
  AutoRoute: disabled LockDown: disabled Policy class: not set
  Forward class: 0 (default)
  Forwarding-Adjacency: disabled
  Loadshare:      0 equal loadshares
  Auto-bw: disabled
  Fast Reroute: Disabled, Protection Desired: None
  Path Protection: Not Enabled
  BFD Fast Detection: Disabled
  Reoptimization after affinity failure: Enabled
  Soft Preemption: Disabled
History:
  Tunnel has been up for: 00:01:20 (since DAY MON 09 12:34:21 UTC 2015)
  Current LSP:
    Uptime: 00:01:20 (since DAY MON 09 12:34:21 UTC 2015)

  Path info (IS-IS 132 level-2):
  Node hop count: 3
  Hop0: 132.10.11.10
  Hop1: 132.6.10.6
  Hop2: 132.6.8.8
  Hop3: 8.8.8.8

```

A quick OAM check verifies the TE LSP is operational and shows us the TE labels per hop. We can also verify the RSVP PATH/RESV messages using similar commands to XE.

```

RP/0/0/CPU0:XRv11#traceroute mpls traffic-eng tunnel-te 2
Tracing MPLS TE Label Switched Path on tunnel-te2, timeout is 2 seconds
[snip]

```

Type escape sequence to abort.

```

0 132.10.11.11 MRU 1500 [Labels: 10010 Exp: 0]

```



```
L 1 132.10.11.10 MRU 1500 [Labels: 6011 Exp: 0] 80 ms
L 2 132.6.10.6 MRU 1500 [Labels: implicit-null Exp: 0] 50 ms
! 3 132.6.8.8 20 ms
```

```
RP/0/0/CPU0:XRv11#show rsvp sender session-type lsp-p2p
```

Destination	Add	DPort	Source	Add	SPort	Pro	Input	IF	Rate	Burst	Prot
8.8.8.8		2	11.11.11.11		2	0		No	0	1K	Off

```
RP/0/0/CPU0:XRv11#show rsvp reservation session-type lsp-p2p
```

Destination	Add	DPort	Source	Add	SPort	Pro	Input	IF	Sty	Serv	Rate	Burst
8.8.8.8		2	11.11.11.11		2	0	Gi0/0/0/0.501		SE	LOAD	0	1K

Another TE attribute is affinity. This is often called link coloring. It is a 4 byte hexadecimal number where each of the 32 bit positions represents a "color". This approach allows for 32 different colors (some routers, like XR, support 256 using extended link attributes) and any combinations of colors is valid. In this topology, 4 colors are used with the following binary and hexadecimal strings. This information is also depicted on the main diagram for quick reference.

Red: 0001 or 0x1

Green: 0010 or 0x2

Blue: 0100 or 0x4

Orange: 1000 or 0x8

Individual links are colored (can be asymmetric like TE metric) administratively and this can be used as a constraint for a given TE LSP. The red path might be high bandwidth, the green path might low latency, and the blue path might be scavenger traffic, as an example. Some links may be both low latency and high bandwidth, and would be colored both red and green. A tunnel uses the affinity mask to select which colors it cares about in terms of constraints. The default affinity and affinity mask on XE and XR is 0x00000000 and 0x0000FFFF, respectively. MPLS-TE made things confusing by reversing the bitwise logic used in ACLs for TE affinity. With ACLs, a value of 0 in the mask means "I care about it" while 1 means "I ignore it". The logic is reversed with affinity masks, so a value of 1 means it is honored. The default values, therefore, mean that the lower-order 16 bits MUST be zero by default. In our earlier examples, we specifically had to set the mask to 0x0 which says "I don't care about any affinity bits". If we fail to clear these affinity bits in a "colored" network, no tunnels will come up. Looking at the PCALC debugs along with TE logging, this becomes clear. RSVP doesn't even get an ERO and no signaling takes place since PCALC cannot find any suitable path that meets the tunnel constraints. The affinity mask is shown as 0xFFFF which is also visible via the TE show commands [ID 3].

```
! CSR1
interface Tunnel3
 ip unnumbered Loopback0
 tunnel mode mpls traffic-eng
```

```

tunnel destination 11.11.11.11
tunnel mpls traffic-eng path-option 10 dynamic

CSR1#debug mpls traffic-eng path lookup
TE-PCALC-API: 1.1.1.1_3->11.11.11.11_3 {7}: P2P LSP Path Lookup called
TE-PCALC: 1.1.1.1_3->11.11.11.11_3 {7}: Path Request Info
  Flags: METRIC_TE
  IP explicit-path: None (dynamic)
  bw 0, min_bw 0, metric: 0
  setup_pri 7, hold_pri 7
  affinity_bits 0x0, affinity_mask 0xFFFF
TE-PCALC-PATH: 1.1.1.1_3->11.11.11.11_3 {7}: Area (isis level-2) Path Lookup
begin
TE-PCALC-PATH: 1.1.1.1_3->11.11.11.11_3 {7}: Get path: Failed to find a path
to destination
TE-PCALC-PATH: 1.1.1.1_3->11.11.11.11_3 {7}: Area (isis level-2) Path Lookup
end: path not found
TE-PCALC-API: 1.1.1.1_3->11.11.11.11_3 {7}: P2P LSP Path Lookup result:
failed
%MPLS TE-5-LSP: LSP 1.1.1.1 3 3: No path to destination, 0000.0000.0011.00
(affinity)

```

The output below shows that the path is "not valid" since PCALC cannot resolve. The affinity mask is shown again and this is the root of the issue. The output also shows us the shortest unconstrained path, which ignores the constraints and computes a cost-based path only for informational purposes.

```

CSR1#show mpls traffic-eng tunnels tunnel 3
Name: CSR1_t3                               (Tunnel3) Destination: 11.11.11.11
Status:
  Admin: up      Oper: down  Path: not valid  Signalling: Down
  path option 10, type dynamic

Config Parameters:
  Bandwidth: 0      kbps (Global)  Priority: 7 7  Affinity: 0x0/0xFFFF
  Metric Type: TE (default)
  AutoRoute: disabled LockDown: disabled Loadshare: 0 [0] bw-based
  auto-bw: disabled

Shortest Unconstrained Path Info:
  Path Weight: 25 (TE)
  Explicit Route: 132.1.6.6 132.6.12.12 132.11.12.11 11.11.11.11
History:
  Tunnel:
    Time since created: 2 minutes, 28 seconds
    Number of LSP IDs (Tun_Instances) used: 8

```

We will specifically configure this tunnel to traverse green links. This assumes we have an end-to-end green path, and we have a few. To accomplish this, we can specify the green affinity bit (0x2) and modify

the mask to only account for that bit (0x2). We see that the tunnel comes up and has selected the path CSR3 > CSR2 > CSR10 > XRv11, which contains only green links. Notice that the links traversed by this tunnel are all green, but not only green; multi-colored links can also be used. The affinity mask simply chose not to test for the presence or absence of the other colors, so PCALC does not care. The cost of this path was 40, but we know the lower cost (unconstrained) path was 25. This is what makes TE so powerful; routing decisions can become arbitrary yet dynamic.

```
! CSR1
interface Tunnel3
  tunnel mpls traffic-eng affinity 0x2 mask 0x2

CSR1#show mpls traffic-eng tunnels tunnel 3 | section Status|RSVP_Path
Status:
  Admin: up      Oper: up      Path: valid      Signalling: connected
  path option 10, type dynamic (Basis for Setup, path weight 40)

RSVP Path Info:
  My Address: 132.1.3.1
  Explicit Route: 132.1.3.3 132.2.3.3 132.2.3.2 132.2.10.10
                  132.10.11.11 11.11.11.11
  Record Route:  NONE
  Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits
```

We can do the opposite as well by instructing the tunnel to take any path that is NOT green [ID 4]. We still need to evaluate the same bit, so the mask does not change as we are only evaluating against the green color. The difference is that we clear the bit in the affinity so as to say "This bit must be zero, and I care about it". CSR1 selects a path that avoids all green paths, traversing CSR6 > XRv12 > XRv11.

```
! CSR1
interface Tunnel4
  description NOT GREEN
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng affinity 0x0 mask 0x2
  tunnel mpls traffic-eng path-option 10 dynamic

CSR1#show mpls traffic-eng tunnels tunnel 4 | section Status|RSVP_Path
Status:
  Admin: up      Oper: up      Path: valid      Signalling: connected
  path option 10, type dynamic (Basis for Setup, path weight 25)

RSVP Path Info:
  My Address: 132.1.6.1
  Explicit Route: 132.1.6.6 132.6.12.12 132.11.12.11 11.11.11.11
  Record Route:  NONE
  Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits
```

The opposite logic is invalid; you cannot set something in a tunnel's affinity then say you don't care about it in the mask. The configuration doesn't make sense and is rejected by the parser. If you don't care about it, set the affinity bit and mask bit to zero.

```
CSR1(config-if)#tunnel mpls traffic-eng affinity 0x2 mask 0x0
% Bits cannot be set in affinity (0x2) if unset in mask (0x0)
```

The logical AND operation is valid when combining colors. From CSR3 to CSR10, we can create a path which requires all transit links be green, blue, and orange at the same time [ID 5]. There isn't a one-line mechanism to do Boolean OR logic, though, as you would normally use separate path-options for this, which is evaluated in more complex examples later. The more aggressive the constraints, the less likely the tunnel is to form. In this case, there is only one path with all three colors and it traverses CSR2. The sum of 0x2, 0x4, and 0x8 is 0xE (14 in decimal) or 1110 in binary. We specify this as the affinity and the bits we care about (mask).

```
! CSR3
interface Tunnel5
description GREEN BLUE ORANGE TOGETHER
ip unnumbered Loopback0
tunnel mode mpls traffic-eng
tunnel destination 10.10.10.10
tunnel mpls traffic-eng affinity 0xE mask 0xE
tunnel mpls traffic-eng path-option 10 dynamic
```

```
CSR3#show mpls traffic-eng tunnels tunnel 5 | section Status|RSVP_Path
Status:
Admin: up      Oper: up      Path: valid      Signalling: connected
path option 10, type dynamic (Basis for Setup, path weight 20)

RSVP Path Info:
My Address: 132.2.3.3
Explicit Route: 132.2.3.2 132.2.10.10 10.10.10.10
Record Route:  NONE
Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits
```

A small adjustment of the affinity mask changes the logic significantly and breaks the tunnel. We can specify the tunnel must traverse links that are blue, green, and orange, but also NOT red. This is accomplished by setting the mask to 0xF or 1111 in binary, which forces PCALC to examine the last 4 bits of the link colors. There is no path from CSR3 to CSR10 that meets the constraints since the CSR2-CSR10 is colored red. We told the affinity mask to care about all four of our colors, but did not instruct the tunnel to use red links.

```
! CSR3
interface Tunnel6
```

```

description GREEN BLUE ORANGE TOGETHER, BUT NOT RED
ip unnumbered Loopback0
tunnel mode mpls traffic-eng
tunnel destination 10.10.10.10
tunnel mpls traffic-eng affinity 0xE mask 0xF
tunnel mpls traffic-eng path-option 10 dynamic

```

```

CSR3#show mpls traffic-eng tunnels tunnel 6 | section Status|RSVP_Path
Status:
  Admin: up      Oper: down  Path: not valid  Signalling: Down
  path option 10, type dynamic

```

```

%MPLS_TE-5-LSP: LSP 3.3.3.3 6_1: No path to destination, 0000.0000.0010.00
(no path)

```

We will repeat a similar set of tests with XR. The affinity configuration in XR is much simpler and more flexible than XE. You can assign strings of text to affinity values and enumerate these strings on an interface; XR can add them together to determine the link attributes, which is a simple, human-readable operation. XRv12 uses this mechanism, plus some legacy syntax. Attribute values (generally a single bit or color) is mapped to a string, and then those strings are assigned to interfaces. The manner in which bits are assigned to strings can vary between the classic hex syntax of specifying bit positions, where 0 is the least significant bit (0x1). You can also use the legacy syntax by specifying the hexadecimal value directly at the link level.

```

! XRv12
mpls traffic-eng
interface GigabitEthernet0/0/0/0.502
  attribute-names RED ORANGE

interface GigabitEthernet0/0/0/0.512
  attribute-names BLUE ORANGE

interface GigabitEthernet0/0/0/0.562
  attribute-flags 0xc

interface GigabitEthernet0/0/0/0.592
  attribute-names RED

affinity-map RED bit-position 0
affinity-map BLUE 0x4
affinity-map GREEN bit-position 1
affinity-map ORANGE 0x8

```

From XRv12, we will create a tunnel to CSR1 that requires an orange path [ID 3]. We will verify this tunnel by checking the outgoing ERO, which is used as input to the RSVP process within PATH messages. We can validate the path using OAM. The "include-strict" option selects a set of colors that must exist on a link, but no others may. If this command were used in this tunnel, it would mean only use orange links

that contain no other colors. Colorless links would also be rejected using strict inclusion. This would be identical to an IOS affinity mask of 0xFFFFFFFF since all bits are being evaluated.

```
! XRv12
interface tunnel-te3
  description ORANGE
  ipv4 unnumbered Loopback0
  logging events all
  destination 1.1.1.1
  affinity include ORANGE
  path-option 10 dynamic
```

```
RP/0/0/CPU0:XRv12#show rsvp sender session-type lsp-p2p destination 1.1.1.1
destination 1.1.1.1 detail | begin Outgoing
```

```
Explicit Route (Outgoing):
```

```
  Strict, 132.6.12.6/32
  Strict, 132.6.9.9/32
  Strict, 132.4.9.4/32
  Strict, 132.1.4.1/32
  Strict, 1.1.1.1/32
```

```
RP/0/0/CPU0:XRv12#traceroute mpls traffic-eng tunnel-te 3
Tracing MPLS TE Label Switched Path on tunnel-te3, timeout is 2 seconds
[snip]
Type escape sequence to abort.
```

```
  0 132.6.12.12 MRU 1500 [Labels: 6014 Exp: 0]
L 1 132.6.12.6 MRU 1500 [Labels: 9013 Exp: 0] 10 ms
L 2 132.6.9.9 MRU 1500 [Labels: 4010 Exp: 0] 10 ms
L 3 132.4.9.4 MRU 1500 [Labels: implicit-null Exp: 0] 10 ms
! 4 132.1.4.1 10 ms
```

We can also invert the logic by select a path that is NOT orange also [ID 4]. We can reference the tunnel by name also, provided we include quotes if the name has spaces. The outgoing ERO is shown at the bottom. MPLS traceroute verifies connectivity and the TE labels. The number in parenthesis is the bit-position from right to left regarding this color. The third bit position is 1000 in binary, or 8. The syntax in XR makes this very straightforward.

```
! XRv12
interface tunnel-te4
  description NOT ORANGE
  ipv4 unnumbered Loopback0
  logging events all
  destination 1.1.1.1
  affinity exclude ORANGE
  path-option 10 dynamic
```

```
RP/0/0/CPU0:XRv12#show mpls traffic-eng tunnels name "NOT ORANGE"
```

```

Name: tunnel-te4 Destination: 1.1.1.1 Ifhandle:0x680
Signalled-Name: XRv12_t4
Status:
  Admin:      up Oper:      up Path:  valid Signalling: connected

  path option 10, type dynamic (Basis for Setup, path weight 20)
  G-PID: 0x0800 (derived from egress interface properties)
  Bandwidth Requested: 0 kbps CT0
  Creation Time: MON DAY 9 13:04:02 2015 (00:04:51 ago)
Config Parameters:
  Bandwidth:      0 kbps (CT0) Priority: 7 7
  Number of affinity constraints: 1
    Exclude bit map      : 0x8
    Exclude ext bit map  :
      Length: 256 bits
      Value : 0x::8
    Exclude affinity name : ORANGE(3)

[snip]
Node hop count: 2
Hop0: 132.9.12.9
Hop1: 132.1.9.1
Hop2: 1.1.1.1

RP/0/0/CPU0:XRv12#traceroute mpls traffic-eng tunnel-te 4
Tracing MPLS TE Label Switched Path on tunnel-te4, timeout is 2 seconds
[snip]
Type escape sequence to abort.

 0 132.9.12.12 MRU 1500 [Labels: 9012 Exp: 0]
L 1 132.9.12.9 MRU 1500 [Labels: implicit-null Exp: 0] 0 ms
! 2 132.1.9.1 10 ms

```

We can combine affinities by enumerating multiple strings. From XRv12, we build a path to CSR5 that is both red and orange. The output shows the bitmap 0x9 which is red (0x1) plus orange (0x8). This simplifies the logic for the operation and improves readability over the IOS configuration. we can see the path traverses CSR10 > CSR2 > CSR5, which is the only available path meeting the affinity constraints.

```

! XRv12
interface tunnel-te5
 description RED AND ORANGE TOGETHER
 ipv4 unnumbered Loopback0
 logging events all
 destination 5.5.5.5
 affinity include RED ORANGE
 path-option 10 dynamic

RP/0/0/CPU0:XRv12#show mpls traffic-eng tunnels 5

```

```

Name: tunnel-te5 Destination: 5.5.5.5 Ifhandle:0x780
Signalled-Name: XRv12_t5
Status:
  Admin:      up Oper:      up Path:  valid Signalling: connected

  path option 10, type dynamic (Basis for Setup, path weight 30)
  G-PID: 0x0800 (derived from egress interface properties)
  Bandwidth Requested: 0 kbps CT0
  Creation Time: MON DAY 9 13:25:08 2015 (00:00:05 ago)
Config Parameters:
  Bandwidth:      0 kbps (CT0) Priority: 7 7
  Number of affinity constraints: 1
    Include bit map      : 0x9
    Include ext bit map  :
      Length: 256 bits
      Value : 0x::9
    Include affinity name : RED(0) ORANGE(3)

[snip]
Node hop count: 3
Hop0: 132.10.12.10
Hop1: 132.2.10.2
Hop2: 132.2.5.5
Hop3: 5.5.5.5

```

Likewise, we can specify affinities we specifically want to exclude in conjunction with those we want to include. We instruct XRv12 to build a path to CSR5 that is red and orange, but not blue [ID 6]. The configuration is very similar to the previous tunnel except with the added blue exclusion. No such path exists, and PCALC is not able to find a path that meets the affinity constraints. The tunnel information shows the affinity names, along with their values, both for inclusion and exclusion.

```

! XRv12
interface tunnel-te6
  description RED AND ORANGE TOGETHER BUT NOT BLUE
  ipv4 unnumbered Loopback0
  logging events all
  destination 5.5.5.5
  affinity include RED ORANGE
  affinity exclude BLUE
  path-option 10 dynamic

%ROUTING-MPLS_TE-5-LSP_PCALC_FAIL : Path calculation failed on tunnel-te6
(signalled-name: XRv12_t6, LSP: 0, path option: 10, explicit path: NONE): No
path to destination, 5.5.5.5 (affinity)

RP/0/0/CPU0:XRv12#show mpls traffic-eng tunnels 6
Name: tunnel-te6 Destination: 5.5.5.5 Ifhandle:0x880
  Signalled-Name: XRv12_t6

```


Status:

Admin: up Oper: down Path: not valid Signalling: Down

path option 10, type dynamic

Last PCALC Error: DAY MON 9 13:37:14 2015

Info: No path to destination, 5.5.5.5 (affinity)

G-PID: 0x0800 (derived from egress interface properties)

Bandwidth Requested: 0 kbps CT0

Creation Time: MON DAY 9 13:37:14 2015 (00:00:17 ago)

Config Parameters:

Bandwidth: 0 kbps (CT0) Priority: 7 7

Number of affinity constraints: 2

Include bit map : 0x9

Include ext bit map :

Length: 256 bits

Value : 0x::9

Include affinity name : RED(0) ORANGE(3)

Exclude bit map : 0x4

Exclude ext bit map :

Length: 256 bits

Value : 0x::4

Exclude affinity name : BLUE(2)

TE tunnels can also request bandwidth. This is a control-plane reservation carried in the RSVP signaling and accounted for by both RSVP and the TED. We configure a TE tunnel from CSR1 to XRv11 requesting 20 Mbps of bandwidth [ID 7]. This is a constraint that gets fed into PCALC, and then to RSVP for signaling. Looking at the RSVP dumps, we can see the PROXY PATH (output from PCALC) contains a request for 2.5 MBps (big B for bytes), which is 20 Mbps. All of the other RSVP PATH attributes are present also, such as the SAO, ERO, and sender Tspec.

! CSR1

interface Tunnel7

description BASIC BW REQ

ip unnumbered Loopback0

tunnel mode mpls traffic-eng

tunnel destination 11.11.11.11

tunnel mpls traffic-eng priority 7 7

tunnel mpls traffic-eng bandwidth 20000

tunnel mpls traffic-eng affinity 0x0 mask 0x0

tunnel mpls traffic-eng path-option 10 dynamic

! CSR1

Incoming PROXY_PATH:

version:1 flags:0000 cksum:0000 ttl:255 reserved:0 length:212

SESSION type 7 length 16:

Tun Dest: 11.11.11.11 Tun ID: 7 Ext Tun ID: 1.1.1.1

HOP type 1 length 12:

Hop Addr: 127.0.0.1 LIH: 0x00000000

```

TIME_VALUES          type 1 length 8 :
  Refresh Period (msec): 30000
EXPLICIT_ROUTE       type 1 length 44:
  1.1.1.1 (Strict IPv4 Prefix, 8 bytes, /32)
  132.1.6.6 (Strict IPv4 Prefix, 8 bytes, /32)
  132.6.10.10 (Strict IPv4 Prefix, 8 bytes, /32)
  132.10.11.11 (Strict IPv4 Prefix, 8 bytes, /32)
  11.11.11.11 (Strict IPv4 Prefix, 8 bytes, /32)
SESSION_ATTRIBUTE    type 7 length 20:
  Setup Prio: 7, Holding Prio: 7
  Flags: (0x4) SE Style
  Session Name: BASIC BW REQ
SENDER_TEMPLATE      type 7 length 12:
  Tun Sender: 1.1.1.1 LSP ID: 1
SENDER_TSPEC         type 2 length 36:
  version=0, length in words=7
  Token bucket fragment (service_id=1, length=6 words
    parameter id=127, flags=0, parameter length=5
    average rate=2500000 bytes/sec, burst depth=1000 bytes
    peak rate    =2500000 bytes/sec
    min unit=0 bytes, max pkt size=2147483647 bytes
[snip]

```

When signaling is complete, the head end receives the final RSVP RESV message containing the TE label for the next-hop and the flowspec record showing the reservation was successful. Note: Shared Explicit (SE) is used to ensure make-before-break (MBB). This is contained in the PATH message as a flag shown above. When an LSR needs to reroute an LSP, the new LSP is built before the old one is torn down. SE prevents double-booking of bandwidth on links during this process.

```

! CSR1
Incoming Resv:
  version:1 flags:0000 cksum:FB86 ttl:255 reserved:0 length:108
SESSION          type 7 length 16:
  Tun Dest: 11.11.11.11 Tun ID: 7 Ext Tun ID: 1.1.1.1
HOP              type 1 length 12:
  Hop Addr: 132.1.6.6 LIH: 0x0000000E
TIME_VALUES      type 1 length 8 :
  Refresh Period (msec): 30000
STYLE           type 1 length 8 :
  Shared-Explicit (SE)
FLOWSPEC         type 2 length 36:
  version = 0 length in words = 7
  service id = 5, service length = 6
  tspec parameter id = 127, flags = 0, length = 5
  average rate = 2500000 bytes/sec, burst depth = 1000 bytes
  peak rate    = 2500000 bytes/sec
  min unit = 0 bytes, max pkt size = 1500 bytes
FILTER_SPEC     type 7 length 12:

```

```
Tun Sender: 1.1.1.1, LSP ID: 1
LABEL          type 1 length 8 :
Labels: 6013
```

We can verify this bandwidth reservation several ways. The reservations are made outbound; because the LSP is unidirectional, CSR1 needs to send 20 Mbps of data, so the reservation is applied to that outgoing interface. The PATH message carries the initial bandwidth request and the RESV confirms it.

```
CSR1#show ip rsvp sender filter session-type 7 tunnel-id 7
Destination      Tun Sender      TunID LSPID Prev Hop      I/F      BPS
11.11.11.11      1.1.1.1          7     1     none           none     20M

CSR1#show ip rsvp reservation filter session-type 7 tunnel-id 7
To               From           Pro DPort Sport Next Hop      I/F      Fi Serv BPS
11.11.11.11      1.1.1.1        0    7      1     132.1.6.6    Gi2.516  SE LOAD 20M
```

We can check bandwidth reservations per interface as tracked by RSVP as well. This is a good summary view of the bandwidth reservations local to a router.

```
CSR1#show ip rsvp interface
interface  rsvp      allocated  i/f max  flow max  sub max  VRF
Gi2        ena       0          750M     750M      0
Gi2.513    ena       0          100M     100M      0
Gi2.514    ena       0          100M     100M      0
Gi2.516    ena      20M        100M     100M      0
Gi2.518    ena       0          100M     100M      0
Gi2.519    ena       0          100M     100M      0
```

Checking the MPLS tunnel characteristics, we can see the Tspec and Fspec rates equal the reservation that was requested.

```
CSR1#show mpls traffic-eng tunnels tunnel 7 | section Signalling
Admin: up      Oper: up      Path: valid      Signalling: connected
RSVP Signalling Info:
  Src 1.1.1.1, Dst 11.11.11.11, Tun_Id 7, Tun_Instance 1
RSVP Path Info:
  My Address: 132.1.6.1
  Explicit Route: 132.1.6.6 132.6.10.10 132.10.11.11 11.11.11.11
  Record Route: NONE
  Tspec: ave rate=20000 kbits, burst=1000 bytes, peak rate=20000 kbits
RSVP Resv Info:
  Record Route: NONE
  Fspec: ave rate=20000 kbits, burst=1000 bytes, peak rate=20000 kbits
```

The ERO indicates that CSR6 is next in the TE tunnel path. Checking its tunnel characteristics, we see similar output to indicate the bandwidth reservation.

```

CSR6#show mpls traffic-eng tunnels source-id 7 | section Signalling
RSVP Signalling Info:
  Src 1.1.1.1, Dst 11.11.11.11, Tun_Id 7, Tun_Instance 1
RSVP Path Info:
  My Address: 132.6.10.6
  Explicit Route: 132.6.10.10 132.10.11.11 11.11.11.11
  Record Route: NONE
  Tspec: ave rate=20000 kbits, burst=1000 bytes, peak rate=20000 kbits
RSVP Resv Info:
  Record Route: NONE
  Fspec: ave rate=20000 kbits, burst=1000 bytes, peak rate=20000 kbits

```

For additional verification, we can check the TED to ensure the bandwidth reservation is accounted for. All nodes must be aware of this allocated bandwidth so that when new tunnels are created, the local PCALC process has a complete and updated view of the topology. CSR6 shows that 20 Mbps are allocated on its interface towards CSR10. This is also present in the IS-IS database with CSR6's LSP (verbose information). The higher number priorities are less important, and the default priority is 7 (lower priority). This is examined next.

```

CSR1#show mpls traffic-eng topology igp-id isis 0000.0000.0006.00 | begin
0010\
  link[3]: Point-to-Point, Nbr IGP Id: 0000.0000.0010.00, nbr_node_id:7,
gen:2748
  frag_id: 0, Intf Address: 132.6.10.6, Nbr Intf Address: 132.6.10.10
  TE metric: 10, IGP metric: 10, attribute flags: 0x6
  SRLGs: None
  physical_bw: 1000000 (kbps), max_reservable_bw_global: 100000 (kbps)
  max_reservable_bw_sub: 0 (kbps)

```

	Total Allocated BW (kbps)	Global Pool Reservable BW (kbps)	Sub Pool Reservable BW (kbps)
	-----	-----	-----
bw[0]:	0	100000	0
bw[1]:	0	100000	0
bw[2]:	0	100000	0
bw[3]:	0	100000	0
bw[4]:	0	100000	0
bw[5]:	0	100000	0
bw[6]:	0	100000	0
bw[7]:	20000	80000	0

```

CSR1#show isis database level-2 CSR6.00-00 verbose | section CSR10
Metric: 10          IS (MT-IPv6) CSR10.00
Metric: 10          IS-Extended CSR10.00
Affinity: 0x00000006
Interface IP Address: 132.6.10.6

```

```

Neighbor IP Address: 132.6.10.10
Physical BW: 1000000 kbits/sec
Reservable Global Pool BW: 100000 kbits/sec
Global Pool BW Unreserved:
  [0]: 100000 kbits/sec, [1]: 100000 kbits/sec
  [2]: 100000 kbits/sec, [3]: 100000 kbits/sec
  [4]: 100000 kbits/sec, [5]: 100000 kbits/sec
  [6]: 100000 kbits/sec, [7]: 80000 kbits/sec

```

Leaving the existing bandwidth reservation up, we create a 40 Mbps bandwidth reservation with a higher priority using a blue path, also from CSR1 to XRv11 [ID 8]. Based on the affinity and TE metric, we know that CSR6 will be the next-hop in the path. We verify the ERO by checking the outgoing RSVP PATH message.

```

! CSR1
interface Tunnel8
  description BLUE PRIORITY BW REQ
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng priority 4 4
  tunnel mpls traffic-eng bandwidth 40000
  tunnel mpls traffic-eng affinity 0x4 mask 0x4
  tunnel mpls traffic-eng path-option 10 dynamic

```

```

CSR1#show ip rsvp sender detail filter session-type 7 tunnel-id 8 | section
outgoing

```

```

ERO: (outgoing)
  132.1.6.6 (Strict IPv4 Prefix, 8 bytes, /32)
  132.6.12.12 (Strict IPv4 Prefix, 8 bytes, /32)
  132.11.12.11 (Strict IPv4 Prefix, 8 bytes, /32)
  11.11.11.11 (Strict IPv4 Prefix, 8 bytes, /32)

```

RSVP now reports a total of 60 Mbps reserved on the link between CSR1 and CSR6. The prioritization of flows happens within the PCALC process as RSVP is just used to signal them. When we check the TED on CSR1 relating to the CSR6-facing link, we can see that 20 Mbps is reserved at priority 7 and 40 Mbps is reserved at priority 4. The remaining reservable bandwidth is the remaining sum of bandwidths across all priorities. If a priority 4 flow requests 40 Mbps, then that 40 Mbps is not available for priority 4 or worse. Higher priorities can preempt (based on the hold priority, which is the second number) but lower ones cannot.

```

CSR1#show ip rsvp interface

```

interface	rsvp	allocated	i/f max	flow max	sub max	VRF
Gi2	ena	0	750M	750M	0	
Gi2.513	ena	0	100M	100M	0	
Gi2.514	ena	0	100M	100M	0	
Gi2.516	ena	60M	100M	100M	0	

```
Gi2.518      ena      0      100M      100M      0
Gi2.519      ena      0      100M      100M      0
```

```
CSR1#show mpls traffic-eng topology level-2 1.1.1.1 | begin 0006
  link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0006.00, nbr_node_id:4,
gen:2774
  frag_id: 0, Intf Address: 132.1.6.1, Nbr Intf Address: 132.1.6.6
  TE metric: 5, IGP metric: 10, attribute flags: 0x4
  SRLGs: None
  physical_bw: 1000000 (kbps), max_reservable_bw_global: 100000 (kbps)
  max_reservable_bw_sub: 0 (kbps)
```

	Total Allocated BW (kbps)	Global Pool Reservable BW (kbps)	Sub Pool Reservable BW (kbps)
	-----	-----	-----
bw[0]:	0	100000	0
bw[1]:	0	100000	0
bw[2]:	0	100000	0
bw[3]:	0	100000	0
bw[4]:	40000	60000	0
bw[5]:	0	60000	0
bw[6]:	0	60000	0
bw[7]:	20000	40000	0

Adding a third tunnel, we configure a reservation for 90 Mbps at the highest priority following similar constraints as the previous tunnel [ID 9]. Tunnels 7 and 8 are preempted since only 10 Mbps of remaining reservable bandwidth is available at all priorities, and both of them are request more than that. Tunnel7 had no affinity constraints so it recalculated to select a alternate lowest-cost path that provided the bandwidth (CSR3). Tunnel8 was constrained by affinity so it selected a higher-cost path (CSR8) to meet its bandwidth requirement. Tunnel9, as the highest priority, is what caused the tunnels to recalculate. Just because bandwidth is no longer available for lower priority flows does not mean those tunnels will fail forever; they just cannot compete for the bandwidth on those links.

```
! CSR1
interface Tunnel9
  description BLUE PRIORITY BW REQ (BEST)
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng priority 0 0
  tunnel mpls traffic-eng bandwidth 90000
  tunnel mpls traffic-eng affinity 0x4 mask 0x4
  tunnel mpls traffic-eng path-option 10 dynamic
```

```
CSR1#show ip rsvp sender detail filter session-type 7 tunnel-id 7 | section
outgoing
```

ERO: (outgoing)

132.1.3.3 (Strict IPv4 Prefix, 8 bytes, /32)
132.3.10.10 (Strict IPv4 Prefix, 8 bytes, /32)
132.10.11.11 (Strict IPv4 Prefix, 8 bytes, /32)
11.11.11.11 (Strict IPv4 Prefix, 8 bytes, /32)

CSR1#show ip rsvp sender detail filter session-type 7 tunnel-id 8 | section outgoing

ERO: (outgoing)

132.1.8.8 (Strict IPv4 Prefix, 8 bytes, /32)
132.6.8.6 (Strict IPv4 Prefix, 8 bytes, /32)
132.6.10.10 (Strict IPv4 Prefix, 8 bytes, /32)
132.5.10.5 (Strict IPv4 Prefix, 8 bytes, /32)
132.5.11.11 (Strict IPv4 Prefix, 8 bytes, /32)
11.11.11.11 (Strict IPv4 Prefix, 8 bytes, /32)

CSR1#show ip rsvp sender detail filter session-type 7 tunnel-id 9 | section outgoing

ERO: (outgoing)

132.1.6.6 (Strict IPv4 Prefix, 8 bytes, /32)
132.6.12.12 (Strict IPv4 Prefix, 8 bytes, /32)
132.11.12.11 (Strict IPv4 Prefix, 8 bytes, /32)
11.11.11.11 (Strict IPv4 Prefix, 8 bytes, /32)

RSVP is now tracking different bandwidth reservations on different interfaces based on the new path calculations. Again, this is the result of the 90 Mbps reservation at the higher priority causing the existing TE tunnels to recalculate.

CSR1#show ip rsvp interface

interface	rsvp	allocated	i/f max	flow max	sub max	VRF
Gi2	ena	0	750M	750M	0	
Gi2.513	ena	20M	100M	100M	0	
Gi2.514	ena	0	100M	100M	0	
Gi2.516	ena	90M	100M	100M	0	
Gi2.518	ena	40M	100M	100M	0	
Gi2.519	ena	0	100M	100M	0	

We won't show all of the link bandwidth updates, but checking CSR1's interface to CSR6, we see that only 10 Mbps is available for all priorities as the 90 Mbps reservation was the highest priority.

CSR1#show mpls traffic-eng topology level-2 1.1.1.1 | begin 0006

link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0006.00, nbr_node_id:4,
gen:2921

frag_id: 0, Intf Address: 132.1.6.1, Nbr Intf Address: 132.1.6.6

TE metric: 5, IGP metric: 10, attribute flags: 0x4

SRLGs: None

physical_bw: 1000000 (kbps), max_reservable_bw_global: 100000 (kbps)

max_reservable_bw_sub: 0 (kbps)

	Total Allocated BW (kbps)	Global Pool Reservable BW (kbps)	Sub Pool Reservable BW (kbps)
	-----	-----	-----
bw[0]:	90000	10000	0
bw[1]:	0	10000	0
bw[2]:	0	10000	0
bw[3]:	0	10000	0
bw[4]:	0	10000	0
bw[5]:	0	10000	0
bw[6]:	0	10000	0
bw[7]:	0	10000	0

During the initial pre-emption, the LSPs may issue log messages that show PCALC's inability to meet the requirements. These messages are often transient, but could lead to short-term outages when TE-FRR is not in use.

```
! CSR1
%MPLS_TE-5-TUN: Tun8: LSP path change nil for 8_35, path verification failed
%MPLS_TE-5-LSP: LSP 1.1.1.1 8_37: No path to destination, 0000.0000.0011.00
(bw or affinity)
```

You cannot configure a tunnel to have a higher setup priority over the hold priority. If a tunnel is setup at a certain priority, the tunnel's ability to sustain that priority should be at least as good as the setup. You can, however, give a tunnel better hold priority so ensure that when it is setup, it is less likely to be pre-empted.

```
CSR1(config-if)#tunnel mpls traffic-eng priority 4 5
% Setup priority (4) may not be higher than hold priority (5)
```

Since the configuration is almost identical in XR, we will configure one test for brevity. A tunnel is built to CSR8 [ID 7] requesting 45 Mbps following the path with the lowest IGP metric (TE metric is the default). No IGP metrics have been modified, but this means that any TE metric modifications will not be considered. XRv12 makes the reservation on the interface towards CSR6 when the LSP is signaled.

```
! XRv12
interface tunnel-te7
 description BASIC BW REQ
 ipv4 unnumbered Loopback0
 logging events all
 priority 5 3
 signalled-bandwidth 45000
 destination 1.1.1.1
 path-selection metric igp
 affinity ignore
```



```
path-option 10 dynamic
```

```
RP/0/0/CPU0:XRv12#show rsvp sender session-type lsp-p2p destination 1.1.1.1
detail | begin Outgoing
```

```
Explicit Route (Outgoing):
```

```
Strict, 132.6.12.6/32
Strict, 132.1.6.1/32
Strict, 1.1.1.1/32
```

Checking the TED, we use XRv12 to check CSR6's link to CSR1, confirming the bandwidth reservation. Notice that despite having a setup priority of 5, the hold priority was higher at 3, so if other LSRs want to preempt this path, they have to do so against a priority 3 flow as this is already established. For priority 3 and below, 55 Mbps is remaining, but higher priority flows can use the full 100 Mbps.

```
RP/0/0/CPU0:XRv12#show rsvp reservation session-type lsp-p2p destination
1.1.1.1
```

Destination	Add	DPort	Source	Add	SPort	Pro	Input	IF	Sty	Serv	Rate	Burst
1.1.1.1	7		12.12.12.12	2	0	Gi0/0/0/0.562	SE	LOAD	45000K	1K		

```
RP/0/0/CPU0:XRv12#show mpls traffic-eng topology isis 0000.0000.0006.00 |
begin 0001
```

```
Link[2]:Point-to-Point, Nbr IGP Id:0000.0000.0001.00, Nbr Node Id:4,
gen:79969
```

```
Frag Id:0, Intf Address:132.1.6.6, Intf Id:0
```

```
Nbr Intf Address:132.1.6.1, Nbr Intf Id:0
```

```
TE Metric:10, IGP Metric:10
```

```
Attribute Flags: 0x4
```

```
Ext Admin Group:
```

```
Length: 256 bits
```

```
Value : 0x::4
```

```
Attribute Names: BLUE(2)
```

```
Switching Capability:None, Encoding:unassigned
```

```
BC Model ID:RDM
```

```
Physical BW:1000000 (kbps), Max Reservable BW Global:100000 (kbps)
```

```
Max Reservable BW Sub:0 (kbps)
```

	Total Allocated BW (kbps)	Global Pool Reservable BW (kbps)	Sub Pool Reservable BW (kbps)
bw[0]:	0	100000	0
bw[1]:	0	100000	0
bw[2]:	0	100000	0
bw[3]:	45000	55000	0
bw[4]:	0	55000	0
bw[5]:	0	55000	0
bw[6]:	0	55000	0
bw[7]:	0	55000	0

Despite having all these fancy TE attributes for dynamic calculations, we can arbitrarily steer traffic through a network using a predefined explicit path. This is essentially a hand-made ERO that gets fed into PCALC for sanity checking, but is ultimately delivered to RSVP with minimal changes. This is generally useful for testing purposes, loose-hop expansion, or link/node exclusion. From CSR1, we create a path to XRv11 that follows the path CSR3 > CSR2 > CSR10 > XRv11 [ID 10]. We can specify this using link addresses or TE IDs; link addresses are more strict, while TE IDs allow any link to be chosen if there are parallel links between nodes. The path below uses the most strict method, which is unnecessary in this topology as there are no sets of parallel links. Technically, you don't need to specify the remote TE ID as the last entry, but it is good practice and makes the code self-documenting.

```
! CSR1
ip explicit-path name PATH_1_3_2_10_11 enable
  next-address 132.1.3.3
  next-address 132.3.9.9
  next-address 132.2.9.2
  next-address 132.2.10.10
  next-address 132.10.11.11

interface Tunnel10
  description EXPLICIT PATH STRICT
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 explicit name PATH_1_3_2_10_11
```

Notice that this pre-defined path is almost identical to the ERO delivered from PCALC to RSVP. As shown in the debug below, PCALC is just verifying that the path makes sense. The CSPF algorithm is still invoked to ensure other constraints (affinity, bandwidth, etc) are accounted for, but that isn't the focus now. Using affinity with explicit paths is possible but doesn't make much sense unless loose-hop path expansion or link/node exclusion is in effect.

```
CSR1#show ip rsvp sender detail filter session-type 7 tunnel-id 10 | section outgoing
```

ERO: (outgoing)

```
132.1.3.3 (Strict IPv4 Prefix, 8 bytes, /32)
132.3.9.9 (Strict IPv4 Prefix, 8 bytes, /32)
132.2.9.2 (Strict IPv4 Prefix, 8 bytes, /32)
132.2.10.10 (Strict IPv4 Prefix, 8 bytes, /32)
132.10.11.11 (Strict IPv4 Prefix, 8 bytes, /32)
11.11.11.11 (Strict IPv4 Prefix, 8 bytes, /32)
```

```
CSR1#debug mpls traffic-eng path lookup
TE-PCALC-API: 1.1.1.1_17->11.11.11.11_10 {7}: P2P LSP Path Lookup called
TE-PCALC: 1.1.1.1_17->11.11.11.11_10 {7}: Path Request Info
```

```

Flags:  IP_EXPLICIT_PATH  METRIC_TE
IP explicit-path: Supplied
132.1.3.3
132.3.9.9
132.2.9.2
132.2.10.10
132.10.11.11
11.11.11.11
bw 0, min_bw 0, metric: 0
setup_pri 7, hold_pri 7
affinity_bits 0x0, affinity_mask 0x0
TE-PCALC-PATH: 1.1.1.1_17->11.11.11.11_10 {7}: Area (isis level-2) Path
Lookup begin
TE-PCALC: Verify Path Lookup: 1.1.1.1_17->11.11.11.11_10 {7}: (protocol nil
area nil)
Flags:  METRIC_TE
sub-lsp weight:0 (Total LSP weight:0)
Hop List:
132.1.3.3
132.3.9.9
132.2.9.2
132.2.10.10
132.10.11.11
11.11.11.11
TE-PCALC-VERIFY: VERIFY to 11.11.11.11 BEGIN:
TE-PCALC-VERIFY: Verify:
TE-PCALC-VERIFY: 0000.0000.0001.00, 1.1.1.1 points to
TE-PCALC-VERIFY: 0000.0000.0003.00, 132.1.3.3
TE-PCALC-VERIFY: Verify:
TE-PCALC-VERIFY: 0000.0000.0003.00, 132.1.3.3 points to
TE-PCALC-VERIFY: 0000.0000.0009.00, 132.3.9.9
TE-PCALC-VERIFY: Verify:
TE-PCALC-VERIFY: 0000.0000.0009.00, 132.3.9.9 points to
TE-PCALC-VERIFY: 0000.0000.0002.00, 132.2.9.2
TE-PCALC-VERIFY: Verify:
TE-PCALC-VERIFY: 0000.0000.0002.00, 132.2.9.2 points to
TE-PCALC-VERIFY: 0000.0000.0010.00, 132.2.10.10
TE-PCALC-VERIFY: Verify:
TE-PCALC-VERIFY: 0000.0000.0010.00, 132.2.10.10 points to
TE-PCALC-VERIFY: 0000.0000.0011.00, 132.10.11.11
TE-PCALC-VERIFY: VERIFY to 11.11.11.11 PASSED

```

A quick OAM check verifies the LSP was built in exactly the manner we specified.

```

CSR1#traceroute mpls traffic-eng tunnel 10
Tracing MPLS TE Label Switched Path on Tunnel10, timeout is 2 seconds
[snip]

```

Type escape sequence to abort.

```

0 132.1.3.1 MRU 1500 [Labels: 3008 Exp: 0]
L 1 132.1.3.3 MRU 1500 [Labels: 9011 Exp: 0] 1 ms
L 2 132.3.9.9 MRU 1500 [Labels: 2013 Exp: 0] 29 ms
L 3 132.2.9.2 MRU 1500 [Labels: 10012 Exp: 0] 21 ms
L 4 132.2.10.10 MRU 1500 [Labels: implicit-null Exp: 0] 27 ms
! 5 132.10.11.11 31 ms

```

An alternative approach is to specify the TE IDs. This is simpler to read and understand, but is a little less strict in cases where there are parallel links between nodes. Given the current topology, the result is identical, since there are no parallel links between any pair of nodes. We can also use a numerical ID (not preferred) to identify paths versus names. Below is an example of using path IDs versus names, but this is similar to using numbered ACLs or named ones; it is generally considered a legacy mechanism. The ERO in the PATH message gets expanded to the link values via PCALC. MPLS traceroute confirms the path is the same as it was in the link-specific path. Notice that this explicit path does not specify the last node; this technically works as long as the path specifies the penultimate hop and the tunnel destination is correct, but is more difficult to read/understand/troubleshoot.

```

! CSR1
ip explicit-path identifier 11 enable
  next-address 3.3.3.3
  next-address 9.9.9.9
  next-address 2.2.2.2
  next-address 10.10.10.10

interface Tunnel11
  description EXPLICIT PATH STRICT TE ID
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 explicit identifier 11

CSR1#show mpls traffic-eng tunnels tunnel 11 | section RSVP_Path
  RSVP Path Info:
    My Address: 132.1.3.1
    Explicit Route: 132.1.3.3 132.3.9.9 132.2.9.2 132.2.10.10
                   132.10.11.11 11.11.11.11
    Record Route: NONE
    Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits

```

```

CSR1#traceroute mpls traffic-eng tunnel 11
Tracing MPLS TE Label Switched Path on Tunnel11, timeout is 2 seconds
[snip]

```

```

Type escape sequence to abort.
0 132.1.3.1 MRU 1500 [Labels: 3011 Exp: 0]
L 1 132.1.3.3 MRU 1500 [Labels: 9009 Exp: 0] 2 ms

```

```

L 2 132.3.9.9 MRU 1500 [Labels: 2014 Exp: 0] 34 ms
L 3 132.2.9.2 MRU 1500 [Labels: 10014 Exp: 0] 26 ms
L 4 132.2.10.10 MRU 1500 [Labels: implicit-null Exp: 0] 25 ms
! 5 132.10.11.11 53 ms

```

We can become even more explicit by using the "verbatim" option. This completely bypasses the PCALC sanity check as the explicit-path is turned into an ERO exactly as written. PCALC receives the explicit-path but the topology check is skipped. This can be used for Inter-AS/Inter-area TE, as well as TE over IGP's that don't support TE extensions. For example, you can do MPLS TE over RIP or EIGRP using verbatim paths since the destination address of each PATH message has been identified. Following the same path, we create a new tunnel [ID 12] with verbatim option. We will recycle the explicit path from tunnel 10. OAM confirms the operation. This feature is examined in great detail in the for Inter-AS/Inter-area TE sections of this book. Since the entire TE topology is within a single IGP flooding domain, detailing this feature here is not interesting nor useful.

```

! CSR1
TE-PCALC-API: 1.1.1.1_1->11.11.11.11_10 {7}: P2P LSP Path Lookup called
TE-PCALC: 1.1.1.1_1->11.11.11.11_10 {7}: Path Request Info
  Flags:  IP_EXPLICIT_PATH  NO_TOPOLOGY_CHECK  METRIC_TE
  IP explicit-path: Supplied
    132.1.3.3
    132.3.9.9
    132.2.9.2
    132.2.10.10
    132.10.11.11
    11.11.11.11
  bw 0, min_bw 0, metric: 0
  setup_pri 7, hold_pri 7
  affinity_bits 0x0, affinity_mask 0x0
TE-PCALC-PATH: Get Path Common: Skip topology check for verbatim path to dest
11.11.11.11
TE-PCALC-PATH: get_path_no_check: share count=0, dest=11.11.11.11
TE-PCALC-PATH: static_path_no_check on verbatim path to 11.11.11.11

CSR1#traceroute mpls traffic-eng tunnel 12
Tracing MPLS TE Label Switched Path on Tunnel12, timeout is 2 seconds
[snip]

Type escape sequence to abort.
  0 132.1.3.1 MRU 1500 [Labels: 3008 Exp: 0]
L 1 132.1.3.3 MRU 1500 [Labels: 9009 Exp: 0] 65 ms
L 2 132.3.9.9 MRU 1500 [Labels: 2012 Exp: 0] 31 ms
L 3 132.2.9.2 MRU 1500 [Labels: 10016 Exp: 0] 36 ms
L 4 132.2.10.10 MRU 1500 [Labels: implicit-null Exp: 0] 27 ms
! 5 132.10.11.11 17 ms

```

Explicit paths can also use loose hop expansion. We can specify a path that must traverse CSR6, but the intermediate hops can be anything (provided other constraints are met). A path from CSR1 to XRv11 that meets the aforementioned requirements is shown below [ID 13].

```
! CSR1
ip explicit-path name PATH_1_6_11_LOOSE enable
  next-address loose 6.6.6.6

interface Tunnel13
  description EXPLICIT PATH LOOSE
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 explicit name PATH_1_6_2_11_LOOSE
```

The signaling for loose paths is different than normal paths. Because PCALC on CSR1 was told to use an explicit path, it finds a path to CSR6 and creates an ERO for it. CSR1's ERO is very short as it only expands to include CSR6. The PCALC debug is shown below.

```
CSR1#show ip rsvp sender detail filter session-type 7 tunnel-id 13 | section
outgoing
```

```
ERO: (outgoing)
  132.1.6.6 (Strict IPv4 Prefix, 8 bytes, /32)
  6.6.6.6 (Strict IPv4 Prefix, 8 bytes, /32)
```

```
! CSR1
TE-PCALC-API: 1.1.1.1_55->11.11.11.11_13 {7}: P2P LSP Path Lookup called
TE-PCALC: 1.1.1.1_55->11.11.11.11_13 {7}: Path Request Info
  Flags: IP_EXPLICIT_PATH METRIC_TE
  IP explicit-path: Supplied
    6.6.6.6 Loose
  bw 0, min_bw 0, metric: 0
  setup_pri 7, hold_pri 7
  affinity_bits 0x0, affinity_mask 0x0
TE-PCALC-PATH: 1.1.1.1_55->11.11.11.11_13 {7}: Area (isis level-2) Path
Lookup begin
TE-PCALC-PATH:Path from 0000.0000.0001.00 -> 0000.0000.0006.00:
  132.1.6.1->132.1.6.6 (admin_weight=5):
    num_hops 2, accumulated_aw 5, min_bw 100000
TE-PCALC-PATH: 1.1.1.1_55->11.11.11.11_13 {7}: Freeing rrr_path_setup_t
TE-PCALC-PATH: 1.1.1.1_55->11.11.11.11_13 {7}: Free all paths in path tree
TE-PCALC: Verify Path Lookup: 1.1.1.1_55->11.11.11.11_13 {7}: (protocol nil
area nil)
  Flags: METRIC_TE
  Last Strict Router: 6.6.6.6
  sub-lsp weight:0 (Total LSP weight:5)
  Hop List:
```

132.1.6.6

6.6.6.6

TE-PCALC-VERIFY: VERIFY to 6.6.6.6 BEGIN:

TE-PCALC-VERIFY: Verify:

TE-PCALC-VERIFY: 0000.0000.0001.00, 1.1.1.1 points to

TE-PCALC-VERIFY: 0000.0000.0006.00, 132.1.6.6

TE-PCALC-VERIFY: VERIFY to 6.6.6.6 PASSED

Debugging RSVP messaging on CSR6, the RSVP PATH message arrives with an ERO destined for CSR6. CSR6 needs to signal the remainder of the LSP, but it cannot make an ERO since it has no basis for determining the path; every other router in the path will determine the path dynamically upon receiving the path message. The outgoing PATH has no ERO.

! CSR6

Incoming Path:

version:1 flags:0000 cksum:F048 ttl:255 reserved:0 length:196

SESSION type 7 length 16:

Tun Dest: 11.11.11.11 Tun ID: 13 Ext Tun ID: 1.1.1.1

HOP type 1 length 12:

Hop Addr: 132.1.6.1 LIH: 0x0000000E

TIME_VALUES type 1 length 8 :

Refresh Period (msec): 30000

EXPLICIT_ROUTE type 1 length 20:

132.1.6.6 (Strict IPv4 Prefix, 8 bytes, /32)

6.6.6.6 (Strict IPv4 Prefix, 8 bytes, /32)

LABEL_REQUEST type 1 length 8 :

Layer 3 protocol ID: 2048

SESSION_ATTRIBUTE type 7 length 28:

Setup Prio: 7, Holding Prio: 7

Flags: (0x4) SE Style

Session Name: EXPLICIT PATH LOOSE

[snip]

Outgoing Path:

version:1 flags:0000 cksum:D78B ttl:254 reserved:0 length:176

SESSION type 7 length 16:

Tun Dest: 11.11.11.11 Tun ID: 13 Ext Tun ID: 1.1.1.1

HOP type 1 length 12:

Hop Addr: 132.6.12.6 LIH: 0x0000000D

TIME_VALUES type 1 length 8 :

Refresh Period (msec): 30000

LABEL_REQUEST type 1 length 8 :

Layer 3 protocol ID: 2048

SESSION_ATTRIBUTE type 7 length 28:

Setup Prio: 7, Holding Prio: 7

Flags: (0x4) SE Style

Session Name: EXPLICIT PATH LOOSE

[snip]

As a result of this, CSR1 has no idea what the actual path is from a control-plane perspective. After CSR6, the actual LSP path is not known to the head-end. We can use the RRO object in the tunnel to record the route in the RESV messages.

```
CSR1#show mpls traffic-eng tunnels tunnel 13 | section RSVP_Signalling
  RSVP Signalling Info:
    Src 1.1.1.1, Dst 11.11.11.11, Tun_Id 13, Tun_Instance 55
  RSVP Path Info:
    My Address: 132.1.6.1
    Explicit Route: 132.1.6.6 6.6.6.6
    Record Route:  NONE
    Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits
  RSVP Resv Info:
    Record Route:  NONE
    Fspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits

! CSR1
interface Tunnel13
 tunnel mpls traffic-eng record-route
```

Clearing the tunnel (shut/no shut) allows the path to be signaled again. Looking at CSR6 RSVP signaling, we can see the RRO being built. The RRO is included in the PATH message to record the PHOPs, which isn't as relevant as the RRO in the RESV message to carry the NHOPs. This information is now available on the headend, even with loose explicit-paths. If this feature seems confusing, it is a result of the artificial setup. This is typically used for Inter-AS/Inter-area TE as it has little purpose within a TE domain. The feature is examined in deeper detail in those appropriate sections.

```
! CSR6
Incoming Path:
[snip]
RECORD_ROUTE          type 1 length 12:
 132.1.6.1/32, Flags:0x0 (No Local Protection)

Outgoing Path:
[snip]
RECORD_ROUTE          type 1 length 20:
 132.6.12.6/32, Flags:0x0 (No Local Protection)
 132.1.6.1/32, Flags:0x0 (No Local Protection)

Incoming Resv:
[snip]
RECORD_ROUTE          type 1 length 20:
 132.6.12.12/32, Flags:0x0 (No Local Protection)
 132.11.12.11/32, Flags:0x0 (No Local Protection)

Outgoing Resv:
```



```
[snip]
RECORD_ROUTE          type 1 length 28:
  132.6.12.6/32, Flags:0x0 (No Local Protection)
  132.6.12.12/32, Flags:0x0 (No Local Protection)
  132.11.12.11/32, Flags:0x0 (No Local Protection)

CSR1#show mpls traffic-eng tunnels tunnel 13 | section RSVP_Signalling
RSVP Signalling Info:
  Src 1.1.1.1, Dst 11.11.11.11, Tun_Id 13, Tun_Instance 56
RSVP Path Info:
  My Address: 132.1.6.1
  Explicit Route: 132.1.6.6 6.6.6.6
  Record Route:
  Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits
RSVP Resv Info:
  Record Route: 132.6.12.6 132.6.12.12 132.11.12.11
  Fspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits
```

Explicit paths on XR are very similar to XE, so for brevity we examine one example using TE IDs in a strict path [ID 11]. In XR, you must specify the last-hop in explicit paths.

```
! XRv11
explicit-path name PATH_11_5_2_3_4_1
  index 10 next-address strict ipv4 unicast 5.5.5.5
  index 20 next-address strict ipv4 unicast 2.2.2.2
  index 30 next-address strict ipv4 unicast 3.3.3.3
  index 40 next-address strict ipv4 unicast 4.4.4.4
  index 50 next-address strict ipv4 unicast 1.1.1.1

interface tunnel-te11
  description EXPLICIT PATH STRICT TE ID
  ipv4 unnumbered Loopback0
  logging events all
  destination 1.1.1.1
  affinity ignore
  path-option 10 explicit name PATH_11_5_2_3_4_1
```

You can verify the ERO using the "dst-port" option and specifying the tunnel ID also. The ERO is a per-link expansion of the TE IDs were specified in the explicit path. This implies that PCALC was involved to evaluate those TE IDs into the appropriate link addresses.

```
RP/0/0/CPU0:XRv11#show rsvp sender session-type lsp-p2p dst-port 11 detail |
begin Outgoing
  Explicit Route (Outgoing):
    Strict, 132.5.11.5/32
    Strict, 132.2.5.2/32
    Strict, 132.2.3.2/32
    Strict, 132.2.3.3/32
```

```
Strict, 132.3.4.4/32
Strict, 132.1.4.1/32
Strict, 1.1.1.1/32
```

Bandwidth can be automatically adjusted for a TE tunnel based on measured traffic also. This can help optimize bandwidth usage to ensure that tunnels are requesting the bandwidth they actually require, versus making reservations that remain under-utilized. The feature samples that average data rate through the tunnel and periodically adjusts the signaled-bandwidth to the largest sample since the last adjustment. Globally, we can enable the feature and define the interval between the samples (5 minute default). On a per-tunnel basis, we can define the interval between adjustments (24 hour default), min/max bandwidth reservations as upper and lower bounds (unlimited by default), and a threshold that signals how much different the sample must be in order to make a change (10% by default). **As a best practice, the timers should follow this logic: load-interval < global sampling frequency < tunnel adjustment frequency.** This way, the load-interval can change sufficiently between samples, and the tunnel adjustments will run less frequently than the sampler. If a tunnel is down when the global sample time hits, the sampler is ignored. Below is an example of auto-bandwidth. Every minute, the global process samples the bandwidth on all tunnels. The bandwidth is adjusted every 5 minutes based on the traffic going through it. Technically, we don't have to specify an initial bandwidth, but we do using 5 Mbps. We specify a lower bound of 2 Mbps and upper bound of 10 Mbps also [ID 14]. This ensures that the tunnel is not grossly over or underutilized.

```
! CSR1
mpls traffic-eng auto-bw timers frequency 60

interface Tunnel14
  description AUTO BW TIMERS
  ip unnumbered Loopback0
  load-interval 30
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng priority 7 7
  tunnel mpls traffic-eng bandwidth 5000
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 dynamic
  tunnel mpls traffic-eng auto-bw frequency 300 max-bw 10000 min-bw 2000
```

The tunnel comes up immediately and we can see 5 Mbps has been reserved. We can check the TE tunnel details to see the auto-bw information. The (300/96) represents the configured adjustment timer over the remaining time until the next evaluation. It counts down to zero. The zero following this field is the last measured bandwidth, which is also zero at present. The samples collected represents how many samples, based on the global interval have occurred. One was skipped since the tunnel was down during the initial configuration.

```
CSR1#show ip rsvp reservation filter session-type 7 tunnel-id 14
Destination      Tun Sender      TunID LSPID Next Hop      I/F      Fi Serv BPS
```

```
11.11.11.11      1.1.1.1      14      1      132.1.6.6      Gi2.516      SE LOAD 5M
```

```
CSR1#show mpls traffic-eng tunnels tunnel 14 | section auto-bw
```

```
auto-bw: (300/96) 0 Bandwidth Requested: 5000
```

```
Samples Missed 1: Samples Collected 3
```

After the timer hits zero, the adjustment happens. Since we don't have any traffic going through the tunnel, we expect the lower-bound measurement of 2 Mbps to be signaled. We see a log message stating that the tunnel has been "reoptimized", which is what happens periodically with TE tunnels anyway, but this was initiated by the auto-bw process. The LSP ID has changed from 1 to 2, and RSVP is now reporting 2 Mbps reserved on the outgoing interface. The tunnel details also show the change. This can introduce churn into the topology as the TED information is re-flooded.

```
! CSR1
```

```
%MPLS_TE-5-TUN: Tun14: installed LSP 14_2 (popt 10) for 14_1 (popt 10),  
reopt. LSP is up
```

```
CSR1#show interfaces tunnel14 | include put_rate
```

```
30 second input rate 0 bits/sec, 0 packets/sec
```

```
30 second output rate 0 bits/sec, 0 packets/sec
```

```
CSR1#show ip rsvp reservation filter session-type 7 tunnel-id 14
```

Destination	Tun Sender	TunID	LSPID	Next Hop	I/F	Fi	Serv	BPS
11.11.11.11	1.1.1.1	14	2	132.1.6.6	Gi2.516	SE	LOAD	2M

```
CSR1#show mpls traffic-eng tunnels tunnel 14 | section auto-bw
```

```
auto-bw: (300/221) 0 Bandwidth Requested: 2000
```

```
Samples Missed 1: Samples Collected 1
```

We can also perform adjustments based on consecutive threshold crossings based on the global interval. This can react more quickly to changing conditions since we don't have to wait for all the samples, we can say that if we are consistently incorrect, we can make an adjustment sooner. XE does not appear to support "underflow", which means we cannot react by reducing bandwidth. The timer mechanism can still react to underflow conditions, but not the event-driven threshold feature [ID 15]. If the bandwidth is more than 5% greater than the current signaled bandwidth 3 times in a row (sampled by the global frequency), then the overflow adjustment occurs.

```
! CSR1
```

```
interface Tunnel15
```

```
description AUTO BW FLOW THRESHOLDS
```

```
ip unnumbered Loopback0
```

```
tunnel mode mpls traffic-eng
```

```
tunnel destination 11.11.11.11
```

```
tunnel mpls traffic-eng priority 7 7
```

```
tunnel mpls traffic-eng bandwidth 5000
```

```
tunnel mpls traffic-eng affinity 0x0 mask 0x0
```

```
tunnel mpls traffic-eng path-option 10 dynamic
tunnel mpls traffic-eng auto-bw overflow-limit 3 overflow-threshold 5 max-bw
8000 min-bw 1500
```

```
CSR1#show ip rsvp reservation filter session-type 7 tunnel-id 15
Destination      Tun Sender      TunID LSPID Next Hop      I/F      Fi Serv BPS
11.11.11.11      1.1.1.1         15    1      132.1.6.6      Gi2.516  SE LOAD 5M
```

```
CSR1#show mpls traffic-eng tunnels tunnel 15 | section auto-bw
auto-bw: (300/239) 0 Bandwidth Requested: 5000
Adjustment Threshold: 5%
Overflow Limit: 3 Overflow Threshold: 5%
Overflow Threshold Crossed: 0
Samples Missed 0: Samples Collected 1
```

Another option includes the ability to simply monitor bandwidth on the tunnel, but take no action. The output looks similar except there are no adjustments ever being made. The tunnel can still request bandwidth but it cannot be adjusted. The output specifically says "collected" and not "requested" since the latter never changes. We don't specify the max-bw since it isn't relevant, and the maximum value is applied for this field.

```
! CSR1
interface Tunnel16
description AUTO BW MONITOR
ip unnumbered Loopback0no ip address
tunnel mode mpls traffic-eng
tunnel destination 11.11.11.11
tunnel mpls traffic-eng priority 7 7
tunnel mpls traffic-eng bandwidth 5000
tunnel mpls traffic-eng affinity 0x0 mask 0x0
tunnel mpls traffic-eng path-option 10 dynamic
tunnel mpls traffic-eng auto-bw frequency 300 max-bw 4294967295 collect-bw
```

```
CSR1#show mpls traffic-eng tunnels tunnel 16 | section auto-bw
auto-bw: (300/189) 0 Bandwidth Last Collected: 0
Samples Missed 1: Samples Collected 1
```

```
CSR1#show ip rsvp reservation filter session-type 7 tunnel-id 16
Destination      Tun Sender      TunID LSPID Next Hop      I/F      Fi Serv BPS
11.11.11.11      1.1.1.1         16    1      132.1.6.6      Gi2.516  SE LOAD 5M
```

The auto-bandwidth feature is very similar in XR. We will configure a TE tunnel from XRv11 to CSR8 requesting 7 Mbps of bandwidth initially, with a min/max of 2.5/9.5 Mbps [ID 14]. The adjustment "application" is every 5 minutes where the global timer takes samples once every minute (IOS measures this in seconds while XR uses minutes). The tunnel will adjust bandwidth when the difference is at least 15% different and 10 kbps different to avoid constant churn. As an unrelated feature, the path-option is

limited to a specific IS-IS process and level, which can be useful when constraining TE tunnels to a single TE topology (could be useful if, for example, XRv11 were an ABR or L1L2 router).

```
! XRv11
mpls traffic-eng
  auto-bw collect frequency 1

interface tunnel-te14
  description AUTO BW TIMERS
  ipv4 unnumbered Loopback0
  load-interval 30
  auto-bw
    bw-limit min 2500 max 9500
    adjustment-threshold 15 min 10
    application 5
    logging events all
    signalled-bandwidth 7000
    destination 8.8.8.8
    affinity ignore
    path-option 10 dynamic isis 132 level 2
```

When the tunnel comes up, we can see the auto-bandwidth details. This is very similar to the XE output. XR also has an auto-bandwidth brief command that shows each tunnel as a row, along with the relevant bandwidth information.

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 14 | begin Auto-bw
Auto-bw: enabled
  No BW Applied
  Bandwidth Min/Max: 2500-9500 kbps
  Application Frequency: 5 min   Jitter: 0s   Time Left: 3m 40s
  Collection Frequency: 1 min
  Samples Collected: 2   Next: 35s
  Highest BW: 0 kbps   Underflow BW: 0 kbps
  Adjustment Threshold: 15%   10 kbps
  Overflow Detection disabled
  Underflow Detection disabled
[snip]
```

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels auto-bw brief
```

Tunnel Name	LSP ID	Last appl BW (kbps)	Requested BW (kbps)	Signalled BW (kbps)	Highest BW (kbps)	Application Time Left
tunnel-te14	2	-	7000	7000	0	3m 18s

After five minutes, we can see the tunnel has been reoptimized to 2.5 Mbps. Since there is no traffic in the tunnel, the minimum bandwidth of 2.5 Mbps is applied. The LSP history also shows that the previous

LSP ID was 2, the new one is 3, since the LSP was "reoptimized" by the auto-bandwidth process. The brief output shows similar output in summary form.

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 14 | begin Auto-bw
```

```
Auto-bw: enabled
```

```
Last BW Applied: 2500 kbps CT0 BW Applications: 1
```

```
Last Application Trigger: Periodic Application
```

```
Bandwidth Min/Max: 2500-9500 kbps
```

```
Application Frequency: 5 min Jitter: 0s Time Left: 5s
```

```
Collection Frequency: 1 min
```

```
Samples Collected: 4 Next: 0s
```

```
Highest BW: 0 kbps Underflow BW: 0 kbps
```

```
Adjustment Threshold: 15% 10 kbps
```

```
Overflow Detection disabled
```

```
Underflow Detection disabled
```

```
Fast Reroute: Disabled, Protection Desired: None
```

```
Path Protection: Not Enabled
```

```
BFD Fast Detection: Disabled
```

```
Reoptimization after affinity failure: Enabled
```

```
Soft Preemption: Disabled
```

```
History:
```

```
Tunnel has been up for: 00:25:01
```

```
Current LSP:
```

```
Uptime: 00:19:55
```

```
Prior LSP:
```

```
ID: 2 Path Option: 10
```

```
Removal Trigger: reoptimization completed
```

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels auto-bw brief
```

Tunnel Name	LSP ID	Last appl BW (kbps)	Requested BW (kbps)	Signalled BW (kbps)	Highest BW (kbps)	Application Time Left
tunnel-te14	3	2500	2500	2500	0	21s

Next, we will look at using flow thresholds on XRv11 [ID 15]. XR also supports the underflow option which allows over-subscribed tunnels to be reduced again, where XE only supports overflow (other direction). This tunnel requests 10 Mbps of bandwidth and is constrained to the range of 8-20 Mbps. If there is a 5% increase in bandwidth that is at least 256 kbps of a change, and this occurs in three samples, the tunnel can react to the overflow condition and request more bandwidth. If there is a 10% reduction in bandwidth that is at least 128 kbps of a change, and this occurs two times, the tunnel can react to the underflow condition and request less bandwidth. Both are considered reoptimizations and resignal the LSP. The application is defaulted to 1440 minutes (24 hours) so without the flow thresholds, it would take a long time for the tunnel to reoptimize using default timers.

```
! XRv11
```

```
interface tunnel-te15
```

```
description AUTO BW FLOW THRESHOLDS
```

```

ipv4 unnumbered Loopback0
load-interval 30
auto-bw
  bw-limit min 8000 max 20000
  overflow threshold 5 min 256 limit 3
  underflow threshold 10 min 128 limit 2
logging events all
signalled-bandwidth 10000
destination 3.3.3.3
affinity ignore
path-option 10 dynamic

```

The tunnel comes up as expected (output not shown). We see a log message showing the underflow condition as there is no traffic in the tunnel after about 2 minutes. The global sample interval is 1 minute, so after two under-threshold conditions, the underflow event is triggered. The new signaled-bandwidth is the minimum, 8 Mbps, per the configuration. The brief output also shows tunnel-te14, which is shutdown, and has LSP ID 0 since the tunnel is not up. This tunnel was shutdown after the last test for clarity.

```

! XRv11
%ROUTING-MPLS_TE-5-LSP_REOPT : tunnel-te15 (signalled-name: XRv11_t15, old
LSP Id: 4, new LSP Id: 5) has been reoptimized; reason: Auto BW Bandwidth
Underflow Change.

```

```

RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 15 | begin Auto-bw
Auto-bw: enabled
  Last BW Applied: 8000 kbps CT0    BW Applications: 2
  Last Application Trigger: Underflow Condition
  Bandwidth Min/Max: 8000-20000 kbps
  Application Frequency: 1440 min   Jitter: 0s    Time Left: 23h 58m 38s
  Collection Frequency: 1 min
  Samples Collected: 1    Next: 38s
  Highest BW: 0 kbps      Underflow BW: 0 kbps
  Adjustment Threshold: 10%    10 kbps
  Overflow Threshold: 5%    256 kbps    Limit: 0/3    Overflow BW
Applications: 0
  Underflow Threshold: 10%    128 kbps    Limit: 1/2    Underflow BW
Applications: 3
  Fast Reroute: Disabled, Protection Desired: None
  Path Protection: Not Enabled
  BFD Fast Detection: Disabled
  Reoptimization after affinity failure: Enabled
  Soft Preemption: Disabled
History:
  Tunnel has been up for: 00:06:52
  Current LSP:
    Uptime: 00:03:21
  Prior LSP:

```

ID: 4 Path Option: 10
Removal Trigger: reoptimization completed

RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels auto-bw brief

Tunnel Name	LSP ID	Last appl BW (kbps)	Requested BW (kbps)	Signalled BW (kbps)	Highest BW (kbps)	Application Time Left
tunnel-te14	0	2500	7000	7000	0	2m 0s
tunnel-te15	7	8000	8000	8000	0	23h 58m 55s

Last, we will examine bandwidth monitoring on XR for TE tunnels. We create a tunnel to CSR1 only for collecting bandwidth statistics [ID 16]. The tunnel requests 512 kbps of bandwidth and this value can never change since we are only monitoring BW on this tunnel. The "Highest BW" field represents the highest sample of bandwidth seen. The underflow bandwidth is the difference between the highest measured bandwidth and current bandwidth.

```
! XRv11
interface tunnel-te16
  description AUTO BW MONITOR
  ipv4 unnumbered Loopback0
  load-interval 30
  auto-bw
    collect-bw-only
  logging events all
  signalled-bandwidth 512
  destination 1.1.1.1
  affinity ignore
  path-option 10 dynamic
```

RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 16 | begin Auto-bw

```
Auto-bw: enabled (collect bw only)
No BW Applied
Collect-only Requested BW: 512 kbps
Bandwidth Min/Max: 0-4294967295 kbps
Application Frequency: 1440 min    Jitter: 0s    Time Left: 23h 59m 37s
Collection Frequency: 1 min
Samples Collected: 0    Next: 2s
Highest BW: 0 kbps    Underflow BW: 0 kbps
Adjustment Threshold: 10%    10 kbps
Overflow Detection disabled
Underflow Detection disabled
```

RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels auto-bw brief

Tunnel Name	LSP ID	Last appl BW (kbps)	Requested BW (kbps)	Signalled BW (kbps)	Highest BW (kbps)	Application Time Left
tunnel-te14	0	2500	7000	7000	0	1m 26s
tunnel-te15	0	8000	10000	10000	0	23h 55m 21s

Many of these features can be combined to create some very sophisticated TE tunnels using multiple path-options. Individual path options have can separate LSP attributes for maximum flexibility. On CSR1, we configure a tunnel to CSR5 to follow these requirements, in sequence:

1. Red path, 40 Mbps reservation at priority 5
2. Any color but must avoid CSR9, 105 Mbps reservation
3. Blue path, 20 Mbps reservation, must transit XRv12, should record the route taken, and the path can never be changed
4. Any path as a last resort, and should monitor the bandwidth used for future plans

We create several LSP attribute classes to account for these options. Many of these options can be configured directly on the tunnel, which apply to all path-options. Applying an LSP attribute class to a path-option overrides whichever options are specified at the tunnel interface level. For example, affinity is cleared on the tunnel, so options 2 and 4 don't need to specify affinity since ignoring it is the tunnel default. These requirements use a combination of explicit and dynamic paths to meet the transit routing constraints for some options. One restriction is that all bandwidth requests for all path-options must have the same priority, so 5 is used for all options and is defined in the tunnel for inheritance [ID 17].

```
! CSR1
mpls traffic-eng lsp attributes LSP_ATT_RED40
  affinity 0x1 mask 0x1
  bandwidth 40000

mpls traffic-eng lsp attributes LSP_ATT_NO_R9
  bandwidth 105000

mpls traffic-eng lsp attributes LSP_ATT_BLUE20
  affinity 0x4 mask 0x4
  bandwidth 20000
  lockdown
  record-route

mpls traffic-eng lsp attributes LSP_ATT_FALLBACK
  auto-bw collect-bw

ip explicit-path name EP_NO_R9 enable
  exclude-address 9.9.9.9

ip explicit-path name EP_TRANSIT_XRV12 enable
  next-address loose 12.12.12.12

interface Tunnel17
  description COMPLEX
  ip unnumbered Loopback0
```

```

tunnel mode mpls traffic-eng
tunnel destination 5.5.5.5
tunnel mpls traffic-eng priority 5 5
tunnel mpls traffic-eng affinity 0x0 mask 0x0
tunnel mpls traffic-eng path-option 1 dynamic attributes LSP_ATT_RED40
tunnel mpls traffic-eng path-option 2 explicit name EP_NO_R9 attributes
LSP_ATT_NO_R9
tunnel mpls traffic-eng path-option 3 explicit name EP_TRANSIT_XRV12
attributes LSP_ATT_BLUE20
tunnel mpls traffic-eng path-option 100 dynamic attributes LSP_ATT_FALLBACK

```

We can enable debugging on CSR1 to look at the PCALC lookup process. PCALC is invoked for each path-option in sequence. The first option tries to find a red path from CSR1 to CSR5, but there are none. This option fails due to affinity which the log message reveals. The combination of syslog and debug messages is always ideal for troubleshooting.

```

! CSR1, path-option 1
TE-PCALC-API: 1.1.1.1_13->5.5.5.5_17 {7}: P2P LSP Path Lookup called
TE-PCALC: 1.1.1.1_13->5.5.5.5_17 {7}: Path Request Info
  Flags:  METRIC_TE
  IP explicit-path: None (dynamic)
  bw 40000, min_bw 0, metric: 0
  setup_pri 5, hold_pri 5
  affinity_bits 0x1, affinity_mask 0x1
TE-PCALC-PATH: 1.1.1.1_13->5.5.5.5_17 {7}: Area (isis level-2) Path Lookup
begin
TE-PCALC-PATH: 1.1.1.1_13->5.5.5.5_17 {7}: Get path: Failed to find a path to
destination
TE-PCALC-PATH: 1.1.1.1_13->5.5.5.5_17 {7}: Area (isis level-2) Path Lookup
end: path not found
TE-PCALC-API: 1.1.1.1_13->5.5.5.5_17 {7}: P2P LSP Path Lookup result: failed
%MPLS_TE-5-LSP: LSP 1.1.1.1 17_13: No path to destination, 0000.0000.0005.00
(affinity)

```

The second option inherits the "ignored" affinity attribute but is requesting more bandwidth than is available. PCALC returns success since there is a path, but the bandwidth admission control mechanism rejects this as a viable path.

```

! CSR1, path-option 2
TE-PCALC-API: 1.1.1.1_14->5.5.5.5_17 {7}: P2P LSP Path Lookup called
TE-PCALC: 1.1.1.1_14->5.5.5.5_17 {7}: Path Request Info
  Flags:  IP_EXPLICIT_PATH  METRIC_TE
  IP explicit-path: Supplied
  9.9.9.9
  bw 105000, min_bw 0, metric: 0
  setup_pri 5, hold_pri 5
  affinity_bits 0x0, affinity_mask 0x0

```

```

TE-PCALC-PATH: 1.1.1.1_14->5.5.5.5_17 {7}: Area (isis level-2) Path Lookup
begin
TE-PCALC-PATH:Path from 0000.0000.0001.00 -> 0000.0000.0005.00:
  132.5.10.10->132.5.10.5 (admin_weight=25):
  132.6.10.6->132.6.10.10 (admin_weight=15):
  132.1.6.1->132.1.6.6 (admin_weight=5):
  num_hops 4, accumulated_aw 25, min_bw 100000
TE-PCALC-PATH: 1.1.1.1_14->5.5.5.5_17 {7}: Area (isis level-2) Path Lookup
end: path found
TE-PCALC-API: 1.1.1.1_14->5.5.5.5_17 {7}: P2P LSP Path Lookup result: success
TE-PCALC: 1.1.1.1_14->5.5.5.5_17 {7}: modify bandwidth: [0000.0000.0005.00]
(isis level-2)
[snip]
%MPLS_TE-5-LSP: LSP 1.1.1.1 17_14: Path Error from 1.1.1.1: Admission control
Failure: Requested bandwidth unavailable (flags 4)
%MPLS_TE-5-LSP: LSP 1.1.1.1 17_14: DOWN: path error
%MPLS_TE-5-TUN: Tun17: installed LSP nil for 17_14 (popt 2), path error
%MPLS_TE-5-TUN: Tun17: LSP path change nil for 17_14, path error

```

The third option chooses a blue path with 20 Mbps of bandwidth that transits XRv12. Since the PATH message terminates on XRv12, a lot of path control is lost after that point. This is another semi-awkward usage of loose-hop expansion within an TE domain. We see that CSR10 is in the path despite not being a blue link, which demonstrates the danger of using loose-hop inclusion. Nonetheless, this is a valid path-option.

```

! CSR1, path-option 3
TE-PCALC-API: 1.1.1.1_16->5.5.5.5_17 {7}: P2P LSP Path Lookup called
TE-PCALC: 1.1.1.1_16->5.5.5.5_17 {7}: Path Request Info
  Flags: IP_EXPLICIT_PATH METRIC_TE
  IP explicit-path: Supplied
    12.12.12.12 Loose
  bw 20000, min_bw 0, metric: 0
  setup_pri 5, hold_pri 5
  affinity_bits 0x4, affinity_mask 0x4
TE-PCALC-PATH: 1.1.1.1_16->5.5.5.5_17 {7}: Area (isis level-2) Path Lookup
begin
[snip]
TE-PCALC-PATH: 1.1.1.1_16->5.5.5.5_17 {7}: Area (isis level-2) Path Lookup
end: path found
TE-PCALC-API: 1.1.1.1_16->5.5.5.5_17 {7}: P2P LSP Path Lookup result: success
%MPLS_TE-5-TUN: Tun17: installed LSP 17_16 (popt 3) for nil, got 1st feasible
path opt
%MPLS_TE-5-LSP: LSP 1.1.1.1 17_16: UP
%MPLS_TE-5-TUN: Tun17: LSP path change 17_16 for nil, normal

```

We can verify the tunnel is functional by checking the tunnel details and using OAM for traceroute. The RRO seems to be missing the CSR10-XRv12 hop as well. Notice that all of the other path-options are

listed, by option 3 is used for “setup”. This implies that options 1 and 2 were untenable while option 100 was not evaluated yet.

```
CSR1#show mpls traffic-eng tunnels tunnel 17 | section Status|RSVP_Signalling
Status:
  Admin: up      Oper: up      Path: valid      Signalling: connected
  path option 3, type explicit EP_TRANSIT_XRV12 (Basis for Setup, path
weight 15)
  path option 1, type dynamic
  path option 2, type explicit EP_NO_R9
  path option 100, type dynamic

RSVP Signalling Info:
  Src 1.1.1.1, Dst 5.5.5.5, Tun_Id 17, Tun_Instance 16
RSVP Path Info:
  My Address: 132.1.6.1
  Explicit Route: 132.1.6.6 132.6.12.12 12.12.12.12
  Record Route:
  Tspec: ave rate=20000 kbits, burst=1000 bytes, peak rate=20000 kbits
RSVP Resv Info:
  Record Route: 132.6.12.6 132.6.12.12 132.5.10.10 132.5.10.5
  Fspec: ave rate=20000 kbits, burst=1000 bytes, peak rate=20000 kbits
```

```
CSR1#traceroute mpls traffic-eng tunnel 17
Tracing MPLS TE Label Switched Path on Tunnel17, timeout is 2 seconds
[snip]
```

```
Type escape sequence to abort.
 0 132.1.6.1 MRU 1500 [Labels: 6010 Exp: 0]
L 1 132.1.6.6 MRU 1500 [Labels: 92010 Exp: 0] 3 ms
L 2 132.6.12.12 MRU 1500 [Labels: 10010 Exp: 0] 44 ms
L 3 132.10.12.10 MRU 1500 [Labels: implicit-null Exp: 0] 21 ms
! 4 132.5.10.5 29 ms
```

We will demonstrate a complicated TE tunnel on XR also [ID 17]. The XR options are surprisingly limited as the only significant things we can do are specify the affinity and bandwidth. Options like bandwidth priority and record-route are global to the tunnel, not the LSP attributes per path-option. In this example, XRv12 follows similar rules as CSR1 except builds a tunnel to CSR4, minus the loose next-hop.

```
! XRv12
explicit-path name EP_NO_R9
  index 1 exclude-address ipv4 unicast 9.9.9.9

interface tunnel-te17
  description COMPLEX
  ipv4 unnumbered Loopback0
  logging events all
  destination 4.4.4.4
```

```

record-route
affinity ignore
path-option 1 dynamic attribute-set LSP_ATT_RED40
path-option 2 explicit name EP_NO_R9 attribute-set LSP_ATT_NO_R9
path-option 100 dynamic lockdown attribute-set LSP_ATT_ORANGE20

mpls traffic-eng
attribute-set path-option LSP_ATT_NO_R9
signalled-bandwidth 105000 class-type 0
no affinity ignore

attribute-set path-option LSP_ATT_RED40
signalled-bandwidth 40000 class-type 0
affinity include RED

attribute-set path-option LSP_ATT_ORANGE20
signalled-bandwidth 20000 class-type 0
affinity include ORANGE

```

Because there is a red path with 40 Mbps available, that first option is valid. The tunnel is up and operational and the backup path-options are also listed; only details for the active path options are expanded as shown below. The RSVP signaling information is included in the "detail" output for the tunnel, and this is where we can see the RESV RRO. The path goes from CSR9 to CSR3 to CSR4, which are all red links as expected. We also quickly confirm the 40 Mbps reservation is properly allocated by RSVP.

```

RP/0/0/CPU0:XRv12#show mpls traffic-eng tunnels 17
Name: tunnel-te17 Destination: 4.4.4.4 Ifhandle:0x980
Signalled-Name: XRv12_t17
Status:
  Admin:    up Oper:    up Path:  valid Signalling: connected

  path option 1, type dynamic (Basis for Setup, path weight 30)
    Path-option attribute: LSP_ATT_RED40
    Number of affinity constraints: 1
      Include bit map      : 0x1
      Include ext bit map  :
      Length: 256 bits
      Value : 0x::1
      Include affinity name : RED(0)
    Bandwidth: 40000 (CT0)
  path option 2, type explicit EP_NO_R9
    Path-option attribute: LSP_ATT_NO_R9
  path option 100, (LOCKDOWN) type dynamic
    Path-option attribute: LSP_ATT_ORANGE20

```

```

RP/0/0/CPU0:XRv12#show mpls traffic-eng tunnels 17 detail | begin Resv Info
Resv Info:
Record Route:

```

```
IPv4 132.3.9.9, flags 0x0
IPv4 132.3.4.3, flags 0x0
IPv4 132.3.4.4, flags 0x0
```

Fspec: avg rate=40000 kbits, burst=1000 bytes, peak rate=40000 kbits

RP/0/0/CPU0:XRv12#show rsvp interface

*: RDM: Default I/F B/W % : 75% [default] (max resv/bc0), 0% [default] (bc1)

Interface	MaxBW (bps)	MaxFlow (bps)	Allocated (bps)	MaxSub (bps)
Gi0/0/0/0.502	100M	100M	0 (0%)	0
Gi0/0/0/0.512	100M	100M	0 (0%)	0
Gi0/0/0/0.562	100M	100M	0 (0%)	0
Gi0/0/0/0.592	100M	100M	40M (40%)	0

31.1.3 Directing traffic into TE tunnels and tunnel stitching

So far, we have not transferred any real traffic inside of our TE tunnels other than OAM tests. Without TE, the routers typically use LDP for transport label bindings based on the BGP next-hops, PW destinations, or other destinations reachable through IGP. The entire purpose of TE is to modify this IGP-based behavior. For these examples, we will use CSR1 and XRv11 as endpoints and direct traffic between these PEs into TE tunnels using various methods. First, we see that they currently use LDP labels. CSR1 has a VPNv4 route (and label) for XRv13 within the VPN. Based on the IGP route to XRv11's loopback, we see three possible LDP labels that can be used.

CSR1#show bgp vpnv4 unicast vrf TE 13.13.13.13/32

BGP routing table entry for 132:1:13.13.13.13/32, version 8

Paths: (1 available, best #1, table TE)

Advertised to update-groups:

1

Refresh Epoch 1

13

```
11.11.11.11 (metric 30) (via default) from 11.11.11.11 (11.11.11.11)
Origin IGP, metric 0, localpref 100, valid, internal, best
Extended Community: RT:132:1
mpls labels in/out nodelabel/91011
rx pathid: 0, tx pathid: 0x0
```

CSR1#show ip route 11.11.11.11

Routing entry for 11.11.11.11/32

Known via "isis", distance 115, metric 30, type level-2

Redistributing via isis 132

Last update from 132.1.9.9 on GigabitEthernet2.519, 17:01:37 ago

Routing Descriptor Blocks:

```
132.1.9.9, from 11.11.11.11, 17:01:37 ago, via GigabitEthernet2.519
Route metric is 30, traffic share count is 1
* 132.1.6.6, from 11.11.11.11, 17:01:37 ago, via GigabitEthernet2.516
Route metric is 30, traffic share count is 1
132.1.3.3, from 11.11.11.11, 17:01:37 ago, via GigabitEthernet2.513
Route metric is 30, traffic share count is 1
```

```
CSR1#show mpls ldp bindings 11.11.11.11 32 neighbor 9.9.9.9
lib entry: 11.11.11.11/32, rev 45
remote binding: lsr: 9.9.9.9:0, label: 9009
```

```
CSR1#show mpls ldp bindings 11.11.11.11 32 neighbor 6.6.6.6
lib entry: 11.11.11.11/32, rev 45
remote binding: lsr: 6.6.6.6:0, label: 6009
```

```
CSR1#show mpls ldp bindings 11.11.11.11 32 neighbor 3.3.3.3
lib entry: 11.11.11.11/32, rev 45
remote binding: lsr: 3.3.3.3:0, label: 3009
```

Although somewhat unrelated to this exact task, it can be confusing to know which ECMP path will be taken for VPN traffic from CSR7 to XRv13. The VPN CEF table points to CSR9 while the global table between BGP next-hops points to CSR3. The latter is actually correct; since the transport label lookup happens in the global table, the global FIB is consulted. We confirm this with a traceroute inside the VPN.

```
CSR1#show ip cef vrf TE exact-route 7.7.7.7 13.13.13.13
7.7.7.7 -> 13.13.13.13 => label 91011 label 9009 TAG adj out of
GigabitEthernet2.519, addr 132.1.9.9
```

```
CSR1#show ip cef exact-route 1.1.1.1 11.11.11.11
1.1.1.1 -> 11.11.11.11 => label 3009 TAG adj out of GigabitEthernet2.513,
addr 132.1.3.3
```

```
CSR7#traceroute 13.13.13.13 source 7.7.7.7
Type escape sequence to abort.
Tracing the route to 13.13.13.13
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.7.1 0 msec 1 msec 1 msec
 2 132.1.3.3 [MPLS: Labels 3009/91011 Exp 0] 98 msec 52 msec 37 msec
 3 132.3.10.10 [MPLS: Labels 10009/91011 Exp 0] 32 msec 42 msec 50 msec
 4 132.10.11.11 [MPLS: Label 91011 Exp 0] 168 msec 64 msec 42 msec
 5 10.11.13.13 17 msec * 19 msec
```

The first and most straightforward way to get this flow into the tunnel is using static routes. We no longer want to use LDP labels for the topmost label, which means we have to learn the route to the BGP next-hop via a TE tunnel. The router knows that it should use the TE label when the routing converges in this way. We configure a basic tunnel that can take any path not transiting CSR9. We also configure a static route to direct all traffic to XRv11's loopback into the TE tunnel. This method includes ALL IPv4 traffic between the two, including existing BGP sessions, pings, etc. If the tunnel is down for whatever reason, regular routing rules apply. This isn't a special case but a default behavior of the static route. If the specified outgoing interface is down, the static route is removed from the RIB unless marked with

the “permanent” option. The route with the next lowest AD out of a functional interface is used, so IGP routes + LDP labels can be a fallback for failed or shutdown TE tunnels.

```
! CSR1
interface Tunnel18
  description STEER WITH STATIC
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 explicit name EP_NO_R9

ip route 11.11.11.11 255.255.255.255 Tunnel18
```

We verify that the tunnel is up (path is CSR6 > CSR10 > XRv11) and that the static route was installed successfully.

```
CSR1#show mpls traffic-eng tunnels tunnel 18 | section RSVP_Path
  RSVP Path Info:
    My Address: 132.1.6.1
    Explicit Route: 132.1.6.6 132.6.10.10 132.10.11.11 11.11.11.11
    Record Route: NONE
    Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits

CSR1#show ip route 11.11.11.11
Routing entry for 11.11.11.11/32
  Known via "static", distance 1, metric 0 (connected)
  Routing Descriptor Blocks:
    * directly connected, via Tunnel18
      Route metric is 0, traffic share count is 1
```

Next, we can find the new outgoing label by checking the RSVP RESV details for this specific tunnel. Using traceroute inside the VPN on CSR7, we can see that the traffic is now being transported inside the TE tunnel. Even outside the VPN from CSR1's global table, the traffic is inside the tunnel, as mentioned earlier.

```
CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 18 |
include Label
  Label: 6011 (outgoing)

CSR7#traceroute 13.13.13.13 source 7.7.7.7
Type escape sequence to abort.
Tracing the route to 13.13.13.13
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.7.1 1 msec 1 msec 0 msec
 2 132.1.6.6 [MPLS: Labels 6011/91011 Exp 0] 29 msec 25 msec 25 msec
 3 132.6.10.10 [MPLS: Labels 10011/91011 Exp 0] 24 msec 24 msec 24 msec
```



```
4 132.10.11.11 [MPLS: Label 91011 Exp 0] 16 msec 105 msec 41 msec
5 10.11.13.13 28 msec * 74 msec
```

```
CSR1#traceroute 11.11.11.11 source 1.1.1.1
Type escape sequence to abort.
Tracing the route to 11.11.11.11
VRF info: (vrf in name/id, vrf out name/id)
 1 132.1.6.6 [MPLS: Label 6011 Exp 0] 5 msec 8 msec 8 msec
 2 132.6.10.10 [MPLS: Label 10011 Exp 0] 26 msec 25 msec 25 msec
 3 132.10.11.11 8 msec * 46 msec
```

The return traffic from XRv11 is taking an LDP path since there isn't a corresponding TE tunnel configured on XRv11. We will quickly verify this statement below. The route is learned via IGP and does not point to a TE tunnel. The router has LDP labels for both paths and selects CSR10's label of 10007 per the FIB. We verify it via several show commands and traceroute.

```
RP/0/0/CPU0:XRv11#show route ipv4 unicast 1.1.1.1
Routing entry for 1.1.1.1/32
  Known via "isis 132", distance 115, metric 30, type level-2
  Routing Descriptor Blocks
    132.10.11.10, from 1.1.1.1, via GigabitEthernet0/0/0/0.501
      Route metric is 30
    132.11.12.12, from 1.1.1.1, via GigabitEthernet0/0/0/0.512
      Route metric is 30
  No advertising protos.
```

```
RP/0/0/CPU0:XRv11#show mpls ldp bindings 1.1.1.1/32 neighbor 10.10.10.10
1.1.1.1/32, rev 33
  Local binding: label: 91004
  Remote bindings: (3 peers)
    Peer          Label
    -----
    10.10.10.10:0 10007
```

```
RP/0/0/CPU0:XRv11#show mpls ldp bindings 1.1.1.1/32 neighbor 12.12.12.12
1.1.1.1/32, rev 33
  Local binding: label: 91004
  Remote bindings: (3 peers)
    Peer          Label
    -----
    12.12.12.12:0 92006
```

```
RP/0/0/CPU0:XRv11#show cef exact-route 11.11.11.11 1.1.1.1
1.1.1.1/32, version 138, internal 0x1000001 0x0 (ptr 0xa140f6f4) [1], 0x0
(0xa13f48e4), 0xa28 (0xa193807c)
  local adjacency 132.10.11.10
  Prefix Len 32, traffic index 0, precedence n/a, priority 3
  via GigabitEthernet0/0/0/0.501
```

```
via 132.10.11.10, GigabitEthernet0/0/0/0.501, 9 dependencies, weight 0,
class 0 [flags 0x0]
  path-idx 0 NHID 0x0 [0xa0e8f34c 0x0]
  next hop 132.10.11.10
  local adjacency
  local label 91004 labels imposed {10007}
```

```
RP/0/0/CPU0:XRv11#traceroute 1.1.1.1 source 11.11.11.11
Type escape sequence to abort.
Tracing the route to 1.1.1.1
```

```
 1 132.10.11.10 [MPLS: Label 10007 Exp 0] 19 msec  9 msec  0 msec
 2 132.6.10.6 [MPLS: Label 6007 Exp 0] 9 msec  0 msec  0 msec
 3 132.1.6.1 0 msec  *  0 msec
```

We will quickly demonstrate TE with static routing on XRv11 as well. The constraints are similar except neither CSR9 nor CSR10 can be in the transit path.

```
! XRv11
explicit-path name PATH_NO_9_10
  index 10 exclude-address ipv4 unicast 9.9.9.9
  index 20 exclude-address ipv4 unicast 10.10.10.10

interface tunnel-te18
  description STEER WITH STATIC
  ipv4 unnumbered Loopback0
  logging events all
  destination 1.1.1.1
  affinity ignore
  path-option 10 explicit name PATH_NO_9_10

router static
  address-family ipv4 unicast
    1.1.1.1/32 tunnel-te18
```

We ensure the tunnel is operational (path is CSR5 > CSR 2 > CSR3 > CSR1) and verify the static route was installed.

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 18 role head | begin Path
info
```

```
Path info (IS-IS 132 level-2):
```

```
Node hop count: 4
Hop0: 132.5.11.5
Hop1: 132.2.5.2
Hop2: 132.2.3.2
Hop3: 132.2.3.3
Hop4: 132.1.3.1
Hop5: 1.1.1.1
```

```
RP/0/0/CPU0:XRv11#show route ipv4 unicast 1.1.1.1
Routing entry for 1.1.1.1/32
  Known via "static", distance 1, metric 0 (connected)
  Routing Descriptor Blocks
    directly connected, via tunnel-te18
      Route metric is 0
  No advertising protos.
```

Last, we verify the outgoing label via the RSVP RESV details and test the path through traceroute. As we will see in the MVPN and per-VRF TE sections, this approach can be problematic for multicast transport due to RPF inconsistencies. Because TE tunnels cannot run PIM (doesn't make sense), RPF is now broken because potential MDT endpoints.

```
RP/0/0/CPU0:XRv11#show rsvp reservation session-type lsp-p2p destination
1.1.1.1 detail | include Label
Labels: Outgoing downstream: 5012.
```

```
RP/0/0/CPU0:XRv13#traceroute 7.7.7.7 source 13.13.13.13
Type escape sequence to abort.
Tracing the route to 7.7.7.7
```

```
 1  10.11.13.11 0 msec  0 msec  0 msec
 2  132.5.11.5 [MPLS: Labels 5012/1009 Exp 0] 9 msec  9 msec  9 msec
 3  132.2.5.2 [MPLS: Labels 2010/1009 Exp 0] 9 msec  119 msec  39 msec
 4  132.2.3.3 [MPLS: Labels 3010/1009 Exp 0] 29 msec  39 msec  29 msec
 5  10.1.7.1 [MPLS: Label 1009 Exp 0] 39 msec  29 msec  19 msec
 6  10.1.7.7 29 msec  *  9 msec
```

Another option is using "auto-route". There are two main flavors: announce and destination. Destination is only supported on IOS and XE platforms and is almost identical to the static routing approach except is more dynamic. It automatically programs a static route for the tunnel destination out of the tunnel upon which it was configured [ID 19]. This can increase scalability because static routes don't need to be manually added. Changing the tunnel destination will automatically update the static route, also. Using an identical tunnel we just did, we will apply auto-route destination to achieve the same effect. Regarding MVPN RPF, this introduces the same issues as static routing (discussed in detail in other sections).

```
! CSR1
interface Tunnel19
  description STEER WITH AUTOROUTE DESTINATION
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng autoroute destination
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
```

```
tunnel mpls traffic-eng path-option 10 explicit name EP_NO_R9
```

We can verify the path by checking the RSVP PATH message's outgoing ERO. We also have an automatic static route for the tunnel's destination, which works well for PE-PE tunnels.

```
CSR1#show ip rsvp sender detail filter session-type 7 tunnel-id 19 | section outgoing
```

ERO: (outgoing)

```
132.1.6.6 (Strict IPv4 Prefix, 8 bytes, /32)
132.6.10.10 (Strict IPv4 Prefix, 8 bytes, /32)
132.10.11.11 (Strict IPv4 Prefix, 8 bytes, /32)
11.11.11.11 (Strict IPv4 Prefix, 8 bytes, /32)
```

```
CSR1#show ip route 11.11.11.11
```

Routing entry for 11.11.11.11/32

Known via "static", distance 1, metric 0 (connected)

Routing Descriptor Blocks:

* directly connected, via Tunnel19

Route metric is 0, traffic share count is 1

We can check the outgoing label by looking at the LFIB details for the destination prefix, and then verify it with traceroute. Within the VPN, we can verify the internal FIB details to see the TE label at the top of the stack as well.

```
CSR1#show mpls forwarding-table 11.11.11.11 detail
```

Local	Outgoing	Prefix	Bytes	Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched		interface	
1011	Pop Label	11.11.11.11/32	0		Tu19	point2point
MAC/Encaps=18/22, MRU=1500, Label Stack{6010}, via Gi2.516						
000C2993FE00000C29FBA33981000DBC8847 0177A000						
No output feature configured						

```
CSR1#traceroute 11.11.11.11 source 1.1.1.1
```

Type escape sequence to abort.

Tracing the route to 11.11.11.11

VRF info: (vrf in name/id, vrf out name/id)

```
1 132.1.6.6 [MPLS: Label 6010 Exp 0] 15 msec 8 msec 8 msec
2 132.6.10.10 [MPLS: Label 10010 Exp 0] 26 msec 25 msec 17 msec
3 132.10.11.11 20 msec * 11 msec
```

```
CSR1#show ip cef vrf TE 13.13.13.13 internal | begin output_chain
```

output chain:

label 91011

label implicit-null

TAG midchain out of Tunnel19 7F021339A698

label 6010

TAG adj out of GigabitEthernet2.516, addr 132.1.6.6 7F020E3BC130

As mentioned earlier, auto-route destination is not supported in XRv 5.3.0, and this is probably XRv specific.

```
! XRv11
interface tunnel-te88
  autoroute destination
!!% The requested operation is not supported: This feature is not supported
in this release.
```

The other option for auto-route creates an IGP route through the TE tunnel. This seems highly suspect since link-state protocols do not advertise routes per se, they advertise topology information. Routers collect this information, and assuming everyone has the same view, can guarantee loop-free paths through the topological graph. This feature does not make any changes to the underlying OSPF LSDB or IS-IS LSPDB, but simply announces the destination to the head-end router as being reachable through the tunnel. Even though it looks like an IGP route, the TE label is used since the outgoing interface is a TE tunnel. The cost of the route learned via the TE tunnel is the same as the shortest IGP path between the endpoints. We test this using the same type of tunnel as before [ID 20].

```
! CSR1
interface Tunnel20
  description STEER WITH AUTOROUTE ANNOUNCE
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng autoroute destination
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 explicit name EP_NO_R9
```

We can verify the tunnel is up very quickly by checking the outgoing label. Seeing that it is 6011, we assume the path transits CSR6. When we check the RIB, we see the route is IS-IS but recurses out of the TE tunnel. In the context of label lookups, this isn't an IGP + LDP binding, even though the route is IGP-installed. Only the RSVP label is used in this particular example.

```
CSR1#show mpls traffic-eng tunnels tunnel 20 | include Label
InLabel   : -
OutLabel  : GigabitEthernet2.516, 6011
```

```
CSR1#show ip route 11.11.11.11
Routing entry for 11.11.11.11/32
  Known via "isis", distance 115, metric 30, type level-2
  Redistributing via isis 132
  Last update from 11.11.11.11 on Tunnel20, 00:00:01 ago
  Routing Descriptor Blocks:
  * 11.11.11.11, from 11.11.11.11, 00:00:01 ago, via Tunnel20
    Route metric is 30, traffic share count is 1
```

A quick look at CSR1's IS-IS LSP shows no direct link to XRv11, indicating the IS-IS LSPDB did not change. However, when we check the IS-IS topology, the protocol is aware that reachability to XRv11 is achieved through a new best path, which is the TE tunnel. IS-IS does track all the TE tunnels, since it carries all of the TE attributes. The two processes (TE and IS-IS/OSPF) communicate closely within software.

```
CSR1#show isis database level-2 CSR1.00-00 detail
Tag 132:
IS-IS Level-2 LSP CSR1.00-00
LSPID                LSP Seq Num    LSP Checksum  LSP Holdtime  ATT/P/OL
CSR1.00-00            * 0x0000010C   0x18B6        996           0/0/0
  Auth:                Length: 17
  Area Address: 00
  NLPID:               0xCC 0x8E
  Topology:            IPv4 (0x0)
                     IPv6 (0x2)
  Router ID:          1.1.1.1
  Hostname: CSR1
  Metric: 10           IS (MT-IPv6) CSR4.00
  Metric: 10           IS (MT-IPv6) CSR6.00
  Metric: 10           IS (MT-IPv6) CSR8.00
  Metric: 10           IS (MT-IPv6) CSR3.00
  Metric: 10           IS (MT-IPv6) CSR9.00
  Metric: 10           IS-Extended CSR4.00
  Metric: 10           IS-Extended CSR8.00
  Metric: 10           IS-Extended CSR6.00
  Metric: 10           IS-Extended CSR3.00
  Metric: 10           IS-Extended CSR9.00
  IP Address:          1.1.1.1
  Metric: 0            IP 1.1.1.1/32
  IPv6 Address:        ::1:1:1:1
  Metric: 0            IPv6 (MT-IPv6) ::1:1:1:1/128
```

```
CSR1#show isis topology level-2 XRv11
Tag 132:
IS-IS 0 level-2 path to XRv11
System Id      Metric    Next-Hop      Interface    SNPA
XRv11          30      XRv11         Tu20         *MPLS TE-Tunnel
```

```
CSR1#show isis mpls traffic-eng tunnel
System Id      Tunnel Name    BW Metric    Nexthop      Metric    Mode
XRv11.00      Tunnel20      0           11.11.11.11
```

A quick traceroute test from CSR1 shows that the feature is working based on the outgoing label. The FIB internal details also show this label bound to the destination prefix 11.11.11.11/32.

```
CSR1#traceroute 11.11.11.11 source 1.1.1.1
Type escape sequence to abort.
Tracing the route to 11.11.11.11
```

```
VRF info: (vrf in name/id, vrf out name/id)
 1 132.1.6.6 [MPLS: Label 6011 Exp 0] 12 msec 11 msec 8 msec
 2 132.6.10.10 [MPLS: Label 10011 Exp 0] 27 msec 17 msec 25 msec
 3 132.10.11.11 15 msec * 20 msec
```

```
CSR1#show ip cef 11.11.11.11 internal | begin output_chain
output chain:
  IP midchain out of Tunnel20 7F021339A698
  label 6011
  TAG adj out of GigabitEthernet2.516, addr 132.1.6.6 7F020E3BC130
```

A minor adjustment to this feature allows us to change the IGP metric for this auto-route. We can specify an absolute value instead of the IS-IS calculated one, which is based on the non-TE best path. It was 30 before, but we can change the auto-route to be 31, as an example. Now, the best-path is no longer through the TE tunnel but is via the ordinary paths. This can be used for advanced failover scenarios. For example, if the regular path IGP metric is 30 or less, that will be preferred. If the IGP cost is too high (greater than 30), we decide we want to use the TE tunnel and apply bandwidth reservations to compensate for the less optimal routing paths.

```
! CSR1
interface Tunnel20
 tunnel mpls traffic-eng autoroute metric absolute 31

CSR1#show ip route 11.11.11.11
Routing entry for 11.11.11.11/32
  Known via "isis", distance 115, metric 30, type level-2
  Redistributing via isis 132
  Last update from 132.1.9.9 on GigabitEthernet2.519, 00:00:04 ago
  Routing Descriptor Blocks:
    132.1.9.9, from 11.11.11.11, 00:00:04 ago, via GigabitEthernet2.519
      Route metric is 30, traffic share count is 1
    132.1.6.6, from 11.11.11.11, 00:00:04 ago, via GigabitEthernet2.516
      Route metric is 30, traffic share count is 1
  * 132.1.3.3, from 11.11.11.11, 00:00:04 ago, via GigabitEthernet2.513
      Route metric is 30, traffic share count is 1

CSR1#show mpls traffic-eng autoroute 11.11.11.11
MPLS TE autorouting enabled
  destination 0000.0000.0011.00, area isis level-2, has 1 tunnels
    Tunnel20 (load balancing metric 0, nexthop 11.11.11.11, absolute
metric 31)
      (flags: Announce)
```

We can also use a relative modifier to scale the metric up or down. In this case, the IGP-computed metric can be scaled up or down within a range of +/- 10. This is mostly useful for when there are multiple TE tunnels going to the same destination, both using autoroute, and one should be preferred over the other. Another tunnel [ID 21] has been added to prove this point, and the original tunnel has

the absolute metric removed. This new tunnel has a lower IGP metric by 1, and is thus preferred. This is another dimension to flexibility in TE in addition to the path-options within a given tunnel. This provides prioritization between tunnels as well.

```
! CSR1
interface Tunnel20
  no tunnel mpls traffic-eng autoroute metric absolute 31

interface Tunnel21
  description STEER WITH AUTOROUTE ANNOUNCE -1 METRIC
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng autoroute announce
  tunnel mpls traffic-eng autoroute metric relative -1
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 explicit name EP_NO_R9

CSR1#show ip route 11.11.11.11
Routing entry for 11.11.11.11/32
  Known via "isis", distance 115, metric 29, type level-2
  Redistributing via isis 132
  Last update from 11.11.11.11 on Tunnel21, 00:00:47 ago
  Routing Descriptor Blocks:
  * 11.11.11.11, from 11.11.11.11, 00:00:47 ago, via Tunnel21
    Route metric is 29, traffic share count is 1
```

We can also see the paths side-by-side as tracked by the auto-route feature. This can help with troubleshooting when trying to determine which tunnel should be used.

```
CSR1#show mpls traffic-eng autoroute 11.11.11.11
MPLS TE autorouting enabled
  destination 0000.0000.0011.00, area isis level-2, has 2 tunnels
    Tunnel20      (load balancing metric 0, nexthop 11.11.11.11)
                  (flags: Announce)
    Tunnel21      (load balancing metric 0, nexthop 11.11.11.11, relative
metric -1)
                  (flags: Announce)
```

We will quickly test autoroute announce with metric manipulations on XR as well [ID 20, 21]. First, we create a single tunnel using the same constraints as earlier with an exact metric of 28. We check the RSVP PATH message's outgoing ERO to see the path; CSR5 is the next-hop. The LFIB reveals the TE label as 5012 and we can verify that the remote prefix 1.1.1.1/32 is using this label.

```
! XRv11
interface tunnel-te20
  description STEER WITH AUTOROUTE ANNOUNCE
```



```

ipv4 unnumbered Loopback0
logging events all
autoroute announce
  metric absolute 28
destination 1.1.1.1
affinity ignore
path-option 10 explicit name PATH_NO_9_10

```

```

RP/0/0/CPU0:XRv11#show rsvp sender detail session-type lsp-p2p destination
1.1.1.1 | begin Outgoing

```

Explicit Route (Outgoing):

```

  Strict, 132.5.11.5/32
  Strict, 132.2.5.2/32
  Strict, 132.2.3.2/32
  Strict, 132.2.3.3/32
  Strict, 132.1.3.1/32
  Strict, 1.1.1.1/32

```

```

RP/0/0/CPU0:XRv11#show route ipv4 unicast 1.1.1.1
Routing entry for 1.1.1.1/32

```

Known via "isis 132", distance 115, metric 28, type level-2

Routing Descriptor Blocks

1.1.1.1, from 1.1.1.1, **via tunnel-te20**

Route metric is 28

No advertising protos.

```

RP/0/0/CPU0:XRv11#show mpls forwarding tunnels 20

```

Tunnel Name	Outgoing Label	Outgoing Interface	Next Hop	Bytes Switched
tt20	5012	Gi0/0/0/0.551	132.5.11.5	326

```

RP/0/0/CPU0:XRv11#show mpls forwarding prefix 1.1.1.1/32 detail

```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
91004	Pop	1.1.1.1/32	tt20	1.1.1.1	505

Version: 145, Priority: 3
MAC/Encaps: 18/22, MTU: 1500
Label Stack (Top -> Bottom): { 5012 Imp-Null }
NHID: 0
Packets Switched: 7

Next, we create the second tunnel with a relative metric of -4 [ID 21]. We check the tunnel path and see that it also uses CSR5. This brings the metric to 26, which is better than the previous tunnel, so the RIB outgoing interface and TE label binding should change as well.

```

! XRv11

```

```

interface tunnel-te21
  description STEER WITH AUTOROUTE ANNOUNCE -4 METRIC
  ipv4 unnumbered Loopback0
  logging events all
  autoroute announce
    metric relative -4
  destination 1.1.1.1
  affinity ignore
  path-option 10 explicit name PATH_NO_9_10

```

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 21 | begin Path info
```

```
Path info (IS-IS 132 level-2):
```

```

Node hop count: 4
Hop0: 132.5.11.5
Hop1: 132.2.5.2
Hop2: 132.2.3.2
Hop3: 132.2.3.3
Hop4: 132.1.3.1
Hop5: 1.1.1.1

```

```
RP/0/0/CPU0:XRv11#show route ipv4 unicast 1.1.1.1
```

```
Routing entry for 1.1.1.1/32
```

```
Known via "isis 132", distance 115, metric 26, type level-2
```

```
Routing Descriptor Blocks
```

```
1.1.1.1, from 1.1.1.1, via tunnel-te21
```

```
Route metric is 26
```

```
No advertising protos.
```

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 21 detail | include Label
```

```
Outgoing Interface: GigabitEthernet0/0/0/0.551, Outgoing Label: 5013
```

```
RP/0/0/CPU0:XRv11#show mpls forwarding prefix 1.1.1.1/32 detail | include
```

```
Label
```

```
Label Label or ID Interface Switched
```

```
Label Stack (Top -> Bottom): { 5013 Imp-Null }
```

The auto-route information shows both paths along with their metrics. The IS-IS topology can then show which tunnel is used when a combination of absolute and relative metrics are used; it isn't immediately obvious which one is better using this output.

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng autoroute 1.1.1.1
```

```
Destination 1.1.1.1 has 2 tunnels in IS-IS 132 level 2
```

```
tunnel-te20 (traffic share 0, nexthop 1.1.1.1, absolute metric 28)
```

```
(IS-IS 132 level-2, IPV4 Unicast)
```

```
Signalled-Name: XRv11_t20
```

```
tunnel-te21 (traffic share 0, nexthop 1.1.1.1, relative metric -4)
```

```
(IS-IS 132 level-2, IPV4 Unicast)
```

```
Signalled-Name: XRv11_t21
```

```
RP/0/0/CPU0:XRv11#show isis topology level 2 systemid 0000.0000.0001
IS-IS 132 paths to IPv4 Unicast (Level-2) routers
System Id      Metric  Next-Hop      Interface      SNPA
CSR1           26      CSR1          tt21           *PtoP*
```

Another method is policy routing. This is generally the least desirable option, but can give additional flexibility as to what traffic goes into the TE tunnel. PBR does not work for VPN traffic and can only be used within a single routing instance, so for L3VPN, this is not an option. For global MPLS transport, such as 6PE, this can work. We use CSR4 as a test client briefly by forcing traffic to 11.11.11.11 to CSR1, and also disabling LDP on the interface to CSR1. This allows the traffic to XRv11 to be raw IP so CSR1 can steer it into the TE tunnel with PBR. CSR1 has a normal tunnel configuration with nothing fancy since the PBR is applied separately [ID 22]. Access-list based forwarding (ABF) is the XR equivalent and is not supported for this use.

```
! CSR4
interface GigabitEthernet2.514
 no mpls ldp igp autoconfig

ip route 11.11.11.11 255.255.255.255 132.1.4.1

! CSR1
interface Tunnel22
 description STEER WITH PBR
 ip unnumbered Loopback0
 tunnel mode mpls traffic-eng
 tunnel destination 11.11.11.11
 tunnel mpls traffic-eng affinity 0x0 mask 0x0
 tunnel mpls traffic-eng path-option 10 explicit name EP_NO_R9

ip access-list extended ACL_TO_XRV11
 permit ip any host 11.11.11.11

route-map PM_PBR_XRV11_TE permit 10
 match ip address ACL_TO_XRV11
 set interface Tunnel22

interface GigabitEthernet2.514
 ip policy route-map PM_PBR_XRV11_TE
```

We quick verify the tunnel path and label. The next-hop is CSR6 using label 6010.

```
CSR1#show mpls traffic-eng tunnels tunnel 22 | section Label|RSVP_Path
InLabel   : -
OutLabel  : GigabitEthernet2.516, 6010
RSVP Path Info:
My Address: 132.1.6.1
```

```
Explicit Route: 132.1.6.6 132.6.10.10 132.10.11.11 11.11.11.11
Record Route: NONE
Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits
```

When CSR4 sends traffic towards XRv11, no label is applied since LDP is disabled on that interface and the route is static (which counts as an IGP route from the perspective of using LDP labels). Using traceroute, we can confirm this traffic is entering the tunnel. The PBR counters on CSR1 should increment every time a packet enters the tunnel via PBR.

```
CSR4#show mpls interfaces gigabitEthernet 2.514
Interface          IP          Tunnel  BGP Static Operational
GigabitEthernet2.514 No          Yes     No  No      Yes
```

```
CSR4#show ip cef 11.11.11.11
11.11.11.11/32
  nexthop 132.1.4.1 GigabitEthernet2.514
```

```
CSR4#traceroute 11.11.11.11 source loopback0
Type escape sequence to abort.
Tracing the route to 11.11.11.11
VRF info: (vrf in name/id, vrf out name/id)
 1 132.1.4.1 1 msec 0 msec 0 msec
 2 132.1.6.6 [MPLS: Label 6010 Exp 0] 11 msec 8 msec 8 msec
 3 132.6.10.10 [MPLS: Label 10010 Exp 0] 18 msec 16 msec 17 msec
 4 132.10.11.11 15 msec * 10 msec
```

```
route-map PM_PBR_XRV11_TE, permit, sequence 10
Match clauses:
  ip address (access-lists): ACL_TO_XRV11
Set clauses:
  interface Tunnel22
Policy routing matches: 42 packets, 2947 bytes
```

Last, we clean up CSR4 so it is functioning as an LSR again (cleanup configuration not shown).

```
CSR4#show ip cef 11.11.11.11
11.11.11.11/32
  nexthop 132.3.4.3 GigabitEthernet2.534 label 3009
  nexthop 132.4.9.9 GigabitEthernet2.549 label 9009
```

```
CSR4#show mpls interfaces gigabitEthernet 2.514
Interface          IP          Tunnel  BGP Static Operational
GigabitEthernet2.514 Yes (ldp)    Yes     No  No      Yes
```

Another option for steering traffic into the tunnel is using the CEF forwarding adjacency. Earlier, we saw that the auto-route options did not make any underlying changes to the IGP link-state topology, but rather changed the local routing table. In contrast, this alternative technique does actually create a

point-to-point link between the tunnel head and tail as part of the IGP topology. In order for this to work, the TE tunnel must be bidirectionally configured with forwarding-adjacency. We can also directly adjust the IGP metric on these interfaces to ensure there are no unintended routing issues; this feature is seen by all nodes in the OSPF area or IS-IS level, so set the cost appropriately. The hold-time value causes the tunnel to wait 10 seconds before informing the network of a down event. This ensures that brief tunnel flaps do not reflood OSPF/IS-IS topology information.

```
! CSR1
interface Tunnel24
  description STEER WITH FORWARDING ADJ
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng forwarding-adjacency holdtime 10000
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 explicit name EP_NO_R9
  isis metric 25

! XRV11
interface tunnel-te24
  description STEER WITH FORWARDING ADJ
  ipv4 unnumbered Loopback0
  logging events all
  destination 1.1.1.1
  forwarding-adjacency
    holdtime 10000
  affinity ignore
  path-option 10 explicit name PATH_NO_9_10

router isis 132
  interface tunnel-te24
    address-family ipv4 unicast
    metric 25

CSR1#show mpls traffic-eng tunnels tunnel 24 | section Status|Config
Status:
  Admin: up          Oper: up          Path: valid          Signalling: connected
  path option 10, type explicit EP_NO_R9 (Basis for Setup, path weight 25)

Config Parameters:
  Bandwidth: 0          kbps (Global)  Priority: 7 7        Affinity: 0x0/0x0
  Metric Type: TE (default)
  AutoRoute: disabled LockDown: disabled Loadshare: 0 [0] bw-based
  Forwarding adjacency: holdtime 10000 ms
  auto-bw: disabled
```

With the tunnel up, we can begin the detailed verification process. CSR1 accounts for this tunnel as a forwarding-adjacency and has a special show command for it.

```

CSR1#show mpls traffic-eng forwarding-adjacency 11.11.11.11
destination 0000.0000.0011.00, area isis level-2, has 1 tunnels
Tunnel24 (load balancing metric 0, nexthop 11.11.11.11)
(flags: Forward-Adjacency holdtime 10000)

```

Checking the IS-IS database, we can see a new link to XRv11 was installed. There isn't an IS-IS neighbor (no hello packets, etc), but the link exists in the topology. The metric of this P2P link is 25 as configured on the tunnel interface.

```

CSR1#show isis database level-2 CSR1.00-00 detail | include IS-Extend
Metric: 10          IS-Extended CSR4.00
Metric: 10          IS-Extended CSR6.00
Metric: 10          IS-Extended CSR8.00
Metric: 25          IS-Extended XRv11.00
Metric: 10          IS-Extended CSR3.00
Metric: 10          IS-Extended CSR9.00

```

```

CSR1#show isis neighbors

```

Tag 132:

System Id	Type	Interface	IP Address	State	Holdtime	Circuit Id
CSR3	L2	Gi2.513	132.1.3.3	UP	24	00
CSR4	L2	Gi2.514	132.1.4.4	UP	25	00
CSR6	L2	Gi2.516	132.1.6.6	UP	22	00
CSR8	L2	Gi2.518	132.1.8.8	UP	27	00
CSR9	L2	Gi2.519	132.1.9.9	UP	26	00

Some additional, but less useful, IS-IS show commands are below. We can see the TE tunnels and topology information in summary form as viewed by IS-IS.

```

CSR1#show isis mpls traffic-eng tunnel | begin Forward
Forwarding-adjacencies

```

System Id	Tunnel Name	BW	Metric	Nexthop	Metric	Type
XRv11.00	Tunnel24	0		11.11.11.11	25	L2

```

CSR1#show isis topology level-2 00.0000.0000.0011.00

```

Tag 132:

IS-IS 0 level-2 path to XRv11

System Id	Metric	Next-Hop	Interface	SNPA
XRv11	25	XRv11	Tu24	*MPLS TE-Tunnel

Most importantly, we verify that the tunnel endpoint is reachable through the TE tunnel. If the metric was defaulted to 10, then many other endpoints would be reachable through the tunnel also. For example, 10.10.10.10/32 would have a metric of 20 via this new TE tunnel and three other LDP paths, causing some awkward traffic patterns.

```

CSR1#show ip route 11.11.11.11
Routing entry for 11.11.11.11/32
  Known via "isis", distance 115, metric 25, type level-2
  Redistributing via isis 132
  Last update from 11.11.11.11 on Tunnel24, 00:14:54 ago
  Routing Descriptor Blocks:
    * 11.11.11.11, from 11.11.11.11, 00:14:54 ago, via Tunnel24
      Route metric is 25, traffic share count is 1

```

The verification on XR is very similar, with most of the commands being identical. The TE process is aware of the forwarding-adjacency as an announcement mechanism and IS-IS sees it as a topology link. The other show commands are included for completeness also.

```

RP/0/0/CPU0:XRv11#show mpls traffic-eng forwarding-adjacency 1.1.1.1
destination 1.1.1.1 has 1 tunnels
    tunnel-te24      (traffic share 0, next-hop 1.1.1.1)
                    (Adjacency Announced: yes, holdtime 10000)
                    (IS-IS 132 level-2, IPV4 Unicast)

```

```

RP/0/0/CPU0:XRv11#show isis database 0000.0000.0011.00 level 2 detail |
include IS-Extend
Metric: 10          IS-Extended CSR5.00
Metric: 10          IS-Extended CSR10.00
Metric: 10          IS-Extended XRv12.00
Metric: 25          IS-Extended CSR1.00
Metric: 10          MT (IPv6 Unicast) IS-Extended CSR5.00
Metric: 10          MT (IPv6 Unicast) IS-Extended CSR10.00
Metric: 10          MT (IPv6 Unicast) IS-Extended XRv12.00

```

```

RP/0/0/CPU0:XRv11#show isis mpls traffic-eng tunnel
IS-IS 132 Level-2 MPLS Traffic Engineering tunnels
System Id   Tunnel  Bandw  Nexthop  Metric  Mode      IPv4/IPv6  IPV4/IPv6  Chkpt ID
              FA      AA
CSR1        tt24    0      1.1.1.1  25      Fixed     En/Dis     Dis/Dis     00000000

```

```

RP/0/0/CPU0:XRv11#show isis topology level 2 systemid 0000.0000.0001
IS-IS 132 paths to IPv4 Unicast (Level-2) routers
System Id   Metric  Next-Hop      Interface      SNPA
CSR1        25      CSR1          tt24           *PtoP*

```

```

RP/0/0/CPU0:XRv11#show route ipv4 1.1.1.1
Routing entry for 1.1.1.1/32
  Known via "isis 132", distance 115, metric 25, type level-2
  Routing Descriptor Blocks
    1.1.1.1, from 1.1.1.1, via tunnel-te24
      Route metric is 25
  No advertising protos.

```

To ensure the feature is working, we quickly confirm the labels along the TE LSP from CSR1 to XRV11. The path goes CSR6 > XRV12 > XRV11. Traceroute confirms that this path is used for the L3VPN traffic as expected.

```
CSR1#show mpls traffic-eng tunnels tunnel 24 | include Label
  InLabel   : -
  OutLabel  : GigabitEthernet2.516, 6010
```

```
CSR6#show mpls forwarding-table labels 6010
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id  Switched     interface
6010       92011      1.1.1.1 24 [13]  34134        Gi2.562     132.6.12.12
```

```
RP/0/0/CPU0:XRV12#show mpls forwarding labels 92011
Local  Outgoing  Prefix      Outgoing  Next Hop      Bytes
Label  Label      or ID       Interface  Next Hop      Switched
-----
92011  Pop         24          Gi0/0/0/0.512 132.11.12.11  30974
```

```
CSR7#traceroute 13.13.13.13 source 7.7.7.7
Type escape sequence to abort.
Tracing the route to 13.13.13.13
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.7.1 1 msec 0 msec 1 msec
 2 132.1.6.6 [MPLS: Labels 6010/91012 Exp 0] 94 msec 31 msec 28 msec
 3 132.6.12.12 [MPLS: Labels 92011/91012 Exp 0] 31 msec 27 msec 31 msec
 4 132.11.12.11 [MPLS: Label 91012 Exp 0] 27 msec 25 msec 25 msec
 5 10.11.13.13 31 msec * 36 msec
```

Because the feature must be configured bidirectionally, we should verify it works both ways. XRV11's tunnel to CSR1 goes CSR5 > CSR2 > CSR3 > CSR1. I am not certain why this suboptimal path was chosen, but it might have been due to network instability or the 100kbps CSR1000v bandwidth limit that caused certain IGP adjacencies to be down at the moment this was configured. In any event, note that the tunnels don't have to follow the same path bidirectionally, just that any valid LSP exists. We trace the labels hop-by-hop, then verify with traceroute within the VPN.

```
RP/0/0/CPU0:XRV11#show rsvp reservation session-type lsp-p2p destination
1.1.1.1 detail | include Label
  Labels: Outgoing downstream: 5013.
```

```
CSR5#show mpls forwarding-table labels 5013
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id  Switched     interface
5013       2019      11.11.11.11 24 [7]  \
                    56471        Gi2.525     132.2.5.2
```

```
CSR2#show mpls forwarding-table labels 2019
```


Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Label	Outgoing interface	Next Hop
2019	3008	11.11.11.11 24 [7]	\	56663	Gi2.523	132.2.3.3

```
CSR3#show mpls forwarding-table labels 3008
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Label	Outgoing interface	Next Hop
3008	Pop Label	11.11.11.11 24 [7]	\	54258	Gi2.513	132.1.3.1

```
RP/0/0/CPU0:XRv13#traceroute 7.7.7.7 source 13.13.13.13
```

```
Type escape sequence to abort.
```

```
Tracing the route to 7.7.7.7
```

```

 1  10.11.13.11 0 msec  19 msec  0 msec
 2  132.5.11.5 [MPLS: Labels 5013/1009 Exp 0] 39 msec  29 msec  19 msec
 3  132.2.5.2 [MPLS: Labels 2019/1009 Exp 0] 19 msec  19 msec  19 msec
 4  132.2.3.3 [MPLS: Labels 3008/1009 Exp 0] 39 msec  39 msec  39 msec
 5  10.1.7.1 [MPLS: Label 1009 Exp 0] 19 msec  29 msec  19 msec
 6  10.1.7.7 29 msec  *  29 msec
```

Not all TE tunnels need to "home run" all the way from PE to PE. Tunnels can exist from PE-PE, PE-P, P-P, and P-PE if necessary. We can create more hierarchical TE architectures in this way, much like using H-VPLS with U-PE and N-PE pseudowires. In PE-PE and P-PE tunnels, all transport signaling is done by RSVP as the tunnel provides a new LSP; LDP is not necessary at all in this case. When doing PE-P or P-P options, we need some other mechanism to binds labels for the remote PE IP prefix. The tunnels don't terminate at the final destination, so the transport labels need to be allocated in other ways. In other words, we still need to resolve that FEC since the TE-tunnel does not go end-to-end. In the example below, a pair of TE tunnels are created. One goes from CSR1 to XRv12, and the other goes from XRv12 to XRv11. The ultimate goal is to get traffic from CSR to XRv11 using these two tunnels, "stitching" them together. The first tunnel takes an explicit path through CSR8 and CSR6, while the second takes any red path that is also not blue [ID 23].

```

! CSR1
ip explicit-path name PATH_1_8_6_12 enable
  next-address 8.8.8.8
  next-address 6.6.6.6
  next-address 12.12.12.12

interface Tunnel23
  description PE-P
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 12.12.12.12
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 explicit name PATH_1_8_6_12
```

```

! XRv12
interface tunnel-te23
  description P-PE
  ipv4 unnumbered Loopback0
  logging events all
  destination 11.11.11.11
  affinity include RED
  affinity exclude BLUE
  path-option 10 dynamic

```

We quickly verify the tunnel paths are correct and within constraints, and then use OAM to verify the data-plane [ID 23]. On XRv12, we should specify that we want to see the head-end of tunnel 23, since we used the same tunnel ID for both and XRv12 is the head for one and the tail for the other. XRv12 shows being both a head and tail within this "extended" LSP.

```

CSR1#show mpls traffic-eng tunnels tunnel 23 | section RSVP_Path
  RSVP Path Info:
    My Address: 132.1.8.1
    Explicit Route: 132.1.8.8 132.6.8.6 132.6.12.12 12.12.12.12
    Record Route: NONE
    Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits

```

```

CSR1#traceroute mpls traffic-eng tunnel 23
Tracing MPLS TE Label Switched Path on Tunnel23, timeout is 2 seconds
[snip]
Type escape sequence to abort.
 0 132.1.8.1 MRU 1500 [Labels: 8012 Exp: 0]
L 1 132.1.8.8 MRU 1500 [Labels: 6011 Exp: 0] 2 ms
L 2 132.6.8.6 MRU 1500 [Labels: implicit-null Exp: 0] 33 ms
! 3 132.6.12.12 35 ms

```

```

RP/0/0/CPU0:XRv12#show mpls traffic-eng tunnels 23 role head | begin Path
info
  Path info (IS-IS 132 level-2):
    Node hop count: 2
    Hop0: 132.10.12.10
    Hop1: 132.10.11.11
    Hop2: 11.11.11.11

```

```

RP/0/0/CPU0:XRv12#traceroute mpls traffic-eng tunnel-te 23
Tracing MPLS TE Label Switched Path on tunnel-te23, timeout is 2 seconds
[snip]
Type escape sequence to abort.
 0 132.10.12.12 MRU 1500 [Labels: 10011 Exp: 0]
L 1 132.10.12.10 MRU 1500 [Labels: implicit-null Exp: 0] 0 ms
! 2 132.10.11.11 1 ms

```

```

RP/0/0/CPU0:XRv12#show mpls traffic-eng tunnels brief up

```

TUNNEL NAME	DESTINATION	STATUS	STATE
tunnel-te23	11.11.11.11	up	up
PE-P	12.12.12.12	up	up

Each LSP appears to be fully functional. We will configure a static route to XRv11 on CSR1 so we can send VPN traffic into the tunnel. Auto-route destination would not be appropriate here, since the destination of CSR1's tunnel (XRv12) is not the address to which we are sending traffic (XRv11). Auto-route announce technically could work on CSR1, but we will use a static route for simplicity. We can use auto-route announce (or destination) on XRv12 for the second leg without causing any complexities. We quickly verify that each router is directing traffic to XRv11 out of the appropriate TE tunnel.

```
! CSR1
ip route 11.11.11.11 255.255.255.255 Tunnel23

! XRv12
interface tunnel-te23
  autoroute announce

CSR1#show ip route 11.11.11.11
Routing entry for 11.11.11.11/32
  Known via "static", distance 1, metric 0 (connected)
  Routing Descriptor Blocks:
  * directly connected, via Tunnel23
    Route metric is 0, traffic share count is 1

RP/0/0/CPU0:XRv12#show route ipv4 unicast 11.11.11.11
Routing entry for 11.11.11.11/32
  Known via "isis 132", distance 115, metric 10, type level-2
  Routing Descriptor Blocks
    11.11.11.11, from 11.11.11.11, via tunnel-te23
    Route metric is 10
  No advertising protos.
```

When we traceroute from CSR1 to XRv11 to verify the path, we that CSR6 is performing PHP of the TE label as expected, and the traffic is raw IP on the hop to XRv12. This works for traffic in the core since we can route IP or MPLS. As such, there is no breakage in the BGP sessions, etc. We see that this breaks the VPN service between CSR7 and XRv13.

```
CSR1#traceroute 11.11.11.11 source loopback0
Type escape sequence to abort.
Tracing the route to 11.11.11.11
VRF info: (vrf in name/id, vrf out name/id)
 1 132.1.8.8 [MPLS: Label 8012 Exp 0] 10 msec 8 msec 8 msec
 2 132.6.8.6 [MPLS: Label 6011 Exp 0] 8 msec 17 msec 17 msec
 3 132.6.12.12 16 msec 16 msec 8 msec
 4 132.10.12.10 [MPLS: Label 10011 Exp 0] 33 msec 25 msec 25 msec
 5 132.10.11.11 15 msec * 24 msec
```

```

CSR7#ping 13.13.13.13 source 7.7.7.7
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 13.13.13.13, timeout is 2 seconds:
Packet sent with a source address of 7.7.7.7
.....
Success rate is 0 percent (0/5)

```

Because CSR6 is performing PHP of the TE label, and the next label is the VPN label, this is being exposed to XRv12 when it was allocated by XRv11. We confirm this with MPLS packet debugging on XRv12; it receives a packet from CSR6 but cannot route it since there isn't an LFIB entry for label 91011, which is the VPN label allocated by XRv11 for XRv13's loopback.

```

CSR1#show bgp vpnv4 unicast vrf TE labels | include 13\.
13.13.13.13/32 11.11.11.11 nolabel/91011

! XRv12
netio[309]: mpls_switch-2: received an mpls packet on
GigabitEthernet0_0_0_0.562: proto=0, direction = ingress
netio[309]: mpls_switch-2: Setting default table-id
netio[309]: Flag set to 0x00000001
netio[309]: mpls_switch: GigabitEthernet0_0_0_0.562, mpls eos 1, ttl 252, len
126, inlabel 91011, tbl_id=0x0, vrf_id=0x0 in=0x600
netio[309]: [mpls_switch:2818] Pkt Drop: mpls_switch:
GigabitEthernet0_0_0_0.562, No LFIB entry found for in_label 91011

```

To solve this problem, we need to introduce another label into the stack. At imposition, CSR1 is currently accounting for a transport label to XRv12 and a VPN label for the final destination. There is no label representing the remote PE anywhere in the stack. Without this label, XRv12 will not know how to forward traffic along the second leg of the LSP. The solution is to enable LDP on the TE tunnel, which is basically telling the software to originate a targeted session to the tunnel destination. Rather than manually configure the destination to reciprocate, we can configure XRv12 to accept targeted sessions and respond accordingly. We can verify the LDP neighbors are up on both sides. CSR1 initiates the session and is considered active (TCP client), and XRv12 receives the session request and is considered passive (TCP server).

```

! CSR1
interface Tunnel23
 mpls ip

! XRv12
mpls ldp
 address-family ipv4
  discovery targeted-hello accept

CSR1#show mpls ldp neighbor 12.12.12.12

```

```

Peer LDP Ident: 12.12.12.12:0; Local LDP Ident 1.1.1.1:0
TCP connection: 12.12.12.12.20748 - 1.1.1.1.646
State: Oper; Msgs sent/rcvd: 19/18; Downstream
Up time: 00:00:46
LDP discovery sources:
  Targeted Hello 1.1.1.1 -> 12.12.12.12, active
Addresses bound to peer LDP Ident:
  12.12.12.12      132.9.12.12      132.6.12.12      132.11.12.12
  132.10.12.12

```

```

RP/0/0/CPU0:XRv12#show mpls ldp neighbor 1.1.1.1
Peer LDP Identifier: 1.1.1.1:0
TCP connection: 1.1.1.1:646 - 12.12.12.12:20748; MD5 on
Graceful Restart: No
Session Holdtime: 180 sec
State: Oper; Msgs sent/rcvd: 19/20; Downstream-Unsolicited
Up time: 00:00:59
LDP Discovery Sources:
  IPv4: (1)
    Targeted Hello (12.12.12.12 -> 1.1.1.1, passive)
  IPv6: (0)
Addresses bound to this peer:
  IPv4: (6)
    1.1.1.1      132.1.3.1      132.1.4.1      132.1.6.1
    132.1.8.1    132.1.9.1
  IPv6: (0)

```

With an LDP session established between CSR1 and XRv12, labels can be exchanged. CSR1 is already routing to 11.11.11.11/32 out of the TE tunnel, which means the TE label is imposed. We also learn label 92003, which is XRv12's local LDP label for XRv11's loopback. If packets arrive on any of XRv12's LDP enabled interfaces with this label, XRv12 will be able to route the packet according to its LFIB. We confirm that CSR1 imposes this label above the VPN label in its FIB, and confirm that XRv12 can route MPLS packets with label 92003 out of the proper TE tunnel (the second leg). On ingress, XRv12 "swaps" this LDP local label for the TE label learned from CSR10 via RSVP. I say "swaps" because the LFIB shows a pop followed by a push, which is effectively a swap.

```

CSR1#show mpls ldp bindings 11.11.11.11 32 neighbor 12.12.12.12
lib entry: 11.11.11.11/32, rev 45
  remote binding: lsr: 12.12.12.12:0, label: 92003

```

```

CSR1#show ip cef vrf TE 13.13.13.13
13.13.13.13/32
  nexthop 11.11.11.11 Tunnel23 label 92003 91011

```

```

RP/0/0/CPU0:XRv12#show mpls forwarding labels 92003 detail
Local   Outgoing   Prefix      Outgoing    Next Hop    Bytes
Label   Label       or ID       Interface
-----

```

```

92003 Pop          11.11.11.11/32      tt23          11.11.11.11      2268
Version: 273, Priority: 3
MAC/Encaps: 18/22, MTU: 1500
Label Stack (Top -> Bottom): { 10011 Imp-Null }
NHID: 0
Packets Switched: 33

```

We can run our traceroute tests again from the global and VPN tables to ensure the tunnel stitching was successful. The global table does not reveal any IP-only hops, and the VPN traffic has a fully MPLS-enabled path for transport. The TE label is topmost to steer traffic as needed, the LDP label is used at the stitch point to direct traffic towards the remote PE, and the VPN label remains unchanged at the bottom to represent the final destination. The second leg of the tunnel did not require tLDP since the tunnel terminates on the final PE.

```

CSR1#traceroute 11.11.11.11 source 1.1.1.1
Type escape sequence to abort.
Tracing the route to 11.11.11.11
VRF info: (vrf in name/id, vrf out name/id)
 1 132.1.8.8 [MPLS: Labels 8012/92003 Exp 0] 27 msec 33 msec 25 msec
 2 132.6.8.6 [MPLS: Labels 6011/92003 Exp 0] 15 msec 17 msec 17 msec
 3 132.6.12.12 [MPLS: Label 92003 Exp 0] 19 msec 54 msec 53 msec
 4 132.10.12.10 [MPLS: Label 10011 Exp 0] 42 msec 25 msec 25 msec
 5 132.10.11.11 15 msec * 63 msec

```

```

CSR7#traceroute 13.13.13.13 source 7.7.7.7
Type escape sequence to abort.
Tracing the route to 13.13.13.13
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.7.1 1 msec 1 msec 1 msec
 2 132.1.8.8 [MPLS: Labels 8012/92003/91011 Exp 0] 25 msec 25 msec 25 msec
 3 132.6.8.6 [MPLS: Labels 6011/92003/91011 Exp 0] 35 msec 33 msec 33 msec
 4 132.6.12.12 [MPLS: Labels 92003/91011 Exp 0] 33 msec 33 msec 33 msec
 5 132.10.12.10 [MPLS: Labels 10011/91011 Exp 0] 85 msec 58 msec 37 msec
 6 132.10.11.11 [MPLS: Label 91011 Exp 0] 35 msec 33 msec 33 msec
 7 10.11.13.13 33 msec * 24 msec

```

We also don't have to fully stitch tunnels together. Another design option would be to tunnel traffic to XRv12, then let XRv12 use the dynamic LDP path to deliver it to XRv12. We still need LDP on the PE-P tunnel for the reasons outlined above, but stitching TE tunnels together is not a requirement. We can use a combination of LDP and RSVP for the PE-to-PE transport. We can test this by shutting down tunnel 23 on XRv12 (not shown). Now, XRv12 learns the route via IGP out of a non-TE interface, so an ordinary LDP label can be used. Since XRv12 is the penultimate hop, it can pop label 92003 in ingress rather than swap it, as shown in the LFIB.

```

RP/0/0/CPU0:XRv12#show route ipv4 unicast 11.11.11.11
Routing entry for 11.11.11.11/32

```

Known via "isis 132", distance 115, metric 10, type level-2
 Routing Descriptor Blocks
 132.11.12.11, from 11.11.11.11, via GigabitEthernet0/0/0/0.512
 Route metric is 10
 No advertising protos.

```
RP/0/0/CPU0:XRv12#show mpls forwarding labels 92003 detail
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
92003	Pop	11.11.11.11/32	Gi0/0/0/0.512	132.11.12.11	4080

Version: 277, Priority: 3
 MAC/Encaps: 18/18, MTU: 1500
 Label Stack (Top -> Bottom): { Imp-Null }
 NHID: 0
 Packets Switched: 34

The LSP is shorter by one hop, but the logic is the same. We verify the path in the global and VPN tables, noting that the initial TE and LDP labels are identical as before until XRv12 is introduced. CSR1 isn't aware of any changes in the path since the path from XRv12 to XRv11 was always beyond CSR1's purview.

```
CSR1#traceroute 11.11.11.11 source 1.1.1.1
Type escape sequence to abort.
Tracing the route to 11.11.11.11
VRF info: (vrf in name/id, vrf out name/id)
 1 132.1.8.8 [MPLS: Labels 8012/92003 Exp 0] 36 msec 25 msec 25 msec
 2 132.6.8.6 [MPLS: Labels 6011/92003 Exp 0] 15 msec 17 msec 16 msec
 3 132.6.12.12 [MPLS: Label 92003 Exp 0] 16 msec 17 msec 16 msec
 4 132.11.12.11 16 msec * 18 msec
```

```
CSR7#traceroute 13.13.13.13 source 7.7.7.7
Type escape sequence to abort.
Tracing the route to 13.13.13.13
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.7.1 1 msec 1 msec 0 msec
 2 132.1.8.8 [MPLS: Labels 8012/92003/91011 Exp 0] 75 msec 50 msec 41 msec
 3 132.6.8.6 [MPLS: Labels 6011/92003/91011 Exp 0] 35 msec 33 msec 34 msec
 4 132.6.12.12 [MPLS: Labels 92003/91011 Exp 0] 33 msec 33 msec 34 msec
 5 132.11.12.11 [MPLS: Label 91011 Exp 0] 33 msec 42 msec 33 msec
 6 10.11.13.13 23 msec * 52 msec
```

31.2 TE Fast-ReRoute (FRR) and rapid provisioning

MPLS-TE is an excellent option for FRR. Other options, such as IP LFA, are much newer and not as widely supported. IP FRR is discussed in a dedicated sections.

31.2.1 Link (NHOP), Node (NNHOP), and Path protection – Manual

Using MPLS TE for fast reroute (FRR) is a very common way to achieve HA in MPLS networks. If a link or node within the path of a TE tunnel fails, a pre-sigaled backup path is available and traffic can be routed into the backup tunnel as soon as the failure is detected. TE FRR introduces some new terminology as well.

PLR: The point of local repair is the head-end of the backup path. When a failure is detected, it routes packets into the backup tunnel. This is much faster than signaling back to the original TE LSP headend to recalculate since the FRR occurs locally at the point of failure.

MP: The merge point is the tail-end of the backup path. When traffic arrives at the MP, it is routed along the normal TE path again, merging the traffic back on the original path. The PLR and MP work in concept to ensure that TE nodes upstream and downstream from the failure are minimally affected while FRR is active.

There are three main types of protection offered, in order from least to most desirable.

Path protection: Pre-signals a backup path from head to tail on a per path-option basis to be used in case of a failure anywhere along the path. Easy to configure and automatically protects all hops in the path without having to use PLRs. It is slowest to converge and scales the worst, and is not supported on XRv.

NHOP protection: Next-hop protection (link protection) protects the path to the next-hop by routing around failed links. This may scale better than NNHOP protection if many LSPs transit an LSR but have many different NNHOPs.

NNHOP protection: Next-next-hop protection (node protection) protects the path to the hop after the next-hop by routing around failed nodes. This offers better protection than NHOP but may not scale.

RFC 4090 details two ways of achieving FRR in the data-plane.

1. **N:1 or many-to-one or facility backup.** The PLR pushes a new FRR label (wraps the existing MPLS packet) as part of a new TE tunnel. The MP advertises a null label (implicit or explicit), effectively allowing it to see the original TE label so that the traffic can be merged easily since the incoming label would be the same as the main path. Many LSPs can be tunneled in the FRR tunnel in this way, providing more scalability at the low cost of 4 bytes extra encapsulation.

2. **1:1 or one-to-one or detour backup.** The PLR swaps the top label rather than pushes a new one. The MP advertises a normal label so the penultimate hop performs a label swap. This means that the label arriving at the MP will be different than the main path, and also different for each LSP. This scales poorly and isn't even supported in Cisco platforms.

We will configure a basic TE tunnel on CSR1 towards CSR8. CSR1 has two path options; the first is an explicit path through the network, the second is a secondary dynamic option requiring a green path and 2 Mbps. The tunnel uses auto-route destination to steer traffic into it, and also requests FRR protection.

```
! CSR1
mpls traffic-eng lsp attributes LSP_ATT_GREEN
  affinity 0x2 mask 0x2
  bandwidth 2000

ip explicit-path name PATH_1_3_10_6_8 enable
  next-address 3.3.3.3
  next-address 2.2.2.2
  next-address 10.10.10.10
  next-address 6.6.6.6
  next-address 8.8.8.8

interface Tunnel30
  description BASIC TE FRR
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 8.8.8.8
  tunnel mpls traffic-eng autoroute destination
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 explicit name PATH_1_3_10_6_8
  tunnel mpls traffic-eng path-option 20 dynamic attributes LSP_ATT_GREEN
  tunnel mpls traffic-eng fast-reroute
```

Debugging RSVP message dumps on CSR3, we can see this FRR request inside the RSVP PATH message from CSR1. The outgoing path and incoming RESV are not shown, but the final RESV sent to CSR1 contains an RRO with additional detail; it includes the labels at each hop in the TE tunnel. It also records where the backup tunnels are in the network, and right now there aren't any. Each entry says there is "no local protection" so FRR isn't completely configured yet.

```
! CSR3
Incoming Path:
  version:1 flags:0000 cksum:FD26 ttl:255 reserved:0 length:212
SESSION                               type 7 length 16:
  Tun Dest:   8.8.8.8  Tun ID: 30  Ext Tun ID: 1.1.1.1
HOP                                           type 1 length 12:
  Hop Addr: 132.1.3.1 LIH: 0x00000019
TIME_VALUES                               type 1 length 8 :
  Refresh Period (msec): 30000
EXPLICIT_ROUTE                             type 1 length 44:
  132.1.3.3 (Strict IPv4 Prefix, 8 bytes, /32)
  132.3.10.10 (Strict IPv4 Prefix, 8 bytes, /32)
  132.6.10.6 (Strict IPv4 Prefix, 8 bytes, /32)
  132.6.8.8 (Strict IPv4 Prefix, 8 bytes, /32)
```

```

8.8.8.8 (Strict IPv4 Prefix, 8 bytes, /32)
LABEL_REQUEST      type 1 length 8 :
  Layer 3 protocol ID: 2048
SESSION_ATTRIBUTE  type 7 length 20:
  Setup Prio: 7, Holding Prio: 7
  Flags: (0x7) Local Prot desired, Label Recording, SE Style
  Session Name: BASIC TE FRR
[snip]

```

Outgoing Resv:

```

version:1 flags:0000 cksum:A674 ttl:255 reserved:0 length:176
SESSION          type 7 length 16:
  Tun Dest:      8.8.8.8  Tun ID: 30  Ext Tun ID: 1.1.1.1
[snip]
RECORD_ROUTE     type 1 length 68:
  3.3.3.3/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3012
  10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 10011
  6.6.6.6/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 6011
  8.8.8.8/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3

```

We can verify that CSR1 has this information stored in its RESV message from CSR3. The format is almost the same so this is a quick way to verify the protection for a given TE tunnel. We can see that CSR1's FRR database has no protection listed for this tunnel. The RRO shown below is in almost identical format to the RSVP dump.

```
CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 30
```

Reservation:

```

Tun Dest:      8.8.8.8  Tun ID: 30  Ext Tun ID: 1.1.1.1
Tun Sender: 1.1.1.1  LSP ID: 3
Next Hop: 132.1.3.3 on GigabitEthernet2.513
Label: 3012 (outgoing)
Reservation Style is Shared-Explicit, QoS Service is Controlled-Load
Resv ID handle: 1D00040E.
Average Bitrate is 0 bits/sec, Maximum Burst is 1K bytes
Min Policed Unit: 0 bytes, Max Pkt Size: 1500 bytes

```

RRO:

```

3.3.3.3/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 3012
10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 10011
6.6.6.6/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 6011
8.8.8.8/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 3

```

```
Status:
Policy: Accepted. Policy source(s): MPLS/TE
```

```
CSR1#show mpls traffic-eng fast-reroute database
P2P Headend FRR information:
Protected tunnel      In-label Out intf/label      FRR intf/label      Status
-----
[no output]
[snip]
```

Before continuing, we can verify the TE tunnel is at least operational using MPLS OAM in the core and traceroute in the VPN. This shows us that auto-route is working properly and that there are no data-plane filters that might block traffic. It also allows us to verify the exact label values used to confirm it matches what the RESV RRO contained. The labels 3012, 10011, 6011, and implicit-null are shown in the RRO above.

```
CSR1#traceroute mpls traffic-eng tunnel 30
Tracing MPLS TE Label Switched Path on Tunnel30, timeout is 2 seconds
[snip]
```

```
Type escape sequence to abort.
 0 132.1.3.1 MRU 1500 [Labels: 3012 Exp: 0]
L 1 132.1.3.3 MRU 1500 [Labels: 10011 Exp: 0] 2 ms
L 2 132.3.10.10 MRU 1500 [Labels: 6011 Exp: 0] 29 ms
L 3 132.6.10.6 MRU 1500 [Labels: implicit-null Exp: 0] 26 ms
! 4 132.6.8.8 99 ms
```

```
CSR7#traceroute 14.14.14.14 source 7.7.7.7
Type escape sequence to abort.
Tracing the route to 14.14.14.14
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.7.1 0 msec 1 msec 1 msec
 2 132.1.3.3 [MPLS: Labels 3012/8009 Exp 0] 58 msec 31 msec 30 msec
 3 132.3.10.10 [MPLS: Labels 10011/8009 Exp 0] 41 msec 36 msec 40 msec
 4 132.6.10.6 [MPLS: Labels 6011/8009 Exp 0] 40 msec 31 msec 30 msec
 5 10.8.14.8 [MPLS: Label 8009 Exp 0] 16 msec 15 msec 15 msec
 6 10.8.14.14 24 msec * 69 msec
```

First we will look at NHOP protection. Let's assume that the link between CSR3 and CSR10 is known to be unreliable [ID 30]. We know our tunnel transits this link but want to be able to route around it quickly if/when it fails. We configure CSR3 to be our PLR and CSR10 to be our MP; a new tunnel is created from CSR3 to CSR10 that specifically avoids this link using explicit-path excluding the link addresses. We can configure the tunnel however we like, including bandwidth reservations and affinity. The tunnel also requires colors green, blue, and orange. We also specify the outgoing interface through which the original tunnel (the one we are backing up) is transiting. This tells RSVP that RESV messages arriving on that interface can use Tunnel 30 as a facility backup if FRR is requested.

```

! CSR3
ip explicit-path name PATH_AVOID_CSR3_CSR10 enable
  exclude-address 132.3.10.3
  exclude-address 132.3.10.10

interface Tunnel30
  description REPAIR PATH (PLR)
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 10.10.10.10
  tunnel mpls traffic-eng affinity 0xE mask 0xE
  tunnel mpls traffic-eng path-option 10 explicit name PATH_AVOID_CSR3_CSR10

interface GigabitEthernet2.530
  mpls traffic-eng backup-path Tunnel30

```

We quickly verify that the backup tunnel comes up and doesn't traverse the link directly to CSR10. The tunnel routes through CSR2 to reach CSR10 and avoids the protected link.

```

CSR3#show mpls traffic-eng tunnels tunnel 30 | section Status|RSVP_Path
Status:
  Admin: up          Oper: up      Path: valid      Signalling: connected
  path option 10, type explicit PATH_AVOID_CSR3_CSR10 (Basis for Setup,
path weight 20)

RSVP Path Info:
  My Address: 132.2.3.3
  Explicit Route: 132.2.3.2 132.2.10.10 10.10.10.10
  Record Route: NONE
  Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits

```

We check the RESV details on the tunnel head again and see that NHOP protection is now available to protect 3.3.3.3/32's "next-hop". This protects the link from CSR3 to CSR10 using a backup tunnel that avoids that link. We can see the flag is different for this hop (0x21 versus 0x20) compared to the ones that offer no protection. Notice that the output says "avail" but not "in use".

```

CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 30
Reservation:
  Tun Dest: 8.8.8.8 Tun ID: 30 Ext Tun ID: 1.1.1.1
  Tun Sender: 1.1.1.1 LSP ID: 19
  Next Hop: 132.1.3.3 on GigabitEthernet2.513
  Label: 3011 (outgoing)
  Reservation Style is Shared-Explicit, QoS Service is Controlled-Load
  Resv ID handle: 2500040E.
  Average Bitrate is 0 bits/sec, Maximum Burst is 1K bytes
  Min Policed Unit: 0 bytes, Max Pkt Size: 1500 bytes

```

```

RRO:
  3.3.3.3/32, Flags:0x21 (Local Prot Avail/to NHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3011
  10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 10010
  6.6.6.6/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 6011
  8.8.8.8/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3
Status:
Policy: Accepted. Policy source(s): MPLS/TE

```

On the PLR, we can verify the FRR configuration by checking the FRR database, which tracks the LSPs to be backed up along with the available FRR tunnels. Specifying the incoming label 3011, we can see how this specific LSP is being backed up. The summary view shows the incoming label, along with the ordinary LSP's outgoing interface and label. It also includes the FRR tunnel; because this entry is "ready" and not "active", we know FRR is not currently in effect.

```

CSR3#show mpls traffic-eng fast-reroute database labels 3011
[snip]
P2P LSP midpoint frr information:
LSP identifier          In-label Out intf/label    FRR intf/label    Status
-----
1.1.1.1 30 [19]        3011      Gi2.530:10010      Tu30:10010        ready

```

The final step in the configuration is configuring some kind of mechanism to detect a failed link. We can do this using RSVP hellos. Without a mechanism like this, RSVP will not be able to tell when the link fails, and will not be able to converge around it. The logic is identical to IGPs which necessitate some kind of active discovery mechanism when using TE-FRR. Enabling the command globally allows FRR to use the hellos, and it also needs to be enabled at the link level. We tune to timers to 5 second hellos with a neighbor being declared dead after 4 are missed, for a total of 20 seconds dead detection time.

```

! CSR3 and CSR10
ip rsvp signalling hello

interface GigabitEthernet2.530
  ip rsvp signalling hello
  ip rsvp signalling hello refresh interval 5000
  ip rsvp signalling hello refresh misses 4

```

Verifying on CSR3, we can see that RSVP hellos are enabled for FRR. CSR10 won't produce much output for the show commands because it is the "passive" side of the link. CSR3 is the tunnel head and has FRR bound to RSVP hellos as a client. The output even describes the difference between active and passive. The detailed output mostly shows various counters related to the session.

```
CSR3#show ip rsvp hello
```

```
Hello:
```

```
RSVP Hello for Fast-Reroute/Reroute: Enabled
```

```
Statistics: Disabled
```

```
BFD for Fast-Reroute/Reroute: Disabled
```

```
RSVP Hello for Graceful Restart: Disabled
```

```
CSR3#show ip rsvp hello instance summary
```

```
Active Instances:
```

Client	Neighbor	I/F	State	LostCnt	LSPs	Interval
FRR	132.3.10.10	Gi2.530	Up	0	1	5000

```
Passive Instances:
```

```
- None -
```

```
Active = Actively tracking neighbor state on behalf of clients:
```

```
RR = ReRoute, FRR = Fast ReRoute, or GR = Graceful Restart
```

```
Passive = Responding to hello requests from neighbor
```

```
CSR3#show ip rsvp hello instance detail
```

```
Neighbor 132.3.10.10 (router ID: 10.10.10.10) Source 132.3.10.3
```

```
Type: Active (sending requests)
```

```
I/F: GigabitEthernet2.530
```

```
State: Up (Since: 2015 MON DAY 14 13:14:08 )
```

```
Clients: Fast Reroute
```

```
LSPs protecting: 1
```

```
Missed acks: 4, IP DSCP: 0x30
```

```
Refresh Interval (msec)
```

```
Configured: 5000
```

```
Statistics: (from 225 samples)
```

```
Min: 5000
```

```
Max: 5001
```

```
Average: 5000
```

```
Waverage: 5000 (Weight = 0.8)
```

```
Current: 5000
```

```
Last sent Src_instance: 0x1FC275D3
```

```
Last recv nbr's Src_instance: 0x5256334A
```

```
Counters:
```

```
Communication with neighbor lost:
```

```
Num times: 0
```

```
Reasons:
```

```
Missed acks: 0
```

```
Bad Src_Instance received: 0
```

```
Bad Dst_Instance received: 0
```

```
I/F went down: 0
```

```
Neighbor disabled Hello: 0
```

```
Msgs Received: 1473
```

```
Sent: 1484
```

```
Suppressed: 0
```

CSR3 also has details regarding the FRR client and the neighbor capabilities supporting that FRR client.

```
CSR3#show ip rsvp hello client lsp detail
Hello Client LSPs (all lsp tree)
Tun Dest: 8.8.8.8 Tun ID: 30 Ext Tun ID: 1.1.1.1
Tun Sender: 1.1.1.1 LSP ID: 19
Lsp flags: 0x24
Lsp RR DN nbr: 132.3.10.10 FRR
```

```
CSR3#show ip rsvp hello client nbr detail
Hello Client Neighbors
Remote addr 132.3.10.10, Local addr 132.3.10.3
Nbr State: Normal Type: Reroute
Nbr Hello State: Up
LSPs protecting: 1
I/F: Gi2.530
```

CSR10 shows the session as passive as it is responding to hello requests. The detail shows the message counts.

```
CSR10#show ip rsvp hello instance summary | section Passive
Passive Instances:
Neighbor I/F
132.3.10.3 Gi2.530

CSR10#show ip rsvp hello instance detail
Neighbor 132.3.10.3 (router ID: 3.3.3.3) Source 132.3.10.10
Type: Passive (responding to requests)
I/F: GigabitEthernet2.530
Last sent Src_instance: 0x5256334A
Last recv nbr's Src_instance: 0x1FC275D3
Counters:
Msgs Received: 1444
Sent: 1444
```

Debugging RSVP dumps, we can see the hellos being exchanged back and forth. The ordinary RSVP hello debugging does not yield any output. At this point we can assume everything is working properly. Different colors are used to identify the different hello messages (checksums will match).

```
! CSR3
Outgoing Hello:
version:1 flags:0000 cksum:BD94 ttl:1 reserved:0 length:20
HELLO type HELLO REQUEST length 12:
Src_Instance: 0x1FC275D3, Dst_Instance: 0x5256334A
```

```
Incoming Hello:
```

```
version:1 flags:0000 cksum:BD93 ttl:1 reserved:0 length:20
HELLO type HELLO ACK length 12:
Src_Instance: 0x5256334A, Dst_Instance: 0x1FC275D3
```

! CSR10

Incoming Hello:

```
version:1 flags:0000 cksum:BD94 ttl:1 reserved:0 length:20
HELLO type HELLO REQUEST length 12:
Src_Instance: 0x1FC275D3, Dst_Instance: 0x5256334A
```

Outgoing Hello:

```
version:1 flags:0000 cksum:BD93 ttl:1 reserved:0 length:20
HELLO type HELLO ACK length 12:
Src_Instance: 0x5256334A, Dst_Instance: 0x1FC275D3
```

To test FRR, we will temporarily remove the backup path-option from the main LSP. This will give us more time to analyze traffic inside the FRR tunnel. When an LSP fails and FRR kicks in, the RSVP PATHERR message is sent back towards the head-end, causing a recalculation using the next path-option in sequence. Once we remove the alternative path-option, there is no other choice, so traffic is stuck in the FRR tunnel for some time. It is good design to have alternate path options for this reason when FRR is in use.

! CSR1

```
interface Tunnel30
no tunnel mpls traffic-eng path-option 20 dynamic attributes LSP_ATT_GREEN
```

To break the link, we will change the encapsulation on CSR10 to an incorrect VLAN. Shutting down the interface triggers a graceful shutdown, allowing IGP and RSVP to converge immediately, which is not a good way to test FRR. First, we will enable RSVP signaling debugs on CSR1 and CSR3 so we can watch the error event propagation. We also start a ping with a 1 second timeout on CSR7 (within the VPN) to XRv14. We expect to see about 20 dropped packets since we know it takes RSVP 20 seconds to determine there was a failure (also depends on IGP timers).

! CSR10

```
interface GigabitEthernet2.530
encapsulation dot1Q 999
```

After ~20 seconds, CSR3 determines there is a failure and sends an RSVP PATHERR message back to CSR1. CSR3 no longer has any RSVP neighbors. CSR3 should have locally repaired the tunnel.

! CSR3

```
RSVP: 1.1.1.1_29->8.8.8.8_30[Src] {7}: building error_spec object with err-
node addr: 132.1.3.3
RSVP: 1.1.1.1_29->8.8.8.8_30[Src] {7}: Sending PathError message to 132.1.3.1
RSVP: Triggering outgoing Path refresh
```



```
CSR3#show ip rsvp hello instance summary
Active Instances:
- None -
```

```
Passive Instances:
- None -
```

The labels have changed slightly from before since I was bringing the tunnel up and down for testing, but the output is clearly different. The FRR database shows this tunnel is "active". The label is still the ordinary TE label to CSR10. The new label pushed is 2010, so label 10012 is tunneled inside label 2010 across CSR2.

```
CSR3#show mpls traffic-eng fast-reroute database labels 3010 detail
FRR Database Summary:
  Protected interfaces      : 1
  Protected LSPs/Sub-LSPs  : 1
  Backup tunnels           : 1
  Active interfaces        : 1
  FRR Active tunnels       : 1
```

P2P LSPs:

```
Tun ID: 30, LSP ID: 60, Source: 1.1.1.1
Destination: 8.8.8.8
State      : active
InLabel    : 3010
OutLabel   : Gi2.530:10012
FRR OutLabel : Tu30:10012
```

```
CSR3#show mpls traffic-eng tunnels tunnel 30 | include Label
InLabel   : -
OutLabel  : GigabitEthernet2.523, 2010
```

CSR1 receives this RSVP PATHERR message, which is a notification that the tunnel has been locally repaired. This is a good notification, but also informs CSR1 to recalculate a new path. The goal of TE-FRR is not to redirect traffic long-term, but to allow the PLR-to-head signaling to take place, and continue passing traffic during that short time. In this case, the head-end tunnel times out after several minutes, allowing us to view the process. CSR1 claims the tunnel is pending a reroute, but the tunnel remains up since it was locally repaired. The RSVP RESV details also show that NHOP protection is available, but is also "in use". The flags have changed again to indicate this.

```
! CSR1
RSVP: 1.1.1.1_29->8.8.8.8_30[Src] {7}: Received PathError message from
132.1.3.3 (on GigabitEthernet2.513)
%MPLS_TE-5-LSP: LSP 1.1.1.1 30_29: Path Error from 132.1.3.3: Notify: Tunnel
locally repaired (flags 0)
```

```

CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 30
Reservation:
  Tun Dest:    8.8.8.8  Tun ID: 30  Ext Tun ID: 1.1.1.1
  Tun Sender: 1.1.1.1  LSP ID: 60
  Next Hop: 132.1.3.3 on GigabitEthernet2.513
  Label: 3010 (outgoing)
  Reservation Style is Shared-Explicit, QoS Service is Controlled-Load
  Resv ID handle: 18000404.
  Average Bitrate is 0 bits/sec, Maximum Burst is 1K bytes
  Min Policed Unit: 0 bytes, Max Pkt Size: 1500 bytes
RRO:
  3.3.3.3/32, Flags:0x23 (Local Prot Avail/In Use/to NHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3010
  10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 10012
  6.6.6.6/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 6012
  8.8.8.8/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3
Status:
Policy: Accepted. Policy source(s): MPLS/TE

CSR1#show mpls traffic-eng tunnels tunnel 30 | section Status
Status:
  Admin: up      Oper: up      Path: valid      Signalling: connected
  path option 10, type explicit PATH_1_3_10_6_8 (Basis for Setup, path
weight 40)
  Change in required resources detected: reroute pending
  Currently Signalled Parameters:
    Bandwidth: 0      kbps (Global)  Priority: 7  7  Affinity:
0x0/0x0
    Metric Type: TE (default)

```

On CSR7, we can see 18 dropped packets (275 - 257), which is approximately correct given the 20 second RSVP dead-peer detection time. Traffic is now using the FRR tunnel since CSR1 cannot recalculate a new path, given that we removed the backup path option. We can also confirm the imposition of the third label (FRR) on the PLR (CSR3), which tunnels traffic across the backup path to the MP (CSR10). The label arriving on CSR10 is the normal TE label, which means the backup tunnel can be used for many LSPs (facility backup).

```

CSR7#ping 14.14.14.14 source 7.7.7.7 repeat 1000000 timeout 1
Type escape sequence to abort.
Sending 1000000, 100-byte ICMP Echos to 14.14.14.14, timeout is 1 seconds:
Packet sent with a source address of 7.7.7.7
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!.....
.....!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

```

Success rate is 93 percent (257/275), round-trip min/avg/max = 1/28/194 ms

```
CSR7#traceroute 14.14.14.14 source 7.7.7.7
Type escape sequence to abort.
Tracing the route to 14.14.14.14
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.7.1 1 msec 0 msec 1 msec
 2 132.1.3.3 [MPLS: Labels 3010/8009 Exp 0] 2 msec 2 msec 9 msec
 3 132.2.3.2 [MPLS: Labels 2010/10012/8009 Exp 0] 41 msec 32 msec 36 msec
 4 132.2.10.10 [MPLS: Labels 10012/8009 Exp 0] 35 msec 31 msec 45 msec
 5 132.6.10.6 [MPLS: Labels 6012/8009 Exp 0] 36 msec 35 msec 37 msec
 6 10.8.14.8 [MPLS: Label 8009 Exp 0] 82 msec 13 msec 11 msec
 7 10.8.14.14 12 msec * 2 msec
```

Using RSVP hellos is very uncommon since BFD has been introduced. Like many other protocols, the TE-FRR process can be a BFD client, obviating the need to run a new dead peer detection mechanism. We will enable this on all XE-to-XE links in the topology; XRv does not support BFD, and XR in general does not support RSVP slow hellos. The configuration is almost identical to the RSVP hellos except we add the word "bfd" to the end. This cannot coexist with the legacy RSVP hellos, which must be removed first on CSR3 and CSR10 before configuring this (not shown). Checking CSR3, we can see that BFD for FRR is now enabled instead of RSVP hellos.

```
! All XE MPLS routers
ip rsvp signalling hello bfd

interface GigabitEthernet2.XXX
 ip rsvp signalling hello bfd

CSR3#show ip rsvp hello
Hello:
  RSVP Hello for Fast-Reroute/Reroute: Disabled
  Statistics: Disabled
  BFD for Fast-Reroute/Reroute: Enabled
  RSVP Hello for Graceful Restart: Disabled
```

BFD TE-FRR neighbors won't show up until a TE tunnel runs through it. As described in the BFD section, there is no neighbor discovery, but higher-level processes (TE-FRR) can notify BFD when they need fast fall-over. The signaling of the original LSP requesting FRR builds this. In this case, the main LSP is shutdown on CSR1 but the FRR tunnel is up. On the PLR (CSR3), we can see the details of the BFD neighbor with respect to TE-FRR now. This is a host-downstream neighbor as it is the NHOP of the backup TE tunnel.

```
CSR3#show ip rsvp hello bfd nbr detail
Hello Client Neighbors
Remote addr 132.2.3.2, Local addr 132.2.3.3
Type: Active
```

```
I/F: Gi2.523
State: Up (for 00:17:38)
Clients: HST
LSPs protecting: 1 (frr: 0, hst upstream: 0 hst downstream: 1)
Communication with neighbor lost: 0
```

```
CSR3#show bfd neighbors client te-frr
```

```
IPv4 Sessions
```

NeighAddr	LD/RD	RH/RS	State	Int
132.2.3.2	4100/4097	Up	Up	
Gi2.523				

Bringing up the original LSP adds two new TE-FRR clients to BFD, the PHOP (CSR1) and NHOP (CSR10), to use RSVP terminology. The RSVP details show the PHOP as a passive neighbor and the NHOP as an active neighbor. The NHOP neighbor is protected by FRR which is indicated by the value "1"; the output above has a zero since the backup tunnel is not protected by FRR.

```
CSR3#show bfd neighbors client te-frr
```

```
IPv4 Sessions
```

NeighAddr	LD/RD	RH/RS	State	Int
132.1.3.1	4101/4100	Up	Up	Gi2.513
132.2.3.2	4100/4097	Up	Up	Gi2.523
132.3.10.10	4103/4104	Up	Up	Gi2.530

```
CSR3#show ip rsvp hello bfd nbr detail
```

```
Hello Client Neighbors
```

```
Remote addr 132.1.3.1, Local addr 132.1.3.3
```

```
Type: Passive
```

```
I/F: Gi2.513
```

```
State: Up (for 00:00:53)
```

```
Clients: None
```

```
LSPs protecting: 0
```

```
Communication with neighbor lost: 0
```

```
Remote addr 132.3.10.10, Local addr 132.3.10.3
```

```
Type: Active
```

```
I/F: Gi2.530
```

```
State: Up (for 00:00:53)
```

```
Clients: FRR
```

```
LSPs protecting: 1 (frr: 1, hst upstream: 0 hst downstream: 0)
```

```
Communication with neighbor lost: 0
```

We quickly check the RSVP RESV details on CSR1 to ensure the tunnel still has FRR and that enabling BFD did not break any RSVP signaling configuration. The PLR shows the backup tunnel as ready and shows the in/out labels for the original LSP from CSR1.

```
CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 30 |
begin RRO
```

```
RRO:
```

```
3.3.3.3/32, Flags:0x21 (Local Prot Avail/to NHOP, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 3013
10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 10014
6.6.6.6/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 6014
8.8.8.8/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 3
```

```
Status:
```

```
Policy: Accepted. Policy source(s): MPLS/TE
```

```
CSR3#show mpls traffic-eng fast-reroute database labels 3013
```

```
P2P LSP midpoint frr information:
```

LSP identifier	In-label	Out intf/label	FRR intf/label	Status
1.1.1.1 30 [112]	3013	Gi2.530:10014	Tu30:10014	ready

Starting a ping from CSR7 and breaking the link again by changing encapsulation, we see the failover is about 1 second as BFD requires 900 ms for dead peer detections based on current configuration. The PLR marks this LSP as being actively protected, and the PLR's LFIB shows a label swap followed by a push. Not only did the traffic get re-routed faster, we removed another set of packets (RSVP hellos) from the network, saving some bandwidth and router resources.

```
CSR7#ping 14.14.14.14 source 7.7.7.7 repeat 1000000 timeout 1
```

```
Type escape sequence to abort.
```

```
Sending 1000000, 100-byte ICMP Echos to 14.14.14.14, timeout is 1 seconds:
```

```
Packet sent with a source address of 7.7.7.7
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!
```

```
Success rate is 99 percent (147/148), round-trip min/avg/max = 1/27/166 ms
```

```
CSR3#show mpls traffic-eng fast-reroute database labels 3013
```

```
P2P LSP midpoint frr information:
```

LSP identifier	In-label	Out intf/label	FRR intf/label	Status
1.1.1.1 30 [112]	3013	Gi2.530:10014	Tu30:10014	active

```
CSR3#show mpls forwarding-table labels 3013 detail
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
3013	10014	1.1.1.1 30 [112]	20738	Tu30	point2point
MAC/Encaps=18/26, MRU=1496, Label Stack{2014 10014}, via Gi2.523					
000C295CE1E9000C29D781FE81000DC38847 007DE0000271E000					
No output feature configured					

CSR1 shows that the NHOP protection is now in use for this LSP. The flags change again to indicate this. Although the behavior is identical because BFD and RSVP hellos, it is always recommended to use BFD when available.

```
CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 30 |
begin RRO
RRO:
  3.3.3.3/32, Flags:0x23 (Local Prot Avail/In Use/to NHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3013
  10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 10014
  6.6.6.6/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 6014
  8.8.8.8/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3
Status:
Policy: Accepted. Policy source(s): MPLS/TE
```

We can also configure NNHOP protection, also called node protection. Rather than protect a link between two routers, we can avoid a node entirely. We will configure CSR9 with an FRR tunnel that avoids CSR6 completely, protecting the PLR's next-next-hop using any link colors. The destination of the tunnel should be the next-hop after the protected node, which is CSR8, the actual tail end. CSR8 is technically the MP also in this case. We can see this NNHOP backup tunnel goes through CSR9 to CSR8 after the tunnel is signaled.

```
! CSR10
ip explicit-path name PATH_AVOID_CSR6 enable
  exclude-address 6.6.6.6

interface Tunnel30
  description REPAIR PATH NNHOP (PLR)
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 8.8.8.8
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 explicit name PATH_AVOID_CSR6

interface GigabitEthernet2.560
  mpls traffic-eng backup-path Tunnel30
```

```
CSR10#show mpls traffic-eng tunnels tunnel 30 | section RSVP_Path
RSVP Path Info:
  My Address: 132.9.10.10
  Explicit Route: 132.9.10.9 132.8.9.8 8.8.8.8
  Record Route: NONE
  Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits
```

Checking CSR1, we can see that CSR10 is offering NNHOP (node protection) in connection with CSR3 offering NHOP (link) protection. Each individual hop in the TE LSP can offer different protection. We see that the NNHOP protection is available, but not in use, and the flags are different than the rest (0x29). In an ideal network, every single hop would have some kind of FRR protection.

```
CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 30 |
begin RRO
RRO:
  3.3.3.3/32, Flags:0x21 (Local Prot Avail/to NHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3014
  10.10.10.10/32, Flags:0x29 (Local Prot Avail/to NNHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 10010
  6.6.6.6/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 6015
  8.8.8.8/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3
Status:
Policy: Accepted. Policy source(s): MPLS/TE
```

The RESV RRO shows label 10010 arriving to CSR10 along the LSP, so we can check the FRR database against this inbound label. Along the normal LSP, this inbound label is swapped to 6015 which is also present in the RESV RRO. When NNHOP protection is in use, we no longer need to preserve the NHOP label since CSR6 won't be seeing this traffic at all. Instead, we can swap label 10010 for label 9011, which is the FRR label in for the backup tunnel. Technically, this isn't a label swap operation, it is pushing implicit-null and then 9011, as we will see later. The router is probably smart enough to do a swap for efficiency reasons in this special case as the LFIB suggests.

```
CSR10#show mpls traffic-eng fast-reroute database labels 10010 detail
[snip]
P2P LSPs:
  Tun ID: 30, LSP ID: 137, Source: 1.1.1.1
  Destination: 8.8.8.8
  State       : ready
  InLabel     : 10010
  OutLabel    : Gi2.560:6015
  FRR OutLabel : Tu30:implicit-null

CSR10#show mpls traffic-eng tunnels tunnel 30 | include Label
  InLabel  : -
  OutLabel : GigabitEthernet2.590, 9011
```

Changing the encapsulation on CSR6 so the link to CSR10 fails, we see the PLR in action. Traffic is redirected into the tunnel and the LFIB reflects this. The FRR database is updated and a RESV message is sent to CSR1 notifying it that NNHOP protection is now activated on CSR10.

```

CSR10#show mpls forwarding-table labels 10010 detail
Local      Outgoing   Prefix      Bytes Label  Outgoing   Next Hop
Label      Label        or Tunnel Id Switched      interface
10010      Pop Label    1.1.1.1 30 [137] 4888          Tu30        point2point
          MAC/Encaps=18/22, MRU=1500, Label Stack{9011}, via Gi2.590
          000C29E04F84000C2949716481000E068847 02333000
          No output feature configured

CSR10#show mpls traffic-eng fast-reroute database labels 10010
P2P LSP midpoint frr information:
LSP identifier          In-label Out intf/label    FRR intf/label    Status
-----
1.1.1.1 30 [137]       10010    Gi2.560:6015      Tu30:implicit-nu  active

CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 30 |
begin RRO
RRO:
  3.3.3.3/32, Flags:0x21 (Local Prot Avail/to NHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3014
  10.10.10.10/32, Flags:0x2B (Local Prot Avail/In Use/to NNHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 10010
  6.6.6.6/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 6015
  8.8.8.8/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3
Status:
Policy: Accepted. Policy source(s): MPLS/TE

```

We confirm connectivity through this new path within the VPN. Notice there aren't 3 labels anymore since we don't need to tunnel traffic to the next-hop, thus preserving the original TE label. We can swap it to the FRR label since we are essentially taking a new path. Technically, we "pushed" the NNHOP's label to reach the tail-end, but since it was the same router, no additional label was needed. The next-next-hop label was 3, which means the tail-end and MP are the same router.

```

CSR7#traceroute 14.14.14.14 source 7.7.7.7
Type escape sequence to abort.
Tracing the route to 14.14.14.14
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.7.1 0 msec 1 msec 1 msec
 2 132.1.3.3 [MPLS: Labels 3014/8009 Exp 0] 2 msec 2 msec 21 msec
 3 132.3.10.10 [MPLS: Labels 10010/8009 Exp 0] 49 msec 31 msec 35 msec
 4 132.9.10.9 [MPLS: Labels 9011/8009 Exp 0] 31 msec 31 msec 31 msec
 5 10.8.14.8 [MPLS: Label 8009 Exp 0] 50 msec 1 msec 7 msec
 6 10.8.14.14 24 msec * 2 msec

```

We can also have multiple FRR backup tunnels in use at the same time protecting the same LSP. Assuming the CSR3-CSR10 link fails at the same time CSR6 fails, both the NHOP and NNHOP backups can

be in use concurrently (link breaks not shown). We can verify this on each PLR by checking the FRR database and seeing both entries as "active". We can also verify within the VPN using traceroute; we see the NHOP protection across CSR2 and the NNHOP across CSR9 concurrently. I highlight the LSP information also (Tunnel ID 30, LSP ID 164) to prove it is the same TE tunnel.

```
CSR3#show mpls traffic-eng fast-reroute database
P2P LSP midpoint frr information:
LSP identifier          In-label Out intf/label    FRR intf/label    Status
-----
1.1.1.1 30 [164]       3014     Gi2.530:10013      Tu30:10013        active
```

```
CSR10#show mpls traffic-eng fast-reroute database
P2P LSP midpoint frr information:
LSP identifier          In-label Out intf/label    FRR intf/label    Status
-----
1.1.1.1 30 [164]       10013     Gi2.560:6010      Tu30:implicit-nu  active
```

```
CSR7#traceroute 14.14.14.14 source 7.7.7.7
Type escape sequence to abort.
Tracing the route to 14.14.14.14
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.7.1 0 msec 1 msec 1 msec
 2 132.1.3.3 [MPLS: Labels 3014/8009 Exp 0] 2 msec 2 msec 9 msec
 3 132.2.3.2 [MPLS: Labels 2015/10013/8009 Exp 0] 37 msec 41 msec 31 msec
 4 132.2.10.10 [MPLS: Labels 10013/8009 Exp 0] 36 msec 35 msec 31 msec
 5 132.9.10.9 [MPLS: Labels 9011/8009 Exp 0] 40 msec 36 msec 40 msec
 6 10.8.14.8 [MPLS: Label 8009 Exp 0] 16 msec 15 msec 20 msec
 7 10.8.14.14 24 msec * 2 msec
```

A certain node may offer both NHOP and NNHOP protection concurrently. A router can provide multiple backup tunnels on a given interface as well. On CSR3, we will configure NNHOP protection (in addition to the NHOP protect) using a new tunnel [ID 31]. This will prefer a green path to the NNHOP, which is CSR6. We quickly verify the tunnel path; interestingly, it traverses CSR1, which looks like a loop, but technically is not since this is a different LSP. The path continues through CSR9 to merge onto CSR6.

```
! CSR3
ip explicit-path name PATH_AVOID_CSR10 enable
exclude-address 10.10.10.10

interface Tunnel31
description REPAIR PATH NNHOP (PLR)
ip unnumbered Loopback0
tunnel mode mpls traffic-eng
tunnel destination 6.6.6.6
tunnel mpls traffic-eng affinity 0x2 mask 0x2
tunnel mpls traffic-eng path-option 10 explicit name PATH_AVOID_CSR10
```

```
interface GigabitEthernet2.530
 mpls traffic-eng backup-path Tunnel31
```

```
CSR3#show mpls traffic-eng tunnels tunnel 31 | section RSVP_Path
RSVP Path Info:
  My Address: 132.1.3.3
  Explicit Route: 132.1.3.1 132.1.9.9 132.6.9.6 6.6.6.6
  Record Route: NONE
  Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits
```

We can check the FRR database to see that the original tunnel is now being backed up by the new NNNHOP tunnel and not the original NHOP tunnel at CSR3. TE tunnels requesting FRR will always prefer NNNHOP over NHOP when it is available (there are other comparison criteria which are evaluated in detail later). We confirm this protection on CSR1; notice that NNNHOP protection is available at both CSR3 and CSR10 now.

```
CSR3#show mpls traffic-eng fast-reroute database backup-interface tunnel 31
detail
[snip]
P2P LSPs:
  Tun ID: 30, LSP ID: 212, Source: 1.1.1.1
  Destination: 8.8.8.8
  State : ready
  InLabel : 3020
  OutLabel : Gi2.530:10013
  FRR OutLabel : Tu31:6010
```

```
CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 30 |
begin RRO
RRO:
  3.3.3.3/32, Flags:0x29 (Local Prot Avail/to NNNHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3020
  10.10.10.10/32, Flags:0x29 (Local Prot Avail/to NNNHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 10013
  6.6.6.6/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 6010
  8.8.8.8/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3
Status:
Policy: Accepted. Policy source(s): MPLS/TE
```

Notice that the outgoing label in the FRR database is the NNNHOP label of 6010. Normally, CSR10 would have performed the swap from 10013 to 6010, but assuming CSR10 fails, CSR3 will perform this swap from 3020 to 6010, and then push the FRR label of 1015.

```
CSR3#show mpls traffic-eng tunnels tunnel 31 | include Label
InLabel : -
```

OutLabel : GigabitEthernet2.513, 1015

```
CSR3#show mpls traffic-eng fast-reroute database backup-interface tunnel 31
[snip]
```

P2P LSP midpoint frr information:

LSP identifier	In-label	Out intf/label	FRR intf/label	Status
1.1.1.1 30 [212]	3020	Gi2.530:10013	Tu31:6010	ready

We quickly test this new NNHOP protection by changing the encapsulation on CSR10's link to CSR3. Rather than avoiding that link, CSR3 will now avoid CSR10 entirely, making CSR6 the MP and using CSR6's local label for the original LSP for merging. The FRR database marks this entry as active since FRR is in effect.

```
CSR3#show mpls traffic-eng fast-reroute database backup-interface tunnel 31
[snip]
```

P2P LSP midpoint frr information:

LSP identifier	In-label	Out intf/label	FRR intf/label	Status
1.1.1.1 30 [212]	3020	Gi2.530:10013	Tu31:6010	active

```
CSR3#show mpls forwarding-table labels 3020 detail
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Label	Outgoing interface	Next Hop
3020	6010	1.1.1.1 30 [212]	5792		Tu31	point2point
MAC/Encaps=18/26, MRU=1496, Label Stack{1015 6010}, via Gi2.513						
000C29FBA339000C29D781FE81000DB98847 003F70000177A000						
No output feature configured						

```
CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 30 |
begin RRO
```

RRO:

3.3.3.3/32, Flags:0x2B (Local Prot Avail/In Use/to NNHOP, Node-id)

Label subobject: Flags 0x1, C-Type 1, Label 3020

10.10.10.10/32, Flags:0x29 (Local Prot Avail/to NNHOP, Node-id)

Label subobject: Flags 0x1, C-Type 1, Label 10013

6.6.6.6/32, Flags:0x20 (No Local Protection, Node-id)

Label subobject: Flags 0x1, C-Type 1, Label 6010

8.8.8.8/32, Flags:0x20 (No Local Protection, Node-id)

Label subobject: Flags 0x1, C-Type 1, Label 3

Status:

Policy: Accepted. Policy source(s): MPLS/TE

```
CSR7#traceroute 14.14.14.14 source 7.7.7.7
```

Type escape sequence to abort.

Tracing the route to 14.14.14.14

VRF info: (vrf in name/id, vrf out name/id)

1 10.1.7.1 1 msec 0 msec 0 msec

```

2 132.1.3.3 [MPLS: Labels 3020/8009 Exp 0] 3 msec 18 msec 98 msec
3 132.1.3.1 [MPLS: Labels 1015/6010/8009 Exp 0] 19 msec 26 msec 47 msec
4 132.1.9.9 [MPLS: Labels 9016/6010/8009 Exp 0] 51 msec 161 msec 26 msec
5 132.6.9.6 [MPLS: Labels 6010/8009 Exp 0] 39 msec 39 msec 30 msec
6 10.8.14.8 [MPLS: Label 8009 Exp 0] 36 msec 39 msec 34 msec
7 10.8.14.14 25 msec * 2 msec

```

We can also set a special flag in the RSVP PATH message to signal that node protection should be preferred over link protection. Because IOS XE and XR both automatically prefer NNHOP over NHOP, this option doesn't do much, but may help in multi-vendor environments where the explicit flag in the RSVP PATH message is used to select NNHOP. The PATH and RESV messages are shown below. The PATH message shows the "node-protection desired" flag while the RSV message is unchanged, since NNHOP protection is already preferred on CSR3 (PLR).

```

! CSR1
interface Tunnel30
 tunnel mpls traffic-eng fast-reroute node-protect

CSR1#show ip rsvp sender detail filter session-type 7 tunnel-id 30 | section
Flags
  Flags: (0x17) Local Prot desired, Label Recording, SE Style
             Node Prot desired

CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 30 |
begin RRO
RRO:
  3.3.3.3/32, Flags:0x29 (Local Prot Avail/to NNHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3022
  10.10.10.10/32, Flags:0x29 (Local Prot Avail/to NNHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 10022
  6.6.6.6/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 6016
  8.8.8.8/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3
Status:
Policy: Accepted. Policy source(s): MPLS/TE

```

Another special flag can be assigned to request bandwidth protection. This is used as a tie-breaker when there are FRR tunnels with an aggregate bandwidth greater than the aggregate bandwidth offered by the set of FRR tunnels. FRR tunnels will request bandwidth, and backup tunnels can provide it. Reserving bandwidth on backup tunnels can be wasteful/inefficient (it is an option, though). Alternatively, we can specify the backup bandwidth on the repair tunnels. This serves as an admission control mechanism to ensure that, when using facility backups, there isn't an oversubscription of bandwidth inside backup tunnels for primary TE LSPs that require dedicated bandwidth. The bandwidth is not actually reserved in the TED, but it does help balance the FRR flows across various backup paths when a failure occurs. The main tunnel requests 1 Mbps of bandwidth, while the NHOP and NNHOP backups on CSR3 offer 3 Mbps

and 2 Mbps of backup bandwidth, respectively. We can see the PATH message "bandwidth-protection desired" flag set, and the RESV message also indicates that bandwidth protection is enabled for that hop using the NNHOP tunnel.

```
! CSR1
interface Tunnel30
  tunnel mpls traffic-eng bandwidth 1000
  tunnel mpls traffic-eng fast-reroute bw-protect

! CSR3
interface Tunnel30
  tunnel mpls traffic-eng backup-bw 3000

interface Tunnel31
  tunnel mpls traffic-eng backup-bw 2000

CSR1#show ip rsvp sender detail filter session-type 7 tunnel-id 30 | section
Flags
  Flags: (0x1F) Local Prot desired, Label Recording, SE Style, Bandwidth
Prot desired

CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 30 |
begin RRO
RRO:
  3.3.3.3/32, Flags:0x2D (Local Prot Avail/Has BW/to NNHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3018
  10.10.10.10/32, Flags:0x29 (Local Prot Avail/to NNHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 10014
  6.6.6.6/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 6016
  8.8.8.8/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3
Status:
Policy: Accepted. Policy source(s): MPLS/TE
```

If we configure the main TE LSP to request more bandwidth than the NNHOP tunnel can provide, but the NHOP tunnel can provide it, the NHOP tunnel is preferred since bandwidth-protection was enabled. NNHOP tunnels with no bandwidth are still preferred over NHOP tunnels with explicit bandwidth, even when bandwidth-protection is configured. This is because a backup tunnel with “no” bandwidth is assumed to have unlimited bandwidth. The exact sequence of preferences is detailed more in the DS-TE section.

```
! CSR1
interface Tunnel30
  tunnel mpls traffic-eng bandwidth 3000
```

```

CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 30 |
begin RRO
RRO:
  3.3.3.3/32, Flags:0x25 (Local Prot Avail/Has BW/to NHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3017
  10.10.10.10/32, Flags:0x29 (Local Prot Avail/to NNHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 10022
  6.6.6.6/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 6017
  8.8.8.8/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3
Status:
Policy: Accepted. Policy source(s): MPLS/TE

```

Bandwidth protection takes precedence over node protection when both flags are set concurrently in the PATH message. Even after adding the node-protection flag to the main LSP head-end, the NHOP tunnel is preferred due to the bandwidth-backup request.

```

! CSR1
interface Tunnel30
  tunnel mpls traffic-eng fast-reroute bw-protect node-protect

```

```

CSR1#show ip rsvp sender detail filter session-type 7 tunnel-id 30 | section
Flags
  Flags: (0x1F) Local Prot desired, Label Recording, SE Style, Bandwidth
  Prot desired Node Prot desired

```

```

CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 30 |
begin RRO
RRO:
  3.3.3.3/32, Flags:0x25 (Local Prot Avail/Has BW/to NHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3022
  10.10.10.10/32, Flags:0x29 (Local Prot Avail/to NNHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 10012
  6.6.6.6/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 6024
  8.8.8.8/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3
Status:
Policy: Accepted. Policy source(s): MPLS/TE

```

Because the vast majority of the TE-FRR configuration and complexities lies with the PLR, we will not be testing this again with XRv as it does not support BFD. However, we will configure dummy tunnels on XRv11 to show the syntax for the features and perform basic control-plane verifications of the RSVP signaling. The head-end LSP is shown below, which requests FRR, along with node protection and bandwidth protection. We also verify the tunnel comes up.

```

! XRv11
explicit-path name PATH_11_5_2_3_10_6_8
  index 10 next-address strict ipv4 unicast 5.5.5.5
  index 20 next-address strict ipv4 unicast 2.2.2.2
  index 30 next-address strict ipv4 unicast 3.3.3.3
  index 40 next-address strict ipv4 unicast 10.10.10.10
  index 50 next-address strict ipv4 unicast 6.6.6.6
  index 60 next-address strict ipv4 unicast 8.8.8.8

interface tunnel-te30
  description BASIC TE FRR
  ipv4 unnumbered Loopback0
  logging events all
  signalled-bandwidth 3000
  destination 8.8.8.8
  fast-reroute protect node bandwidth
  affinity ignore
  path-option 10 explicit name PATH_11_5_2_3_10_6_8

```

```
P/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 30 brief
```

TUNNEL NAME	DESTINATION	STATUS	STATE
tunnel-te30	8.8.8.8	up	up

We can check the locally-generated RSVP PATH message to verify the flags: local protection, node protection, and bandwidth protection are all enabled. Because we have built this tunnel over the CSR3-CSR10 link, we have the NHOP and NHHOP tunnel protection there, as well as the NNHOP protection CSR10-CSR6. The RSVP RESV message doesn't show this in XR, so we look at the TE tunnel details instead. Specifically, we can see there is protection available, along with bandwidth protection, at CSR3. The word "node" does not appear, so we assume this is NHOP protection. At CSR10, we have available "node" protection (NNHOP) but no bandwidth protection. The XR outputs are more difficult to interpret than the XE ones inside the RESV message, but the summary information shown near the beginning is succinct and clear.

```

RP/0/0/CPU0:XRv11#show rsvp sender session-type lsp-p2p dst-port 30 detail
PATH: IPv4-LSP Session addr: 8.8.8.8. TunID: 30. LSPIID: 2.
Source addr: 11.11.11.11. ExtID: 11.11.11.11.
Prot: Local, Node, BW. Backup tunnel: No.
Setup Priority: 7, Reservation Priority: 7
Rate: 3M bits/sec. Burst: 1K bytes. Peak: 3M bits/sec.
[snip]

```

```

RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 30 detail | begin Resv Info
Resv Info:
Record Route:
  IPv4 5.5.5.5, flags 0x20 (Node-ID)
  Label 5013, flags 0x1
  IPv4 2.2.2.2, flags 0x20 (Node-ID)

```

```

Label 2017, flags 0x1
IPv4 3.3.3.3, flags 0x25 (Node-ID, Protection: available, b/w)
Label 3021, flags 0x1
IPv4 10.10.10.10, flags 0x29 (Node-ID, Protection: available, node)
Label 10016, flags 0x1
IPv4 6.6.6.6, flags 0x20 (Node-ID)
Label 6012, flags 0x1
IPv4 8.8.8.8, flags 0x20 (Node-ID)
Label 3, flags 0x1
Fspec: avg rate=3000 kbits, burst=1000 bytes, peak rate=3000 kbits

```

We can quickly check the FRR databases on the PLRs, CSR3 and CSR10, to verify this protected tunnel is shown. The RSVP RESV shown about gives us the labels per hop of the original LSP, so we can use those as input for the FRR database to specifically look at the TE tunnel from XRv11 to CSR8.

```

CSR3#show mpls traffic-eng fast-reroute database labels 3021
P2P LSP midpoint frr information:
LSP identifier          In-label Out intf/label   FRR intf/label   Status
-----
11.11.11.11 30 [2]      3021     Gi2.530:10016      Tu30:10016       ready

CSR10#show mpls traffic-eng fast-reroute database labels 10016
P2P LSP midpoint frr information:
LSP identifier          In-label Out intf/label   FRR intf/label   Status
-----
11.11.11.11 30 [2]      10016    Gi2.560:6012      Tu30:implicit-nu ready

```

Backup tunnel configuration is similar to XE as the backup tunnel itself has no special characteristics and is not shown. Assigning a backup tunnel to a link is shown below.

```

! XRv11
interface tunnel-te31
 description DUMMY BACKUP PATH
 shutdown

mpls traffic-eng
 interface GigabitEthernet0/0/0/0.551
  backup-path tunnel-te 31

```

Link and node protection are the best ways to do TE-FRR. An alternative exists which is a little slower and less scalable, but not nearly as slow as having disparate path-options. This is called path protection, which is configured on a per path-option basis and is used to backup that path-option. In addition to the main path-option, this “protecting” or “backup” path is pre-sigaled. In IOS, backup-paths must be explicit, while the main path can be explicit or dynamic. As soon as the head-end receives the PATHERR message from the network core, it can immediately switch traffic over into the backup tunnel. This is still slower than NHOP/NNHOP protection since local repair is always faster, but it does not require any

configuration changes in the network core. This might be a compromise if different organizations/administrators control P routers versus PE routers. We can create a list of backup path-options to allow multiple choices for a single path-option we want to backup. Only the best option in the list is pre-sigaled. These backup paths cannot use NHOP/NNHOP protection for themselves, since they are backup paths already. For efficiency, we can re-use some older explicit-paths for backup options. There is no reliance on RESV RROs with label recording as the headend performs the rerouting (no concept of PLR).

```
! CSR1
ip explicit-path name EP_1_4_3_10_11 enable
  next-address 4.4.4.4
  next-address 3.3.3.3
  next-address 10.10.10.10
  next-address 11.11.11.11

mpls traffic-eng path-option list name POPT_LIST
  path-option 100 explicit name PATH_1_3_2_10_11
  path-option 110 explicit name EP_NO_R9

interface Tunnel32
  description PATH PROTECTION
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng autoroute destination
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 explicit name EP_1_4_3_10_11
  tunnel mpls traffic-eng path-option protect 10 list name POPT_LIST bandwidth
  3000
```

CSR1 now shows two tunnels from 1.1.1.1 to 11.11.11.11. Since we created a 3 Mbps bandwidth reservation for the backup path, we can easily tell the difference at a glance. It is clear that both paths are properly signaled. Also notice that label 4014 is used for the primary path with label 3022 used for the backup path.

```
CSR1#show ip rsvp reservation filter session-type 7 destination 11.11.11.11
Destination      Tun Sender      TunID LSPID Next Hop      I/F      Fi Serv
BPS
11.11.11.11      1.1.1.1         32    33    132.1.3.3      Gi2.513  SE LOAD 3M
11.11.11.11      1.1.1.1         32    34    132.1.4.4      Gi2.514  SE LOAD 0
```

```
CSR1#show ip rsvp reservation detail filter session-type 7 destination
11.11.11.11 | include Label|LSP_ID
  Tun Sender: 1.1.1.1  LSP ID: 33
  Label: 3022 (outgoing)
  Tun Sender: 1.1.1.1  LSP ID: 34
  Label: 4014 (outgoing)
```

Verifying the tunnel status, we see that the ordinary path-option 10 is used for the tunnel setup. The path protection engine tells us that the primary and backup paths share 1 link and 2 nodes, which is not ideal. Normally you would ensure the two paths were as different as possible so that the failure of a single link/node doesn't break both paths. Path-option 100 was the most desirable backup option and is the "basis for protect".

```
CSR1#show mpls traffic-eng tunnels tunnel 32 | section Status
Status:
  Admin: up      Oper: up      Path: valid      Signalling: connected
  path option 10, type explicit EP_1_4_3_10_11 (Basis for Setup, path
weight 40)
  Path Protection: 1 Common Link(s), 2 Common Node(s)
  path protect option 10, type list name POPT_LIST
  Inuse path-option 100, type explicit (Basis for Protect, path weight
50)
```

For additional detail on the two paths, we can look at the specific protection attributes of the tunnel. We can see that the CSR10-XRv11 connection is the common link, with CSR3 and CSR10 being the common nodes.

```
CSR1#show mpls traffic-eng tunnels tunnel 32 protection
PATH PROTECTION
LSP Head, Tunnel32, Admin: up, Oper: up
Src 1.1.1.1, Dest 11.11.11.11, Instance 34
Fast Reroute Protection: None
Path Protection: 1 Common Link(s), 2 Common Node(s)
Primary lsp path: 132.1.4.4 132.3.4.3
                  132.3.10.10 132.10.11.11
                  11.11.11.11
Protect lsp path: 132.1.3.3 132.3.9.9
                  132.2.9.2 132.2.10.10
                  132.10.11.11 11.11.11.11
```

A quick look on XRv11 shows both tunnels terminating. This helps prove that the backup path is fully signaled in advanced.

```
RP/0/0/CPU0:XRv11#show rsvp reservation destination 11.11.11.11
Destination Add DPort      Source Add SPort Pro   Input IF Sty Serv   Rate Burst
-----
11.11.11.11  32      1.1.1.1  33    0      No  SE  LOAD  3000K  1K
11.11.11.11  32      1.1.1.1  34    0      No  SE  LOAD   0    1K
```

Using autoroute destination for simplicity, traffic from CSR1 to XRv11 now traverses this tunnel. We can test it from inside the VPN, expecting to see label 4014 at the first MPLS hop.

```

CSR7#traceroute 13.13.13.13 source 7.7.7.7
Type escape sequence to abort.
Tracing the route to 13.13.13.13
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.7.1 1 msec 0 msec 1 msec
 2 132.1.4.4 [MPLS: Labels 4014/91012 Exp 0] 37 msec 25 msec 25 msec
 3 132.3.4.3 [MPLS: Labels 3023/91012 Exp 0] 15 msec 17 msec 16 msec
 4 132.3.10.10 [MPLS: Labels 10017/91012 Exp 0] 21 msec 46 msec 38 msec
 5 132.10.11.11 [MPLS: Label 91012 Exp 0] 28 msec 25 msec 25 msec
 6 10.11.13.13 25 msec * 40 msec

```

Assuming CSR4's link to CSR1 fails (not shown), the backup path should route around it. We know this is not a shared link or shared node, so the backup path is still effective according to the protection details we saw earlier. Shortly after shutting down this interface, we send additional VPN traffic to verify the new label 3022 is used along the backup LSP. We can also verify the tunnel status to see the backup path in use. The closer the failure is to the head-end, the less distance the PATHERR has to travel, which means faster convergence. With NHOP/NNHOP FRR, the failure location is irrelevant, which leads to a more consistent FRR behavior.

```

CSR7#traceroute 13.13.13.13 source 7.7.7.7
Type escape sequence to abort.
Tracing the route to 13.13.13.13
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.7.1 0 msec 1 msec 1 msec
 2 132.1.3.3 [MPLS: Labels 3022/91012 Exp 0] 18 msec 16 msec 24 msec
 3 132.3.9.9 [MPLS: Labels 9024/91012 Exp 0] 22 msec 24 msec 25 msec
 4 132.2.9.2 [MPLS: Labels 2022/91012 Exp 0] 20 msec 17 msec 20 msec
 5 132.2.10.10 [MPLS: Labels 10016/91012 Exp 0] 38 msec 38 msec 33 msec
 6 132.10.11.11 [MPLS: Label 91012 Exp 0] 33 msec 25 msec 21 msec
 7 10.11.13.13 29 msec * 20 msec

```

To verify the control-plane in greater details, we can check CSR1's tunnel information. Several log messages indicate that the main path via CSR4 is no longer available and that path-protection is now in use.

```

! CSR1
%MPLS_TE-5-LSP: LSP 1.1.1.1 32_35: No addresses to connect 1.1.1.1 to 4.4.4.4
%MPLS_TE-5-LSP: LSP 1.1.1.1 32_34: DOWN: path verification failed
%MPLS_TE-5-TUN: Tun32: installed LSP nil for 32_34 (popt 10), path
verification failed
%MPLS_TE-5-TUN: Tun32: LSP path change nil for 32_34, path verification
failed
%MPLS_TE-5-TUN: Tun32: installed LSP 32_33 (popt 10) for nil, Path protected
LSP failure
%MPLS_TE-5-TUN: Tun32: LSP path change 32_33 for nil, protected failure

```

The tunnel status and protection details essentially say that the backup LSP is now in use. The tunnel details are a bit redundant, but the message is clear.

```
CSR1#show mpls traffic-eng tunnels tunnel 32 | section Status
Status:
  Admin: up      Oper: up      Path: valid      Signalling: connected
  path protect option 10, type list name POPT_LIST
  Inuse path-option 100, type explicit (Basis for Protect, path weight
50)
  path option 10, type explicit EP_1_4_3_10_11
  Path Protection: Backup lsp in use.
  path protect option 10, type list name POPT_LIST
  Inuse path-option 100, type explicit (Basis for Protect, path weight
50)
```

```
CSR1#show mpls traffic-eng tunnels tunnel 32 protection
PATH PROTECTION
  LSP Head, Tunnel32, Admin: up, Oper: up
  Src 1.1.1.1, Dest 11.11.11.11, Instance 33
  Fast Reroute Protection: None
  Path Protection: Backup lsp in use.
```

RSVP is only tracking one reservation now for the backup LSP. Notice that the LSP ID for the backup path is the same, so the LSP has never been resigned, as expected.

```
CSR1#show ip rsvp reservation
To          From      Pro DPort Sport Next Hop      I/F      Fi Serv BPS
11.11.11.11 1.1.1.1 0    32    33    132.1.3.3    Gi2.513  SE LOAD 3M
```

Bringing CSR4's interface back up will cause the original path-option to recompute and ultimately preempt the backup path. At this point, both the main and backup paths are resigned (new LSP IDs) but no traffic is lost during the transition. The primary LSP is signaled first, then the traffic is moved onto it. Shortly thereafter (probably concurrently), the backup path is torn down and resigned per the "new" path option. Remember that this technique allows multiple path-options to have multiple different backup paths, and the backup paths are only used for a short time until the alternative "primary" path-options are available.

```
CSR1#show ip rsvp reservation
To          From      Pro DPort Sport Next Hop      I/F      Fi Serv BPS
11.11.11.11 1.1.1.1 0    32    42    132.1.4.4    Gi2.514  SE LOAD 0
11.11.11.11 1.1.1.1 0    32    43    132.1.3.3    Gi2.513  SE LOAD 3M
```

The feature is very similar on XR. XR is more flexible with its backup options since the backup can also be dynamic. However, XR does not support backup path-option lists, so only one backup can be specified per path-option. In this example, we re-use an old explicit-path and back it up with any blue path.

```

! XRv1
interface tunnel-te32
  description PATH PROTECTION
  ipv4 unnumbered Loopback0
  logging events all
  destination 1.1.1.1
  affinity ignore
  path-option 10 explicit name PATH_11_5_2_3_4_1 protected-by 100
  path-option 100 dynamic attribute-set ATT_BLUE

mpls traffic-eng
  attribute-set path-option ATT_BLUE
  affinity include 0x4

```

RSVP only shows one LSP, which is incorrect. There is a missing step to actually enable the feature.

```

RP/0/0/CPU0:XRv11#show rsvp reservation destination 1.1.1.1

```

Destination	Add	DPort	Source	Add	SPort	Pro	Input	IF	Sty	Serv	Rate	Burst
1.1.1.1		32	11.11.11.11		2	0	Gi0/0/0/0.551		SE	LOAD	0	1K

To actually activate the feature we need to enable it explicitly. The command to do so is not supported on XRv, but the remaining configuration will be left in place.

```

! XRv11
interface tunnel-te32
  path-protection
!!% The requested operation is not supported: Path-protection is not
supported on this platform

```

There are some additional minor options with respect to MPLS TE also. We can also specify how often the router scans for protected LSPs to "promote". This refers to selecting a better backup path when many are available. The value of 0 disables promotion, and this is the default. We will set the timer to one day (3600 seconds * 24 hours).

```

CSR1#show mpls traffic-eng tunnels brief | include Promotion
Periodic FRR Promotion: Not Running

```

```

! CSR1
mpls traffic-eng fast-reroute timers promotion 86400

```

```

CSR1#show mpls traffic-eng tunnels brief | include Promotion
Periodic FRR Promotion: every 86400 seconds, next in 86395
seconds

```

We can quickly configure this on XRv11 as well, as the syntax is very similar. There does not appear to be a mechanism to verify this configuration, though.

```
! XRv11
mpls traffic-eng
  fast-reroute timers promotion 86400
```

By default, the router does not reoptimize a tunnel when a new link in the TED comes up. This is true for newly-configured links or failed links that return to service. While it seems valuable for existing LSPs to use these new links, this can introduce churn into the network which is why it is disabled. For networks where tunnel optimization is more important than stability/flooding, we can enable this behavior globally. We will use CSR8 and XRv12 to configure these reoptimization features.

```
! CSR8
mpls traffic-eng reoptimize events link-up
```

```
! XRv12
mpls traffic-eng
  reoptimize events link-up
```

There are several reoptimization timers as well. We can delay how long an old tunnel can remain signaled before removing it after reoptimization. This might be valuable to have both tunnels active for a longer period of time in case the newly optimized tunnel fails. This is controlled with the "cleanup" timer. Additionally, we can delay the time before using the newly optimized tunnel. This is the "installation" timer and it should be less than the cleanup timer. In XE, the default cleanup timer is 10 seconds and the default installation timer is 3 seconds. This means that a newly optimized tunnel won't be used for 3 seconds, and 7 seconds later, the old tunnel is torn down. We will adjust the values slightly to 4 and 11 seconds so that the 7-second "dual tunnel" timer difference still exists. Increasing these delays can also ensure that older routers have time to program the new, optimized LSP labels into the LFIB, which may take time. If the traffic switches over to the new path too quickly, packet loss may occur. The defaults in XR are 20 seconds for delay and 20 seconds for installation, so the switchover and teardown happens at about the same time.

```
! CSR8
mpls traffic-eng reoptimize timers delay installation 4
mpls traffic-eng reoptimize timers delay cleanup 11
```

```
! XRv12
mpls traffic-eng
  reoptimize timers delay cleanup 11
  reoptimize timers delay installation 4
```

XR also supports some additional delay options. When a head-end router receives a PATHERR notifying it that FRR has occurred (from the PLR) we can delay how long to wait before reoptimizing. The default is 0 seconds, which makes sense since using FRR for extended periods of time is not optimal. Assuming path-protection was supported on XRv, we can also delay the time before trying to reoptimize a primary

path. This timer is 3 minutes (180 seconds) by default, which is very long, but since the backup path is pre-sigaled and assumed to be good, there isn't a rush. We will set it to 0 to disable it, which would imply the backup path is used forever. Last, we can delay tunnel reoptimization when an affinity failure occurs. If there isn't a path using the proper link colors, we can delay the subsequent attempts to find a path, assuming there are multiple path-options. The command is not hidden, but also does not appear to be documented.

```
! XRv12
mpls traffic-eng
  reoptimize timers delay after-frr 5
  reoptimize timers delay path-protection 0
  reoptimize timers delay after-affinity-failure 15
```

The path-protection reoptimization timer command isn't supported on XRv, but other commands are.

```
! XRv12
mpls traffic-eng
  reoptimize timers delay path-protection 0
!!% The requested operation is not supported: Path-protection is not
supported on this platform
```

Tunnels are automatically reoptimized every hour (3600 seconds) by default. We can also control this timer globally as well. A value of 0 disables periodic reoptimization, which we configure for variety. The command is significantly different on XE and XR.

```
! CSR8
mpls traffic-eng reoptimize timers frequency 0
```

```
! XRv12
mpls traffic-eng
  reoptimize 0
```

We can determine how often TE information is flooded when bandwidth changes on a per-link basis. Flooding this information too frequently can result in excessive overhead and tunnel churn. Infrequent flooding leads to outdated decision-making and bandwidth admission control. In our case, we set some new thresholds so that at 25%, 50%, and 100% of bandwidth thresholds, the TE information is reflooded. Ideally it makes sense to configure this consistently everywhere, but we will limit it to CSR3's link to CSR10 for demonstration. The default triggers for down/up are shown below. The defaults represent a more balanced model than the demonstration below and I would recommend using the default values. Regardless of these thresholds, an LSP that fails to get set up will trigger a reflood. XR has a similar configuration, and we configure this on the XRv11 link facing CSR10 for demonstration purposes.

```
! CSR3
interface GigabitEthernet2.530
```

```
mpls traffic-eng flooding thresholds up 25 50 100
mpls traffic-eng flooding thresholds down 100 50 25
```

```
! XRv11
mpls traffic-eng
interface GigabitEthernet0/0/0/0.501
  flooding thresholds up 25 50 100
  flooding thresholds down 100 50 25
```

```
CSR3#show mpls traffic-eng link-management bandwidth-allocation gig2.513 |
include Threshold
  Up Thresholds:   15 30 45 60 75 80 85 90 95 96 97 98 99 100 (default)
  Down Thresholds: 100 99 98 97 96 95 90 85 80 75 60 45 30 15 (default)
```

```
CSR3#show mpls traffic-eng link-management bandwidth-allocation gig2.530 |
include Threshold
  Up Thresholds:   25 50 100
  Down Thresholds: 100 50 25
```

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng link-management bandwidth-allocation
interface gig0/0/0/0.551 | include Threshold
  Up Thresholds   : 15 30 45 60 75 80 85 90 95 96 97 98 99 100 (default)
  Down Thresholds : 100 99 98 97 96 95 90 85 80 75 60 45 30 15 (default)
```

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng link-management bandwidth-allocation
interface gig0/0/0/0.501 | include Threshold
  Up Thresholds   : 25 50 100
  Down Thresholds : 100 50 25
```

We can also adjust some link-level parameters as well with respect to flooding and bandwidth. On both XE and XR, we can specify how long bandwidth should be held on an interface when a PATH message is sent out of it, while waiting for a RESV to come back in. If the PATH messages are lost, the bandwidth should not remain reserved forever, as the LSP will be invalid. The default is 15 seconds. We can also adjust how frequently link-state information is flooded on each link; this is different than the flooding thresholds which were bandwidth based (event-driven). This is 3 minutes (180 seconds) by default on XR and 60 seconds by default in XE, and we will disable it by using a value of 0. This is not recommend since it relies entirely on the bandwidth threshold process for updated flooding, which by definition is less accurate.

```
! CSR8
mpls traffic-eng link-management timers periodic-flooding 0
mpls traffic-eng link-management timers bandwidth-hold 16

! XRv12
mpls traffic-eng
  link-management timers bandwidth-hold 16
  link-management timers periodic-flooding 0
```



```
CSR8#show mpls traffic-eng link-management bandwidth-allocation | section
System
```

```
System Information::
```

```
Links Count:          3
Bandwidth Hold Time:  max. 16 seconds
```

```
RP/0/0/CPU0:XRv12#show mpls traffic-eng link-managemen bandwidth-allocation
System Information::
```

```
Links Count          : 4
Bandwidth Hold time  : 16 seconds
[snip]
```

```
CSR8#show mpls traffic-eng link-management summary | section IGP Area
```

```
IGP Area ID:: isis level-2
```

```
Flooding Protocol:    ISIS
Flooding Status:      data flooded
Periodic Flooding:    disabled
Flooded Links:        3
IGP System ID:        0000.0000.0008.00
MPLS TE Router ID:    8.8.8.8
Neighbors:            3
```

```
RP/0/0/CPU0:XRv12#show mpls traffic-eng link-management summary | begin level
```

```
IGP Area[1]:: IS-IS 132 level 2
```

```
Flooding Protocol    : IS-IS
Flooding Status      : flooded
Periodic Flooding    : disabled
Flooded Links        : 4
IGP System ID        : 0000.0000.0012
MPLS TE Router ID    : 12.12.12.12
IGP Neighbors        : 4
```

To support uniform and ling-pipe MPLS QoS models (described in another section), we can tell the TE process to allocate explicit-null labels on the tail end.

```
! CSR8
```

```
mpls traffic-eng signalling advertise explicit-null
```

```
! XRv12
```

```
mpls traffic-eng
```

```
signalling advertise explicit-null
```

Using tunnel30 on CSR1, which was the tunnel requesting FRR from CSR1 to CSR8, we observe the result of this command. The RRO clearly shows label 0, which is IPv4 explicit-null, as opposed to label 3, which is implicit-null. However, when we check the LFIB of CSR6, we can see it is still performing PHP.

```
CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 30
```

Reservation:

Tun Dest: 8.8.8.8 Tun ID: 30 Ext Tun ID: 1.1.1.1
Tun Sender: 1.1.1.1 LSP ID: 1
Next Hop: 132.1.3.3 on GigabitEthernet2.513
Label: 3016 (outgoing)
Reservation Style is Shared-Explicit, QoS Service is Controlled-Load
Resv ID handle: 01000427.
Average Bitrate is 3M bits/sec, Maximum Burst is 1K bytes
Min Policed Unit: 0 bytes, Max Pkt Size: 1500 bytes
RRO:

3.3.3.3/32, Flags:0x25 (Local Prot Avail/Has BW/to NHOP, Node-id)
Label subobject: Flags 0x1, C-Type 1, Label 3016
10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
Label subobject: Flags 0x1, C-Type 1, Label 10000
6.6.6.6/32, Flags:0x20 (No Local Protection, Node-id)
Label subobject: Flags 0x1, C-Type 1, Label 6003
8.8.8.8/32, Flags:0x20 (No Local Protection, Node-id)
Label subobject: Flags 0x1, C-Type 1, Label 0

Status:

Policy: Accepted. Policy source(s): MPLS/TE

CSR6#show mpls forwarding-table interface Gi2.568

Local	Outgoing	Prefix	Bytes Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched	interface	
6003	Pop Label	1.1.1.1 30 [1]	348	Gi2.568	132.6.8.8
6018	Pop Label	8.8.8.8/32	0	Gi2.568	132.6.8.8

Verifying the signaled labels, we can see that CSR8 is allocating explicit-null, but CSR6 is not interpreting it properly. Using traceroute inside the VPN and MPLS OAM, we can see that PHP is actually occurring, despite the tail-end advertising exp-null in the RESV message.

CSR8#show mpls traffic-eng tunnels source-id 30 | include Label

InLabel : GigabitEthernet2.568, explicit-null
OutLabel : -

CSR6#show mpls traffic-eng tunnels source-id 30 | include Label

InLabel : GigabitEthernet2.560, 6001
OutLabel : GigabitEthernet2.568, implicit-null

CSR1#traceroute mpls traffic-eng tunnel 30

Tracing MPLS TE Label Switched Path on Tunnel30, timeout is 2 seconds
[snip]

Type escape sequence to abort.

0 132.1.3.1 MRU 1500 [Labels: 3004 Exp: 0]
L 1 132.1.3.3 MRU 1500 [Labels: 10014 Exp: 0] 1 ms
L 2 132.3.10.10 MRU 1500 [Labels: 6009 Exp: 0] 1 ms
L 3 132.6.10.6 MRU 1500 [Labels: implicit-null Exp: 0] 10 ms
! 4 132.6.8.8 22 ms

```

CSR7#traceroute 14.14.14.14 source 7.7.7.7
Type escape sequence to abort.
Tracing the route to 14.14.14.14
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.7.1 1 msec 0 msec 1 msec
 2 132.1.3.3 [MPLS: Labels 3004/8008 Exp 0] 2 msec 2 msec 9 msec
 3 132.3.10.10 [MPLS: Labels 10014/8008 Exp 0] 36 msec 31 msec 31 msec
 4 132.6.10.6 [MPLS: Labels 6009/8008 Exp 0] 30 msec 31 msec 42 msec
 5 10.8.14.8 [MPLS: Label 8008 Exp 0] 16 msec 15 msec 21 msec
 6 10.8.14.14 18 msec * 2 msec

```

Unfortunately, the mechanism to fix this is a hidden command shown below. CSR6 needs to be told to honor the explicit-null label allocated from the tail-end. We can verify this is now working using traceroute from inside the VPN or using MPLS OAM.

```

! CSR6
mpls traffic-eng signalling interpret explicit-null verbatim

CSR6#show mpls traffic-eng tunnels source-id 30 | include Label
  InLabel   : GigabitEthernet2.560, 6002
  OutLabel  : GigabitEthernet2.568, explicit-null

```

```

CSR6#show mpls forwarding-table interface Gi2.568
Local      Outgoing   Prefix      Bytes Label  Outgoing   Next Hop
Label      Label       or Tunnel Id   Switched     interface
6002       explicit-n  1.1.1.1 30 [4]    364          Gi2.568     132.6.8.8
6018       Pop Label   8.8.8.8/32      0            Gi2.568     132.6.8.8

```

```

CSR7#traceroute 14.14.14.14 source 7.7.7.7
Type escape sequence to abort.
Tracing the route to 14.14.14.14
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.7.1 0 msec 0 msec 1 msec
 2 132.1.3.3 [MPLS: Labels 3019/8008 Exp 0] 2 msec 2 msec 9 msec
 3 132.3.10.10 [MPLS: Labels 10017/8008 Exp 0] 42 msec 31 msec 30 msec
 4 132.6.10.6 [MPLS: Labels 6002/8008 Exp 0] 37 msec 30 msec 31 msec
 5 10.8.14.8 [MPLS: Labels 0/8008 Exp 0] 16 msec 24 msec 15 msec
 6 10.8.14.8 [MPLS: Labels 0/8008 Exp 0] 16 msec 16 msec 15 msec
 7 10.8.14.14 30 msec * 2 msec

```

```

CSR1#traceroute mpls traffic-eng tunnel 30
Tracing MPLS TE Label Switched Path on Tunnel30, timeout is 2 seconds
[snip]
Type escape sequence to abort.
 0 132.1.3.1 MRU 1500 [Labels: 3019 Exp: 0]
L 1 132.1.3.3 MRU 1500 [Labels: 10017 Exp: 0] 2 ms
L 2 132.3.10.10 MRU 1500 [Labels: 6002 Exp: 0] 28 ms

```

```
L 3 132.6.10.6 MRU 1500 [Labels: explicit-null Exp: 0] 25 ms
! 4 132.6.8.8 23 ms
```

XR is a little more intelligent and automatically assumes that receiving an exp-null label in the RSVP RESV message is a good enough reason to use it. We configure XRv11 to allocate explicit-null labels with a tunnel that traverses XRv12; we will reuse tunnel2 on CSR1.

```
! XRv11
mpls traffic-eng
  signalling advertise explicit-null
```

```
RP/0/0/CPU0:XRv11#show rsvp reservation session-type lsp-p2p dst-port 2
detail | include Label
Labels: Local downstream: 0.
```

```
RP/0/0/CPU0:XRv12#show mpls traffic-eng tunnels role mid detail | include
Label
InLabel: GigabitEthernet0/0/0/0.562, 92010
OutLabel: GigabitEthernet0/0/0/0.512, explicit-null
```

```
RP/0/0/CPU0:XRv12#show mpls forwarding | include Exp
92010 Exp-Null-v4 2 Gi0/0/0/0.512 132.11.12.11 0
```

We can confirm this is operational using MPLS OAM. We cannot test it within the VPN since there isn't a steering mechanism for this tunnel, but seeing explicit-null in the label stack shows it is working correctly.

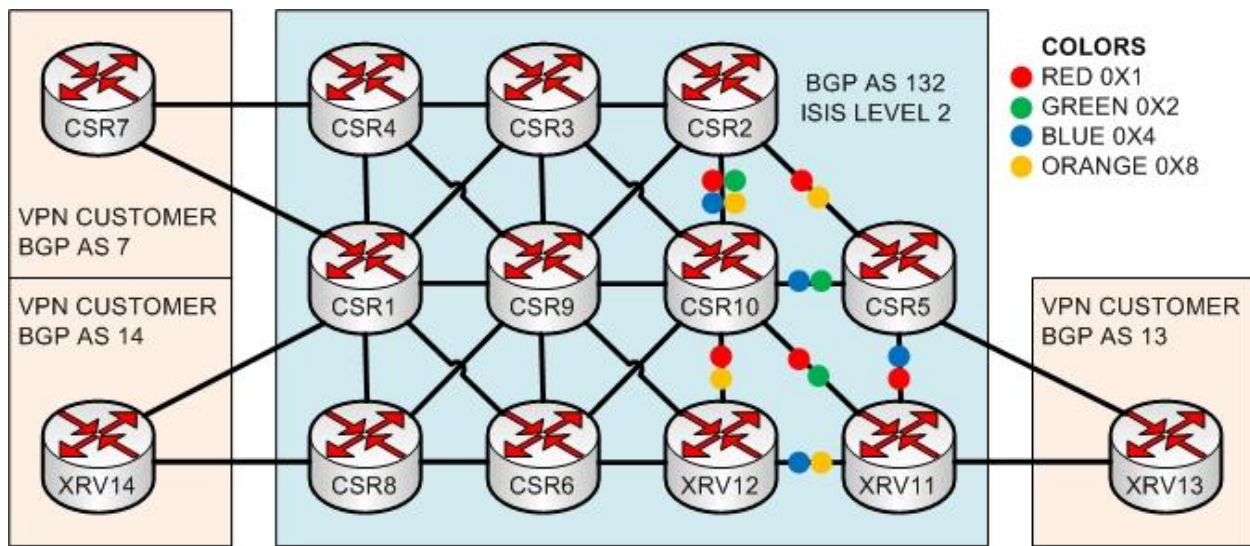
```
CSR1#traceroute mpls traffic-eng tunnel 2
Tracing MPLS TE Label Switched Path on Tunnel2, timeout is 2 seconds
[snip]
Type escape sequence to abort.
0 132.1.6.1 MRU 1500 [Labels: 6011 Exp: 0]
L 1 132.1.6.6 MRU 1500 [Labels: 92010 Exp: 0] 1 ms
L 2 132.6.12.12 MRU 1500 [Labels: explicit-null Exp: 0] 40 ms
! 3 132.11.12.11 41 ms
```

When backing up LSPs with bandwidth-protection, routers will typically try to minimize the number of LSPs pre-empted. Tunnels requesting bandwidth protection take priority over those that don't, so if a backup tunnel has no admission, non-bandwidth-protected LSPs can be pre-empted by those that are. We can alternatively configure the feature to minimize the amount of wasted bandwidth by packing each backup tunnel full with as many LSPs as possible (more efficient). More preemption may occur as a result of this increased backup bandwidth efficiency. We will enable the bandwidth optimization on CSR3.

```
CSR3(config)#mpls traffic-eng fast-reroute backup-prot-preempt ?
optimize-bw Reduce bandwidth wastage (default: minimize LSPs preempted)
```

31.2.2 Automatic tunnels (with OSPF)

This section uses a similar topology as the main TE topology with some minor changes. First, OSPF is used, mostly because it supports a feature for auto-tunnels not supported in IS-IS. It also allows us to test OSPF briefly with TE, although its behavior is generally identical to IS-IS. Additionally, many link colors have been removed for simplicity as they are mutually exclusive with some types of auto-tunnels. The diagram does not specify these uncolored links with "X"; they are simply blank.



Examining basic OSPF TE extensions, we can see there are several new LSAs inside of area 0. Specifically, there are 47 Opaque-Area (LSA-10) LSAs which carry the TE information. These are flooded only within an area, much like LSA1 and LSA2.

```
CSR1#show ip ospf 132 0 database database-summary
```

```
OSPF Router with ID (1.1.1.1) (Process ID 132)
```

```
Area 0 database summary
```

LSA Type	Count	Delete	Maxage
Router	9	0	0
Network	0	0	0
Summary Net	0	0	0
Summary ASBR	0	0	0
Type-7 Ext	0	0	0
Prefixes redistributed in Type-7	0		
Opaque Link	0	0	0
Opaque Area	47	0	0
Subtotal	56	0	0

Looking into the details of CSR1's local-originated LSAs, we see 6 different LSAs. OSPF generates one LSA10 to represent the node itself, which always has the Link ID of 1.0.0.0. The other LSA10s represent each one of CSR1's TE-enabled links, of which there are 5.

```
CSR1#show ip ospf 132 0 database self-originate
```

```
OSPF Router with ID (1.1.1.1) (Process ID 132)
```

```
Router Link States (Area 0)
```

Link ID	ADV Router	Age	Seq#	Checksum	Link count
1.1.1.1	1.1.1.1	303	0x80000005	0x00F0AE	6

```
Type-10 Opaque Link Area Link States (Area 0)
```

Link ID	ADV Router	Age	Seq#	Checksum	Opaque ID
1.0.0.0	1.1.1.1	359	0x80000001	0x0058D1	0
1.0.0.7	1.1.1.1	303	0x80000001	0x005681	7
1.0.0.8	1.1.1.1	322	0x80000001	0x0007C8	8
1.0.0.9	1.1.1.1	345	0x80000001	0x002B91	9
1.0.0.11	1.1.1.1	344	0x80000001	0x00D3DC	11
1.0.0.12	1.1.1.1	347	0x80000001	0x008424	12

To prove that the LSAs are only created for TE-enabled links, we will disable TE tunnels on the link to CSR4 (not shown). CSR1 immediately MAXAGEs the corresponding LSA10 to mark it for deletion. It is flushed seconds later.

```
CSR1#show ip ospf 132 0 database self-originate
```

```
OSPF Router with ID (1.1.1.1) (Process ID 132)
```

```
Router Link States (Area 0)
```

Link ID	ADV Router	Age	Seq#	Checksum	Link count
1.1.1.1	1.1.1.1	853	0x80000005	0x00F0AE	6

```
Type-10 Opaque Link Area Link States (Area 0)
```

Link ID	ADV Router	Age	Seq#	Checksum	Opaque ID
1.0.0.0	1.1.1.1	909	0x80000001	0x0058D1	0
1.0.0.7	1.1.1.1	853	0x80000001	0x005681	7
1.0.0.8	1.1.1.1	3608	0x80000002	0x00B77A	8
1.0.0.9	1.1.1.1	895	0x80000001	0x002B91	9
1.0.0.11	1.1.1.1	894	0x80000001	0x00D3DC	11
1.0.0.12	1.1.1.1	897	0x80000001	0x008424	12

Before continuing, we re-enable the TE configuration on the link to CSR4. The LSA10 with Link ID 1.0.0.0 is very simple, containing only the TE-RID. It does not represent a link, but only the TE node (vertex) itself. One of the remaining five LSAs is examined to show what the link-based variant contains. This shows all expected details, such as everything you'd see in an LSA-1 (link type, local/remote addresses, etc) plus the TE information. It is clearly shown in the output and needs no explanation. Also notice that the bandwidth is measured in Bps (bytes/sec), and 1.25 MBps is equivalent to 1 Gbps.

```
CSR1#show ip ospf 132 0 database opaque-area 1.0.0.0 adv-router 1.1.1.1
```

```
OSPF Router with ID (1.1.1.1) (Process ID 132)
```

```
Type-10 Opaque Link Area Link States (Area 0)
```

```
LS age: 1030
Options: (No TOS-capability, DC)
LS Type: Opaque Area Link
Link State ID: 1.0.0.0
Opaque Type: 1
Opaque ID: 0
Advertising Router: 1.1.1.1
LS Seq Number: 80000001
Checksum: 0x58D1
Length: 28
Fragment number : 0
```

```
MPLS TE router ID : 1.1.1.1
```

```
Number of Links : 0
```

```
CSR1#show ip ospf 132 0 database opaque-area 1.0.0.7 adv-router 1.1.1.1
```

```
OSPF Router with ID (1.1.1.1) (Process ID 132)
```

```
Type-10 Opaque Link Area Link States (Area 0)
```

```
LS age: 1017
Options: (No TOS-capability, DC)
LS Type: Opaque Area Link
Link State ID: 1.0.0.7
Opaque Type: 1
Opaque ID: 7
Advertising Router: 1.1.1.1
LS Seq Number: 80000001
Checksum: 0x5681
Length: 132
Fragment number : 7
```

```
Link connected to Point-to-Point network
```

```

Link ID : 3.3.3.3
Interface Address : 132.1.3.1
Neighbor Address : 132.1.3.3
Admin Metric : 1
Maximum bandwidth : 125000000
Maximum reservable bandwidth : 12500000
Number of Priority : 8
Priority 0 : 12500000    Priority 1 : 12500000
Priority 2 : 12500000    Priority 3 : 12500000
Priority 4 : 12500000    Priority 5 : 12500000
Priority 6 : 12500000    Priority 7 : 12500000
Affinity Bit : 0x0
IGP Metric : 1

```

```

Number of Links : 1

```

Like with IS-IS, the link between CSR2 and CSR3 is a broadcast network, which means a DR is elected and a corresponding LSA2 is originated by that DR. This appears slightly different to the TE topology. There aren't any new LSAs, but CSR3's "Link ID" in the LSA shows the DR address, which is CSR3. The TE topology show commands that are protocol-agnostic are still identical to IS-IS in this regard and are not shown again. The TE process can interpret this topology properly since the existing LSA1/LSA2 are still in the OSPF LSDB and available for review.

```

CSR1#show ip ospf 132 0 database opaque-area 1.0.0.7 adv-router 3.3.3.3

```

```

      OSPF Router with ID (1.1.1.1) (Process ID 132)

```

```

          Type-10 Opaque Link Area Link States (Area 0)

```

```

LS age: 51
Options: (No TOS-capability, DC)
LS Type: Opaque Area Link
Link State ID: 1.0.0.7
Opaque Type: 1
Opaque ID: 7
Advertising Router: 3.3.3.3
LS Seq Number: 80000001
Checksum: 0x7678
Length: 124
Fragment number : 7

```

```

Link connected to Broadcast network

```

```

Link ID : 132.2.3.3
Interface Address : 132.2.3.3
Admin Metric : 1
Maximum bandwidth : 125000000
Maximum reservable bandwidth : 12500000
Number of Priority : 8

```



```

Priority 0 : 12500000    Priority 1 : 12500000
Priority 2 : 12500000    Priority 3 : 12500000
Priority 4 : 12500000    Priority 5 : 12500000
Priority 6 : 12500000    Priority 7 : 12500000
Affinity Bit : 0x0
IGP Metric : 1

```

Number of Links : 1

A quick look on XRv11 shows similar output. Link ID 1.0.0.0 represents the local node, and the other 3 represent XRv11's TE-enabled links. The details of the node and a single LSA are shown below. XR supports the extended administrator group (EAG), which allows for additional affinity extensions. The EAG is an array of 32-bit unsigned integers of length 8, effectively creating 256 possible colors. Like XE, the bandwidth is still measured in Bps (it even says so in the output).

```

RP/0/0/CPU0:XRv11#show ospf 132 0 database self-originate
      OSPF Router with ID (11.11.11.11) (Process ID 132)

```

Router Link States (Area 0)

Link ID	ADV Router	Age	Seq#	Checksum	Link count
11.11.11.11	11.11.11.11	188	0x8000002b	0x006998	7

Type-10 Opaque Link Area Link States (Area 0)

Link ID	ADV Router	Age	Seq#	Checksum	Opaque ID
1.0.0.0	11.11.11.11	188	0x80000029	0x003081	0
1.0.0.6	11.11.11.11	188	0x80000029	0x0065e1	6
1.0.0.7	11.11.11.11	188	0x80000029	0x00c930	7
1.0.0.8	11.11.11.11	188	0x80000029	0x007fe4	8

```

RP/0/0/CPU0:XRv11#show ospf 132 0 database opaque-area 1.0.0.0 self-originate
      OSPF Router with ID (11.11.11.11) (Process ID 132)

```

Type-10 Opaque Link Area Link States (Area 0)

```

LS age: 252
Options: (No TOS-capability, DC)
LS Type: Opaque Area Link
Link State ID: 1.0.0.0
Opaque Type: 1
Opaque ID: 0
Advertising Router: 11.11.11.11
LS Seq Number: 80000029
Checksum: 0x3081
Length: 28

```

MPLS TE router ID : 11.11.11.11

Number of Links : 0

```
RP/0/0/CPU0:XRv11#show ospf 132 0 database opaque-area 1.0.0.6 self-originate
      OSPF Router with ID (11.11.11.11) (Process ID 132)
```

Type-10 Opaque Link Area Link States (Area 0)

```
LS age: 264
Options: (No TOS-capability, DC)
LS Type: Opaque Area Link
Link State ID: 1.0.0.6
Opaque Type: 1
Opaque ID: 6
Advertising Router: 11.11.11.11
LS Seq Number: 80000029
Checksum: 0x65e1
Length: 168
```

Link connected to Point-to-Point network

Link ID : 10.10.10.10

(all bandwidths in bytes/sec)

Interface Address : 132.10.11.11

Neighbor Address : 132.10.11.10

Admin Metric : 1

Maximum bandwidth : 125000000

Maximum reservable bandwidth global: 12500000

Number of Priority : 8

Priority 0 :	12500000	Priority 1 :	12500000
Priority 2 :	12500000	Priority 3 :	12500000
Priority 4 :	12500000	Priority 5 :	12500000
Priority 6 :	12500000	Priority 7 :	12500000

Affinity Bit : 0x3

IGP Metric : 1

Extended Administrative Group : Length: 8

EAG[0]: 0x3

EAG[1]: 0

EAG[2]: 0

EAG[3]: 0

EAG[4]: 0

EAG[5]: 0

EAG[6]: 0

EAG[7]: 0

Number of Links : 1

The first type of auto-tunnels we will examine are backup. Much like the NHOP and NNHOP tunnels we configured in previous sections, Cisco has a mechanism to automatically create these backup tunnels based on LSPs flowing through a router. The manual mechanism required a backup tunnel to be

manually configured, then assigned to an interface as a candidate backup. For a large-scale network, these backups would ideally exist to protect every link and/or every node in the SP core. Manually configuring these (even if done by a controller) can be time consuming and make the configuration very long. To test it, we will configure a basic TE tunnel from CSR1 to XRv11. We will use an explicit-path to take a long route intentionally to maximize testing. We verify the tunnel is operational with the RRO inside the RSVP RESV message.

```
! CSR1
ip explicit-path name EP_1_4_3_2_10_12_11 enable
  next-address 4.4.4.4
  next-address 3.3.3.3
  next-address 2.2.2.2
  next-address 10.10.10.10
  next-address 12.12.12.12
  next-address 11.11.11.11

interface Tunnel100
  description BASIC TE FRR (AUTO BACKUP)
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng autoroute destination
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 explicit name EP_1_4_3_2_10_12_11
  tunnel mpls traffic-eng fast-reroute
```

```
CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 100 |
begin RRO
```

```
RRO:
  4.4.4.4/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 4017
  3.3.3.3/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3016
  2.2.2.2/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 2016
  10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 10014
  12.12.12.12/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 92012
  132.10.12.12/32, Flags:0x0 (No Local Protection)
    Label subobject: Flags 0x1, C-Type 1, Label 92012
  11.11.11.11/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3
  132.11.12.11/32, Flags:0x0 (No Local Protection)
    Label subobject: Flags 0x1, C-Type 1, Label 3
Status:
Policy: Accepted. Policy source(s): MPLS/TE
```

We will enable CSR3 to provide NHOP and NNHOP protection for this LSP. Enabling "auto-tunnel backup" is done in global configuration and applies to any LSP transiting the router. First, we will enable the tunnels without any fancy options. We only need one command for this and will enable debugging to see the result. The debug is very straightforward to read since it shows the exact commands used to build the backup tunnels. We can see that the tunnel enumeration starts at 65436, which is the first number in the default range of 65436 - 65535 (100 tunnels). Logging is disabled for the tunnels and loopback0 is used for the IP address by default. The process automatically removes affinity configurations (which is odd as there are none) and relies on the default affinity of 0x0 / mask 0xFFFF. So far, this is OK, since most of the links are colorless. The explicit-path that is generated for tunnel65436 is supposed to avoid using CSR3's link to CSR2; this is an NHOP tunnel. The process repeats again for tunnel65437, which is identical in every way except is configured to avoid CSR2 by Node-ID; this is an NNHOP tunnel. Auto-tunnel backup will always try to make NHOP and NNHOP tunnels by default for each LSP it protects.

```
! CSR3
mpls traffic-eng auto-tunnel backup

CSR3#debug mpls traffic-eng auto-tunnel backup all
TE_AUTO_TUN: FRR lopri process rcv msg TSPTUN_BACKUP_RPC_REMOVE_ALL, ready=1
TE_AUTO_TUN: Delaying further auto-tunnel backup configuration.
TE_AUTO_TUN: FRR lopri process rcv msg TSPTUN_CREATE_BACKUP, ready=1
TE_AUTO_TUN: Create Auto tunnel
TE_AUTO_TUN: backup CLI command:
interface tunnel65436
no logging event link-status
ip unnumbered Loopback0
tunnel destination 2.2.2.2
tunnel mode mpls traffic-eng
end

TE_AUTO_TUN: backup CLI command:
interface tunnel65436
no tunnel mpls traffic-eng affinity
end

TE_AUTO_TUN: CLI command:
ip explicit-path name __dynamic_tunnel65436
index 1 exclude-address 132.2.3.3

TE_AUTO_TUN: backup CLI command:
interface tunnel65436
tunnel mpls traffic-eng path-option 1 exp name __dynamic_tunnel65436
end

TE_AUTO_TUN: Create Auto tunnel
TE_AUTO_TUN: backup CLI command:
interface tunnel65437
```

```
no logging event link-status
ip unnumbered Loopback0
tunnel destination 10.10.10.10
tunnel mode mpls traffic-eng
end
```

```
TE_AUTO_TUN: backup CLI command:
interface tunnel65437
no tunnel mpls traffic-eng affinity
end
```

```
TE_AUTO_TUN: CLI command:
ip explicit-path name __dynamic_tunnel65437
index 1 exclude-address 2.2.2.2
```

```
TE_AUTO_TUN: backup CLI command:
interface tunnel65437
tunnel mpls traffic-eng path-option 1 exp name __dynamic_tunnel65437
end
```

The log messages below indicate that both tunnels are operational. Checking the RSVP PATH message summary, we can see that CSR3 is the head-end for both tunnels since there is no PHOP.

```
! CSR3
%MPLS_TE-5-TUN: Tun65436: installed LSP 65436_1 (popt 1) for nil, got 1st
feasible path opt
%MPLS_TE-5-TUN: Tun65437: installed LSP 65437_1 (popt 1) for nil, got 1st
feasible path opt
%MPLS_TE-5-LSP: LSP 3.3.3.3 65437_1: UP
%MPLS_TE-5-TUN: Tun65437: LSP path change 65437_1 for nil, normal
%MPLS_TE-5-LSP: LSP 3.3.3.3 65436_1: UP
%MPLS_TE-5-TUN: Tun65436: LSP path change 65436_1 for nil, normal
```

```
CSR3#show ip rsvp sender filter session-type 7
```

Destination	Tun Sender	TunID	LSPID	Prev Hop	I/F	BPS
2.2.2.2	3.3.3.3	65436	1	none	none	0
10.10.10.10	3.3.3.3	65437	1	none	none	0
11.11.11.11	1.1.1.1	100	2	132.3.4.4	Gi2.534	0

There is also a specific show command to see the backup auto-tunnel details. Later, we will look at adjusting these parameters.

```
CSR3#show mpls traffic-eng auto-tunnel backup
State: Enabled
Auto backup tunnels: 1 (up: 1, down: 0)
Tunnel ID Range: 65436 - 65535
Create Nhop Only: No
Check for deletion of unused tunnels every: 3600 Sec
```

SRLG: Not configured

Config:

unnumbered-interface: Loopback0
Affinity/Mask: 0x0/0xFFFF

The PATH details show the EROs used to build these tunnels. The NHOP tunnel uses CSR9 as an alternate path to CSR3 while the NNHOP tunnel just goes directly to CSR10, since that is a shorter path.

```
CSR3#show ip rsvp sender detail filter session-type 7 tunnel-id 65436 |  
section outgoing  
ERO: (outgoing)  
132.3.9.9 (Strict IPv4 Prefix, 8 bytes, /32)  
132.2.9.2 (Strict IPv4 Prefix, 8 bytes, /32)  
2.2.2.2 (Strict IPv4 Prefix, 8 bytes, /32)
```

```
CSR3#show ip rsvp sender detail filter session-type 7 tunnel-id 65437 |  
section outgoing  
ERO: (outgoing)  
132.3.10.10 (Strict IPv4 Prefix, 8 bytes, /32)  
10.10.10.10 (Strict IPv4 Prefix, 8 bytes, /32)
```

The protected LSP uses label 3016 when sending traffic to CSR3. Checking the FRR database against this label, we see an entry showing that the NNHOP tunnel is currently available (but not in use) and using label 10014. We can confirm this update in the original RSVP RESV message on the headend. NNHOP is preferred over NHOP when both are available and all other parameters are equal (backup bandwidth, etc). All labels show in the FRR database are present in the RESV RRO of the protected LSP.

```
CSR3#show mpls traffic-eng fast-reroute database labels 3016 detail | begin  
100  
Tun ID: 100, LSP ID: 2, Source: 1.1.1.1  
Destination: 11.11.11.11  
State : ready  
InLabel : 3016  
OutLabel : Gi2.523:2016  
FRR OutLabel : Tu65437:10014
```

```
CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 100 |  
begin RRO  
RRO:  
4.4.4.4/32, Flags:0x20 (No Local Protection, Node-id)  
Label subobject: Flags 0x1, C-Type 1, Label 4017  
3.3.3.3/32, Flags:0x29 (Local Prot Avail/to NNHOP, Node-id)  
Label subobject: Flags 0x1, C-Type 1, Label 3016  
2.2.2.2/32, Flags:0x20 (No Local Protection, Node-id)  
Label subobject: Flags 0x1, C-Type 1, Label 2016  
10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
```

Label subobject: Flags 0x1, C-Type 1, Label 10014

[snip]

To quickly test the NNHOP backup tunnel, we will break the link to CSR2 by changing the encapsulation on the link to CSR3 (not shown). The auto-tunnel process is just used for provisioning; everything else is just normal TE-FRR. CSR1 receives a PATHERR message from CSR4 that the tunnel was locally repaired. Because there were no other path-options available, the tunnel cannot presently be reoptimized around the failure, and traffic will remain in the TE-FRR tunnel.

```
! CSR1
%MPLS_TE-5-LSP: LSP 1.1.1.1 100_2: Path Error from 132.3.4.3: Notify: Tunnel
locally repaired (flags 0)
%MPLS_TE-5-LSP: LSP 1.1.1.1 100_3: No addresses to connect 3.3.3.3 to 2.2.2.2
```

We verify that the PLR shows the NNHOP tunnel as active for the protected LSP and now swaps label 3016 for label 10014. Notice that a label push need not occur since the PLR and MP are directly connected; the PLR learned label 10014 from the RESV RRO so from CSR10's perspective, the LFIB has not changed.

```
CSR3#show mpls traffic-eng fast-reroute database labels 3016 detail | begin
100
```

```
Tun ID: 100, LSP ID: 2, Source: 1.1.1.1
Destination: 11.11.11.11
```

```
State      : active
InLabel    : 3016
OutLabel   : Gi2.523:2016
FRR OutLabel : Tu65437:10014
```

```
CSR3#show mpls forwarding-table labels 3016 detail
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Outgoing interface	Next Hop
3016	10014	1.1.1.1 100 [2]	3232	Tu65437	point2point
MAC/Encaps=18/22, MRU=1500, Label Stack{10014}, via Gi2.530					
000C29497164000C29D781FE81000DCA8847 0271E000					
No output feature configured					

The headend, of course, knows TE-FRR is used as it generated a log message earlier. The RSVP RESV message also shows this as well.

```
CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 100 |
begin RRO
```

```
RRO:
 4.4.4.4/32, Flags:0x20 (No Local Protection, Node-id)
   Label subobject: Flags 0x1, C-Type 1, Label 4017
 3.3.3.3/32, Flags:0x2B (Local Prot Avail/In Use/to NNHOP, Node-id)
```

```
Label subobject: Flags 0x1, C-Type 1, Label 3016
10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
Label subobject: Flags 0x1, C-Type 1, Label 10014
[snip]
```

To reduce the number of tunnels required, we can configure the auto-tunnel process to only create NHOP tunnels. These may scale better; if there were many LSPs going from CSR2 to CSR3, but then from CSR3 to many different next-hops, many NNHOP tunnels would be required, when a single NHOP tunnel could at least protect a commonly used link. Debugging shows the auto-tunnel process deleting both tunnels from the configuration, which is followed by several syslog messages to indicate it. Notice that the original NHOP tunnel is then resigaled as a result of the configuration change. Tunnel 65337 is shutdown (NNHOP) and tunnel65436 is signaled (NHOP).

```
! CSR3
mpls traffic-eng auto-tunnel backup nhop-only
```

```
TE_AUTO_TUN: backup CLI command:
no ip explicit-path name __dynamic_tunnel65436
end
```

```
TE_AUTO_TUN: backup CLI command:
no ip explicit-path name __dynamic_tunnel65437
end
```

```
TE_AUTO_TUN: FRR lopri process rcv msg TSPTUN_CREATE_BACKUP, ready=1
TE_AUTO_TUN: Create Auto tunnel
TE_AUTO_TUN: backup CLI command:
interface tunnel65436
no logging event link-status
ip unnumbered Loopback0
tunnel destination 2.2.2.2
tunnel mode mpls traffic-eng
end
```

```
TE_AUTO_TUN: backup CLI command:
interface tunnel65436
no tunnel mpls traffic-eng affinity
end
```

```
TE_AUTO_TUN: CLI command:
ip explicit-path name __dynamic_tunnel65436
index 1 exclude-address 132.2.3.3
```

```
TE_AUTO_TUN: backup CLI command:
interface tunnel65436
tunnel mpls traffic-eng path-option 1 exp name __dynamic_tunnel65436
end
```



```
%MPLS_TE-5-TUN: Tun65436: installed LSP 65436_1 (popt 1) for nil, got 1st
feasible path opt
%MPLS_TE-5-LSP: LSP 3.3.3.3 65437_1: DOWN: signalling shutdown
%MPLS_TE-5-TUN: Tun65437: installed LSP nil for 65437_1 (popt 1), signalling
shutdown
%MPLS_TE-5-TUN: Tun65437: LSP path change nil for 65437_1, signalling
shutdown
%MPLS_TE-5-LSP: LSP 3.3.3.3 65436_1: UP
%MPLS_TE-5-TUN: Tun65436: LSP path change 65436_1 for nil, normal
```

Now, we can see that the “Nhop Only” flag has changed to yes in the auto-tunnel backup summary.

```
CSR3#show mpls traffic-eng auto-tunnel backup
State: Enabled
  Auto backup tunnels: 1 (up: 1, down: 0)
  Tunnel ID Range: 65436 - 65535
  Create Nhop Only: Yes
  Check for deletion of unused tunnels every: 3600 Sec
  SRLG: Not configured

Config:
  unnumbered-interface: Loopback0
  Affinity/Mask: 0x0/0xFFFF
```

We check the RSVP RESV RRO on the head-end to see if the labels have changed on the original LSP (it has been signaled a few times). The RRO also shows that NHOP protection is available. The local label for CSR3 is now 3015, so we check the TE-FRR database on CSR3 to ensure there is an entry for it. We can tell this is an NHOP tunnel since the FRR label is the same as the outlabel, which indicates the NHOP label is persistent. The PLR will just push a new label in the path to the MP.

```
CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 100 |
begin RRO
RRO:
  4.4.4.4/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 4016
  3.3.3.3/32, Flags:0x21 (Local Prot Avail/to NHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3015
  2.2.2.2/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 2015
[snip]
```

```
CSR3#show mpls traffic-eng fast-reroute database labels 3015 detail | begin
100
Tun ID: 100, LSP ID: 14, Source: 1.1.1.1
Destination: 11.11.11.11
State      : ready
InLabel    : 3015
OutLabel   : Gi2.523:2015
```

```
FRR OutLabel : Tu65436:2015
```

To verify the NHOP tunnel path, we verify CSR3's tunnel details. Again, it selects CSR9, who allocates label 9015 for the backup tunnel.

```
CSR3#show mpls traffic-eng tunnels tunnel 65436 | section RSVP Path
  RSVP Path Info:
    My Address: 132.3.9.3
    Explicit Route: 132.3.9.9 132.2.9.2 2.2.2.2
    Record Route: NONE
    Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits
```

```
CSR3#show mpls traffic-eng tunnels tunnel 65436 | include Label
InLabel : -
OutLabel : GigabitEthernet2.539, 9015
```

We simulate a failure again by bringing CSR2's interface down facing CSR3. CSR1 shows the normal log messages to indicate it.

```
! CSR1
%MPLS_TE-5-LSP: LSP 1.1.1.1 100_15: No addresses to connect 3.3.3.3 to
2.2.2.2
%MPLS_TE-5-LSP: LSP 1.1.1.1 100_14: Path Error from 132.3.4.3: Notify: Tunnel
locally repaired (flags 0)
```

We check CSR3's FRR database to see the state is now active with tunnel65436 being used, which is NHOP protection. The LFIB shows a swap from 3015 to 2015, as would normally be done, but then a push of label 9015 which is used to tunnel the LSP to the MP (CSR2).

```
CSR3#show mpls traffic-eng fast-reroute database labels 3015 detail | begin
100
Tun ID: 100, LSP ID: 14, Source: 1.1.1.1
Destination: 11.11.11.11
State : active
InLabel : 3015
OutLabel : Gi2.523:2015
FRR OutLabel : Tu65436:2015
```

```
CSR3#show mpls forwarding-table labels 3015 detail
Local      Outgoing   Prefix          Bytes Label   Outgoing   Next Hop
Label      Label      or Tunnel Id    Switched      interface
3015       2015       1.1.1.1 100 [14] 1565          Tu65436     point2point
MAC/Encaps=18/26, MRU=1496, Label Stack{9015 2015}, via Gi2.539
000C29E04F84000C29D781FE81000DD38847 02337000007DF000
No output feature configured
```

CSR1 shows that the NHOP tunnel is now in use, showing that FRR is still working.

```

CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 100 |
begin RRO
RRO:
  4.4.4.4/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 4016
  3.3.3.3/32, Flags:0x23 (Local Prot Avail/In Use/to NHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3015
  2.2.2.2/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 2015

```

The concept of shared risk link group (SRLG) is similar to SRLG seen in the IP FRR environment. We can specify links to "share risk" which tells the auto-tunnel process to not prefer, or totally discredit, these links as candidate paths for FRR. This is also called "fate sharing". For example, let's assume that CSR3's links to CSR9 and CSR2 both use the same physical conduit to carry the cabling. If the link to CSR2 fails due to the conduit being damaged, it is highly likely the link to CSR9 will also fail, so we should not prefer that link if possible. By assigning these groups to the same SRLG, which is carried in the TE extensions inside OSPF Type-10 LSAs or IS-IS LSPs, the auto-tunnel process will consider it. The command to enable SRLG for MPLS-TE is different than the one used for IP FRR.

```

! CSR3
interface GigabitEthernet2.523
  mpls traffic-eng srlg 29

interface GigabitEthernet2.539
  mpls traffic-eng srlg 29

```

After configuring it, we will verify it locally on CSR3 by checking the raw OSPF LSAs and the TE topology. Normally you would go straight to the TE topology instead of the raw IGP information. The last octet of the OSPF LSA10 (7 and 10 in this case) correspond to the frag_id listed in the TED. This eliminates the guesswork when querying the LSDP for TE information.

```

CSR3#show ip ospf 132 0 database opaque-area 1.0.0.7 adv-router 3.3.3.3
[snip]
Link connected to Broadcast network
Link ID : 132.2.3.3
Interface Address : 132.2.3.3
[snip]
Shared Risk Link Groups:29

```

```

CSR3#show ip ospf 132 0 database opaque-area 1.0.0.10 adv-router 3.3.3.3
[snip]
Link connected to Point-to-Point network
Link ID : 9.9.9.9
Interface Address : 132.3.9.3
[snip]

```

Shared Risk Link Groups:29

```
CSR3#show mpls traffic-eng topology 3.3.3.3 brief
IGP Id: 3.3.3.3, MPLS TE Id:3.3.3.3 Router Node  (ospf 132  area 0)
[snip]
link[3]: Point-to-Point, Nbr IGP Id: 9.9.9.9, nbr_node_id:7, gen:72
frag_id: 10, Intf Address: 132.3.9.3, Nbr Intf Address: 132.3.9.9
TE metric: 1, IGP metric: 1, attribute flags: 0x0
SRLGs: 29

link[4]: Broadcast, DR: 132.2.3.3, nbr_node_id:14, gen:72
frag_id: 7, Intf Address: 132.2.3.3
TE metric: 1, IGP metric: 1, attribute flags: 0x0
SRLGs: 29
```

Next, we need to tell the auto-tunnel process how to interpret the SRLG. We can either demand that the SRLG never be used, which means that if an SRLG path is the only path, no tunnel will be made. We can be more liberal and prefer that SRLG not be used, which essentially creates multiple path-options to fall back to SRLG paths as a last resort. First, we look at the strict option. As soon as we configure this feature, assuming debugging is on, we can see the original NHOP tunnel be torn down and a new one created with a new explicit-path. The tunnel configuration itself is identical, but the explicit-path now contains "exclude-srlg". This options is not available in the normal explicit-path configuration on XE as SRLGs are specific to auto-backup tunnels.

```
! CSR3
mpls traffic-eng auto-tunnel backup srlg exclude force

! CSR3
TE_AUTO_TUN: FRR lopri process rcv msg TSPTUN_BACKUP_RPC_REMOVE_ALL, ready=1
%MPLS_TE-5-LSP: LSP 3.3.3.3 65436_1: DOWN: signalling shutdown
%MPLS_TE-5-TUN: Tun65436: installed LSP nil for 65436_1 (popt 1), signalling
shutdown
%MPLS_TE-5-TUN: Tun65436: LSP path change nil for 65436_1, signalling
shutdown
TE_AUTO_TUN: backup CLI command:
no ip explicit-path name __dynamic_tunnel65436
end

TE_AUTO_TUN: FRR lopri process rcv msg TSPTUN_CREATE_BACKUP, ready=1
TE_AUTO_TUN: Create Auto tunnel
TE_AUTO_TUN: backup CLI command:
interface tunnel65436
no logging event link-status
ip unnumbered Loopback0
tunnel destination 2.2.2.2
tunnel mode mpls traffic-eng
end
```

```
TE_AUTO_TUN: backup CLI command:
interface tunnel65436
no tunnel mpls traffic-eng affinity
end
```

```
TE_AUTO_TUN: CLI command:
ip explicit-path name __dynamic_tunnel65436
  index 1 exclude-address 132.2.3.3
  index 2 exclude-srlg 132.2.3.3
```

```
TE_AUTO_TUN: backup CLI command:
interface tunnel65436
tunnel mpls traffic-eng path-option 1 exp name __dynamic_tunnel65436
end
```

```
%MPLS_TE-5-TUN: Tun65436: installed LSP 65436_1 (popt 1) for nil, got 1st
feasible path opt
```

```
%MPLS_TE-5-LSP: LSP 3.3.3.3 65436_1: UP
```

```
%MPLS_TE-5-TUN: Tun65436: LSP path change 65436_1 for nil, normal
```

The tunnel still comes up, but avoids the link to CSR9 (SRLG) while also ignoring the link to CSR2 (NHOP protection). Specifically, the path routes via CSR10 > CSR9 > CSR2 to protect the direct link from CSR3 to CSR2 and remaining compliant with the SRLG policy. Technically, this is a valid NHOP-tunnel, despite transiting the NNHOP router (CSR10) along the way. Notice that the SRLG exclusion method is "forced".

```
CSR3#show mpls traffic-eng auto-tunnel backup
```

```
State: Enabled
```

```
Auto backup tunnels: 1 (up: 1, down: 0)
```

```
Tunnel ID Range: 65436 - 65535
```

```
Create Nhop Only: Yes
```

```
Check for deletion of unused tunnels every: 3600 Sec
```

```
SRLG Exclude: Forced
```

```
Config:
```

```
unnumbered-interface: Loopback0
```

```
Affinity/Mask: 0x0/0xFFFF
```

```
CSR3#show mpls traffic-eng tunnels tunnel 65436 | section RSVP Path
```

```
RSVP Path Info:
```

```
My Address: 132.3.10.3
```

```
Explicit Route: 132.3.10.10 132.9.10.9 132.2.9.2 2.2.2.2
```

```
Record Route: NONE
```

```
Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits
```

CSR3 records this NHOP tunnel as candidate to backup the primary LSP from CSR1 to XRv11. As expected, the PLR (CSR3) informs the headend (CSR1) that NHOP protection is available.

```
CSR3#show mpls traffic-eng fast-reroute database labels 3016 detail | begin 100
```

```
Tun ID: 100, LSP ID: 25, Source: 1.1.1.1
```

```
Destination: 11.11.11.11
```

```
State : ready
```

```
InLabel : 3016
```

```
OutLabel : Gi2.523:2016
```

```
FRR OutLabel : Tu65436:2016
```

```
CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 100 | begin RRO
```

```
RRO:
```

```
4.4.4.4/32, Flags:0x20 (No Local Protection, Node-id)
```

```
Label subobject: Flags 0x1, C-Type 1, Label 4017
```

```
3.3.3.3/32, Flags:0x21 (Local Prot Avail/to NHOP, Node-id)
```

```
Label subobject: Flags 0x1, C-Type 1, Label 3016
```

```
2.2.2.2/32, Flags:0x20 (No Local Protection, Node-id)
```

```
Label subobject: Flags 0x1, C-Type 1, Label 2016
```

```
[snip]
```

Breaking the link between CSR2 and CSR3, the NHOP tunnel transitions to the active state. The LFIB reflects the corresponding swap and push operations as discussed earlier.

```
! CSR1
```

```
%MPLS_TE-5-LSP: LSP 1.1.1.1 100_26: No addresses to connect 3.3.3.3 to 2.2.2.2
```

```
%MPLS_TE-5-LSP: LSP 1.1.1.1 100_25: Path Error from 132.3.4.3: Notify: Tunnel locally repaired (flags 0)
```

```
CSR3#show mpls traffic-eng fast-reroute database labels 3016 detail | begin 100
```

```
Tun ID: 100, LSP ID: 25, Source: 1.1.1.1
```

```
Destination: 11.11.11.11
```

```
State : active
```

```
InLabel : 3016
```

```
OutLabel : Gi2.523:2016
```

```
FRR OutLabel : Tu65436:2016
```

```
CSR3#show mpls forwarding-table labels 3016 detail
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Label	Outgoing interface	Next Hop
3016	2016	1.1.1.1 100 [25]	654		Tu65436	point2point
MAC/Encaps=18/26, MRU=1496, Label Stack{10015 2016}, via Gi2.530						
000C29497164000C29D781FE81000DCA8847 0271F000007E0000						
No output feature configured						

We can verify this LSP using MPLS OAM. Notice that CSR10 appears twice in the path. Ideally, the backup tunnels won't traverse nodes in the protected LSP, but this is technically acceptable.

```
CSR1#traceroute mpls traffic-eng tunnel 100
Tracing MPLS TE Label Switched Path on Tunnel100, timeout is 2 seconds
[snip]
```

Type escape sequence to abort.

```
0 132.1.4.1 MRU 1500 [Labels: 4017 Exp: 0]
L 1 132.1.4.4 MRU 1500 [Labels: 3016 Exp: 0] 2 ms
L 2 132.3.4.3 MRU 1500 [Labels: 10015/2016 Exp: 0/0] 28 ms
L 3 132.3.10.10 MRU 1500 [Labels: 9014/2016 Exp: 0/0] 28 ms
L 4 132.9.10.9 MRU 1500 [Labels: 2016 Exp: 0] 33 ms
L 5 132.2.9.2 MRU 1500 [Labels: 10014 Exp: 0] 33 ms
L 6 132.2.10.10 MRU 1500 [Labels: 92012 Exp: 0] 25 ms
L 7 132.10.12.12 MRU 1500 [Labels: implicit-null Exp: 0] 25 ms
! 8 132.11.12.11 37 ms
```

Next, we will analyze the other SRLG option, which is to prefer non-SRLG paths. The "force" option only created a single path-option that excluded the link address and SRLG (Boolean AND logic). The "preferred" option uses the same first path-option as the "force" option, which means it first tries to find a path that satisfies both conditions. The second path-option is more relaxed and represents a regular NHOP tunnel with no SRLG restrictions as a fallback mechanism. To make the test more interesting, we will break the links between CSR3 and CSR10, as well as CSR3 and CSR1. This means that CSR3 will likely route back via CSR4 to make the NHOP tunnel to CSR2, creating a microloop. This is the same behavior we would have seen with the force option as well, since the path-option sequence still makes it very unlikely that an SRLG path will be selected, as desired.

```
! CSR3
mpls traffic-eng auto-tunnel backup srlg exclude preferred
```

```
! CSR3
TE_AUTO_TUN: FRR lopri process rcv msg TSPTUN_BACKUP_RPC_REMOVE_ALL, ready=1
%MPLS_TE-5-LSP: LSP 3.3.3.3 65436_1: DOWN: signalling shutdown
%MPLS_TE-5-TUN: Tun65436: installed LSP nil for 65436_1 (popt 1), signalling
shutdown
%MPLS_TE-5-TUN: Tun65436: LSP path change nil for 65436_1, signalling
shutdown
TE_AUTO_TUN: backup CLI command:
no ip explicit-path name __dynamic_tunnel65436
end
```

```
TE_AUTO_TUN: FRR lopri process rcv msg TSPTUN_CREATE_BACKUP, ready=1
TE_AUTO_TUN: Create Auto tunnel
TE_AUTO_TUN: backup CLI command:
interface tunnel65436
no logging event link-status
ip unnumbered Loopback0
tunnel destination 2.2.2.2
```

```
tunnel mode mpls traffic-eng
end
```

```
TE_AUTO_TUN: backup CLI command:
interface tunnel65436
no tunnel mpls traffic-eng affinity
end
```

```
TE_AUTO_TUN: CLI command:
ip explicit-path name __dynamic_tunnel65436
  index 1 exclude-address 132.2.3.3
  index 2 exclude-srlg 132.2.3.3
```

```
TE_AUTO_TUN: CLI command:
ip explicit-path name __dynamic_tunnel65436_pathopt2
  index 1 exclude-address 132.2.3.3
```

```
TE_AUTO_TUN: backup CLI command:
interface tunnel65436
tunnel mpls traffic-eng path-option 1 exp name __dynamic_tunnel65436
end
```

```
TE_AUTO_TUN: backup CLI command:
interface tunnel65436
tunnel mpls traffic-eng path-option 2 exp name __dynamic_tunnel65436_pathopt2
end
```

```
%MPLS_TE-5-LSP: LSP 3.3.3.3 65436 3: UP
%MPLS_TE-5-TUN: Tun65436: LSP path change 65436_3 for nil, reoptimization
```

We can also verify the configuration without using debugs, shown below. The SRLG exclusion mode is now "preferred". The ERO for the particular tunnel is shown also; noticed that the path routes via CSR4 > CSR9 > CSR2.

```
CSR3#show mpls traffic-eng auto-tunnel backup
State: Enabled
  Auto backup tunnels: 1 (up: 1, down: 0)
  Tunnel ID Range: 65436 - 65535
  Create Nhop Only: Yes
  Check for deletion of unused tunnels every: 3600 Sec
  SRLG Exclude: Preferred

Config:
  unnumbered-interface: Loopback0
  Affinity/Mask: 0x0/0xFFFF

CSR3#show mpls traffic-eng tunnels tunnel 65436 | section RSVP Path
RSVP Path Info:
  My Address: 132.3.4.3
  Explicit Route: 132.3.4.4 132.4.9.9 132.2.9.2 2.2.2.2
```



```
Record Route: NONE
Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits
```

We perform the basic checks for verification. CSR1 is aware that NHOP protection is available because the PLR is tracking it in its FRR database. The local label for CSR3 (for the protected path) has changed to 3015 since the LSP was signaled after the last test.

```
CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 100 |
begin RRO
```

```
RRO:
  4.4.4.4/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 4016
  3.3.3.3/32, Flags:0x21 (Local Prot Avail/to NHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3015
  2.2.2.2/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 2015
[snip]
```

```
CSR3#show mpls traffic-eng fast-reroute database labels 3015 detail | begin
100
```

```
Tun ID: 100, LSP ID: 36, Source: 1.1.1.1
Destination: 11.11.11.11
State       : ready
InLabel     : 3015
OutLabel    : Gi2.523:2015
FRR OutLabel : Tu65436:2015
```

Breaking the link between CSR2 and CSR3, we can see that the NHOP protection still works despite it being suboptimal.

```
! CSR1
%MPLS_TE-5-LSP: LSP 1.1.1.1 100_37: No addresses to connect 3.3.3.3 to
2.2.2.2
%MPLS TE-5-LSP: LSP 1.1.1.1 100_36: Path Error from 132.3.4.3: Notify: Tunnel
locally repaired (flags 0)
```

```
CSR1#show ip rsvp reservation detail filter session-type 7 tunnel-id 100 |
begin RRO
```

```
RRO:
  4.4.4.4/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 4016
  3.3.3.3/32, Flags:0x23 (Local Prot Avail/In Use/to NHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3015
  2.2.2.2/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 2015
[snip]
```

```
CSR3#show mpls traffic-eng fast-reroute database labels 3015 detail | begin 100
```

```
Tun ID: 100, LSP ID: 36, Source: 1.1.1.1
Destination: 11.11.11.11
```

```
State      : active
InLabel    : 3015
OutLabel   : Gi2.523:2015
FRR OutLabel : Tu65436:2015
```

```
CSR3#show mpls forwarding-table labels 3015 detail
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Label	Outgoing interface	Next Hop
3015	2015	1.1.1.1 100 [36]	1102		Tu65436	point2point
MAC/Encaps=18/26, MRU=1496, Label Stack{4017 2015}, via Gi2.534						
000C29D8E5F5000C29D781FE81000DCE8847 00FB1000007DF000						
No output feature configured						

Testing with MPLS traceroute, we can see the traffic immediately routes back to CSR4 using a different label for the FRR tunnel. We still avoid the protected link as well as the SRLGs.

```
CSR1#traceroute mpls traffic-eng tunnel 100
```

```
Tracing MPLS TE Label Switched Path on Tunnel100, timeout is 2 seconds
```

```
[snip]
```

```
Type escape sequence to abort.
```

```
0 132.1.4.1 MRU 1500 [Labels: 4016 Exp: 0]
L 1 132.1.4.4 MRU 1500 [Labels: 3015 Exp: 0] 1 ms
L 2 132.3.4.3 MRU 1500 [Labels: 4017/2015 Exp: 0/0] 28 ms
L 3 132.3.4.4 MRU 1500 [Labels: 9016/2015 Exp: 0/0] 26 ms
L 4 132.4.9.9 MRU 1500 [Labels: 2015 Exp: 0] 22 ms
L 5 132.2.9.2 MRU 1500 [Labels: 10013 Exp: 0] 22 ms
L 6 132.2.10.10 MRU 1500 [Labels: 92013 Exp: 0] 28 ms
L 7 132.10.12.12 MRU 1500 [Labels: implicit-null Exp: 0] 22 ms
! 8 132.11.12.11 39 ms
```

We can enable this feature on CSR10 as well to test additional minor features. CSR10 will be configured to provide NHOP and NNHOP protection using paths that are blue. The tunnel-IDs of the automatically created backup tunnels will be in the range of 1500 and 1505 and tunnels not in use (that is, not bound to an LSP requesting FRR) should be removed after every minute, building in a 10 second buffer before removal.

```
! CSR10
```

```
mpls traffic-eng auto-tunnel backup timers removal unused 50 10
mpls traffic-eng auto-tunnel backup tunnel-num min 1500 max 1505
mpls traffic-eng auto-tunnel backup config affinity 0x4 mask 0x4
```

When using an aggressive scan interval, the parser warns you about router performance. Scanning every 50 seconds to discover unused tunnels, then removing them 10 seconds later, meets the 1-minute

removal requirement. We can verify the basic configuration was accepted as well. Notice that only one tunnel is shown when we expected to see two.

```
CSR10(config)#mpls traffic-eng auto-tunnel backup timers removal unused 50 10
```

AUTO TUNNEL: Note that a scan interval of less than 60 seconds, may result in high CPU utilization.

```
CSR10#show mpls traffic-eng auto-tunnel backup
State: Enabled
  Auto backup tunnels: 1 (up: 1, down: 0)
  Tunnel ID Range: 1500 - 1505
  Create Nhop Only: No
  Check for deletion of unused tunnels every: 50 Sec
  SRLG: Not configured

Config:
  unnumbered-interface: Loopback0
  Affinity/Mask: 0x4/0x4
```

Checking the path of the tunnel, we can see that it takes a relatively long path via CSR5 > XRv11 > XRv12. The tunnel even traverses the NNHOP, which is expected since this is a different LSP than the main one (used for FRR only). Checking the tunnel labels to determine the incoming label on the tunnel transiting CSR10 (midpoint router), we see label 10015, as well as the FRR tunnel. The FRR database confirms that tunnel1500 is protecting the original LSP. The reason the NHOP tunnel did not get created is that the NHOP path transits the NNHOP node, so using NHOP protection makes no sense. The auto-tunnel feature is smart enough to account for this case and will conserve resources.

```
CSR10#show ip rsvp sender detail filter session-type 7 tunnel-id 1501 |
section outgoing
ERO: (outgoing)
  132.5.10.5 (Strict IPv4 Prefix, 8 bytes, /32)
  132.5.11.11 (Strict IPv4 Prefix, 8 bytes, /32)
  11.11.11.11 (Strict IPv4 Prefix, 8 bytes, /32)
```

```
CSR10#show mpls traffic-eng tunnels role middle | include Label
InLabel   : GigabitEthernet2.520, 10015
OutLabel  : GigabitEthernet2.502, 92012
FRR OutLabel : Tunnel1501, implicit-null
```

```
CSR10#show mpls traffic-eng fast-reroute database labels 10015 detail | begin
100
Tun ID: 100, LSP ID: 47, Source: 1.1.1.1
Destination: 11.11.11.11
State      : ready
InLabel    : 10015
OutLabel   : Gi2.502:92012
```

```
FRR OutLabel : Tu1501:implicit-null
```

Since XRv does not support BFD, FRR is not going to work well, so we won't test the data plane on this tunnel. Testing the feature briefly on XRv, we will create a tunnel using explicit-paths from XRv11 to CSR1. the tunnel will request FRR and XRv12 will create the auto-tunnels. The configuration in XR is much more limited than XE at the global level. The only affinity option is to ignore-all, there is no option for NHOP-only, and there is only one timer for removal timer (scan time only, unused tunnels are removed immediately). Additional options are available at the link level, such as NHOP-only, SRLG exclusion, and tunnel attribute-sets. For variety, we will build NHOP and NNHOP tunnels to protect the LSPs traversing the XRv12-CSR10 link and CSR10 as a node. We will use the strict SRLG option to help engineer the backup tunnels. SRLGs configured on XR are similar to IP FRR, unlike XE which has different mechanisms between IP and TE FRR.

```
! XRv11
interface tunnel-te100
  description BASIC TE FRR (AUTO BACKUP)
  ipv4 unnumbered Loopback0
  logging events all
  destination 1.1.1.1
  fast-reroute
  affinity ignore
  path-option 10 explicit name EP_11_12_6_10_3_1

! XRv12
mpls traffic-eng
interface GigabitEthernet0/0/0/0.562
  auto-tunnel backup
  exclude srlg
auto-tunnel backup
  timers removal unused 0
  tunnel-id min 1200 max 1210
  affinity ignore

srlg
interface GigabitEthernet0/0/0/0.502
  8 value 100
interface GigabitEthernet0/0/0/0.512
  8 value 100
interface GigabitEthernet0/0/0/0.562
  8 value 100
```

We can verify the configuration was successful by checking the TE topology's view of all SRLGs, as well as basic auto-tunnel show command. XR provides very detailed information where XE provides only a summary. We can also specify the SRLG value, IGP PID with area/level, and other filters. Notice that XRv12 can see the SRLGs we configured earlier since these are carried in the TE topology. Using

consistent SRLG values network-wide is important since they have global significance. Of note, we see that the tunnels are down for some reason.

```
RP/0/0/CPU0:XRv12#show mpls traffic-eng topology srlg ospf 132 area 0
```

SRLG	Admin Weight	Interface Addr	TE Router ID	IGP Area	ID
29	1	132.3.9.3	3.3.3.3	OSPF 132	area 0
29	1	132.2.3.3	3.3.3.3	OSPF 132	area 0
100	1	132.6.12.12	12.12.12.12	OSPF 132	area 0
100	1	132.11.12.12	12.12.12.12	OSPF 132	area 0
100	1	132.10.12.12	12.12.12.12	OSPF 132	area 0

```
RP/0/0/CPU0:XRv12#show mpls traffic-eng auto-tunnel backup
```

AutoTunnel Backup Configuration:

Interfaces count: 2

Unused removal timeout: 0s

Configured tunnel number range: 1200-1210

AutoTunnel Backup Summary:

AutoTunnel Backups:

2 created, 0 up, 2 down, 2 unused

1 NHOP, 1 NNHOP, 0 SRLG strict, 0 SRLG preferred, 0 SRLG

weighted

Protected LSPs:

0 NHOP, 0 NHOP+SRLG

0 NNHOP, 0 NNHOP+SRLG

Protected S2L Sharing Families:

0 NHOP, 0 NHOP+SRLG

0 NNHOP, 0 NNHOP+SRLG

Protected S2Ls:

0 NHOP, 0 NHOP+SRLG

0 NNHOP, 0 NNHOP+SRLG

Cumulative Counters (last cleared 20:04:07 ago):

	Total	NHOP	NNHOP
Created:	2	1	1
Connected:	2	1	1
Removed (down):	0	0	0
Removed (unused):	0	0	0
Removed (in use):	0	0	0
Range exceeded:	0	0	0

AutoTunnel Backups:

Tunnel Name	State	Protection Offered	Prot. Flows*	Protected Interface	Protected Node
tunnel-te1200	down	NNHOP	0	Gi0/0/0/0.562	6.6.6.6
tunnel-te1201	down	NHOP	0	Gi0/0/0/0.562	N/A

Looking into the details of tunnel1200 as an example, we see several auto-tunnel backup parameters, as well as the reason for the tunnel failure. We need to specify the IPv4 unnumbered address somehow, but there wasn't an option under the MPLS-TE stanza. Instead, we use a global command to specify that loopback0 should be used by default, an assumption not made by XR. Finding the problem is easy, as shown below, but knowing the command to fix it can be difficult.

```
RP/0/0/CPU0:XRv12#show mpls traffic-eng tunnels 1200 | begin Auto Backup
Auto Backup:
  Protected LSPs: 0
  Protected S2L Sharing Families: 0
  Protected S2L: 0
  Protected i/f: Gi0/0/0/0.562          Protected node: 6.6.6.6
  Attribute-set: Not configured
  Protection: NNHOP
  Unused removal timeout: not running
  Reason for the tunnel being down: No IP source address is configured
Displayed 1 (of 2) heads, 0 (of 2) midpoints, 0 (of 0) tails
Displayed 0 up, 1 down, 0 recovering, 0 recovered heads

! XRv12
ipv4 unnumbered mpls traffic-eng loopback0
```

Now, the tunnels comes up. Tunnel1207 is NNHOP and tunnel1208 is NHOP, both of which are operational and also offer SRLG protection. I've cleared the tunnels several times so their numbers are now 1207 and 1208. The summary view shows that the NNHOP tunnel is protecting one LSP at present. The output also specifies that two of the tunnels are operating in SRLG "strict" mode, which is like "force" mode in XE.

```
RP/0/0/CPU0:XRv12#show mpls traffic-eng auto-tunnel backup
AutoTunnel Backup Configuration:
  Interfaces count: 1
  Unused removal timeout: 0s
  Configured tunnel number range: 1200-1210

AutoTunnel Backup Summary:
  AutoTunnel Backups:
    2 created, 2 up, 0 down, 1 unused
    1 NHOP, 1 NNHOP, 2 SRLG strict, 0 SRLG preferred, 0 SRLG
weighted
  Protected LSPs:
    0 NHOP, 0 NHOP+SRLG
    0 NNHOP, 1 NNHOP+SRLG
  Protected S2L Sharing Families:
    0 NHOP, 0 NHOP+SRLG
    0 NNHOP, 0 NNHOP+SRLG
  Protected S2Ls:
    0 NHOP, 0 NHOP+SRLG
```

0 NNHOP, 0 NNHOP+SRLG

Cumulative Counters (last cleared 21:59:16 ago):

	Total	NHOP	NNHOP
Created:	9	5	4
Connected:	35	12	23
Removed (down):	5	3	2
Removed (unused):	1	1	0
Removed (in use):	1	0	1
Range exceeded:	0	0	0

AutoTunnel Backups:

Tunnel Name	State	Protection Offered	Prot. Flows*	Protected Interface	Protected Node
tunnel-te1207	up	NNHOP+SRLG	1	Gi0/0/0/0.562	6.6.6.6
tunnel-te1208	up	NHOP+SRLG	0	Gi0/0/0/0.562	N/A

*Prot. Flows = Total Protected LSPs, S2Ls and S2L Sharing Families

Looking into the NNHOP tunnel details, we can see that we have protected the link to CSR6 as well as CSR6 itself as a node. The SRLG dictates we cannot use the direct link to CSR10, which would have been convenient and a shorter path. Instead, we use CSR9 for NNHOP protection. We can see this tunnel is being used since it is protecting an LSP.

RP/0/0/CPU0:XRv12#show mpls traffic-eng tunnels 1207 | begin Auto Backup

Auto Backup:

Protected LSPs: 1

Protected S2L Sharing Families: 0

Protected S2L: 0

Protected i/f: Gi0/0/0/0.562 Protected node: 6.6.6.6

Attribute-set: Not configured

Protection: NNHOP+SRLG (SRLG strict)

Unused removal timeout: not running

Path info (OSPF 132 area 0):

Node hop count: 2

Hop0: 132.9.12.9

Hop1: 132.9.10.10

Hop2: 10.10.10.10

The NHOP tunnel is also up and transiting CSR9, but is not protecting the node 6.6.6.6 (CSR6). NNHOP is preferred over NHOP when both are available and all other things are considered equal. This second tunnel protects an interface (a link) only.

RP/0/0/CPU0:XRv12#show mpls traffic-eng tunnels 1208 | begin Auto Backup

Auto Backup:

Protected LSPs: 0

```
Protected S2L Sharing Families: 0
Protected S2L: 0
Protected i/f: Gi0/0/0/0.562
Attribute-set: Not configured
Protection: NHOP+SRLG (SRLG strict)
Unused removal timeout: not running
```

```
Path info (OSPF 132 area 0):
Node hop count: 2
Hop0: 132.9.12.9
Hop1: 132.6.9.6
Hop2: 6.6.6.6
```

We can quickly validate the backup LSPs using MPLS OAM. Notice the different labels in use, which we verify by checking the tunnel details. These two outputs represent the NNHOP and NHOP tunnels respectively.

```
RP/0/0/CPU0:XRv12#traceroute mpls traffic-eng tunnel-te 1207
Tracing MPLS TE Label Switched Path on tunnel-te1207, timeout is 2 seconds
[snip]
Type escape sequence to abort.
```

```
 0 132.9.12.12 MRU 1500 [Labels: 9017 Exp: 0]
L 1 132.9.12.9 MRU 1500 [Labels: implicit-null Exp: 0] 0 ms
! 2 132.9.10.10 10 ms
```

```
RP/0/0/CPU0:XRv12#traceroute mpls traffic-eng tunnel-te 1208
Tracing MPLS TE Label Switched Path on tunnel-te1208, timeout is 2 seconds
[snip]
Type escape sequence to abort.
```

```
 0 132.9.12.12 MRU 1500 [Labels: 9020 Exp: 0]
L 1 132.9.12.9 MRU 1500 [Labels: implicit-null Exp: 0] 10 ms
! 2 132.6.9.6 1 ms
```

```
RP/0/0/CPU0:XRv12#show mpls traffic-eng tunnels 1207 detail | include Label
Outgoing Interface: GigabitEthernet0/0/0/0.592, Outgoing Label: 9017
```

```
RP/0/0/CPU0:XRv12#show mpls traffic-eng tunnels 1208 detail | include Label
Outgoing Interface: GigabitEthernet0/0/0/0.592, Outgoing Label: 9020
```

Checking the FRR database confirms that the original tunnel is protected with NNHOP support. We can tell because the NNHOP label 10013 is used (local label on CSR10) and the tunnel ID is 1207. The headend RSVP RESV RRO also shows protection available with the "node" flag, implying NNHOP protection.

```
RP/0/0/CPU0:XRv12#show mpls traffic-eng fast-reroute database
```


LSP midpoint FRR information:

LSP Identifier	Local Label	Out Intf/Label	FRR Intf/Label	Status
11.11.11.11 100 [2]	92012	Gi0/0/0/0.562:6014	tt1207:10013	Ready

RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 100 detail | begin Resv

Resv Info:

Record Route:

IPv4 12.12.12.12, flags 0x29 (Node-ID, Protection: available, node)

Label 92012, flags 0x1

IPv4 132.11.12.12, flags 0x9 (Protection: available, node)

Label 92012, flags 0x1

IPv4 6.6.6.6, flags 0x20 (Node-ID)

Label 6014, flags 0x1

IPv4 10.10.10.10, flags 0x20 (Node-ID)

Label 10013, flags 0x1

IPv4 3.3.3.3, flags 0x21 (Node-ID, Protection: available)

Label 3016, flags 0x1

IPv4 1.1.1.1, flags 0x20 (Node-ID)

Label 3, flags 0x1

Fspec: avg rate=0 kbits, burst=1000 bytes, peak rate=0 kbits

To build only NHOP tunnels, we configure this on a per-interface basis in XR. We will apply this to XRv12 so that the NHOP tunnel is used. Because we did not enable debug earlier, we will do it quickly now to watch XRv12 tear down the existing NNHOP tunnel without touching the NHOP tunnel. The debug essentially verifies NHOP protection and disables NNHOP protection.

! XRv12

mpls traffic-eng

interface GigabitEthernet0/0/0/0.562

auto-tunnel backup

nhop-only

! XRv12

te_control[1044]: DBG-AUTOBKP[1]: te_verify_mpls_te_auto_backup:2999: Verify:

Adding auto-tunnel backup on I/F GigabitEthernet0_0_0_0.562

te_control[1044]: DBG-AUTOBKP[1]:

te_verify_mpls_te_auto_backup_nhop_only:3138: Verify: Adding auto-tunnel

backup NHOP only on I/F GigabitEthernet0_0_0_0.562

te_control[1044]: DBG-AUTOBKP[1]: te_apply_mpls_te_auto_backup:3272: Apply:

Adding auto-tunnel backup on I/F GigabitEthernet0_0_0_0.562

te_control[1044]: DBG-AUTOBKP[1]: te_frr_autobackup_enable:2505: protected

interface GigabitEthernet0_0_0_0.562 already present

te_control[1044]: DBG-AUTOBKP[1]: te_frr_autobackup_enable:2510: protected

interface GigabitEthernet0_0_0_0.562 already configured

```

te_control[1044]: DBG-AUTOBKP[1]:
te_apply_mpls_te_auto_backup_nhop_only:3371: Apply: Adding auto-tunnel backup
NHOP only on I/F GigabitEthernet0_0_0_0.562
te_control[1044]: DBG-AUTOBKP[1]: te_frr_autobackup_set_cfg:2663: removing
NNHOP backups on GigabitEthernet0_0_0_0.562
te_control[1044]: DBG-AUTOBKP[1]: te_frr_autobackup_clear_intf:1408: checking
GigabitEthernet0_0_0_0.562, backup tunnel 1207
te_control[1044]: DBG-AUTOBKP[1]:
te_frr_autobackup_start_unused_pruning_timer:583: Pruning timer is disabled.
Return.
te_control[1044]: DBG-AUTOBKP[1]: te_frr_autobackup_remove:1236: Auto-tunnel
backup 1207 removed
te_control[1044]: DBG-AUTOBKP[1]: te_frr_autobackup_clear_intf:1408: checking
GigabitEthernet0_0_0_0.562, backup tunnel 1208
te_control[1044]: DBG-AUTOBKP[1]:
te_frr_autobackup_stop_unused_pruning_timer:461: pruning timer to stop is not
running for bkup 1208
te_control[1044]: DBG-AUTOBKP[1]: te_frr_autobackup_need_to_create_nnhop:1701
(Tunnel ID: 100): NNHOP not configured GigabitEthernet0_0_0_0.562
te_control[1044]: DBG-AUTOBKP[1]: te_frr_autobackup_need_to_create_nhop:1622
(Tunnel ID: 100): already nhop protected GigabitEthernet0_0_0_0.562

```

Now, we can see that the NHOP tunnel is protecting the LSP since NNHOP protection is not available. We confirm this by checking the FRR database as well. The head-end also shows that protection is available, but does not specify node protection, so NHOP protection is implied.

```
RP/0/0/CPU0:XRv12#show mpls traffic-eng tunnels 1208 | begin Auto Backup
```

```

Auto Backup:
  Protected LSPs: 1
  Protected S2L Sharing Families: 0
  Protected S2L: 0
  Protected i/f: Gi0/0/0/0.562
  Attribute-set: Not configured
  Protection: NHOP+SRLG (SRLG strict)
  Unused removal timeout: not running

```

```

Path info (OSPF 132 area 0):
Node hop count: 2
Hop0: 132.9.12.9
Hop1: 132.6.9.6
Hop2: 6.6.6.6

```

```
RP/0/0/CPU0:XRv12#show mpls traffic-eng fast-reroute database
```

```
LSP midpoint FRR information:
```

LSP Identifier	Local Label	Out Intf/ Label	FRR Intf/ Label	Status
11.11.11.11 100 [2]	92012	Gi0/0/0/0.562:6014	tt1208:6014	Ready

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 100 detail | begin Resv
Resv Info:
Record Route:
  IPv4 12.12.12.12, flags 0x21 (Node-ID, Protection: available)
  Label 92012, flags 0x1
  IPv4 132.11.12.12, flags 0x1 (Protection: available)
  Label 92012, flags 0x1
  IPv4 6.6.6.6, flags 0x20 (Node-ID)
  Label 6014, flags 0x1
  IPv4 10.10.10.10, flags 0x20 (Node-ID)
  Label 10013, flags 0x1
  IPv4 3.3.3.3, flags 0x21 (Node-ID, Protection: available)
  Label 3016, flags 0x1
  IPv4 1.1.1.1, flags 0x20 (Node-ID)
  Label 3, flags 0x1
Fspec: avg rate=0 kbits, burst=1000 bytes, peak rate=0 kbits
```

XE supports primary auto-tunnels as well, which creates a one-hop tunnel to every directly connected peer reachable through a TE-enabled interface. There is no way to ignore affinity for these tunnels, since link-coloring doesn't make sense in conjunction with this feature. That is why I removed most of the link colors from the XE-dense parts of the network. We will enable this feature on CSR8 using a single command and with debugging enabled. Like auto-backup, these tunnel commands are shown clearly in the debug. Unlike auto-backup, the default range of tunnels is 65336 to 65435 to avoid overlap. The tunnels also have logging enabled, so we will see their statuses. Each tunnel is built to the one-hop neighbors using an explicit-path to specify using the interface towards that neighbor (these neighbor addresses are interpreted from the TED). Most importantly, each tunnel requests FRR protection and is configured with autoroute announce. Now, traffic to each peer is routed inside of this one-hop tunnel and is FRR-protected with a single command.

```
! CSR8
mpls traffic-eng auto-tunnel primary onehop

CSR8#debug mpls traffic-eng auto-tunnel primary all
TE_AUTO_TUN: Found a new router id 1.1.1.1 off of GigabitEthernet2.518
TE_AUTO_TUN: Create Auto tunnel
TE_AUTO_TUN: primary CLI command:
interface tunnel65336
no logging event link-status
ip unnumbered Loopback0
tunnel destination 1.1.1.1
tunnel mode mpls traffic-eng
end

TE_AUTO_TUN: primary CLI command:
interface tunnel65336
tunnel mpls traffic-eng autoroute announce
```

```
tunnel mpls traffic-eng fast-reroute
end
```

TE_AUTO_TUN: CLI command:

```
ip explicit-path name __dynamic_tunnel65336
  index 1 next-address 132.1.8.8
```

TE_AUTO_TUN: primary CLI command:

```
interface tunnel65336
tunnel mpls traffic-eng path-option 1 exp name __dynamic_tunnel65336
end
```

TE_AUTO_TUN: Found Down Tunnel65336 to router id 1.1.1.1 out
GigabitEthernet2.518

TE_AUTO_TUN: Found a new router id 6.6.6.6 off of GigabitEthernet2.568

TE_AUTO_TUN: Create Auto tunnel

TE_AUTO_TUN: primary CLI command:

```
interface tunnel65337
no logging event link-status
ip unnumbered Loopback0
tunnel destination 6.6.6.6
tunnel mode mpls traffic-eng
end
```

TE_AUTO_TUN: primary CLI command:

```
interface tunnel65337
tunnel mpls traffic-eng autoroute announce
tunnel mpls traffic-eng fast-reroute
end
```

TE_AUTO_TUN: CLI command:

```
ip explicit-path name __dynamic_tunnel65337
  index 1 next-address 132.6.8.8
```

TE_AUTO_TUN: primary CLI command:

```
interface tunnel65337
tunnel mpls traffic-eng path-option 1 exp name __dynamic_tunnel65337
end
```

TE_AUTO_TUN: Found Down Tunnel65337 to router id 6.6.6.6 out
GigabitEthernet2.568

TE_AUTO_TUN: Found a new router id 9.9.9.9 off of GigabitEthernet2.589

TE_AUTO_TUN: Create Auto tunnel

TE_AUTO_TUN: primary CLI command:

```
interface tunnel65338
no logging event link-status
ip unnumbered Loopback0
tunnel destination 9.9.9.9
tunnel mode mpls traffic-eng
```

end

```
TE_AUTO_TUN: primary CLI command:
interface tunnel65338
tunnel mpls traffic-eng autoroute announce
tunnel mpls traffic-eng fast-reroute
end
```

```
TE_AUTO_TUN: CLI command:
ip explicit-path name __dynamic_tunnel65338
  index 1 next-address 132.8.9.8
```

```
TE_AUTO_TUN: primary CLI command:
interface tunnel65338
tunnel mpls traffic-eng path-option 1 exp name __dynamic_tunnel65338
end
```

```
TE_AUTO_TUN: Found Down Tunnel65338 to router id 9.9.9.9 out
GigabitEthernet2.589
%MPLS_TE-5-TUN: Tun65336: installed LSP 65336_1261 (popt 1) for nil, got 1st
feasible path opt
%MPLS_TE-5-TUN: Tun65337: installed LSP 65337_793 (popt 1) for nil, got 1st
feasible path opt
%MPLS_TE-5-TUN: Tun65338: installed LSP 65338_4178 (popt 1) for nil, got 1st
feasible path opt
%MPLS_TE-5-LSP: LSP 8.8.8.8 65337 793: UP
%MPLS_TE-5-TUN: Tun65337: LSP path change 65337_793 for nil, normal
%MPLS_TE-5-LSP: LSP 8.8.8.8 65336 1261: UP
%MPLS_TE-5-TUN: Tun65336: LSP path change 65336_1261 for nil, normal
%MPLS_TE-5-LSP: LSP 8.8.8.8 65338 4178: UP
%MPLS_TE-5-TUN: Tun65338: LSP path change 65338_4178 for nil, normal
```

We can quickly check the auto-primary configuration. The output below seems to indicate we can change the tunnel ID range, enabled LDP on the tunnel (for stitching PE-P and P-P tunnels), and change the unnumbered address. There is no ability to adjust affinities, as described earlier.

```
CSR8#show mpls traffic-eng auto-tunnel primary
```

```
State: Enabled
Auto primary tunnels: 3 (up: 3, down: 0)
Tunnel ID Range: 65336 - 65435
Check for deletion of FRR Active onehop tunnels every:0 Sec

Config:
unnumbered-interface: Loopback0
mpls ip: FALSE
```

We can also verify the important aspects of this feature: autoroute and FRR protection. Autoroute shows 3 different destinations, one per tunnel, as expected. The routing table shows the destinations to each router also reachable through the tunnel.

```
CSR8#show mpls traffic-eng autoroute
MPLS TE autorouting enabled
destination 1.1.1.1, area ospf 132 area 0, has 1 tunnels
  Tunnel65336 (load balancing metric 0, nexthop 1.1.1.1)
    (flags: Announce)
destination 6.6.6.6, area ospf 132 area 0, has 1 tunnels
  Tunnel65337 (load balancing metric 0, nexthop 6.6.6.6)
    (flags: Announce)
destination 9.9.9.9, area ospf 132 area 0, has 1 tunnels
  Tunnel65338 (load balancing metric 0, nexthop 9.9.9.9)
    (flags: Announce)

CSR8#show ip route 1.1.1.1
Routing entry for 1.1.1.1/32
  Known via "ospf 132", distance 110, metric 2, type intra area
  Last update from 1.1.1.1 on Tunnel65336, 00:09:21 ago
  Routing Descriptor Blocks:
  * 1.1.1.1, from 1.1.1.1, 00:09:21 ago, via Tunnel65336
    Route metric is 2, traffic share count is 1

CSR8#show ip route 6.6.6.6
Routing entry for 6.6.6.6/32
  Known via "ospf 132", distance 110, metric 2, type intra area
  Last update from 6.6.6.6 on Tunnel65337, 00:09:27 ago
  Routing Descriptor Blocks:
  * 6.6.6.6, from 6.6.6.6, 00:09:27 ago, via Tunnel65337
    Route metric is 2, traffic share count is 1

CSR8#show ip route 9.9.9.9
Routing entry for 9.9.9.9/32
  Known via "ospf 132", distance 110, metric 2, type intra area
  Last update from 9.9.9.9 on Tunnel65338, 00:09:30 ago
  Routing Descriptor Blocks:
  * 9.9.9.9, from 9.9.9.9, 00:09:30 ago, via Tunnel65338
    Route metric is 2, traffic share count is 1
```

The RSVP RESV RRO is very short since the tunnel is onehop, and the label is implicit-null, but FRR has been requested. Primary one-hop tunnels, whether automatically or manually configured, always use a null label of sorts.

```
CSR8#show ip rsvp reservation detail filter session-type 7 | section include
Tun_Dest|RRO
Tun_Dest: 1.1.1.1 Tun ID: 65336 Ext Tun ID: 8.8.8.8
RRO:
```

```

1.1.1.1/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 3
Tun Dest: 6.6.6.6 Tun ID: 65337 Ext Tun ID: 8.8.8.8
RRO:
6.6.6.6/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 3
Tun Dest: 9.9.9.9 Tun ID: 65338 Ext Tun ID: 8.8.8.8
RRO:
9.9.9.9/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 3

```

Despite this seemingly harmless command, the network is very broken. Because of autoroute announce, CSR8 is now routing to all remote destination through a TE tunnel that runs PE-P. VPN traffic will no longer work since the transport label is implicit-null. The VPN label is exposed to P routers way too soon, and the VPN traffic will be dropped. We need a way to learn labels from the tail-end routers (at the other end of the one-hop tunnels) for the LSP endpoint.

```

CSR8#show bgp vpnv4 unicast vrf TE 13.13.13.13/32
BGP routing table entry for 132:1:13.13.13.13/32, version 18
Paths: (1 available, best #1, table TE)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  13
    11.11.11.11 (metric 4) (via default) from 11.11.11.11 (11.11.11.11)
      Origin IGP, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:132:1
      mpls labels in/out nolabel/91013
      rx pathid: 0, tx pathid: 0x0

```

```

CSR8#show ip route 11.11.11.11
Routing entry for 11.11.11.11/32
  Known via "ospf 132", distance 110, metric 4, type intra area
  Last update from 9.9.9.9 on Tunnel65338, 00:12:33 ago
  Routing Descriptor Blocks:
    9.9.9.9, from 11.11.11.11, 00:12:33 ago, via Tunnel65338
      Route metric is 4, traffic share count is 1
    * 6.6.6.6, from 11.11.11.11, 00:12:33 ago, via Tunnel65337
      Route metric is 4, traffic share count is 1

```

```

CSR8#show mpls traffic-eng tunnels tunnel 65337 | include Label
InLabel : -
OutLabel : GigabitEthernet2.568, implicit-null

```

```

CSR8#show mpls traffic-eng tunnels tunnel 65338 | include Label
InLabel : -
OutLabel : GigabitEthernet2.589, implicit-null

```

```

RP/0/0/CPU0:XRv14#traceroute 13.13.13.13 source 14.14.14.14
Type escape sequence to abort.
Tracing the route to 13.13.13.13

 1  10.8.14.8 0 msec  0 msec  0 msec
 2  * * *
 3  [snip]

```

We can solve this problem by enabling LDP on the primary auto-tunnels. We assume the P routers will accept the LDP targeted sessions, and if they don't, we would need to enable that also (not shown). Issuing this command immediately updates all of the tunnels with the "mpls ip" command, as expected. The "mpls ip" flag in the show command also changes to TRUE to indicate it is enabled. Last, we verify the LDP targeted sessions are working properly. In the vast majority of auto-primary deployments, this command would be necessary unless all of the PEs are directly connected.

```

! CSR8
mpls traffic-eng auto-tunnel primary config mpls ip

! CSR8
TE_AUTO_TUN: FRR lopri process rcv msg TSPTUN_ONEHOP_SET_MPLS_IP_ALL, ready=1
TE_AUTO_TUN: primary CLI command:
interface tunnel65336
mpls ip
end

TE_AUTO_TUN: Tunnel65336 not a candidate for primary
TE_AUTO_TUN: primary CLI command:
interface tunnel65337
mpls ip
end

TE_AUTO_TUN: Tunnel65337 not a candidate for primary
TE_AUTO_TUN: primary CLI command:
interface tunnel65338
mpls ip
end

CSR8#show mpls traffic-eng auto-tunnel primary

State: Enabled
  Auto primary tunnels: 3 (up: 3, down: 0)
  Tunnel ID Range: 65336 - 65435
  Check for deletion of FRR Active onehop tunnels every:0 Sec

Config:
  unnumbered-interface: Loopback0
  mpls ip: TRUE

```



```

CSR8#show mpls ldp discovery | begin Target
Tunnel65336 (ldp): Targeted -> 1.1.1.1
Tunnel65337 (ldp): Targeted -> 6.6.6.6
Tunnel65338 (ldp): Targeted -> 9.9.9.9
Targeted Hellos:
  8.8.8.8 -> 1.1.1.1 (ldp): active, xmit/recv
    LDP Id: 1.1.1.1:0
  8.8.8.8 -> 6.6.6.6 (ldp): active, xmit/recv
    LDP Id: 6.6.6.6:0
  8.8.8.8 -> 9.9.9.9 (ldp): active, xmit/recv
    LDP Id: 9.9.9.9:0

```

Now, traffic flows as expected. We now have LDP label bindings for XRv11's loopback. Technically an "implicit-null" would be on the top of the stack if it were a real label, followed by the LDP label. Tracing to CSR1 wouldn't have required LDP on the tunnel since the auto-tunnel was PE-PE, so only the VPN label is used. The labels below are the LDP-learned labels for 11.11.11.11/32 from CSR6 and CSR9.

```

CSR8#show ip cef 11.11.11.11
11.11.11.11/32
  nexthop 6.6.6.6 Tunnel65337 label 6011
  nexthop 9.9.9.9 Tunnel65338 label 9011

```

```

RP/0/0/CPU0:XRv14#traceroute 13.13.13.13 source 14.14.14.14
Type escape sequence to abort.
Tracing the route to 13.13.13.13

```

```

 1  10.8.14.8 0 msec  0 msec  0 msec
 2  132.8.9.9 [MPLS: Labels 9011/91013 Exp 0] 49 msec  39 msec  39 msec
 3  132.9.12.12 [MPLS: Labels 92007/91013 Exp 0] 29 msec  69 msec  59 msec
 4  132.11.12.11 [MPLS: Label 91013 Exp 0] 59 msec  59 msec  29 msec
 5  10.11.13.13 39 msec  *    29 msec

```

```

RP/0/0/CPU0:XRv14#traceroute 7.7.7.7 source 14.14.14.14
Type escape sequence to abort.
Tracing the route to 7.7.7.7

```

```

 1  10.8.14.8 0 msec  0 msec  0 msec
 2  10.1.7.1 [MPLS: Label 1008 Exp 0] 0 msec  0 msec  0 msec
 3  10.1.7.7 0 msec  *    0 msec

```

This feature works well with auto-tunnel backup. Since we already have a bunch of one-hop primary tunnels requesting FRR, we can back them up with NHOP auto-backups. After enabling these, we can see three new tunnels were created to backup each of the existing primary tunnels. Based on the labels received, two of them use CSR9 and one of them uses CSR1 for NHOP protection. Pay close attention to the tunnel IDs; the protected and FRR tunnels appear to be the same at a glance, but the third digit is

different. It is coincidental that the auto-primary and auto-backup interface numbers happen to align as they did. The “protected” tunnels are primary and the FRR interfaces are backup.

```
! CSR8
mpls traffic-eng auto-tunnel backup nhop-only

CSR8#show mpls traffic-eng fast-reroute database
P2P Headend FRR information:
Protected tunnel      In-label Out intf/label  FRR intf/label  Status
-----
Tunnel65336          Tun hd   Gi2.518:implicit Tu65436:implicit ready
Tunnel65337          Tun hd   Gi2.568:implicit Tu65437:implicit ready
Tunnel65338          Tun hd   Gi2.589:implicit Tu65438:implicit ready
```

```
CSR8#show mpls traffic-eng tunnels tunnel 65436 | include Label
InLabel   : -
OutLabel  : GigabitEthernet2.589, 9015
```

```
CSR8#show mpls traffic-eng tunnels tunnel 65437 | include Label
InLabel   : -
OutLabel  : GigabitEthernet2.589, 9009
```

```
CSR8#show mpls traffic-eng tunnels tunnel 65438 | include Label
InLabel   : -
OutLabel  : GigabitEthernet2.518, 1012
```

We can also adjust the removal timer for these tunnels. After a link failure, the tunnels can be removed from the configuration if desired. By default, the feature is disabled (tunnels are never removed).

```
! CSR8
mpls traffic-eng auto-tunnel primary timers removal rerouted 3600

CSR8#show mpls traffic-eng auto-tunnel primary
State: Enabled
Auto primary tunnels: 3 (up: 3, down: 0)
Tunnel ID Range: 65336 - 65435
Check for deletion of FRR Active onhop tunnels every:3600 Sec

Config:
unnumbered-interface: Loopback0
mpls ip: TRUE
```

Although not clearly documented nor well-advertised, XR supports this feature as well. It is hidden within the "mesh" confines, but it does work the same way. Since XRv12 already has auto-tunnel backup configured, we will enable the one-hop tunnels there.

```
! XRv12
```

```

mpls traffic-eng
  auto-tunnel mesh
  group 999
    onehop
  tunnel-id min 9990 max 9999

```

Checking the status of these tunnels, we can see that XRv12 created 4 tunnels, one for each next-hop. Only two of them are up, but all four are marked with a % sign to indicate "onehop" tunnels.

```

RP/0/0/CPU0:XRv12#show mpls traffic-eng tunnels brief

```

TUNNEL NAME	DESTINATION	STATUS	STATE
*tunnel-te1208	6.6.6.6	up	up
%tunnel-te9990	132.6.12.6	up	up
%tunnel-te9991	132.9.12.9	up	up
%tunnel-te9992	132.10.12.10	down	down
%tunnel-te9993	132.11.12.11	down	down
CSR6_t65338	12.12.12.12	up	up
CSR6_t65439	9.9.9.9	up	up
CSR6_t65440	12.12.12.12	up	up

```

* = automatically created backup tunnel
% = automatically created mesh onehop tunnel
Displayed 5 (of 5) heads, 1 (of 1) midpoints, 2 (of 2) tails
Displayed 3 up, 2 down, 0 recovering, 0 recovered heads

```

The reason those two tunnels are down is the exact reason why affinity was removed from most of the topology; we cannot adjust the affinity of these tunnels, so there is no colorless path to reach the neighbors if affinity is set. The diagram shows this clearly as PCALC cannot find a valid path.

```

RP/0/0/CPU0:XRv12#show mpls traffic-eng tunnels 9992
Name: tunnel-te9992 Destination: 132.10.12.10 Ifhandle:0x1080 (auto-tunnel
mesh onehop)

```

```

  Signalled-Name: autom_XRv12_t9992_mg999

```

```

  Status:

```

```

    Admin:      up Oper: down Path: not valid Signalling: Down

```

```

    path option 10, type dynamic

```

```

    Last PCALC Error: Sat Oct 3 17:29:26 2015

```

```

      Info: No path to destination, 132.10.12.10 (affinity or reverselink or
exclude-link or hop-limit)

```

```

RP/0/0/CPU0:XRv12#show mpls traffic-eng tunnels 9993
Name: tunnel-te9993 Destination: 132.11.12.11 Ifhandle:0x1180 (auto-tunnel
mesh onehop)

```

```

  Signalled-Name: autom_XRv12_t9993_mg999

```

```

  Status:

```

```

    Admin:      up Oper: down Path: not valid Signalling: Down

```

```

    path option 10, type dynamic

```

```
Last PCALC Error: Sat Oct 3 17:29:26 2015
Info: No path to destination, 132.11.12.11 (affinity or hop-limit)
```

A quick look at the affinity information for the local node confirms this. The links to CSR6 and CSR9 have no link coloring and they are functioning properly. The links to CSR10 and XRv11 are colored and thus cannot be used by this feature.

```
RP/0/0/CPU0:XRv12#show mpls traffic-eng topology 12.12.12.12 affinity
IGP Id: 12.12.12.12, MPLS TE Id: 12.12.12.12 Router Node (OSPF 132 area 0)
Link[0]:      Intf Address: 132.6.12.12, Nbr Intf Address: 132.6.12.6
Attribute Flags: 0x0
Ext Admin Group:
Length: 256 bits
Value : 0x::
Attribute Names:
Link[1]:      Intf Address: 132.11.12.12, Nbr Intf Address: 132.11.12.11
Attribute Flags: 0xc
Ext Admin Group:
Length: 256 bits
Value : 0x::c
Attribute Names: BLUE(2) ORANGE(3)
Link[2]:      Intf Address: 132.10.12.12, Nbr Intf Address: 132.10.12.10
Attribute Flags: 0x9
Ext Admin Group:
Length: 256 bits
Value : 0x::9
Attribute Names: RED(0) ORANGE(3)
Link[3]:      Intf Address: 132.9.12.12, Nbr Intf Address: 132.9.12.9
Attribute Flags: 0x0
Ext Admin Group:
Length: 256 bits
Value : 0x::
Attribute Names:
```

Design tip: This technique of auto-backup NHOP-tunnels with auto-primary is best enabled everywhere. In doing this, we can automatically protect all links in the topology. This alleviates the need for individual TE tunnels to request FRR on a per-tunnel basis. Since every link is already protected, and every link has an FRR tunnel across it with autoroute, we can assume that any traffic crossing that link is protected as well. This is difficult to test since there is no PLR-originated PATHERR message back to the head-end (on a non-primary tunnel) to indicate local repair has happened. FRR happens by virtue of the links being protected; even normal LSPs can be protected if other TE tunnels are not used. This is the best way to protect mLDP LSM traffic which is detailed in another section. With autoroute enabled, we can simply enable this feature everywhere (along with tLDP) and have fully connectivity with FRR protection.

Last, we will examine automatic tunnel mesh-groups. This allows remote PEs to automatically build tunnels to one another based on an ACL that specifies all of the destinations. We can apply any

constraints we want, and in this case, the path must not be blue. It is somewhat manual when using an access-list in this way, but can greatly reduce configuration. This solution does not require any additional IGP extensions other than the TE ones already in use. We will build a mesh of TE tunnels between CSR1, CSR8, and XRv11. We will use autoroute destination so we can override any existing primary tunnels on CSR8 that are using autoroute announce. On XRv11, we use autoroute announce because we must, and also we have to specify the TE tunnel default source address in global configuration mode. Only numbered ACLs are supported which makes the configuration for difficult to interpret at a glance.

```
! CSR1 and CSR8
mpls traffic-eng auto-tunnel mesh
mpls traffic-eng auto-tunnel mesh tunnel-num min 5000 max 5099

interface Auto-Template500
 ip unnumbered Loopback0
 tunnel mode mpls traffic-eng
 tunnel destination access-list 5
 tunnel mpls traffic-eng autoroute destination
 tunnel mpls traffic-eng affinity 0x0 mask 0x4
 tunnel mpls traffic-eng path-option 10 dynamic

! CSR1 only
access-list 5 permit 8.8.8.8
access-list 5 permit 11.11.11.11

! CSR8 only
access-list 5 permit 1.1.1.1
access-list 5 permit 11.11.11.11

! XRv11
ipv4 unnumbered mpls traffic-eng Loopback0

ipv4 prefix-list PL_MESH_ENDPOINT
 10 permit 1.1.1.1/32
 20 permit 8.8.8.8/32

mpls traffic-eng
 auto-tunnel mesh
  group 500
   attribute-set ATT_NOT_BLUE
   destination-list PL_MESH_ENDPOINT
   timers removal unused 0
   tunnel-id min 5000 max 5099
 attribute-set auto-mesh ATT_NOT_BLUE
  logging events lsp-status state
  logging events lsp-status reroute
  autoroute announce
  affinity 0x0 mask 0x4
```

We quickly check all three routers to ensure the auto-mesh was built properly. Each router should have two tunnels going to the other 2 routers in the mesh.

```
CSR1#show mpls traffic-eng auto-tunnel mesh
```

```
Auto-Template500:
```

```
Using access-list 5 to clone the following tunnel interfaces:
```

Destination	Interface
-----	-----
8.8.8.8	Tunnel5000
11.11.11.11	Tunnel5001

```
Mesh tunnel interface numbers: min 5000 max 5099
```

```
CSR8#show mpls traffic-eng auto-tunnel mesh
```

```
Auto-Template500:
```

```
Using access-list 5 to clone the following tunnel interfaces:
```

Destination	Interface
-----	-----
1.1.1.1	Tunnel5000
11.11.11.11	Tunnel5001

```
Mesh tunnel interface numbers: min 5000 max 5099
```

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng auto-tunnel mesh
```

```
Auto-tunnel Mesh Global Configuration:
```

```
Unused removal timeout: 0s (Disabled)
```

```
Configured tunnel number range: 5000-5099
```

```
Auto-tunnel Mesh Groups Summary:
```

```
Mesh Groups count: 1
```

```
Mesh Groups Destinations count: 2
```

```
Mesh Groups Tunnels count:
```

```
2 created, 2 up, 0 down, 0 FRR enabled
```

```
Mesh Group: 500 (2 Destinations)
```

```
Status: Enabled
```

```
Attribute-set: ATT_NOT_BLUE
```

```
Destination-list: PL_MESH_ENDPOINT
```

```
Recreate timer: Not running
```

Destination	Tunnel ID	State	Unused timer
-----	-----	-----	-----
1.1.1.1	5002	up	Not running
8.8.8.8	5003	up	Not running

```
Displayed 2 tunnels, 2 up, 0 down, 0 FRR enabled
```

```
Auto-mesh Cumulative Counters:
```

```
Last cleared: MON DAY 1 20:57:16 2015 (1d18h ago)
```

	Total
Created:	4
Connected:	4
Removed (unused):	0
Removed (in use):	2
Range exceeded:	0

Next, we quickly check each PE's FIB to ensure the tunnels are being used for routing. This proves that autoroute can be used with the announce or destination options. Normal static or policy routing would not work since the tunnel IDs are dynamically allocated. Forwarding-adjacency also works, provided the tunnels are configured bidirectionally. On XE and XR, the FIBs never show the TE label, so do not be concerned when the labels are not revealed below.

```
CSR1#show ip cef 8.8.8.8
8.8.8.8/32
  attached to Tunnel5000
```

```
CSR1#show ip cef 11.11.11.11
11.11.11.11/32
  attached to Tunnel5001
```

```
CSR8#show ip cef 1.1.1.1
1.1.1.1/32
  attached to Tunnel5000
```

```
CSR8#show ip cef 11.11.11.11
11.11.11.11/32
  attached to Tunnel5001
```

```
RP/0/0/CPU0:XRv11#show cef ipv4 1.1.1.1
1.1.1.1/32, version 108, internal 0x1000001 0x0 (ptr 0xa13a7374) [1], 0x0
(0xa138c950), 0xa20 (0xa14aa190)
Prefix Len 32, traffic index 0, precedence n/a, priority 3
  via 1.1.1.1, tunnel-te5002, 3 dependencies, weight 0, class 0 [flags 0x0]
    path-idx 0 NHID 0x0 [0xa0e8f73c 0xa0e8f694]
    next hop 1.1.1.1
    local adjacency
      local label 91004      labels imposed {ImplNull}
```

```
RP/0/0/CPU0:XRv11#show cef ipv4 8.8.8.8
8.8.8.8/32, version 109, internal 0x1000001 0x0 (ptr 0xa13a75f4) [1], 0x0
(0xa138c974), 0xa20 (0xa14aa2d0)
Prefix Len 32, traffic index 0, precedence n/a, priority 3
  via 8.8.8.8, tunnel-te5003, 3 dependencies, weight 0, class 0 [flags 0x0]
    path-idx 0 NHID 0x0 [0xa0e8f544 0xa0e8f5ec]
    next hop 8.8.8.8
    local adjacency
      local label 91008      labels imposed {ImplNull}
```

Rather than test all six tunnels, we will test two to ensure the feature works. From CSR1 to XRv11, we know the tunnel ID is 5000. Checking the outgoing label, we find value 3010. Testing from inside the VPN, we can see that the first label in the stack, on the first hop, is 3010. This implies the auto-mesh LSP is being used for transport as desired.

```
CSR1#show mpls traffic-eng tunnels tunnel 5001 | include Label
InLabel : -
OutLabel : GigabitEthernet2.513, 3010
```

```
CSR7#traceroute 13.13.13.13 source 7.7.7.7
Type escape sequence to abort.
Tracing the route to 13.13.13.13
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.7.1 1 msec 1 msec 0 msec
 2 132.1.3.3 [MPLS: Labels 3010/91013 Exp 0] 31 msec 17 msec 17 msec
 3 132.3.10.10 [MPLS: Labels 10016/91013 Exp 0] 16 msec 19 msec 32 msec
 4 132.10.11.11 [MPLS: Label 91013 Exp 0] 32 msec 26 msec 32 msec
 5 10.11.13.13 25 msec * 14 msec
```

From XRv11 to CSR8, we perform a similar verification. Tunnel ID 5003 is used to forward traffic to CSR8, which uses label 10018. The first hop of the traceroute inside the VPN uses this value, which proves that the auto-mesh works on XRv also.

```
RP/0/0/CPU0:XRv11#show rsvp reservation session-type lsp-p2p destination
8.8.8.8 detail | include Label
Labels: Outgoing downstream: 10018.
```

```
RP/0/0/CPU0:XRv13#traceroute 14.14.14.14 source 13.13.13.13
Type escape sequence to abort.
Tracing the route to 14.14.14.14

 1 10.11.13.11 0 msec 0 msec 0 msec
 2 132.10.11.10 [MPLS: Labels 10018/8013 Exp 0] 89 msec 9 msec 9 msec
 3 132.6.10.6 [MPLS: Labels 6014/8013 Exp 0] 9 msec 9 msec 9 msec
 4 10.8.14.8 [MPLS: Labels 0/8013 Exp 0] 9 msec 9 msec 29 msec
 5 10.8.14.8 [MPLS: Labels 0/8013 Exp 0] 19 msec 9 msec 9 msec
 6 10.8.14.14 19 msec * 9 msec
```

OSPF also supports a fancier way to build auto-mesh tunnels. The XR documentation seems to suggest this is supported on XR, but it isn't. It only works on XE for OSPFv2; there is no support for IS-IS currently. We will use this to automatically build a mesh of tunnels to support the VPLS customers behind CSR1, CSR4, and CSR5. The absence of a static ACL means that all routers have identical configurations. We also use an explicit-path selecting any path that does not traverse CSR9. In general, only XE can include explicit-paths (only makes sense to use exclusion or loose hop inclusion); XR must use dynamic paths.


```

! CSR1, CSR4, and CSR5
mpls traffic-eng auto-tunnel mesh
mpls traffic-eng auto-tunnel mesh tunnel-num min 5000 max 5099

router ospf 132
  mpls traffic-eng mesh-group 501 Loopback0 area 0

ip explicit-path name EP_NO_R9 enable
  exclude-address 9.9.9.9

interface Auto-Template501
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination mesh-group 501
  tunnel mpls traffic-eng autoroute destination
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 explicit name EP_NO_R9

```

We can verify the tunnel IDs for each destination on each router for further analysis.

```

CSR1#show mpls traffic-eng auto-tunnel mesh | begin 501
Auto-Template501:
  Using mesh-group 501 to clone the following tunnel interfaces:

```

Destination	Interface
-----	-----
4.4.4.4	Tunnel5002
5.5.5.5	Tunnel5003

Mesh tunnel interface numbers: min 5000 max 5099

```

CSR4#show mpls traffic-eng auto-tunnel mesh | begin 501
Auto-Template501:
  Using mesh-group 501 to clone the following tunnel interfaces:

```

Destination	Interface
-----	-----
1.1.1.1	Tunnel5000
5.5.5.5	Tunnel5001

Mesh tunnel interface numbers: min 5000 max 5099

```

CSR5#show mpls traffic-eng auto-tunnel mesh | begin 501
Auto-Template501:
  Using mesh-group 501 to clone the following tunnel interfaces:

```

Destination	Interface

```

-----
1.1.1.1      Tunnel5000
4.4.4.4      Tunnel5001

```

Mesh tunnel interface numbers: min 5000 max 5099

Checking the tunnel briefs for tunnels in the "up" state on CSR4 (for brevity), we can see 2 auto-mesh tunnels in the summary. This represents the tunnels for which the local router is the head-end. Below, we see 4 tunnels total: CSR4 is the head for the first two as the destinations are CSR1 and CSR5. CSR4 is the tail end for the other 2 since the tunnel names contain the hostnames of the corresponding head ends.

```
CSR4#show mpls traffic-eng tunnels up brief
```

Signalling Summary:

```

LSP Tunnels Process:      running
Passive LSP Listener:     running
RSVP Process:             running
Forwarding:               enabled
auto-tunnel:

```

```

    backup Disabled (0 ), id-range:65436-65535
    onehop Disabled (0 ), id-range:65336-65435
    mesh Enabled (2 ), id-range:5000-5099

```

```

Periodic reoptimization:    every 3600 seconds, next in 394 seconds
Periodic FRR Promotion:     Not Running
Periodic auto-bw collection: every 300 seconds, next in 94 seconds

```

P2P TUNNELS/LSPs:

TUNNEL NAME	DESTINATION	UP IF	DOWN IF	STATE/PROT
CSR4_t5000	1.1.1.1	-	Gi2.514	up/up
CSR4_t5001	5.5.5.5	-	Gi2.534	up/up
CSR1_t5002	4.4.4.4	Gi2.514	-	up/up
CSR5_t5001	4.4.4.4	Gi2.534	-	up/up

Displayed 2 (of 2) heads, 0 (of 0) midpoints, 2 (of 2) tails

This information was dynamically exchanged using a new TLV to carry the mesh-group ID (MG-ID) inside of an opaque-area LSA (Type-10). The Link ID is always 4.0.0.0 and each MG-ID is 8 bytes long. The first 4 bytes represent the actual MG-ID, shown in hex (0x000001F5 = 501). The second 4 bytes represent the router's mesh-group IP address which serves as the tunnel destination for everyone else, also in hex (0x040404 = 4.4.4.4). Looking at CSR4's locally generated MG-ID LSA10, we can see this information carried in the TLV.

```
CSR4#show ip ospf 132 0 database opaque-area 4.0.0.0 adv-router 4.4.4.4
```

```
OSPF Router with ID (4.4.4.4) (Process ID 132)
```

Type-10 Opaque Link Area Link States (Area 0)

LS age: 430
Options: (No TOS-capability, DC)
LS Type: Opaque Area Link
Link State ID: 4.0.0.0
Opaque Type: 4
Opaque ID: 0
Advertising Router: 4.4.4.4
LS Seq Number: 80000001
Checksum: 0x4AC3
Length: 32

Capability Type: Mesh-group
Length: 8
Value:
0000 01F5 0404 0404

If we add another MG-ID to CSR4, perhaps to create an unrelated meshed of tunnels, this single LSA can be extended without having to create an LSA for each MG-ID. We can select a different "destination" interface as well, which doesn't make much sense in this case, but is possible. The original MG-ID still exists, with the addition of MG-ID 502 (0x000001F6) and the IP address from the specified link (0x84030404 = 132.3.4.4). I use yellow for MG-ID 501 and green for MG-ID 502. Each new MG-ID adds 8 bytes of length to the TLV.

```
! CSR4
router ospf 132
  mpls traffic-eng mesh-group 502 GigabitEthernet2.534 area 0

CSR4#show ip ospf 132 0 database opaque-area 4.0.0.0 adv-router 4.4.4.4
```

OSPF Router with ID (4.4.4.4) (Process ID 132)

Type-10 Opaque Link Area Link States (Area 0)

LS age: 34
Options: (No TOS-capability, DC)
LS Type: Opaque Area Link
Link State ID: 4.0.0.0
Opaque Type: 4
Opaque ID: 0
Advertising Router: 4.4.4.4
LS Seq Number: 80000003
Checksum: 0xB3C0
Length: 40

Capability Type: Mesh-group
Length: 16

Value:

0000 01F5 0404 0404 0000 01F6 8403 0404

The reason I suspect XR simply does not support this feature is because it cannot read this new TLV. I was not able to find a way to enable a feature to correct this, nor could I find a way to encode an MG-ID locally into the XR OSPF process. If XR cannot understand the MG-ID TLV, it is highly unlikely that the feature is supported.

```
RP/0/0/CPU0:XRv11#show ospf 132 0 database opaque-area 4.0.0.0 adv-router 4.4.4.4
```

```
OSPF Router with ID (11.11.11.11) (Process ID 132)
```

```
Type-10 Opaque Link Area Link States (Area 0)
```

```
LS age: 126
Options: (No TOS-capability, DC)
LS Type: Opaque Area Link
Link State ID: 4.0.0.0
Opaque Type: 4
Opaque ID: 0
Advertising Router: 4.4.4.4
LS Seq Number: 80000003
Checksum: 0xb3c0
Length: 40
```

Unknown TLV: Type: 3 Length: 16

A more graceful way to view this information is to look at the TE topology in brief form. Looking at CSR1, it can see CSR4's remote MG-IDs. It displays all 8 bytes of the TLV value, both the MG-ID and tunnel destination. CSR1 only cares about MG-ID 501 in this specific case, as MG-ID 502 was just a demonstration. The group/destination pairings are shown in different colors for clarity as they were above.

```
CSR1#show mpls traffic-eng topology 4.4.4.4 brief | include mg
Area mg-id's:
: mg-id 501 4.4.4.4 :: mg-id 502 132.3.4.4 :
```

We will verify that each of the PEs is routing correctly using the mesh tunnels before using MPLS OAM to verify the labels. The tunnel IDs being used for each remote destination appear correct.

```
CSR1#show ip cef 4.4.4.4
4.4.4.4/32
attached to Tunnel15002
```

```
CSR1#show ip cef 5.5.5.5
5.5.5.5/32
```

attached to Tunnel5003

```
CSR4#show ip cef 1.1.1.1
1.1.1.1/32
    attached to Tunnel5000
```

```
CSR4#show ip cef 5.5.5.5
5.5.5.5/32
    attached to Tunnel5001
```

```
CSR5#show ip cef 1.1.1.1
1.1.1.1/32
    attached to Tunnel5000
```

```
CSR5#show ip cef 4.4.4.4
4.4.4.4/32
    attached to Tunnel5001
```

We will test connectivity from CSR1 to CSR5. The outgoing label is 3022, which means CSR3 is the next-hop. MPLS OAM can verify that the tunnel is operation since the L2VPN cannot verify this.

```
CSR1#show mpls traffic-eng tunnels tunnel 5003 | include Label
InLabel   : -
OutLabel  : GigabitEthernet2.513, 3022
```

```
CSR1#traceroute mpls traffic-eng tunnel 5003
Tracing MPLS TE Label Switched Path on Tunnel5003, timeout is 2 seconds
[snip]
```

Type escape sequence to abort.

```
0 132.1.3.1 MRU 1500 [Labels: 3022 Exp: 0]
L 1 132.1.3.3 MRU 1500 [Labels: 2020 Exp: 0] 2 ms
L 2 132.2.3.2 MRU 1500 [Labels: implicit-null Exp: 0] 28 ms
! 3 132.2.5.5 23 ms
```

We can confirm this further in the control-plane using ordinary MPLS show commands. The label stack includes the RSVP-bound label 3022 since the next-hop to the targeted LDP endpoint is reachable via a TE tunnel. Using EPC, we can verify that the L2VPN traffic is using this TE LSP. We send traffic from XRv14 to XRv13, which uses CSR1 as the ingress L2VPN PE, and capture traffic heading to CSR3. Label 3022 (0xBCE) is indeed being used for transport, with label 5011 (0x1393) being used for the PW demultiplexer.

```
CSR1#show l2vpn atom vc destination 5.5.5.5 vcid 71314 detail | include
label_stack
    Output interface: Tu5003, imposed label stack {3022 5011}
```

```
RP/0/0/CPU0:XRv14#ping 10.0.0.13 source 10.0.0.14
```

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 10.0.0.13, timeout is 2 seconds:

!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 1/12/29 ms

CSR1#show monitor capture CAP buffer detail

```
11 144 1.035994 00:0C:29:FB:A3:39 -> 00:0C:29:D7:81:FE MPLS unicast
0000: 000C29D7 81FE000C 29FBA339 81000DB9 ..).....)..9....
0010: 884700BC E0FF0139 31FF0000 0000000C .G.....91.....
0020: 2955663D 000C29BE D8F90800 45000064 )Uf=..).....E..d
0030: 00000000 FF01A77E 0A00000E 0A00000D .....~.....
```

For additional testing, we will perform the same verification from CSR5 to CSR4. We already verified the tunnel ID was 5001 and the CEF was properly programmed. The outgoing label is 2018 which means CSR2 is the next-hop in the path. We can confirm this with MPLS traceroute.

CSR5#show ip rsvp reservation detail filter session-type 7 destination

4.4.4.4 | include Label

Label: 2018 (outgoing)

CSR5#traceroute mpls traffic-eng tunnel 5001

Tracing MPLS TE Label Switched Path on Tunnel5001, timeout is 2 seconds

[snip]

Type escape sequence to abort.

```
0 132.2.5.5 MRU 1500 [Labels: 2018 Exp: 0]
L 1 132.2.5.2 MRU 1500 [Labels: 3020 Exp: 0] 1 ms
L 2 132.2.3.3 MRU 1500 [Labels: implicit-null Exp: 0] 29 ms
! 3 132.3.4.4 21 ms
```

The L2VPN control-plane verification shows the proper label stack as well. When we send traffic from XRv3 to CSR7, we verify the proper label stack on the outgoing packets. Label 2018 (0x7E2) is the topmost, RSVP-allocated label and label 4010 (0xFAA) is the PW label.

CSR5#show mpls l2transport vc destination 4.4.4.4 vcid 71314 detail | include label_stack

Output interface: Tu5001, imposed label stack {2018 4010}

RP/0/0/CPU0:XRv13#ping 10.0.0.7 source 10.0.0.13

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 10.0.0.7, timeout is 2 seconds:

!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 1/10/29 ms

CSR5#show monitor capture CAP buffer detailed

```
5 144 1.061994 00:0C:29:5F:11:A1 -> 00:0C:29:5C:E1:E9 MPLS unicast
0000: 000C295C E1E9000C 295F11A1 81000DC5 ..)\....)_.....
0010: 8847007E 20FF00FA A1FF0000 0000000C .G.~ .....
```

```

0020:  29664C2C 000C2955 663D0800 45000064    ) fL, ...) Uf=..E..d
0030:  00000000 FF01A785 0A00000D 0A000007    .....

```

Additional Reading – Reference configurations "te-auto"

31.3 CBTS (IOS) and PBTS (XR)

This section is a continuation of the manual TE lab and reuses some of the same tunnels for testing (configurations are consolidated at the beginning of the section). Class and Policy Based Tunnel Selection (CBTS/PBTS) are mechanisms of mapping EXP values to different tunnels in XE and XR, respectively. The feature qualifies as a TE QoS mechanism but is not technically DiffServ-aware TE (DS-TE), which is different. The two are not mutually exclusive as we will see later. We will begin with CBTS. A master tunnel [ID 40] is created that bundles all of the member tunnels. The master determines the steering mechanism (autoroute, etc) but isn't a real TE tunnel. It's just a software interface so the routing process knows how to deliver traffic. The members have their own path-options along with mappings of EXP values. In this example, voice and video (EXP 5 and 4) are mapped to tunnel41 and follow the blue path, which we can pretend is low latency. Tunnel42 encompasses all other EXP values which is what the "default" keyword accomplishes. Notice that the member tunnels do not need a steering mechanism.

```

! CSR1
interface Tunnel40
  description CBTS MASTER
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng autoroute destination
  tunnel mpls traffic-eng exp-bundle master
  tunnel mpls traffic-eng exp-bundle member Tunnel41
  tunnel mpls traffic-eng exp-bundle member Tunnel42

interface Tunnel41
  description CBTS VOICE AND VIDEO
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng affinity 0x4 mask 0x4
  tunnel mpls traffic-eng path-option 10 dynamic
  tunnel mpls traffic-eng exp 4 5

interface Tunnel42
  description CBTS EVERYTHING ELSE
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 explicit name EP_1_4_3_10_11

```

```
tunnel mpls traffic-eng exp default
```

When the tunnels come up, we see only two exist, not three, since tunnel 40 is just a "traffic cop" to direct traffic into the proper member tunnels based on EXP.

```
CSR1#show ip rsvp reservation
```

To	From	Pro	DPort	Sport	Next Hop	I/F	Fi	Serv
BPS								
11.11.11.11	1.1.1.1	0	41	1	132.1.6.6	Gi2.516	SE	LOAD 0
11.11.11.11	1.1.1.1	0	42	1	132.1.4.4	Gi2.514	SE	LOAD 0

We can confirm this by checking all of the tunnel statuses. The master tunnel is up, but the output is limited to the member tunnels and their EXP mappings. We can see that the member tunnels look like regular TE tunnels with their EROs going in totally different directions, given independent tunnel constraints. Notice that label 4014 is used for the default traffic and label 6020 is used for voice/video.

```
CSR1#show mpls traffic-eng tunnels tunnel 40 | section Status
```

Status: Master

Admin: up Oper: up Signalling: N/A

Member Tunnels: Member Autoroute: Inactive

Tunnel41: Config Exp: 4 5

Tunnel42: Config Exp: default

```
CSR1#show mpls traffic-eng tunnels tunnel 41 | section Label|RSVP_Path
```

InLabel : -

OutLabel : GigabitEthernet2.516, 6020

RSVP Path Info:

My Address: 132.1.6.1

Explicit Route: 132.1.6.6 132.6.12.12 132.11.12.11 11.11.11.11

Record Route: NONE

Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits

```
CSR1#show mpls traffic-eng tunnels tunnel 42 | section Label|RSVP_Path
```

InLabel : -

OutLabel : GigabitEthernet2.514, 4014

RSVP Path Info:

My Address: 132.1.4.1

Explicit Route: 132.1.4.4 132.3.4.3 132.3.10.10 132.10.11.11
11.11.11.11

Record Route: NONE

Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits

We can also see the EXP mappings per destination. This shows the "configured" versus "actual", which is valuable to determine where traffic actually ends up. The D and NE flags are evaluated later.

```
CSR1#show mpls traffic-eng exp 11.11.11.11
```


Destination: 11.11.11.11

Master: Tunnel40	Status: up				
Members	Status	Conf	Exp	Actual	Exp
Tunnel41	up (Active)	4	5	4	5
Tunnel42	up (Active)	Default		0	1 2 3 6 7

(D) : Destination is different

(NE): Exp values not configured on tunnel

We quickly check the autoroute process. The routing is tied to the master tunnel40, not the members. The routing table also shows this as well. Traffic entering the master tunnel is further classified by EXP into the appropriate member tunnels based on the mappings shown above.

```
CSR1#show mpls traffic-eng autoroute 11.11.11.11
MPLS TE autorouting enabled
destination 11.11.11.11 has 1 tunnels
Tunnel40 (load balancing metric 0, nexthop 11.11.11.11)
(flags: Destination)
```

```
CSR1#show ip route 11.11.11.11
Routing entry for 11.11.11.11/32
Known via "static", distance 1, metric 0 (connected)
Routing Descriptor Blocks:
* directly connected, via Tunnel40
Route metric is 0, traffic share count is 1
```

A quick test from within the L3VPN shows that the CBTS configuration is working. Traffic at EXP 0 uses label 4014, which is for default traffic. Extended traceroute lets us specify DSCP 40 (CS5), which is voice traffic. This uses label 6020 which is correct.

```
CSR7#traceroute 13.13.13.13 source 7.7.7.7
Type escape sequence to abort.
Tracing the route to 13.13.13.13
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.7.1 1 msec 1 msec 0 msec
 2 132.1.4.4 [MPLS: Labels 4014/91012 Exp 0] 30 msec 25 msec 27 msec
 3 132.3.4.3 [MPLS: Labels 3027/91012 Exp 0] 22 msec 16 msec 17 msec
 4 132.3.10.10 [MPLS: Labels 10015/91012 Exp 0] 17 msec 19 msec 43 msec
 5 132.10.11.11 [MPLS: Label 91012 Exp 0] 29 msec 25 msec 17 msec
 6 10.11.13.13 25 msec * 14 msec
```

```
CSR7#traceroute
Protocol [ip]:
Target IP address: 13.13.13.13
Source address: 7.7.7.7
DSCP Value [0]: 40
[snip]
Type escape sequence to abort.
```

```

Tracing the route to 13.13.13.13
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.7.1 0 msec 0 msec 1 msec
 2 132.1.6.6 [MPLS: Labels 6020/91012 Exp 5] 46 msec 33 msec 34 msec
 3 132.6.12.12 [MPLS: Labels 92010/91012 Exp 5] 33 msec 33 msec 33 msec
 4 132.11.12.11 [MPLS: Label 91012 Exp 5] 33 msec 33 msec 33 msec
 5 10.11.13.13 24 msec * 21 msec

```

MPLS OAM can be used to verify this as well, provided we specify the master tunnel as the target. We can easily test different EXP values from within the SP network this way. This time, we will test video (EXP 4) and ensure it uses label 6020. EXP 3 still uses the default tunnel, which is correct. Notice that the OAM returning information shows EXP 0 for the packet that was in tunnel40; I assume this is a cosmetic behavior of the MPLS traceroute, since it categorized the traffic properly.

```

CSR1#traceroute mpls traffic-eng tunnel 40 exp 3
Tracing MPLS TE Label Switched Path on Tunnel40, timeout is 2 seconds
[snip]
Type escape sequence to abort.
 0 132.1.4.1 MRU 1500 [Labels: 4014 Exp: 0]
L 1 132.1.4.4 MRU 1500 [Labels: 3027 Exp: 0] 2 ms
L 2 132.3.4.3 MRU 1500 [Labels: 10015 Exp: 0] 27 ms
L 3 132.3.10.10 MRU 1500 [Labels: implicit-null Exp: 0] 20 ms
! 4 132.10.11.11 36 ms

```

```

CSR1#traceroute mpls traffic-eng tunnel 40 exp 4
Tracing MPLS TE Label Switched Path on Tunnel40, timeout is 2 seconds
[snip]
Type escape sequence to abort.
 0 132.1.6.1 MRU 1500 [Labels: 6020 Exp: 4]
L 1 132.1.6.6 MRU 1500 [Labels: 92010 Exp: 4] 2 ms
L 2 132.6.12.12 MRU 1500 [Labels: implicit-null Exp: 4] 43 ms
! 3 132.11.12.11 21 ms

```

These member tunnels can be protected by FRR, make bandwidth reservations, and do most other things ordinary TE tunnels can do. We will protect the voice/video tunnel with FRR (with bandwidth protection) and give it 2 Mbps of bandwidth. Even though there aren't any NHOP/NNHOP backup tunnels along the path, the features do work. We can check the RSVP RESV details to see the RRO with labels, along with the bandwidth reservation. Notice that the second tunnel42 does not have a bandwidth reservation nor an RRO, as expected.

```

! CSR
interface Tunnel41
 tunnel mpls traffic-eng bandwidth 2000
 tunnel mpls traffic-eng fast-reroute bw-protect

CSR1#show ip rsvp reservation detail filter session-type 7 destination
11.11.11.11

```

Reservation:

Tun Dest: 11.11.11.11 Tun ID: 41 Ext Tun ID: 1.1.1.1

Tun Sender: 1.1.1.1 LSP ID: 3

Next Hop: 132.1.6.6 on GigabitEthernet2.516

Label: 6007 (outgoing)

Reservation Style is Shared-Explicit, QoS Service is Controlled-Load

Resv ID handle: 3900041E.

Created: 17:47:00 UTC DAY MON 26 2015

Average Bitrate is 2M bits/sec, Maximum Burst is 1K bytes

Min Policed Unit: 0 bytes, Max Pkt Size: 1500 bytes

RRO:

6.6.6.6/32, Flags:0x20 (No Local Protection, Node-id)

Label subobject: Flags 0x1, C-Type 1, Label 6007

12.12.12.12/32, Flags:0x20 (No Local Protection, Node-id)

Label subobject: Flags 0x1, C-Type 1, Label 92010

132.6.12.12/32, Flags:0x0 (No Local Protection)

Label subobject: Flags 0x1, C-Type 1, Label 92010

11.11.11.11/32, Flags:0x20 (No Local Protection, Node-id)

Label subobject: Flags 0x1, C-Type 1, Label 3

132.11.12.11/32, Flags:0x0 (No Local Protection)

Label subobject: Flags 0x1, C-Type 1, Label 3

Status:

Policy: Accepted. Policy source(s): MPLS/TE

Reservation:

Tun Dest: 11.11.11.11 Tun ID: 42 Ext Tun ID: 1.1.1.1

Tun Sender: 1.1.1.1 LSP ID: 1

Next Hop: 132.1.4.4 on GigabitEthernet2.514

Label: 4014 (outgoing)

Reservation Style is Shared-Explicit, QoS Service is Controlled-Load

Resv ID handle: 0C000410.

Created: 17:25:32 UTC DAY MON 26 2015

Average Bitrate is 0 bits/sec, Maximum Burst is 1K bytes

Min Policed Unit: 0 bytes, Max Pkt Size: 1500 bytes

Status:

Policy: Accepted. Policy source(s): MPLS/TE

If we remove the "default" option from tunnel42, the tunnel is considered inactive. It is still a signaled and valid LSP from the perspective of RSVP, but cannot be used for CBTS. We can see that all 8 EXP values now map to the voice/video tunnel. Since there was nothing configured on tunnel42, the router makes no assumptions about what it should transport, and so uses it for nothing. Notice the NE flag on the tunnel42 entry below.

```
! CSR1
interface Tunnel42
 no tunnel mpls traffic-eng exp default

CSR1#show mpls traffic-eng exp 11.11.11.11
Destination: 11.11.11.11
```

Master: Tunnel40	Status: up		
Members	Status	Conf Exp	Actual Exp
Tunnel41	up (Active)	4 5	0 1 2 3 4 5 6 7
Tunnel42	up (Inactive)NE	Unassigned	

(D) : Destination is different

(NE) : Exp values not configured on tunnel

Unfortunately, OAM is not helpful in this situation, as it states there is no label. MPLS OAM actually appears to look at the configured EXP, not the actual EXP. Tracing from the VPN yields the proper results, which is a path via CSR6. Though undesirable from a design perspective, this is in accordance with the output above. The label changed to 6007 since enabling FRR caused the tunnel to be resigaled; label 6007 is shown above in the RSVP RESV RRO.

```
CSR1#traceroute mpls traffic-eng tunnel 40 exp 3
Tracing MPLS TE Label Switched Path on Tunnel40, timeout is 2 seconds
[snip]
```

Type escape sequence to abort.

```
0 0.0.0.0 MRU 0 [No Label]
```

Q 1 *

```
CSR7#traceroute 13.13.13.13 source 7.7.7.7
```

Type escape sequence to abort.

Tracing the route to 13.13.13.13

VRF info: (vrf in name/id, vrf out name/id)

```
1 10.1.7.1 1 msec 1 msec 0 msec
2 132.1.6.6 [MPLS: Labels 6007/91012 Exp 0] 50 msec 34 msec 33 msec
3 132.6.12.12 [MPLS: Labels 92010/91012 Exp 0] 33 msec 33 msec 41 msec
4 132.11.12.11 [MPLS: Label 91012 Exp 0] 33 msec 33 msec 33 msec
5 10.11.13.13 26 msec * 32 msec
```

Next, we will configure a few explicit values on tunnel42, but not all. The lower priority traffic is forced into tunnel42, and the system automatically placed all remaining EXP values into tunnel42 as well. When a "default" is not specified, the tunnel with the lowest explicit EXP is used for all remaining values. This configuration should give us the expected result, like the original setup, as the "actual EXP" is the same.

! CSR1

```
interface Tunnel42
```

```
tunnel mpls traffic-eng exp 0 1
```

```
CSR1#show mpls traffic-eng exp 11.11.11.11
```

Destination: 11.11.11.11

Master: Tunnel40	Status: up		
Members	Status	Conf Exp	Actual Exp
Tunnel41	up (Active)	4 5	4 5

```
Tunnel42          up    (Active)          0 1          0 1 2 3 6 7
```

To prove the OAM issue, we can easily verify the path is correct for EXP 1, but not EXP 2, since OAM is only looking at the configuration. This traffic is now properly placed in the non-voice/video tunnel.

```
CSR1#traceroute mpls traffic-eng tunnel 40 exp 1
Tracing MPLS TE Label Switched Path on Tunnel40, timeout is 2 seconds
[snip]
```

Type escape sequence to abort.

```
 0 132.1.4.1 MRU 1500 [Labels: 4014 Exp: 1]
L 1 132.1.4.4 MRU 1500 [Labels: 3027 Exp: 1] 2 ms
L 2 132.3.4.3 MRU 1500 [Labels: 10015 Exp: 1] 33 ms
L 3 132.3.10.10 MRU 1500 [Labels: implicit-null Exp: 1] 27 ms
! 4 132.10.11.11 38 ms
```

```
CSR1#traceroute mpls traffic-eng tunnel 40 exp 2
Tracing MPLS TE Label Switched Path on Tunnel40, timeout is 2 seconds
[snip]
```

Type escape sequence to abort.

```
 0 0.0.0.0 MRU 0 [No Label]
```

```
Q 1 *
```

We can add new tunnels to the bundle at any time. We will use tunnel43 for network control traffic. We can have up to 8 tunnels, one for each EXP.

```
! CSR1
interface Tunnel43
  description CBTS NET CONTROL
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 explicit name EP_NO_R9
  tunnel mpls traffic-eng exp 6 7

interface Tunnel40
  tunnel mpls traffic-eng exp-bundle member Tunnel43
```

There still isn't a default tunnel, so the remaining values are mapped to the tunnel with the lowest explicit EXP. This is still the correct operation.

```
CSR1#show mpls traffic-eng exp 11.11.11.11
Destination: 11.11.11.11
```

```
Master: Tunnel40          Status: up
```

Members	Status	Conf Exp	Actual Exp
Tunnel41	up (Active)	4 5	4 5
Tunnel42	up (Active)	0 1	0 1 2 3
Tunnel43	up (Active)	6 7	6 7

The combinations of tunnel failures, EXP mappings, and routing options are vast and this document does not cover them all. We will evaluate a few, though. If we shut down one tunnel causing the member LSP to fail, the values are moved onto the other tunnels. Shutting down tunnel41 moves EXP4 and EXP5 to tunnel42, which has the lowest configured EXP value.

```
CSR1#show mpls traffic-eng exp 11.11.11.11
Destination: 11.11.11.11
```

Master: Tunnel40		Status: up	
Members	Status	Conf Exp	Actual Exp
Tunnel41	down	4 5	
Tunnel42	up (Active)	0 1	0 1 2 3 4 5
Tunnel43	up (Active)	6 7	6 7

If we shut down tunnel43, the traffic is moved into tunnel41. I assume this is because EXP4 and EXP5 are the "next lowest" values. It would not be appropriate to "promote" lower values like 4 and 5 into a tunnel designed for 6 and 7. However, "demoting" higher values to the next-best tunnel appears to be functional.

```
CSR1#show mpls traffic-eng exp 11.11.11.11
Destination: 11.11.11.11
```

Master: Tunnel40		Status: up	
Members	Status	Conf Exp	Actual Exp
Tunnel41	up (Active)	4 5	4 5 6 7
Tunnel42	up (Active)	0 1	0 1 2 3
Tunnel43	down	6 7	

If we shut down tunnel42, the traffic is moved into tunnel41. This is the "next lowest" EXP and is "more" acceptable than moving it into the tunnel carrying higher EXPs.

```
CSR1#show mpls traffic-eng exp 11.11.11.11
Destination: 11.11.11.11
```

Master: Tunnel40		Status: up	
Members	Status	Conf Exp	Actual Exp
Tunnel41	up (Active)	4 5	0 1 2 3 4 5
Tunnel42	down	0 1	
Tunnel43	up (Active)	6 7	6 7

An error condition exists when a member tunnel has an incorrect tunnel destination, as shown below. Because the tunnel is still a member of a bundle with the correct destination, the tunnel is considered inactive. It is still a valid LSP and RSVP will happily signal the path to the new (incorrect) destination. Traffic will not flow correctly, and the EXP values mapped to that tunnel are moved to a valid bundle member.

```
interface Tunnel43
  description CBTS NET CONTROL
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 12.12.12.12
```

```
CSR1#show ip rsvp reservation
```

To	From	Pro	DPort	Sport	Next Hop	I/F	Fi	Serv
BPS								
11.11.11.11	1.1.1.1	0	41	9	132.1.6.6	Gi2.516	SE	LOAD 2M
11.11.11.11	1.1.1.1	0	42	3	132.1.4.4	Gi2.514	SE	LOAD 0
12.12.12.12	1.1.1.1	0	43	4	132.1.6.6	Gi2.516	SE	LOAD 0

While making the change, we can enable debugging to see CBTS remove the tunnel from the bundle. The show command also shows the D flag since the member has a different destination than the master. EXP 6 and 7 are moved to tunnel41 as the next-best valid option.

```
CSR1#show mpls traffic-eng exp 11.11.11.11
Destination: 11.11.11.11
```

Master: Tunnel40		Status: up						
Members	Status	Conf	Exp	Actual	Exp			
Tunnel41	up (Active)	4	5	4	5	6	7	
Tunnel42	up (Active)	0	1	0	1	2	3	
Tunnel43	up (Inactive)D	6	7					

```
(D) : Destination is different
(NE): Exp values not configured on tunnel
```

```
CSR1#debug mpls traffic-eng exp
TE-HE-BNDL: Member destination changed.. remove from bucket
TE-HE-BNDL: Member tunnel property change. walk update adj bundle
TE-HE-BNDL: Member dest different from bundle dest
```

Like any TE tunnel, routing still determines how traffic is steered into the tunnel head. If change the master to use autoroute announce, then create a non-CBTS tunnel with autoroute announce with a better metric, none of the traffic will enter the CBTS tunnel at all.

```
! CSR1
interface Tunnel40
```

```

tunnel mpls traffic-eng autoroute announce
no tunnel mpls traffic-eng autoroute destination

interface Tunnel44
description CBTS COMPETITOR
ip unnumbered Loopback0
tunnel mode mpls traffic-eng
tunnel destination 11.11.11.11
tunnel mpls traffic-eng autoroute announce
tunnel mpls traffic-eng autoroute metric relative -1
tunnel mpls traffic-eng affinity 0x2 mask 0x2
tunnel mpls traffic-eng path-option 10 dynamic

```

We can check the autoroute cache and RIB to verify that only the new tunnel44 is used.

```

CSR1#show mpls traffic-eng autoroute 11.11.11.11
MPLS TE autorouting enabled
  destination 0000.0000.0011.00, area isis level-2, has 2 tunnels
    Tunnel40      (load balancing metric 0, nexthop 11.11.11.11)
                  (flags: Announce)
    Tunnel44      (load balancing metric 0, nexthop 11.11.11.11, relative
metric -1)
                  (flags: Announce)

CSR1#show ip route 11.11.11.11
Routing entry for 11.11.11.11/32
  Known via "isis", distance 115, metric 29, type level-2
  Redistributing via isis 132
  Last update from 11.11.11.11 on Tunnel44, 00:01:47 ago
  Routing Descriptor Blocks:
  * 11.11.11.11, from 11.11.11.11, 00:01:47 ago, via Tunnel44
    Route metric is 29, traffic share count is 1

```

The label for this new tunnel is 3018, which is not CBTS related at all. When we traceroute from inside the VPN, all traffic goes into this tunnel, and CBTS is totally bypassed. We will shut down tunnel44 before continuing so CBTS is active again. The point is that CBTS master tunnel is not magic and competes directly with any other TE tunnels based on the routing convergence.

```

CSR1#show mpls traffic-eng tunnels tunnel 44 | include Label
InLabel   : -
OutLabel  : GigabitEthernet2.513, 3018

CSR7#traceroute 13.13.13.13 source 7.7.7.7
Type escape sequence to abort.
Tracing the route to 13.13.13.13
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.7.1 1 msec 0 msec 1 msec
 2 132.1.3.3 [MPLS: Labels 3018/91012 Exp 0] 26 msec 16 msec 16 msec

```



```

3 132.2.3.2 [MPLS: Labels 2023/91012 Exp 0] 17 msec 37 msec 38 msec
4 132.2.10.10 [MPLS: Labels 10016/91012 Exp 0] 39 msec 38 msec 38 msec
5 132.10.11.11 [MPLS: Label 91012 Exp 0] 35 msec 25 msec 25 msec
6 10.11.13.13 34 msec * 22 msec

```

```

CSR7#traceroute
Protocol [ip]:
Target IP address: 13.13.13.13
Source address: 7.7.7.7
DSCP Value [0]: 48
[snip]
Type escape sequence to abort.
Tracing the route to 13.13.13.13
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.7.1 1 msec 0 msec 1 msec
 2 132.1.3.3 [MPLS: Labels 3018/91012 Exp 6] 53 msec 25 msec 17 msec
 3 132.2.3.2 [MPLS: Labels 2023/91012 Exp 6] 17 msec 16 msec 25 msec
 4 132.2.10.10 [MPLS: Labels 10016/91012 Exp 6] 38 msec 43 msec 35 msec
 5 132.10.11.11 [MPLS: Label 91012 Exp 6] 25 msec 33 msec 19 msec
 6 10.11.13.13 40 msec * 40 msec

```

PBTS is similar to CBTS except is supported on XR. CBTS is not supported on XR and PBTS is not supported on XE, so the features are mutually exclusive per platform. PBTS is simpler to configure and understand since there is no need for a tunnel bundle. The first set of tunnels [ID 50 - 52] capture the Network Control, Voice/Video, and everything else. If no default tunnel is available, the tunnel with the lowest EXP value carries the remaining traffic. Each tunnel is totally independent in terms of constraints (affinity, bandwidth, path-selection, etc). You cannot explicitly map EXP 0.

```

! Xrv11
interface tunnel-te50
  description PBTS NET CONTROL
  ipv4 unnumbered Loopback0
  logging events all
  autoroute announce
  destination 1.1.1.1
  policy-class 6 7
  affinity 0x8 mask 0x8
  path-option 10 dynamic

interface tunnel-te51
  description PBTS VOICE AND VIDEO
  ipv4 unnumbered Loopback0
  logging events all
  signalled-bandwidth 5000
  autoroute announce
  destination 1.1.1.1
  policy-class 4 5
  affinity ignore

```

```

path-option 10 explicit name PATH_11_5_2_3_4_1

interface tunnel-te52
description PBTS DEFAULT CLASS
ipv4 unnumbered Loopback0
logging events all
autoroute announce
destination 1.1.1.1
policy-class default
affinity 0x4 mask 0x4
path-option 10 dynamic

```

We will quickly check to see that the tunnels came up successfully. We also validate the PBTS mappings per tunnel.

```

RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels destination 1.1.1.1 up |
utility egrep '^Name|Admin|Policy'
Name: tunnel-te50 Destination: 1.1.1.1 Ifhandle:0x1080
Admin: up Oper: up Path: valid Signalling: connected
AutoRoute: enabled LockDown: disabled Policy class: 6 7
Name: tunnel-te51 Destination: 1.1.1.1 Ifhandle:0x1180
Admin: up Oper: up Path: valid Signalling: connected
AutoRoute: enabled LockDown: disabled Policy class: 4 5
Name: tunnel-te52 Destination: 1.1.1.1 Ifhandle:0x1280
Admin: up Oper: up Path: valid Signalling: connected
AutoRoute: enabled LockDown: disabled Policy class: default

```

We also check the labels bound to each tunnel for verification purposes. We also verify that autoroute is working correctly, showing all three tunnels.

```

RP/0/0/CPU0:XRv11#show rsvp reservation session-type lsp-p2p destination
1.1.1.1 detail | utility egrep 'Label|TunID'
RESV: IPv4-LSP Session addr: 1.1.1.1. TunID: 50. LSPIId: 2.
Labels: Outgoing downstream: 92010.
RESV: IPv4-LSP Session addr: 1.1.1.1. TunID: 51. LSPIId: 3.
Labels: Outgoing downstream: 5012.
RESV: IPv4-LSP Session addr: 1.1.1.1. TunID: 52. LSPIId: 2.
Labels: Outgoing downstream: 5002.

RP/0/0/CPU0:XRv11#show mpls traffic-eng autoroute
Destination 1.1.1.1 has 3 tunnels in IS-IS 132 level 2
tunnel-te50 (traffic share 0, nexthop 1.1.1.1, metric 0)
(IS-IS 132 level-2, IPV4 Unicast)
Signalled-Name: XRv11_t50
tunnel-te51 (traffic share 0, nexthop 1.1.1.1, metric 0)
(IS-IS 132 level-2, IPV4 Unicast)
Signalled-Name: XRv11_t51
tunnel-te52 (traffic share 0, nexthop 1.1.1.1, metric 0)

```

```
(IS-IS 132 level-2, IPV4 Unicast)
Signalled-Name: XRv11_t52
```

Since the auto-route metrics and traffic-shares are set to 0, we can verify the actual class-based forwarding by checking the FIB. All three entries are shown in the FIB and the classes correspond with each tunnel as specified by PBTS.

```
RP/0/0/CPU0:XRv11#show cef ipv4 1.1.1.1/32
1.1.1.1/32, version 186, internal 0x1000001 0x0 (ptr 0xa14176f4) [1], 0x0
(0xa13fcb00), 0xa20 (0xa191e050)
Prefix Len 32, traffic index 0, precedence n/a, priority 3
  via 1.1.1.1, tunnel-te50, 3 dependencies, weight 0, class 6 7 [flags 0x0]
    path-idx 0 NHID 0x0 [0xa0e977e4 0xa0e9788c]
    next hop 1.1.1.1
    local adjacency
      local label 91006      labels imposed {ImplNull}
  via 1.1.1.1, tunnel-te51, 3 dependencies, weight 0, class 4 5 [flags 0x0]
    path-idx 1 NHID 0x0 [0xa0e97544 0xa0e975ec]
    next hop 1.1.1.1
    local adjacency
      local label 91006      labels imposed {ImplNull}
  via 1.1.1.1, tunnel-te52, 3 dependencies, weight 0, class default [flags
0x0]
    path-idx 2 NHID 0x0 [0xa0e97694 0xa0e9773c]
    next hop 1.1.1.1
    local adjacency
      local label 91006      labels imposed {ImplNull}
```

The FIB details specifically detail PBTS parameters also. The distribution won't be 1:3 as it is class based, but the FIB shows all three tunnels, their paths, and their position in the hash list.

```
RP/0/0/CPU0:XRv11#show cef ipv4 1.1.1.1/32 detail | begin Weight
Weight distribution:
slot 0, weight 0, normalized_weight 1, class default
slot 1, weight 0, normalized_weight 1, class 4 5
slot 2, weight 0, normalized_weight 1, class 6 7

PBTS class information:
class 4:  1 paths, offset 1
class 5:  1 paths, offset 1
class 6:  1 paths, offset 2
class 7:  1 paths, offset 2
class default: 1 paths, offset 0

Load distribution: 0 1 2 (refcount 3)

Hash  OK  Interface                Address
0     Y   tunnel-te52                point2point
```

1	Y	tunnel-te51	point2point
2	Y	tunnel-te50	point2point

Because XR does not allow us to specify TOS values in traceroute, and XRv does not support MQC-based QoS at all, it is hard to verify operation without changing the topology to use XE and a CE. As a temporary solution, and because no tunnels currently traverse the link between XRv11 and CSR10, we will disable LDP on that link on CSR10, as well as configure a static route to direct traffic to 1.1.1.1/32 towards XRv11. At the end of the PBTS section, these changes are removed.

```
! CSR10
interface GigabitEthernet2.501
  no mpls ldp igp autoconfig

ip route 1.1.1.1 255.255.255.255 132.10.11.11
```

XRv11 will perform a lookup in its FIB since the traffic is IP, not MPLS. It will then place the traffic into the proper TE tunnel based on the first 3 bits of the TOS byte. Below, we test DSCP CS6, CS5, and CS2. Each of them end up in the proper tunnels as we can confirm by checking the first label along the TE-LSPs.

```
CSR10#traceroute
Protocol [ip]:
Target IP address: 1.1.1.1
Source address: 10.10.10.10
DSCP Value [0]: 48
[snip]
Tracing the route to 1.1.1.1
VRF info: (vrf in name/id, vrf out name/id)
 1 132.10.11.11 1 msec 0 msec 0 msec
 2 132.11.12.12 [MPLS: Label 92010 Exp 6] 19 msec 17 msec 17 msec
 3 132.6.12.6 [MPLS: Label 6020 Exp 6] 16 msec 43 msec 36 msec
 4 132.6.9.9 [MPLS: Label 9009 Exp 6] 35 msec 48 msec 30 msec
 5 132.4.9.4 [MPLS: Label 4013 Exp 6] 15 msec 16 msec 66 msec
```

```
CSR10#traceroute
Protocol [ip]:
Target IP address: 1.1.1.1
Source address: 10.10.10.10
DSCP Value [0]: 40
[snip]
Tracing the route to 1.1.1.1
VRF info: (vrf in name/id, vrf out name/id)
 1 132.10.11.11 1 msec 0 msec 1 msec
 2 132.5.11.5 [MPLS: Label 5012 Exp 5] 12 msec 8 msec 8 msec
 3 132.2.5.2 [MPLS: Label 2017 Exp 5] 13 msec 27 msec 27 msec
 4 132.2.3.3 [MPLS: Label 3015 Exp 5] 25 msec 15 msec 16 msec
 5 132.3.4.4 [MPLS: Label 4002 Exp 5] 15 msec 6 msec 9 msec
```

```
6 132.1.4.1 8 msec * 9 msec
```

```
CSR10#traceroute
Protocol [ip]:
Target IP address: 1.1.1.1
Source address: 10.10.10.10
DSCP Value [0]: 16
[snip]
Tracing the route to 1.1.1.1
VRF info: (vrf in name/id, vrf out name/id)
 1 132.10.11.11 1 msec 1 msec 1 msec
 2 132.5.11.5 [MPLS: Label 5002 Exp 2] 9 msec 8 msec 21 msec
 3 132.5.10.10 [MPLS: Label 10018 Exp 2] 25 msec 32 msec 30 msec
 4 132.6.10.6 [MPLS: Label 6008 Exp 2] 29 msec 20 msec 16 msec
 5 132.1.6.1 13 msec * 25 msec
```

Next, we will shut down the default tunnel52 and verify that the traffic traverses tunnel51. This is the "next lowest" traffic class, so all lower priority traffic uses that in the absence of a default tunnel. Notice that the CEF details make no mention of this behavior, so the user must know that EXP values 0-3 are mapped to tunnel51 in this case using label 5012.

```
RP/0/0/CPU0:XRv11#show cef ipv4 1.1.1.1/32 detail | begin Weight
Weight distribution:
slot 0, weight 0, normalized_weight 1, class 4 5
slot 1, weight 0, normalized_weight 1, class 6 7

PBTS class information:
class 4: 1 paths, offset 0
class 5: 1 paths, offset 0
class 6: 1 paths, offset 1
class 7: 1 paths, offset 1

Load distribution: 0 1 (refcount 3)

Hash OK Interface Address
0 Y tunnel-te51 point2point
1 Y tunnel-te50 point2point
```

```
CSR10#traceroute
Protocol [ip]:
Target IP address: 1.1.1.1
Source address: 10.10.10.10
DSCP Value [0]: 8
[snip]
Tracing the route to 1.1.1.1
VRF info: (vrf in name/id, vrf out name/id)
 1 132.10.11.11 1 msec 1 msec 1 msec
 2 132.5.11.5 [MPLS: Label 5012 Exp 1] 21 msec 8 msec 8 msec
```

```

3 132.2.5.2 [MPLS: Label 2017 Exp 1] 10 msec 15 msec 15 msec
4 132.2.3.3 [MPLS: Label 3015 Exp 1] 15 msec 15 msec 16 msec
5 132.3.4.4 [MPLS: Label 4002 Exp 1] 15 msec 8 msec 8 msec
6 132.1.4.1 8 msec * 12 msec

```

We bring up tunnel52 again and note the new label is 5013. We shut down tunnel50, the "best" tunnel. Traffic is moved into the default tunnel52 as expected, per the FIB. We confirm this by using traceroute to see label 5013 at the front of the LSP. Tunnel51 is still servicing classes 4 and 5, which we confirm with DSCP CS4.

```

RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 52 detail | include Label
    Outgoing Interface: GigabitEthernet0/0/0/0.551, Outgoing Label: 5013

```

```

RP/0/0/CPU0:XRv11#show cef ipv4 1.1.1.1/32 | include class
    via 1.1.1.1, tunnel-te51, 3 dependencies, weight 0, class 4 5 [flags 0x0]
    via 1.1.1.1, tunnel-te52, 3 dependencies, weight 0, class default [flags
0x0]

```

```

CSR10#traceroute
Protocol [ip]:
Target IP address: 1.1.1.1
Source address: 10.10.10.10
DSCP Value [0]: 56
[snip]
Tracing the route to 1.1.1.1
VRF info: (vrf in name/id, vrf out name/id)
 1 132.10.11.11 1 msec 1 msec 1 msec
 2 132.5.11.5 [MPLS: Label 5013 Exp 7] 20 msec 8 msec 8 msec
 3 132.5.10.10 [MPLS: Label 10012 Exp 7] 26 msec 32 msec 25 msec
 4 132.6.10.6 [MPLS: Label 6011 Exp 7] 29 msec 38 msec 16 msec
 5 132.1.6.1 17 msec * 7 msec

```

```

CSR10#traceroute
Protocol [ip]:
Target IP address: 1.1.1.1
Source address: 10.10.10.10
DSCP Value [0]: 32
[snip]
Tracing the route to 1.1.1.1
VRF info: (vrf in name/id, vrf out name/id)
 1 132.10.11.11 1 msec 1 msec 1 msec
 2 132.5.11.5 [MPLS: Label 5012 Exp 4] 21 msec 9 msec 9 msec
 3 132.2.5.2 [MPLS: Label 2017 Exp 4] 15 msec 15 msec 15 msec
 4 132.2.3.3 [MPLS: Label 3015 Exp 4] 15 msec 16 msec 15 msec
 5 132.3.4.4 [MPLS: Label 4002 Exp 4] 15 msec 16 msec 9 msec
 6 132.1.4.1 9 msec * 20 msec

```

Forwarding-adjacency is also supported for PBTS. So long as there is a single tunnel from the tail to head (CSR1 to XRv11) that is also configured for the feature, all of the PBTS tunnels can rely on it. We can quickly reconfigure XRv11's PBTS tunnels, as well as bring up tunnel24 on CSR1 (not shown), which is enabled for forwarding-adjacency.

```
! XRv11
interface tunnel-te50
  no autoroute announce
  forwarding-adjacency
  holdtime 10000

interface tunnel-te51
  no autoroute announce
  forwarding-adjacency
  holdtime 10000

interface tunnel-te52
  no autoroute announce
  forwarding-adjacency
  holdtime 10000

router isis 132
  interface tunnel-te50
    address-family ipv4 unicast
    metric 25

  interface tunnel-te51
    address-family ipv4 unicast
    metric 25

  interface tunnel-te52
    address-family ipv4 unicast
    metric 25
```

The IS-IS topology shows three separate links to CSR1 from XRv11. The FIB has the same look as before since all of the paths appear "equal" until we start sending traffic.

```
RP/0/0/CPU0:XRv11#show isis topology systemid 0000.0000.0001
IS-IS 132 paths to IPv4 Unicast (Level-2) routers
```

System Id	Metric	Next-Hop	Interface	SNPA
CSR1	25	CSR1	tt52	*PtoP*
CSR1	25	CSR1	tt51	*PtoP*
CSR1	25	CSR1	tt50	*PtoP*

```
RP/0/0/CPU0:XRv11#show cef ipv4 1.1.1.1/32 | include class
  via 1.1.1.1, tunnel-te50, 9 dependencies, weight 0, class 6 7 [flags 0x0]
  via 1.1.1.1, tunnel-te51, 9 dependencies, weight 0, class 4 5 [flags 0x0]
```

via 1.1.1.1, tunnel-te52, 9 dependencies, weight 0, class default [flags 0x0]

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng forwarding-adjacency 1.1.1.1
destination 1.1.1.1 has 4 tunnels
    tunnel-te24      (traffic share 0, next-hop 1.1.1.1)
                     (Adjacency Announced: no, holdtime 10000)
    tunnel-te50      (traffic share 0, next-hop 1.1.1.1)
                     (Adjacency Announced: yes, holdtime 10000)
                     (IS-IS 132 level-2, IPV4 Unicast)
    tunnel-te51      (traffic share 0, next-hop 1.1.1.1)
                     (Adjacency Announced: yes, holdtime 10000)
                     (IS-IS 132 level-2, IPV4 Unicast)
    tunnel-te52      (traffic share 0, next-hop 1.1.1.1)
                     (Adjacency Announced: yes, holdtime 10000)
                     (IS-IS 132 level-2, IPV4 Unicast)
```

For confirmation, we verify the labels on the tunnels. Then, we perform three traceroutes on CSR10 to verify PBTS works with forwarding-adjacency in a quasi-P2MP environment. with CBTS, this behavior is different since there is a tunnel-bundle and only one link.

```
RP/0/0/CPU0:XRv11#show rsvp reservation session-type lsp-p2p destination
1.1.1.1 detail | utility egrep 'Label|TunID'
RESV: IPv4-LSP Session addr: 1.1.1.1. TunID: 50. LSPIId: 4.
    Labels: Outgoing downstream: 92010.
RESV: IPv4-LSP Session addr: 1.1.1.1. TunID: 51. LSPIId: 4.
    Labels: Outgoing downstream: 5002.
RESV: IPv4-LSP Session addr: 1.1.1.1. TunID: 52. LSPIId: 4.
    Labels: Outgoing downstream: 5015.
```

```
CSR10#traceroute ip
Target IP address: 1.1.1.1
Source address: 10.10.10.10
DSCP Value [0]: 48
[snip]
Tracing the route to 1.1.1.1
VRF info: (vrf in name/id, vrf out name/id)
 1 132.10.11.11 1 msec 1 msec 0 msec
 2 132.11.12.12 [MPLS: Label 92010 Exp 6] 28 msec 16 msec 25 msec
 3 132.6.12.6 [MPLS: Label 6004 Exp 6] 34 msec 25 msec 18 msec
 4 132.6.9.9 [MPLS: Label 9000 Exp 6] 42 msec 30 msec 42 msec
 5 132.4.9.4 [MPLS: Label 4013 Exp 6] 25 msec 34 msec 16 msec
 6 132.1.4.1 25 msec * 56 msec
```

```
CSR10#traceroute ip
Target IP address: 1.1.1.1
Source address: 10.10.10.10
DSCP Value [0]: 40
```



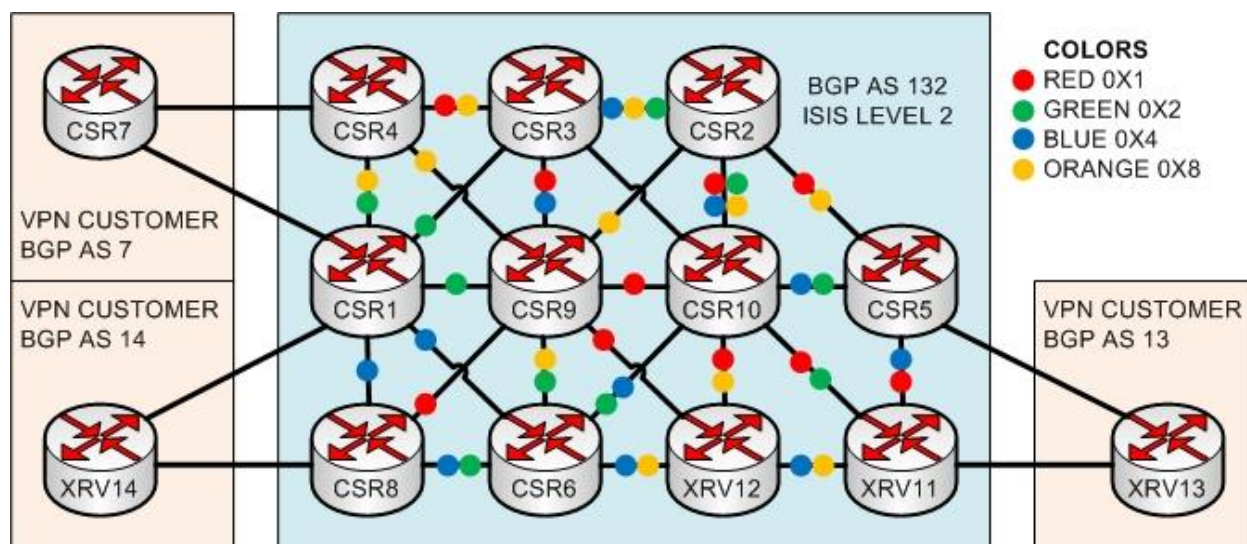
```
[snip]
Tracing the route to 1.1.1.1
VRF info: (vrf in name/id, vrf out name/id)
 1 132.10.11.11 1 msec 1 msec 1 msec
 2 132.5.11.5 [MPLS: Label 5002 Exp 5] 18 msec 8 msec 8 msec
 3 132.2.5.2 [MPLS: Label 2019 Exp 5] 9 msec 15 msec 26 msec
 4 132.2.3.3 [MPLS: Label 3017 Exp 5] 21 msec 15 msec 15 msec
 5 132.3.4.4 [MPLS: Label 4015 Exp 5] 15 msec 17 msec 11 msec
 6 132.1.4.1 5 msec * 7 msec
```

```
CSR10#traceroute ip
Target IP address: 1.1.1.1
Source address: 10.10.10.10
DSCP Value [0]: 24
```

```
[snip]
Tracing the route to 1.1.1.1
VRF info: (vrf in name/id, vrf out name/id)
 1 132.10.11.11 1 msec 1 msec 1 msec
 2 132.5.11.5 [MPLS: Label 5015 Exp 3] 5 msec 10 msec 30 msec
 3 132.5.10.10 [MPLS: Label 10015 Exp 3] 27 msec 25 msec 25 msec
 4 132.6.10.6 [MPLS: Label 6005 Exp 3] 25 msec 20 msec 14 msec
 5 132.1.6.1 20 msec * 16 msec
```

31.4 DiffServ-aware Traffic Engineering (DS-TE)

DiffServ-aware TE grants finer granularity to bandwidth reservations. Multiple bandwidth pools can exist with different amounts of bandwidth to service different traffic types. The pre-standard Cisco version accounted for two pools: global and sub. The global pool is what we have already analyzed, which represents the more general pool of bandwidth. If the bandwidth reservation is unspecified, the global pool is assumed. The sub-pool is typically used for higher priority flows, such as voice, and is accounted for separately from the global pool. The IETF later extended DS-TE to encompass up to 8 different class-types (CT), each of which can have flows within the existing 8 priorities, making 64 possible combinations of TE classes; currently, only 2 of the CTs are used (seen later). As a general comment, routers migrating from pre-standard to standard (IETF) modes can be configured to use both. In this model, the local router floods/advertises pre-standard information, but is capable of processing both pre-standard and IETF information. The network diagram is identical to the TE-manual diagram (affinity on all links), but is shown again for clarity.



31.4.1 Pre-standard Model

The pre-standard version is very straightforward to configure; Cisco only changed two commands to account for DS-TE originally. The "ip rsvp bandwidth" command at the link-level includes a "sub-pool" option to specify available sub-pool bandwidth. To use this sub-pool bandwidth, TE tunnels can specify "sub-pool" as an option within the "tunnel mpls traffic-eng bandwidth" command. The feature does not have to be explicitly enabled on core LSRs and all of the show commands are the same. The pre-standard version behaves like the IETF Russian Dolls Model (RDM) except with only two bandwidth constraints (BC). Because RSVP has not been extended to support the pre-standard version, global pool bandwidth is signaled as "controlled-load" service, whereas sub pool bandwidth is signaled as "guaranteed" service. Those terms are discussed later, but the point is that "pre-standard" is functionally similar to IETF RDM minus the RSVP extensions. We will enable an RSVP global pool size of 100 Mbps and a sub pool size of 20 Mbps on all links in the topology. Only CSR5 and XRV11 are shown for brevity.

```
! CSR5
interface GigabitEthernet2.525
 ip rsvp bandwidth 100000 sub-pool 20000

interface GigabitEthernet2.550
 ip rsvp bandwidth 100000 sub-pool 20000

interface GigabitEthernet2.551
 ip rsvp bandwidth 100000 sub-pool 20000

! XRV11
rsvp
interface GigabitEthernet0/0/0/0.501
 bandwidth 100000 sub-pool 20000

interface GigabitEthernet0/0/0/0.512
```

```
bandwidth 100000 sub-pool 20000
```

```
interface GigabitEthernet0/0/0/0.551
bandwidth 100000 sub-pool 20000
```

As a side note, when configuring pre-standard or IETF RDM DS-TE, it is important to ensure that higher numbered BCs have less or equal bandwidth to the next lower BC. In this case, it doesn't make sense for the sub-pool (BC1) to have more bandwidth than the global pool (BC0).

```
CSR3(config-subif)#ip rsvp bandwidth rdm bc0 10000 bc1 20000
Reservable sub pool (pool1) bandwidth must be less than
or equal to RSVP Reservable Bandwidth.
```

We can verify the configuration was successful in both routers by checking the RSVP interface summary. The RSVP interface details don't reveal anything new, but show the same output in verbose fashion.

```
CSR5#show ip rsvp interface
interface  rsvp      allocated  i/f max  flow max  sub max  VRF
Gi2        ena        0          750M     750M      0
Gi2.525    ena        0          100M     100M      20M
Gi2.550    ena        0          100M     100M      20M
Gi2.551    ena        0          100M     100M      20M
```

```
RP/0/0/CPU0:XRv11#show rsvp interface
*: RDM: Default I/F B/W % : 75% [default] (max resv/bc0), 0% [default] (bc1)
Interface  MaxBW (bps)  MaxFlow (bps)  Allocated (bps)  MaxSub (bps)
-----
Gi0/0/0/0.501    100M        100M          0 ( 0%)         20M
Gi0/0/0/0.512    100M        100M          0 ( 0%)         20M
Gi0/0/0/0.551    100M        100M          0 ( 0%)         20M
```

```
CSR5#show ip rsvp interface detail gig2.550 | section Bandwidth
Bandwidth:
  Curr allocated: 0 bits/sec
  Max. allowed (total): 100M bits/sec
  Max. allowed (per flow): 100M bits/sec
  Max. allowed for LSP tunnels using sub-pools (pool 1): 20M bits/sec
  Set aside by policy (total): 0 bits/sec
```

```
RP/0/0/CPU0:XRv11#show rsvp interface gig0/0/0/0.501 detail
*: RDM: Default I/F B/W % : 75% [default] (max resv/bc0), 0% [default] (bc1)
INTERFACE: GigabitEthernet0/0/0/0.501 (ifh=0x400).
VRF ID: 0x60000000 (Default).
BW (bits/sec): Max=100M. MaxFlow=100M.
                Allocated=0 (0%). MaxSub=20M.
Signalling: No DSCP marking. No rate limiting.
States in: 0. Max missed msgs: 4.
Max out-of-band missed msgs: 38000.
```

Expiry timer: Not running. Refresh interval: 45s.
 Normal Refresh timer: Not running. Out-of-band refresh interval: 0s.
 Summary refresh timer: Not running.
 Refresh reduction local: Enabled. Summary Refresh: Enabled (1472 bytes max).
 Reliable summary refresh: Disabled. Bundling: Enabled. (1500 bytes max).
 Ack hold: 400 ms, Ack max size: 1500 bytes. Retransmit: 2100ms.

Checking the TE topology from CSR1 (a remote node) to look at CSR5's link to XRv11, we can see this sub-pool bandwidth is now available. All routers in IS-IS level 2 should see this. For completeness, we verify XRv11's link to CSR5 as well.

```
CSR1#show mpls traffic-eng topology igp-id isis 0000.0000.0005.00
IGP Id: 0000.0000.0005.00, MPLS TE Id:5.5.5.5 Router Node (isis level-2) id
24
```

```
link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0011.00, nbr_node_id:27,
gen:1698
frag_id: 0, Intf Address: 132.5.11.5, Nbr Intf Address: 132.5.11.11
TE metric: 10, IGP metric: 10, attribute flags: 0x5
SRLGs: None
physical_bw: 1000000 (kbps), max_reservable_bw_global: 100000 (kbps)
max_reservable_bw_sub: 20000 (kbps)
```

	Total Allocated BW (kbps)	Global Pool Reservable BW (kbps)	Sub Pool Reservable BW (kbps)
	-----	-----	-----
bw[0]:	0	100000	20000
bw[1]:	0	100000	20000
bw[2]:	0	100000	20000
bw[3]:	0	100000	20000
bw[4]:	0	100000	20000
bw[5]:	0	100000	20000
bw[6]:	0	100000	20000
bw[7]:	0	100000	20000
[snip]			

```
CSR1#show mpls traffic-eng topology 11.11.11.11
```

```
IGP Id: 0000.0000.0011.00, MPLS TE Id:11.11.11.11 Router Node (isis level-
2) id 27
link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0005.00, nbr_node_id:24,
gen:1570
frag_id: 0, Intf Address: 132.5.11.11, Nbr Intf Address: 132.5.11.5
TE metric: 10, IGP metric: 10, attribute flags: 0x5
SRLGs: None
physical_bw: 1000000 (kbps), max_reservable_bw_global: 100000 (kbps)
max_reservable_bw_sub: 20000 (kbps)
```

	Total Allocated BW (kbps)	Global Pool Reservable BW (kbps)	Sub Pool Reservable BW (kbps)
	-----	-----	-----
bw[0]:	0	100000	20000
bw[1]:	0	100000	20000
bw[2]:	0	100000	20000
bw[3]:	0	100000	20000
bw[4]:	0	100000	20000
bw[5]:	0	100000	20000
bw[6]:	0	100000	20000
bw[7]:	0	100000	20000
[snip]			

As a final verification, we quickly check the ISIS LSPDB for CSR5's LSP. The sub-pool bandwidth is clearly shown in kbps, broken down by priority.

```
CSR1#show isis database l2 verbose CSR5.00-00 | section XRv11
Metric: 10          IS-Extended XRv11.00
Affinity: 0x00000005
Interface IP Address: 132.5.11.5
Neighbor IP Address: 132.5.11.11
Physical BW: 1000000 kbits/sec
Reservable Global Pool BW: 100000 kbits/sec
Reservable Sub Pool BW: 20000 kbits/sec
Global Pool BW Unreserved:
  [0]: 100000 kbits/sec, [1]: 100000 kbits/sec
  [2]: 100000 kbits/sec, [3]: 100000 kbits/sec
  [4]: 100000 kbits/sec, [5]: 100000 kbits/sec
  [6]: 100000 kbits/sec, [7]: 100000 kbits/sec
Sub Pool BW Unreserved:
  [0]: 20000 kbits/sec, [1]: 20000 kbits/sec
  [2]: 20000 kbits/sec, [3]: 20000 kbits/sec
  [4]: 20000 kbits/sec, [5]: 20000 kbits/sec
  [6]: 20000 kbits/sec, [7]: 20000 kbits/sec
Metric: 10          IS (MT-IPv6) XRv11.00
```

Next, we will configure a DS-TE tunnel [ID 200] requesting 5 Mbps of sub-pool bandwidth from CSR1 to XRv11. The TE tunnel configuration is identical to other TE tunnels except the bandwidth is identified as "sub-pool". We will re-use existing path-options from previous labs for brevity. When the tunnel configuration is complete, we verify that it is functional and that the bandwidth is identified as "Sub".

```
! CSR1
interface Tunnel200
description PRE-STD DS-TE GLOBAL POOL
ip unnumbered Loopback0
tunnel mode mpls traffic-eng
tunnel destination 11.11.11.11
```

```
tunnel mpls traffic-eng autoroute destination
tunnel mpls traffic-eng priority 7 7
tunnel mpls traffic-eng bandwidth sub-pool 5000
tunnel mpls traffic-eng affinity 0x0 mask 0x0
tunnel mpls traffic-eng path-option 10 explicit name EP_NO_R9
```

```
CSR1#show mpls traffic-eng tunnels tunnel 200 | section Status|Config
Status:
  Admin: up          Oper: up          Path: valid          Signalling: connected
  path option 10, type explicit EP_NO_R9 (Basis for Setup, path weight 25)

Config Parameters:
  Bandwidth: 5000 kbps (Sub) Priority: 7 7 Affinity: 0x0/0x0
  Metric Type: TE (default)
  AutoRoute: disabled LockDown: disabled Loadshare: 5000 [400000] bw-based
  AutoRoute destination: enabled
  auto-bw: disabled
```

Debugging RSVP dump-messages and look at the outgoing PATH message, we can see that this is identified as “guaranteed service” with some generally irrelevant parameters. This is just a mechanism for RSVP to identify/classify the flow differently. Notice that the PATH message length is 252 bytes.

```
! CSR1
Outgoing Path:
  version:1 flags:0000 cksum:E808 ttl:255 reserved:0 length:252
SESSION          type 7 length 16:
  Tun Dest: 11.11.11.11 Tun ID: 200 Ext Tun ID: 1.1.1.1
[snip]
Guaranteed Service break bit=0 service length=8
          Path Delay (microseconds):0
          Path Jitter (microseconds):12
          Path delay since shaping (microseconds):0
          Path Jitter since shaping (microseconds):12
```

Checking RSVP, we can see that 5 Mbps was reserved. The RSVP interface summary does not show that these 5 Mbps are from the sub-pool, nor does the detailed command.

```
CSR1#show ip rsvp interface
```

interface	rsvp	allocated	i/f max	flow max	sub max	VRF
Gi2	ena	0	750M	750M	0	
Gi2.513	ena	0	100M	100M	20M	
Gi2.514	ena	0	100M	100M	20M	
Gi2.516	ena	5M	100M	100M	20M	
Gi2.518	ena	0	100M	100M	20M	
Gi2.519	ena	0	100M	100M	20M	

```
CSR1#show ip rsvp interface detail gig2.516 | section Bandwidth
Bandwidth:
```

```

Curr allocated: 5M bits/sec
Max. allowed (total): 100M bits/sec
Max. allowed (per flow): 100M bits/sec
Max. allowed for LSP tunnels using sub-pools (pool 1): 20M bits/sec
Set aside by policy (total): 0 bits/sec

```

The TED does a better job of differentiating between pools. We can tell these 5 Mbps are used from the sub-pool because it was subtracted from both the sub-pool and global-pool. Global-pool reservations are only subtracted from the global-pool. Similar to TE priorities, bandwidth reserved within more important pools is subtracted from lesser pools, but not vice versa.

```

CSR1#show mpls traffic-eng topology 1.1.1.1
[snip]
  link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0006.00, nbr_node_id:21,
gen:2618
  frag_id: 0, Intf Address: 132.1.6.1, Nbr Intf Address: 132.1.6.6
  TE metric: 5, IGP metric: 10, attribute flags: 0x4
  SRLGs: None
  physical_bw: 1000000 (kbps), max_reservable_bw_global: 100000 (kbps)
  max_reservable_bw_sub: 20000 (kbps)

```

	Total Allocated BW (kbps)	Global Pool Reservable BW (kbps)	Sub Pool Reservable BW (kbps)
	-----	-----	-----
bw[0]:	0	100000	20000
bw[1]:	0	100000	20000
bw[2]:	0	100000	20000
bw[3]:	0	100000	20000
bw[4]:	0	100000	20000
bw[5]:	0	100000	20000
bw[6]:	0	100000	20000
bw[7]:	5000	95000	15000

[snip]

To prove this, we temporarily change the bandwidth from sub-pool to global-pool on tunnel200 and look at the TED again. Now, the 5 Mbps was only subtracted from the global-pool. This proves that sub-pool reservations count against global pool reservations on a link. This makes sense because it ensures global-pool flows do not book more bandwidth than is available when sub-pool flows (more important) are also considered.

```

! CSR1
interface Tunnel200
  tunnel mpls traffic-eng bandwidth 5000

CSR1#show mpls traffic-eng topology 1.1.1.1
[snip]

```

```

link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0006.00, nbr_node_id:21,
gen:2762
frag_id: 0, Intf Address: 132.1.6.1, Nbr Intf Address: 132.1.6.6
TE metric: 5, IGP metric: 10, attribute flags: 0x4
SRLGs: None
physical_bw: 1000000 (kbps), max_reservable_bw_global: 100000 (kbps)
max_reservable_bw_sub: 20000 (kbps)

```

	Total Allocated BW (kbps)	Global Pool Reservable BW (kbps)	Sub Pool Reservable BW (kbps)
	-----	-----	-----
bw[0]:	0	100000	20000
bw[1]:	0	100000	20000
bw[2]:	0	100000	20000
bw[3]:	0	100000	20000
bw[4]:	0	100000	20000
bw[5]:	0	100000	20000
bw[6]:	0	100000	20000
bw[7]:	5000	95000	20000

[snip]

Debugging RSVP dump-messages again, we can see this is just a normal “controlled-load” service with no fancy options. The difference in RSVP PATH message length is 32 bytes (252 bytes for guaranteed service minus 220 bytes for controlled-load service) . Since the "service length" of the sub-pool PATH message is 8, I assume this is a measurement counting 4-byte words. Each of the sub-service fields is a 32-bit (4-byte) unsigned integer as a result.

```

! CSR1
Outgoing Path:
  version:1 flags:0000 cksum:FF75 ttl:255 reserved:0 length:220
SESSION          type 7 length 16:
  Tun Dest:   11.11.11.11  Tun ID: 200  Ext Tun ID: 1.1.1.1
[snip]
Controlled Load Service  break bit=0  service length=0

```

For additional verification, we perform a quick data-plane verification of the LSP. CSR1 is using RSVP-bound label 6016 to send traffic to CSR6. Using traceroute on CSR1, we verify that traffic to 11.11.11.11 is forwarded inside the TE tunnel.

```

CSR1#show ip rsvp reservation detail filter session-type 7 destination
11.11.11.11 | include Label
  Label: 6016 (outgoing)

CSR1#traceroute 11.11.11.11 source 1.1.1.1
Type escape sequence to abort.
Tracing the route to 11.11.11.11

```



```
VRF info: (vrf in name/id, vrf out name/id)
 1 132.1.6.6 [MPLS: Label 6016 Exp 0] 16 msec 8 msec 8 msec
 2 132.6.10.10 [MPLS: Label 10013 Exp 0] 16 msec 16 msec 25 msec
 3 132.10.11.11 9 msec * 21 msec
```

Before continuing, we change the 5 Mbps reservation back to sub-pool bandwidth. Unlike CBTS, all of the traffic towards XRv11's loopback will use this tunnel. The difference is that bandwidth is reserved differently; a bandwidth reservation on a single tunnel cannot reserve both global-pool and sub-pool bandwidth concurrently. Because of this, the underlying QoS queuing/shaping architecture should match the DS-TE configuration. We will create a quick policy on CSR1 to demonstrate this. The PMAP_SHAPE policy is applied to all 5 of CSR1's MPLS-facing sub-interfaces (application of policy is not shown). The math behind the 26 bytes of overhead is described next.

```
! CSR1
class-map match-all CMAP_VOICE
  match mpls experimental topmost 5

policy-map PMAP_QUEUE
  class CMAP_VOICE
    priority 5000
  class class-default
    queue-limit 16 packets

policy-map PMAP_SHAPE
  class class-default
    shape average 100000000  account user-defined 26
    service-policy PMAP_QUEUE
```

We know that 5 Mbps have been reserved; let's assume this is a layer 2 reservation which accounts for necessary overhead already. Cisco queuing and policing always accounts for layer 2 overhead, and shaping can optionally be configured to do so also. The overhead of packets traversing the network will vary depending on the MPLS label stack, but a safe assumption would be: 14 bytes for Ethernet, 4 bytes of 802.1q, and two 4-byte MPLS shim headers. This is a total of 26 bytes, and we can quickly prove it by using our QoS policy. Sending a single 100-byte packet at DSCP CS5 will show us how Cisco queuing accounts for overhead. The policy-map details show a single 126 byte packet, which verifies the 26 bytes of overhead as expected.

```
CSR7#ping 13.13.13.13 source 7.7.7.7 tos 160 repeat 1
Type escape sequence to abort.
Sending 1, 100-byte ICMP Echos to 13.13.13.13, timeout is 2 seconds:
Packet sent with a source address of 7.7.7.7
!
Success rate is 100 percent (1/1), round-trip min/avg/max = 21/21/21 ms
```

```
CSR1#show policy-map interface gig2.516 | section VOICE
  Class-map: CMAP_VOICE (match-all)
```

```
1 packets, 126 bytes
```

```
5 minute offered rate 0000 bps, drop rate 0000 bps
```

```
Match: mpls experimental topmost 5
```

```
Priority: 5000 kbps, burst bytes 125000, b/w exceed drops: 0
```

The math is not always precise. For example, sending VPN traffic to CSR8 results in only a single MPLS label due to PHP, so the overhead is 4 bytes less for a total of 22 bytes. This isn't terribly significant for DS-TE but it is worth mentioning. The reason for the detailed synchronization between RSVP reservations and QoS schedulers is because RSVP is only a control-plane reservation. DS-TE is useless without appropriate queuing mechanisms at the link-level. Because we stated that the 5 Mbps accounts for layer 2 overhead, the actual user traffic would be slightly less than 5 Mbps.

```
CSR7#ping 14.14.14.14 so 7.7.7.7 tos 160 repeat 1
```

```
Type escape sequence to abort.
```

```
Sending 1, 100-byte ICMP Echos to 14.14.14.14, timeout is 2 seconds:
```

```
Packet sent with a source address of 7.7.7.7
```

```
!
```

```
Success rate is 100 percent (1/1), round-trip min/avg/max = 2/2/2 ms
```

```
CSR1#show policy-map interface gig2.518 | section VOICE
```

```
Class-map: CMAP_VOICE (match-all)
```

```
1 packets, 122 bytes
```

```
5 minute offered rate 0000 bps, drop rate 0000 bps
```

```
Match: mpls experimental topmost 5
```

```
Priority: 5000 kbps, burst bytes 125000, b/w exceed drops: 0
```

For completeness, we will configure pre-standard DS-TE on XRv11 as well. The only difference from a standard TE-tunnel in XR is that the signalled-bandwidth is identified as sub-pool bandwidth. We can re-use old explicit-paths; later we will test FRR on the link between CSR3 and CSR10, so we want to use that link. Once configured, we verify the tunnel comes up and routes the way the explicit-path specified.

```
! XRv11
```

```
interface tunnel-te200
```

```
description PRE-STD DS-TE GLOBAL POOL
```

```
ipv4 unnumbered Loopback0
```

```
logging events all
```

```
signalled-bandwidth sub-pool 11000
```

```
destination 8.8.8.8
```

```
affinity ignore
```

```
path-option 10 explicit name PATH_11_5_2_3_10_6_8
```

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 200 role head | begin Path  
info
```

```
Path info (IS-IS 132 level-2):
```

```
Node hop count: 6
```

```
Hop0: 132.5.11.5
```

```
Hop1: 132.2.5.2
```

```

Hop2: 132.2.3.2
Hop3: 132.2.3.3
Hop4: 132.3.10.10
Hop5: 132.6.10.6
Hop6: 132.6.8.8
Hop7: 8.8.8.8

```

Checking XRv11's local TE topology information, we see that 11 Mbps of sub-pool bandwidth was allocated at priority 7 (worst). This was also reduced from the global pool, as expected, which is consistent with the RDM BC-model discussed later. It is interesting to note that XR shows the BC model is RDM, which is somewhat true since the pre-standard version behaves like IETF RDM in many ways.

```

RP/0/0/CPU0:XRv11#show mpls traffic-eng topology 11.11.11.11 | begin 0005
Link[0]:Point-to-Point, Nbr IGP Id:0000.0000.0005.00, Nbr Node Id:6,
gen:815

```

```

Frag Id:0, Intf Address:132.5.11.11, Intf Id:0
Nbr Intf Address:132.5.11.5, Nbr Intf Id:0
TE Metric:10, IGP Metric:10
Attribute Flags: 0x5
Ext Admin Group:
    Length: 256 bits
    Value : 0x::5

```

```

Attribute Names:
Unnamed bits    : 0 2
Switching Capability:None, Encoding:unassigned

```

```

BC Model ID:RDM
Physical BW:1000000 (kbps), Max Reservable BW Global:100000 (kbps)
Max Reservable BW Sub:20000 (kbps)

```

	Total Allocated BW (kbps)	Global Pool Reservable BW (kbps)	Sub Pool Reservable BW (kbps)
	-----	-----	-----
bw[0]:	0	100000	20000
bw[1]:	0	100000	20000
bw[2]:	0	100000	20000
bw[3]:	0	100000	20000
bw[4]:	0	100000	20000
bw[5]:	0	100000	20000
bw[6]:	0	100000	20000
bw[7]:	11000	89000	9000

To test FRR with DS-TE, we will begin by modifying an existing NHOP tunnel on CSR3. This is an NHOP backup tunnel that requires green, blue, and orange colors on a link. The tunnel routes via CSR2 as this is the only option meeting the affinity constraints. The tunnel currently does not specify any backup bandwidth.

```

! CSR3
interface Tunnel130

```

```

description REPAIR PATH NHOP (PLR)
ip unnumbered Loopback0
tunnel mode mpls traffic-eng
tunnel destination 10.10.10.10
tunnel mpls traffic-eng affinity 0xE mask 0xE
tunnel mpls traffic-eng path-option 10 explicit name PATH_AVOID_CSR3_CSR10

```

```

CSR3#show mpls traffic-eng tunnels tunnel 30 | section RSVP Path
  RSVP Path Info:
    My Address: 132.2.3.3
    Explicit Route: 132.2.3.2 132.2.10.10 10.10.10.10
    Record Route: NONE
    Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits

```

For FRR to be effective within the context of DS-TE, we need tunnels that request FRR on CSR1 and XRv11, along with bandwidth protection. We add two tunnels [ID 210 and 211] to test this. Tunnel210 will request global pool bandwidth while tunnel211 will request sub-pool bandwidth. We can look at the RSVP RESV RRO on both CSR1 and XRv11 to see the tunnels are protected at CSR3, the PLR. First, we configure and verify the CSR1 tunnels. The NHOP backup tunnel currently has "unlimited" bandwidth since no backup-bandwidth is specified. Furthermore, not specifying a bandwidth type, such as global or sub-pool, means that the tunnel can back up a tunnel requesting bandwidth from any pool. We also shut down tunnel200 (not shown) so that it does not interfere with this test.

```

! CSR1
interface Tunnel210
  description DS-TE GLOBAL-POOL BW FRR
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng priority 7 7
  tunnel mpls traffic-eng bandwidth 9000
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 explicit name EP_1_4_3_10_11
  tunnel mpls traffic-eng fast-reroute bw-protect

interface Tunnel211
  description DS-TE SUB-POOL BW FRR
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng priority 7 7
  tunnel mpls traffic-eng bandwidth sub-pool 3000
  tunnel mpls traffic-eng affinity 0x0 mask 0x0
  tunnel mpls traffic-eng path-option 10 explicit name EP_1_4_3_10_11
  tunnel mpls traffic-eng fast-reroute bw-protect

```

Since tunnels 210 and 211 are requesting FRR, their RSVP RESV RROs should reflect the available NHOP protection. In the FRR section, we saw that backup tunnels with no "backup-bw" identified were

assumed to have unlimited backup bandwidth. Despite requesting bandwidth protection, the tunnel does not specify any, so the assumption of having "unlimited" bandwidth really means that no bandwidth protection is provided. For this reason, CSR1 sees the PLR as offering local NHOP protection, but no bandwidth protection.

```
CSR1#show ip rsvp reservation detail filter session-type 7 | section
Tun_ID|RRO
  Tun Dest:   11.11.11.11  Tun ID: 210  Ext Tun ID: 1.1.1.1
  RRO:
    4.4.4.4/32, Flags:0x20 (No Local Protection, Node-id)
      Label subobject: Flags 0x1, C-Type 1, Label 4001
    3.3.3.3/32, Flags:0x21 (Local Prot Avail/to NHOP, Node-id)
      Label subobject: Flags 0x1, C-Type 1, Label 3007
    10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
      Label subobject: Flags 0x1, C-Type 1, Label 10008
    11.11.11.11/32, Flags:0x20 (No Local Protection, Node-id)
      Label subobject: Flags 0x1, C-Type 1, Label 0
    132.10.11.11/32, Flags:0x0 (No Local Protection)
      Label subobject: Flags 0x1, C-Type 1, Label 0
  Tun Dest:   11.11.11.11  Tun ID: 211  Ext Tun ID: 1.1.1.1
  RRO:
    4.4.4.4/32, Flags:0x20 (No Local Protection, Node-id)
      Label subobject: Flags 0x1, C-Type 1, Label 4008
    3.3.3.3/32, Flags:0x21 (Local Prot Avail/to NHOP, Node-id)
      Label subobject: Flags 0x1, C-Type 1, Label 3013
    10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
      Label subobject: Flags 0x1, C-Type 1, Label 10013
    11.11.11.11/32, Flags:0x20 (No Local Protection, Node-id)
      Label subobject: Flags 0x1, C-Type 1, Label 0
    132.10.11.11/32, Flags:0x0 (No Local Protection)
      Label subobject: Flags 0x1, C-Type 1, Label 0
```

Next, we configure the equivalent tunnels on XRv11. The tunnels are virtually identical with different bandwidth reservations in different pools.

```
! XRv11
interface tunnel-te210
  description DS-TE GLOBAL-POOL BW FRR
  ipv4 unnumbered Loopback0
  logging events all
  signalled-bandwidth 11000
  destination 8.8.8.8
  fast-reroute protect bandwidth
  affinity ignore
  path-option 10 explicit name PATH_11_5_2_3_10_6_8

interface tunnel-te211
  description DS-TE SUB-POOL BW FRR
```

```
ipv4 unnumbered Loopback0
logging events all
signalled-bandwidth sub-pool 4000
destination 8.8.8.8
fast-reroute protect bandwidth
affinity ignore
path-option 10 explicit name PATH_11_5_2_3_10_6_8
```

Like the CSR1 DS-TE tunnels, both tunnels are requesting bandwidth protection but it is not available. Local protection is supported at CSR3 for both tunnels, though.

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 210 detail | begin Resv Info
Resv Info:
Record Route:
  IPv4 5.5.5.5, flags 0x20 (Node-ID)
  Label 5017, flags 0x1
  IPv4 2.2.2.2, flags 0x20 (Node-ID)
  Label 2018, flags 0x1
  IPv4 3.3.3.3, flags 0x21 (Node-ID, Protection: available)
  Label 3021, flags 0x1
  IPv4 10.10.10.10, flags 0x20 (Node-ID)
  Label 10021, flags 0x1
  IPv4 6.6.6.6, flags 0x20 (Node-ID)
  Label 6015, flags 0x1
  IPv4 8.8.8.8, flags 0x20 (Node-ID)
  Label 0, flags 0x1
Fspec: avg rate=11000 kbits, burst=1000 bytes, peak rate=11000 kbits
```

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 211 detail | begin Resv Info
Resv Info:
Record Route:
  IPv4 5.5.5.5, flags 0x20 (Node-ID)
  Label 5018, flags 0x1
  IPv4 2.2.2.2, flags 0x20 (Node-ID)
  Label 2019, flags 0x1
  IPv4 3.3.3.3, flags 0x21 (Node-ID, Protection: available)
  Label 3022, flags 0x1
  IPv4 10.10.10.10, flags 0x20 (Node-ID)
  Label 10022, flags 0x1
  IPv4 6.6.6.6, flags 0x20 (Node-ID)
  Label 6016, flags 0x1
  IPv4 8.8.8.8, flags 0x20 (Node-ID)
  Label 0, flags 0x1
Fspec: avg rate=4000 kbits, burst=1000 bytes, peak rate=4000 kbits
```

CSR3 sees all four tunnels in its FRR database as well. Each one is mapped to tunnel30, the NHOP tunnel with unlimited bandwidth servicing "any" pool.

```
CSR3#show mpls traffic-eng fast-reroute database | begin midpoint
```

P2P LSP midpoint frr information:

LSP identifier	In-label	Out	intf/label	FRR intf/label	Status
-----	-----	-----	-----	-----	-----
1.1.1.1 210 [25]	3023	Gi2.530:10023	Tu30:10023	ready	
1.1.1.1 211 [25]	3024	Gi2.530:10024	Tu30:10024	ready	
11.11.11.11 210 [5]	3021	Gi2.530:10021	Tu30:10021	ready	
11.11.11.11 211 [5]	3022	Gi2.530:10022	Tu30:10022	ready	

The backup tunnel details show that "any" pool is assumed, since no pool was specified. Also, the total in-use bandwidth on the tunnel is 27 Mbps, none of which is actually protected. 27 Mbps is the sum of all 4 tunnels requested bandwidth but does not specify global or sub pool.

```
CSR3#show mpls traffic-eng tunnels tunnel 30 backup
REPAIR PATH NHOP (PLR)
  LSP Head, Admin: up, Oper: up
  Tun ID: 30, LSP ID: 24, Source: 3.3.3.3
  Destination: 10.10.10.10
  Fast Reroute Backup Provided:
    Protected i/fs: Gi2.530
    Protected LSPs/Sub-LSPs: 4, Active: 0
    Backup BW: any pool unlimited; inuse: 27000 kbps
  Backup flags: 0x0
```

As a test, we will configure the tunnel to offer 7 Mbps of sub-pool backup bandwidth. This is just enough to backup both tunnel211 interfaces on CSR1 and XRv11. At this point, both tunnel210 interfaces now have no protection at all, since the NHOP tunnel services only sub-pool flows. The FRR tunnel has now specified both the type and quantity of bandwidth, making it much more restrictive/selective in terms of which LSPs it is capable of protecting.

```
! CSR3
interface Tunnel30
  tunnel mpls traffic-eng backup-bw sub-pool 7000
```

We can verify that only the tunnel211 interfaces are backed up on the PLR by checking the FRR database and the tunnel30 backup details. The bandwidth type is sub-pool and the amount is 7 Mbps.

```
CSR3#show mpls traffic-eng fast-reroute database | begin midpoint
P2P LSP midpoint frr information:
LSP identifier      In-label Out intf/label  FRR intf/label  Status
-----
1.1.1.1 211 [25]    3024     Gi2.530:10024    Tu30:10024      ready
11.11.11.11 211 [5] 3022     Gi2.530:10022    Tu30:10022      ready
```

```
CSR3#show mpls traffic-eng tunnels tunnel 30 backup
REPAIR PATH NHOP (PLR)
  LSP Head, Admin: up, Oper: up
  Tun ID: 30, LSP ID: 24, Source: 3.3.3.3
```

```

Destination: 10.10.10.10
Fast Reroute Backup Provided:
  Protected i/fs: Gi2.530
  Protected LSPs/Sub-LSPs: 2, Active: 0
  Backup BW: sub-pool; limit: 7000 kbps, inuse: 7000 kbps (BWP inuse: 7000 kbps)
  Backup flags: 0x0

```

On CSR1, we notice that tunnel210 is no longer protected by the NHOP tunnel at all. This is because it is requesting global pool bandwidth and the NHOP tunnel only provides sub-pool bandwidth backup. Tunnel211, however, now has local protection and available bandwidth, since it is requesting sub-pool bandwidth and the backup tunnel has a specific amount of bandwidth available (versus unlimited). Tunnel210 would not be protected even if it wasn't asking for bandwidth-protection simply because the bandwidth pools are mismatched between the head-end and PLR tunnels.

```

CSR1#show ip rsvp reservation detail filter session-type 7 | section
Tun_ID|RRO
Tun Dest: 11.11.11.11 Tun ID: 210 Ext Tun ID: 1.1.1.1
RRO:
  4.4.4.4/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 4015
  3.3.3.3/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3023
  10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 10023
  11.11.11.11/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 0
  132.10.11.11/32, Flags:0x0 (No Local Protection)
    Label subobject: Flags 0x1, C-Type 1, Label 0
Tun Dest: 11.11.11.11 Tun ID: 211 Ext Tun ID: 1.1.1.1
RRO:
  4.4.4.4/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 4016
  3.3.3.3/32, Flags:0x25 (Local Prot Avail/Has BW/to NHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3024
  10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 10024
  11.11.11.11/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 0
  132.10.11.11/32, Flags:0x0 (No Local Protection)
    Label subobject: Flags 0x1, C-Type 1, Label 0

```

Checking XRv11, we see the same effect. Tunnel210 now has no protection at all, while tunnel211 now has bandwidth protection via the NHOP tunnel on CSR3.

```

RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 210 detail | begin Resv Info
Resv Info:
Record Route:

```



```

IPv4 5.5.5.5, flags 0x20 (Node-ID)
Label 5017, flags 0x1
IPv4 2.2.2.2, flags 0x20 (Node-ID)
Label 2018, flags 0x1
IPv4 3.3.3.3, flags 0x20 (Node-ID)
Label 3021, flags 0x1
IPv4 10.10.10.10, flags 0x20 (Node-ID)
Label 10021, flags 0x1
IPv4 6.6.6.6, flags 0x20 (Node-ID)
Label 6015, flags 0x1
IPv4 8.8.8.8, flags 0x20 (Node-ID)
Label 0, flags 0x1
Fspec: avg rate=11000 kbits, burst=1000 bytes, peak rate=11000 kbits

```

```

RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 211 detail | begin Resv Info
Resv Info:

```

```

Record Route:
IPv4 5.5.5.5, flags 0x20 (Node-ID)
Label 5018, flags 0x1
IPv4 2.2.2.2, flags 0x20 (Node-ID)
Label 2019, flags 0x1
IPv4 3.3.3.3, flags 0x25 (Node-ID, Protection: available, b/w)
Label 3022, flags 0x1
IPv4 10.10.10.10, flags 0x20 (Node-ID)
Label 10022, flags 0x1
IPv4 6.6.6.6, flags 0x20 (Node-ID)
Label 6016, flags 0x1
IPv4 8.8.8.8, flags 0x20 (Node-ID)
Label 0, flags 0x1
Fspec: avg rate=4000 kbits, burst=1000 bytes, peak rate=4000 kbits

```

We can also specify the type of bandwidth being protected without specifying an amount. This will limit to tunnel to protecting only a specific pool, as seen above, but not specifying a bandwidth value. In this example, we adjust the NHOP tunnel to support only global pool but with unlimited bandwidth. Now, the FRR database on the PLR shows protection for tunnel210 interfaces since the global pool NHOP tunnel cannot backup sub pool LSPs. There is no limit on the bandwidth and 20 Mbps are in use (sum of CSR1 and XRv11 tunnel210 global pool bandwidth requests).

```

! CSR3
interface Tunnel30
 tunnel mpls traffic-eng backup-bw global-pool unlimited

```

```

CSR3#show mpls traffic-eng fast-reroute database | begin midpoint
P2P LSP midpoint frr information:

```

LSP identifier	In-label	Out intf/label	FRR intf/label	Status
1.1.1.1 210 [30]	3008	Gi2.530:10014	Tu30:10014	ready
11.11.11.11 210 [5]	3021	Gi2.530:10021	Tu30:10021	ready

```

CSR3#show mpls traffic-eng tunnels tunnel 30 backup
REPAIR PATH NHOP (PLR)
  LSP Head, Admin: up, Oper: up
  Tun ID: 30, LSP ID: 24, Source: 3.3.3.3
  Destination: 10.10.10.10
  Fast Reroute Backup Provided:
    Protected i/fs: Gi2.530
    Protected LSPs/Sub-LSPs: 2, Active: 0
    Backup BW: global pool; limit 0 kbps, inuse: 20000 kbps (BWP inuse: 0
kbps)
  Backup flags: 0x1

```

As expected, CSR1's tunnel210 is protected again, but without bandwidth protection. We only specified the type of bandwidth, not the amount, so unlimited bandwidth really means no bandwidth. Tunnel211 currently has no protection at all. This is slightly different from the previous example since the protected LSPs are not claiming to have bandwidth protection.

```

CSR1#show ip rsvp reservation detail filter session-type 7 | section
Tun_ID|RRO
  Tun Dest:   11.11.11.11   Tun ID: 210   Ext Tun ID: 1.1.1.1
  RRO:
    4.4.4.4/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 4017
    3.3.3.3/32, Flags:0x21 (Local Prot Avail/to NHOP, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3008
    10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 10014
    11.11.11.11/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 0
    132.10.11.11/32, Flags:0x0 (No Local Protection)
    Label subobject: Flags 0x1, C-Type 1, Label 0
  Tun Dest:   11.11.11.11   Tun ID: 211   Ext Tun ID: 1.1.1.1
  RRO:
    4.4.4.4/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 4001
    3.3.3.3/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 3017
    10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 10017
    11.11.11.11/32, Flags:0x20 (No Local Protection, Node-id)
    Label subobject: Flags 0x1, C-Type 1, Label 0
    132.10.11.11/32, Flags:0x0 (No Local Protection)
    Label subobject: Flags 0x1, C-Type 1, Label 0

```

The same is true on XRv11. Tunnel211 now has no protection while tunnel210 has NHOP protection without any backup bandwidth guarantee. The point is that specifying the bandwidth type, and not the amount, can be used to differentiate between tunnels used for backups.

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 210 detail | begin Resv Info
Resv Info:
```

```
Record Route:
```

```
IPv4 5.5.5.5, flags 0x20 (Node-ID)
```

```
Label 5017, flags 0x1
```

```
IPv4 2.2.2.2, flags 0x20 (Node-ID)
```

```
Label 2018, flags 0x1
```

```
IPv4 3.3.3.3, flags 0x21 (Node-ID, Protection: available)
```

```
Label 3021, flags 0x1
```

```
IPv4 10.10.10.10, flags 0x20 (Node-ID)
```

```
Label 10021, flags 0x1
```

```
IPv4 6.6.6.6, flags 0x20 (Node-ID)
```

```
Label 6015, flags 0x1
```

```
IPv4 8.8.8.8, flags 0x20 (Node-ID)
```

```
Label 0, flags 0x1
```

```
Fspec: avg rate=11000 kbits, burst=1000 bytes, peak rate=11000 kbits
```

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 211 detail | begin Resv Info
Resv Info:
```

```
Record Route:
```

```
IPv4 5.5.5.5, flags 0x20 (Node-ID)
```

```
Label 5018, flags 0x1
```

```
IPv4 2.2.2.2, flags 0x20 (Node-ID)
```

```
Label 2019, flags 0x1
```

```
IPv4 3.3.3.3, flags 0x20 (Node-ID)
```

```
Label 3022, flags 0x1
```

```
IPv4 10.10.10.10, flags 0x20 (Node-ID)
```

```
Label 10022, flags 0x1
```

```
IPv4 6.6.6.6, flags 0x20 (Node-ID)
```

```
Label 6016, flags 0x1
```

```
IPv4 8.8.8.8, flags 0x20 (Node-ID)
```

```
Label 0, flags 0x1
```

```
Fspec: avg rate=4000 kbits, burst=1000 bytes, peak rate=4000 kbits
```

To demonstrate multiple backup tunnel selection, we will configure a pair of basic NNHOP tunnels on CSR3 [ID 31 and 32]. The tunnel logic is recycled from earlier but modified for this test. Tunnel31 will provide 5 Mbps of bandwidth for the sub-pool, which will offer protection for it. Tunnel32 won't specify a bandwidth type, but will offer 25 Mbps of general bandwidth (any pool). Both tunnels avoid CSR10 completely. We need two separate tunnels since CSR1 and XRv11 headends have different NNHOPs. As mentioned earlier, this is why NHOP protection often scales better at the expense of less resiliency.

```
! CSR3
interface Tunnel31
  description REPAIR PATH NNHOP (PLR) FOR CSR1
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
```

```
tunnel mpls traffic-eng backup-bw sub-pool 5000
tunnel mpls traffic-eng affinity 0x0 mask 0x0
tunnel mpls traffic-eng path-option 10 explicit name PATH_AVOID_CSR10
```

```
interface Tunnel32
description REPAIR PATH NNHOP (PLR) FOR XRv11
ip unnumbered Loopback0
tunnel mode mpls traffic-eng
tunnel destination 6.6.6.6
tunnel mpls traffic-eng backup-bw 25000
tunnel mpls traffic-eng affinity 0x0 mask 0x0
tunnel mpls traffic-eng path-option 10 explicit name PATH_AVOID_CSR10
```

We quickly verify the paths used by the tunnels. Tunnel31 routes via CSR2 and CSR5 to reach XRv11 to protect CSR1's LSP. Tunnel32 routes via CSR1 to reach CSR6 to protect XRv11's LSP.

```
CSR3#show ip rsvp sender detail filter session-type 7 tunnel 31 | section
outgoing
```

```
ERO: (outgoing)
132.2.3.2 (Strict IPv4 Prefix, 8 bytes, /32)
132.2.5.5 (Strict IPv4 Prefix, 8 bytes, /32)
132.5.11.11 (Strict IPv4 Prefix, 8 bytes, /32)
11.11.11.11 (Strict IPv4 Prefix, 8 bytes, /32)
```

```
CSR3#show ip rsvp sender detail filter session-type 7 tunnel 32 | section
outgoing
```

```
ERO: (outgoing)
132.1.3.1 (Strict IPv4 Prefix, 8 bytes, /32)
132.1.6.6 (Strict IPv4 Prefix, 8 bytes, /32)
6.6.6.6 (Strict IPv4 Prefix, 8 bytes, /32)
```

When there are multiple competing tunnels, the pool type adds a new dimension to the backup tunnel selection. Below is a chart summarizing the selection process. The chart can be generalized with a few general comments:

1. *NNHOP tunnels are preferred over NHOP tunnels*
2. *A tunnel with limited (specified) bandwidth is preferred over unlimited bandwidth*
3. *Specific pool tunnels (global, sub, or specific BC) are preferred over "any" (unspecified) pool tunnels*

Preference	Tunnel type	Bandwidth type	Bandwidth amount
1 (best)	NNHOP	Specific pool/BC	Limited
2	NNHOP	Any	Limited
3	NNHOP	Specific pool/BC	Unlimited
4	NNHOP	Any	Unlimited
5	NHOP	Specific pool/BC	Limited
6	NHOP	Any	Limited
7	NHOP	Specific pool/BC	Unlimited

8 (worst)	NHOP	Any	Unlimited

The FRR database on CSR3 shows very interesting results. We will analyze each row of the database individually. For clarity, the tunnels and their backup-bandwidths are shown again.

```
! CSR3
interface Tunnel30
 tunnel mpls traffic-eng backup-bw global-pool unlimited

interface Tunnel31
 tunnel mpls traffic-eng backup-bw sub-pool 5000

interface Tunnel32
 tunnel mpls traffic-eng backup-bw 25000
```

First, we see tunnel210 from CSR1. It is protected by the original NHOP tunnel30, which offers unlimited global-pool bandwidth. This is option #7 on the chart above, making it relatively undesirable. This LSP cannot choose tunnel31 since that NNHOP backup tunnel protects only sub-pool traffic (limited quantity), which would be option #1. This type of tunnel is the most desirable choice but also the most restrictive/selective. It also cannot choose tunnel32 because this tunnel goes to an incorrect NNHOP for this LSP, but it would have been a better choice since any limited bandwidth is better than specific unlimited bandwidth.

Second, we see tunnel211 also from CSR1. This uses tunnel31 for backup, which offers NNHOP protection for limited sub-pool bandwidth. This is the best possible option. The LSP could not use tunnel30, since it was requesting sub pool bandwidth while tunnel30 only offered global pool protection.

Third is tunnel210 from XRv11. It is protected by tunnel32, offering NNHOP protection with a limited bandwidth amount in any pool (option #2). The LSP could have potentially used tunnel30 as well, since it was requesting global pool bandwidth, but option #2 is better than option #7.

Fourth is tunnel211 from XRv11. It also uses tunnel32 for protection since it is the only feasible option. Tunnel30 is not a candidate since it only backs up global pool bandwidth but the head-end requests sub-pool bandwidth.

```
CSR3#show mpls traffic-eng fast-reroute database | begin midpoint
P2P LSP midpoint frr information:
LSP identifier          In-label Out intf/label    FRR intf/label    Status
-----
1.1.1.1 210 [31]        3020     Gi2.530:10019          Tu30:10019        ready
1.1.1.1 211 [28]        3017     Gi2.530:10017          Tu31:implicit-nu  ready
11.11.11.11 210 [5]     3021     Gi2.530:10021          Tu32:6015         ready
11.11.11.11 211 [5]     3022     Gi2.530:10022          Tu32:6016         ready
```

If we temporarily shut down tunnel32, we notice 2 changes. CSR1's tunnels are not affected at all, since they selected other available backups in the first place. XRv11 tunnel210, which selected tunnel32, now chooses tunnel30. As mentioned earlier, tunnel30 was a candidate for protection but was not selected since it was less desirable. XRv11 tunnel211 now has no feasible backup paths, and thus is unprotected. When every LSR in a network is also a candidate PLR (which is common), there are likely to be many backup tunnels. This selection process is important to determine the sequence in which LSPs are protected by specific FRR tunnels.

```
CSR3#show mpls traffic-eng fast-reroute database | begin midpoint
P2P LSP midpoint frr information:
```

LSP identifier	In-label	Out intf/label	FRR intf/label	Status
1.1.1.1 210 [31]	3020	Gi2.530:10019	Tu30:10019	ready
1.1.1.1 211 [28]	3017	Gi2.530:10017	Tu31:implicit-nu	ready
11.11.11.11 210 [5]	3021	Gi2.530:10021	Tu30:10021	ready

When tunnel32 comes back up, the PLR notifies the head-end that protection is available within the RSVP RESV message without any user intervention. The head-end does not need to resignal the LSP, which is why the LSP IDs remain unchanged, as do the TE labels. The LSP promotion for backup tunnels can be controlled by the FRR timers discussed earlier.

```
CSR3#show mpls traffic-eng fast-reroute database | begin midpoint
P2P LSP midpoint frr information:
```

LSP identifier	In-label	Out intf/label	FRR intf/label	Status
1.1.1.1 210 [31]	3020	Gi2.530:10019	Tu30:10019	ready
1.1.1.1 211 [28]	3017	Gi2.530:10017	Tu31:implicit-nu	ready
11.11.11.11 210 [5]	3021	Gi2.530:10021	Tu32:6015	ready
11.11.11.11 211 [5]	3022	Gi2.530:10022	Tu32:6016	ready

Additional Reading – Reference configurations “ds-te-pre-std”

31.4.2 IETF Russian Dolls Model (RDM)

The lab is a continuation from the DS-TE pre-standard lab. The Russian Dolls Model (RDM) method is the default IETF DS-TE implementation in Cisco routers. It is defined in RFC 4127 and allows different CTs to share bandwidth on a link. This allows for most efficient bandwidth sharing between different CTs, but still allows for preemption when higher priority classes require bandwidth. The IETF implementation is vendor interoperability and extends the sub-pool concept to multiple CTs, briefly described earlier. To enable IETF DS-TE in RDM mode, only one command is required.

```
! All MPLS XE routers
mpls traffic-eng ds-te mode ietf
```

```
! All MPLS XR routers
mpls traffic-eng
```

```
ds-te mode ietf
```

Although we have not defined any bandwidth constraints (BC), entering this global command changes quite a bit. In XE, all of the existing RSVP bandwidth pools configured at the link level are converted to the IETF RDM format. The global pool is identified as BC0 and the sub-pool is identified as BC1, but otherwise the command is very similar. Because the pre-standard DS-TE implementation is based on RDM, the configuration can be easily converted back and forth by IOS. Either syntax is accepted, and both XE and XR don't care either way. XR does not change the commands from the global/sub pool format by default, so we will leave XRv12 with the legacy global/sub pool commands since they still work. I will update XRv11 for completeness to show the alternative (new) syntax.

```
! CSR1, this happens automatically
interface GigabitEthernet2.516
 ip rsvp bandwidth rdm bc0 100000 bc1 20000
```

```
! XRv11, must be configured manually
rsvp
 interface GigabitEthernet0/0/0/0.501
   bandwidth rdm bc0 100000 bc1 20000
```

```
interface GigabitEthernet0/0/0/0.512
 bandwidth rdm bc0 100000 bc1 20000
```

```
interface GigabitEthernet0/0/0/0.551
 bandwidth rdm bc0 100000 bc1 20000
```

The RSVP commands, like before, are not terribly useful for DS-TE and haven't changed much. RSVP still uses terms like "global" and "sub" pool. The existing tunnel200 is still online and requesting 5 Mbps of sub-pool bandwidth.

```
CSR1#show ip rsvp interface
```

interface	rsvp	allocated	i/f max	flow max	sub max	VRF
Gi2	ena	0	750M	750M	0	
Gi2.513	ena	0	100M	100M	20M	
Gi2.514	ena	0	100M	100M	20M	
Gi2.516	ena	5M	100M	100M	20M	
Gi2.518	ena	0	100M	100M	20M	
Gi2.519	ena	0	100M	100M	20M	

Looking at the MPLS TE show commands for TE-classes, we can see there is new information. A small table is shown which defines 4 default TE-classes out of a possible 64. Only class-types 0 (global) and 1 (sub) are currently defined, but the 8 priority values still exist. XE routers appear capable of carrying up to 8 TE classes at any time, as seen in the TED later. The chart below seems to indicate that higher number TE-classes are more important. For example, TE-class 0 uses global-pool bandwidth (CT 0) and

has a priority of 7, which is the worst. TE-class 5 use sub-pool bandwidth (CT 1) and has a priority of 0, which is the best.

```
CSR1#show mpls traffic-eng ds-te te-class
DS-TE Mode: IETF
```

TE-Class	Class-Type	Priority
* 0	0	7
* 1	1	7
* 4	0	0
* 5	1	0
* - default setting		

Class-Type: 0 = Global-pool, 1 = Sub-pool

The TE tunnel configured earlier is still operational in the new RDM-based DS-TE network. Looking at the TED on CSR1, and specifically looking at the link to CSR6, we can see the output is considerably different. It shows us the RDM model, along with BC0 and BC1 reservable bandwidth quantities. A BC defines how much bandwidth a CT can use. Thus, BC0 is 100 Mbps and is the maximum reservable bandwidth between all TE-classes within CT0. BC1 is 20 Mbps and is the maximum reservable bandwidth for TE-classes using CT1. The bandwidth reservations shown below are confusing at first glance. TE-class 1 represents sub-pool flows that are priority 7. The existing TE tunnel is priority 7, the default, because we never set it. It is also using sub-pool bandwidth (CT1), so we know it is part of TE-class 1. The 5 Mbps reservation is counted against TE-class 1 and all lower classes as well; this behavior is consistent with non-DS-TE as well when only bandwidth prioritization for global-pool resources was considered. TE-class 0 is also priority 7 but represents the global-pool bandwidth, so reducing it by 5 Mbps when a sub-pool flow of the same priority is created is consistent with pre-standard DS-TE as well. TE-classes 4 and 5, despite being CT0 and CT1 respectively, are not affected by this reservation.

```
CSR1#show mpls traffic-eng topology 1.1.1.1
[snip]
```

```
link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0006.00, nbr_node_id:21,
gen:2772
```

```
frag_id: 0, Intf Address: 132.1.6.1, Nbr Intf Address: 132.1.6.6
TE metric: 5, IGP metric: 10, attribute flags: 0x4
SRLGs: None
physical_bw: 1000000 (kbps),
```

BC Model Id: RDM

BC0 (max_reservable): 100000 (kbps)

BC0 (max_reservable_bw_global): 100000 (kbps)

BC1 (max_reservable_bw_sub): 20000 (kbps)

	Total Allocated BW (kbps)	Reservable BW (kbps)
	-----	-----
TE-Class[0]:	5000	95000


```

TE-Class[1]:          5000          15000
TE-Class[2]:           0             0
TE-Class[3]:           0             0
TE-Class[4]:           0          100000
TE-Class[5]:           0          20000
TE-Class[6]:           0             0
TE-Class[7]:           0             0
[snip]

```

We also check the ISIS LSP. This differs significantly from the pre-standard version since there isn't a separate accounting of global and sub pools. Instead, the BC-TLV signals that the IETF RDM model is being used. The "global pool" bandwidth represents the bandwidth of BC0, which is the all-encompassing bandwidth value. 100 Mbps are available to BC0 with 20 Mbps of it being available to BC1, as shown in the LSP. For brevity, we examine the link to CSR8. The TE-classes are carried instead of the bandwidth pool types.

```

CSR1#show isis database verbose CSR1-00.00 | section CSR8
Metric: 10          IS (MT-IPv6) CSR8.00
Metric: 10          IS-Extended CSR8.00
Affinity: 0x00000004
Interface IP Address: 132.1.8.1
Neighbor IP Address: 132.1.8.8
Physical BW: 1000000 kbits/sec
Reservable Global Pool BW: 100000 kbits/sec
TE-Class BW Unreserved:
  [0]: 100000 kbits/sec, [1]: 20000 kbits/sec
  [2]: 0 kbits/sec, [3]: 0 kbits/sec
  [4]: 100000 kbits/sec, [5]: 20000 kbits/sec
  [6]: 0 kbits/sec, [7]: 0 kbits/sec
BC-TLV Header
  BC Model-Id:RDM
Bandwidth constraints[0]: 100000 kbits/sec
Bandwidth constraints[1]: 20000 kbits/sec

```

The CT of a TE tunnel is carried inside of the RSVP PATH message as a class type object (CTO). The Cisco pre-standard version could not use this CTO since it required changes to RSVP; it is specified to the IETF BC models. When CSR1 signals this tunnel, the CT is carried along the path so routers know how to allocate bandwidth for it when multiple BC pools exist. The TE-class itself is not explicitly signaled in RSVP, which is why all routers must agree on what the TE-class mappings are. The combination of the CT and the LSP priority (carried in the SAO) make up the TE-class.

```

R1#debug ip rsvp dump-message
Outgoing Path:
  version:1 flags:0000 cksum:A908 ttl:255 reserved:0 length:260
SESSION          type 7 length 16:
  Tun Dest: 11.11.11.11 Tun ID: 200 Ext Tun ID: 1.1.1.1

```

```

HOP                                type 1 length 12:
  Hop Addr: 132.1.6.1 LIH: 0x0000000D
TIME_VALUES                        type 1 length 8 :
  Refresh Period (msec): 30000
EXPLICIT_ROUTE                    type 1 length 36:
  132.1.6.6 (Strict IPv4 Prefix, 8 bytes, /32)
  132.6.10.10 (Strict IPv4 Prefix, 8 bytes, /32)
  132.10.11.11 (Strict IPv4 Prefix, 8 bytes, /32)
  11.11.11.11 (Strict IPv4 Prefix, 8 bytes, /32)
LABEL_REQUEST                     type 1 length 8 :
  Layer 3 protocol ID: 2048
CLASS_TYPE                        type 1 length 8 :
  Class-Type value: 1
SESSION_ATTRIBUTE                 type 7 length 36:
  Setup Prio: 7, Holding Prio: 7
  Flags: (0x4) SE Style
  Session Name: PRE-STD DS-TE GLOBAL POOL
[snip]

```

To test out how bandwidth is divided when there are multiple priorities, next we configure a new tunnel [ID 201] that uses global-pool bandwidth but is the highest priority. The tunnel will take any green path to XRv11 and reserve 7 Mbps of global-pool bandwidth. This maps to TE-class 4, which is a combination of CT0 and priority 0. Auto-route is not configured since this tunnel is for demonstration only.

```

! CSR1
interface Tunnel201
  description RDM DS-TE GLOBAL POOL (HI PRI)
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 11.11.11.11
  tunnel mpls traffic-eng priority 0 0
  tunnel mpls traffic-eng bandwidth 7000 class-type 0
  tunnel mpls traffic-eng affinity 0x2 mask 0x2
  tunnel mpls traffic-eng path-option 10 dynamic

```

It is interesting to note that a lower-priority CT cannot reduce bandwidth from a higher-priority CT. Even though TE-class 4 is "better" than TE-class 1, it uses bandwidth from BC0 which can never take bandwidth from BC1. TE-class 0 is affected by this 7 Mbps reservation since it is within CT0, just like TE-class 4. TE-class 1 is within CT1 and cannot be affected. In short, when a flow has a lower CT but a higher priority (TE-class 4), higher CTs with lower priorities (TE-class 1) are not affected.

```

CSR1#show mpls traffic-eng topology 1.1.1.1
[snip]
  link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0003.00, nbr_node_id:29,
gen:2799
  frag_id: 0, Intf Address: 132.1.3.1, Nbr Intf Address: 132.1.3.3
  TE metric: 10, IGP metric: 10, attribute flags: 0x2

```

```

SRLGs: None
physical_bw: 1000000 (kbps),

BC Model Id: RDM
BC0 (max_reservable): 100000 (kbps)
BC0 (max_reservable_bw_global): 100000 (kbps)
BC1 (max_reservable_bw_sub): 20000 (kbps)

```

	Total Allocated BW (kbps)	Reservable BW (kbps)
	-----	-----
TE-Class[0]:	7000	93000
TE-Class[1]:	0	20000
TE-Class[2]:	0	0
TE-Class[3]:	0	0
TE-Class[4]:	7000	93000
TE-Class[5]:	0	20000
TE-Class[6]:	0	0
TE-Class[7]:	0	0

[snip]

Building another tunnel to XRv11 [ID 202], we will configure a 1 Mbps CT1 reservation at priority 3. Priority 3 was not specified as a valid TE-class by default. Unfortunately, there are no debug or syslog messages to indicate this is a problem, and the tunnel never comes up. I assume that the TE process has no idea how to signal this bandwidth since it does not fit into a predefined TE-class.

```

! CSR1
interface Tunnel202
description RDM DS-TE SUB POOL (MED PRI)
ip unnumbered Loopback0
tunnel mode mpls traffic-eng
tunnel destination 11.11.11.11
tunnel mpls traffic-eng priority 3 3
tunnel mpls traffic-eng bandwidth 1000 class-type 1
tunnel mpls traffic-eng affinity 0x8 mask 0x8
tunnel mpls traffic-eng path-option 10 dynamic

```

```

CSR1#show mpls traffic-eng tunnels tunnel 202 | section Status|Config
Status:
Admin: up          Oper: down    Path: not valid   Signalling: Down
path option 10, type dynamic

```

```

Config Parameters:
Bandwidth: 1000    kbps (CT1)  Priority: 3  3    Affinity: 0x8/0x8
Metric Type: TE (default)
AutoRoute: disabled LockDown: disabled Loadshare: 1000 [0] bw-based
auto-bw: disabled

```

We can customize the TE-classes as well; the TE-classes seen earlier were just the defaults. We create a new TE-class 3 that uses CT1 and priority 3. This must be consistent on all MPLS routers within the DS-TE domain. After configuring this new class, we verify it on an XE and an XR router. XE shows non-default entries by omitting the asterisk while XR explicitly states whether a TE-class is a default or configured entry.

```
! All MPLS XE routers
mpls traffic-eng ds-te te-classes
  te-class 3 class-type 1 priority 3
```

```
! All MPLS XR routers
mpls traffic-eng
  ds-te te-classes
    te-class 3 class-type 1 priority 3
```

```
CSR1#show mpls traffic-eng ds-te te-class
DS-TE Mode: IETF
```

TE-Class	Class-Type	Priority
* 0	0	7
* 1	1	7
3	1	3
* 4	0	0
* 5	1	0
* - default setting		
Class-Type: 0 = Global-pool, 1 = Sub-pool		

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng ds-te te-class
te-class 0: class-type 0 priority 7 status default
te-class 1: class-type 1 priority 7 status default
te-class 2: unused
te-class 3: class-type 1 priority 3 status configured
te-class 4: class-type 0 priority 0 status default
te-class 5: class-type 1 priority 0 status default
te-class 6: unused
te-class 7: unused
```

With the new TE-class 3 configured, tunnel202 comes up and we can check its bandwidth impacts on the TE topology. The tunnel routes through CSR4 which is the only orange link towards XRv11. Because the reservation was made at TE-class 3, all lower TE-classes that have lesser or equal CTs will be affected (they are within BC1 or less). In this case, the reservation affects all lower priority TE classes for BC0 and BC1. TE-classes 4 and 5 are not affected since they are higher priority; the "cascade" effect of bandwidth reduction only applies when both the CT and priority are less or equal to the level at which the reservation was made. TE-classes 4 and 5 are unaffected, despite TE-class 4 being BC0.

```
CSR1#show mpls traffic-eng topology 1.1.1.1
```

```
[snip]
  link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0004.00, nbr_node_id:18,
gen:2822
  frag_id: 0, Intf Address: 132.1.4.1, Nbr Intf Address: 132.1.4.4
  TE metric: 10, IGP metric: 10, attribute flags: 0xA
  SRLGs: None
  physical_bw: 1000000 (kbps),

  BC Model Id: RDM
  BC0 (max_reservable): 100000 (kbps)
  BC0 (max_reservable_bw_global): 100000 (kbps)
  BC1 (max_reservable_bw_sub): 20000 (kbps)

      Total Allocated    Reservable
      BW (kbps)          BW (kbps)
      -----
TE-Class[0]:           1000          99000
TE-Class[1]:           1000          19000
TE-Class[2]:              0              0
TE-Class[3]:           1000          19000
TE-Class[4]:              0          100000
TE-Class[5]:              0           20000
TE-Class[6]:              0              0
TE-Class[7]:              0              0
[snip]
```

RDM tunnels on XR are very similar to XE. Recycling tunnel200, we just change the syntax to the RDM version (which isn't required) for clarity. We can verify the reservation by checking the RSVP details, but again, this doesn't show us BC-level granularity.

```
! XRv11
interface tunnel-te200
  signalled-bandwidth 11000 class-type 0

RP/0/0/CPU0:XRv11#show rsvp interface gig0/0/0/0.551
*: RDM: Default I/F B/W % : 75% [default] (max resv/bc0), 0% [default] (bc1)

INTERFACE: GigabitEthernet0/0/0/0.551 (ifh=0xB00).
  BW (bits/sec): Max=100M. MaxFlow=100M.
                 Allocated=11M (11%).
                 BC0=100M. BC1=20M.
```

The TE topology will show this reservation using almost identical output to XE. The output is shown below for reference; we clearly see 11 Mbps reserved within TE-class 0. This represents BC0 at the lowest priority, which is correct.

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng ds-te te-class
te-class 0: class-type 0 priority 7 status default
```

```

te-class 1: class-type 1 priority 7 status default
te-class 2: unused
te-class 3: class-type 1 priority 3 status configured
te-class 4: class-type 0 priority 0 status default
te-class 5: class-type 1 priority 0 status default
te-class 6: unused
te-class 7: unused

```

```

RP/0/0/CPU0:XRv11#show mpls traffic-eng topology 11.11.11.11 | begin 0005
  Link[0]:Point-to-Point, Nbr IGP Id:0000.0000.0005.00, Nbr Node Id:6,
  gen:7234

```

```

  Frag Id:0, Intf Address:132.5.11.11, Intf Id:0
  Nbr Intf Address:132.5.11.5, Nbr Intf Id:0
  TE Metric:10, IGP Metric:10
  Attribute Flags: 0x5
  Ext Admin Group:
    Length: 256 bits
    Value : 0x::5
  Attribute Names:
  Unnamed bits   : 0 2
  Switching Capability:None, Encoding:unassigned

```

```

  BC Model ID:RDM

```

```

  Physical BW:1000000 (kbps), Max Reservable BW:100000 (kbps)

```

```

  BC0:100000 (kbps) BC1:20000 (kbps)

```

	Total Allocated BW (kbps)	Reservable BW (kbps)
	-----	-----
TE-class[0]:	11000	89000
TE-class[1]:	0	20000
TE-class[2]:	0	0
TE-class[3]:	0	20000
TE-class[4]:	0	100000
TE-class[5]:	0	20000
TE-class[6]:	0	0
TE-class[7]:	0	0

To test FRR with RDM, we will use tunnels 210 and 211 on CSR1 and XRv11 again. We will update them to the IETF RDM syntax for clarity. All previous tunnels are shutdown for this test.

```

! CSR1
interface Tunnel210
  description DS-TE RDM BC0 BW FRR
  tunnel mpls traffic-eng bandwidth 9000 class-type 0

interface Tunnel211
  description DS-TE RDM BC1 BW FRR
  tunnel mpls traffic-eng bandwidth 3000 class-type 1

! XRv11

```

```

interface tunnel-te210
  description DS-TE RDM BC0 BW FRR
  signalled-bandwidth 11000 class-type 0

interface tunnel-te211
  description DS-TE RDM BC1 BW FRR
  signalled-bandwidth 4000 class-type 1

```

Since pre-standard DS-TE operates just like RDM with only two pools, the FRR behavior is identical. We update the existing backup tunnels on CSR3 to the IETF RDM syntax for clarity.

```

! CSR3
interface Tunnel30
  tunnel mpls traffic-eng backup-bw class-type 0 unlimited

interface Tunnel31
  tunnel mpls traffic-eng backup-bw class-type 1 5000

interface Tunnel32
  tunnel mpls traffic-eng backup-bw 25000

```

The only changes we have made thus far have been cosmetic. There isn't any behavior change with DS-TE and FRR when switching between different modes or BC-models. Verifying each individual tunnel again would be highly redundant, so we will check the FRR database on CSR3 as a shortcut. Again, we see all 4 tunnels receiving some level of protection. The behavior is identical to pre-standard RDM with respect to FRR protection, except the configuration syntax changes. The backup tunnels that each LSP selects are based on the same 3-point selection process we evaluated earlier. The chart shown in the previous chapter holds true for IETF DS-TE using RDM.

```

CSR3#show mpls traffic-eng fast-reroute database | begin midpoint
P2P LSP midpoint frr information:
LSP identifier          In-label Out intf/label    FRR intf/label    Status
-----
1.1.1.1 210 [18]        3005     Gi2.530:10008        Tu30:10008        ready
1.1.1.1 211 [16]        3007     Gi2.530:10024        Tu31:implicit-nu  ready
11.11.11.11 210 [4]     3023     Gi2.530:10015        Tu32:6018         ready
11.11.11.11 211 [4]     3001     Gi2.530:10014        Tu32:6019         ready

```

As a quick test, we shut down tunnel32 again to ensure we get the same result, which also supports the claim that the selection process is consistent between pre-standard and IETF DS-TE implementation.

```

CSR3#show mpls traffic-eng fast-reroute database | begin midpoint
P2P LSP midpoint frr information:
LSP identifier          In-label Out intf/label    FRR intf/label    Status
-----
1.1.1.1 210 [18]        3005     Gi2.530:10008        Tu30:10008        ready

```

```

1.1.1.1 211 [16]          3007      Gi2.530:10024      Tu31:implicit-nu ready
11.11.11.11 210 [4]      3023      Gi2.530:10015      Tu30:10015        ready

```

We quickly verify the bandwidth reserved on the link to CSR10 for additional verification. We can see the RDM BC model in action; the 7 Mbps reduced from TE-class 1 is also reduced from TE-class 0 based on the bandwidth sharing mechanism of RDM. With MAM, we will see that TE-class 0 would not have been affected by TE-class 1 reservations since CTs are entirely isolated.

```

CSR3#show mpls traffic-eng topology 3.3.3.3 | begin 0010
  link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0010.00, nbr_node_id:18,
gen:563
  frag_id: 0, Intf Address: 132.3.10.3, Nbr Intf Address: 132.3.10.10
  TE metric: 10, IGP metric: 10, attribute flags: 0x0
  SRLGs: None
  physical_bw: 1000000 (kbps),

  BC Model Id: RDM
  BC0 (max_reservable): 100000 (kbps)
  BC0 (max_reservable_bw_global): 100000 (kbps)
  BC1 (max_reservable_bw_sub): 20000 (kbps)

```

	Total Allocated BW (kbps)	Reservable BW (kbps)
	-----	-----
TE-Class[0]:	27000	73000
TE-Class[1]:	7000	13000
TE-Class[2]:	0	0
TE-Class[3]:	0	20000
TE-Class[4]:	0	100000
TE-Class[5]:	0	20000
TE-Class[6]:	0	0
TE-Class[7]:	0	0

Additional Reading – Reference configurations “ds-te-rdm”

31.4.3 IETF Maximum Allocation Model (MAM)

This lab continues from the RDM lab. Maximum Allocation Model (MAM) is defined in RFC 4125 and creates total isolation between the CTs. There is no bandwidth sharing between them which means bandwidth can be wasted when higher priority flows are not using all available bandwidth within a BC. Different priorities across different CTs have no effect on one another. This is essentially a way to map BCn to CTn, where $0 \leq n \leq 7$. Each CT will always receive its bandwidth share. This model is effective when the premium traffic (voice, etc) needs better bandwidth protection and total isolation. Like RDM, this is an IETF model so the logic of CT, BC, and TE-class is the same. When switching from RDM to MAM, XE does not conveniently replace all of the link-level RSVP commands with the MAM commands, so we must do it manually. This is because the RDM and MAM commands are functionally different and both commands can exist on a link concurrently. For clarity, I remove the RDM commands and add the MAM

commands. Although the sum of all BCs can total more than the max-reservable bandwidth, it makes more sense to have them sum to exactly that number. In this case, 80 Mbps are reserved for BC0 and 20 Mbps are reserved for BC1.

```
! All MPLS XE routers
mpls traffic-eng ds-te bc-model mam

! All MPLS XR routers
mpls traffic-eng
  ds-te bc-model mam

! CSR1 (example)
interface GigabitEthernet2.516
  no ip rsvp bandwidth rdm bc0 100000 bc1 20000
  ip rsvp bandwidth mam max-reservable-bw 100000 bc0 80000 bc1 20000

! XRv11 (example)
rsvp
  interface GigabitEthernet0/0/0/0.501
    no bandwidth
    bandwidth mam max-reservable-bw 100000 bc0 80000 bc1 20000
```

Before looking at the TE topology, we check the underlying data structures to see how this changes the ISIS LSP. The format has changed slightly for MAM compared to RDM. The BC model is carried in the BC-TLV which identifies this as MAM, whereas before it showed RDM. The BC values are listed individually as 80 Mbps and 20 Mbps for BC0 and BC1, respectively. The "global pool" bandwidth now represents the max-reservable bandwidth of 100 Mbps, which in our case is the sum of BC0 and BC1. With RDM, BC0 showed 100 Mbps of bandwidth, but MAM completely separates the class-types (and therefore the BCs). Technically, routers could be running different BC-models within a network, but it would be difficult to design a QoS solution if routers interpret BCs differently. We can see that TE-classes 0 and 4, the ones based on BC0 by default, now only claim to have 80 Mbps available for reservations using MAM.

```
CSR1#show isis database verbose CSR1-00.00 | section CSR8
Metric: 10          IS (MT-IPv6) CSR8.00
Metric: 10          IS-Extended CSR8.00
Affinity: 0x00000004
Interface IP Address: 132.1.8.1
Neighbor IP Address: 132.1.8.8
Physical BW: 1000000 kbits/sec
Reservable Global Pool BW: 100000 kbits/sec
TE-Class BW Unreserved:
  [0]: 80000 kbits/sec, [1]: 20000 kbits/sec
  [2]: 0 kbits/sec, [3]: 20000 kbits/sec
  [4]: 80000 kbits/sec, [5]: 20000 kbits/sec
  [6]: 0 kbits/sec, [7]: 0 kbits/sec
```

```

BC-TLV Header
  BC Model-Id:MAM
Bandwidth constraints[0]: 80000 kbits/sec
Bandwidth constraints[1]: 20000 kbits/sec

```

These configuration changes have no affect on existing TE-classes, TE tunnels, or other IETF-related details. TE-class 3 still exists as we configured it for RDM. Changing the BC model just changes how bandwidth is treated with respect to CTs and LSP priorities. First, we will look at tunnel200 again. The tunnel still routes via CSR6, so CSR1 should see a 5 Mbps reservation on this link against TE-class 1. This class represents BC1 (in MAM, this means CT1) at priority 7 (worst). There are 15 Mbps remaining of BC1 bandwidth (sub-pool), but BC0 is unaffected. With RDM, TE-class 0 would have 75 Mbps remaining since the 5 Mbps would have been counted against all lower BCs as well. In this way, MAM keeps bandwidth allocations between CTs totally separate.

```

CSR1#show mpls traffic-eng topology 1.1.1.1
[snip]
link[3]: Point-to-Point, Nbr IGP Id: 0000.0000.0006.00, nbr_node_id:2,
gen:77
frag_id: 0, Intf Address: 132.1.6.1, Nbr Intf Address: 132.1.6.6
TE metric: 5, IGP metric: 10, attribute flags: 0x4
SRLGs: None
physical_bw: 1000000 (kbps),

```

```

BC Model Id: MAM
BC0 (max_reservable): 100000 (kbps)
BC0 (max_reservable_bw_global): 80000 (kbps)
BC1 (max_reservable_bw_sub): 20000 (kbps)

```

	Total Allocated BW (kbps)	Reservable BW (kbps)
	-----	-----
TE-Class[0]:	0	80000
TE-Class[1]:	5000	15000
TE-Class[2]:	0	0
TE-Class[3]:	0	20000
TE-Class[4]:	0	80000
TE-Class[5]:	0	20000
TE-Class[6]:	0	0
TE-Class[7]:	0	0

Next, we check tunnel201 which routes via CSR3 since it is the best green path to XRv11. On the link to CSR3, CSR1 reserves 7 Mbps for TE-class 4, which uses BC0 at priority 0 (highest). Like RDM, this also does not affect BC1, but does affect lower priority BC0-mapped TE-classes, such as TE-class 0. In this way, MAM still keeps the CTs totally separate but honors the importance of prioritization.

```

CSR1#show mpls traffic-eng topology 1.1.1.1

```

```
[snip]
link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0003.00, nbr_node_id:5,
gen:77
frag_id: 0, Intf Address: 132.1.3.1, Nbr Intf Address: 132.1.3.3
TE metric: 10, IGP metric: 10, attribute flags: 0x2
SRLGs: None
physical_bw: 1000000 (kbps),

BC Model Id: MAM
BC0 (max_reservable): 100000 (kbps)
BC0 (max_reservable_bw_global): 80000 (kbps)
BC1 (max_reservable_bw_sub): 20000 (kbps)

          Total Allocated   Reservable
          BW (kbps)         BW (kbps)
          -----
TE-Class[0]:           7000           73000
TE-Class[1]:              0           20000
TE-Class[2]:              0              0
TE-Class[3]:              0           20000
TE-Class[4]:           7000           73000
TE-Class[5]:              0           20000
TE-Class[6]:              0              0
TE-Class[7]:              0              0
```

Last, we analyze tunnel202's behavior with MAM. It routes via CSR4 since that is the only orange path towards XRv11. TE-class 3 uses BC1 (only maps to CT1 with MAM) and has priority 3. It reserves 1 Mbps of BC1 (sub-pool) bandwidth. Lower priority BC1 classes, such as TE-class 1, have their available bandwidth reduced by 1 Mbps as well. TE-class 0, despite being the "lowest priority" traffic, is totally unaffected by the BC1 reservations as it maps to BC0.

```
CSR1#show mpls traffic-eng topology 1.1.1.1
[snip]
link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0004.00, nbr_node_id:6,
gen:77
frag_id: 0, Intf Address: 132.1.4.1, Nbr Intf Address: 132.1.4.4
TE metric: 10, IGP metric: 10, attribute flags: 0xA
SRLGs: None
physical_bw: 1000000 (kbps),

BC Model Id: MAM
BC0 (max_reservable): 100000 (kbps)
BC0 (max_reservable_bw_global): 80000 (kbps)
BC1 (max_reservable_bw_sub): 20000 (kbps)
```

```
          Total Allocated   Reservable
          BW (kbps)         BW (kbps)
          -----
```

TE-Class[0]:	0	80000
TE-Class[1]:	1000	19000
TE-Class[2]:	0	0
TE-Class[3]:	1000	19000
TE-Class[4]:	0	80000
TE-Class[5]:	0	20000
TE-Class[6]:	0	0
TE-Class[7]:	0	0

MAM DS-TE on XR is very similar to RDM DS-TE in terms of tunnel configuration. In this case, we have a BC1 reservation of 11 Mbps, which should not affect any lower-priority BC0 TE-classes. We quickly verify the tunnel is up before continuing.

```
! XRv11
interface tunnel-te200
  signalled-bandwidth 11000 class-type 1

RP/0/0/CPU0:XRv11#show mpls traffic-eng tunnels 200 role head | begin Path
info
  Path info (IS-IS 132 level-2):
  Node hop count: 6
  Hop0: 132.5.11.5
  Hop1: 132.2.5.2
  Hop2: 132.2.3.2
  Hop3: 132.2.3.3
  Hop4: 132.3.10.10
  Hop5: 132.6.10.6
  Hop6: 132.6.8.8
  Hop7: 8.8.8.8
```

Since CSR5 is the next-hop, we can look at the TE topology to confirm the MAM behavior on XR. Despite 11 Mbps being reserved for TE-class 1, this does not affect TE-class 0 at all as 80 Mbps are still available. This TE-class 0 value would have been reduced to 69 Mbps with RDM.

```
RP/0/0/CPU0:XRv11#show mpls traffic-eng topology 11.11.11.11 | begin 0005
  Link[0]:Point-to-Point, Nbr IGP Id:0000.0000.0005.00, Nbr Node Id:6,
  gen:11341
    Frag Id:0, Intf Address:132.5.11.11, Intf Id:0
    Nbr Intf Address:132.5.11.5, Nbr Intf Id:0
    TE Metric:10, IGP Metric:10
    Attribute Flags: 0x5
    Ext Admin Group:
      Length: 256 bits
      Value : 0x::5
    Attribute Names:
    Unnamed bits : 0 2
    Switching Capability:None, Encoding:unassigned
    BC Model ID:MAM
```

Physical BW:1000000 (kbps), Max Reservable BW:100000 (kbps)
 BC0:80000 (kbps) BC1:20000 (kbps)

	Total Allocated BW (kbps)	Reservable BW (kbps)
	-----	-----
TE-class[0]:	0	80000
TE-class[1]:	11000	9000
TE-class[2]:	0	0
TE-class[3]:	0	20000
TE-class[4]:	0	80000
TE-class[5]:	0	20000
TE-class[6]:	0	0
TE-class[7]:	0	0

We quickly test FRR with MAM also. The tunnel configurations are identical as they were for RDM, minus descriptions, but are shown again for clarity. All existing DS-TE tunnels are shut down for the FRR testing.

```
! CSR1
interface Tunnel210
  description DS-TE MAM BC0 BW FRR
  tunnel mpls traffic-eng bandwidth 9000 class-type 0

interface Tunnel211
  description DS-TE MAM BC1 BW FRR
  tunnel mpls traffic-eng bandwidth 3000 class-type 1

! XRv11
interface tunnel-te210
  description DS-TE MAM BC0 BW FRR
  signalled-bandwidth 11000 class-type 0

interface tunnel-te211
  description DS-TE MAM BC1 BW FRR
  signalled-bandwidth 4000 class-type 1
```

The NHOP and NNHOP tunnels on CSR3 are totally unchanged, but shown again for clarity.

```
! CSR3
interface Tunnel30
  tunnel mpls traffic-eng backup-bw class-type 0 unlimited

interface Tunnel31
  tunnel mpls traffic-eng backup-bw class-type 1 5000

interface Tunnel32
  tunnel mpls traffic-eng backup-bw 25000
```

We can verify that all of the tunnels are up and protected by FRR by checking the PLR. This output is identical (except for labels) to the outputs seen for the pre-standard and IETF RDM versions. The selection process of NHOP vs. NNHOP, specified vs. unspecified pools, and limited vs. unlimited bandwidth is identical.

```
CSR3#show mpls traffic-eng fast-reroute database | begin midpoint
P2P LSP midpoint frr information:
LSP identifier          In-label Out intf/label    FRR intf/label    Status
-----
1.1.1.1 210 [2]         3024     Gi2.530:10016      Tu30:10016        ready
1.1.1.1 211 [2]         3014     Gi2.530:10024      Tu31:implicit-nu  ready
11.11.11.11 210 [3]     3017     Gi2.530:10001      Tu32:6017         ready
11.11.11.11 211 [3]     3015     Gi2.530:10020      Tu32:6018         ready
```

To verify MAM behavior once more, we check the TE topology towards CSR10. A total of 7 Mbps (3 from CSR1 and 4 from XRv11) are reserved for TE-class 1, which includes BC1. A total of 20 Mbps (9 from CSR1 and 11 from XRv11) are reserved for TE-class 0, which includes BC0. There are 60 Mbps remaining for TE-class 0 with this model since the 7 Mbps against TE-class 1 do not count against TE-class 0, as they would for RDM. RDM would have 73 Mbps remaining for TE-class 0 if that model was in use; this is because all 100 Mbps are available to BC0, so reducing all "better or equal" flows counts as 27 Mbps (seen earlier).

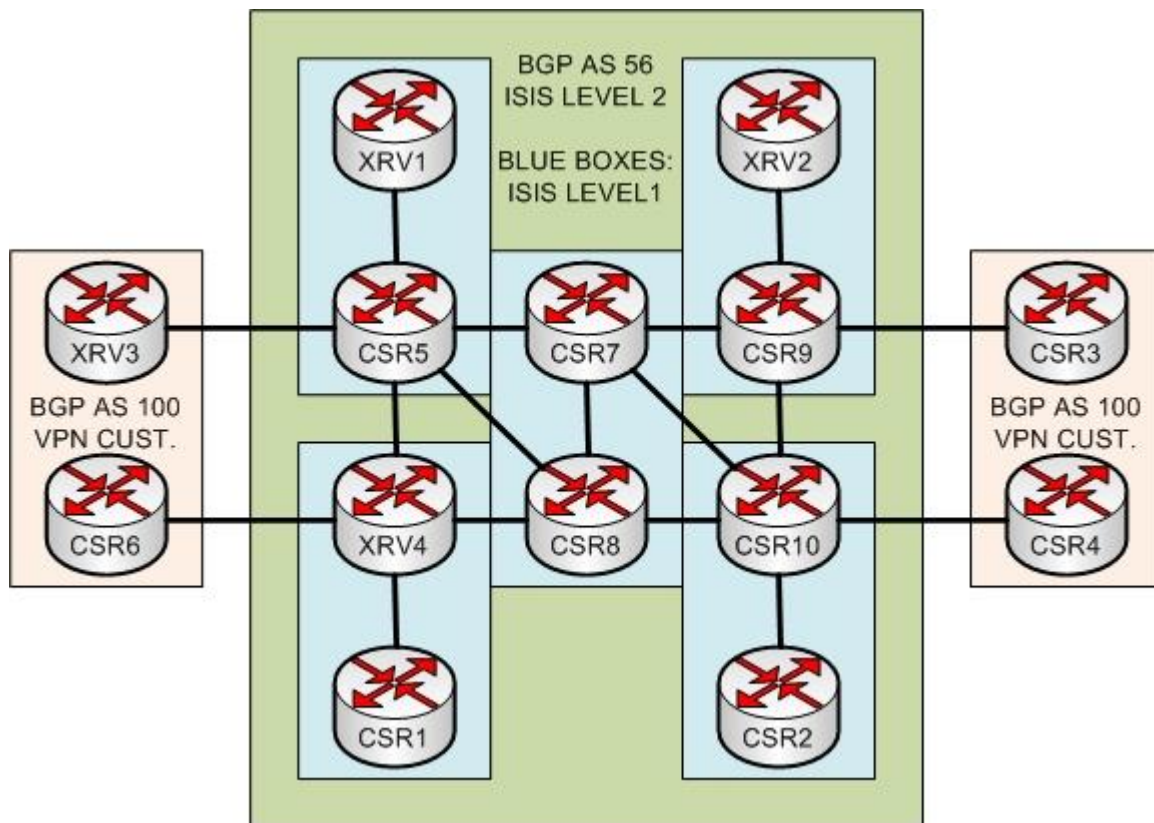
```
CSR3#show mpls traffic-eng topology 3.3.3.3 | begin 0010
link[3]: Point-to-Point, Nbr IGP Id: 0000.0000.0010.00, nbr_node_id:18,
gen:518
frag_id: 0, Intf Address: 132.3.10.3, Nbr Intf Address: 132.3.10.10
TE metric: 10, IGP metric: 10, attribute flags: 0x0
SRLGs: None
physical_bw: 1000000 (kbps),

BC Model Id: MAM
BC0 (max_reservable): 100000 (kbps)
BC0 (max_reservable_bw_global): 80000 (kbps)
BC1 (max_reservable_bw_sub): 20000 (kbps)
```

	Total Allocated BW (kbps)	Reservable BW (kbps)
	-----	-----
TE-Class[0]:	20000	60000
TE-Class[1]:	7000	13000
TE-Class[2]:	0	0
TE-Class[3]:	0	20000
TE-Class[4]:	0	80000
TE-Class[5]:	0	20000
TE-Class[6]:	0	0
TE-Class[7]:	0	0

31.4.4 Per-VRF TE techniques

For additional granularity when using MPLS-TE with L3VPN, both XE and XR support TE tunnels on a per-VRF basis. This is particularly useful on a PE that has several L3VPN customers, each with different requirements. It is compatible with all of the previously examined techniques, such as CBTS/PBTS and DS-TE. The network is similar to the BGP RR-group and RT retention lab except that XRv4 has been swapped with CSR6. This introduces an XR router into the core network for testing per-VRF TE. The basic IS-IS configuration is not detailed here.



First, we quickly verify that XRv4 has IS-IS connectivity to all nodes since the configuration is omitted for brevity. The core LSRs are level-2 peers while CSR1, a route-reflector, is a level-1 peer.

```
RP/0/0/CPU0:XRv4#show isis neighbors
```

```
IS-IS 56 neighbors:
```

System Id	Interface	SNPA	State	Holdtime	Type	IETF-NSF
R1	Gi0/0/0/0.514	*PtoP*	Up	24	L1	Capable
R8	Gi0/0/0/0.584	*PtoP*	Up	28	L2	Capable
R5	Gi0/0/0/0.554	*PtoP*	Up	25	L2	Capable

This lab uses primary one-hop TE tunnels between each pair of routers in the core rather than LDP directly on the core interfaces. As a result, we quickly check the TED for the proper TE relationships.

XRv4 sees CSR5 and CSR8 as peer vertices in the TE graph, and vice versa. This is critical since the use of primary tunnels relies on MPLS-TE being functional in the core to signal the primary tunnels.

```
RP/0/0/CPU0:XRv4#show mpls traffic-eng topology brief 56.0.0.14 | include IGP Id
IGP Id: 0000.0000.0014.00, MPLS TE Id: 56.0.0.14 Router Node (IS-IS 56 level-2)
  Link[0]:Point-to-Point, Nbr IGP Id:0000.0000.0008.00, Nbr Node Id:2, gen:366096
  Link[1]:Point-to-Point, Nbr IGP Id:0000.0000.0005.00, Nbr Node Id:3, gen:366097
```

```
RP/0/0/CPU0:XRv4#show mpls traffic-eng topology brief 56.0.0.5 | include IGP Id
IGP Id: 0000.0000.0005.00, MPLS TE Id: 56.0.0.5 Router Node (IS-IS 56 level-2)
  Link[0]:Point-to-Point, Nbr IGP Id:0000.0000.0007.00, Nbr Node Id:5, gen:366102
  Link[1]:Point-to-Point, Nbr IGP Id:0000.0000.0008.00, Nbr Node Id:2, gen:366103
  Link[2]:Point-to-Point, Nbr IGP Id:0000.0000.0014.00, Nbr Node Id:1, gen:366104
```

```
RP/0/0/CPU0:XRv4#show mpls traffic-eng topology brief 56.0.0.8 | include IGP Id
IGP Id: 0000.0000.0008.00, MPLS TE Id: 56.0.0.8 Router Node (IS-IS 56 level-2)
  Link[0]:Point-to-Point, Nbr IGP Id:0000.0000.0007.00, Nbr Node Id:5, gen:366098
  Link[1]:Point-to-Point, Nbr IGP Id:0000.0000.0010.00, Nbr Node Id:7, gen:366099
  Link[2]:Point-to-Point, Nbr IGP Id:0000.0000.0014.00, Nbr Node Id:1, gen:366100
  Link[3]:Point-to-Point, Nbr IGP Id:0000.0000.0005.00, Nbr Node Id:3, gen:366101
```

LDP is enabled on the primary tunnels since many of them are PE-P or P-P tunnels. Since XR's auto-primary tunnel feature is not as powerful as XE's (difficult to get mLDp traffic into them, etc), we will create 2 manual primary tunnels on XRv4: one to CSR5 and one to CSR8. The debugs in the auto-primary testing conducted earlier revealed that these tunnels consist of next-hop explicit-paths, autoroute announce, and FRR. These tunnels are unrelated to the per-VRF TE technique but is important to achieve basic end-to-end LSP reachability.

```
! XRv4
explicit-path name EP_5
  index 10 next-address strict ipv4 unicast 56.5.14.5
```

```
explicit-path name EP_8
  index 10 next-address strict ipv4 unicast 56.8.14.8
```

```
interface tunnel-te505
  description 1HOP TO CSR5
  ipv4 unnumbered Loopback0
  autoroute announce
  destination 56.0.0.5
  fast-reroute
  path-option 10 explicit name EP_5
```

```
interface tunnel-te508
  description 1HOP TO CSR8
  ipv4 unnumbered Loopback0
  autoroute announce
  destination 56.0.0.8
  fast-reroute
```



```
path-option 10 explicit name EP_8
```

```
mpls ldp
interface tunnel-te505
interface tunnel-te508
```

We quickly verify that these primary tunnels come up. Since LDP is enabled on them, we expect to see tLDP sessions towards CSR5 and CSR8.

```
RP/0/0/CPU0:XRv4#show mpls traffic-eng tunnels brief role head
```

TUNNEL NAME	DESTINATION	STATUS	STATE
tunnel-te505	56.0.0.5	up	up
tunnel-te508	56.0.0.8	up	up

Displayed 2 (of 2) heads, 0 (of 1) midpoints, 0 (of 4) tails
Displayed 2 up, 0 down, 0 recovering, 0 recovered heads

```
RP/0/0/CPU0:XRv4#show mpls ldp neighbor brief
```

Peer	GR	NSR	Up Time	Discovery	Addresses	Labels			
				ipv4	ipv6	ipv4	ipv6	ipv4	ipv6
56.0.0.8:0	N	N	1w6d	1	0	5	0	16	0
56.0.0.5:0	N	N	1w6d	1	0	8	0	16	0

XRv4 must also be configured to accept tLDP sessions since CSR5 and CSR8 will build auto-primary tunnels to XRv4. This is nothing new; we quickly allow XRv4 to accept these sessions then verify that it is the tail end for a pair of primary tunnels from CSR5 and CSR8.

```
! XRv4
mpls ldp
address-family ipv4
discovery targeted-hello accept
```

```
RP/0/0/CPU0:XRv4#show mpls traffic-eng tunnels brief role tail
```

TUNNEL NAME	DESTINATION	STATUS	STATE
R5_t500	56.0.0.14	up	up
R8_t500	56.0.0.14	up	up

Displayed 0 (of 2) heads, 0 (of 0) midpoints, 2 (of 2) tails
Displayed 0 up, 0 down, 0 recovering, 0 recovered heads

XRv4 has the same logical VPNv4/v6 BGP configuration as CSR6 did (the routers swapped places), so it is not discussed in detail. We quickly verify that all VPNV4/v6 sessions to all RRs are functional. 12 routes are received from each RR, which makes sense since each of the 3 remote PEs contributes 4 routes each.

```
RP/0/0/CPU0:XRv4#show bgp vpnv4 unicast summary | begin ^Neigh
```

Neighbor	Spk	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	St/PfxRcd
56.0.0.1	0	56	121873	115809	223	0	0	1w6d	12
56.0.0.2	0	56	121800	115743	223	0	0	1w6d	12

56.0.0.11	0	56	115796	115786	223	0	0	1w6d	12
56.0.0.12	0	56	115791	115785	223	0	0	1w6d	12

```
RP/0/0/CPU0:XRv4#show bgp vpnv6 unicast summary | begin ^Neigh
```

Neighbor	Spk	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	St/PfxRcd
56.0.0.1	0	56	121874	115809	195	0	0	1w6d	12
56.0.0.2	0	56	121800	115743	195	0	0	1w6d	12
56.0.0.11	0	56	115796	115786	195	0	0	1w6d	12
56.0.0.12	0	56	115791	115785	195	0	0	1w6d	12

We also verify that XRv4 is connected to the L3VPN customer which is now CSR6. We can see both the IPv4/v6 peerings are operational in both RED VPN, with RIPv2 and OSPFv3 functioning properly in the BLUE VPN.

```
RP/0/0/CPU0:XRv4#show bgp vrf RED ipv4 unicast summary | begin ^Neigh
```

Neighbor	Spk	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	St/PfxRcd
10.6.14.6	0	100	21522	19361	231	0	0	00:06:26	4

```
RP/0/0/CPU0:XRv4#show bgp vrf RED ipv6 unicast summary | begin ^Neigh
```

Neighbor	Spk	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	St/PfxRcd
fd00:10:6:14::6	0	100	21513	19352	195	0	0	1w6d	4

```
RP/0/0/CPU0:XRv4#show route vrf BLUE rip
```

R	200.6.0.0/18	[120/1]	via 10.6.14.6,	1w6d,	GigabitEthernet0/0/0/0.200
R	200.6.64.0/18	[120/1]	via 10.6.14.6,	1w6d,	GigabitEthernet0/0/0/0.200
R	200.6.128.0/18	[120/1]	via 10.6.14.6,	1w6d,	GigabitEthernet0/0/0/0.200
R	200.6.192.0/18	[120/1]	via 10.6.14.6,	1w6d,	GigabitEthernet0/0/0/0.200

```
RP/0/0/CPU0:XRv4#show ospfv3 vrf BLUE neighbor
```

Neighbors for OSPFv3 56, VRF BLUE

Neighbor ID	Pri	State	Dead Time	Interface ID	Interface
200.6.0.1	1	FULL/	00:00:33	38	GigabitEthernet0/0/0/0.200

Neighbor is up for 00:00:12

Since the BLUE VPN is using mLDP, which does not appear to work on XRv, we will verify the RED VPN PIM neighbors across the default MDT. Although MVPN is not terribly relevant for per-VRF TE, we need to ensure we don't "break" it with our tunnel configuration. Testing MVPN is a major component of this section since TE can have a significant impact on multicast routing behavior.

```
RP/0/0/CPU0:XRv4#show pim vrf RED neighbor | begin ^Neigh
```

Neighbor Address	Interface	Uptime	Expires	DR pri	Flags
10.6.14.14*	GigabitEthernet0/0/0/0.100	1w6d	00:01:44	1	(DR) B P E
56.0.0.9	mdtRED	1w5d	00:01:39	1	P
56.0.0.10	mdtRED	1w5d	00:01:37	1	P
56.0.0.15	mdtRED	1w5d	00:01:27	1	P
56.0.0.14*	mdtRED	1w5d	00:01:20	1	(DR) P

To test the data-plane quickly, we use MPLS OAM (LSP ping) from XRv4 to a few PEs. We follow-up with some VRF-aware pings inside CSR6 to test the L3VPN service. This concludes our basic verification; the VPN-aware unicast and multicast testing will be conducted as TE tunnels are built.

```
RP/0/0/CPU0:XRv4#ping mpls ipv4 56.0.0.9/32 source 56.0.0.14
[snip]
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/4/10 ms
```

```
RP/0/0/CPU0:XRv4#ping mpls ipv4 56.0.0.10/32 source 56.0.0.14
[snip]
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/4/10 ms
```

```
R6#ping vrf RED 100.3.64.1 source 100.6.64.1
[snip]
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 10/17/30 ms
```

```
R6#ping vrf BLUE 2200:A213::1 source 2200:A206::1
[snip]
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/6/15 ms
```

The first per-VRF TE technique is the most straightforward and is configured on CSR5. Under each VRF, we can specify a custom BGP next-hop; this references a loopback interface. For maximum scalability, all VRFs on a single PE use the BGP VPNv4/v6 “update-source” as the LSP endpoint by default. This reduces the size of the RIB, LIB, LFIB, and many other internal data structures. However, this “LSP aggregation” also restricts granularity when different VRFs have different traffic forwarding requirements. Specifying different BGP next-hops per VRF (or per group of VRFs) increases the number of routes in the network (along with their corresponding label bindings), but permits greater granularity. Notice that the loopbacks are PIM-enabled and tagged with a route-tag of 56. PIM is required to support MVPN in this specific design (discussed later) and the route-tag allows these loopbacks to leak from level-2 to level-1, satisfying the XR route-reflector’s best-path requirements. **Always define the loopback interfaces and advertise them into IGP before changing the BGP next-hop under the VRF**; this will prevent traffic blackholes in case PEs learn per-VRF loopbacks before they have corresponding IGP routes to them.

```
! CSR5
interface Loopback100
  description RED TE
  ip address 56.0.0.105 255.255.255.255
  ip pim passive
  isis tag 56

interface Loopback200
  description BLUE TE
```

```

ip address 56.0.0.205 255.255.255.255
ip pim passive
isis tag 56

router isis 56
passive-interface Loopback100
passive-interface Loopback200

vrf definition BLUE
address-family ipv4
  bgp next-hop Loopback200
address-family ipv6
  bgp next-hop ipv4 Loopback200

vrf definition RED
address-family ipv4
  bgp next-hop Loopback100
address-family ipv6
  bgp next-hop ipv4 Loopback100

```

Notice that none of the configuration above explicitly references MPLS-TE. We are simply changing the BGP next-hop which gives the LSP a different endpoint. These changes occur on the tail-end router; since LSPs are unidirectional, we are changing the routing towards a VPN endpoint, not from it. At this point, we will perform several control-plane verifications. First and foremost, we ensure that the loopbacks were properly advertised into IGP and have valid LDP label bindings. Since every core router will also learn these prefixes via IS-IS, they should allocate labels for them also.

```

R5#show isis database R5.00-00 detail level-2 | include [12]05
Metric: 0          IP 56.0.0.105/32
Metric: 0          IP 56.0.0.205/32

R5#show mpls ldp bindings 56.0.0.105 32
lib entry: 56.0.0.105/32, rev 28
  local binding: label: imp-null
  remote binding: lsr: 56.0.0.14:0, label: 94002
  remote binding: lsr: 56.0.0.7:0, label: 7003
  remote binding: lsr: 56.0.0.8:0, label: 8004

R5#show mpls ldp bindings 56.0.0.205 32
lib entry: 56.0.0.205/32, rev 30
  local binding: label: imp-null
  remote binding: lsr: 56.0.0.8:0, label: 8001
  remote binding: lsr: 56.0.0.14:0, label: 94003
  remote binding: lsr: 56.0.0.7:0, label: 7013

```

Next, we verify that the VRF configuration was correct by checking the VRF details for the RED and BLUE VPNs. This displays the actual IPv4/v6 address used as the BGP next-hop; notice that the IPv6 address is converted to 6VPE next-hop as expected.

```
R5#show vrf detail RED | include ^Address|BGP
Address family ipv4 unicast (Table ID = 0x4):
  BGP next-hop: Loopback100 56.0.0.105
Address family ipv6 unicast (Table ID = 0x1E000001):
  BGP next-hop: Loopback100 (ipv4) ::FFFF:56.0.0.105
Address family ipv4 multicast not active
```

```
R5#show vrf detail BLUE | include ^Address|BGP
Address family ipv4 unicast (Table ID = 0x5):
  BGP next-hop: Loopback200 56.0.0.205
Address family ipv6 unicast (Table ID = 0x1E000002):
  BGP next-hop: Loopback200 (ipv4) ::FFFF:56.0.0.205
Address family ipv4 multicast not active
```

To verify that the VPNv4/v6 next-hops were actually changed, we will verify the route-reflector BGP tables. Since the selective RT retention policies are in effect, not all RRs will have all routes. CSR1 and XRv1 will have RED routes while CSR2 and XRv2 will have BLUE routes. We will check the IPv4 routes on CSR1/CSR2 and the IPv6 routes on XRv1/XRv2. Recall that XR always displays 6VPE next-hops as normal IPv4 addresses without the ::FFFF: prepended.

```
R1#show bgp vpnv4 unicast rd 56:100 neighbors 56.0.0.5 routes | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 56:100
*>i 100.13.0.0/18    56.0.0.105      0      100      0 100 ?
*>i 100.13.64.0/18   56.0.0.105      0      100      0 100 ?
*>i 100.13.128.0/18  56.0.0.105      0      100      0 100 ?
*>i 100.13.192.0/18  56.0.0.105      0      100      0 100 ?
```

```
R2#show bgp vpnv4 unicast rd 56:200 neighbors 56.0.0.5 routes | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 56:200
*>i 200.13.0.0/18    56.0.0.205      1      100      0 ?
*>i 200.13.64.0/18   56.0.0.205      1      100      0 ?
*>i 200.13.128.0/18  56.0.0.205      1      100      0 ?
*>i 200.13.192.0/18  56.0.0.205      1      100      0 ?
```

```
RP/0/0/CPU0:XRv1#show bgp vpnv6 unicast neighbors 56.0.0.5 routes | begin
Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 56:100
*>i2100:a113::/32    56.0.0.105      0      100      0 100 ?
*>i2100:b113::/32    56.0.0.105      0      100      0 100 ?
*>i2100:c113::/32    56.0.0.105      0      100      0 100 ?
```

```
*>i2100:d113::/32      56.0.0.105      0      100      0 100 ?
```

```
RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast neighbors 56.0.0.5 routes | begin
Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 56:200					
*>i2200:a213::1/128	56.0.0.205	1	100	0	?
*>i2200:b213::1/128	56.0.0.205	1	100	0	?
*>i2200:c213::1/128	56.0.0.205	1	100	0	?
*>i2200:d213::1/128	56.0.0.205	1	100	0	?

We will use CSR10 as our tunnel head-end for all testing in this lab. CSR5, CSR9, and XRv4 will be the tunnel tail ends towards which we can test different per-VRF TE techniques. From CSR10's perspective, we can quickly check the different BGP next-hops and the number of routes referencing each next-hop. The number 8 is shown instead of the number 4 because CSR10 receives two copies of each route thanks to having 2 RRs per VPN. We can see that CSR10 has seen the new BGP next-hops. CSR9 and XRv4, at present, are still using their loopback0 addresses for BGP next-hops since we have not modified those yet.

```
R10#show bgp vpnv4 unicast vrf RED nexthops
# Paths    Nexthop Address
Route Distinguisher: 56:100 (default for vrf RED)
      8    56.0.0.9
      4    10.4.10.4 (RED)
      8    56.0.0.14
      8    56.0.0.105
```

```
R10#show bgp vpnv6 unicast vrf BLUE nexthops
# Paths    Nexthop Address
Route Distinguisher: 56:200 (default for vrf BLUE)
      8    ::FFFF:56.0.0.9
      4    FE80::4 (FE80::4) (BLUE)
     10    ::FFFF:56.0.0.14
      8    ::FFFF:56.0.0.205
```

Next, we will configure TE tunnels on CSR10 towards CSR5. We will begin with the BLUE VPN by creating a simple explicit-path. The tunnel destination is always the MPLS TE-ID and is not the new loopback address; the new loopback is used only for routing as we will see later. Notice that absence of autoroute and forwarding-adjacency on the tunnel; these are not appropriate options for per-VRF TE. If we configure "announce" or "forwarding-adjacency", all new loopbacks would be reachable via this TE tunnel which defeats the purpose of per-VRF TE. If we configure "destination", a static route is created towards 56.0.0.5/32, which doesn't capture 56.0.0.205/32 and is therefore ineffective.

```
! CSR10
ip explicit-path name EP_9_7_5 enable
next-address 56.0.0.9
```

```
next-address 56.0.0.7
next-address 56.0.0.5
```

```
interface Tunnel5102
description TO CSR5 BLUE
ip unnumbered Loopback0
tunnel mode mpls traffic-eng
tunnel destination 56.0.0.5
tunnel mpls traffic-eng path-option 10 explicit name EP_9_7_5
```

We verify that the tunnel comes up and use LSP traceroute to verify the path. This proves that the tunnel is functional and it correctly routes through CSR9, CSR7, and CSR5.

```
R10#show mpls traffic-eng tunnels tunnel 5102 | section Status
Status:
  Admin: up      Oper: up      Path: valid      Signalling: connected
  path option 10, type explicit EP_9_7_5 (Basis for Setup, path weight 30)
```

```
R10#traceroute mpls traffic-eng tunnel 5102 source 56.0.0.10
Tracing MPLS TE Label Switched Path on Tunnel5102, timeout is 2 seconds
[snip]
Type escape sequence to abort.
 0 56.9.10.10 MRU 1500 [Labels: 9013 Exp: 0]
L 1 56.9.10.9 MRU 1500 [Labels: 7022 Exp: 0] 9 ms
L 2 56.7.9.7 MRU 1500 [Labels: implicit-null Exp: 0] 3 ms
! 3 56.5.7.5 4 ms
```

We need to steer traffic to 56.0.0.205/32 into the tunnel somehow; the most straightforward method is using a static route. This was discussed in the “traffic steering” part of the TE section. Once configured, we verify the routing using normal IP traceroute. The hops and labels from this output match the LSP traceroute above, so we are confident that traffic to 56.0.0.205/32 is now inside this tunnel.

```
! CSR10
ip route 56.0.0.205 255.255.255.255 Tunnel5102
```

```
R10#traceroute 56.0.0.205 source 56.0.0.10
Type escape sequence to abort.
Tracing the route to 56.0.0.205
VRF info: (vrf in name/id, vrf out name/id)
 1 56.9.10.9 [MPLS: Label 9013 Exp 0] 4 msec 4 msec 5 msec
 2 56.7.9.7 [MPLS: Label 7022 Exp 0] 16 msec 15 msec 14 msec
 3 56.5.7.5 18 msec 10 msec 9 msec
```

We will quickly trace the LSP from CSR10 to CSR5 to verify the full operation using a VPNv4 route. CSR10’s BGP best-path is from CSR5 and uses VPN label 5031. The VPN next-hop is the per-VRF value of 56.0.0.205 and is reachable via a TE tunnel. This is a result of the static route above; the TE tunnel uses label 9013, so CSR9 imposes {9013 5031} on top of the IPv4 packets from CSR4.

```
R10#show bgp vpnv4 unicast vrf BLUE 200.13.0.0 bestpath
BGP routing table entry for 56:200:200.13.0.0/18, version 397
Paths: (2 available, best #2, table BLUE)
  Not advertised to any peer
  Refresh Epoch 3
  Local
    56.0.0.205 (via default) from 56.0.0.2 (56.0.0.2)
      Origin incomplete, metric 1, localpref 100, valid, internal, best
      Extended Community: RT:56:213
      Originator: 56.0.0.5, Cluster list: 56.0.0.2
      mpls labels in/out nolabel/5031
      rx pathid: 0, tx pathid: 0x0
```

```
R10#show ip route 56.0.0.205
Routing entry for 56.0.0.205/32
  Known via "static", distance 1, metric 0 (connected)
  Routing Descriptor Blocks:
    * directly connected, via Tunnel5102
      Route metric is 0, traffic share count is 1
```

```
R10#show mpls traffic-eng tunnels tunnel 5102 | include Label
InLabel   : -
OutLabel  : GigabitEthernet2.590, 9013
```

```
R10#show ip cef vrf BLUE 200.13.0.0 internal | begin output_chain
output chain:
  label 5031
  label implicit-null
  TAG midchain out of Tunnel5102 7F0D872520C8
  label 9013
  TAG adj out of GigabitEthernet2.590, addr 56.9.10.9 7F0D87258428
```

CSR7 and CSR9 are basic P routers; they perform swap and pop operations, respectively. This exposes label 5031 to CSR5. CSR5 receives label 5031 and delivers the traffic to CSR3 inside VRF BLUE.

```
R9#show mpls forwarding-table labels 9013
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id    Switched     interface
9013       7022       56.0.0.10 5102 [1]    \
                                     1608      Gi2.579    56.7.9.7
```

```
R7#show mpls forwarding-table labels 7022
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id    Switched     interface
7022       Pop Label  56.0.0.10 5102 [1]    \
                                     1954      Gi2.557    56.5.7.5
```

```
R5#show mpls forwarding-table labels 5031 detail
```


Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Label	Outgoing interface	Next Hop
5031	No Label	200.13.0.0/18[V]	1314		Gi2.200	10.5.13.13

MAC/Encaps=22/22, MRU=1504, Label Stack{}

005056A9EA54005056A9DC6381000DE1810000C80800

VPN route: BLUE

No output feature configured

We use traceroute on the CE inside the VRF to verify this customized BLUE path. Since the RED VPN has a different BGP net-hop and its traffic is not routed inside tunnel 5102, we can create another path for it (or leave it as the dynamic LDP path).

```
R4#traceroute vrf BLUE 200.13.0.1 source 200.4.0.1
Type escape sequence to abort.
Tracing the route to 200.13.0.1
VRF info: (vrf in name/id, vrf out name/id)
 1 10.4.10.10 5 msec 3 msec 3 msec
 2 56.9.10.9 [MPLS: Labels 9013/5031 Exp 0] 5 msec 6 msec 12 msec
 3 56.7.9.7 [MPLS: Labels 7022/5031 Exp 0] 31 msec 32 msec 31 msec
 4 10.5.13.5 [MPLS: Label 5031 Exp 0] 18 msec 8 msec 15 msec
 5 10.5.13.13 20 msec 13 msec 14 msec
```

Before continuing, we will test multicast between these endpoints to ensure we haven't broken anything. CSR4 will issue an IGMPv3 join towards CSR10 requesting traffic for group 232.200.4.4; XRv3 is the source of this traffic. We verify that CSR10 receives this C(S,G) information and creates the proper C-MRIB state. The RPF neighbor is 56.0.0.205 which is CSR5; this is derived from the VPNv4 next-hop.

```
! CSR4
interface GigabitEthernet2.200
 ip igmp join-group 232.200.4.4 source 200.13.0.1

R10#show ip mroute vrf BLUE 232.200.4.4 200.13.0.1 | begin \
(200.13.0.1, 232.200.4.4), 1w3d/00:02:45, flags: sTI
  Incoming interface: Lspvif0, RPF nbr 56.0.0.205
  Outgoing interface list:
    GigabitEthernet2.200, Forward/Sparse-Dense, 1w3d/00:02:45
```

As expected, CSR10 cannot perform RPF for this endpoint since it has no legitimate multicast route. The static route overrode the IS-IS route and cannot be used for RPF. Interestingly, CSR10 doesn't care. It has a direct PIM neighbor inside the MDT for this peer and can freely exchange PIM messages. CSR5 was smart enough to change its MDT source to 56.0.0.205 for this VRF. The PIM neighbor address and the RPF address identified in the C-MRIB entry are the same, so PIM signaling works fine.

```
R10#show ip pim vrf BLUE neighbor | begin ^Neigh
Neighbor          Interface          Uptime/Expires    Ver   DR
Address                                     Prio/Mode
```

```
56.0.0.205      Lspvif0      1w5d/00:01:40      v2      1 / DR S P G
56.0.0.9        Lspvif0      2w3d/00:01:31      v2      1 / S P G
```

```
R10#show ip rpf 56.0.0.205
failed, no route exists
```

Despite the RPF failure, there are no problems. CSR5 receives a single C(S,G) SSM join from CSR10. CSR10 sees CSR5 as 56.0.0.205 for unicast and multicast connectivity within VRF RED; this consistency simplifies L3VPN operation when using per-VRF TE.

```
R5#show ip mroute vrf BLUE ssm
(200.13.0.1, 232.200.4.4), 00:03:29/00:02:57, flags: sT
Incoming interface: GigabitEthernet2.200, RPF nbr 10.5.13.13
Outgoing interface list:
Lspvif0, Forward/Sparse, 00:03:29/00:02:57
```

Pinging this group from XRv3 (assuming multicast-routing is enabled) yields positive results. When using per-VRF TE over an mLDP-based MVPN solution, there do not appear to be any MVPN concerns. The RPF failure is irrelevant since there was no P(S,G) tree to build. Even if CSR10 needed to join a specific P2MP mLDP tree, mLDP is allowed to use TE tunnels anyway, while PIM cannot. We prove that connectivity works by checking the multicast counters at the ingress (CSR5) and egress (CSR10) PEs. I also use EPC on the egress PE towards the customer to see the multicast being forwarded properly.

```
RP/0/0/CPU0:XRv3#ping vrf BLUE 232.200.4.4 source 200.13.0.1 count 100
```

```
R5#show ip mroute vrf BLUE 232.200.4.4 count | begin ^Group
Group: 232.200.4.4, Source count: 1, Packets forwarded: 52, Packets received:
52
Source: 200.13.0.1/32, Forwarding: 52/0/122/0, Other: 52/0/0
```

```
R10#show ip mroute vrf BLUE 232.200.4.4 count | begin ^Group
Group: 232.200.4.4, Source count: 1, Packets forwarded: 57, Packets received:
57
Source: 200.13.0.1/32, Forwarding: 57/0/122/0, Other: 57/0/0
```

```
R10#show monitor capture CAP buffer detail
0 122 0.000000 00:50:56:A9:F9:61 -> 01:00:5E:48:04:04 VLAN-tagged
frame
0000: 01005E48 04040050 56A9F961 81000DD4 ..^H...PV..a....
0010: 810000C8 08004500 00640047 0000FC01 .....E..d.G....
0020: 0977C80D 0001E8C8 04040800 1E30B09C .w.....0..
0030: 0047ABCD ABCDABCD ABCDABCD ABCDABCD .G.....
```

Next, we will repeat the process for VPN RED on CSR10 towards CSR5. Because the VPN next-hops are different, VPN RED can use an entirely different TE strategy. We will introduce a basic CBTS policy for VPN RED only; one tunnel will carry EXP 0-4 (low priority) while the other carries EXP 5-7 (high priority).

CBTS with per-VRF TE adds another dimension of granularity since we can separate traffic by VRF and EXP at the same time. CBTS is not discussed in detail here as it has a dedicated section.

```
! CSR10
ip explicit-path name EP_8_14_5 enable
  next-address 56.0.0.8
  next-address 56.0.0.14
  next-address 56.0.0.5

interface Tunnel5101
  description TO CSR5 RED CBTS MASTER
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 56.0.0.5
  tunnel mpls traffic-eng exp-bundle master
  tunnel mpls traffic-eng exp-bundle member Tunnel51017
  tunnel mpls traffic-eng exp-bundle member Tunnel51010

interface Tunnel51010
  description TO CSR5 RED LOW PRI
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 56.0.0.5
  tunnel mpls traffic-eng path-option 10 dynamic
  tunnel mpls traffic-eng exp default

interface Tunnel51017
  description TO CSR5 RED HI PRI
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 56.0.0.5
  tunnel mpls traffic-eng priority 7 7
  tunnel mpls traffic-eng bandwidth 65000
  tunnel mpls traffic-eng path-option 10 explicit name EP_8_14_5
  tunnel mpls traffic-eng exp 5 6 7
```

We quickly verify that the tunnels are up and that the EXP mappings are correct. We can accomplish this with a single command. Recall that the tunnel destination is always the remote TE-ID, not the per-VRF BGP next-hop value.

```
R10#show mpls traffic-eng exp 56.0.0.5
Destination: 56.0.0.5
  Master: Tunnel5101      Status: up
  Members      Status      Conf Exp      Actual Exp
  Tunnel51017  up (Active)  5 6 7        5 6 7
  Tunnel51010  up (Active)  Default      0 1 2 3 4
```

We quickly verify both paths using MPLS traceroute. We specify the master tunnel as the target while using EXP values to verify that CBTS is making the proper selection. We must force exp-null whenever XR is the PHP router, which was discussed in the MPLS OAM section; it isn't terribly relevant for this test. The main point is that the tunnels are following totally different paths.

```
R10#traceroute mpls traffic-eng tunnel 5101 exp 4
Tracing MPLS TE Label Switched Path on Tunnel5101, timeout is 2 seconds
[snip]
Type escape sequence to abort.
 0 56.8.10.10 MRU 1500 [Labels: 8018 Exp: 4]
L 1 56.8.10.8 MRU 1500 [Labels: implicit-null Exp: 4] 14 ms
! 2 56.5.8.5 3 ms

R10#traceroute mpls traffic-eng tunnel 5101 exp 5 force-explicit-null
Tracing MPLS TE Label Switched Path on Tunnel5101, timeout is 2 seconds
[snip]
Type escape sequence to abort.
 0 56.8.10.10 MRU 1500 [Labels: 8015/explicit-null Exp: 5/5]
L 1 56.8.10.8 MRU 1500 [Labels: 94006/explicit-null Exp: 5/5] 8 ms
L 2 56.8.14.14 MRU 1500 [Labels: implicit-null/explicit-null Exp: 5/5] 3 ms
! 3 56.5.14.5 5 ms
```

Just like VPN BLUE, a simple static route to 56.0.0.105/32 will direct traffic into the CBTS master tunnel. We use regular IP traceroute with varying DSCP values (0 and 56, or CS7) to test the TOS-to-EXP copy mechanism. We confirm that the MPLS labels match between the LSP and IP traceroutes.

```
! CSR10
ip route 56.0.0.105 255.255.255.255 Tunnel5101

R10#traceroute 56.0.0.105 source 56.0.0.10
Type escape sequence to abort.
Tracing the route to 56.0.0.105
VRF info: (vrf in name/id, vrf out name/id)
 1 56.8.10.8 [MPLS: Label 8018 Exp 0] 7 msec 3 msec 6 msec
 2 56.5.8.5 19 msec 8 msec 11 msec

R10#traceroute ip
Target IP address: 56.0.0.105
Source address: 56.0.0.10
DSCP Value [0]: 56
[snip]
VRF info: (vrf in name/id, vrf out name/id)
 1 56.8.10.8 [MPLS: Label 8015 Exp 7] 7 msec 6 msec 5 msec
 2 56.8.14.14 [MPLS: Label 94006 Exp 7] 15 msec 5 msec 5 msec
 3 56.5.14.5 8 msec 11 msec 11 msec
```

For brevity, we will use traceroute from CSR4 to quickly verify connectivity rather than trace the next-hop lookups. Traffic is classified based on their TOS/traffic-class values; we can confirm the topmost label of the VPN traffic matches the labels seen in the traceroutes above. The first traceroute uses IPP0 (label 8018) while the second uses IPP5 (label 8015).

```
R4#traceroute vrf RED 100.13.0.1 source 100.4.0.1
Type escape sequence to abort.
Tracing the route to 100.13.0.1
VRF info: (vrf in name/id, vrf out name/id)
 1 10.4.10.10 5 msec 4 msec 4 msec
 2 56.8.10.8 [MPLS: Labels 8018/5001 Exp 0] 6 msec 6 msec 6 msec
 3 10.5.13.5 [MPLS: Label 5001 Exp 0] 24 msec 20 msec 124 msec
 4 10.5.13.13 19 msec 15 msec 13 msec

R4#traceroute vrf RED ip
Target IP address: 100.13.0.1
Source address: 100.4.0.1
DSCP Value [0]: 40
[snip]
VRF info: (vrf in name/id, vrf out name/id)
 1 10.4.10.10 6 msec 3 msec 4 msec
 2 56.8.10.8 [MPLS: Labels 8015/5001 Exp 5] 6 msec 7 msec 7 msec
 3 56.8.14.14 [MPLS: Labels 94006/5001 Exp 5] 50 msec 31 msec 31 msec
 4 10.5.13.5 [MPLS: Label 5001 Exp 5] 16 msec 16 msec 15 msec
 5 10.5.13.13 20 msec 18 msec 14 msec
```

We have introduced a significant issue into the network by configuring this static route in VPN RED. Specifically, we have broken MVPN connectivity between CSR5 and CSR10. PIM notifies us of our folly on CSR10 a few minutes after configuring the static route.

```
! CSR10
%PIM-5-NBRCHG: VRF RED: neighbor 56.0.0.105 DOWN on interface Tunnel3 non DR
```

We will quickly backtrack to check some critical MVPN components. Since BGP IPv4 MDT is used for building P(S,G) trees to form the default MDT, we will start there. Earlier, we saw that CSR5 was smart enough to change the MDT source for mLDP trees. It is likewise smart enough to change the P-source in the GRE-based MDT trees as well. Below, we can see a P(S,G) of (56.0.0.105, 232.0.0.100) learned by CSR10. This happened automatically when we changed the BGP next-hop to 56.0.0.105 under the VRF. The next-hop of this particular prefix is 56.0.0.5, but this is less significant since the P-source is derived from the address in the IPv4 MDT prefix, which is 56.0.0.105.

```
R10#show bgp ipv4 mdt vrf RED 56.0.0.105/32 bestpath
BGP routing table entry for 56:100:56.0.0.105/32 version 18
Paths: (4 available, best #4, table IPv4-MDT-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 2
```

Local

```
56.0.0.5 from 56.0.0.1 (56.0.0.1)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  Originator: 56.0.0.5, Cluster list: 56.0.0.1,
  MDT group address: 232.0.0.100
  rx pathid: 0, tx pathid: 0x0
```

We can guess that, like VPN BLUE, the static route has introduced RPF issues into the network. CSR10 is not able to join CSR5's MDT, which means it cannot receive PIM messages from CSR5. That is why the PIM neighbor failed on CSR10 but not on CSR5 (CSR5 is still able to join CSR10's tree without issue). Unlike mLDLP, this is significant because CSR10 cannot issue the P(S,G) join towards CSR5.

```
R10#show ip mroute 232.0.0.100 56.0.0.105 | begin \
(56.0.0.105, 232.0.0.100), 1w5d/stopped, flags: sTIZ
  Incoming interface: Null, RPF nbr 0.0.0.0
  Outgoing interface list:
    MVRP RED, Forward/Sparse-Dense, 1w5d/00:01:52
```

```
R10#show ip rpf 56.0.0.105
  failed, no route exists
```

There are two solutions to this problem: static multicast route or BGP IPv4 multicast. The former is quick and easy, and since the focus of this lab is not MVPN, we will use that method. The latter method is more dynamic and would be more appropriate in large-scale deployments. Note that the IGP command "mpls traffic-eng multicast-intact" does not work with static routes for IS-IS or OSPF; we already have this command enabled to overcome the auto-primary tunnels used for MPLS transport. We can be intelligent when identifying the next-hop for this RPF fixup route by specifying CSR5's original loopback0 address. This recurses to an IGP route that will always be IGP learned, and since "mpls traffic-eng multicast-intact" is configured, its RPF interface will be an Ethernet interface of sorts (never a TE tunnel). This is more effective than specifying a hard next-hop, like CSR9 or CSR7 link addresses, since IGP may change over time. It's a semi-dynamic way to fix RPF with static multicast routes, in a sense.

```
! CSR10
ip mroute 56.0.0.105 255.255.255.255 56.0.0.5
```

```
R10#show ip rpf 56.0.0.105
RPF information for ? (56.0.0.105)
  RPF interface: GigabitEthernet2.570
  RPF neighbor: ? (56.7.10.7)
  RPF route/mask: 56.0.0.105/32
  RPF type: multicast (static)
  Doing distance-preferred lookups across tables
  RPF topology: ipv4 multicast base, originated from ipv4 unicast base
```

CSR10's PIM neighbor with CSR5 immediately reforms after CSR10 builds the P(S,G) tree. The P(S,G) state on CSR10 now references a valid RPF neighbor via the static mroute. CSR10 builds the P(S,G) tree

towards CS5 via CSR7. We won't trace the entire tree; the presence of PIM neighbors is proof that the default MDT has been signaled properly.

```
! CSR10
%PIM-5-NBRCHG: VRF RED: neighbor 56.0.0.105 UP on interface Tunnel3

R10#show ip mroute 232.0.0.100 56.0.0.105 | begin \((
(56.0.0.105, 232.0.0.100), 1w5d/stopped, flags: sTIZ
Incoming interface: GigabitEthernet2.570, RPF nbr 56.7.10.7, Mroute
Outgoing interface list:
MVRF RED, Forward/Sparse-Dense, 1w5d/00:02:32
```

With the MVPN control-plane repaired, we quickly test data traffic. Using a similar group address, we configure another IGMPv3 join on CSR4. This is inside the RED VPN and targets XRv3 again. CSR10 has the proper C(S,G) state and sends the C-PIM join to CSR5 who receives it successfully.

```
! CSR4
interface GigabitEthernet2.100
 ip igmp join-group 232.100.4.4 source 100.13.0.1

R10#show ip mroute vrf RED 232.100.4.4 100.13.0.1 | begin \((
(100.13.0.1, 232.100.4.4), 1w3d/00:02:18, flags: sTI
Incoming interface: Tunnel3, RPF nbr 56.0.0.105
Outgoing interface list:
GigabitEthernet2.100, Forward/Sparse-Dense, 1w3d/00:02:18

R5#show ip mroute vrf RED 232.100.4.4 100.13.0.1 | begin \((
(100.13.0.1, 232.100.4.4), 00:14:10/00:03:06, flags: sT
Incoming interface: GigabitEthernet2.100, RPF nbr 10.5.13.13
Outgoing interface list:
Tunnel3, Forward/Sparse, 00:14:10/00:03:06
```

XRv3 begins sending traffic to this group from the proper source. CSR5 and CSR10 both tally packets in their C-MRIB counters, and outbound EPC on CSR1 shows the multicast being delivered to CSR4.

```
R5#show ip mroute vrf RED 232.100.4.4 count | begin ^Group
Group: 232.100.4.4, Source count: 1, Packets forwarded: 19, Packets received:
19
Source: 100.13.0.1/32, Forwarding: 19/0/122/0, Other: 19/0/0

R10#show ip mroute vrf RED 232.100.4.4 count | begin ^Group
Group: 232.100.4.4, Source count: 1, Packets forwarded: 25, Packets received:
25
Source: 100.13.0.1/32, Forwarding: 25/0/142/0, Other: 25/0/0
```

```
R10#show monitor capture CAP buffer detailed
```

```

0 122      0.000000  00:50:56:A9:F9:61 -> 01:00:5E:64:04:04 VLAN-tagged
frame
0000:  01005E64 04040050 56A9F961 81000DD4  ..^d...PV..a....
0010:  81000064 08004500 0064000E 0000FD01  ...d...E..d.....
0020:  6D14640D 0001E864 04040800 AE69209C  m.d....d.....i .
0030:  000EABCD ABCDABCD ABCDABCD ABCDABCD  .....

```

The per-VRF TE method demonstrated on CSR5 is by far the most straightforward method. There are always alternatives, and we will demonstrate this on CSR9. The concept is the same; we will adjust BGP next-hops on CSR9 outbound towards the RRs to newly-configured loopbacks. This can be accomplished using an outbound route-map; for even finer granularity, we can achieve per-prefix or per-RT TE using this method. To keep the network configuration consistent, we will configure per-RT TE on CSR9, which is effectively the same as per-VRF TE since each VRF in each VPN uses a common RT format. First, we will create two loopbacks, advertise them into IGP, and construct our per-RT “next-hop-self” route-maps. The route-map should always have a “permit any” as the final entry so that non-TE VRFs can continue to use loopback0 as their VPN endpoint for dynamic LDP LSPs.

```

! CSR9
interface Loopback100
  description RED TE
  ip address 56.0.0.109 255.255.255.255
  ip pim passive
  isis tag 56

interface Loopback200
  description BLUE TE
  ip address 56.0.0.209 255.255.255.255
  ip pim passive
  isis tag 56

router isis 56
  passive-interface Loopback100
  passive-interface Loopback200

ip extcommunity-list expanded EXTCOML_BLUE_RT permit RT:56:2..$
ip extcommunity-list expanded EXTCOML_RED_RT permit RT:56:1..$

route-map RM_VPN_NHOP permit 10
  match extcommunity EXTCOML_RED_RT
  set ip next-hop 56.0.0.109
  set ipv6 next-hop ::FFFF:56.0.0.109
route-map RM_VPN_NHOP permit 20
  match extcommunity EXTCOML_BLUE_RT
  set ip next-hop 56.0.0.209
  set ipv6 next-hop ::FFFF:56.0.0.209
route-map RM_VPN_NHOP permit 1000

```


Before we apply the route-maps to BGP, we will ensure these new loopbacks have been advertised into IGP properly. This preliminary check will prevent blackholed traffic during a migration. Additionally, binding LDP labels to these prefixes is a good idea so that if the TE tunnel fails, a dynamic LDP LSP could be used as a last resort.

```
R9#show isis database R9.00-00 detail level-2 | include [12]09
Metric: 0          IP 56.0.0.109/32
Metric: 0          IP 56.0.0.209/32
```

```
R9#show mpls ldp bindings 56.0.0.109 32
lib entry: 56.0.0.109/32, rev 31
  local binding: label: imp-null
  remote binding: lsr: 56.0.0.10:0, label: 10029
  remote binding: lsr: 56.0.0.7:0, label: 7015
```

```
R9#show mpls ldp bindings 56.0.0.209 32
lib entry: 56.0.0.209/32, rev 33
  local binding: label: imp-null
  remote binding: lsr: 56.0.0.10:0, label: 10030
  remote binding: lsr: 56.0.0.7:0, label: 7016
```

Next, we will apply the route-maps to all VPNv4 and VPNv6 peers. We do not need to create two route-maps since the “set ip next-hop” and “set ipv6 next-hop” commands are complementary and mutually-exclusive. Only one can apply for a given route as this depends on the prefix type being matched. As a side note, peer-groups or peer-policy templates could have shortened this configuration.

```
! CSR9
router bgp 56
  address-family vpnv4
    neighbor 56.0.0.1 route-map RM_VPN_NHOP out
    neighbor 56.0.0.2 route-map RM_VPN_NHOP out
    neighbor 56.0.0.11 route-map RM_VPN_NHOP out
    neighbor 56.0.0.12 route-map RM_VPN_NHOP out

  address-family vpnv6
    neighbor 56.0.0.1 route-map RM_VPN_NHOP out
    neighbor 56.0.0.2 route-map RM_VPN_NHOP out
    neighbor 56.0.0.11 route-map RM_VPN_NHOP out
    neighbor 56.0.0.12 route-map RM_VPN_NHOP out
```

After applying the configuration, I perform a soft clear for VPNv4 and VPNv6 to ensure the changes took effect. Rather than check all 4 RRs for the next-hop change, we will jump straight to CSR10 for a cursory check. CSR10 now sees next-hops 56.0.0.109 and 56.0.0.209 within the RED and BLUE VPNs, respectively. We see 8 routes each, as expected, since there are 4 routes times 2 route-reflectors.

```
R9#clear bgp vpnv4 unicast * soft
```

```
R9#clear bgp vpnv6 unicast * soft
```

```
R10#show bgp vpnv4 unicast vrf RED nexthops
```

```
# Paths    Nexthop Address
Route Distinguisher: 56:100 (default for vrf RED)
  8  56.0.0.109
  4  10.4.10.4 (RED)
  8  56.0.0.14
  8  56.0.0.105
```

```
R10#show bgp vpnv4 unicast vrf BLUE nexthops
```

```
# Paths    Nexthop Address
Route Distinguisher: 56:200 (default for vrf BLUE)
  8  56.0.0.209
  4  10.4.10.4 (BLUE)
  8  56.0.0.14
  8  56.0.0.205
```

Because we applied a route-map directly to BGP and did not “inform” the VRF of these next-hop changes, we do not expect any change in the MDT source. That is, the P-source for the MDT in both VPNs should still be 56.0.0.9. Checking the RED VPN, the IPv4 MDT routes all carry P-source 56.0.0.9 which is contained in the prefix.

```
R10#show bgp ipv4 mdt vrf RED | include 56.0.0.9
```

```
*>i 56.0.0.9/32      56.0.0.9      0      100      0 ?
* i      56.0.0.9      0      100      0 ?
* i      56.0.0.9      0      100      0 ?
* i      56.0.0.9      0      100      0 ?
```

This means that a static mroute will not be necessary to build to P(S,G) tree back to CSR9 since CSR10 has a valid RPF route for this prefix already. With respect to GRE-based MVPN, this appears “easier” than the purpose-built method of changing the BGP next-hop under the VRF (method on CSR5).

```
R10#show ip rpf 56.0.0.9
```

```
RPF information for ? (56.0.0.9)
```

```
RPF interface: GigabitEthernet2.590
```

```
RPF neighbor: ? (56.9.10.9)
```

```
RPF route/mask: 56.0.0.9/32
```

```
RPF type: unicast (isis 56)
```

```
Doing distance-preferred lookups across tables
```

```
RPF topology: ipv4 multicast base, originated from ipv4 unicast base
```

```
R10#show ip mroute 232.0.0.100 56.0.0.9 | begin \((
```

```
(56.0.0.9, 232.0.0.100), 2w4d/stopped, flags: sTIZ
```

```
Incoming interface: GigabitEthernet2.590, RPF nbr 56.9.10.9
```

```
Outgoing interface list:
```

```
MVRF RED, Forward/Sparse-Dense, 2w4d/00:00:17
```

One final check is warranted before continuing. Recall the purpose of the BGP connector attribute from the inter-AS MPLS option B discussion. It is used to ensure that the traffic arriving from the PMSI is coming from the actual ingress PE (and not an ASBR, etc). If the attribute is present, its value must match that of the PIM neighbor. This was value not changed by CSR9 since the route-map was applied directly to BGP; notice the difference between the next-hop and connector attribute values.

```
R10#show bgp vpnv4 unicast vrf RED 100.3.0.0 bestpath
BGP routing table entry for 56:100:100.3.0.0/18, version 389
Paths: (2 available, best #2, table RED)
  Advertised to update-groups:
    2
  Refresh Epoch 2
  100
    56.0.0.109 (via default) from 56.0.0.1 (56.0.0.1)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:56:103
      Originator: 56.0.0.9, Cluster list: 56.0.0.1
      Connector Attribute: count=1
        type 1 len 12 value 56:100:56.0.0.9
      mpls labels in/out nlabel/9026
      rx pathid: 0, tx pathid: 0x0
```

Because VRF-aware PIM is also unaware of this next-hop change, the MDT source remains loopback0, which is correct. The RPF check for incoming packets from the PMSI should pass since both the connector attribute and PIM neighbor address are 56.0.0.9.

```
R10#show ip pim vrf RED neighbor | begin ^Neigh
Neighbor          Interface          Uptime/Expires    Ver    DR
Address
56.0.0.105        Tunnel3              00:47:28/00:01:30 v2      1 / S P G
56.0.0.14         Tunnel3              1w5d/00:01:43    v2      1 / DR P G
56.0.0.9          Tunnel3              2w4d/00:01:31    v2      1 / S P G
```

We will configure a basic TE tunnel from CSR10 to CSR9 to service VPN RED. The tunnel will avoid the direct path because it isn't very interesting. For brevity, we add the static route now since we already know it is required.

```
! CSR9
ip explicit-path name EP_AVOID_LINK_9_10 enable
  exclude-address 56.9.10.9
  exclude-address 56.9.10.10

interface Tunnel9101
  description TO CSR9 RED
  ip unnumbered Loopback0
```

```

tunnel mode mpls traffic-eng
tunnel destination 56.0.0.9
tunnel mpls traffic-eng path-option 10 explicit name EP_AVOID_LINK_9_10

ip route 56.0.0.109 255.255.255.255 Tunnel9101

```

We quickly confirm the tunnel is up, , reveal the first-hop label, then use IP traceroute to verify. This proves that traffic is being directed into the tunnel properly.

```

R10#show mpls traffic-eng tunnels tunnel 9101 brief | begin TUNNEL
TUNNEL NAME          DESTINATION      UP IF      DOWN IF     STATE/PROT
TO CSR9 RED          56.0.0.9        -          Gi2.570     up/up

R10#show mpls traffic-eng tunnels tunnel 9101 | include Label
InLabel   : -
OutLabel  : GigabitEthernet2.570, 7018

R10#traceroute 56.0.0.109 source 56.0.0.10
Type escape sequence to abort.
Tracing the route to 56.0.0.109
VRF info: (vrf in name/id, vrf out name/id)
 1 56.7.10.7 [MPLS: Label 7018 Exp 0] 8 msec 4 msec 4 msec
 2 56.7.9.9 3 msec 4 msec 3 msec

```

To complete the unicast verification, we perform a traceroute from CSR4 to CSR3. I use IPv6 for variety, but since 6VPE next-hops are IPv4-based, both IPv4 and IPv6 will follow these per-RT tunnels.

```

R4#traceroute vrf RED ipv6
Target IPv6 address: 2100:D103::1
Source address: 2100:D104::1
[snip]
 1 FD00:10:4:10::10 4 msec 4 msec 4 msec
 2 ::FFFF:56.7.10.7 [MPLS: Labels 7018/9033 Exp 0] 15 msec 6 msec 24 msec
 3 FD00:10:3:9::9 [MPLS: Label 9033 Exp 0] 17 msec 17 msec 17 msec
 4 FD00:10:3:9::3 22 msec 25msec 15 msec

```

Next, we will confirm RED MVPN traffic flows correctly. CSR4 joins an SSM group and sends the IGMPv3 join to CSR10. CSR10 passes it to CSR9 and both routers show the correct C(S,G) SSM state below.

```

! CSR4
interface GigabitEthernet2.100
 ip igmp join-group 232.100.4.4 source 100.3.0.1

R10#show ip mroute vrf RED 232.100.4.4 100.3.0.1 | begin \
(100.3.0.1, 232.100.4.4), 1w3d/00:02:03, flags: sTI
Incoming interface: Tunnel3, RPF nbr 56.0.0.9
Outgoing interface list:

```

```
GigabitEthernet2.100, Forward/Sparse-Dense, 1w3d/00:02:03
```

```
R9#show ip mroute vrf RED 232.100.4.4 100.3.0.1 | begin \  
(100.3.0.1, 232.100.4.4), 1w3d/00:02:34, flags: sT  
  Incoming interface: GigabitEthernet2.100, RPF nbr 10.3.9.3  
  Outgoing interface list:  
    Tunnel3, Forward/Sparse, 1w3d/00:02:34
```

CSR3 begins sending traffic from the proper source. Both MVPN endpoints sees tallies for this traffic, and CSR10 uses outbound EPC to verify delivery to CSR4. Everything appears to work well without any RPF fixups needed. Also notice CSR10's RPF lookup inside the VPN references the BGP connector attribute. In summary, the BGP route-map option works well with MVPN profile 0.

```
R3#ping vrf RED 232.100.4.4 source 100.3.0.1 repeat 100
```

```
R9#show ip mroute vrf RED 232.100.4.4 100.3.0.1 count | begin ^Group  
Group: 232.100.4.4, Source count: 1, Packets forwarded: 6, Packets received:  
6  
  Source: 100.3.0.1/32, Forwarding: 6/0/122/0, Other: 6/0/0
```

```
R10#show ip mroute vrf RED 232.100.4.4 100.3.0.1 count | begin ^Group  
Group: 232.100.4.4, Source count: 3, Packets forwarded: 112, Packets  
received: 112  
  Source: 100.3.0.1/32, Forwarding: 12/0/142/0, Other: 12/0/0
```

```
R10#show ip rpf vrf RED 100.3.0.0  
RPF information for ? (100.3.0.0)  
  RPF interface: Tunnel3  
  RPF neighbor: ? (56.0.0.9)  
  RPF route/mask: 100.3.0.0/18  
  RPF type: unicast (bgp 56)  
  Doing distance-preferred lookups across tables  
  BGP originator: 56.0.0.9  
  RPF topology: ipv4 multicast base, originated from ipv4 unicast base
```

```
R10#show monitor capture CAP buffer detailed  
  0 122 0.000000 00:50:56:A9:F9:61 -> 01:00:5E:64:04:04 VLAN-tagged  
frame  
  0000: 01005E64 04040050 56A9F961 81000DD4 ..^d...PV..a....  
  0010: 81000064 08004500 00640056 0000FD01 ...d..E..d.V....  
  0020: 6CD66403 0001E864 04040800 7E150014 1.d....d....~...  
  0030: 00120000 000083A4 7C6AABCD ABCDABCD .....|j.....
```

Next, we will repeat the same test for the BLUE VPN, which relies on mLDP MP2MP trees. Recall that mLDP-based MVPN does not impose the BGP connector attribute onto VPN updates. It also does not rely on IPv4 MDT to tell it the P-source; in our setup, it is derived from the BGP next-hop. As such, CSR10 will see a PIM neighbor of 56.0.0.209 rather than 56.0.0.9; we theorize that this will cause issues with

PIM signaling later. To prove it, we will create another TE tunnel to CSR9 and direct traffic towards 56.0.0.209 into the tunnel.

```
! CSR10
ip explicit-path name EP_8_7_9 enable
  next-address 56.0.0.8
  next-address 56.0.0.7
  next-address 56.0.0.9

interface Tunnel9102
  description TO CSR9 BLUE
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 56.0.0.9
  tunnel mpls traffic-eng path-option 10 explicit name EP_8_7_9

ip route 56.0.0.209 255.255.255.255 Tunnel9102
```

This should immediately work for unicast flows; the tunnel comes up and routes traffic without issue. We verify the tunnel is up and reveal the first-hop label, then use IP traceroute as confirmation.

```
R10#show mpls traffic-eng tunnels tunnel 9102 brief | begin TUNNEL
TUNNEL NAME          DESTINATION      UP IF    DOWN IF    STATE/PROT
TO CSR9 BLUE         56.0.0.9        -        Gi2.580    up/up

R10#show ip rsvp reservation detail filter session-type 7 tunnel-id 9102 |
include Label
Label: 8006 (outgoing)

R10#traceroute 56.0.0.209 source 56.0.0.10
Type escape sequence to abort.
Tracing the route to 56.0.0.209
VRF info: (vrf in name/id, vrf out name/id)
 1 56.8.10.8 [MPLS: Label 8006 Exp 0] 7 msec 5 msec 4 msec
 2 56.7.8.7 [MPLS: Label 7021 Exp 0] 3 msec 5 msec 4 msec
 3 56.7.9.9 5 msec 5 msec 6 msec
```

We don't expect any issues with our unicast LSPs inside the VPN, so we verify it quickly. Using IPv4, we verify the path to CSR3 inside VPN BLUE ends up in the tunnel with 8006 as the topmost label.

```
R4#traceroute vrf BLUE 200.3.0.1 source 200.4.0.1
Type escape sequence to abort.
Tracing the route to 200.3.0.1
VRF info: (vrf in name/id, vrf out name/id)
 1 10.4.10.10 6 msec 4 msec 4 msec
 2 56.8.10.8 [MPLS: Labels 8006/9034 Exp 0] 6 msec 5 msec 7 msec
 3 56.7.8.7 [MPLS: Labels 7021/9034 Exp 0] 31 msec 31 msec 31 msec
```

```
4 10.3.9.9 [MPLS: Label 9034 Exp 0] 15 msec 14 msec 16 msec
5 10.3.9.3 20 msec 10 msec 10 msec
```

Next, we will repeat the same MVPN test for the BLUE VPN, which relies on mLDP MP2MP trees. We configure an IGMPv3 join on CSR4 then verify CSR10's reception of the message. Right away we can see a problem; CSR10 thinks 56.0.0.209 is the RPF neighbor as this was derived from the BGP next-hop.

```
! CSR4
interface GigabitEthernet2.200
 ip igmp join-group 232.200.4.4 source 200.3.0.1

R10#show ip mroute vrf BLUE 232.200.4.4 200.3.0.1 | begin \
(200.3.0.1, 232.200.4.4), 1w3d/00:02:03, flags: sTI
  Incoming interface: Lspvif0, RPF nbr 56.0.0.209
  Outgoing interface list:
    GigabitEthernet2.200, Forward/Sparse-Dense, 1w3d/00:02:03
```

CSR9 never receives the join because CSR10 cannot send it. We estimate that the lack of an RPF path to 56.0.0.209 prevents MVPN from functioning properly in this design. This is a similar problem we had with the traditional per-VRF design.

```
R10#show ip rpf 56.0.0.209
  failed, no route exists

R9#show ip mroute vrf BLUE 232.200.4.4 200.3.0.1
  Group 232.200.4.4 not found
```

As a quick workaround, we configure a static mroute on CSR10 so that 56.0.0.209's RPF path is equivalent to that of 56.0.0.9. This fixes the RPF issue.

```
! CSR10
ip mroute 56.0.0.209 255.255.255.255 56.0.0.9

R10#show ip rpf 56.0.0.209
RPF information for ? (56.0.0.209)
  RPF interface: GigabitEthernet2.590
  RPF neighbor: ? (56.9.10.9)
  RPF route/mask: 56.0.0.209/32
  RPF type: multicast (static)
  Doing distance-preferred lookups across tables
  RPF topology: ipv4 multicast base, originated from ipv4 unicast base
```

Despite our best efforts, this does not solve the problem. As mentioned earlier, PIM signaling is broken between CSR10 and CSR9. A fundamental behavior of PIM is that the RPF neighbor and the PIM neighbor must have the same IP address in order for PIM signaling to work. For example, if the PIM neighbor is 56.0.0.9, it would not make sense to send PIM signaling to 56.0.0.209 because RPF says so.

```
R10#show ip pim vrf BLUE neighbor | begin ^Neigh
Neighbor          Interface          Uptime/Expires    Ver    DR
Address
56.0.0.9           Lspvif0          00:01:00/00:01:43 v2      1 / S P G
56.0.0.205         Lspvif0          00:01:00/00:01:43 v2      1 / DR S P G
```

In reality, CSR10 simply does nothing. After watching its debug for a few minutes, we can confirm this behavior. C-PIM joins are being sent to CSR5 since the PIM neighbor and RPF neighbor are both 56.0.0.205, which was automatically coordinated by the per-VRF TE feature. With CSR9, we have a disconnect due to our manual next-hop changes; CSR10's resulting confusion means the tree never gets built. This was not a problem for the RED VPN multicast connectivity with CSR9 since 56.0.0.9 was both the PIM neighbor and RPF neighbor IP address. Thus, **when using the route-map method on XE, certain MVPN profiles (such as mLDP profile 1 currently in use) appear unsupported.** Although this method alleviated the need to configure an mroute in the RED VPN, it is overall the less preferred option.

```
R10#debug ip pim vrf BLUE
PIM debugging is on
```

```
PIM(2): Insert (200.13.0.1,232.200.4.4) join in nbr 56.0.0.205's queue
PIM(2): Building Join/Prune packet for nbr 56.0.0.205
PIM(2): Adding v2 (200.13.0.1/32, 232.200.4.4), S-bit Join
PIM(2): Send v2 join/prune to 56.0.0.205 (Lspvif0)
```

```
PIM(2): Insert (200.13.0.1,232.200.4.4) join in nbr 56.0.0.205's queue
PIM(2): Building Join/Prune packet for nbr 56.0.0.205
PIM(2): Adding v2 (200.13.0.1/32, 232.200.4.4), S-bit Join
PIM(2): Send v2 join/prune to 56.0.0.205 (Lspvif0)
```

Because this static mroute will not solve the problem, I raise the AD to 255 so that it is retained in the configuration but has no effect. We leave multicast broken for the BLUE VPN between CSR9 and CSR10.

```
! CSR10
ip mroute 56.0.0.209 255.255.255.255 56.0.0.9 255
```

Last, we examine the per-VRF TE feature on XRv4. Since LSM does not appear to be fully supported on XRv as a PE (imposition operations appear to work for some profiles), we will focus on the RED VPN. XR does not have an equivalent “bgp next-hop” command anywhere in the parser. As such, we will use a combination of the per-VRF and route-map approaches configured on CSR5 and CSR9. First, we will configure new loopbacks, advertise them into IGP, and verify success. We also verify LDP label bindings for these prefixes as a backup.

```
! XRv4
interface Loopback100
  ipv4 address 56.0.0.114 255.255.255.255
```



```
interface Loopback200
  ipv4 address 56.0.0.214 255.255.255.255
```

```
router isis 56
  interface Loopback100
    passive
    address-family ipv4 unicast
    tag 56
```

```
interface Loopback200
  passive
  address-family ipv4 unicast
  tag 56
```

```
RP/0/0/CPU0:XRv4#show isis database XRv4.00-00 detail level 2 | include
[12]14
```

```
  Metric: 0          IP-Extended 56.0.0.114/32
  Metric: 0          IP-Extended 56.0.0.214/32
```

```
RP/0/0/CPU0:XRv4#show mpls ldp bindings 56.0.0.114/32
56.0.0.114/32, rev 33
```

```
  Local binding: label: ImpNull
```

```
  Remote bindings: (2 peers)
```

Peer	Label
56.0.0.5:0	5014
56.0.0.8:0	8007

```
RP/0/0/CPU0:XRv4#show mpls ldp bindings 56.0.0.214/32
56.0.0.214/32, rev 39
```

```
  Local binding: label: ImpNull
```

```
  Remote bindings: (2 peers)
```

Peer	Label
56.0.0.5:0	5023
56.0.0.8:0	8000

Next, we will stage our RPL constructs. Using parameterization, we can feed IPv4 or IPv6 next-hops to be applied to RED or BLUE routes based on the RTs they carry. Using parameterization in this way is similar to XE using IPv4 and IPv6 next-hop applications in the same route-map clause. This also gives us the additional flexibility of defining different next-hops for IPv4 and IPv6; this is also possible in XE by simply specifying different loopbacks.

```
! XRv4
extcommunity-set rt RT_RED
  ios-regex '56:1..$'
end-set
```

```

extcommunity-set rt RT_BLUE
  ios-regex '56:2..$'
end-set

```

```

route-policy RPL_SET_NHOP($NHRED, $NHBLUE)
  if extcommunity rt matches-any RT_RED then
    set next-hop $NHRED
  elseif extcommunity rt matches-any RT_BLUE then
    set next-hop $NHBLUE
  else
    pass
  endif
end-policy

```

XR gives administrators the flexibility to define per-VRF MDT sources. This is the key to providing MVPN services when the BGP next-hop differs. XE has no equivalent command since it expects you to use the “bgp next-hop” feature, not the manual route-maps. Even though we won’t be testing BLUE MVPN traffic on XRv4, the configuration is shown for completeness. Specifying the MDT source under the global IPv4/v6 processes makes it apply to all VRFs (which is loopback0). This can be overridden on a per-VRF basis using the more specific VRF-aware commands shown below. This is a good approach because other VRFs that are not using per-VRF TE can continue to use a common MDT source.

```

! XRv4
multicast-routing
  address-family ipv4
    mdt source Loopback0
  address-family ipv6
    mdt source Loopback0

vrf RED
  address-family ipv4
    mdt source Loopback100
  interface all enable
  address-family ipv6
    mdt source Loopback100
  interface all enable

vrf BLUE
  address-family ipv4
    mdt source Loopback200
  interface all enable
  address-family ipv6
    mdt source Loopback200
  interface all enable

```

Changing the MDT source and BGP next-hop should happen concurrently, since both must be synchronized in order for unicast and multicast services to work. We quickly apply our BGP policy to minimize any issues; normally both changes would be grouped into a single commit action. Note that 6VPE next-hops in XR are entered as ordinary IPv4 addresses. **It is critical to instruct BGP to enforce outbound changes for iBGP peers**, which is not the default behavior in XR. This is required here since the RRs are all iBGP peers; another use case is unified MPLS on the ABRs between core and aggregation IGPs. Just like in XE, we could have simplified this configuration using neighbor-groups or AF-groups.

```
! XRv4
router bgp 56
  ibgp policy out enforce-modifications
  neighbor 56.0.0.1
    address-family vpnv4 unicast
      route-policy RPL_SET_NHOP(56.0.0.114, 56.0.0.214) out
    address-family vpnv6 unicast
      route-policy RPL_SET_NHOP(56.0.0.114, 56.0.0.214) out

  neighbor 56.0.0.2
    address-family vpnv4 unicast
      route-policy RPL_SET_NHOP(56.0.0.114, 56.0.0.214) out
    address-family vpnv6 unicast
      route-policy RPL_SET_NHOP(56.0.0.114, 56.0.0.214) out

  neighbor 56.0.0.11
    address-family vpnv4 unicast
      route-policy RPL_SET_NHOP(56.0.0.114, 56.0.0.214) out
    address-family vpnv6 unicast
      route-policy RPL_SET_NHOP(56.0.0.114, 56.0.0.214) out

  neighbor 56.0.0.12
    address-family vpnv4 unicast
      route-policy RPL_SET_NHOP(56.0.0.114, 56.0.0.214) out
    address-family vpnv6 unicast
      route-policy RPL_SET_NHOP(56.0.0.114, 56.0.0.214) out
```

We quickly check CSR10 to ensure that it sees these BGP next-hop changes. As this point, we can see that CSR5, CSR9, and XRv4 all use custom next-hops in both VPNs. This is the expected output.

```
R10#show bgp vpnv4 unicast vrf RED nexthops
# Paths      Nexthop Address
Route Distinguisher: 56:100 (default for vrf RED)
      8      56.0.0.109
      4      10.4.10.4 (RED)
      8      56.0.0.114
      8      56.0.0.105
```

```
R10#show bgp vpnv4 unicast vrf BLUE nexthops
```

```

# Paths    Nexthop Address
Route Distinguisher: 56:200 (default for vrf BLUE)
      8    56.0.0.209
      4    10.4.10.4 (BLUE)
      8    56.0.0.214
      8    56.0.0.205

```

Before continuing, we will ensure the MVPN control-plane is correct. CSR10 maintains a PIM neighbor with 56.0.0.114. This is the expected IP address since we changed the MDT source manually. Looking at a VPNv4 route's detailed output, we can see the BGP next-hop and connector attribute both reflect this MDT source. This is an ideal situation; just like CSR5's "bgp next-hop" method, all addressing is consistent which simplifies troubleshooting significantly.

```

R10#show ip pim vrf RED neighbor | begin ^Neigh
Neighbor          Interface          Uptime/Expires    Ver    DR
Address
56.0.0.114        Tunnel3          00:21:56/00:01:26 v2      1 / DR P G
56.0.0.9          Tunnel3          00:21:59/00:01:23 v2      1 / S P G
56.0.0.105        Tunnel3          00:21:59/00:01:25 v2      1 / S P G

```

```

R10#show bgp vpnv4 unicast vrf RED 100.6.0.0/18 bestpath
BGP routing table entry for 56:100:100.6.0.0/18, version 74
Paths: (2 available, best #1, table RED)
  Advertised to update-groups:
    2
  Refresh Epoch 2
  100
    56.0.0.114 (via default) from 56.0.0.1 (56.0.0.1)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:56:114
      Originator: 56.0.0.14, Cluster list: 56.0.0.1
      Connector Attribute: count=1
        type 1 len 12 value 56:100:56.0.0.114
      mpls labels in/out nlabel/94027
      rx pathid: 0, tx pathid: 0x0

```

Since we aren't verifying BLUE MVPN traffic, we will configure the BLUE VPN TE tunnel first. We create a basic tunnel with no fancy features and add the required static route.

```

! CSR10
ip explicit-path name EP_7_8_14 enable
  next-address 56.0.0.7
  next-address 56.0.0.8
  next-address 56.0.0.14

interface Tunnel10142
  description TO XRV4 BLUE

```

```

ip unnumbered Loopback0
tunnel mode mpls traffic-eng
tunnel destination 56.0.0.14
tunnel mpls traffic-eng path-option 10 explicit name EP_7_8_14

ip route 56.0.0.214 255.255.255.255 Tunnel10142

```

A quick verification shows that the tunnel is up. With a first hop label of 9032 verified by the global IP traceroute, we verify this is consistent from inside the VPN using IPv6. Everything works as expected. So far, from the perspective of CSR10, everything is the same.

```

R10#show mpls traffic-eng tunnels tunnel 10142 brief | begin TUNNEL
TUNNEL NAME          DESTINATION      UP IF      DOWN IF     STATE/PROT
TO XRV4 BLUE         56.0.0.14       -          Gi2.570     up/up

```

```

R10#show mpls traffic-eng tunnels tunnel 10142 | include Label
InLabel   : -
OutLabel  : GigabitEthernet2.570, 7019

```

```

R10#traceroute 56.0.0.214 source 56.0.0.10
Type escape sequence to abort.
Tracing the route to 56.0.0.214
VRF info: (vrf in name/id, vrf out name/id)
 1 56.7.10.7 [MPLS: Label 7019 Exp 0] 5 msec 5 msec 17 msec
 2 56.7.8.8 [MPLS: Label 8008 Exp 0] 30 msec 36 msec 30 msec
 3 56.8.14.14 18 msec 13 msec 13 msec

```

```

R4#traceroute vrf BLUE ipv6
Target IPv6 address: 2200:B206::1
Source address: 2200:B204::1
[snip]
 1 FD00:10:4:10::10 5 msec 4 msec 3 msec
 2 ::FFFF:56.7.10.7 [MPLS: Labels 7019/94009 Exp 0] 8 msec 8 msec 13 msec
 3 ::FFFF:56.7.8.8 [MPLS: Labels 8008/94009 Exp 0] 34 msec 34 msec 34 msec
 4 FD00:10:6:14::14 [MPLS: Label 94009 Exp 0] 23 msec 22 msec 23 msec
 5 FD00:10:6:14::6 22 msec 15 msec 15 msec

```

We repeat the same test for VPN RED. The configuration is very basic and shown below.

```

! CSR10
ip explicit-path name EP_9_7_5_14 enable
next-address 56.0.0.9
next-address 56.0.0.7
next-address 56.0.0.5
next-address 56.0.0.14

interface Tunnel10141
description TO XRV4 RED

```

```

ip unnumbered Loopback0
tunnel mode mpls traffic-eng
tunnel destination 56.0.0.14
tunnel mpls traffic-eng path-option 10 explicit name EP_9_7_5_14

```

```
ip route 56.0.0.114 255.255.255.255 Tunnel10141
```

We verify that the tunnel is up before continuing. The label used is 9032 and this is revealed by the global and VPN-aware IP traceroutes. Since there is nothing new from CSR10's perspective, we move very quickly through these verifications.

```

R10#show mpls traffic-eng tunnels tunnel 10141 | section Status|Label
Status:
  Admin: up      Oper: up      Path: valid      Signalling: connected
  path option 10, type explicit EP_9_7_5_14 (Basis for Setup, path weight
40)

```

```

InLabel  : -
OutLabel  : GigabitEthernet2.590, 9032

```

```

R10#traceroute 56.0.0.114 source 56.0.0.10
Type escape sequence to abort.
Tracing the route to 56.0.0.114
VRF info: (vrf in name/id, vrf out name/id)
 1 56.9.10.9 [MPLS: Label 9032 Exp 0] 7 msec 6 msec 6 msec
 2 56.7.9.7 [MPLS: Label 7011 Exp 0] 6 msec 6 msec 6 msec
 3 56.5.7.5 [MPLS: Label 5022 Exp 0] 3 msec 8 msec 15 msec
 4 56.5.14.14 15 msec 9 msec 8 msec

```

```

R4#traceroute vrf RED ipv6
Target IPv6 address: 2100:B106::1
Source address: 2100:B104::1
[snip]
 1 FD00:10:4:10::10 5 msec 3 msec 3 msec
 2 ::FFFF:56.9.10.9 [MPLS: Labels 9032/94032 Exp 0] 9 msec 8 msec 10 msec
 3 ::FFFF:56.7.9.7 [MPLS: Labels 7011/94032 Exp 0] 23 msec 23 msec 23 msec
 4 ::FFFF:56.5.7.5 [MPLS: Labels 5022/94032 Exp 0] 23 msec 23 msec 23 msec
 5 FD00:10:6:14::14 [MPLS: Label 94032 Exp 0] 23 msec 23 msec 23 msec
 6 FD00:10:6:14::6 21 msec 17 msec 15 msec

```

To test MVPN, we quickly add an SSM join to CSR4 as we have done many times before. The source is CSR6, which means the C(S,G) join should be sent towards XRv4 across the RED default MDT. Like CSR5's VPN RED test, we expect that adding the static route will cause RPF issues. Just like with CSR5, the PIM neighbor fails on CSR10 after a few minutes. Note that the failure of the PIM neighbor is not a result of having configured a multicast listener on CSR4; it would have happened no matter what.

```
! CSR4
```

```

interface GigabitEthernet2.100
 ip igmp join-group 232.100.4.4 source 100.6.0.1

R10#show ip mroute vrf RED 232.100.4.4 100.6.0.1 | begin \
(100.6.0.1, 232.100.4.4), 00:40:05/00:02:53, flags: sTI
  Incoming interface: Tunnel3, RPF nbr 56.0.0.114
  Outgoing interface list:
    GigabitEthernet2.100, Forward/Sparse-Dense, 00:40:05/00:02:53

R10#show ip rpf 56.0.0.114
  failed, no route exists

! CSR10
%PIM-5-NBRCHG: VRF RED: neighbor 56.0.0.114 DOWN on interface Tunnel3 DR

```

The easiest fix is adding a static mroute towards XRv4's loopback0, which is an IGP route for which CSR10 has a valid RPF. The PIM neighbor reforms as expected, and XRv4 shows the C(S,G) state from CSR10.

```

! CSR10
ip mroute 56.0.0.114 255.255.255.255 56.0.0.14

%PIM-5-NBRCHG: VRF RED: neighbor 56.0.0.114 UP on interface Tunnel3

R10#show ip rpf 56.0.0.114
RPF information for ? (56.0.0.114)
  RPF interface: GigabitEthernet2.580
  RPF neighbor: ? (56.8.10.8)
  RPF route/mask: 56.0.0.114/32
  RPF type: multicast (static)
  Doing distance-preferred lookups across tables
  RPF topology: ipv4 multicast base, originated from ipv4 unicast base

RP/0/0/CPU0:XRv4#show pim vrf RED topology 232.100.4.4 | begin 232
(100.6.0.1,232.100.4.4)SPT SSM Up: 00:43:40
JP: Join(00:00:10) RPF: GigabitEthernet0/0/0/0.100,10.6.14.6 Flags:
  mdtRED 00:43:40 fwd Join(00:03:01)

```

As a final verification, CSR6 will begin sending multicast traffic into the network. XRv4 encapsulates this into the P(S,G) representing the RED default MDT, which is (56.0.0.114, 232.0.0.100) in this case. The source of these packets is XRv4's loopback100 as all routes in the core will have valid RPF routes for this. CSR10 also expects traffic from this MDT source since that is the address with which a PIM neighbor was build. The C-multicast counters are increasing and CSR10's outbound EPC proves that the operation was successful.

```

R6#ping vrf RED 232.100.4.4 source 100.6.0.1 repeat 100

```

```

RP/0/0/CPU0:XRv4#show mfib vrf RED route 232.100.4.4 100.6.0.1 | begin 232
(100.6.0.1,232.100.4.4),    Flags:
  Up: 00:47:35
  Last Used: 00:00:01
  SW Forwarding Counts: 36/36/3600
  SW Replication Counts: 36/0/0
  SW Failure Counts: 0/0/0/0/0
  mdtRED Flags:  F NS MI, Up:00:47:35
  GigabitEthernet0/0/0/0.100 Flags:  A, Up:00:47:35

R10#show ip mroute vrf RED 232.100.4.4 100.6.0.1 count | begin ^Group
Group: 232.100.4.4, Source count: 3, Packets forwarded: 42, Packets received:
42
  Source: 100.6.0.1/32, Forwarding: 42/0/142/0, Other: 42/0/0

R10#show monitor capture CAP buffer detailed
  0 122      0.000000  00:50:56:A9:F9:61 -> 01:00:5E:64:04:04 VLAN-tagged
frame
  0000:  01005E64 04040050 56A9F961 81000DD4  ..^d...PV..a....
  0010:  81000064 08004500 00640023 0000FD01  ...d..E..d.#....
  0020:  6D066406 0001E864 04040800 139B0008  m.d....d.....
  0030:  00050000 000083F6 E6ABABCD ABCDABCD  .....

```

Additional Reading – Reference configurations "per-vrf-te"

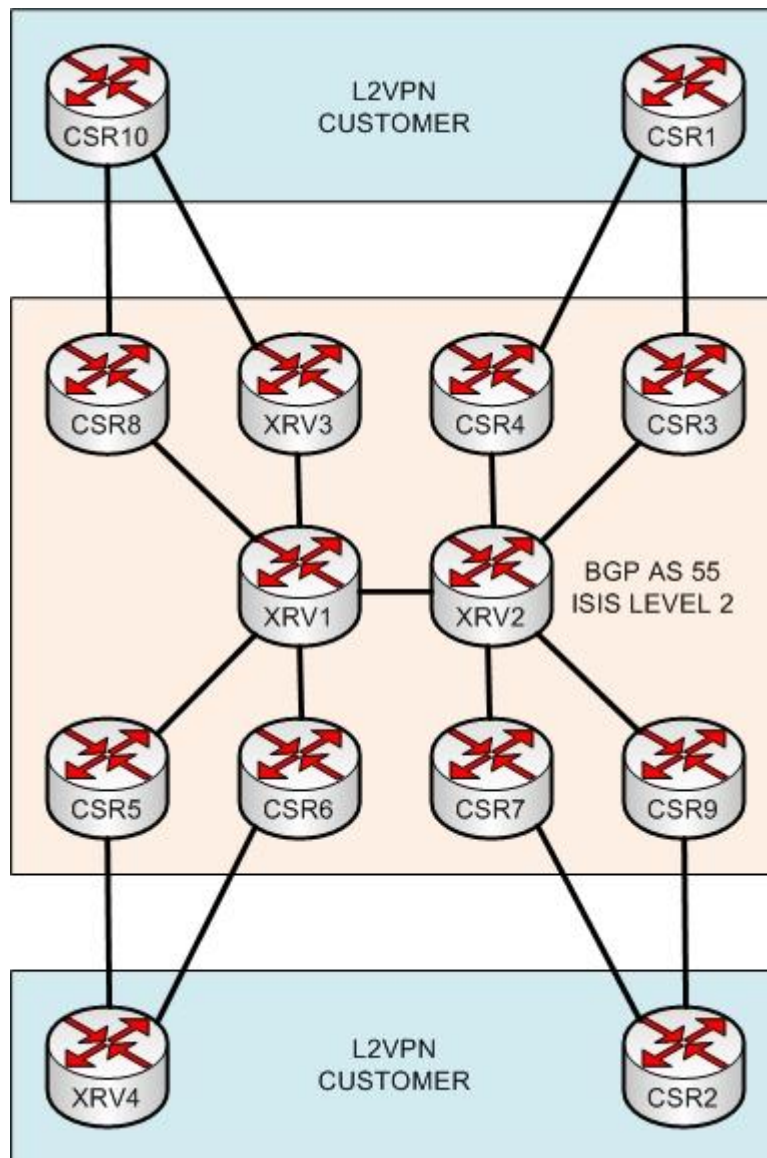
32. Describe, implement, and troubleshoot E-LAN and E-TREE (extended to general L2VPN)

32.1 MPLS encapsulated L2VPN

32.1.1 Static configuration

32.1.1.1 E-LINE (VPWS)

Virtual Private Wire Service (VPWS) is a self-explanatory term that is meant to form P2P links between sites. E-LINE or EoMPLS are subsets of VPWS as they do exactly that with Ethernet connections. There is no MAC address learning or tracking as there is with VPLS (E-LAN and E-TREE). Several pseudowires (PWs) are built for examination. Because there are many PW knobs and different syntaxes to configure them, we will examine a few at the same time to start. A customer requires VPWS between CSR10 and XRv4. There should be some redundancy in the network also to account for either CSR5 or CSR6 failing. We will create two PWs: CSR8-CSR5 and CSR8-CSR6. The PW to CSR5 will be considered primary while the one to CSR6 will be considered secondary. The network diagram is shown below.



Because they will generally have the same configuration and capabilities, we can create a PW template (not a PW class) to create some CLI inheritance. These PWs will be signaled via LDP and will have the control-word (CW) enabled, which is discussed in detail later. They are also MPLS encapsulated and should include sequence numbers to prevent out of order packet delivery, which is a function of the CW discussed later. The primary path also selects a TE tunnel as its preferred path, which grants 2 Mbps of bandwidth. The backup PW follows the LDP LSP. Because the configurations are very basic, we can add many advanced features to the initial configurations.

```
! CSR8
interface Tunnel85
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 55.0.0.5
```

```

tunnel mpls traffic-eng priority 7 7
tunnel mpls traffic-eng bandwidth 2000
tunnel mpls traffic-eng path-option 1 dynamic

template type pseudowire TMP_PW_R10_XRV14
encapsulation mpls
sequencing both
control-word include
signaling protocol ldp

interface pseudowire10141
description PRIMARY PW R10-XRV14
source template type pseudowire TMP_PW_R10_XRV14
neighbor 55.0.0.5 10141
preferred-path interface Tunnel85

interface pseudowire10142
description BACKUP PW R10-XRV14
source template type pseudowire TMP_PW_R10_XRV14
neighbor 55.0.0.6 10142

```

We also must define the access circuit (AC). We will use dot1q-tunneling so that we can reuse VLAN 3580 to carry multiple EVCs from the perspective of CSR10. This will help scale the testing as we add more EVC configuration.

```

! CSR8
interface GigabitEthernet2
service instance 3504 ethernet
encapsulation dot1q 3580 second-dot1q 3504
rewrite ingress tag pop 2 symmetric

```

At this point, the PWs are doing nothing and would not come up even if the other end was configured. The PWs must be bound to an L2VPN Xconnect context, which is the new syntax. Both PWs are added and prioritized within the same group; the group is just a string and is not defined anywhere else. Lower priorities are preferred, but backup PWs can still be signaled ahead of time, similar to path-protection with MPLS-TE. The Ethernet service-instance (access circuit or AC) is also bound here.

```

! CSR8
l2vpn xconnect context XC_R10_XRV14
member pseudowire10141 group R10_XRV14 priority 3
member pseudowire10142 group R10_XRV14 priority 6
member GigabitEthernet2 service-instance 3504

```

Before doing anything, we can quickly verify the L2VPN XC context. We expect that the PWs are down, but the summary view can show us the PW/AC bindings before we continue. The AC also shows down

since the EVC is broken. Notice that despite being down, the backup PW shows that its XC state is inactive as it is still in standby mode.

```
R8#show l2vpn service xconnect name XC_R10_XRV14
Legend: St=State      XC St=State in the L2VPN Service      Prio=Priority
          UP=Up        DN=Down          AD=Admin Down      IA=Inactive
          SB=Standby   HS=Hot Standby    RV=Recovering      NH=No Hardware
          m=manually selected
```

Interface	Group	Encapsulation	Prio	St	XC St
VPWS name: XC_R10_XRV14, State: DN					
pw10141	R10_XRV14	55.0.0.5:10141 (MPLS)	3	DN	DN
pw10142	R10_XRV14	55.0.0.6:10142 (MPLS)	6	DN	IA
Gi2		Gi2:3580/3504 (Eth VLAN)	0	UP	DN

There is no point in verifying anything else on CSR8 since the network is broken. On CSR5, we will configure the PW, AC, and bind them in an L2VPN cross-connect. The configuration is very similar to CSR8. There is no TE tunnel here just for brevity.

```
! CSR5
interface pseudowire10141
 encapsulation mpls
 neighbor 55.0.0.8 10141
 sequencing both
 control-word include
 signaling protocol ldp

interface GigabitEthernet2
 service instance 1014 ethernet
 encapsulation dot1q 3554 second-dot1q 1014
 rewrite ingress tag pop 2 symmetric

l2vpn xconnect context XC_XRV14_R10
 member pseudowire10141
 member GigabitEthernet2 service-instance 1014
```

Checking CSR8 again, we can see that PW and AC came fully up. The confirms that the configuration is correct.

```
R8#show l2vpn service xconnect name XC_R10_XRV14 | begin Interface
```

Interface	Group	Encapsulation	Prio	St	XC St
VPWS name: XC_R10_XRV14, State: UP					
pw10141	R10_XRV14	55.0.0.5:10141 (MPLS)	3	UP	UP
pw10142	R10_XRV14	55.0.0.6:10142 (MPLS)	6	DN	IA
Gi2		Gi2:3580/3504 (Eth VLAN)	0	UP	UP

We can enable debugging on both ends to understand the process while the PW is being signaled. Using this single debug is verbose, so take care when sending it to the console.

```
! CSR5 and CSR8
debug mpls l2transport vc event
```

Next, we clear the PW on CSR8. The AToM process receives the command to UNPROVISION the PW with VCID 10141. It does this by shutting down the PW which is shown below. CSR8 then begins the process of withdrawing the label bindings associated with the PW. It does this for the local label 8000 and also notifies the remote end, CSR5, by sending a label withdraw. The C-bit in this context relates to the presence or absent of the control-word, which is discussed more later.

```
R8#clear l2vpn service peer 55.0.0.5 vcid 10141
```

```
! CSR8
AToM[55.0.0.5, 10141]: Received UNPROVISION, pw10141 [i/f 4009]
AToM[55.0.0.5, 10141]: Set last error: Local access circuit reported shutdown
AToM[55.0.0.5, 10141]: Local not ready
AToM[55.0.0.5, 10141]: Evt local not ready, established->remote ready
AToM[55.0.0.5, 10141]: ... NMS: VC oper state: DOWN
AToM[55.0.0.5, 10141]: ... NMS: err codes: no-err
AToM[55.0.0.5, 10141]: ... SYSLOG: VC is DOWN
AToM[55.0.0.5, 10141]: ... PW MIB: VC state is: DOWN
AToM[55.0.0.5, 10141]: .. PWID 14: unbound from label 8000 (hold time 200
secs)
AToM[55.0.0.5, 10141]: ... Status: DOWN(hard-down, not-ready) (0xC0)
AToM LDP[55.0.0.5]: Sending label withdraw msg vc type 5, cbit 1,
AToM LDP[55.0.0.5]: , vc id 10141,
AToM LDP[55.0.0.5]: group id 0, vc label 8000,
AToM LDP[55.0.0.5]: status 0x7/LDP 0x1/ADJ 0x0,
AToM LDP[55.0.0.5]: mtu 1500, vc handle 0xE2000006
AToM LDP[55.0.0.5]: VCCV CC 0x7, CV 0x2
```

CSR5 receives the label withdrawal from 55.0.0.8 for VCID 10141 bound to remote label 8000. It also frees its local binding since label 5020 is unused now.

```
! CSR5
AToM LDP[55.0.0.8]: Received label withdraw msg, id 739, GR instance 2 vc
type 5, cbit 1,
AToM LDP[55.0.0.8]: , vc id 10141,
AToM LDP[55.0.0.8]: group id 0, vc label 8000,
AToM LDP[55.0.0.8]: status 0x0/LDP 0x0/ADJ 0x0,
AToM LDP[55.0.0.8]: mtu 0, vc handle 0xBC000006
AToM LDP[55.0.0.8]: VCCV CC 0x0, CV 0x0
AToM LDP[55.0.0.8, 10141]: Receive label withdraw
AToM[55.0.0.8, 10141]: Evt remote not ready, established->local ready
```

```

AToM[55.0.0.8, 10141]: . Delete remote vc label binding
ToM[55.0.0.8, 10141]: ... SYSLOG: VC is DOWN, PW Err
AToM[55.0.0.8, 10141]: ... PW MIB: VC state is: DOWN
AToM[55.0.0.8, 10141]: Free local binding
AToM[55.0.0.8, 10141]: . Hold timer not started; free label
AToM[55.0.0.8, 10141]: . PWID 12: unbound from label 5020 (hold time 201
secs)
AToM[55.0.0.8, 10141]: . Local label old state: Advertised, new state: None
AToM[55.0.0.8, 10141]: . Local label old state: None, new state: None
AToM[55.0.0.8, 10141]: Evt ldp down, local ready->local standby, AC-ready

```

On CSR8, the software notifies AToM to provision the PW again. This is the equivalent of “no shutdown” on the PW.

```

! CSR8
AToM[55.0.0.5, 10141]: Received PROVISION, pw10141 [i/f 4009]
AToM[55.0.0.5, 10141]: Circuit attributes, Provisioned:
AToM[55.0.0.5, 10141]: . Status: DOWN(hard-down, not-ready) (0xC0)
AToM[55.0.0.5, 10141]: . Circuit directive: Go Active
AToM[55.0.0.5, 10141]: . Switch handle: 4135
AToM[55.0.0.5, 10141]: . FATPW is not enabled
AToM[55.0.0.5, 10141]: . Process attrs
AToM[55.0.0.5, 10141]: . Evt provision, idle->provisioned
AToM[55.0.0.5, 10141]: .. Provision vc

```

After the LDP session comes up, the two routers can exchange labels. CSR8 allocates local label 8022 and advertises this to CSR5 via the label mapping message. CSR5 receives this remote label, allocates label 5021, and advertises this to CSR8 using the same message.

```

! CSR8
AToM LDP[55.0.0.5]: Sending label mapping msg vc type 5, cbit 1,
AToM LDP[55.0.0.5]: , vc id 10141,
AToM LDP[55.0.0.5]: group id 0, vc label 8022,
AToM LDP[55.0.0.5]: status 0x7/LDP 0x1/ADJ 0x0,
AToM LDP[55.0.0.5]: mtu 1500, vc handle 0xE2000006
AToM LDP[55.0.0.5]: VCCV CC 0x7, CV 0x2

! CSR5
AToM LDP[55.0.0.8]: Received label mapping msg, id 756, GR instance 3 vc type
5, cbit 1,
AToM LDP[55.0.0.8]: , vc id 10141,
AToM LDP[55.0.0.8]: group id 0, vc label 8022,
AToM LDP[55.0.0.8]: status 0x7/LDP 0x1/ADJ 0x0,
AToM LDP[55.0.0.8]: mtu 1500, vc handle 0xBC000006
AToM LDP[55.0.0.8]: VCCV CC 0x7, CV 0x2
AToM LDP[55.0.0.8]: Sending label mapping msg vc type 5, cbit 1,
AToM LDP[55.0.0.8]: , vc id 10141,
AToM LDP[55.0.0.8]: group id 0, vc label 5021,

```

```

AToM LDP[55.0.0.8]:    status 0x7/LDP 0x1/ADJ 0x0,
AToM LDP[55.0.0.8]:    mtu 1500, vc handle 0xBC000006
AToM LDP[55.0.0.8]:    VCCV CC 0x7, CV 0x2

```

! CSR8

```

AToM LDP[55.0.0.5]: Received label mapping msg, id 1233, GR instance 4 vc
type 5, cbit 1,
AToM LDP[55.0.0.5]:    , vc id 10141,
AToM LDP[55.0.0.5]:    group id 0, vc label 5021,
AToM LDP[55.0.0.5]:    status 0x7/LDP 0x1/ADJ 0x0,
AToM LDP[55.0.0.5]:    mtu 1500, vc handle 0xE2000006
AToM LDP[55.0.0.5]:    VCCV CC 0x7, CV 0x2

```

There are even more pages of less-than-useful debug that I've removed, but towards the end of the session, the two routers exchange status updates which detail the characteristics of the PW. This includes the supported VCCV CCs and CVs, support for CW, CW sequencing, and other key parameters. This completes the PW negotiation.

! CSR8

```

AToM[55.0.0.5, 10141]: ..... Send status update
AToM[55.0.0.5, 10141]: ..... Collect circuit attrs
AToM[55.0.0.5, 10141]: ..... Not fast switchover capable, cap FDEF525
AToM[55.0.0.5, 10141]: ..... MSPW: LocCC CW [1], RA [2], TTL [3] Rem CW
[1], RA [2], TTL [3]
AToM[55.0.0.5, 10141]: ..... MSPW: [0] VCID 10141 Loc 55.0.0.8 Rem
55.0.0.5
AToM[55.0.0.5, 10141]: ..... Not group switchover capable, cap FDEF525
AToM[55.0.0.5, 10141]: ..... Circuit attributes, Send update:
AToM[55.0.0.5, 10141]: ..... Interface handle: 0x4009
AToM[55.0.0.5, 10141]: ..... Status: UP (0x1)
AToM[55.0.0.5, 10141]: ..... Payload encap: Ethernet
AToM[55.0.0.5, 10141]: ..... Circuit Encap: Ethernet
AToM[55.0.0.5, 10141]: ..... VCCV CC: 0x7
AToM[55.0.0.5, 10141]: ..... Segment type: AToM
AToM[55.0.0.5, 10141]: ..... Control Word: yes
AToM[55.0.0.5, 10141]: ..... Fast Switchover: no
AToM[55.0.0.5, 10141]: ..... MTU: 1500
AToM[55.0.0.5, 10141]: ..... I/F Str: pw10141
AToM[55.0.0.5, 10141]: ..... Circuit string: 55.0.0.5:10141
AToM[55.0.0.5, 10141]: ..... NMS: VC oper state: DOWN
AToM[55.0.0.5, 10141]: ..... NMS: err codes: no-err
AToM[55.0.0.5, 10141]: ..... SYSLOG: VC is DOWN
AToM[55.0.0.5, 10141]: ..... Check if can activate dataplane
AToM[55.0.0.5, 10141]: ..... Activating dataplane on Active RP
AToM[55.0.0.5, 10141]: ..... Evt dataplane activate, establishing-
>activating
AToM[55.0.0.5, 10141]: ..... Activating data plane
AToM[55.0.0.5, 10141]: ..... Activate dataplane

```

```

AToM[55.0.0.5, 10141]: ..... Need to setup the dataplane
AToM[55.0.0.5, 10141]: ..... Setup dataplane, PWID 14
AToM[55.0.0.5, 10141]: ..... Provision SSM with PWID 14, VC ID 10141,
Block ID 0
AToM[55.0.0.5, 10141]: ..... Set imp flags: cw seq ra vcw
AToM[55.0.0.5, 10141]: ..... : ttl
AToM[55.0.0.5, 10141]: ..... Set segment count to 1
AToM[55.0.0.5, 10141]: ..... Set disp flags: cw seq
AToM[55.0.0.5, 10141]: ..... Provision SSM with 4135/28714 (sw/seg)
AToM: 150 cumulative msgs handled. rc=0
AToM[55.0.0.5, 10141]: Receive SSM dataplane up notification
AToM[55.0.0.5, 10141]: Receive SSM dataplane up notification
AToM[55.0.0.5, 10141]: Evt dataplane up, activating->established
AToM[55.0.0.5, 10141]: . Dataplane activated
AToM[55.0.0.5, 10141]: ... NMS: VC oper state: UP
AToM[55.0.0.5, 10141]: ... SYSLOG: VC is UP
AToM[55.0.0.5, 10141]: ... PW MIB: VC state is: UP

! CSR5
AToM[55.0.0.8, 10141]: ..... Send status update
AToM[55.0.0.8, 10141]: ..... Collect circuit attrs
AToM[55.0.0.8, 10141]: ..... Not fast switchover capable, cap FDEF525
AToM[55.0.0.8, 10141]: ..... MSPW: LocCC CW [1], RA [2], TTL [3] Rem CW
[1], RA [2], TTL [3]
AToM[55.0.0.8, 10141]: ..... MSPW: [0] VCID 10141 Loc 55.0.0.5 Rem
55.0.0.8
AToM[55.0.0.8, 10141]: ..... Not group switchover capable, cap FDEF525
AToM[55.0.0.8, 10141]: ..... Circuit attributes, Send update:
AToM[55.0.0.8, 10141]: ..... Interface handle: 0x4005
AToM[55.0.0.8, 10141]: ..... Status: UP (0x1)
AToM[55.0.0.8, 10141]: ..... Payload encap: Ethernet
AToM[55.0.0.8, 10141]: ..... Circuit Encap: Ethernet
AToM[55.0.0.8, 10141]: ..... VCCV CC: 0x7
AToM[55.0.0.8, 10141]: ..... Segment type: AToM
AToM[55.0.0.8, 10141]: ..... Control Word: yes
[snip, same as CSR8]
AToM[55.0.0.8, 10141]: ... PW MIB: VC state is: UP

```

We can quickly check CSR8 to ensure both PWs are operational. We can look at the MPLS VC output to see this as well. The backup PW is still down since CSR6 does not have any PWs configured yet.

```

R8#show l2vpn service xconnect name XC_R10_XRV14 | begin Interface
  Interface          Group          Encapsulation          Prio   St   XC St
  -----          -
VPWS name: XC_R10_XRV14, State: UP
  pw10141            R10_XRV14    55.0.0.5:10141 (MPLS)   3       UP   UP
  pw10142            R10_XRV14    55.0.0.6:10142 (MPLS)   6       DN   IA
  Gi2                Gi2:3580/3504 (Eth VLAN) 0       UP   UP

```

```
R8#show mpls l2transport vc 10141
```

Local intf	Local circuit	Dest address	VC ID	Status
Gi2	Eth VLAN 3580/3504	55.0.0.5	10141	UP

We will quickly configure the PW on CSR6 before continuing so we can verify it. We will use the legacy syntax on CSR6, which involves creating a PW-class and binding it to an xconnect under the EVC stanza. Any new features introduced by Cisco (such as FAT-PW, etc) will not be supported by the legacy syntax, so it is best to use the newer syntax for future growth.

```
! CSR6
pseudowire-class PW_1014
  encapsulation mpls
  control-word

interface GigabitEthernet2
  service instance 3504 ethernet
  encapsulation dot1q 3546 exact
  rewrite ingress tag pop 1 symmetric
  xconnect 55.0.0.8 10142 encapsulation mpls pw-class PW_1014
```

We quickly verify the PW came up. In this case, it is marked as STANDBY since it is not in use, but this is better than DOWN.

```
R8#show l2vpn service xconnect name XC_R10_XRV14 | begin Interface
```

Interface	Group	Encapsulation	Prio	St	XC	St
-----	-----	-----	----	--	-----	
VPWS name: XC_R10_XRV14, State: UP						
pw10141	R10_XRV14	55.0.0.5:10141 (MPLS)	3	UP	UP	
pw10142	R10_XRV14	55.0.0.6:10142 (MPLS)	6	SB	IA	
Gi2		Gi2:3580/3504 (Eth VLAN)	0	UP	UP	

```
R8#show mpls l2transport vc 10142
```

Local intf	Local circuit	Dest address	VC ID	Status
-----	-----	-----	-----	-----
Gi2	Eth VLAN 3580/3504	55.0.0.6	10142	STANDBY

I've cleared the PWs a few more times to look at other debugs that proved less valuable. For this reason, the exact label bindings are different than what we saw earlier as the sessions were resigned. By looking at the details of each VC, we can see the full label stack that is used. The primary circuit uses PW label 5020 from CSR5 and the backup circuit uses PW label 6000 from CSR6. Also note the preferred path (MPLS TE) for the primary PW only. The control-word was negotiated and sequencing is enabled on both primary/backup circuits. The primary PW also shows "default path" is ready, which is the LDP path to be used if the tunnel fails. In the case of the backup PW, the default path is the active (and only) available path. Notice that there is no traffic flowing over the backup PW, only the primary one.


```

R8#show mpls l2transport vc 10141 detail
Local interface: Gi2 up, line protocol up, Eth VLAN 3580/3504 up
  Interworking type is Ethernet
  Destination address: 55.0.0.5, VC ID: 10141, VC status: up
    Output interface: Tu85, imposed label stack {91009 5020}
    Preferred path: Tunnel85, active
    Default path: ready
  Next hop: point2point
  Create time: 00:47:39, last status change time: 00:29:05
  Last label FSM state change time: 00:29:05
  Signaling protocol: LDP, peer 55.0.0.5:0 up
    Targeted Hello: 55.0.0.8(LDP Id) -> 55.0.0.5, LDP is UP
    Graceful restart: not configured and not enabled
    Non stop routing: not configured and not enabled
    Status TLV support (local/remote) : enabled/supported
      LDP route watch : enabled
      Label/status state machine : established, LruRru
      Last local dataplane status rcvd: No fault
      Last BFD dataplane status rcvd: Not sent
      Last BFD peer monitor status rcvd: No fault
      Last local AC circuit status rcvd: No fault
      Last local AC circuit status sent: No fault
      Last local PW i/f circ status rcvd: No fault
      Last local LDP TLV status sent: No fault
      Last remote LDP TLV status rcvd: No fault
      Last remote LDP ADJ status rcvd: No fault
    MPLS VC labels: local 8000, remote 5020
    Group ID: local 0, remote 0
    MTU: local 1500, remote 1500
    Remote interface description:
    Sequencing: receive enabled, send enabled
    Sequencing resync disabled
    Control Word: On
    SSO Descriptor: 55.0.0.5/10141, local label: 8000
    Dataplane:
      SSM segment/switch IDs: 4126/4124 (used), PWID: 14
    VC statistics:
      transit packet totals: receive 32, send 21
      transit byte totals: receive 2540, send 2644
      transit packet drops: receive 0, seq error 0, send 0

R8#show mpls l2transport vc 10142 detail
Local interface: Gi2 up, line protocol up, Eth VLAN 3580/3504 up
  Interworking type is Ethernet
  Destination address: 55.0.0.6, VC ID: 10142, VC status: standby
    Output interface: Gi2.581, imposed label stack {91007 6000}
    Preferred path: not configured
    Default path: active
  Next hop: 55.8.11.11

```

```

Create time: 00:48:01, last status change time: 01:15:35
  Last label FSM state change time: 00:18:08
Signaling protocol: LDP, peer 55.0.0.6:0 up
  Targeted Hello: 55.0.0.8(LDP Id) -> 55.0.0.6, LDP is UP
  Graceful restart: not configured and not enabled
  Non stop routing: not configured and not enabled
  Status TLV support (local/remote)    : enabled/supported
    LDP route watch                     : enabled
    Label/status state machine          : established, LrdRru
  Last local dataplane status rcvd: No fault
  Last BFD dataplane status rcvd: Not sent
  Last BFD peer monitor status rcvd: No fault
  Last local AC circuit status rcvd: DOWN(standby)
  Last local AC circuit status sent: No fault
  Last local PW i/f circ status rcvd: No fault
  Last local LDP TLV status sent: DOWN(standby)
  Last remote LDP TLV status rcvd: No fault
  Last remote LDP ADJ status rcvd: No fault
  MPLS VC labels: local 8023, remote 6000
  Group ID: local 0, remote 0
  MTU: local 1500, remote 1500
  Remote interface description:
  Sequencing: receive enabled, send enabled
  Sequencing resync disabled
  Control Word: On
  SSO Descriptor: 55.0.0.6/10142, local label: 8023
  Dataplane:
    SSM segment/switch IDs: 45088/24605 (used), PWID: 15
  VC statistics:
    transit packet totals: receive 0, send 0
    transit byte totals:  receive 0, send 0
    transit packet drops: receive 0, seq error 0, send 0

```

Let's find out how those label stacks were built. We can check the MPLS L2VPN bindings which we used for MPLS OAM/VCCV in another section. The local and remote labels per VC are listed here, which match what we saw above. The C-bit represents the CW, which is enabled on both VCs.

```

R8#show mpls l2transport binding 10141
  Destination Address: 55.0.0.5, VC ID: 10141
    Local Label: 8000
      Cbit: 1, VC Type: Ethernet, GroupID: 0
      MTU: 1500, Interface Desc: n/a
      VCCV: CC Type: CW [1], RA [2], TTL [3]
      CV Type: LSPV [2]
    Remote Label: 5020
      Cbit: 1, VC Type: Ethernet, GroupID: 0
      MTU: 1500, Interface Desc: n/a
      VCCV: CC Type: CW [1], RA [2], TTL [3]

```

CV Type: LSPV [2]

```
R8#show mpls l2transport binding 10142
Destination Address: 55.0.0.6, VC ID: 10142
Local Label: 8023
  Cbit: 1, VC Type: Ethernet, GroupID: 0
  MTU: 1500, Interface Desc: n/a
  VCCV: CC Type: CW [1], RA [2], TTL [3]
  CV Type: LSPV [2]
Remote Label: 6000
  Cbit: 1, VC Type: Ethernet, GroupID: 0
  MTU: 1500, Interface Desc: n/a
  VCCV: CC Type: CW [1], RA [2], TTL [3]
  CV Type: LSPV [2]
```

For the transport label, we know the primary PW goes through a TE tunnel, so we can check the outbound label on that tunnel. It shows label 91009, allocated by XRv1, which is correct. The backup PW uses the LDP path via 55.0.0.6/32, which is bound to label 91007. To recap what we saw earlier, the full label stacks are {91009 5020} and {91007 6000} for primary and backup PWs, respectively. The first show command viewing the details of the VC can tell us all of this at once. Remember the LDP label for the backup PW, because we will see this later when testing failover.

```
R8#show mpls traffic-eng tunnels tunnel 85 | include OutLabel
OutLabel : GigabitEthernet2.581, 91009

R8#show mpls ldp bindings 55.0.0.6 32 neighbor 55.0.0.11
lib entry: 55.0.0.6/32, rev 8
remote binding: lsr: 55.0.0.11:0, label: 91007
```

A quick data plane check between XRv4 and CSR10 shows the label stack. XRv4 pings CSR10, and when CSR10 replies, the label stack in use is {91009 5020}. Notice that 0x16381 = 91009 and 0x139C = 5020 and is highlighted in yellow. The CW is shown in green; this is discussed in detail later, but the output shows a 4-byte buffer with some unknown values at present. Be careful not to confuse the CW for an MPLS shim header despite them both being 4 bytes. The IP source/destination addresses are shown in cyan, which shows a flow from 10.10.14.10 to 10.10.14.14.

```
R8#show monitor capture CAP buffer detailed
 6 144 3.027007 00:50:56:A9:FB:1C -> 00:50:56:A9:9C:60 MPLS unicast
0000: 005056A9 9C600050 56A9FB1C 81000DFD .PV..`.PV.....
0010: 88471638 10FF0139 C1FF0000 00190050 .G.8...9.....P
0020: 56A9862A 005056A9 F9610800 45000064 V...*.PV..a..E..d
0030: 00010000 FF018B6C 0A0A0E0A 0A0A0E0E .....l.....
```

We can quickly perform similar verifications on CSR5, which is the other end of the operational PW. There is no TE tunnel on the path back to CSR8, so the LDP default path is active.

```
R5#show mpls l2transport vc 10141
```

Local intf	Local circuit	Dest address	VC ID	Status
Gi2	Eth VLAN 3554/1014	55.0.0.8	10141	UP

```
R5#show mpls l2transport vc 10141 detail
Local interface: Gi2 up, line protocol up, Eth VLAN 3554/1014 up
Interworking type is Ethernet
Destination address: 55.0.0.8, VC ID: 10141, VC status: up
Output interface: Gi2.551, imposed label stack {91008 8000}
Preferred path: not configured
Default path: active
Next hop: 55.5.11.11
[snip]
```

The VC label bindings are identical to those seen at CSR8 except reversed. CSR5 will use PW label 8000 when sending traffic along VC 10141.

```
R5#show mpls l2transport binding 10141
Destination Address: 55.0.0.8, VC ID: 10141
Local Label: 5020
  Cbit: 1, VC Type: Ethernet, GroupID: 0
  MTU: 1500, Interface Desc: n/a
  VCCV: CC Type: CW [1], RA [2], TTL [3]
  CV Type: LSPV [2]
Remote Label: 8000
  Cbit: 1, VC Type: Ethernet, GroupID: 0
  MTU: 1500, Interface Desc: n/a
  VCCV: CC Type: CW [1], RA [2], TTL [3]
  CV Type: LSPV [2]
```

The transport label is 91008, signaled from XRv1 using LDP. As seen earlier, the entire label stack is {91008 8000}.

```
R5#show mpls ldp bindings 55.0.0.8 32 neighbor 55.0.0.11
lib entry: 55.0.0.8/32, rev 23
remote binding: lsr: 55.0.0.11:0, label: 91008
```

Another useful command involves looking at a summary view of allocated labels and PWs. This shows label 5020 locally allocated for VC 10141.

```
R5#show mpls l2transport pwid | include 10141|Label
```

Label	Peer-Address	VCID	PWID	In-Use	FirstUse	ReusedAt	FreedAt
5020	55.0.0.8	10141	12	Yes	13:42:08	Never	Never

Just like earlier, we verify the data plane by capturing outbound on CSR5. We see 0x16380 = 91008 and 0x1F40 = 8000, which matches the expected label stack. The CW is shown in green, and the IP source and destination addresses are 10.01.014.14 and 10.10.14.10, respectively.

```
R5#show monitor capture CAP buffer detailed
  1 144      0.938007 00:50:56:A9:DC:63 -> 00:50:56:A9:9C:60 MPLS unicast
0000: 005056A9 9C600050 56A9DC63 81000DDF .PV..`.PV..c....
0010: 88471638 00FF01F4 01FF0000 001C0050 .G.8.....P
0020: 56A9F961 005056A9 862A0800 45000064 V..a.PV...*..E..d
0030: 00000000 FF018B6D 0A0A0E0E 0A0A0E0A .....m.....
```

We will begin testing the failover capability of this design. First, we will “break” the TE tunnel. Rather than outright destroy it, we will more elegantly pre-empt it with another TE tunnel. This is a more realistic reason for an already-signaled TE tunnel to fail. A high priority tunnel will request all available bandwidth (150 Mbps) on the link to XRv1. We aren’t using this tunnel for anything else other than to cause Tunnel 85 to fail, so we don’t have to use a steering mechanism.

```
! CSR8
interface Tunnel999
 description TAKE ALL BANDWIDTH
 ip unnumbered Loopback0
 tunnel mode mpls traffic-eng
 tunnel destination 55.0.0.11
 tunnel mpls traffic-eng priority 0 0
 tunnel mpls traffic-eng bandwidth 150000
 tunnel mpls traffic-eng path-option 1 dynamic
```

We see log messages on CSR8 that indicate tunnel ID 85 cannot remain operational as the downstream bandwidth request cannot be fulfilled. The VC now falls back to its default path, as the tunnel cannot be used. The default path moves from “ready” to “active” when the preferred path is unavailable.

```
! CSR8
TE-LM-BW: tunnel 55.0.0.8_8->55.0.0.5_85 {7}: Downstream bw hold request
failed
%LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel85, changed state to
down
```

```
R8#show mpls l2transport vc 10141 detail | include Pref
Preferred path: Tunnel85, no route
Default path: active
```

CSR8 needs to figure out what transport label to use, and it gets this from LDP via XRv1. The route to the PW endpoint is IGP learned, so we should see an LDP label from XRv1.

```
R8#show mpls ldp bindings 55.0.0.5 32 neighbor 55.0.0.11
 lib entry: 55.0.0.5/32, rev 10
```

```
remote binding: lsr: 55.0.0.11:0, label: 91006
```

Testing the data plane outbound from CSR8 again, we can see the LDP transport label in use, yielding a label stack of {91006 5020}, shown in yellow. Notice that $0x1637E = 91006$ and $0x139C = 5020$. We are still not using the backup PW since the default path of the primary PW is still preferred over the backup PW entirely. The xconnect doesn't care about whether the preferred TE path is up or not so long as the primary PW can remain up.

```
R8#show monitor capture CAP buffer detailed
  2 144      2.143989 00:50:56:A9:FB:1C -> 00:50:56:A9:9C:60 MPLS unicast
0000: 005056A9 9C600050 56A9FB1C 81000DFD .PV..`.PV.....
0010: 88471637 E0FF0139 C1FF0000 001F0050 .G.7...9.....P
0020: 56A9862A 005056A9 F9610800 45000064 V..*.PV..a..E..d
0030: 00000000 FF018B6D 0A0A0E0A 0A0A0E0E .....m.....
```

We must force the primary PW down entirely before the backup PW can be used. Rather than do an unrealistic test and shutdown the PW interface, we will perform some LDP label filtering. CSR8 will no longer accept bindings for 55.0.0.5/32 from XRv1, which breaks the FEC entirely, assuming the TE tunnel is not used. The LDP binding database, FIB, and LFIB show no label.

```
! CSR8
mpls ldp neighbor 55.0.0.11 labels accept ACL_FILTER_R5_LABEL
ip access-list standard ACL_FILTER_R5_LABEL
deny 55.0.0.5
permit any
```

```
R8#show mpls ldp bindings 55.0.0.5 32 neighbor 55.0.0.11
[no output]
```

```
R8#show ip cef 55.0.0.5
55.0.0.5/32
  nexthop 55.8.11.11 GigabitEthernet2.581
```

```
R8#show mpls forwarding-table 55.0.0.5 32
Local      Outgoing    Prefix          Bytes Label    Outgoing    Next Hop
Label      Label        or Tunnel Id    Switched       interface
8016       No Label     55.0.0.5/32     0              Gi2.581     55.8.11.11
```

This will force the primary PW to fail. Now the backup PW goes from STANDBY to UP.

```
R8#show mpls l2transport vc 10141
Local intf  Local circuit          Dest address    VC ID    Status
-----
Gi2         Eth VLAN 3580/3504     55.0.0.5       10141    DOWN
```

```
R8#show mpls l2transport vc 10142
Local intf  Local circuit          Dest address    VC ID    Status
```

```

-----
Gi2                Eth VLAN 3580/3504                55.0.0.6                10142                UP

R8#show l2vpn service xc name XC_R10_XRV14 | begin Interface
  Interface          Group          Encapsulation          Prio   St   XC St
  -----          -
VPWS name: XC_R10_XRV14, State: UP
  pw10141            R10_XRV14      55.0.0.5:10141 (MPLS)    3      DN   IA
  pw10142            R10_XRV14      55.0.0.6:10142 (MPLS)    6      UP   UP
  Gi2                Gi2:3580/3504 (Eth VLAN)  0      UP   UP

```

Because the layer 2 features on XRv are unsupported in the data-plane, I cannot put its uplinks into a bridge-domain and apply an IP address to a BVI. Instead, I will quickly switch interfaces on XRv14 for testing to use backup PW. This will allow XRv4 to send traffic to CSR10, but I acknowledge that it is sloppy and unrealistic.

```

! XRv4
interface GigabitEthernet0/0/0/0.546
  description LINE TO CSR10 VIA CSR6 (BACKUP)
  no shutdown

interface GigabitEthernet0/0/0/0.1014
  description LINE TO CSR10 VIA CSR6 (PRIMARY)
  shutdown

```

Testing the data plane outbound on CSR8, we can see the LDP label for CSR6's loopback is 91007 (0x1637F) which we saw earlier. The PW label is 0x1770, which is 6000, and is what we saw in the PW verifications earlier as well.

```

R8#show monitor capture CAP buffer detailed
  2 144      2.432021 00:50:56:A9:FB:1C -> 00:50:56:A9:9C:60 MPLS unicast
0000: 005056A9 9C600050 56A9FB1C 81000DFD .PV..`.PV.....
0010: 88471637 F0FF0177 01FF0000 00090050 .G.7...w.....P
0020: 56A9862A 005056A9 F9610800 45000064 V...*.PV..a..E..d
0030: 00000000 FF018B6D 0A0A0E0A 0A0A0E0E .....m.....

```

We can verify this from CSR6's perspective also, which is identical except for using the legacy configuration syntax. We can see the label stack quickly, then verify the components. The PW label 8023 was advertised from CSR8 via targeted LDP, and the transport label 91008 was advertised from XRv1 via LDP.

```

R6#show mpls l2transport vc 10142 detail | include label_stack
  Output interface: Gi2.561, imposed label stack {91008 8023}

R6#show mpls l2transport binding 10142
  Destination Address: 55.0.0.8, VC ID: 10142

```

```

Local Label: 6000
  Cbit: 1,    VC Type: Ethernet,    GroupID: 0
  MTU: 1500,  Interface Desc: n/a
  VCCV: CC Type: CW [1], RA [2], TTL [3]
  CV Type: LSPV [2]

```

```

Remote Label: 8023
  Cbit: 1,    VC Type: Ethernet,    GroupID: 0
  MTU: 1500,  Interface Desc: n/a
  VCCV: CC Type: CW [1], RA [2], TTL [3]
  CV Type: LSPV [2]

```

```

R6#show mpls ldp bindings 55.0.0.8 32 neighbor 55.0.0.11
  lib entry: 55.0.0.8/32, rev 23
  remote binding: lsr: 55.0.0.11:0, label: 91008

```

Measuring outbound on CSR6 shows us the correct label stack (0x16380 = 91008, 0x1F57 = 8023) in the label packets going across the backup PW.

```

R6#show monitor capture CAP buffer detailed
  5  144    2.288046  00:50:56:A9:DE:0D -> 00:50:56:A9:9C:60 MPLS unicast
0000:  005056A9 9C600050 56A9DE0D 81000DE9  .PV..`.PV.....
0010:  88471638 00FF01F5 71FF0000 00000050  .G.8....q.....P
0020:  56A9F961 005056A9 862A0800 45000064  V..a.PV...*.E..d
0030:  00020000 FF018B6B 0A0A0E0E 0A0A0E0A  .....k.....

```

We can recover the primary PW by making any of its paths available again: the TE tunnel or the LDP path. In this case, we will leave the LDP filter in place and simply shutdown the greedy tunnel 999. Tunnel 85 comes back up, which allows the primary PW to come up. This automatically puts the backup PW into standby mode.

```

R5#show mpls l2transport vc 10141
Local intf    Local circuit    Dest address    VC ID    Status
-----
Gi2           Eth VLAN 3554/1014  55.0.0.8       10141    UP

```

```

R6#show mpls l2transport vc 10142
Local intf    Local circuit    Dest address    VC ID    Status
-----
Gi2           Eth VLAN 3546    55.0.0.8       10142    STANDBY

```

```

R8#show l2vpn service xconnect name XC_R10_XRV14 | begin Interface
Interface      Group      Encapsulation      Prio  St  XC St
-----
VPWS name: XC_R10_XRV14, State: UP
pw10141        R10_XRV14  55.0.0.5:10141 (MPLS)  3     UP  UP
pw10142        R10_XRV14  55.0.0.6:10142 (MPLS)  6     SB  IA
Gi2            Gi2:3580/3504 (Eth VLAN)  0     UP  UP

```


We still do not have a default path, but since the preferred path is available, the primary PW can still operate. In a real network, you would likely only observe this case when LDP was not running at all; that is to say, TE would be the primary transport between all PEs.

```
R8#show mpls l2transport vc 10141 detail | include path:
  Preferred path: Tunnel85, active
  Default path:
```

A final control-plane check confirms that the new RSVP-TE label and PW label are used for this VC.

```
R8#show ip rsvp reservation detail filter session-type 7 tunnel-id 85 |
include Label
  Label: 91009 (outgoing)
```

```
R8#show mpls l2transport binding 10141 | include Remote
  Remote Label: 5021
```

```
R8#show mpls l2transport vc 10141 detail | include label_stack
  Output interface: Tu85, imposed label stack {91009 5021}
```

Removing the LDP filter facing XRv1 brings back the default path, also. This returns the network to its normal state.

```
R8#show mpls l2transport vc 10141 detail | include path:
  Preferred path: Tunnel85, active
  Default path: ready
```

To complete the customer network, we will provision an entirely new PW to connect CSR10 with CSR1. We will configure this manually with no signaling protocol; this removes LDP from the process. This comes with several drawbacks, to include a lack of status updates, along with several features like CW sequencing and PW redundancy. Not using LDP means the local and remote label bindings must also be defined manually per PW. When we try to do this on CSR8 under a PW interface, we cannot set a meaningful local label.

```
R8(config-if)#label ?
<0-0> Local pseudowire label
```

The reason is because you must first identify a range of static labels in the configuration. After making this adjustment, we verify the new range is applied. Then, we can configure a local label in accordance with that range.

```
R8(config)#mpls label range 8000 8999 static 800 899

R8#show mpls label range
Downstream Generic label region: Min/Max label: 8000/8999
```

Range for static labels: Min/Max label: 800/899

```
R8(config-if)#label ?  
<800-899> Local pseudowire label
```

The full configuration on CSR8 is as follows. The CW is enabled by default, and does work with static PWs, but I have disabled it for variety. We should see the Ethernet frames immediately after the PW labels in the packet captures now without the additional 4 bytes added by the CW.

```
! CSR8  
interface pseudowire110  
  description PW R10-R1  
  encapsulation mpls  
  neighbor 55.0.0.4 110  
  control-word exclude  
  signaling protocol none  
  label 810 410  
  pseudowire type 5  
  no status  
  
interface GigabitEthernet2  
  service instance 3510 ethernet  
  encapsulation dot1q 3580 second-dot1q 3510  
  rewrite ingress tag pop 2 symmetric  
  
l2vpn xconnect context XC_R10_R1  
  member pseudowire110  
  member GigabitEthernet2 service-instance 3510
```

We will use the legacy configuration on CSR4 but the concept is the same. We also must specify static MPLS label range on CSR4 (not shown), or else we see the issue below again.

```
R4(cfg-if-ether-vc-xconn)#mpls label ?  
<0-0> Local pseudowire label  
  
! CSR4  
pseudowire-class PW_R10_R1  
  encapsulation mpls  
  protocol none  
  no status  
  
interface GigabitEthernet2  
  service instance 3514 ethernet  
  encapsulation dot1q 3514 exact  
  rewrite ingress tag pop 1 symmetric  
  xconnect 55.0.0.8 110 encapsulation mpls manual pw-class PW_R10_R1  
  mpls label 410 810  
  no mpls control-word
```

The normal control-plane verifications are limited now because the PWs will always show UP unless the target FEC (no route to PW endpoint, no corresponding label, etc) is broken. For variety, we will use the “new” syntax for verification in this section.

```
R4#show l2vpn atom vc vcid 110
```

Interface	Peer ID	VC ID	Service		Status
			Type	Name	
pw100007	55.0.0.8	110	p2p	Gi2-SI:3514:3514	UP

```
R8#show l2vpn atom vc vcid 110
```

Interface	Peer ID	VC ID	Service		Status
			Type	Name	
pw110	55.0.0.4	110	p2p	XC_R10_R1	UP

We can see the label stacks as well. The PW labels were statically allocated while the transport labels came from LDP. Due to the static nature of manually-configured PWs, the administrator must ensure the local/remote labels are configured correctly on both sides.

```
R4#show l2vpn atom vc vcid 110 detail | include label_stack
Output interface: Gi2.542, imposed label stack {92008 810}
```

```
R8#show l2vpn atom vc vcid 110 detail | include label_stack
Output interface: Gi2.581, imposed label stack {91003 410}
```

We can see the VCCV bindings from the above command also, assuming we don’t use a filter. The legacy command didn’t show VCCV information, but the new command does. We will check it from one side, as this information is “assumed” since it cannot be signaled. It is interesting to note that VCCV says Type-1 CC using the control-word is supported, but it actually isn’t.

```
R4#show l2vpn atom vc vcid 110 detail | section VCCV
VCCV CV type 0x02                                0x02
          LSPV [2]                                LSPV [2]
VCCV CC type 0x07                                0x07
          CW [1], RA [2], TTL [3]                CW [1], RA [2], TTL [3]
```

A quick VCCV operation confirms that the PW is operational. It is valuable to check both directions since LSPs are always unidirectional.

```
R4#ping mpls pseudowire 55.0.0.8 110
Sending 5, 72-byte MPLS Echos to 55.0.0.8,
    timeout is 2 seconds, send interval is 0 msec:
[snip]
Type escape sequence to abort.
```

!!!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 4/5/8 ms

```
R8#ping mpls pseudowire 55.0.0.4 110
Sending 5, 72-byte MPLS Echos to 55.0.0.4,
    timeout is 2 seconds, send interval is 0 msec:
[snip]
Type escape sequence to abort.
```

!!!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 4/5/10 ms

Sending traffic from CSR1 to CSR10, and monitoring outbound on CSR4, yields the output below. The labels 0x16768 and 0x32A translate into decimal 92008 and 810 respectively, which is the correct label stack for VCID 110. Notice there is no control word following the MPLS label stack. The IP addresses inside the L2VPN are highlighted in green, which reveals a flow from 10.1.10.1 to 10.1.10.10.

```
R4#show monitor capture CAP buffer detailed
  2 140      0.268984 00:50:56:A9:2C:57 -> 00:50:56:A9:0E:6F MPLS unicast
0000: 005056A9 0E6F0050 56A92C57 81000DD6 .PV..o.PV.,W....
0010: 88471676 80FF0032 A1FF0050 56A9F961 .G.v...2...PV..a
0020: 005056A9 1AAA0800 45000064 00AC0000 .PV.....E..d....
0030: FF0192E0 0A010A01 0A010A0A 08009010 .....
```

We will enable RIPv2 on all three customer routers to ensure XRv4 can communicate with CSR1 across CSR10. Running IGP for customers is the biggest driver for L2VPN as it simplifies their enterprise networking. The L2VPN isn't great because traffic hairpins through CSR10, but it does work. A direct connection from XRv4 to CSR1 would make more sense to create optimal traffic flows. The RIP configuration is not shown because it is very basic and is not the focus of this test.

```
R1#show ip route vrf LINE 10.10.14.0
Routing Table: LINE
Routing entry for 10.10.14.0/24
  Known via "rip", distance 120, metric 1
  Redistributing via rip
  Last update from 10.1.10.10 on GigabitEthernet2.514, 00:00:18 ago
  Routing Descriptor Blocks:
  * 10.1.10.10, from 10.1.10.10, 00:00:18 ago, via GigabitEthernet2.514
    Route metric is 1, traffic share count is 1
```

```
RP/0/0/CPU0:XRv4#show route vrf LINE ipv4 10.1.10.0
Routing entry for 10.1.10.0/24
  Known via "rip", distance 120, metric 1
  Routing Descriptor Blocks
  10.10.14.10, from 10.10.14.10, via GigabitEthernet0/0/0/0.1014
    Route metric is 1
  No advertising protos.
```

When XRv4 sends traffic to CSR1, we capture both in and out on CSR8's interface facing the core. There are four relevant packets to see in this test; the initial request entering and existing CSR8, and the reply doing the same in the opposite direction. First, the initial request arrives on PW 10141 with a single label in the stack (yellow) followed by the CW (green) The output below shows that this label is 8011 (0x1F4B). The packet size is 840 bytes.

```
R8#show l2vpn atom binding 10141 | include Local
```

```
Local Label: 8011
```

```
R8#show monitor capture CAP buffer detailed
```

```
10 840 4.247988 00:50:56:A9:9C:60 -> 00:50:56:A9:FB:1C MPLS unicast
0000: 005056A9 FB1C0050 56A99C60 81000DFD .PV....PV..`....
0010: 884701F4 B1FE0000 002F0050 56A9F961 .G...../.PV..a
0020: 005056A9 862A0800 45000320 00000000 .PV..*..E.. ....
0030: FF018CC3 0A0A0E0E 0A010A01 08002B32 .....+2
```

Next, the packet is routed to CSR10, then hairpins back into the MPLS core via CSR8 as the PE. Notice the length is the same despite having two labels. This is because the PW between CSR8 and CSR4 is not using the CW, but the PW between CSR5 and CSR8 is. Labels 91003 (0x1637B) and 410 (19A) are pushed onto the stack as expected.

```
R8#show l2vpn atom binding 110 | include Remote
```

```
Remote Label: 410
```

```
R8#show ip cef 55.0.0.4
```

```
55.0.0.4/32
```

```
nexthop 55.8.11.11 GigabitEthernet2.581 label 91003
```

```
R8#show monitor capture CAP buffer detailed
```

```
11 840 4.247988 00:50:56:A9:FB:1C -> 00:50:56:A9:9C:60 MPLS unicast
0000: 005056A9 9C600050 56A9FB1C 81000DFD .PV..`.PV.....
0010: 88471637 B0FF0019 A1FF0050 56A91AAA .G.7.....PV...
0020: 005056A9 F9610800 45000320 00000000 .PV..a..E.. ....
0030: FE018DC3 0A0A0E0E 0A010A01 08002B32 .....+2
```

Traffic traverses the network, ultimately reaching CSR4 where all labels are removed. CSR1 responds to the ICMP echo, and CSR4 sends the reply back towards CS8. Traffic arrives from CSR4 along PW 110. We see this packet is 4 bytes smaller as it contains only the PW label (XRv1 performs PHP) and there is no CW. The incoming label is 810 (0x32A) which is bound to PW 110, sending traffic to CSR10.

```
R8#show l2vpn atom binding 110 | include Local
```

```
Local Label: 810
```

```
R8#show mpls forwarding-table labels 810
```

Local	Outgoing	Prefix	Bytes	Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched		interface	

```
810      No Label      12ckt(1)      11798      Gi2      point2point
```

```
R8#show monitor capture CAP buffer detailed
 12 836      4.252977 00:50:56:A9:9C:60 -> 00:50:56:A9:FB:1C MPLS unicast
0000: 005056A9 FB1C0050 56A99C60 81000DFD .PV....PV..`....
0010: 88470032 A1FD0050 56A9F961 005056A9 .G.2...PV..a.PV.
0020: 1AAA0800 45000320 00000000 FF018CC3 ....E.. .....
0030: 0A010A01 0A0A0E0E 00003332 C0B00000 .....32....
```

Last, we see our largest packet yet. It contains two MPLS labels {91009 5021}, derived from 0x16381 and 0x139D respectively, and represents the primary PW towards XRv4 via CSR5. Remember that the transport label comes from RSVP-TE as the outgoing interface is a TE tunnel. The CW (green) is included as well, which is why the packet is 844 bytes.

```
R8#show l2vpn atom vc vcid 10141 detail | include label_stack
Output interface: Tu85, imposed label stack {91009 5021}
```

```
R8#show monitor capture CAP buffer detailed
 13 844      4.264985 00:50:56:A9:FB:1C -> 00:50:56:A9:9C:60 MPLS unicast
0000: 005056A9 9C600050 56A9FB1C 81000DFD .PV..`.PV.....
0010: 88471638 10FF0139 D1FF0000 008E0050 .G.8...9.....P
0020: 56A9862A 005056A9 F9610800 45000320 V..*.PV..a..E..
0030: 00000000 FE018DC3 0A010A01 0A0A0E0E .....
```

32.1.1.2 Advanced PW features (CW, Status, etc)

With basic connectivity achieved, we can now explore additional PW features. This section is a continuation from the section and uses the same three PWs already built. While simple to configure, the control-word (CW) has many critical details to understand. It 32 bit (4 byte) field inserted after the PW label, so it is the same length as an ordinary MPLS shim header. In the previous set of examples, we saw the CW in the EPC captures but did not verify its operation nor purpose. The CW is capable of doing five things:

1. **Pad small packets:** Depending on the media type, padding may be needed to meet the minimum frame size when using different media. If the length field in the CW is nonzero at the egress PE, it knows padding was applied and will remove it. Typically the length field is zero as padding frames is uncommon with Ethernet due to header sizes.
2. **Carry control bits for certain layer 2 protocols:** Mostly relevant for ATM and FR, where techniques like FECN, BECN, DE, EFCI, and CLP are needed. The CW is required for some layer 2 protocols (ATM, FR) and optional for others (Ethernet, PPP).
3. **Preserve the sequence of frames:** Like sequence numbers in TCP, this assigns incremental values to each unit using a 16 bit unsigned counter (1 to 65535). The command "sequencing both" enables it for transmitted and received packets, and it is disabled by default. Static PWs

(without LDP) cannot use this feature. Out of sequence frames are not retransmitted, but are simply dropped. The counter goes from 1 to 65535 and wraps around to 1 again (circular).

4. **Facilitate load sharing:** Cisco routers examine the first 4 bits after the MPLS header to determine the payload. For IPv4, this is 0x4, and IOS will perform load sharing (if the FIB indicates so) based on the source and destination addresses. For Ethernet frames, this first hexadecimal digit could be a 4 also depending on the destination MAC address. This could invalidate load sharing strategies as the router will try to parse an IPv4 source/destination address pair from the Ethernet frame. The same is true for IPv6 except Cisco looks for a 0x6. CWs added to the layer 2 frames will begin with either 0x0 (general data) or 0x1 (OAM). By seeing values that are neither 0x4 nor 0x6, the router assumes the frame is non-IP and performs load-sharing based on the bottom label, which is correct. Coupled with FAT-PW, this technique works well for core routers that are unaware of the MPLS service details.
5. **Facilitate fragmentation/reassembly:** Using Beginning and Ending (B and E) bits, AToM frames can be fragmented and reassembled for transit over MPLS. P routers cannot fragment AToM packets.

CWs that begin with bits 0000 represent ordinary data packets. Those that begin with bits 0001 represent PW-ACH (VCCV) test frames. The MPLS OAM section shows VCCV CWs and they always begin with 0x1; this is just a way to multiplex control traffic onto the same “channel” (FEC) as the data traffic. The data frame seen in captures in the previous chapter starts with 0x0. The remaining 28 bits vary based on what is being encapsulated, but we will focus on Ethernet. The next 12 bits are reserved for future use (RFC 4448) and are typically zero. Encapsulations that require the CW will use these bits for things like fragmentation bits, the length field, and other signaling flags. The last 16 bits of all CWs represent the sequence numbers.

To test the sequence numbers, we send traffic from CSR10 to XRv4 using the primary PW where the CW is already enabled. Sequence numbers are enabled bidirectionally from the previous example. Capturing in both directions on CSR8, we can see the sequence numbers increasing as packets are received and sent. The packets from CSR8 to CSR5 are shown in yellow and the return traffic from CSR5 to CSR8 is shown in cyan. Only the CW is highlighted. The “yellow” flow counts 53, 54, 55 and the “cyan” flow counts 15, 16, 17.

```
R8#show monitor capture CAP buffer detailed
 3 744      1.136986 00:50:56:A9:FB:1C -> 00:50:56:A9:9C:60 MPLS unicast
0000: 005056A9 9C600050 56A9FB1C 81000DFD .PV..`.PV.....
0010: 88471638 20FF0139 31FF0000 00530050 .G.8 ..91....S.P
0020: 56A9862A 005056A9 F9610800 450002BC V...*.PV...a..E...
0030: 002D0000 FF0188E8 0A0A0E0A 0A0A0E0E .-.....

 4 740      1.139992 00:50:56:A9:9C:60 -> 00:50:56:A9:FB:1C MPLS unicast
0000: 005056A9 FB1C0050 56A99C60 81000DFD .PV....PV..`.....
```

```

0010: 884701F4 B1FE0000 00150050 56A9F961 .G.....PV..a
0020: 005056A9 862A0800 450002BC 002D0000 .PV..*..E....-..
0030: FF0188E8 0A0A0E0E 0A0A0E0A 00009B4D .....M

5 744 1.141991 00:50:56:A9:FB:1C -> 00:50:56:A9:9C:60 MPLS unicast
0000: 005056A9 9C600050 56A9FB1C 81000DFD .PV..`.PV.....
0010: 88471638 20FF0139 31FF0000 00540050 .G.8 ..91....T.P
0020: 56A9862A 005056A9 F9610800 450002BC V..*.PV..a..E...
0030: 002E0000 FF0188E7 0A0A0E0A 0A0A0E0E .....

6 740 1.206990 00:50:56:A9:9C:60 -> 00:50:56:A9:FB:1C MPLS unicast
0000: 005056A9 FB1C0050 56A99C60 81000DFD .PV....PV..`....
0010: 884701F4 B1FE0000 00160050 56A9F961 .G.....PV..a
0020: 005056A9 862A0800 450002BC 002E0000 .PV..*..E.....
0030: FF0188E7 0A0A0E0E 0A0A0E0A 00009B46 .....F

7 744 1.208989 00:50:56:A9:FB:1C -> 00:50:56:A9:9C:60 MPLS unicast
0000: 005056A9 9C600050 56A9FB1C 81000DFD .PV..`.PV.....
0010: 88471638 20FF0139 31FF0000 00550050 .G.8 ..91....U.P
0020: 56A9862A 005056A9 F9610800 450002BC V..*.PV..a..E...
0030: 002F0000 FF0188E6 0A0A0E0A 0A0A0E0E ./.....

8 740 1.324980 00:50:56:A9:9C:60 -> 00:50:56:A9:FB:1C MPLS unicast
0000: 005056A9 FB1C0050 56A99C60 81000DFD .PV....PV..`....
0010: 884701F4 B1FE0000 00170050 56A9F961 .G.....PV..a
0020: 005056A9 862A0800 450002BC 002F0000 .PV..*..E..../..
0030: FF0188E6 0A0A0E0E 0A0A0E0A 00009B01 .....

```

It makes sense to use sequencing in both directions, but it can also be configured unidirectionally. In this example, CSR8 sends sequence numbers and CSR5 processes them upon receipt, but this is not enabled in the reverse direction. On CSR8, this specific PW setting overrides the inherited template. The “yellow” CWs show traffic from CSR8 with sequence numbers 10 and 11, while the “cyan” CWs show the return traffic from CSR5 with no sequencing.

```

! CSR8
interface pseudowire10141
 sequencing transmit

R8#show l2vpn atom vc vcid 10141 detail | include Seq
Sequencing: receive disabled, send enabled
Sequencing resync disabled

! CSR5
interface pseudowire10141
 sequencing receive

R5#show l2vpn atom vc vcid 10141 detail | include Seq
Sequencing: receive enabled, send disabled

```


Sequencing resync disabled

```
R8#show monitor capture CAP buffer detailed
 4  744      0.877005  00:50:56:A9:FB:1C -> 00:50:56:A9:9C:60 MPLS unicast
0000:  005056A9 9C600050 56A9FB1C 81000DFD  .PV...`.PV.....
0010:  88471638 20FF0139 E1FF0000 00100050  .G.8 ..9.....P
0020:  56A9862A 005056A9 F9610800 450002BC  V...*.PV..a..E...
0030:  00320000 FF0188E3 0A0A0E0A 0A0A0E0E  .2.....

 5  740      0.880011  00:50:56:A9:9C:60 -> 00:50:56:A9:FB:1C MPLS unicast
0000:  005056A9 FB1C0050 56A99C60 81000DFD  .PV....PV..`....
0010:  884701F5 81FE0000 00000050 56A9F961  .G.....PV..a
0020:  005056A9 862A0800 450002BC 00320000  .PV...*.E....2..
0030:  FF0188E3 0A0A0E0E 0A0A0E0A 0000915D  .....]

 6  744      0.881003  00:50:56:A9:FB:1C -> 00:50:56:A9:9C:60 MPLS unicast
0000:  005056A9 9C600050 56A9FB1C 81000DFD  .PV...`.PV.....
0010:  88471638 20FF0139 E1FF0000 00110050  .G.8 ..9.....P
0020:  56A9862A 005056A9 F9610800 450002BC  V...*.PV..a..E...
0030:  00330000 FF0188E2 0A0A0E0A 0A0A0E0E  .3.....

 7  740      0.947009  00:50:56:A9:9C:60 -> 00:50:56:A9:FB:1C MPLS unicast
0000:  005056A9 FB1C0050 56A99C60 81000DFD  .PV....PV..`....
0010:  884701F5 81FE0000 00000050 56A9F961  .G.....PV..a
0020:  005056A9 862A0800 450002BC 00330000  .PV...*.E....3..
0030:  FF0188E2 0A0A0E0E 0A0A0E0A 00009156  .....V
```

If the sender and receiver PEs remain unsynchronized for long periods of time, this could cause outages since out-of-order frames are simply dropped. On CSR5 and CSR8, we specify that if 500 packets are seen out of order, the sequence numbers are reset. This is like a “reset” button that allows the PW to recover from an variety of forwarding issues in the core. We won’t try to test this in the data plane as it would be very difficult.

```
! CSR5 and CSR8
interface pseudowire10141
 sequencing resync 500
```

```
R8#show l2vpn atom vc vcid 10141 detail | include Seq
Sequencing: receive disabled, send enabled
Sequencing resync enabled, threshold 500
```

```
R5#show l2vpn atom vc vcid 10141 detail | include Seq
Sequencing: receive enabled, send disabled
Sequencing resync enabled, threshold 500
```

The resync configuration has no effect on CSR8 at present since it is only adding sequence numbers for the benefit of CSR5 and not processing them locally. That is to say, the resync feature on matters when

sequence numbers are actually being processed by the egress LER, so “sequencing transmit” does not count. We will return the PW to its baseline by sequencing in both directions. We can manually clear the sequence numbers using the “clear mpls counters” command.

```
R8#show l2vpn atom vc vcid 10141 detail | include Seq
Sequencing: receive enabled, send enabled
Sequencing resync enabled, threshold 500
```

```
R8#clear mpls counters
Clear "show mpls forwarding-table" counters [confirm]
mpls forward counters cleared
```

```
R5#show l2vpn atom vc vcid 10141 detail | include Seq
Sequencing: receive enabled, send enabled
Sequencing resync enabled, threshold 500
```

```
R5#clear mpls counters
Clear "show mpls forwarding-table" counters [confirm]
mpls forward counters cleared
```

Another important concept is the C-bit, or the control word bit. It is set to 1 if the CW is present on a PW. The PW ID FEC element carried by LDP will carry this value (along with the PW ID and other generic but important fields). For encapsulations that require the CW, such as ATM and FR, the C-bit should always be 1. The mechanics of how the C-bit is negotiated is very detailed, but in summary, if the CW is optional (Ethernet) and the two endpoints disagree on whether the CW should be used or not, the PW will come up with CW disabled. Notice that the primary/backup PWs from CSR8 use the CW, while the manual PW to CSR4 does not. This determines whether or not the CW will appear in the MPLS packets.

```
R8#show l2vpn atom binding 10141 | include Cbit
Cbit: 1,      VC Type: Ethernet,      GroupID: 0
Cbit: 1,      VC Type: Ethernet,      GroupID: 0
```

```
R8#show l2vpn atom binding 10142 | include Cbit
Cbit: 1,      VC Type: Ethernet,      GroupID: 0
Cbit: 1,      VC Type: Ethernet,      GroupID: 0
```

```
R8#show l2vpn atom binding 110 | include Cbit
Cbit: 0,      VC Type: Ethernet,      GroupID: 0
Cbit: 0,      VC Type: Ethernet,      GroupID: 0
```

LDP-based PWs can also use the concept of “status” to exchange information about the link with one another. This includes the status of the local AC where the remote end has no visibility. Status is enabled by default, and we can verify it on the LDP PWs from CSR8. The first line of the output is most important; it shows whether status is enabled locally and supported by the remote end. The state machine codes are straightforward: L/R means local/remote router, r/n means ready or not-ready, u/d means up/down. In this case, both local and remote nodes are ready and up, so status is working.

```
R8#show l2vpn atom vc vcid 10141 detail | section Status TLV
```

```
Status TLV support (local/remote)      : enabled/supported
LDP route watch                        : enabled
Label/status state machine             : established, LruRru
Local dataplane status received        : No fault
BFD dataplane status received          : Not sent
BFD peer monitor status received       : No fault
Status received from access circuit    : No fault
Status sent to access circuit          : No fault
Status received from pseudowire i/f    : No fault
Status sent to network peer            : No fault
Status received from network peer      : No fault
Adjacency status of remote peer        : No fault
Status TLV      enabled                supported
```

We can simulate a fault on CSR5 by shutting down the Ethernet flow point (EFP) on CSR5. This means that, from CSR8's perspective, the remote AC is down. This is a good reason to start using the backup PW to CSR6 and is one reason why status is critical when using backup PWs. The debugs below show this happen in real-time, but they are quite chatty. CSR8 receives the status message of DOWN AC from the remote node and updates its state machine accordingly. The remote node is ready, but down, as indicated by "LruRrd".

```
! CSR8
debug l2vpn atom vc status event
debug l2vpn atom vc status fsm
```

```
AToM[55.0.0.5, 10141]: Received remote peer status
AToM[55.0.0.5, 10141]: . Remote status DOWN AC(rx/tx faults)
AToM[55.0.0.5, 10141]: . Remote status change, DOWN AC(rx/tx faults)
AToM[55.0.0.5, 10141]: .. Check if remote connected
AToM[55.0.0.5, 10141]: ... Remote label is ready
AToM[55.0.0.5, 10141]: ..... S:Evt remote ready in LruRru
AToM[55.0.0.5, 10141]: ..... S:Evt remote down, LruRru->LruRrd
AToM[55.0.0.5, 10141]: ..... S:Act send SSS(DOWN)
AToM[55.0.0.5, 10141]: ..... Remote      : DOWN AC(rx/tx faults)
AToM[55.0.0.5, 10141]: ..... Overall    : DOWN AC(rx/tx faults)
```

CSR8 also sets its own DOWN AC status and sends it to CSR5, probably to ensure the PW goes completely down. This might be a software optimization or requirement to get the backup PW working correctly, but we know the local AC is fine. The state becomes "LrdRrd" as the AC is "down" at both ends.

```
! CSR8
AToM[55.0.0.5, 10141]: ..... Update AC for status change from UP
AToM[55.0.0.5, 10141]: ..... Send AC STATUS(DOWN AC(rx/tx faults))
AToM[55.0.0.5, 10141]: ..... S:Evt ADJ up in LruRrd
```

```

AToM[55.0.0.5, 10141]: ..... S:Act send SSS(DOWN), notify(UP)
AToM[55.0.0.5, 10141]: ..... Remote      : DOWN AC(rx/tx faults)
AToM[55.0.0.5, 10141]: ..... Overall    : DOWN AC(rx/tx faults)
AToM[55.0.0.5, 10141]: ..... [filtered AC]
AToM[55.0.0.5, 10141]: ..... Send notify(UP)
AToM[55.0.0.5, 10141]: ..... Overall    : No fault
AToM[55.0.0.5, 10141]: ..... [filtered LDP]
AToM[55.0.0.5, 10141]: ... Yes, connected
AToM[55.0.0.5, 10141]: .. keeping dataplane up
AToM[55.0.0.5, 10141]: .. sync new remote status to standby
AToM[55.0.0.5, 10141]: .. Receive AC STATUS(DOWN(standby))
AToM[55.0.0.5, 10141]: ... AC status DOWN(standby)
AToM[55.0.0.5, 10141]: ..... S:Evt local down, LruRrd->LrdRrd

```

Even though CSR8's AC technically isn't down, CSR8 reports it that way, but the reason issue is highlighted below. CSR5 notified CSR8 that its AC was down so the entire PW fails.

```

R8#show l2vpn atom vc vcid 10141 detail | section Status TLV
  Status TLV support (local/remote)      : enabled/supported
    LDP route watch                      : enabled
    Label/status state machine            : established, LrdRrd
    Local dataplane status received       : No fault
    BFD dataplane status received         : Not sent
    BFD peer monitor status received      : No fault
    Status received from access circuit   : DOWN(standby)
    Status sent to access circuit         : DOWN AC(rx/tx faults)
    Status received from pseudowire i/f   : No fault
    Status sent to network peer           : DOWN(standby)
    Status received from network peer     : DOWN AC(rx/tx faults)
    Adjacency status of remote peer       : No fault
  Status TLV   enabled                  supported

```

We should verify that the backup PW has been activated as a result of this failure notification. This is the expected behavior.

```

R8#show l2vpn service xconnect name XC_R10_XRV14 | begin Interface
  Interface      Group      Encapsulation      Prio  St  XC St
  -----
VPWS name: XC_R10_XRV14, State: UP
  pw10141        R10_XRV14    55.0.0.5:10141 (MPLS)  3      DN  IA
  pw10142        R10_XRV14    55.0.0.6:10142 (MPLS)  6      UP  UP
  Gi2            Gi2:3580/3504 (Eth VLAN)  0      UP  UP

```

Bringing the AC back up on CSR5 triggers an UP AC status message, which brings the PW back up and puts the backup PW into standby again.

```

! CSR8
AToM[55.0.0.5, 10141]: Received remote peer status

```

```

AToM[55.0.0.5, 10141]: . Remote status UP
AToM[55.0.0.5, 10141]: . Remote status change, UP
AToM[55.0.0.5, 10141]: .. Check if remote connected
AToM[55.0.0.5, 10141]: ... Remote label is ready
AToM[55.0.0.5, 10141]: ..... S:Evt remote ready in LrdRrd
AToM[55.0.0.5, 10141]: ..... S:Evt remote up, LrdRrd->LrdRru
[snip]

```

```

R8#show l2vpn atom vc vcid 10141 detail | section Status TLV
  Status TLV support (local/remote)      : enabled/supported
    LDP route watch                      : enabled
    Label/status state machine           : established, LruRru
  Local dataplane status received        : No fault
  BFD dataplane status received          : Not sent
  BFD peer monitor status received       : No fault
  Status received from access circuit    : No fault
  Status sent to access circuit          : No fault
  Status received from pseudowire i/f    : No fault
  Status sent to network peer            : No fault
  Status received from network peer      : No fault
  Adjacency status of remote peer        : No fault
  Status TLV enabled                    : supported

```

```
R8#show l2vpn atom vc vcid 10141
```

Interface	Peer ID	VC ID	Service		Status
			Type	Name	
pw10141	55.0.0.5	10141	p2p	XC_R10_XRV14	UP

```
R8#show l2vpn atom vc vcid 10142
```

Interface	Peer ID	VC ID	Service		Status
			Type	Name	
pw10142	55.0.0.6	10142	p2p	XC_R10_XRV14	STANDBY

There are sub-features of “status” also. One of them is the route-watch feature which is enabled by default when status is enabled. This listens for routing events to trigger redundancy changes. The command is sparsely documented so we will research it here.

```

R8#show l2vpn atom vc vcid 10141 detail | include route
  LDP route watch                      : enabled

```

We will disable this on CSR8’s primary PW. Status in general is still enabled, but route-watch is not. The command was made for cross-OS support, but even when demonstrating RIB changes, nothing is different besides the show command. I believe this feature is specifically meant to support the “status redundancy” preferential forwarding treatment available only with ATM ACs.

```

! CSR8
interface pseudowire10141
 no status control-plane route-watch

R8#show l2vpn atom vc vcid 10141 detail | include Status TLV|route
  Status TLV support (local/remote)      : enabled/supported
    LDP route watch                      : disabled
  Status TLV      enabled                  supported

```

Of note are the statuses of the backup PW endpoints. CSR8 reports that it is locally down and communicates this to CSR6. CSR6 acknowledges that it is locally up, but has been placed in STANDBY mode by CSR8. CSR8 “received” notification from the AC that it was in standby, which triggers the appropriate status message to CSR6. The reason CSR8 is down is because the xconnect to which this PW is bound already has a functioning, primary PW.

```

R8#show l2vpn atom vc vcid 10142 detail | section Status TLV
  Status TLV support (local/remote)      : enabled/supported
    LDP route watch                      : enabled
  Label/status state machine              : established, LrdRru
  Local dataplane status received         : No fault
  BFD dataplane status received           : Not sent
  BFD peer monitor status received        : No fault
  Status received from access circuit      : DOWN(standby)
  Status sent to access circuit            : No fault
  Status received from pseudowire i/f     : No fault
  Status sent to network peer              : DOWN(standby)
  Status received from network peer       : No fault
  Adjacency status of remote peer         : No fault
  Status TLV      enabled                  supported

```

```

R6#show l2vpn atom vc vcid 10142 detail | section Status TLV
  Status TLV support (local/remote)      : enabled/supported
    LDP route watch                      : enabled
  Label/status state machine              : established, LruRrd
  Local dataplane status received         : No fault
  BFD dataplane status received           : Not sent
  BFD peer monitor status received        : No fault
  Status received from access circuit      : No fault
  Status sent to access circuit            : DOWN(standby)
  Status received from pseudowire i/f     : No fault
  Status sent to network peer              : No fault
  Status received from network peer       : DOWN(standby)
  Adjacency status of remote peer         : No fault
  Status TLV      enabled                  supported

```

Because status is an LDP TLV, it is not available on manually configured (static) PWs. The state machine is always UP and there is no mechanism for status messaging to be exchanged.

```

R8#show l2vpn atom vc vcid 110 detail | section Status TLV
Status TLV support (local/remote) : disabled/N/A
LDP route watch : enabled
Label/status state machine : established, LruRru
Local dataplane status received : No fault
BFD dataplane status received : Not sent
BFD peer monitor status received : No fault
Status received from access circuit : No fault
Status sent to access circuit : No fault
Status received from pseudowire i/f : No fault
Status sent to network peer : No status
Status received from network peer : Not sent
Adjacency status of remote peer : No fault
Status TLV disabled N/A

```

To ensure certain advanced features are supported on certain platforms (in certain roles), you can check the capabilities. The list is very long and covers all supported layer 2 encapsulations, but I filter based on Ethernet. Notice that, for example, sequence checking isn't supported by either P or PE roles. This may be a hardware limitation of the virtual routers since the parser accepts all sequencing commands and counts the numbers correctly as verified earlier.

```

R4#show l2vpn atom hw-capability interface Gig2 | section Eth
Interface GigabitEthernet2
Transport type Eth VLAN
Core functionality:
MPLS label disposition supported
Control word processing supported
Sequence number processing not supported
VCCV CC Type CW [1] processing supported
Edge functionality:
MPLS label imposition supported
Control word processing supported
Sequence number processing not supported
Fast backup switchover is not supported
Group switchover is not supported
EFP default is supported
EFP range is supported
Imposition rewrite is supported
Disposition rewrite is supported
QinQ is supported
QinQ any is supported
Transport type Ethernet
Core functionality:
MPLS label disposition supported
Control word processing supported
Sequence number processing not supported
VCCV CC Type CW [1] processing supported

```

Edge functionality:

- MPLS label imposition supported
- Control word processing supported
- Sequence number processing not supported
- Fast backup switchover is not supported
- Group switchover is not supported
- EFP default is supported
- EFP range is supported
- Imposition rewrite is supported
- Disposition rewrite is supported
- QinQ is supported
- QinQ any is supported

Layer 2 protocol tunneling is commonly used on switches to transport specific layer 2 frames such as PAgP, LACP, VTP, BPDU, CDP, VTP, and others. There are two other options as well: forward and peer. Forward means to pass the traffic onward with no local processing, where tunneling implies some kind of encapsulation/MAC address changes. Peer implies the router will process the traffic locally and not forward it on. On the CSR1000v, only certain protocols are supported for certain actions.

“Tunnel” feature does not appear supported on the CSR1000v for any traffic type.

```
R4(config-if-srv)#l2protocol tunnel stp
l2protocol action not supported
```

```
R4(config-if-srv)#l2protocol tunnel lacp
l2protocol action not supported
```

```
R4(config-if-srv)#l2protocol tunnel pagp
l2protocol action not supported
```

“Forward” and “peer” features are only supported for STP.

```
R4(config-if-srv)#l2protocol forward stp
```

```
R4(config-if-srv)#l2protocol forward lacp
l2protocol action not supported
```

```
R4(config-if-srv)#l2protocol forward pagp
l2protocol action not supported
```

```
R4(config-if-srv)#l2protocol peer stp
```

```
R4(config-if-srv)#l2protocol peer lacp
l2protocol action not supported
```

```
R4(config-if-srv)#l2protocol peer pagp
l2protocol action not supported
```


I assume unspecified protocols, such as VTP, BPDU, and CDP are processed as “peer”. Enabling CDP on CSR2 and CSR7 (not shown) suggests this. CSR2 and CSR7 are CDP neighbors despite CSR2 not being explicitly configured to treat CDP messages as “peer” on the EFP.

```
R7#show cdp neighbors | begin Device
Device ID      Local Intrfce  Holdtme    Capability  Platform  Port ID
R2             Gig 2         151        R I        CSR1000V  Gig 2.527
```

Last, we examine logging. The syntax is very straightforward and we specify whether to log all PW events or just redundancy events. Even though this is part of the “new” syntax, it works for legacy PW configurations also.

```
! All XE PE routers
l2vpn
 logging pseudowire status
 logging redundancy

! XRv3
l2vpn
 logging
  pseudowire
```

If we clear the static PW towards CSR4, we see many syslog messages. Notice there are no LDP messages since LDP is not used for signaling this PW.

```
R8#clear l2vpn service xconnect name XC_R10_R1
%XCONNECT-5-PW_STATUS: MPLS peer 55.0.0.4 vcid 110, VC state DOWN
%XCONNECT-5-PW_STATUS: MPLS peer 55.0.0.4 vcid 110, VC state DOWN, PW Err
%XCONNECT-5-PW_STATUS: MPLS peer 55.0.0.4 vcid 110, VC state DOWN, PW Err
%XCONNECT-5-PW_STATUS: MPLS peer 55.0.0.4 vcid 110, VC state ADMIN DOWN
%XCONNECT-5-PW_STATUS: MPLS peer 55.0.0.4 vcid 110, VC state DOWN
%XCONNECT-5-PW_STATUS: MPLS peer 55.0.0.4 vcid 110, VC state UP
```

To cause a redundancy event, we can shut down the remote EFP bound to the primary PW on CSR5. CSR8 shows the redundancy failover logs.

```
! CSR8
%XCONNECT-5-REDUNDANCY: XC VPWS: Activating secondary member 55.0.0.6:10142
%XCONNECT-5-PW_STATUS: MPLS peer 55.0.0.5 vcid 10141, VC state DOWN, Rem AC
Err
%XCONNECT-5-PW_STATUS: MPLS peer 55.0.0.6 vcid 10142, VC state UP
```

Bringing the AC on the primary PW back up activates the primary member again. These logs are very helpful for troubleshooting L2VPN and should be enabled at all times.

```
! CSR8
```

```
%XCONNECT-5-REDUNDANCY: XC VPWS: Activating primary member 55.0.0.5:10141
%XCONNECT-5-PW_STATUS: MPLS peer 55.0.0.5 vcid 10141, VC state STANDBY
%XCONNECT-5-PW_STATUS: MPLS peer 55.0.0.6 vcid 10142, VC state STANDBY
%XCONNECT-5-PW_STATUS: MPLS peer 55.0.0.5 vcid 10141, VC state UP
```

32.1.1.3 E-LAN and E-TREE (VPLS)

The Metro Ethernet Forum (MEF) defines MP2MP and P2MP Ethernet services as well. E-LAN provides a traditional LAN service to customers, allowing them to control all aspects of their layer 3 routing protocols between multiple sites. This can be useful for removing routing complexity from customers and allowing them to run non-IP protocols over an SP network, and in this case, an MPLS-based SP network. Virtual Private LAN Service (VPLS) is generally synonymous with E-LAN and is similar to VPWS/E-LINE in many ways. It still relies on the PW construct and all of the associated features, such as CW, status, LDP signaling, etc. However, since there are additional PWs to multiple nodes (not just P2P), there needs to be some kind of “glue” to bond these connections into a single VPLS instance. This is what the virtual forwarding instance (VFI) solves. PWs are identified as “members” of a VFI, and this VFI serves as a single layer 2 interface for the VPLS instance. Then, the VFI is connected to the AC via a bridge-domain, which is similar to integrating routing and bridging (IRB) on the IOS XE/XR platforms.

The goal is to connect CSR1, CSR2, XRv4, and CSR10 together via VPLS. The PEs will be CSR4, CSR6, CSR7, and XRv3. The relevant VFI configurations are shown below; PW configurations are all basic and only serve to achieve basic connectivity between the nodes. Since each PW is independent, their configuration can vary, so long as the encapsulation is the same. I use a mix of new (l2vpn vfi context) and legacy (l2 vfi) configurations to achieve this. I also mix up the manner in which PWs are bound to the VFI, sometimes creating a separate interface PW, sometimes defining it within the VFI context. The AC configuration is not shown as it is identical in logic to VPWS/E-LINE. CSR4 uses the new L2VPN VFI with some in-line PWs, CSR6 uses all “new” syntax, and CSR7 uses strictly legacy syntax. For brevity, I do not show the configurations of each individual PW interface as the interface enumeration reveals the endpoints.

```
! CSR4
l2vpn vfi context LAN
  vpn id 3467
  member 55.0.0.13 413 encapsulation mpls
  member pseudowire46
  member pseudowire47
bridge-domain 3467
  member GigabitEthernet2 service-instance 3541
  member vfi LAN
```

```
! CSR6
l2vpn vfi context LAN
  vpn id 3467
  member pseudowire67
  member pseudowire46
  member pseudowire613
```

```

bridge-domain 3467
  member GigabitEthernet2 service-instance 3564
  member vfi LAN

! CSR7
l2 vfi LAN manual
  vpn id 3467
  bridge-domain 3467
  neighbor 55.0.0.4 47 pw-class PW_VPLS
  neighbor 55.0.0.13 713 pw-class PW_VPLS
  neighbor 55.0.0.6 67 pw-class PW_VPLS

interface GigabitEthernet2
  service instance 3527 ethernet
  bridge-domain 3467

! XRv3
l2vpn
  bridge group LAN
  bridge-domain LAN
  interface GigabitEthernet0/0/0/0.503
  vfi LAN
  vpn-id 3467
  neighbor 55.0.0.4 pw-id 413
  neighbor 55.0.0.6 pw-id 613
  neighbor 55.0.0.7 pw-id 713

```

Strictly speaking, the VPN ID does not matter when doing manually-configured VPLS, but I configure it out of habit. Clearly, this mechanism scales poorly since (N-1) PWs are required for connectivity between all nodes. Before checking specific VPLS parameters, we can check to ensure the VCs are up on all nodes. We will use a brief command on XRv; notice that the AC never comes up. This is a limitation of the XRv data plane, so testing reachability to CSR10 via the E-LAN is not possible. It is beneficial to see the PWs come up in the control-plane so that the VPLS instance can be verified with show commands, at a minimum.

```

R4#show l2vpn atom vc | include LAN
pw46      55.0.0.6      46      vfi      LAN      UP
pw47      55.0.0.7      47      vfi      LAN      UP
pw100007  55.0.0.13     413     vfi      LAN      UP

R6#show l2vpn atom vc | include LAN
pw46      55.0.0.4      46      vfi      LAN      UP
pw67      55.0.0.7      67      vfi      LAN      UP
pw613     55.0.0.13     613     vfi      LAN      UP

R7#show l2vpn atom vc | include LAN
pw100009  55.0.0.4      47      vfi      LAN      UP
pw100006  55.0.0.6      67      vfi      LAN      UP

```

```
pw100008 55.0.0.13 713 vfi LAN UP
```

```
RP/0/0/CPU0:XRv3#show l2vpn bridge-domain bd-name LAN brief
```

Legend: pp = Partially Programmed.

Bridge Group:Bridge-Domain Name	ID	State	Num ACs/up	Num PWs/up
LAN:LAN	0	up	1/0	3/3

The bridge-domain is like a VLAN in that it maintains separate CAM entries per bridged instance. In this case, we are using it for VPLS. From CSR4's perspective, there are 4 "ports" on this virtual switch. 3 of them are VPLS PWs and one is the AC, which is an EFP with service ID 3541. The aging time of 300 seconds (5 minutes) begins counting down when a MAC address is learned and resets to 300 each time a new frame is received with the MAC address in question. Notice that, like a normal switch, MAC learning is enabled by default. AED stands for "authoritative edge device" which is used for overlay transport virtualization (OTV), which is discussed later. It has no relevance for MPLS L2VPNs.

```
R4#show bridge-domain
```

```
Bridge-domain 3467 (4 ports in all)
```

```
State: UP Mac learning: Enabled
```

```
Aging-Timer: 300 second(s)
```

```
GigabitEthernet2 service instance 3541
```

```
vfi LAN neighbor 55.0.0.7 47
```

```
vfi LAN neighbor 55.0.0.6 46
```

```
vfi LAN neighbor 55.0.0.13 413
```

AED	MAC address	Policy	Tag	Age	Pseudoport
0	0050.56A9.1AAA	forward	dynamic	279	GigabitEthernet2.EFP3541
0	0050.56A9.862A	forward	dynamic	278	LAN.1004019
1	FFFF.FFFF.FFFF	flood	static	0	OLIST_PTR:0xe808b800
0	0050.56A9.BE8A	forward	dynamic	279	LAN.1004018

To confirm these are the correct MAC addresses, we can check the ARP cache on CSR1 to verify the IP addresses match each destination. The address ending in 1AAA is local to CSR4 and is identified as reachable via the EFP. Addresses BE8A and 862A are for CSR2 and XRv4 respectively, and are reachable over VPLS via the VFI. The pseudoport numbers for each remote peer are not terribly significant and do not appear well documented. CSR10's MAC address does not show up because XRv3 cannot forward L2VPN traffic.

```
R1#show arp vrf LAN
```

Protocol	Address	Age (min)	Hardware Addr	Type	Interface
Internet	10.0.0.1	-	0050.56a9.1aaa	ARPA	GigabitEth2.541
Internet	10.0.0.2	176	0050.56a9.be8a	ARPA	GigabitEth2.541
Internet	10.0.0.14	49	0050.56a9.862a	ARPA	GigabitEth2.541

We can also get a summary view of all the PWs within a VFI along with the MPLS labels used to build the VPLS instance using a single command. Notice that the VFI itself is given a PW interface for internal software management purposes, although we know it isn't actually a PW.

```

R4#show l2vpn service vfi name LAN detail | begin Interface
Interface          Group          Encapsulation          Prio   St   XC St
-----
VPLS name: LAN, State: UP
pw100008
pw100007            core_pw      55.0.0.13:413 (MPLS)    0      UP  UP
                  Local VC label 4010
                  Remote VC label 93009

pw46                core_pw      55.0.0.6:46 (MPLS)     0      UP  UP
                  Local VC label 4013
                  Remote VC label 6007

pw47                core_pw      55.0.0.7:47 (MPLS)     0      UP  UP
                  Local VC label 4012
                  Remote VC label 7033

```

Checking the LFIB, each PW gets its own set of local and remote labels, since the data plane treats these as independent paths. The purpose of the VFI/bridge-domain combination is to add some switch-like intelligence. The MAC address learning happens just like ordinary an switch; unknown unicast flooding occurs first until bridge-domain data structures update their CAM tables. The first entry with local label 410 is from an E-LINE PW to CSR8 which is from the last section. The VPLS-bound PWs do not have outgoing interfaces in the LFIB since received traffic is delivered to an EFP. Label 4010 maps to the XRv3 PW and, not surprisingly, has received no data traffic.

```

R4#show mpls forwarding-table | include l2ckt
410      No Label   l2ckt (1)      302670      Gi2         point2point
4010     No Label   l2ckt (6)      0           none        point2point
4012     No Label   l2ckt (4)      2580        none        point2point
4013     No Label   l2ckt (5)      2072        none        point2point

```

A quick check of the CW and status of the VPLS PWs can help us understand what the actual encapsulations will look like. We can ignore the output with 55.0.0.8 as the target as this is the PW not part of the VPLS instance. Notice that status is enabled for all VPLS PWs but the CW was not negotiated with XRv3. Locally, CSR4 is configured to attempt CW negotiation, but we can assume XRv3 did not offer it. We can correct that later.

```

R4#show l2vpn atom vc detail | include Control|Status_TLV|Destination
Destination address: 55.0.0.6 VC ID: 46
Status TLV support (local/remote)      : enabled/supported
Control word on (configured: autosense) on
Status TLV enabled                      supported
Destination address: 55.0.0.7 VC ID: 47
Status TLV support (local/remote)      : enabled/supported
Control word on (configured: autosense) on

```

```

Status TLV    enabled                                supported
Destination address: 55.0.0.8 VC ID: 110
Status TLV support (local/remote)                  : disabled/N/A
Control word off (configured: autosense)            off
Status TLV    disabled                              N/A
Destination address: 55.0.0.13 VC ID: 413
Status TLV support (local/remote)                  : enabled/supported
Control word off (configured: autosense)            off
Status TLV    enabled                                supported

```

On CSR1, we will send some IP broadcasts to 255.255.255.255. This translates to a destination MAC address of FFFF.FFFF.FFFF. We see earlier that the bridge-domain (BD) CAM table suggests this traffic should be flooded to all ports in the BD. We expect to see three replicated MPLS packets sent from CSR4 towards the CSR6, CSR7, and XRv3. The first packet is shown below with its MPLS labels in yellow and CW in green. With label stack {92002 6007}, we know this is for CSR6 because the output earlier showed label 6007 along that PW. We will verify this again, along with the LDP label. The original IP source/destination of 10.0.0.1/255.255.255.255 is shown in cyan for completeness.

```

R4#show monitor capture CAP buffer detailed
 0 144      0.000000 00:50:56:A9:2C:57 -> 00:50:56:A9:0E:6F MPLS unicast
0000: 005056A9 0E6F0050 56A92C57 8100DD6 .PV..o.PV.,W....
0010: 88471676 20FF0177 71FF0000 0000FFFF .G.v ..wq.....
0020: FFFFFFFF 005056A9 1AAA0800 45000064 .....PV.....E..d
0030: 00510000 FF01B147 0A000001 FFFFFFFF .Q.....G.....

```

```

R4#show l2vpn atom binding 46 | include Remote
Remote Label: 6007

```

```

R4#show mpls ldp bindings 55.0.0.6 32 neighbor 55.0.0.12
lib entry: 55.0.0.6/32, rev 16
remote binding: lsr: 55.0.0.12:0, label: 92002

```

```

R4#show l2vpn atom vc vcid 46 detail | include label_stack
Output interface: Gi2.542, imposed label stack {92002 6007}

```

For brevity, the last two packets are shown below. Notice the timestamp for all packets is the same. The last packet lacks a CW, so we speculate this is the packet towards XRv3. We confirm the label stacks quickly.

```

R4#show monitor capture CAP buffer detailed
 1 144      0.000000 00:50:56:A9:2C:57 -> 00:50:56:A9:0E:6F MPLS unicast
0000: 005056A9 0E6F0050 56A92C57 8100DD6 .PV..o.PV.,W....
0010: 88471676 10FF01B7 91FF0000 0000FFFF .G.v.....
0020: FFFFFFFF 005056A9 1AAA0800 45000064 .....PV.....E..d
0030: 00510000 FF01B147 0A000001 FFFFFFFF .Q.....G.....

```

```

R4#show l2vpn atom vc vcid 47 detail | include label_stack

```

Output interface: Gi2.542, imposed label stack {92001 7033}

R4#show monitor capture CAP buffer detailed

```
2 140 0.000000 00:50:56:A9:2C:57 -> 00:50:56:A9:0E:6F MPLS unicast
0000: 005056A9 0E6F0050 56A92C57 81000DD6 .PV..o.PV.,W....
0010: 88471676 70FF16B5 11FFFFFF FFFFFFFF .G.vp.....
0020: 005056A9 1AAA0800 45000064 00510000 .PV.....E..d.Q..
0030: FF01B147 0A000001 FFFFFFFF 08002482 ...G.....$.
```

R4#show l2vpn atom vc vcid 413 detail | include label_stack

Output interface: Gi2.542, imposed label stack {92007 93009}

Next, we will ensure VPLS performs unknown unicast flooding, then desists once a MAC address is learned. With a cleared BD CAM, we send traffic from CSR1 to XRv4. Before traffic is sent, CSR4's BD has no entry for XRv4's MAC address.

R4#show bridge-domain 3467 | include 862[Aa]

[no output]

This packet should be replicated along all three PWs initially. Upon receiving return traffic from XRv4, it will learn XRv4's MAC via the proper PW and cease the flooding. The first three packets below all have the same IP header checksum (yellow), which indicates it is the same packet replicated three times. The timestamp is also identical. The pink label stack represents the LSP to CSR6, the green represents the LSP to CSR7, and the cyan represents the LSP to XRv3. The last two packets are traveling along the correct LSP with no more flooding; notice the change in checksum and the increased timestamp. The LSP is fixed using CSR6 which is correct.

R4#show monitor capture CAP buffer detailed

```
1 144 0.669964 00:50:56:A9:2C:57 -> 00:50:56:A9:0E:6F MPLS unicast
0000: 005056A9 0E6F0050 56A92C57 81000DD6 .PV..o.PV.,W....
0010: 88471676 20FF0177 71FF0000 00000050 .G.v ..wq.....P
0020: 56A9862A 005056A9 1AAA0800 45000064 V..*.PV.....E..d
0030: 00560000 FF01A734 0A000001 0A00000E .V.....4.....
```

```
2 144 0.669964 00:50:56:A9:2C:57 -> 00:50:56:A9:0E:6F MPLS unicast
0000: 005056A9 0E6F0050 56A92C57 81000DD6 .PV..o.PV.,W....
0010: 88471676 10FF01B7 91FF0000 00000050 .G.v.....P
0020: 56A9862A 005056A9 1AAA0800 45000064 V..*.PV.....E..d
0030: 00560000 FF01A734 0A000001 0A00000E .V.....4.....
```

```
3 140 0.669964 00:50:56:A9:2C:57 -> 00:50:56:A9:0E:6F MPLS unicast
0000: 005056A9 0E6F0050 56A92C57 81000DD6 .PV..o.PV.,W....
0010: 88471676 70FF16B5 11FF0050 56A9862A .G.vp.....PV..*
0020: 005056A9 1AAA0800 45000064 00560000 .PV.....E..d.V..
0030: FF01A734 0A000001 0A00000E 08000088 ...4.....
```

```
4 144 0.678966 00:50:56:A9:2C:57 -> 00:50:56:A9:0E:6F MPLS unicast
```

```

0000: 005056A9 0E6F0050 56A92C57 81000DD6 .PV..o.PV.,W....
0010: 88471676 20FF0177 71FF0000 00000050 .G.v ..wq.....P
0020: 56A9862A 005056A9 1AAA0800 45000064 V...*.PV.....E..d
0030: 00570000 FF01A733 0A000001 0A00000E .W.....3.....

 5 144 0.698954 00:50:56:A9:2C:57 -> 00:50:56:A9:0E:6F MPLS unicast
0000: 005056A9 0E6F0050 56A92C57 81000DD6 .PV..o.PV.,W....
0010: 88471676 20FF0177 71FF0000 00000050 .G.v ..wq.....P
0020: 56A9862A 005056A9 1AAA0800 45000064 V...*.PV.....E..d
0030: 00580000 FF01A732 0A000001 0A00000E .X.....2.....

```

```

R4#show bridge-domain 3467 | include 862[Aa]
 0 0050.56A9.862A forward dynamic 295 LAN.1004019

```

We can also verify multicast traffic replication. All multicast, both routed and link-local (LL), is replicated to all PWs in the mesh at all times. This is very inefficient for routed multicast compared to any MVPN profile in conjunction with MPLS L3VPN. The PIM Assert process would occur on the emulated LAN but this does not stop the traffic from flooding over the core network in the first place. Even though IGMP snooping appears to be supported on the bridge-domain, it does not properly track routers (PIM messaging or IGMP queries) or hosts (IGMP joins). It appears to be unsupported in VPLS architectures as the corresponding show commands never show any output. All routers are enabled for OSPFv2 on a broadcast network, while only CSR1 and CSR2 are also PIM routers. The configuration is not shown, but we verify it quickly on all routers using a variety of show commands for variety.

```

P/0/0/CPU0:XRv4#show ospf vrf LAN neighbor | begin ID
Neighbor ID      Pri   State           Dead Time   Address      Interface
10.0.0.1         1     FULL/BDR        00:00:39    10.0.0.1
GigabitEthernet0/0/0/0.564
Neighbor is up for 00:00:30
10.0.0.2         1     FULL/DR         00:00:31    10.0.0.2
GigabitEthernet0/0/0/0.564
Neighbor is up for 00:00:30

```

```

R1#show ip ospf 1 neighbor summary | include FULL
FULL 2

```

```

R2#show ip ospf 1 0 database network self-originate | begin Mask
Network Mask: /24
Attached Router: 10.0.0.2
Attached Router: 10.0.0.1
Attached Router: 10.0.0.14

```

```

R1#show ip pim vrf LAN neighbor | begin ^Neighbor
Neighbor      Interface           Uptime/Expires   Ver   DR
Address
10.0.0.2      GigabitEthernet2.541 02:28:06/00:01:16 v2    1 / DR SPG

```



```

R2#show ip pim vrf LAN interface
Address      Interface      Ver/  Nbr   Query  DR      DR
            Mode      Count Intvl  Prior
10.0.0.2     GigabitEthernet2.527 v2/S  1     30     1       10.0.0.2

```

Capturing outbound on CSR4 shows OSPF hellos being replicated along all PWs. We can assume the same thing happens for PIM as well (I verified it, but it is not shown for brevity). The label stack is ignored since we have verified it several times, and the OSPF destination IP address of 224.0.0.5 is highlighted in green. Because this is layer 2 transport, I also highlight the destination MAC of 0100.5e00.0005 in yellow. The BD evaluates this, sees that the I/G bit is set, and floods it.

```

R4#show monitor capture CAP buffer detailed
 0 128 0.000000 00:50:56:A9:2C:57 -> 00:50:56:A9:0E:6F MPLS unicast
0000: 005056A9 0E6F0050 56A92C57 8100DD6 .PV..o.PV.,W....
0010: 88471676 10FF01B7 91FF0000 00000100 .G.v.....
0020: 5E000005 005056A9 1AAA0800 45C00054 ^....PV.....E..T
0030: 038E0000 0159CAFD 0A000001 E0000005 .....Y.....

 1 128 0.000000 00:50:56:A9:2C:57 -> 00:50:56:A9:0E:6F MPLS unicast
0000: 005056A9 0E6F0050 56A92C57 8100DD6 .PV..o.PV.,W....
0010: 88471676 20FF0177 71FF0000 00000100 .G.v ..wq.....
0020: 5E000005 005056A9 1AAA0800 45C00054 ^....PV.....E..T
0030: 038E0000 0159CAFD 0A000001 E0000005 .....Y.....

 2 124 0.000000 00:50:56:A9:2C:57 -> 00:50:56:A9:0E:6F MPLS unicast
0000: 005056A9 0E6F0050 56A92C57 8100DD6 .PV..o.PV.,W....
0010: 88471676 70FF16B5 11FF0100 5E000005 .G.vp.....^...
0020: 005056A9 1AAA0800 45C00054 038E0000 .PV.....E..T....
0030: 0159CAFD 0A000001 E0000005 02010034 .Y.....4

```

For completeness, we test routed multicast also. With IGMP snooping dysfunctional, we expect it to be flooded just like LL multicast. We use the GLOP group 233.190.239.238 (0xE9BEEFEE) because it stands out nicely in hex and is highlighted in green. The timestamps are identical, and I've highlighted the layer 2 and 3 multicast addresses in yellow. This multicast treatment makes MPLS L3VPN a much better option for multicast transport, especially when there are few listeners and high bandwidth flows. Recall that PIM was not even enabled on XRv4, yet multicast traffic is being flooded to it, adding to the inefficiency of the design.

```

R4#show monitor capture CAP buffer detailed
 1 144 0.646014 00:50:56:A9:2C:57 -> 00:50:56:A9:0E:6F MPLS unicast
0000: 005056A9 0E6F0050 56A92C57 8100DD6 .PV..o.PV.,W....
0010: 88471676 10FF01B7 91FF0000 00000100 .G.v.....
0020: 5E3EEFEE 005056A9 1AAA0800 45000064 ^>...PV.....E..d
0030: 006B0000 FF01D77F 0A000001 E9BEEFEE .k.....

 2 144 0.646014 00:50:56:A9:2C:57 -> 00:50:56:A9:0E:6F MPLS unicast

```

```

0000: 005056A9 0E6F0050 56A92C57 81000DD6 .PV..o.PV.,W....
0010: 88471676 20FF0177 71FF0000 00000100 .G.v ..wq.....
0020: 5E3EEFEE 005056A9 1AAA0800 45000064 ^>...PV.....E..d
0030: 006B0000 FF01D77F 0A000001 E9BEEFEE .k.....

3 140 0.646014 00:50:56:A9:2C:57 -> 00:50:56:A9:0E:6F MPLS unicast
0000: 005056A9 0E6F0050 56A92C57 81000DD6 .PV..o.PV.,W....
0010: 88471676 70FF16B5 11FF0100 5E3EEFEE .G.vp.....^>..
0020: 005056A9 1AAA0800 45000064 006B0000 .PV.....E..d.k..
0030: FF01D77F 0A000001 E9BEEFEE 0800F378 .....x

```

Let's get the CW working between XRv3 and CSR4. CSR4 was expecting to use the CW but XRv3 was not even offering it. We create a PW-class similar to legacy IOS syntax and apply it to the manual PW within the XR bridge-domain. The PW flaps but comes back up with the CW negotiated properly.

```
RP/0/0/CPU0:XRv3#show l2vpn bridge-domain pw-id 413 detail | include Control word
```

```
Control word disabled disabled
```

```

! XRv3
l2vpn
pw-class PW_CW
  encapsulation mpls
  control-word
vfi LAN
  neighbor 55.0.0.4 pw-id 413
  pw-class PW_CW

```

```
RP/0/0/CPU0:XRv3#show l2vpn bridge-domain pw-id 413 detail | include Control word
```

```
Control word enabled enabled
```

```

R4#show l2vpn atom vc vcid 413 detail | include Control
Control word on (configured: autosense) on

```

We know that adding the CW can prevent “wrong” load balancing when the first nibble of the destination MAC address is either a 4 or 6, since the P routers will attempt to load share based on the IPv4/v6 information within the tunneled packet. The CW allows the router to use the bottom label as the traffic would be properly classified as non-IP. This introduces a new issue, specifically one related to forwarding polarization. The core problem is that all traffic in a given PW, as part of a VPLS instance, is always following the same core path. The flow-aware transport of pseudowires (FAT-PW) adds another label to the stack. It sits beneath the PW label and above the CW, if present. If the CW is not used, it sits under the PW and before the payload. The TTL of this shim header is always 1 in the unlikely event the PW label is accidentally popped early, which guarantees the frame will not be forwarded. The label values are determined by the router based on a source/destination hash. FAT negotiation requires new LDP TLVs, specifically the “T” and “R” TLVs for transmit and receive, respectively. LDP signaling must be

enabled; static PWs cannot exchange the FAT TLV. This feature is only available with the new syntax under a PW template or interface PW; it is not available under a PW-class or xconnect stanza.

```
! CSR4
interface pseudowire46
  load-balance flow ip src-ip
  load-balance flow-label both
```

```
! CSR6
interface pseudowire46
  load-balance flow ip dst-ip
  load-balance flow-label both
```

```
R4#show l2vpn atom vc vcid 46 detail | include Load|Flow
Load Balance: ECMP
Flow Label      T=1, R=1                      T=1, R=1
```

```
R6#show l2vpn atom vc vcid 46 detail | include Load|Flow
Load Balance: ECMP
Flow Label      T=1, R=1                      T=1, R=1
```

Theoretically, we should be able to see load sharing per flow based on the Ethernet addressing (or possibly the IP addressing, if we specify). Using the local label of the PW, along with exact source/destination addressing, the LFIB should tell us the flow label. Only one PW is currently enabled for FAT, so we need to use the proper local label. We see four different flow labels based on IP addresses and would expect to see this at the bottom of the label stack. Note: If FAT is not enabled, the “Ethernet” option for load balancing does not show up in the show command at all.

```
R4#show l2vpn atom binding 46 | include Local
Local Label: 4012
```

```
R4#show mpls forwarding-table exact-route label 4000 ipv4 source 1.1.1.1
destination 10.0.0.14 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label          or Tunnel Id    Switched     interface
4000       No Label       12ckt(5)        306          none      point2point
MAC/Encaps=0/0, MRU=0, Label Stack{}
No output feature configured
Flow label: 117702
```

```
R4#show mpls forwarding-table exact-route label 4000 ipv4 source 1.1.1.2
destination 10.0.0.14 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label          or Tunnel Id    Switched     interface
4000       No Label       12ckt(5)        306          none      point2point
MAC/Encaps=0/0, MRU=0, Label Stack{}
No output feature configured
```

Flow label: 119774

```
R4#show mpls forwarding-table exact-route label 4000 ipv4 source 1.1.1.3
destination 10.0.0.14 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
4000       No Label  12ckt(5)        408          none      point2point
MAC/Encaps=0/0, MRU=0, Label Stack{}
No output feature configured
Flow label: 121814
```

```
R4#show mpls forwarding-table exact-route label 4000 ipv4 source 1.1.1.4
destination 10.0.0.14 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
4000       No Label  12ckt(5)        408          none      point2point
MAC/Encaps=0/0, MRU=0, Label Stack{}
No output feature configured
Flow label: 123886
```

I've added 1.1.1.1, 1.1.1.2, 1.1.1.3, and 1.1.1.4 to CSR1 to use as source addresses for load sharing. A simple TCL script on CSR1 will allow us to send a single packet in each "flow". One would expect to see the aforementioned labels in the packets.

```
! CSR1
tclsh
foreach x {
1.1.1.1
1.1.1.2
1.1.1.3
1.1.1.4
} { ping vrf LAN 10.0.0.14 repeat 1 source $x }
```

We can see there is another label at the bottom of the label stack, under the PW label and before the CW. After an hour of testing, I was never able to see any of the parser-displayed MPLS labels in the packet captures. Most of the time, I saw label 0x80000 (decimal 524288) which is highlighted in yellow. As with the first packet, sometimes I saw 0x82080 (decimal 532608) but it was dependent upon the load balancing method. I also highlight the source IP from each flow so you can clearly see the packets should have different hashes (green). I would have preferred to at least see the proper labels, even if the core load balancing was suboptimal due to only having 4 flows.

```
R4#show monitor capture CAP buffer detailed
 1 148 0.340985 00:50:56:A9:2C:57 -> 00:50:56:A9:0E:6F MPLS unicast
0000: 005056A9 0E6F0050 56A92C57 81000DD6 .PV..o.PV.,W....
0010: 88471676 20FF0177 70FF8208 01010000 .G.v ..wp.....
0020: 00000050 56A9862A 005056A9 1AAA0800 ...PV...*.PV....
0030: 45000064 00A20000 FF01AEE7 01010101 E..d.....
```

```

2 148 0.348981 00:50:56:A9:2C:57 -> 00:50:56:A9:0E:6F MPLS unicast
0000: 005056A9 0E6F0050 56A92C57 81000DD6 .PV..o.PV.,W....
0010: 88471676 20FF0177 70FF8000 01010000 .G.v ..wp.....
0020: 00000050 56A9862A 005056A9 1AAA0800 ...PV..*.PV.....
0030: 45000064 00A30000 FF01AEE5 01010102 E..d.....

3 148 0.357983 00:50:56:A9:2C:57 -> 00:50:56:A9:0E:6F MPLS unicast
0000: 005056A9 0E6F0050 56A92C57 81000DD6 .PV..o.PV.,W....
0010: 88471676 20FF0177 70FF8000 01010000 .G.v ..wp.....
0020: 00000050 56A9862A 005056A9 1AAA0800 ...PV..*.PV.....
0030: 45000064 00A40000 FF01AEE3 01010103 E..d.....

4 148 0.366985 00:50:56:A9:2C:57 -> 00:50:56:A9:0E:6F MPLS unicast
0000: 005056A9 0E6F0050 56A92C57 81000DD6 .PV..o.PV.,W....
0010: 88471676 20FF0177 70FF8000 01010000 .G.v ..wp.....
0020: 00000050 56A9862A 005056A9 1AAA0800 ...PV..*.PV.....
0030: 45000064 00A50000 FF01AEE1 01010104 E..d.....

```

Enabling debug on CSR4 to troubleshoot this, I see several messages early in the PW setup that FAT is not supported by CSR6, which is false. I believe this is during the initiation exchange before the “T” and “R” TLVs have been fully exchanged, but the debugging hints that the feature isn’t supported at all.

```

! CSR4
AToM[55.0.0.6, 46]: Received Go Active service directive
AToM[55.0.0.6, 46]: . Receive status update
AToM[55.0.0.6, 46]: ..... NMS: VC oper state: DOWN
AToM[55.0.0.6, 46]: ..... NMS: err codes: no-err
AToM[55.0.0.6, 46]: ..... SYSLOG: VC is DOWN
AToM[55.0.0.6, 46]: .... Local ready
AToM[55.0.0.6, 46]: ..... Local service is ready; send a label
AToM[55.0.0.6, 46]: ..... Alloc local binding
AToM[55.0.0.6, 46]: ..... Alloc label for dynamic
AToM[55.0.0.6, 46]: ..... Populate local binding
AToM[55.0.0.6, 46]: ..... Local AC type Ethernet, PW type VFI
AToM[55.0.0.6, 46]: ..... VFI, capability C000505 from platform/interface
AToM[55.0.0.6, 46]: ..... Capability C000505, returned cap C000505, mask
FFDEFFBF
AToM[55.0.0.6, 46]: ..... Flow-based Load Balancing NOT supported
AToM[55.0.0.6, 46]: ..... Flow Classification src-mac NOT supported
AToM[55.0.0.6, 46]: ..... Flow Classification dst-mac NOT supported
AToM[55.0.0.6, 46]: ..... Flow Classification src-dst-mac NOT supported
AToM[55.0.0.6, 46]: ..... Flow Classification src-ip NOT supported
AToM[55.0.0.6, 46]: ..... Flow Classification dst-ip NOT supported
AToM[55.0.0.6, 46]: ..... Flow Classification src-dst-ip NOT supported
AToM[55.0.0.6, 46]: ..... Flow-label processing NOT supported

```

When the PW is actually made, the “fat” feature appears to be supported and enabled. We will move on past this feature for now. One thing to note is that the “load balance” method seen earlier says ECMP, where the documentation says “flow”. Considered I enabled flow-based load balancing, I cannot explain the discrepancy. This may be attributed to the CSR1000v specifically, but the feature is worth discussing at it solves real-world problems.

```
! CSR4
AToM[55.0.0.4, 46]: ..... Provision SSM with PWID 5, VC ID 46, Block ID 0
AToM[55.0.0.4, 46]: ..... Set imp flags: cw ra
AToM[55.0.0.4, 46]: ..... : ttl
AToM[55.0.0.4, 46]: ..... : fat
AToM[55.0.0.4, 46]: ..... Set segment count to 1
AToM[55.0.0.4, 46]: ..... Set disp flags: cw
AToM[55.0.0.4, 46]: ..... : fat
```

“Routed” VPLS is useful to enable a layer 3 touchpoint on a VPLS PE. This can be use to participate somewhat in the customer routed network. A BDI is equivalent to an SVI or BVI in that it lives transparently in the “bridge-domain”. The BVI number must match the BD number, just as SVIs must match the VLAN IDs they serve on layer 3 switches. On CSR4 and CSR7, we configure BDI interfaces and enable OSPF on them. We add a LAN VRF also so that this routing information doesn’t mingle with the global routing table used in the MPLS core. Only CSR4 is shown for brevity since the configurations are nearly identical on CSR7. Note: BDI interfaces are shutdown by default.

```
! CSR4
interface Loopback3467
 vrf forwarding LAN
 ip address 4.4.4.4 255.255.255.255
 ip ospf 1 area 0

interface BDI3467
 vrf forwarding LAN
 ip address 10.0.0.4 255.255.255.0
 ip ospf priority 0
 ip ospf 1 area 0
 no shutdown
```

From an OSPF perspective, be careful about using multi-access network types over VPLS. Since the multipoint L2VPN network is just a collection of P2P PWs, a sufficient number of links may fail to the point where all nodes do not have direct layer 2 reachability, which is not supported for all NBMA designs. Everyone must have reachability to the DR at all times; consider P2MP network type as an alternative despite some of the flooding inefficiencies it may bring. We won’t test this in detail since basic OSPF NBMA design considerations are beyond the scope of this test. In the current design, both CSR4 and CSR7 are now part of the “broadcast network”. There are 5 total OSPF routers in the VPLS, 3 of which are CEs and 2 of which are PEs. IP connectivity is functional over the network as expected.

```
R4#show ip ospf 1 neighbor
Neighbor ID      Pri   State           Dead Time   Address        Interface
7.7.7.7          0    2WAY/DROTHER     00:00:37    10.0.0.7       BDI3467
10.0.0.1         1    FULL/BDR         00:00:38    10.0.0.1       BDI3467
10.0.0.2         1    2WAY/DROTHER     00:00:33    10.0.0.2       BDI3467
10.0.0.14        1    FULL/DR          00:00:36    10.0.0.14      BDI3467
```

```
R4#ping vrf LAN 10.0.0.14 source 4.4.4.4
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.14, timeout is 2 seconds:
Packet sent with a source address of 4.4.4.4
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 6/21/73 ms
```

```
R7#ping vrf LAN 10.0.0.1 source 7.7.7.7
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.1, timeout is 2 seconds:
Packet sent with a source address of 7.7.7.7
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 7/9/13 ms
```

Just like SVIs, the BDI gets a virtual MAC address so that it can participate in IPv4 ARP and IPv6 ND. The BD CAM now shows several new entries. There is a local “to_bdi” entry that consumes locally-destined traffic. We also see 0001E.E5DD.FFBF which we presume is the MAC address of CSR7’s BDI. It is reachable via the same pseudoport as CSR2’s MAC address that we used for testing earlier.

```
R4#show interface bdi3467 | include bia
Hardware is BDI, address is 001e.4977.7cbf (bia 001e.4977.7cbf)

R4#show bridge-domain 3467
Bridge-domain 3467 (5 ports in all)
State: UP                               Mac learning: Enabled
Aging-Timer: 300 second(s)
  BDI3467 (up)
  GigabitEthernet2 service instance 3541
  vfi LAN neighbor 55.0.0.13 413
  vfi LAN neighbor 55.0.0.7 47
  vfi LAN neighbor 55.0.0.6 46
  AED MAC address      Policy Tag      Age  Pseudoport
  0  0050.56A9.1AAA forward dynamic 298  GigabitEthernet2.EFP3541
  0  0050.56A9.862A forward dynamic 296  LAN.1004015
  0  001E.E5DD.FFBF forward dynamic 295  LAN.1004014
  1  FFFF.FFFF.FFFF flood static 0    OLIST_PTR:0xe80d1800
  0  0050.56A9.BE8A forward dynamic 297  LAN.1004014
  -  001E.4977.7CBF to_bdi static 0    BDI3467
```

The BD also has some MAC address customizations. On CSR6, we enable several. First, we will disable MAC address learning, forcing CSR6 to flood all frames out of all PWs since all unicast is now “unknown”

by definition. This could be useful if the BD/VFI constructs are used for dynamically creating a VPWS (E-LINE) service. Since traditional xconnects never retain MAC addresses (no reason to between P2P peers), the router can emulate that behavior using this command. The memory efficiencies of xconnect-based E-LINE can translate into VFI-based E-LINE using this technique. The feature is generally not useful when there are more than 2 endpoints in the L2VPN. The exception would be if the vast majority of traffic was multicast, then it would not matter much.

```
! CSR6
bridge-domain 3467
  no mac learning

R6#show bridge-domain 3467
Bridge-domain 3467 (4 ports in all)
State: UP                               Mac learning: Disabled
Aging-Timer: 300 second(s)
  GigabitEthernet2 service instance 3564
  vfi LAN neighbor 55.0.0.13 613
  vfi LAN neighbor 55.0.0.4 46
  vfi LAN neighbor 55.0.0.7 67
  AED MAC address      Policy Tag      Age   Pseudoport
  1   FFFF.FFFF.FFFF flood  static 0     OLIST_PTR:0xe806b400
```

Next, we re-enable MAC learning on CSR6 to test other features. We enable a 60 second timer to keep the MAC address table smaller, but expect more frequency unknown unicast flooding. We also instruct the BD not to flood traffic from MAC addresses not in the BD CAM table when the limit is reached; the default is to treat it like unknown unicast and flood it. The traffic is dropped instead using this command, which saves bandwidth at the expense of breaking traffic flows. There do not appear to be any specific show-commands to validate these limit/flood minor features. Notice that two MACs are learned via two separate PWs. This represents the CSR4 BDI + CSR1 MAC (grey port), and the CSR7 BDI + CSR2 MAC (green port). Since XRv3 cannot support L2VPN, no MACs from that pseudoport are shown in the BD CAM.

```
! CSR6
bridge-domain 3467
  mac learning
  mac aging-time 60
  mac limit maximum addresses 16
  mac limit action flooding disable

R6#show bridge-domain 3467
Bridge-domain 3467 (4 ports in all)
State: UP                               Mac learning: Enabled
Aging-Timer: 60 second(s)
  GigabitEthernet2 service instance 3564
  vfi LAN neighbor 55.0.0.13 613
  vfi LAN neighbor 55.0.0.4 46
```



```

vfi LAN neighbor 55.0.0.7 67
AED MAC address      Policy Tag      Age      Pseudoport
0  0050.56A9.1AAA forward dynamic 59      LAN.1004015
0  0050.56A9.862A forward dynamic 54      GigabitEthernet2.EFP3564
0  001E.E5DD.FFBF forward dynamic 60      LAN.1004016
1  FFFF.FFFF.FFFF flood static 0       OLIST_PTR:0xe806b400
0  0050.56A9.BE8A forward dynamic 53      LAN.1004016
0  001E.4977.7CBF forward dynamic 54      LAN.1004015

```

We can also set a global limit for all bridge-domains, as well as disable MAC flooding. We can see the BD CAM only contains two MAC addresses, and the local BDI counts against this maximum.

```

! CSR4
ethernet mac limit maximum addresses 2
ethernet mac limit action flooding disable

R4#show bridge-domain
Bridge-domain 3467 (5 ports in all)
State: UP                               Mac learning: Enabled
Aging-Timer: 300 second(s)
  BDI3467 (up)
  GigabitEthernet2 service instance 3541
  vfi LAN neighbor 55.0.0.13 413
  vfi LAN neighbor 55.0.0.7 47
  vfi LAN neighbor 55.0.0.6 46
AED MAC address      Policy Tag      Age      Pseudoport
1  FFFF.FFFF.FFFF flood static 0       OLIST_PTR:0xe80d1800
0  0050.56A9.BE8A forward dynamic 297     LAN.1004014
-  001E.4977.7CBF to_bdi static 0       BDI3467

```

Because the routers are constantly generating traffic (OSPF, PIM, etc), this limit will be exceeded and the router will alarm us.

```

! CSR4
%BDOMAIN-4-MAC_LIMIT_VIOL_SYS_LIMIT: Attempt to learn a new MAC on
GigabitEthernet2 service instance 3541 (bridge-domain 3467) caused system
limit (2) to be exceeded.

```

Some basic OSPF debugging on CSR4 shows us that, since CSR4 didn't learn XRv4's MAC address, it cannot communicate via unicast to complete the OSPF adjacency.

```

! CSR4
OSPF-1 ADJ   BD3467: Cannot see ourself in hello from 10.0.0.14, state INIT
OSPF-1 ADJ   BD3467: Nbr 10.0.0.14: Clean-up dbase exchange
OSPF-1 ADJ   BD3467: Neighbor change event
OSPF-1 ADJ   BD3467: DR/BDR election
OSPF-1 ADJ   BD3467: Elect BDR 10.0.0.2

```

```

OSPF-1 ADJ    BD3467: Elect DR 10.0.0.2
OSPF-1 ADJ    BD3467: DR: 10.0.0.2 (Id)
OSPF-1 ADJ    BD3467:    BDR: 10.0.0.2 (Id)
OSPF-1 ADJ    BD3467: Remember old DR 10.0.0.14 (id)

```

XRv4 doesn't need to do any unicast exchange with CSR4 or CSR7, but it does with CSR1 since it is the BDR. Because CSR1 cannot reply to the messages (CSR4 isn't forwarding them), the neighbor can never form.

```

RP/0/0/CPU0:XRv4#show ospf vrf LAN neighbor | begin State
Neighbor ID      Pri   State                Dead Time   Address      Interface
4.4.4.4          0    2WAY/DROTHER         00:00:37    10.0.0.4
GigabitEthernet0/0/0/0.564
    Neighbor is up for 00:00:39
7.7.7.7          0    2WAY/DROTHER         00:00:34    10.0.0.7
GigabitEthernet0/0/0/0.564
    Neighbor is up for 00:00:42
10.0.0.1         1    EXSTART/BDR          00:00:31    10.0.0.1
GigabitEthernet0/0/0/0.564
    Neighbor is up for 00:00:37
10.0.0.2         1    FULL/DR              00:00:36    10.0.0.2
GigabitEthernet0/0/0/0.564
    Neighbor is up for 00:00:42

```

The more-specific configuration on the BD does not fix the issue as the system setting is the aggregate of all MAC addresses learned. The global setting should be set only to conserve memory on a router.

```

! CSR4
bridge-domain 3467
    mac limit maximum addresses 100

```

Changing the value from 2 to 200 alleviates the issue. Reducing the BD maximum to 2 generates a similar log message, except it is BD-scope, not system-scope. Interestingly, the BDI does not appear to count in this context, but only the truly dynamic MACs do. The system is now capable of storing many MAC addresses but this specific BD (and thus, the VPLS instance) cannot.

```

! CSR4
ethernet mac limit maximum addresses 200

bridge-domain 3467
    mac limit maximum addresses 2

```

```

! CSR4
%BDOMAIN-4-MAC_LIMIT_VIOL_BD_LIMIT: Attempt to learn a new MAC on (bridge-
domain 3467) caused configured bridge-domain limit (2) to be exceeded.

```

```

R4#show bridge-domain
Bridge-domain 3467 (5 ports in all)
State: UP                               Mac learning: Enabled
Aging-Timer: 300 second(s)
    BDI3467 (up)
    GigabitEthernet2 service instance 3541
    vfi LAN neighbor 55.0.0.13 413
    vfi LAN neighbor 55.0.0.7 47
    vfi LAN neighbor 55.0.0.6 46
    AED MAC address      Policy  Tag      Age  Pseudoport
    0  0050.56A9.1AAA forward dynamic 295  GigabitEthernet2.EFP3541
    1  FFFF.FFFF.FFFF flood  static  0    OLIST_PTR:0xe80d1800
    0  0050.56A9.BE8A forward dynamic 291  LAN.1004014
    -  001E.4977.7CBF to_bdi static  0    BDI3467

```

Even though the BD is not specifically blocking the MAC flooding when the threshold is reached, the system action is. You would think that exceeding the system threshold would incur the system penalty (block) and exceeding the BD threshold would incur the BD penalty (flood), but it doesn't. The traffic is blocked.

```

RP/0/0/CPU0:XRv4#show ospf vrf LAN neighbor 10.0.0.1 | begin 10.0.0.1
Neighbor 10.0.0.1, interface address 10.0.0.1
    In the area 0 via interface GigabitEthernet0/0/0/0.564
    Neighbor priority is 1, State is DOWN, 4 state changes
    [snip]

```

Removing the system-wide “block” action fixes this. The MAC addresses are still not written to BD CAM but can be flooded to the entire VPLS domain (treated as unknown unicast).

```

! CSR4
no ethernet mac limit action flooding disable

RP/0/0/CPU0:XRv4#show ospf vrf LAN neighbor 10.0.0.1 | begin 10.0.0.1
Neighbor 10.0.0.1, interface address 10.0.0.1
    In the area 0 via interface GigabitEthernet0/0/0/0.564
    Neighbor priority is 1, State is FULL, 10 state changes
    [snip]

```

Adding the action to the BD breaks it again. We theorize that the “more strict” MAC threshold action is preferred when the system and BD policies differ.

```

! CSR4
bridge-domain 3467
    mac limit action flooding disable

RP/0/0/CPU0:XRv4#show ospf vrf LAN neighbor 10.0.0.1 | begin 10.0.0.1
Neighbor 10.0.0.1, interface address 10.0.0.1

```

In the area 0 via interface GigabitEthernet0/0/0/0.564
Neighbor priority is 1, **State is EXSTART**, 3 state changes

Before continuing, we remove the BD-level adjustments from CSR4 for clarity.

```
! CSR4
bridge-domain 3467
  no mac limit action flooding disable
  no mac limit maximum addresses 2
```

Last, we will examine E-TREE. This is still a VPLS-based technology, but as defined earlier, it is not a complete LAN. CE devices are split into two categories: root and leaf. Roots can talk to all nodes, and leaves can only talk to roots. This is identical to how a switch with “protected” ports or private VLANs would operate. Nodes behind promiscuous or non-protected ports are roots, while nodes within secondary VLANs or protected ports are leaves. It is also similar to a Cisco ME switch with UNI ports where the UNI ports are leaves and the NNI ports are roots. The configuration for this involves NOT configuring a full PW mesh as we did for E-LAN. Simply skip configuring PWs between leaves and all tenets of static VPLS configuration remain the same. Like E-LAN, we will use a variety of L2VPN protocol-based CLI, legacy configuration, interface PW, and inline members. I also demonstrate that each PW can be different with respect to variables like status, CW, FAT-PW, sequencing, etc. The encapsulation must be the same, and the PW must not be statically configured.

```
! CSR8
interface pseudowire38
  description PW TO CSR3
  encapsulation mpls
  neighbor 55.0.0.3 38
  signaling protocol ldp

interface pseudowire89
  description PW TO CSR9
  encapsulation mpls
  neighbor 55.0.0.9 89
  sequencing both
  signaling protocol ldp

template type pseudowire TMP_VPLS_STATIC
  encapsulation mpls
  no status

l2vpn vfi context TREE
  vpn id 3589
  member 55.0.0.5 58 template TMP_VPLS_STATIC
  member pseudowire89
  member pseudowire38
```

```

! CSR3, root
interface pseudowire39
  description PW TO CSR9
  encapsulation mpls
  neighbor 55.0.0.9 39

interface pseudowire38
  description PW TO CSR8
  encapsulation mpls
  neighbor 55.0.0.8 38

interface pseudowire35
  description PW TO CSR5
  encapsulation mpls
  neighbor 55.0.0.5 35
  control-word exclude

l2vpn vfi context TREE
  vpn id 3589
  member pseudowire39
  member pseudowire38
  member pseudowire35

! CSR9, leaf
interface pseudowire89
  description PW TO CSR8
  encapsulation mpls
  neighbor 55.0.0.8 89
  sequencing both
  signaling protocol ldp

interface pseudowire39
  description PW TO CSR3
  encapsulation mpls
  neighbor 55.0.0.3 39
  signaling protocol ldp

l2vpn vfi context TREE
  vpn id 3589
  member pseudowire89
  member pseudowire39

! CSR5, leaf
pseudowire-class PW_VPLS_TO_R8
  encapsulation mpls
  no status

pseudowire-class PW_VPLS_TO_R3
  encapsulation mpls

```

```
control-word
```

```
12 vfi TREE manual
vpn id 3589
bridge-domain 3589
neighbor 55.0.0.3 35 pw-class PW_VPLS_TO_R3
neighbor 55.0.0.8 58 pw-class PW_VPLS_TO_R8
```

A quick verification at each root makes sense as they will be connected to all leaves, and leaves are not connected to one another. From the perspective of the root nodes, the network looks like an E-LAN.

```
R3#show l2vpn atom vc service-name TREE
```

Interface	Peer ID	VC ID	Service		Status
			Type	Name	
pw35	55.0.0.5	35	vfi	TREE	UP
pw38	55.0.0.8	38	vfi	TREE	UP
pw39	55.0.0.9	39	vfi	TREE	UP

```
R8#show l2vpn atom vc service-name TREE
```

Interface	Peer ID	VC ID	Service		Status
			Type	Name	
pw38	55.0.0.3	38	vfi	TREE	UP
pw100007	55.0.0.5	58	vfi	TREE	UP
pw89	55.0.0.9	89	vfi	TREE	UP

We quickly verify the “special” PW components. PW89 has sequencing, PW35 has mismatched CW settings, and PW58 has status disabled. This is not specific to E-LAN or E-TREE, but is general PW review.

```
R8#show l2vpn atom vc vcid 89 detail | include Seq
```

```
Sequencing: receive enabled, send enabled
Sequencing resync disabled
```

```
R9#show l2vpn atom vc vcid 89 detail | include Seq
```

```
Sequencing: receive enabled, send enabled
Sequencing resync disabled
```

```
R3#show l2vpn atom vc vcid 35 detail | include Control
```

```
Control word off off
```

```
R5#show l2vpn atom vc vcid 35 detail | include Control
```

```
Control word off (configured: enabled) off
```

```
R5#show l2vpn atom vc vcid 58 detail | include Status_TLV
```

```
Status TLV support (local/remote) : disabled/not supported
Status TLV disabled not supported
```

```
R8#show l2vpn atom vc vcid 58 detail | include Status_TLV
Status_TLV support (local/remote) : disabled/not supported
Status TLV disabled not supported
```

We will enable EIGRP as the customer IGP over the E-TREE. Rather than test connectivity individually, we can just look at the IGP neighbors and see that the leaves only form neighbors with the roots, while the roots are fully meshed. Just like protected switchports, PVLANS, or UNI-to-UNI ports, there is no possibility of leaves communicating over the E-TREE directly via layer 2.

```
R1#show eigrrp address-family ipv4 vrf TREE neighbors | begin ^H
H Address Interface Hold Uptime SRTT RTO Q Seq
(sec) (ms) Cnt Num
2 10.0.0.14 Gi2.513 11 00:02:06 38 228 0 7
1 10.0.0.2 Gi2.513 11 00:05:20 10 100 0 1
0 10.0.0.10 Gi2.513 12 00:05:24 8 100 0 1
```

```
R10#show eigrrp address-family ipv4 vrf TREE neighbors | begin ^H
H Address Interface Hold Uptime SRTT RTO Q Seq
(sec) (ms) Cnt Num
2 10.0.0.14 Gi2.508 13 00:02:31 65 390 0 8
1 10.0.0.2 Gi2.508 14 00:05:45 28 168 0 4
0 10.0.0.1 Gi2.508 13 00:05:50 8 100 0 1
```

```
R2#show eigrrp address-family ipv4 vrf TREE neighbors | begin ^H
H Address Interface Hold Uptime SRTT RTO Q Seq
(sec) (ms) Cnt Num
1 10.0.0.10 Gi2.529 14 00:05:59 32 192 0 4
0 10.0.0.1 Gi2.529 13 00:05:59 15 100 0 4
```

```
RP/0/0/CPU0:XRv4#show eigrrp vrf TREE neighbors | begin ^H
H Address Interface Hold Uptime SRTT RTO Q Seq
(sec) (ms) Cnt Num
1 10.0.0.10 Gi0/0/0/0.554 14 00:01:18 15 200 0 8
0 10.0.0.1 Gi0/0/0/0.554 14 00:01:18 1 200 0 8
```

Examining the BD CAM tables on the roots, they have a total of 4 pseudoports with 4 MAC addresses. 1 is learned via the local EFP and three are learned via VPLS, one per PW, representing the other root plus two spokes. The exact pseudoport IDs and MAC addresses aren't relevant currently.

```
R8#show bridge-domain 3589
Bridge-domain 3589 (4 ports in all)
State: UP Mac learning: Enabled
Aging-Timer: 300 second(s)
GigabitEthernet2 service instance 3508
vfi TREE neighbor 55.0.0.5 58
vfi TREE neighbor 55.0.0.9 89
vfi TREE neighbor 55.0.0.3 38
AED MAC address Policy Tag Age Pseudoport
0 0050.56A9.862A forward dynamic 298 TREE.1004017
0 0050.56A9.F961 forward dynamic 298 GigabitEthernet2.EFP3508
```

0	0050.56A9.1AAA	forward	dynamic	298	TREE.100401a
0	0050.56A9.BE8A	forward	dynamic	298	TREE.100401b
1	FFFF.FFFF.FFFF	flood	static	0	OLIST_PTR:0xe8093820

R3#show bridge-domain 3589

Bridge-domain 3589 (4 ports in all)

State: UP Mac learning: Enabled

Aging-Timer: 300 second(s)

GigabitEthernet2 service instance 3531

vfi TREE neighbor 55.0.0.5 35

vfi TREE neighbor 55.0.0.8 38

vfi TREE neighbor 55.0.0.9 39

AED	MAC address	Policy	Tag	Age	Pseudoport
0	0050.56A9.862A	forward	dynamic	300	TREE.1004015
0	0050.56A9.F961	forward	dynamic	299	TREE.1004016
0	0050.56A9.1AAA	forward	dynamic	299	GigabitEthernet2.EFP3531
0	0050.56A9.BE8A	forward	dynamic	299	TREE.1004018
1	FFFF.FFFF.FFFF	flood	static	0	OLIST_PTR:0xe8108820

The spoke BD CAMs only show the root MAC addresses, one per PW. This is in addition to the local MAC address via the connected EFP.

R5#show bridge-domain 3589

Bridge-domain 3589 (3 ports in all)

State: UP Mac learning: Enabled

Aging-Timer: 300 second(s)

GigabitEthernet2 service instance 3554

vfi TREE neighbor 55.0.0.8 58

vfi TREE neighbor 55.0.0.3 35

AED	MAC address	Policy	Tag	Age	Pseudoport
0	0050.56A9.862A	forward	dynamic	298	GigabitEthernet2.EFP3554
0	0050.56A9.F961	forward	dynamic	298	TREE.100401a
0	0050.56A9.1AAA	forward	dynamic	299	TREE.100401b
1	FFFF.FFFF.FFFF	flood	static	0	OLIST_PTR:0xe8079860

R9#show bridge-domain 3589

Bridge-domain 3589 (3 ports in all)

State: UP Mac learning: Enabled

Aging-Timer: 300 second(s)

GigabitEthernet2 service instance 3529

vfi TREE neighbor 55.0.0.3 39

vfi TREE neighbor 55.0.0.8 89

AED	MAC address	Policy	Tag	Age	Pseudoport
0	0050.56A9.F961	forward	dynamic	296	TREE.1004015
0	0050.56A9.1AAA	forward	dynamic	297	TREE.1004016
0	0050.56A9.BE8A	forward	dynamic	300	GigabitEthernet2.EFP3529
1	FFFF.FFFF.FFFF	flood	static	0	OLIST_PTR:0xe80b8420

All normal rules apply with respect to layer 2 connectivity just as they would in any NBMA network; specifically, since Ethernet is used for transport, it looks like a PVLAN network. Assuming we wanted to violate the E-TREE design and enable connectivity between the spokes, there are several workarounds. None of these are VPLS related but can be valuable to overcome NBMA complexities when full reachability is required. The first option is to enable local-proxy-ARP on the root nodes. This allows them to proxy-ARP for hosts in the same subnet, which is even more extreme (and rarely used) than ordinary proxy-ARP. We can enable this on CSR10; when XRv4 ARPs for CSR2's MAC address, CSR10 will reply. The same is true in the other direction. This does not enable direct broadcast/multicast communication, so EIGRP multicast discovery doesn't start working.

```
! CSR10
interface GigabitEthernet2.508
 ip local-proxy-arp
```

```
RP/0/0/CPU0:XRv4#show arp vrf TREE | begin Address
Address      Age      Hardware Addr  State   Type   Interface
10.0.0.1     00:09:58  0050.56a9.1aaa Dynamic ARPA   Gigabi0/0/0/0.554
10.0.0.2     00:00:10  0050.56a9.f961 Dynamic ARPA   Gigabi0/0/0/0.554
10.0.0.10    02:10:03  0050.56a9.f961 Dynamic ARPA   Gigabi0/0/0/0.554
10.0.0.14    -         0050.56a9.862a Interface ARPA   Gigabi0/0/0/0.554
```

This means CSR10 is in the transit path, essentially hairpinning traffic using a suboptimal path. Because CSR2's MAC address was never communicated over the PW between CSR8 and CSR5 in the E-TREE, the BD CAM on CSR5 never learns it.

```
RP/0/0/CPU0:XRv4#traceroute vrf TREE 10.0.0.2
Type escape sequence to abort.
Tracing the route to 10.0.0.2
```

```
 1 10.0.0.10 29 msec 19 msec 0 msec
 2 10.0.0.2 9 msec 19 msec 29 msec
```

```
R5#show bridge-domain 3589
Bridge-domain 3589 (3 ports in all)
State: UP                               Mac learning: Enabled
Aging-Timer: 300 second(s)
  GigabitEthernet2 service instance 3554
    vfi TREE neighbor 55.0.0.8 58
    vfi TREE neighbor 55.0.0.3 35
AED MAC address  Policy Tag      Age  Pseudoport
0  0050.56A9.862A forward dynamic 298  GigabitEthernet2.EFP3554
0  0050.56A9.F961 forward dynamic 296  TREE.100401a
0  0050.56A9.1AAA forward dynamic 300  TREE.100401b
1  FFFF.FFFF.FFFF flood  static  0    OLIST_PTR:0xe8079860
```

We can avoid the ARP process altogether by statically configuring ARP entries on the spokes routers to use a root's MAC address. This may speed up communication since local-proxy-ARP resolution does not occur, and also may be necessary if the root node cannot enable local-proxy-ARP for security reasons. We could use multiple hubs to form some weak load sharing, but I select CSR1 for hairpinning this time. There is still no change to the leaf BD CAM as this technique is very similar to the first. Both CSR2 and XRv4 map one another's IOP addresses to CSR1's MAC address, placing CSR1 in the transit path.

```
! CSR2
arp vrf TREE 10.0.0.14 0050.56a9.1aaa arpa

! XRv4
arp vrf TREE 10.0.0.2 0050.56a9.1aaa arpa

RP/0/0/CPU0:XRv4#show arp vrf TREE 10.0.0.2 | begin Address
Address          Age          Hardware Addr  State    Type    Interface
10.0.0.2         -          0050.56a9.1aaa Static    ARPA    Gigabi0/0/0/0.554

R2#show ip arp vrf TREE 10.0.0.14
Protocol  Address          Age (min)  Hardware Addr  Type    Interface
Internet  10.0.0.14         -          0050.56a9.1aaa ARPA

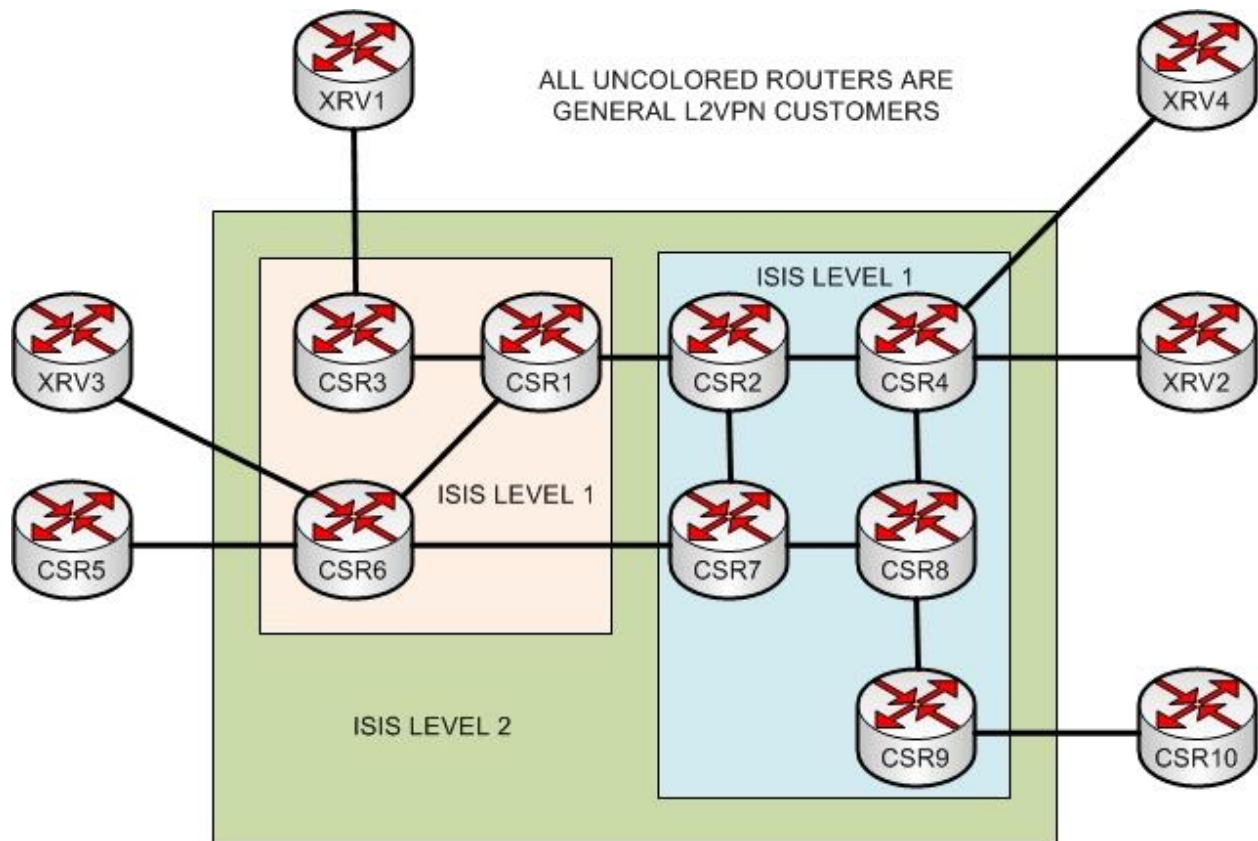
RP/0/0/CPU0:XRv4#traceroute vrf TREE 10.0.0.2
Type escape sequence to abort.
Tracing the route to 10.0.0.2

 1  10.0.0.1 49 msec  0 msec  59 msec
 2  10.0.0.2 19 msec  19 msec  19 msec
```

Additional Reading – Reference configurations "l2vpn-static"

32.1.1.4 Multisegment PW (MS-PW) switching

PW switching is a way to terminate PWs on a common node and have that router bridge the two PWs together. In this way, the PW is "stitched" to provide layer-2 transport end to end. This is typically used in Inter-AS environments (specifically, option B) or when there are disparate MPLS networks. I used a single AS for this test to demonstrate the feature. As we will see, you can chain as many PW switching routers in the path as you like, and I demonstrate using up to 2 PW switching routers. A new network was needed with a larger MPLS core to make this more interesting, and since XRv doesn't support L2VPN, the core is all CSR1000v. I used IS-IS L1 and L2 to add some complexity, as well as MPLS-TE. Rather than do a basic P2P example, I use static VPLS so that we can create many PWs for a combination of direct, single-switched, and double-switched paths.



With 4 PEs in the VPLS instance there are 6 (derived from $(n(n-1))/2$) total PWs. The PWs are configured as follows:

- CSR3 to CSR6 = direct
- CSR3 to CSR4 = double stitched via CSR1 and CSR2, 3 total PWs
- CSR3 to CSR9 = single stitched via CSR2, 2 total PWs
- CSR6 to CSR9 = direct
- CSR6 to CSR4 = single stitched via CSR7 (legacy configuration), 2 total PWs
- CSR4 to CSR9 = direct

For the basic configuration, I extensively use templates to speed up configuration. The VPLS configuration is very basic and uses the new L2VPN method only since that is not the focus here. The VFI contains three member PWs which are shown below. This is a member of a BD which also contains the EFP associated with the CE device. These basic BD, EFP, and L2VPN VFI configurations are not shown.

```
! All core LSRs
l2vpn
 logging pseudowire status
 logging redundancy

template type pseudowire TMP_VPLS
 encapsulation mpls
```

```
control-word include
signaling protocol ldp
```

The direct PWs are very simple. We will examine those three first, remembering that the VFI, BD, and EFP must be configured on each PE as well. The configuration is not shown, but we can verify them quickly. First we examine both ends of the CSR3-CSR6 link, followed by the CSR6-CSR9 and CSR4-CSR9 links. This covers half of the PWs inside of the VPLS instance.

```
R3#show l2vpn atom vc vcid 36 | begin Interface
Interface Peer ID      VC ID      Type      Name      Status
-----
pw36      17.0.0.6      36      vfi      LAN      UP

R6#show l2vpn atom vc vcid 36 | begin Interface
Interface Peer ID      VC ID      Type      Name      Status
-----
pw36      17.0.0.3      36      vfi      LAN      UP

R6#show l2vpn atom vc vcid 69 | begin Interface
Interface Peer ID      VC ID      Type      Name      Status
-----
pw69      17.0.0.9      69      vfi      LAN      UP

R9#show l2vpn atom vc vcid 69 | begin Interface
Interface Peer ID      VC ID      Type      Name      Status
-----
pw69      17.0.0.6      69      vfi      LAN      UP

R4#show l2vpn atom vc vcid 49 | begin Interface
Interface Peer ID      VC ID      Type      Name      Status
-----
pw49      17.0.0.9      49      vfi      LAN      UP

R9#show l2vpn atom vc vcid 49 | begin Interface
Interface Peer ID      VC ID      Type      Name      Status
-----
pw49      17.0.0.4      49      vfi      LAN      UP
```

Now, we will configure a single-stitch path. CSR3 to CSR9 should be stitched together by CSR2. The configuration on CSR3 and CSR9 is very basic and is essentially a PW identifying CSR2 as the endpoint. Again, it is assumed these PW interfaces are members of the L2VPN VFI. The point is that the non-stitched PW endpoints (the PEs in this case) don't have anything special in their configurations for MS-PW.

```
! CSR3
interface pseudowire23
 source template type pseudowire TMP_VPLS
```

```

neighbor 17.0.0.2 23

! CSR9
interface pseudowire29
  source template type pseudowire TMP_VPLS
  neighbor 17.0.0.2 29

```

CSR2 needs to terminate both of these sessions, essentially mirroring the configuration and reversing the neighbor address. The magic on CSR2 involves putting both PWs into an L2VPN xconnect pair. Normally, we used xconnects to include one PW and one EFP when doing normal VPWS/E-LINE. In this case, we can pair two PWs, but no more.

```

! CSR2
interface pseudowire23
  source template type pseudowire TMP_VPLS
  neighbor 17.0.0.3 23

interface pseudowire29
  source template type pseudowire TMP_VPLS
  neighbor 17.0.0.9 29

l2vpn xconnect context BRIDGE_PW39
  member pseudowire23
  member pseudowire29

```

At this point, all PWs should be up. Checking from the PW switching router is the fastest way to verify since it can see both ends, but we will check the endpoints for completeness.

```

R2#show l2vpn atom vc service-name BRIDGE_PW39 | begin Interface
Interface Peer ID      VC ID      Type      Name      Status
-----
pw23      17.0.0.3      23         p2p       BRIDGE_PW39  UP
pw29      17.0.0.9      29         p2p       BRIDGE_PW39  UP

```

```

R3#show l2vpn atom vc vcid 23 | begin Interface
Interface Peer ID      VC ID      Type      Name      Status
-----
pw23      17.0.0.2      23         vfi       LAN        UP

```

```

R9#show l2vpn atom vc vcid 29 | begin Interface
Interface Peer ID      VC ID      Type      Name      Status
-----
pw29      17.0.0.2      29         vfi       LAN        UP

```

Next, let's trace the LSP from CSR3 to CSR9 along this switched PW path. To determine the labels, we can look at the VC details for the quick answer. In this case, looking deeper is beneficial since the L2VPN bindings can also tell us the transit PW switching routers, which we cannot see using the VC show

command. The PW label of 2017 was allocated by CSR2, which is technically the end of the LSP. The L2VPN bindings reveals that 17.0.0.2 (CSR2) is the PW switching point and redirects traffic onto VC 29 towards 17.0.0.9 (CSR9). It is also worth noting that VCCV is not supported end to end when using this feature, and the CW cannot be carried end to end, either. We set the C-bit which means the CW is used, but the PW-ACH enhancement of the CW for OAM purposes is not supported.

```
R3#show l2vpn atom vc vcid 23 detail | include label_stack
Output interface: Gi2.513, imposed label_stack {1002 2017}
```

```
R3#show l2vpn atom binding 23
Destination Address: 17.0.0.2, VC ID: 23
Local Label: 3006
Cbit: 1, VC Type: Ethernet, GroupID: 0
MTU: 1500, Interface Desc: n/a
VCCV: CC Type: RA [2], TTL [3]
CV Type: LSPV [2]
Remote Label: 2017
Cbit: 1, VC Type: Ethernet, GroupID: 0
MTU: 1500, Interface Desc: n/a
VCCV: CC Type: TTL [3]
CV Type: LSPV [2]
PW Switching Point:
VC ID    Local ID    Remote ID    Description
29       17.0.0.2    17.0.0.9
```

```
R3#show mpls ldp bindings 17.0.0.2 32 neighbor 17.0.0.1
lib entry: 17.0.0.2/32, rev 11
remote binding: lsr: 17.0.0.1:0, label: 1002
```

When CSR1 gets the packet, it swaps the label to explicit-null (added for variety; any null label would work), then forwards the packet with label stack {0 2017} to CSR2. After removing the topmost label, CSR2 essentially performs a label swap from its locally-issued PW label to the one issued by CSR9. In doing so, it copies the EXP bits, decrements TTL, then pushes a new transport label (if necessary) to reach CSR9. This is similar to how Inter-AS Option B works for L3VPN since the VPNv4/v6 label is swapped at the ASBR; in both cases, this is technically a new LSP. For this reason, VCCV Type-1 switching with PW-ACH is not supported end to end. We can verify these labels operations in the control plane first.

```
R1#show mpls forwarding-table label 1002
Local      Outgoing    Prefix          Bytes Label    Outgoing    Next Hop
Label      Label       or Tunnel Id    Switched       interface
1002       explicit-n  17.0.0.2/32     3312           Gi2.512     17.1.2.2
```

The LFIB details indicate that the label is swapped to 9002. We can verify this is the correct PW label by checking the AToM bindings again. Notice that the bindings on the router performing the switching

don't show an intermediate hop, since from its perspective, we have only one router performing PW switching. Later examples detail what happens when the chain is longer.

```
R2#show mpls forwarding-table labels 2017 detail
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id Switched      interface
2017       No Label   12ckt(3)    0
Recursive paths, Label Stack{9002}
0232A000
No output feature configured
```

```
R2#show l2vpn atom binding 29
Destination Address: 17.0.0.9, VC ID: 29
Local Label: 2020
  Cbit: 1, VC Type: Ethernet, GroupID: 0
  MTU: 1500, Interface Desc: n/a
  VCCV: CC Type: TTL [3]
        CV Type: LSPV [2]
Remote Label: 9002
  Cbit: 1, VC Type: Ethernet, GroupID: 0
  MTU: 1500, Interface Desc: n/a
  VCCV: CC Type: RA [2], TTL [3]
        CV Type: LSPV [2]
```

The LFIB has indicated that there are recursive paths. This means that the LFIB is not preprogrammed with a transport label for its push operation since the RIB says there are ECMP paths via CSR7 and CSR4.

```
R2#show ip route 17.0.0.9
Routing entry for 17.0.0.9/32
  Known via "isis", distance 115, metric 30, type level-1
  Redistributing via isis 17
  Last update from 17.2.7.7 on GigabitEthernet2.527, 00:08:54 ago
  Routing Descriptor Blocks:
    17.2.7.7, from 17.0.0.9, 00:08:54 ago, via GigabitEthernet2.527
      Route metric is 30, traffic share count is 1
  * 17.2.4.4, from 17.0.0.9, 00:08:54 ago, via GigabitEthernet2.524
      Route metric is 30, traffic share count is 1
```

Because the packet is non-IP, we use the bottom label to determine ECMP selection. The incoming label and bottom label happen to be the same, which is fine. CSR2 selects CSR7 as the next-hop, so we can check the LDP bindings and FIB to verify the label value of 7007.

```
R2#show mpls forwarding-table exact-route label 2017 bottom-label 2017
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id Switched      interface
2017       No Label   12ckt(3)    0             Gi2.527     17.2.7.7
```

```
R2#show mpls ldp bindings 17.0.0.9 32 neighbor 17.0.0.7
lib entry: 17.0.0.9/32, rev 37
remote binding: lsr: 17.0.0.7:0, label: 7007
```

```
R2#show ip cef 17.0.0.9
17.0.0.9/32
nexthop 17.2.4.4 GigabitEthernet2.524 label 4005
nexthop 17.2.7.7 GigabitEthernet2.527 label 7007
```

We perform the same verification for the next singly-attached PW, but this time will verify it with EPC and EXP bit copying. CSR7 uses the legacy configuration but the operation is the same. The endpoint configuration is not shown as those are normal PWs with nothing special.

```
! CSR7
pseudowire-class PW_VPLS
encapsulation mpls
control-word

l2 vfi BRIDGE_PW46 point-to-point
neighbor 17.0.0.6 67 pw-class PW_VPLS
neighbor 17.0.0.4 47 pw-class PW_VPLS
```

We quickly verify that the two PWs are up. From CSR6 to CSR4, we can determine the label stack by checking the AToM bindings and the LDP labels for transport. The command also shows us the PW switching that occurs on CSR7, along with the adjacent PW and LSP endpoint address. The LDP label comes from CSR7, which is attached, so only label 7008 is used for this short PW.

```
R7#show mpls l2transport vc
```

Local intf	Local circuit	Dest address	VC ID	Status
pw100001	17.0.0.6 67	17.0.0.4	47	UP
pw100002	17.0.0.4 47	17.0.0.6	67	UP

```
R6#show l2vpn atom binding 67
Destination Address: 17.0.0.7, VC ID: 67
Local Label: 6002
Cbit: 1, VC Type: Ethernet, GroupID: 0
MTU: 1500, Interface Desc: n/a
VCCV: CC Type: RA [2], TTL [3]
CV Type: LSPV [2]
Remote Label: 7008
Cbit: 1, VC Type: Ethernet, GroupID: 0
MTU: 1500, Interface Desc: n/a
VCCV: CC Type: TTL [3]
CV Type: LSPV [2]
PW Switching Point:
VC ID Local ID Remote ID Description
```



```
47      17.0.0.7      17.0.0.4
```

```
R6#show mpls ldp bindings 17.0.0.7 32 neighbor 17.0.0.7
lib entry: 17.0.0.7/32, rev 11
remote binding: lsr: 17.0.0.7:0, label: imp-null
```

```
R6#show mpls l2transport vc 67 detail | include label_stack
Output interface: Gi2.567, imposed label stack {7008}
```

The LFIB on CSR7 shows a label swap operation occurring when we examine the details. The LFIB says “recursive paths” because there are multiple paths to reach CSR4 per the RIB. Since the traffic is non-IP, the bottom label is used to determine the ECMP forwarding. Since the PW label is the only label, it is used for the LFIB lookup and for ECMP selection.

```
R7#show mpls forwarding-table labels 7008 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id    Switched     interface
7008       No Label   12ckt(1)        662
Recursive paths, Label Stack{4008}
00FA8000
No output feature configured
```

```
R7#show mpls forwarding-table exact-route label 7008 bottom-label 7008
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id    Switched     interface
7008       No Label   12ckt(1)        662          Gi2.527   17.2.7.2
```

```
R7#show ip cef 17.0.0.4
17.0.0.4/32
  nexthop 17.2.7.2 GigabitEthernet2.527 label 2012
  nexthop 17.7.8.8 GigabitEthernet2.578 label 8001
```

We will begin sending DSCP CS1 traffic from CSR5 to XRv2. CSR7 captures this transit traffic for analysis using EPC.

```
R5#ping 10.0.0.12 tos 32
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.12, timeout is 2 seconds:
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 6/8/18 ms
```

We expect to see a label stack of {2012 4008} leaving CSR7 as a result. Capturing in on the link facing CSR6 and out on the link towards CSR2, we can confirm label 7008 (0x1B60) arrives with a CW. The EXP bits are copied (zero), the TTL is reduced from 255 to 254, the PW label is swapped to 4008 (0xFA8), and a new LDP label of 2012 (0x7DC) is pushed. MPLS shim headers are shown in yellow while control-words are shown in green. Shown in cyan is the DSCP CS1 marking; because AToM is not looking deep into the

Ethernet frame, this was not copied to the EXP bits on imposition. Later examples show the 802.1Q CoS being mapped to EXP automatically on imposition when using ATOM.

```
R7#show monitor capture CAP buffer detailed
  1  140      0.404038  00:50:56:A9:DE:0D -> 00:50:56:A9:EA:77 MPLS unicast
0000:  005056A9 EA770050 56A9DE0D 81000DEF  .PV..w.PV.....
0010:  884701B6 01FF0000 00000050 56A90E6F  .G.....PV..o
0020:  005056A9 DC630800 45200064 00190000  .PV..c..E .d....
0030:  FF01A74F 0A000005 0A00000C 0800358F  ...O.....5.

  2  144      0.404038  00:50:56:A9:EA:77 -> 00:50:56:A9:BE:8A MPLS unicast
0000:  005056A9 BE8A0050 56A9EA77 81000DC7  .PV....PV..w....
0010:  8847007D C0FF00FA 81FE0000 00000050  .G.}.....P
0020:  56A90E6F 005056A9 DC630800 45200064  V..o.PV..c..E .d
0030:  00190000 FF01A74F 0A000005 0A00000C  .....O.....
```

A good summary view of all of the PW labels used for the bridging feature is shown below. Even the new L2VPN commands can still show information about legacy configurations since it's just a parser update.

```
R7#show l2vpn service xconnect all detail | begin Interface
Interface          Group          Encapsulation          Prio  St  XC St
-----
VPWS name: BRIDGE_PW46, State: UP
pw100001           17.0.0.6:67 (MPLS)      0      UP  UP
                    Local VC label 7008
                    Remote VC label 6002
pw-class: PW_VPLS
pw100002           17.0.0.4:47 (MPLS)      0      UP  UP
                    Local VC label 7003
                    Remote VC label 4008
pw-class: PW_VPLS
```

We will quickly test the CoS to EXP mapping feature. CSR5 is configured to mark CoS for all outgoing traffic using CoS 1 for the outer tag and CoS 2 for the inner tag. This will allow us to tell which one the router selects for the MPLS EXP bits.

```
! CSR5
policy-map PMAP_COS
class class-default
set cos 1
set cos-inner 2

interface GigabitEthernet2.556
service-policy output PMAP_COS
```

The label stack has not changed, so we won't validate that again. Label 7008 arrived and label stack (2012 4008) exits towards CSR2. This time, we see a combination of 0x4 and 0x5 in the nibble where EXP

is stored. The positions with 0x5 indicate the S-bit being set which is not related to EXP, so we can assume 0x4 is the only set of EXP markings we see. This evaluates to binary 010x where x is a “don’t care” bit. We can assume this is EXP2, carried over from CoS 2, which was set as the inner CoS. I did not set the DSCP on the IP packet this time since we know it was not effective in transitioning directly to MPLS EXP at imposition; the value of 0 is shown in cyan. All new labels pushed by CSR7 (such as 2012) also contain the same EXP bits as the PW label, which is desirable. This proves that EXP is maintained across MS-PWs.

```
R7#show monitor capture CAP buffer detailed
 4 140      0.918004 00:50:56:A9:DE:0D -> 00:50:56:A9:EA:77 MPLS unicast
0000: 005056A9 EA770050 56A9DE0D 81000DEF .PV..w.PV.....
0010: 884701B6 05FF0000 00000050 56A90E6F .G.....PV..o
0020: 005056A9 DC630800 45000064 00230000 .PV..c..E..d.#..
0030: FF01A765 0A000005 0A00000C 0800E544 ...e.....D

 5 144      0.918004 00:50:56:A9:EA:77 -> 00:50:56:A9:BE:8A MPLS unicast
0000: 005056A9 BE8A0050 56A9EA77 81000DC7 .PV....PV..w....
0010: 8847007D C4FF00FA 85FE0000 00000050 .G.}.....P
0020: 56A90E6F 005056A9 DC630800 45000064 V..o.PV..c..E..d
0030: 00230000 FF01A765 0A000005 0A00000C .#.....e.....
```

The last and most complex example is the link from XRv1 to XRv2, which is actually three PWs tied together. CSR1 and CSR2 are both PW switching LSRs and will be swapping PW labels, which means there are three different LSPs.

```
! CSR1
interface pseudowire12
 source template type pseudowire TMP_VPLS
 neighbor 17.0.0.2 12

interface pseudowire13
 source template type pseudowire TMP_VPLS
 neighbor 17.0.0.3 13

l2vpn xconnect context BRIDGE_PW34
 member pseudowire13
 member pseudowire12

! CSR2
interface pseudowire12
 source template type pseudowire TMP_VPLS
 neighbor 17.0.0.1 12

interface pseudowire24
 source template type pseudowire TMP_VPLS
 neighbor 17.0.0.4 24
```

```
l2vpn xconnect context BRIDGE_PW34
member pseudowire12
member pseudowire24
```

First, we quickly ensure all of the PWs are up. I use different commands on each router for variety.

```
R3#show l2vpn atom vc vcid 13 | begin Interface
Interface Peer ID      VC ID      Type      Name      Status
-----
pw13      17.0.0.1      13        vfi        LAN        UP
```

```
R1#show l2vpn atom vc vcid 12 13 | begin Interface
Interface Peer ID      VC ID      Type      Name      Status
-----
pw12      17.0.0.2      12        p2p        BRIDGE_PW34  UP
pw13      17.0.0.3      13        p2p        BRIDGE_PW34  UP
```

```
R2#show l2vpn atom vc service-name BRIDGE_PW34 | begin Interface
Interface Peer ID      VC ID      Type      Name      Status
-----
pw12      17.0.0.1      12        p2p        BRIDGE_PW34  UP
pw24      17.0.0.4      24        p2p        BRIDGE_PW34  UP
```

```
R4#show l2vpn atom vc destination 17.0.0.2 | begin Interface
Interface Peer ID      VC ID      Type      Name      Status
-----
pw24      17.0.0.2      24        vfi        LAN        UP
```

Starting on CSR3, we can see the PW label from CSR1 for the first “leg” of the switched PW path. We also see all of the other PWs beyond the current one, their VC IDs, and their destinations. With explicit-null enabled, CSR3 will push {0 1005} onto the stack and send the MPLS packet towards CSR1.

```
R3#show l2vpn atom binding 13
Destination Address: 17.0.0.1,VC ID: 13
Local Label: 3002
  Cbit: 1,    VC Type: Ethernet,    GroupID: 0
  MTU: 1500,  Interface Desc: n/a
  VCCV: CC Type: RA [2], TTL [3]
  CV Type: LSPV [2]
Remote Label: 1005
  Cbit: 1,    VC Type: Ethernet,    GroupID: 0
  MTU: 1500,  Interface Desc: n/a
  VCCV: CC Type: TTL [3]
  CV Type: LSPV [2]
PW Switching Point:
  VC ID      Local ID      Remote ID      Description
  12         17.0.0.1      17.0.0.2
  24         17.0.0.2      17.0.0.4
```

```
R3#show ip cef 17.0.0.1
17.0.0.1/32
  nexthop 17.1.3.1 GigabitEthernet2.513 label explicit-null
```

```
R3#show l2vpn atom vc vcid 13 detail | include label_stack
  Output interface: Gi2.513, imposed label_stack {0 1005}
```

CSR1 consults its LFIB and swaps 1005 to 3002. We can also see this in the L2VPN xconnect service details. Because there are no recursive lookups, the LFIB pre-populates the entire encapsulation, except the TTL=0 in the show command for both shim headers. Not shown in the LFIB is the push operation to add explicit-null, which CSR2 requests.

```
R1#show l2vpn service xconnect name BRIDGE_PW34 detail | begin Interface
  Interface          Group          Encapsulation          Prio  St  XC St
  -----          -
  VPWS name: BRIDGE_PW34, State: UP
  pw13
    17.0.0.3:13 (MPLS)          0      UP  UP
    Local VC label 1005
    Remote VC label 3002

  pw12
    17.0.0.2:12 (MPLS)          0      UP  UP
    Local VC label 1008
    Remote VC label 2006
```

```
R1#show mpls forwarding-table labels 1005 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id    Switched     interface
1005       No Label   l2ckt(1)        0            Gi2.512   17.1.2.2
MAC/Encaps=18/26, MRU=1496, Label Stack{2006}
005056A9BE8A005056A91AAA81000DB88847 000000000007D6000
No output feature configured
```

```
R1#show ip cef 17.0.0.2
17.0.0.2/32
  nexthop 17.1.2.2 GigabitEthernet2.512 label explicit-null
```

Last time, the PW switching router did not show anything interesting in its AToM PW bindings. Since CSR1 is one of multiple PW switching LSRs, its PW to CSR2 (the next PW in the chain) accounts for all other downstream PWs. Just like CSR3 includes VC 12 and VC 24, CSR1 accounts for VC 24 only. We can also use this to triple-check the outbound PW label of 2006.

```
R1#show l2vpn atom binding 12
  Destination Address: 17.0.0.2, VC ID: 12
  Local Label: 1008
    Cbit: 1,      VC Type: Ethernet,      GroupID: 0
    MTU: 1500,    Interface Desc: n/a
```

```

VCCV: CC Type: TTL [3]
      CV Type: LSPV [2]
Remote Label: 2006
Cbit: 1,      VC Type: Ethernet,      GroupID: 0
MTU: 1500,    Interface Desc: n/a
VCCV: CC Type: TTL [3]
      CV Type: LSPV [2]
PW Switching Point:
VC ID      Local ID      Remote ID      Description
24         17.0.0.2        17.0.0.4

```

CSR2 receives the stack {0 2006} and performs a lookup on the PW label. The label is swapped again to 4009 and explicit-null is not imposed this time. This is the remote PW label learned from CSR4 along VC 24. Notice that for VC 24, CSR2 does not show any PW switching LSRs since this is the last PW in the left-to-right direction.

```

R2#show mpls forwarding-table labels 2006 detail
Local      Outgoing    Prefix          Bytes Label    Outgoing    Next Hop
Label      Label        or Tunnel Id    Switched       interface
2006       No Label     12ckt(1)        2882           Gi2.524     17.2.4.4
MAC/Encaps=18/22, MRU=1500, Label Stack{4009}
005056A92C57005056A9BE8A81000DC48847 00FA9000
No output feature configured

```

```

R2#show l2vpn atom binding 24
Destination Address: 17.0.0.4,VC ID: 24
Local Label: 2009
Cbit: 1,      VC Type: Ethernet,      GroupID: 0
MTU: 1500,    Interface Desc: n/a
VCCV: CC Type: TTL [3]
      CV Type: LSPV [2]
Remote Label: 4009
Cbit: 1,      VC Type: Ethernet,      GroupID: 0
MTU: 1500,    Interface Desc: n/a
VCCV: CC Type: RA [2], TTL [3]
      CV Type: LSPV [2]

```

```

R2#show l2vpn atom vc vcid 24 detail | include label_stack
Output interface: Gi2.524, imposed label stack {4009}

```

CSR2 would show CSR1 as a PW switching LSR if we were tracing the LSP in the opposite direction, which is shown below as well (we aren't tracing that one, though). This is a brief aside only.

```

R2#show l2vpn atom binding 12
Destination Address: 17.0.0.1,VC ID: 12
Local Label: 2006
Cbit: 1,      VC Type: Ethernet,      GroupID: 0
MTU: 1500,    Interface Desc: n/a

```

```

VCCV: CC Type: TTL [3]
      CV Type: LSPV [2]
Remote Label: 1008
Cbit: 1,      VC Type: Ethernet,      GroupID: 0
MTU: 1500,    Interface Desc: n/a
VCCV: CC Type: TTL [3]
      CV Type: LSPV [2]
PW Switching Point:
  VC ID   Local ID   Remote ID   Description
  13      17.0.0.1   17.0.0.3

```

CSR4 receives label 4009 which is the PW label it advertised to CSR2. This is the end of the MPLS encapsulation for the traffic as CSR4 removes the PW label and forwards the Ethernet frame to XRv2. When we check the AToM bindings, we see both PW switching LSRs in the return path, as expected. CSR4 would use PW label 2009 to send traffic back towards CSR2.

```

R4#show l2vpn atom binding 24
Destination Address: 17.0.0.2, VC ID: 24
Local Label: 4009
Cbit: 1,      VC Type: Ethernet,      GroupID: 0
MTU: 1500,    Interface Desc: n/a
VCCV: CC Type: RA [2], TTL [3]
      CV Type: LSPV [2]
Remote Label: 2009
Cbit: 1,      VC Type: Ethernet,      GroupID: 0
MTU: 1500,    Interface Desc: n/a
VCCV: CC Type: TTL [3]
      CV Type: LSPV [2]
PW Switching Point:
  VC ID   Local ID   Remote ID   Description
  12      17.0.0.2   17.0.0.1
  13      17.0.0.1   17.0.0.3

```

We will quickly test the data plane but with one quick modification. CSR1 will be temporarily configured to remark all traffic entering from CSR3 to EXP4. This will allow us to verify if this remarking will be honored despite the PW label swap.

```

! CSR1
policy-map PMAP_REMARK_EXP4
class class-default
set mpls experimental topmost 4

interface GigabitEthernet2.513
service-policy input PMAP_REMARK_EXP4

```

Next, we will enable EPC on CSR1 and CSR2 to capture left-to-right traffic from CSR3 to CSR4. We expect to see EXP bits copied between swapped PW labels and the TTL decremented at each hop as well. XRv1

will send some traffic for analysis. The EXP bits did not get honored, likely because the exp-null label was remarked but was popped shortly thereafter, not swapped. A new exp-null was pushed and the original EXP values were lost. We see incoming labels {0 1005} and outgoing labels {0 2006} as we expect. The TTL was decremented from 255 to 254 on the PW label. Shim headers are in yellow and control-words are in green.

```
RP/0/0/CPU0:XRv1#ping 10.0.0.12
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.12, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/7/19 ms
```

```
R1#show monitor capture CAP buffer detailed
 4 144 1.907002 00:50:56:A9:8C:CF -> 00:50:56:A9:1A:AA MPLS unicast
0000: 005056A9 1AAA0050 56A98CCF 81000DB9 .PV....PV.....
0010: 88470000 00FF003E D1FF0000 00000050 .G.....>.....P
0020: 56A90E6F 005056A9 9C600800 45000064 V..o.PV..`..E..d
0030: 00000000 FF01A782 0A00000B 0A00000C .....

 5 144 1.907002 00:50:56:A9:1A:AA -> 00:50:56:A9:BE:8A MPLS unicast
0000: 005056A9 BE8A0050 56A91AAA 81000DB8 .PV....PV.....
0010: 88470000 00FF007D 61FE0000 00000050 .G.....}a.....P
0020: 56A90E6F 005056A9 9C600800 45000064 V..o.PV..`..E..d
0030: 00000000 FF01A782 0A00000B 0A00000C .....
```

At CSR2, labels {0 2006} arrive and label 4009 goes out. The TTL is decremented from 254 to 253 again during the PW label swap. Notice that CSR4 did not request explicit-null so the packet has one label and thus is 4 bytes smaller.

```
R2#show monitor capture CAP buffer detailed
 2 144 1.042997 00:50:56:A9:1A:AA -> 00:50:56:A9:BE:8A MPLS unicast
0000: 005056A9 BE8A0050 56A91AAA 81000DB8 .PV....PV.....
0010: 88470000 00FF007D 61FE0000 00000050 .G.....}a.....P
0020: 56A90E6F 005056A9 9C600800 45000064 V..o.PV..`..E..d
0030: 00000000 FF01A782 0A00000B 0A00000C .....

 3 140 1.042997 00:50:56:A9:BE:8A -> 00:50:56:A9:2C:57 MPLS unicast
0000: 005056A9 2C570050 56A9BE8A 81000DC4 .PV.,W.PV.....
0010: 884700FA 91FD0000 00000050 56A90E6F .G.....PV..o
0020: 005056A9 9C600800 45000064 00000000 .PV..`..E..d....
0030: FF01A782 0A00000B 0A00000C 08005E4E .....^N
```

I quickly remove LDP explicit-null allocation from CSR1 (not shown) to see if the remarking will be applied to the PW label and thus survive the label swap. This flaps all of the MS-PW sessions, so the actual label values may change, but we won't verify that in detail again. The quick verification is shown below. CSR1 will receive traffic inbound with a single label of 1005, then send {0 2006} out to CSR2.


```
R1#show l2vpn atom vc vcid 13 detail | include local_label
SSO Descriptor: 17.0.0.3/13, local_label: 1005
```

```
R1#show l2vpn atom vc vcid 12 detail | include label_stack
Output interface: Gi2.512, imposed label_stack {0 2006}
```

We retest from XRv11 to XRv12 again and use EPC to capture the packets. My hypothesis held true; removing explicit-null allowed CSR1 to remark the PW label, and these bits persist across the swap. It also guarantees that any new transport labels will have this marking as well. Packets arrive with EXP0 (0x1, S-bit is set) and leave with EXP4. This is 0x8 or binary 1000 in all labels except the bottom, and it's 0x9 in the bottom label which is binary 1001. This is an interesting example because normally explicit-null is meant to sustain SP-core QoS modifications and in this case, it actually did the opposite. Normally, the PW label would have had the proper value from the customer (IP to MPLS imposition) rather than having the topmost exp-null modified on a core router. If we have tested that in the first place, we would have never seen this interesting behavior.

```
RP/0/0/CPU0:XRv1#ping 10.0.0.12
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.12, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/7/19 ms
```

```
R1#show monitor capture CAP buffer detailed
 2  140    0.652011  00:50:56:A9:8C:CF -> 00:50:56:A9:1A:AA MPLS unicast
0000:  005056A9 1AAA0050 56A98CCF 81000DB9  .PV....PV.....
0010:  8847003E D1FF0000 00000050 56A90E6F  .G.>.....PV..o
0020:  005056A9 9C600800 45000064 00000000  .PV..`..E..d....
0030:  FF01A782 0A00000B 0A00000C 08003E4E  .....>N

 3  144    0.652011  00:50:56:A9:1A:AA -> 00:50:56:A9:BE:8A MPLS unicast
0000:  005056A9 BE8A0050 56A91AAA 81000DB8  .PV....PV.....
0010:  88470000 08FF007D 69FE0000 00000050  .G.....}i.....P
0020:  56A90E6F 005056A9 9C600800 45000064  V..o.PV..`..E..d
0030:  00000000 FF01A782 0A00000B 0A00000C  .....
```

Last, we will look at some basic troubleshooting of this feature. Most of it applies to ordinary PWs also, but the introduction of IS-IS levels creates additional complexity. Just like ordinary MPLS L3VPN, summarization between PE loopbacks is generally not supported unless you introduce route leaking, unified MPLS (BGP labeled-unicast), or other methods. The L1L2 routers leak all PEs loopbacks back and forth for full reachability. However, not all remote PEs need all remote loopbacks. CSR3, for example, does not need CSR4 or CSR9 loopbacks since it does not have any MPLS services targeting those addresses, yet it has IS-IS IA routes as the L1L2 leaked them from L2 to L1.

```
R3#show ip route 17.0.0.4
```

```

Routing entry for 17.0.0.4/32
  Known via "isis", distance 115, metric 30, type inter area
  Redistributing via isis 17
  Last update from 17.1.3.1 on GigabitEthernet2.513, 10:31:03 ago
  Routing Descriptor Blocks:
    * 17.1.3.1, from 17.0.0.1, 10:31:03 ago, via GigabitEthernet2.513
      Route metric is 30, traffic share count is 1

```

```

R3#show ip route 17.0.0.9
Routing entry for 17.0.0.9/32
  Known via "isis", distance 115, metric 50, type inter area
  Redistributing via isis 17
  Last update from 17.1.3.1 on GigabitEthernet2.513, 10:30:26 ago
  Routing Descriptor Blocks:
    * 17.1.3.1, from 17.0.0.1, 10:30:26 ago, via GigabitEthernet2.513
      Route metric is 50, traffic share count is 1

```

```

R3#show l2vpn atom vc | begin Interface

```

Interface	Peer ID	VC ID	Type	Name	Status
pw13	17.0.0.1	13	vfi	LAN	UP
pw23	17.0.0.2	23	vfi	LAN	UP
pw36	17.0.0.6	36	vfi	LAN	UP

Adding “null” routes for these destinations on CSR3 should have no effect on the VPLS connectivity. The CE behind CSR3 still has full reachability.

```

! CSR3
ip route 17.0.0.4 255.255.255.255 null0
ip route 17.0.0.9 255.255.255.255 null0

```

```

R3#show ip cef 17.0.0.4
17.0.0.4/32
  attached to Null0

```

```

R3#show ip cef 17.0.0.9
17.0.0.9/32
  attached to Null0

```

```

RP/0/0/CPU0:XRv1#ping 10.0.0.12
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.12, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/7/19 ms

```

```

RP/0/0/CPU0:XRv1#ping 10.0.0.10
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.10, timeout is 2 seconds:
!!!!

```

Success rate is 100 percent (5/5), round-trip min/avg/max = 1/9/19 ms

```
RP/0/0/CPU0:XRv1#ping 10.0.0.5
```

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 10.0.0.5, timeout is 2 seconds:

!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 1/7/19 ms

We will temporarily modify the prefix-lists on CSR1 and CSR6 to stop leaking 17.0.0.2/32 from L2 into L1. We theorize this will cause a problem on CSR3 for at least one PW. We will enable debugging on CSR3 before we continue.

```
R3#debug l2vpn atom vc event
```

```
! CSR1
```

```
no ip prefix-list PL_REMOTE_PE seq 15 permit 17.0.0.2/32
```

```
! CSR6
```

```
no ip prefix-list PL_REMOTE_PE seq 15 permit 17.0.0.2/32
```

```
R1#clear isis rib redistribution
```

Clearing all local RIB

```
R6#clear isis rib redistribution
```

Clearing all local RIB

```
R3#show ip route 17.0.0.2
```

```
% Subnet not in table
```

Within a few seconds, we see the PW fail on CSR3. The MPLS process is reporting a failure to ATOM that there is a problem with the next-hop. The data-plane error (dp-err) is set as the current status, and a syslog is generated to tell us there was a PW error. In case you miss the debug, you can also see this with a show command. Oddly, the new L2VPN syntax doesn't show it, so you must use the legacy command.

```
! CSR3
```

```
AToM[17.0.0.2, 23]: Receive SSM dataplane temporary down notification
```

```
AToM[17.0.0.2, 23]: Evt dataplane down, established->activating
```

```
AToM[17.0.0.2, 23]: . Dataplane de-activated
```

```
AToM[17.0.0.2, 23]: . Set last error: MPLS dataplane reported a fault to the  
nexthop
```

```
AToM[17.0.0.2, 23]: ... Build local PW Status SP info
```

```
AToM[17.0.0.2, 23]: ... Send status update
```

```
AToM[17.0.0.2, 23]: .... Not group switchover capable, cap C000505
```

```
AToM[17.0.0.2, 23]: .... Circuit attributes, Send update:
```

```
AToM[17.0.0.2, 23]: ..... Status: DOWN(pw-rx-fault) (0x10)
```

```
AToM[17.0.0.2, 23]: ..... SP Info len: 104
```

```
AToM[17.0.0.2, 23]: ... NMS: VC oper state: DOWN
```

```

AToM[17.0.0.2, 23]: ... NMS:      err codes:    pw-rx-err
AToM[17.0.0.2, 23]: ... NMS:      : + dp-err
AToM[17.0.0.2, 23]: ... SYSLOG: VC is DOWN, PW Err
AToM[17.0.0.2, 23]: ... PW MIB: VC state is: DOWN
AToM[17.0.0.2, 23]: . Notify dataplane down
AToM: 46 cumulative msgs handled. rc=0
%XCONNECT-5-PW_STATUS: MPLS peer 17.0.0.2 vcid 23, VC state DOWN, PW Err

```

```

R3#show mpls l2transport vc 23 detail | include error
Last error: MPLS dataplane reported a fault to the nexthop
transit packet drops:  receive 1, seq error 0, send 0

```

We test reachability in the core network and see that it still works. This is because of the attached-bit being set in the IS-IS LSP from the L1L2 routers. However, this is not sufficient for MPLS LDP label binding, which is the real issue. We cannot summarize here unless we have a FEC binding for the IP prefix plus MPLS label. We could introduce labeled-unicast BGP (like unified MPLS), or just leak the route from the L1L2 routers.

```

R3#ping 17.0.0.2 source loopback0
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 17.0.0.2, timeout is 2 seconds:
Packet sent with a source address of 17.0.0.3
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 5/7/15 ms

```

```

R3#show ip cef 17.0.0.2 detail
17.0.0.2/32, epoch 2, flags [default route]
  local label info: global/3004
  1 RR source [active source]
    Dependent covered prefix type rr, cover 0.0.0.0/0
    recursive via 0.0.0.0/0
    nexthop 17.1.3.1 GigabitEthernet2.513

```

The process is very interesting on CSR2. We didn't turn on debugging, but we can use the event-trace to help debug the issue. The first thing we notice is a pair of syslog messages. VC 23 went down due to a PW error, which was signaled by CSR3 since the FEC is broken. CSR2 also forces the other PW down, VC 29, since this indicates an issue with the end-to-end path. As such, you must ensure all PWs are operational or else the PW switching LSRs won't bring up either circuit. This is useful because the up/down status is proxied, in a way, across the entire MS-PW.

```

! CSR2
%XCONNECT-5-PW_STATUS: MPLS peer 17.0.0.3 vcid 23, VC state DOWN, PW Err
%XCONNECT-5-PW_STATUS: MPLS peer 17.0.0.9 vcid 29, VC state DOWN

```

Next, we will examine the event-trace for VC 23. On this PW, the remote side is down due to a FEC issue. We can see this in the state-machine code where the “Local” send is “ready/up” (Lru) while the “Remote” end is “ready/down” (Rrd). Overall, the entire PW is down due to a remote PW fault.

```
R2#show l2vpn atom vc vcid 23 event-trace | begin event_trace
AToM VC event trace:
AToM[17.0.0.3, 23]: ..... Send AC STATUS(DOWN(pw-tx-fault))
AToM[17.0.0.3, 23]: ..... No PW status SP info to report
AToM[17.0.0.3, 23]: ..... Send status update
AToM[17.0.0.3, 23]: ..... Not group switchover capable, cap 0
AToM[17.0.0.3, 23]: ..... Circuit attributes, Send update:
AToM[17.0.0.3, 23]: ..... Status: DOWN(pw-tx-fault) (0x20)
AToM[17.0.0.3, 23]: ..... NMS: VC oper state: DOWN
AToM[17.0.0.3, 23]: ..... SYSLOG: VC is DOWN, PW Err
AToM[17.0.0.3, 23]: ..... PW MIB: VC state is: DOWN
AToM[17.0.0.3, 23]: ..... S:Evt ADJ up in LruRrd
AToM[17.0.0.3, 23]: ..... S:Act send SSS(DOWN), notify(UP)
AToM[17.0.0.3, 23]: ..... Remote : DOWN(pw-tx-fault)
AToM[17.0.0.3, 23]: ..... Overall : DOWN(pw-tx-fault)
AToM[17.0.0.3, 23]: ..... [filtered AC]
AToM[17.0.0.3, 23]: ..... Send notify(UP)
AToM[17.0.0.3, 23]: ..... Overall : No fault
AToM[17.0.0.3, 23]: ..... [filtered LDP]
AToM[17.0.0.3, 23]: ... Yes, connected
AToM[17.0.0.3, 23]: .. keeping dataplane up
AToM[17.0.0.3, 23]: .. sync new remote status to standby
```

From the perspective of the other PW, it appears as if the local side is down. VC 29 is treating VC 23 like an AC in this case; since the PWs are switched and one of them is down, it is logically equivalent to the “AC” being down from the perspective of VC 29. We can’t see the state machine codes this time, but we would expect it to indicate that the local side is down and the remote side is up.

```
R2#show l2vpn atom vc vcid 29 event-trace | begin event_trace
AToM VC event trace:
AToM[17.0.0.9, 29]: ..... Local AC : DOWN(pw-tx-fault)
AToM[17.0.0.9, 29]: ..... Overall : DOWN(pw-tx-fault)
AToM[17.0.0.9, 29]: ..... Send LDP for status change from UP
AToM[17.0.0.9, 29]: ..... No PW Status SP info to report to LDP
AToM[17.0.0.9, 29]: ..... Start remote up timer
AToM[17.0.0.9, 29]: ..... NMS: VC oper state: DOWN
AToM[17.0.0.9, 29]: ..... SYSLOG: VC is DOWN
AToM[17.0.0.9, 29]: ..... PW MIB: VC state is: DOWN
AToM[17.0.0.9, 29]: ... Local ready
AToM[17.0.0.9, 29]: .... Local service is ready; send a label
AToM[17.0.0.9, 29]: .... Alloc local binding
AToM[17.0.0.9, 29]: ..... No need to update the local binding
AToM[17.0.0.9, 29]: .... Generate local event
AToM[17.0.0.9, 29]: .... Ready, label 2010
```

```

AToM[17.0.0.9, 29]: .... Evt local ready, in established
AToM[17.0.0.9, 29]: ..... Local ready and established
AToM HA[17.0.0.9, 29]: ..... Checkpointing not allowed
AToM[17.0.0.9, 29]: .. Check if can activate dataplane
AToM[17.0.0.9, 29]: ... Dataplane already active
AToM[17.0.0.9, 29]: Evt remote up timer expired, in established

```

We can verify the state machines again by checking the PW details. For VC 29, which we have not seen, we confirm that the local side is down (Lrd) and the remote side is up (Rrd). There is no issue with CSR9.

```

R2#show l2vpn atom vc vcid 23 detail | include state_machine
      Label/status state machine           : established, LruRrd

R2#show l2vpn atom vc vcid 29 detail | include state_machine
      Label/status state machine           : established, LrdRru

```

We fix the problem by adding the 17.0.0.2/32 network back into the prefix-lists to propagate across levels and the network becomes operational again.

```

! CSR1
ip prefix-list PL_REMOTE_PE seq 15 permit 17.0.0.2/32

! CSR6
ip prefix-list PL_REMOTE_PE seq 15 permit 17.0.0.2/32

R1#clear isis rib redistribution
Clearing all local RIB

R6#clear isis rib redistribution
Clearing all local RIB

```

With debug still enabled on CSR3, we verify the issue is resolved. The MPLS process immediately tells the AToM process that the data-plane is now operational.

```

! CSR3
AToM[17.0.0.2, 23]: Receive SSM dataplane up notification
AToM[17.0.0.2, 23]: Receive SSM dataplane up notification
AToM[17.0.0.2, 23]: Evt dataplane up, activating->established
AToM[17.0.0.2, 23]: . Dataplane activated
AToM[17.0.0.2, 23]: ... No PW status SP info to report
AToM[17.0.0.2, 23]: ... Send status update
AToM[17.0.0.2, 23]: .... Not group switchover capable, cap C000505
AToM[17.0.0.2, 23]: .... Circuit attributes, Send update:
AToM[17.0.0.2, 23]: ..... Status: UP (0x1)
AToM[17.0.0.2, 23]: ... NMS: VC oper state:  UP
AToM[17.0.0.2, 23]: ... SYSLOG: VC is UP
AToM[17.0.0.2, 23]: ... PW MIB: VC state is: UP

```

```

AToM: 47 cumulative msgs handled. rc=0
AToM[17.0.0.2, 23]: Evt dataplane up, in established
AToM: 48 cumulative msgs handled. rc=0
%XCONNECT-5-PW_STATUS: MPLS peer 17.0.0.2 vcid 23, VC state UP

```

We verify that there is an IP route, label, and FEC binding associated with this prefix on CSR3.

```

R3#show ip route 17.0.0.2
Routing entry for 17.0.0.2/32
  Known via "isis", distance 115, metric 20, type inter area
  Redistributing via isis 17
  Last update from 17.1.3.1 on GigabitEthernet2.513, 00:04:16 ago
  Routing Descriptor Blocks:
  * 17.1.3.1, from 17.0.0.1, 00:04:16 ago, via GigabitEthernet2.513
    Route metric is 20, traffic share count is 1

```

```

R3#show mpls ldp bindings 17.0.0.2 32 neighbor 17.0.0.1
  lib entry: 17.0.0.2/32, rev 20
  remote binding: lsr: 17.0.0.1:0, label: 1000

```

```

R3#show mpls forwarding-table 17.0.0.2 32
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id   Switched     interface
3005       1000      17.0.0.2/32    0            Gi2.513    17.1.3.1

```

Syslog messages on CSR2 show that both PWs came up, which implies they are up on both ends. We can quickly check the state machine on CSR2 to confirm.

```

R2#show l2vpn atom vc vcid 23 detail | include state_machine
Label/status state machine           : established, LruRru

R2#show l2vpn atom vc vcid 29 detail | include state_machine
Label/status state machine           : established, LruRru

```

The event-traces will be blank, or nearly so, when there are no faults with the circuits. This makes the event-trace valuable for troubleshooting, sometimes better than debug due to its brevity and ability to filter output by regex.

```

R2#show l2vpn atom vc vcid 23 event-trace | begin event_trace
[no output]

```

```

R2#show l2vpn atom vc vcid 29 event-trace | begin event_trace
AToM VC event trace:
AToM[17.0.0.9, 29]: .. Check if can activate dataplane
AToM[17.0.0.9, 29]: ... Dataplane already active

```

A quick CE to CE ping check verifies an operational switched PW.

```
RP/0/0/CPU0:XRv1#ping 10.0.0.10
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.10, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/11/19 ms
```

Last, we quickly enable sequencing across the MS-PW as this should be supported. Sequence numbers are blindly forwarded through the PW switching LSRs and are not validated there. In fact, we don't even need to enable sequencing on the stitching routers since they do not participate at all.

```
! CSR3
interface pseudowire23
    sequencing both

! CSR9
interface pseudowire29
    sequencing both
```

We verify that sequencing has been enabled and is operational bidirectionally on the endpoints CSR3 and CSR9. We also verify that sequencing is not configured for either PW on the PW switching LSR.

```
R3#show l2vpn atom vc vcid 23 detail | include Seq
Sequencing: receive enabled, send enabled
Sequencing resync disabled
```

```
R9#show l2vpn atom vc vcid 29 detail | in Seq
Sequencing: receive enabled, send enabled
Sequencing resync disabled
```

```
R2#show l2vpn atom vc service-name BRIDGE_PW39 detail | include Dest|Seq
Destination address: 17.0.0.3 VC ID: 23
Sequencing: receive disabled, send disabled
Destination address: 17.0.0.9 VC ID: 29
Sequencing: receive disabled, send disabled
```

Recall that the CW is still negotiated on a PW-by-PW basis, even in MS-PW mode. We need to ensure the CW is supported along the entire MS-PW for sequencing to work. We can quickly check CSR2 since it reports CW status for both local and remote. Since we see the word "on" four times, we can assume it is enabled and supported everywhere.

```
R2#show l2vpn atom vc service-name BRIDGE_PW39 detail | include Dest|Control
Destination address: 17.0.0.3 VC ID: 23
Control word on on
Destination address: 17.0.0.9 VC ID: 29
Control word on on
```


If we disable CW on CSR3 only, the entire MS-PW must reform with CW disabled. This setting must be consistent everywhere, and if a node endpoint does not support it, no segments of the PW can.

```
! CSR3
interface pseudowire23
 control-word exclude

R2#show l2vpn atom vc service-name BRIDGE_PW39 detail | include Dest|Control
Destination address: 17.0.0.3 VC ID: 23
Control word off (configured: enabled) off
Destination address: 17.0.0.9 VC ID: 29
Control word off (configured: enabled) off
```

Before continuing, we enable CW on CSR3 again since we need it for sequencing (just remove the CW exclusion under PW 23 on CSR3). EPC on CSR2 shows the sequence numbers for packets moving in both directions. CSR2 is copying them bidirectionally as expected, which allows the endpoints to process the sequence numbers if configured to do so. Because we capture on both interfaces in both directions, we see that same packet (echo and reply) twice. The CW is highlighted, and the low-order 16 bits (0x0009) represents the sequence numbers. They don't have to be the same in both directions, but can be. The echo request sequence numbers are in yellow and the replies are in green.

```
R2#show monitor capture CAP buffer detailed
 3 544 0.725020 00:50:56:A9:1A:AA -> 00:50:56:A9:BE:8A MPLS unicast
0000: 005056A9 BE8A0050 56A91AAA 81000DB8 .PV....PV.....
0010: 88470000 08FE007D A1FF0000 00090050 .G.....}.....P
0020: 56A9F961 005056A9 9C600800 450001F4 V..a.PV..`.E...
0030: 00000000 FF01A5F4 0A00000B 0A00000A .....

 4 544 0.725020 00:50:56:A9:BE:8A -> 00:50:56:A9:2C:57 MPLS unicast
0000: 005056A9 2C570050 56A9BE8A 81000DC4 .PV.,W.PV.....
0010: 884700FA 50FF0232 A1FD0000 00090050 .G..P..2.....P
0020: 56A9F961 005056A9 9C600800 450001F4 V..a.PV..`.E...
0030: 00000000 FF01A5F4 0A00000B 0A00000A .....

 5 544 0.730009 00:50:56:A9:2C:57 -> 00:50:56:A9:BE:8A MPLS unicast
0000: 005056A9 BE8A0050 56A92C57 81000DC4 .PV....PV.,W....
0010: 88470000 00FD007D 41FF0000 00090050 .G.....}A.....P
0020: 56A99C60 005056A9 F9610800 450001F4 V..`.PV..a..E...
0030: 00000000 FF01A5F4 0A00000A 0A00000B .....

 6 544 0.730009 00:50:56:A9:BE:8A -> 00:50:56:A9:1A:AA MPLS unicast
0000: 005056A9 1AAA0050 56A9BE8A 81000DB8 .PV....PV.....
0010: 8847003E E0FF00BB A1FC0000 00090050 .G.>.....P
0020: 56A99C60 005056A9 F9610800 450001F4 V..`.PV..a..E...
0030: 00000000 FF01A5F4 0A00000A 0A00000B .....
```

We send one more packet to ensure the sequencing increments. It goes from 0x0009 to 0x000A, from 9 to 10, as expected.

```
R2#show monitor capture CAP buffer detailed
 3 544      0.796016 00:50:56:A9:1A:AA -> 00:50:56:A9:BE:8A MPLS unicast
0000: 005056A9 BE8A0050 56A91AAA 81000DB8 .PV....PV.....
0010: 88470000 08FE007D A1FF0000 000A0050 .G.....}.....P
0020: 56A9F961 005056A9 9C600800 450001F4 V..a.PV..`.E...
0030: 00000000 FF01A5F4 0A00000B 0A00000A .....

 4 544      0.796016 00:50:56:A9:BE:8A -> 00:50:56:A9:2C:57 MPLS unicast
0000: 005056A9 2C570050 56A9BE8A 81000DC4 .PV.,W.PV.....
0010: 884700FA 50FF0232 A1FD0000 000A0050 .G..P..2.....P
0020: 56A9F961 005056A9 9C600800 450001F4 V..a.PV..`.E...
0030: 00000000 FF01A5F4 0A00000B 0A00000A .....

 5 544      0.801005 00:50:56:A9:2C:57 -> 00:50:56:A9:BE:8A MPLS unicast
0000: 005056A9 BE8A0050 56A92C57 81000DC4 .PV....PV.,W....
0010: 88470000 00FD007D 41FF0000 000A0050 .G.....}A.....P
0020: 56A99C60 005056A9 F9610800 450001F4 V..`.PV..a..E...
0030: 00000000 FF01A5F4 0A00000A 0A00000B .....

 6 544      0.802012 00:50:56:A9:BE:8A -> 00:50:56:A9:1A:AA MPLS unicast
0000: 005056A9 1AAA0050 56A9BE8A 81000DB8 .PV....PV.....
0010: 8847003E E0FF00BB A1FC0000 000A0050 .G.>.....P
0020: 56A99C60 005056A9 F9610800 450001F4 V..`.PV..a..E...
0030: 00000000 FF01A5F4 0A00000A 0A00000B .....
```

The configurations for MS-PW are in the next section since the EVC rewrite operations use the same topology.

32.1.1.5 EVC rewrite operations

This section is a continuation from the previous section and uses the same topology. A basic AToM PW has been provisioned between PE routers CSR6 and CSR4, with XRv3 and XRv4 as the CE endpoints. The EFP configuration is not shown below since we will be modifying that constantly for these tests. **Pitfall: Changing encapsulation on an EFP will often remove the EFP from the L2VPN xconnect context, so be sure to re-add it.**

```
! CSR6 (CSR4 is identical with different PW peer IP)
interface pseudowire4600
description REWRITE TESTING
encapsulation mpls
neighbor 17.0.0.6 4600
control-word include
signaling protocol ldp

l2vpn xconnect context XC_REWRITE
```

```
member pseudowire4600
member GigabitEthernet2 service-instance 4600
```

We quickly verify the PW is up. We don't expect to have CE to CE reachability since we haven't configured working EFP encapsulation/decapsulation settings yet. The AC is still operational, which allows the PW to come up.

```
R4#show l2vpn service xconnect name XC_REWRITE | begin Interface
  Interface          Group          Encapsulation          Prio   St   XC St
  -----          -
  VPWS name: XC_REWRITE, State: UP
  pw4600              17.0.0.6:4600 (MPLS)      0      UP   UP
  Gi2                  Gi2:3544 (Eth VLAN)      0      UP   UP

R6#show l2vpn service xconnect name XC_REWRITE | begin Interface
  Interface          Group          Encapsulation          Prio   St   XC St
  -----          -
  VPWS name: XC_REWRITE, State: UP
  pw4600              17.0.0.4:4600 (MPLS)      0      UP   UP
  Gi2                  Gi2:3563/1 (Eth VLAN)    0      UP   UP
```

The most obvious and simple way to do EVC configuration is to use a single 802.1Q tag. This means the CE sends tagged frames to the EFP, which removes the tag on ingress, then forwards the frame into the SP network untagged. When using the "symmetric" keyword, the egress operational automatically becomes pushing the specified encapsulation before sending it out the AC towards the customer. Using only one VLAN tag doesn't account for QinQ, which would be entirely transparent to the EFPs since only the first tag is being evaluated. CSR6 uses the common and easier "symmetric" method, while CSR4 manually pops/pushes the tag on ingress/egress respectively. Both of the configurations are operationally identical.

```
! CSR4
service instance 4600 ethernet
 encapsulation dot1q 3544
 rewrite ingress tag pop 1
 rewrite egress tag push dot1q 3544

! CSR6
service instance 4600 ethernet
 encapsulation dot1q 3563
 rewrite ingress tag pop 1 symmetric
```

The encapsulations on the CE devices are updated to use QinQ so that VLAN ID 46 is transparently passed across.

```
! XRv3
interface GigabitEthernet0/0/0/0.563
```

```
encapsulation dot1q 3563 second-dot1q 46
```

```
! XRv4
interface GigabitEthernet0/0/0/0.544
 encapsulation dot1q 3563 second-dot1q 46
```

Because the EFPs did not touch the inner dot1Q tag of 46, it will exist in the payload bridged over MPLS. A quick outbound EPC on CSR4 confirms this. Following the Ethernet header containing the MAC addresses of XRv4 and XRv3 (destination then source), we see the dot1Q ethertype 0x8100 followed by the VLAN ID 46 (0x2E). The remote end, regardless of which configuration is used, will simply push the outer dot1Q tag while leaving the inner one alone. The benefit of this approach is that the customer can change (or remove) its inner tags at any time and the EFP would not need to be reconfigured.

```
R6#show monitor capture CAP buffer detailed
 0 148      0.000000 00:50:56:A9:DE:0D -> 00:50:56:A9:EA:77 MPLS unicast
0000: 005056A9 EA770050 56A9DE0D 81000DEF .PV..w.PV.....
0010: 884701B5 D0FF00FA A1FF0000 00000050 .G.....P
0020: 56A9862A 005056A9 EA548100 002E0800 V...*.PV..T.....
0030: 45000064 00000000 FF01A77E 0A00000D E..d.....~....
```

Another option is to match the inner tag also. This allows for additional granularity (less transparency) so that the customer can use a single dot1Q outer tag for AC transport, then inner tags to differentiate the service. This is a classic use-case for the EV family of Ethernet services (EVPL, EV-LAN, and EV-TREE) since the inner tag can serve as the service identifier while the outer tag is used for transport, as an example. The example below performs similar logic as the previous example. We extend the configuration to account for VLAN ID 46 as an inner tag on both EFPs. In this case, we actually remove the VLAN ID from the frames by popping both tags, then pushing both back. We could have continued to pop the top tag only, but when you are already matching the inner tag, popping it saves you 4 bytes of encapsulation. We already lost the transparency by matching the inner tag, so popping it doesn't hurt. EPC shows just the IP ethertype 0x800 and no VLAN tag in the transported frame. The size of the packet on the wire is 144 bytes, which is 4 bytes less than the 148 packet observed above.

```
! CSR4
service instance 4600 ethernet
 encapsulation dot1q 3544 second-dot1q 46
 rewrite ingress tag pop 2
 rewrite egress tag push dot1q 3544 second-dot1q 46
```

```
! CSR6
service instance 4600 ethernet
 encapsulation dot1q 3563 second-dot1q 46
 rewrite ingress tag pop 2 symmetric
```

```
R6#show monitor capture CAP buffer detailed
 1 144      0.256051 00:50:56:A9:DE:0D -> 00:50:56:A9:EA:77 MPLS unicast
```

```

0000: 005056A9 EA770050 56A9DE0D 81000DEF .PV..w.PV.....
0010: 884701B5 D0FF00FA 21FF0000 00000050 .G.....!.....P
0020: 56A9862A 005056A9 EA540800 45000064 V...*.PV..T..E..d
0030: 00000000 FF01A77E 0A00000D 0A00000E .....~.....

```

You can also match ranges of inner tags to funnel them through a common EFP. However, this adds complexity because now there are certain restrictions. The snippet below attempts to configure CSR6 to place any traffic with an outer tag of 3563 and inner tags of 40 to 50 (inclusive) into an EFP. The issue is that returning traffic will likely need the proper tags pushed back on, but it is now ambiguous as to what the inner tag should be. We cannot push 2 tags on egress anymore using the “symmetric” keyword. Generally you would use this feature with transparency by only popping the topmost (or zero) VLAN tags on both ends, which allows the customer to manage the inner tags (EV family of services).

```

R6(config-if-srv)#encapsulation dot1q 3563 second-dot1q 40 - 50
R6(config-if-srv)#rewrite ingress tag pop 2 symmetric
Rewrite change is not logically valid on ServInst 4600 (Gi2)

```

A similar configuration works on CSR4 only because we are statically pushing tag 46 on egress, even if any other tag came in. CSR4 matches any inner dot1q tag, provided it exists. There is nothing preventing you from popping both tags on ingress, but on egress, pushing “symmetrically” is impossible. This snippet below is syntactically valid but may not actually work since there is asymmetry in the VLAN IDs.

```

R4(config-if-srv)#encapsulation dot1q 3544 second-dot1q any
R4(config-if-srv)#rewrite ingress tag pop 2
R4(config-if-srv)#rewrite egress tag push dot1q 3544 second-dot1q 46

```

For the next test, we will create a hybrid solution. This is not an ideal solution, but it demonstrates the power of the tag rewrite feature. As traffic comes in on CSR6, it will be tagged {3563 46} which will match EFP 4600. CSR6 does absolutely nothing to this tag stack on ingress; it encapsulates the Ethernet frame in MPLS and sends it into the core. CSR4 will receive this traffic and would normally reject it because VLAN 3563 does not match any local EFPs. The outer tag differs at the two sites, so CSR4 can translate this top tag to the value it has locally configured, which is 3544. The inner tag is not touched on egress. This allows CSR4 to deliver traffic to XRv4 with tag stack {3544 46}, which is correct. Returning traffic enters CSR4 and has the topmost tag removed, so only the inner tag of 46 will be transported over MPLS. When CSR6 receives the MPLS packets, it adds the proper outer tag and sends it towards XRv3 with stack {3563 46}.

```

! CSR6
service instance 4600 ethernet
 encapsulation dot1q 3563 second-dot1q 40-50
 rewrite egress tag push dot1q 3563

! CSR4
service instance 4600 ethernet

```

```
encapsulation dot1q 3544 second-dot1q 1-4094
rewrite ingress tag pop 1
rewrite egress tag translate 1-to-1 dot1q 3544
```

We enable bidirectional EPC on CSR6 to view the encapsulation of the VLAN tags. The request and response are about 3ms apart (yellow) according to this capture and have 8 bytes different in length. The MPLS labels are highlighted in green, which we don't care about right now, but since CSR7 performs PHP for returning traffic, inbound PW traffic has only one label. This accounts for 4 of the bytes. The remaining 4 byte difference is highlighted in cyan. The outgoing packet contains tag stack {3563 46} as discussed earlier since CSR6 makes no VLAN tag modifications on ingress. CSR4 rewrites the top tag on egress to deliver the proper tag stack to Xrv4. Inbound traffic on CSR4's EFP pops the top tag on ingress, which means only the inner tag of 46 is present on the return flow. This accounts for the remaining 4 bytes. CSR6 pushes tag 3563 back onto the stack before delivering traffic to Xrv3.

```
R6#show monitor capture CAP buffer detail
 1 152    1.206991  00:50:56:A9:DE:0D -> 00:50:56:A9:EA:77 MPLS unicast
0000: 005056A9 EA770050 56A9DE0D 81000DEF .PV..w.PV.....
0010: 884701B5 D0FF00FA A1FF0000 00000050 .G.....P
0020: 56A9862A 005056A9 EA548100 0DEB8100 V...*.PV..T.....
0030: 002E0800 45000064 00000000 FF01A77E ....E..d.....~

 2 144    1.209981  00:50:56:A9:EA:77 -> 00:50:56:A9:DE:0D MPLS unicast
0000: 005056A9 DE0D0050 56A9EA77 81000DEF .PV....PV..w....
0010: 88470177 B1FD0000 00000050 56A9EA54 .G.w.....PV..T
0020: 005056A9 862A8100 002E0800 45000064 .PV...*.....E..d
0030: 00000000 FF01A77E 0A00000E 0A00000D .....~.....
```

Alternatively, we could rewrite the tag on ingress on CSR6 to change 3563 to 3544 so CSR4 doesn't have to do it. This might be a good idea if there are multiple routers using outer tag 3544 in the network and this EFP is tied to a multipoint L2VPN service; it is loosely analogous to MVPN extranet source-side chaining (SSC) versus receiver-side chaining (RSC). Each option has its benefits but the selection depends largely on design goals. For additional complexity, we change the inner tag on Xrv4 to 460, which means additional popping or translation is required. We haven't changed the EFP "match" criteria on either router, which is the encapsulation, but we do change the tag handling. On CSR6 ingress, both tags are rewritten from {3563 40-50} to {3544 460}, which means CSR6 did all the work for CSR4. This is sloppy because any inner tag between 40 and 50 is rewritten to 460 no matter what, but technically it does work. CSR4 shouldn't have to do anything on egress now. Returning traffic will be send over MPLS as untagged since CSR4 blindly pops both tags. As such, CSR6 will have to push both tags back onto the stack. These solutions are complex because the EFP matches multiple inner tags yet pushes a static inner tag on egress; this is just to demonstrate the feature.

```
! CSR6
service instance 4600 ethernet
 encapsulation dot1q 3563 second-dot1q 40-50
 rewrite ingress tag translate 2-to-2 dot1q 3544 second-dot1q 460
```

```

rewrite egress tag push dot1q 3563 second-dot1q 46

! CSR4
service instance 4600 ethernet
  encapsulation dot1q 3544 second-dot1q 1-4094
  rewrite ingress tag pop 2

! XRv4
interface GigabitEthernet0/0/0/0.544
  encapsulation dot1q 3544 second-dot1q 460

```

EPC on CSR6 shows the tag stack {3544 460} exiting towards the MPLS core and an untagged frame arriving in reply. The difference in length is 12 bytes: 4 from the lack of a transport MPLS label (green), and 8 from the lack of two dot1q tags (cyan) that were present on the ICMP requests but not the replies (XRv3 to XRv4).

```

R6#show monitor capture CAP buffer detail
  0  152  0.000000  00:50:56:A9:DE:0D -> 00:50:56:A9:EA:77 MPLS unicast
0000:  005056A9 EA770050 56A9DE0D 81000DEF  .PV..w.PV.....
0010:  884701B5 D0FF00FA A1FF0000 00000050  .G.....P
0020:  56A9862A 005056A9 EA548100 0DD88100  V...*.PV..T.....
0030:  01CC0800 45000064 00000000 FF01A77E  ....E..d.....~

  1  140  0.003997  00:50:56:A9:EA:77 -> 00:50:56:A9:DE:0D MPLS unicast
0000:  005056A9 DE0D0050 56A9EA77 81000DEF  .PV....PV..w....
0010:  88470177 B1FD0000 00000050 56A9EA54  .G.w.....PV..T
0020:  005056A9 862A0800 45000064 00000000  .PV...*..E..d....
0030:  FF01A77E 0A00000E 0A00000D 0000C677  ...~.....w

```

We can also introduce 802.1ad with EFPs. This is basically QinQ except it represents the standardized version. It uses ethertype 0x88A8 for the outer tag versus the standard dot1Q ethertype of 0x8100. The specification also introduces some basic naming conventions: C-Tag and S-tag represent the customer and service tags, respectively, where the latter is imposed/reviewed by the SP for transit across the core. “Tag 1” and “tag 2” relate to the C-tag and S-tag respectively, which is just an alternative naming convention. In this example, we configure a subtle change on CSR6; it will now rewrite the topmost tag to a dot1ad tag of 3544 versus a dot1q tag with the same value. Because CSR4 was not taking any egress actions in the previous example, we also need it to translate this dot1ad tag back to a dot1q tag (essentially just changing the ethertype from 0x88A8 to 0x8100) while maintaining the proper S-tag, or tag 1.

```

! CSR6
service instance 4600 ethernet
  encapsulation dot1q 3563 second-dot1q 40-50
  rewrite ingress tag translate 2-to-2 dot1ad 3544 dot1q 460
  rewrite egress tag push dot1q 3563 second-dot1q 46

```

```

! CSR4
service instance 4600 ethernet
 encapsulation dot1q 3544 second-dot1q 1-4094
 rewrite ingress tag pop 2
 rewrite egress tag translate 1-to-1 dot1q 3544

```

An EPC capture looks almost identical to the capture above, except the first packet's S-tag (tag 1) has an ethertype of 0x88A8 to indicate 802.1ad. The inner tag is still dot1q (0x8100). Nothing else has changed, and the returning traffic is untagged entirely as it was before.

```

R6#show monitor capture CAP buffer detail
 6 152      1.376039 00:50:56:A9:DE:0D -> 00:50:56:A9:EA:77 MPLS unicast
0000: 005056A9 EA770050 56A9DE0D 81000DEF .PV..w.PV.....
0010: 884701B5 D0FF00FA A1FF0000 00000050 .G.....P
0020: 56A9862A 005056A9 EA5488A8 0DD88100 V...*.PV..T.....
0030: 01CC0800 45000064 00010000 FF01A77D ....E..d.....}

 7 140      1.380037 00:50:56:A9:EA:77 -> 00:50:56:A9:DE:0D MPLS unicast
0000: 005056A9 DE0D0050 56A9EA77 81000DEF .PV....PV..w....
0010: 88470177 B1FD0000 00000050 56A9EA54 .G.w.....PV..T
0020: 005056A9 862A0800 45000064 00010000 .PV...*..E..d....
0030: FF01A77D 0A00000E 0A00000D 00009676 ...}.....v

```

We can continue testing with 802.1ad and introduce some new 802.1Q modifiers also. When frames arrive on CSR6, we can remove double-tagged dot1q frames and replace it with a single dot1ad header if we choose. The remote end (CSR4) can translate this same VLAN tag back to a dot1q header for the customer. In this example, we configure XRv4 to stop doing QinQ and only specify a single tag for the connection to the AC. CSR4 can match this single-tagged traffic using the “exact” keyword, which ensures no double-tagged frames can match CSR4’s EFP on ingress. CSR4 removes this top dot1q tag and continues to send untagged frames over MPLS to CSR6, who expects it and pushes the two required dot1q tags for the XRv3.

```

! CSR6
service instance 4600 ethernet
 encapsulation dot1q 3563 second-dot1q 40-50
 rewrite ingress tag translate 2-to-1 dot1ad 3544
 rewrite egress tag push dot1q 3563 second-dot1q 46

```

```

! CSR4
service instance 4600 ethernet
 encapsulation dot1q 3544 exact
 rewrite ingress tag pop 1
 rewrite egress tag translate 1-to-1 dot1q 3544

```

```

! XRv4
interface GigabitEthernet0/0/0/0.544
 encapsulation dot1q 3544

```


We can verify this behavior using EPC. Now there is a single dot1ad (0x88A8) tag with value 3544 (0xDD8) on the ICMP request frames while the return traffic remains untagged as the ethertype of 0x0800 (IPv4) is highlighted. The difference in length is only 8 now: one MPLS label (green) and one VLAN tag.

```
R6#show monitor capture CAP buffer detail
  1  148      0.027999  00:50:56:A9:DE:0D -> 00:50:56:A9:EA:77 MPLS unicast
0000:  005056A9 EA770050 56A9DE0D 81000DEF  .PV..w.PV.....
0010:  884701B5 D0FF00FA B1FF0000 00000050  .G.....P
0020:  56A9862A 005056A9 EA5488A8 0DD80800  V...*.PV..T.....
0030:  45000064 00000000 FF01A77E 0A00000D  E..d.....~....

  2  140      0.031004  00:50:56:A9:EA:77 -> 00:50:56:A9:DE:0D MPLS unicast
0000:  005056A9 DE0D0050 56A9EA77 81000DEF  .PV....PV..w....
0010:  88470177 B1FD0000 00000050 56A9EA54  .G.w.....PV..T
0020:  005056A9 862A0800 45000064 00000000  .PV...*.E..d....
0030:  FF01A77E 0A00000E 0A00000D 00006677  ...~.....fw
```

We can specify the payload ethertype from a basic list, and also have different EFPs for different CoS values. Without changing CSR4, we direct CSR6 to only support IPv4 traffic (0x0800). This means our basic ping from XRv3 to XRv4 works for IPv4 only. Other ethertype options include IPv6 (0x86DD), PPPoE discovery (0x8863), PPPoE session (0x8864), or PPPoE all (both 0x8863 and 0x8864).

```
! CSR6
service instance 4600 ethernet
 encapsulation dot1q 3563 second-dot1q 40-50 etype ipv4
 rewrite ingress tag translate 2-to-1 dot1ad 3544
 rewrite egress tag push dot1q 3563 second-dot1q 46

RP/0/0/CPU0:XRv3#ping 10.0.0.14
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.14, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/9/29 ms

RP/0/0/CPU0:XRv3#ping 2001:10::14
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2001:10::14, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
```

If we add IPv6 (0x86DD) to the list, it works as well. This is how we can separate IPv4 and IPv6 traffic into different EFPs if we choose. Using this example, adding a new EFP to match “pppoe-all” would effectively segregate PPPoE from IPv4/v6 traffic, which might be desirable.

```
! CSR6
```

```

service instance 4600 ethernet
  encapsulation dot1q 3563 second-dot1q 40-50 etype ipv4 , ipv6
  rewrite ingress tag translate 2-to-1 dot1ad 3544
  rewrite egress tag push dot1q 3563 second-dot1q 46

```

```

RP/0/0/CPU0:XRv3#ping 2001:10::14
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2001:10::14, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/8/19 ms

```

Next, we will specify only CoS 0 traffic can enter this EFP. Everything still works, but this short ping test is a false positive. As soon as we clear the IPv6 ND cache, IPv6 is broken with this new configuration. This would also obviously break any other control traffic that marks its traffic at layer 2. IPv4 ARP is automatically accounted for when 0x0800 (IPv4) is permitted.

```

! CSR6
service instance 4600 ethernet
  encapsulation dot1q 3563 second-dot1q 40-50 cos 0 etype ipv4 , ipv6
  rewrite ingress tag translate 2-to-1 dot1ad 3544
  rewrite egress tag push dot1q 3563 second-dot1q 46

```

```

RP/0/0/CPU0:XRv3#ping 10.0.0.14
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.14, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/7/19 ms

```

```

RP/0/0/CPU0:XRv3#ping 2001:10:0:0::14
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2001:10::14, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 9/19/29 ms

```

```

RP/0/0/CPU0:XRv3#clear ipv6 neighbors gigabitEthernet 0/0/0/0.563

```

```

RP/0/0/CPU0:XRv3#ping 2001:10:0:0::14
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2001:10::14, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)

```

Be careful when using the IPv6 ethertype filter and CoS at the same time. Unlike IPv4 ARP, which is a separate ethertype, IPv6 ND relies on IPv6-ICMP and uses CoS 6 (binary 110x) in the Ethernet frame, shown in yellow. This immediately precedes the dot1ad VLAN ID which is 3544 (cyan). IPv6 ND also uses CS7 in the IPv6 header (green), but that isn't relevant here, it's shown for completeness. The first two packets show an IPv6 ND exchange, and have priority values set. The return packet doesn't show the

priority bits because they were popped by the EFP at CSR4 on ingress, but they would have been there had the VLAN tag been allowed into the SP core. The following packets show the ping session with no QoS values set (0 for layer 2 and layer 3 headers). This capture was taken with the “cos” keyword removed from the EFP configuration, because this traffic would have never arrived at the EFP otherwise.

```
R6#show monitor capture CAP buffer detail
 2  120      0.797007  00:50:56:A9:DE:0D -> 00:50:56:A9:EA:77 MPLS unicast
0000:  005056A9 EA770050 56A9DE0D 81000DEF .PV..w.PV.....
0010:  884701B5 DCFF00FA BDFF0000 00003333 .G.....33
0020:  FF000014 005056A9 EA5488A8 CDD886DD .....PV..T.....
0030:  6E000000 00203AFF 20010010 00000000 n.... :. ....

 3  112      0.802012  00:50:56:A9:EA:77 -> 00:50:56:A9:DE:0D MPLS unicast
0000:  005056A9 DE0D0050 56A9EA77 81000DEF .PV....PV..w....
0010:  88470177 B1FD0000 00000050 56A9EA54 .G.w.....PV..T
0020:  005056A9 862A86DD 6E000000 00203AFF .PV..*..n.... :.
0030:  20010010 00000000 00000000 00000014 .....

 8  148      0.811014  00:50:56:A9:DE:0D -> 00:50:56:A9:EA:77 MPLS unicast
0000:  005056A9 EA770050 56A9DE0D 81000DEF .PV..w.PV.....
0010:  884701B5 D0FF00FA B1FF0000 00000050 .G.....P
0020:  56A9862A 005056A9 EA5488A8 0DD886DD V...*.PV..T.....
0030:  60000000 003C3A3C 20010010 00000000 `....<:< .....

 9  140      0.816004  00:50:56:A9:EA:77 -> 00:50:56:A9:DE:0D MPLS unicast
0000:  005056A9 DE0D0050 56A9EA77 81000DEF .PV....PV..w....
0010:  88470177 B1FD0000 00000050 56A9EA54 .G.w.....PV..T
0020:  005056A9 862A86DD 60000000 003C3A40 .PV..*..`....<:@
0030:  20010010 00000000 00000000 00000014 .....

```

A simple solution is to allow CoS 0 and 6, but nothing in between. This would allow low priority traffic, plus relevant control traffic, to utilize this EFP. Alternatively, we could statically configure IPv6 ND peers, but this is sloppy and doesn’t allow for other IPv6 ND communications unrelated to LAN discovery, such as NUD probes, redirects, etc.

```
! CSR6
service instance 4600 ethernet
 encapsulation dot1q 3563 second-dot1q 40-50 cos 0,6 etype ipv4 , ipv6
 rewrite ingress tag translate 2-to-1 dotlad 3544
 rewrite egress tag push dot1q 3563 second-dot1q 46

```

```
RP/0/0/CPU0:XRv3#ping 10.0.0.14
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.14, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/7/19 ms

```

```
RP/0/0/CPU0:XRv3#ping 2001:10::14
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2001:10::14, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/10/19 ms
```

For completeness, the EFP show commands are shown below. They basically give the same information as the running-config, plus some details regarding default values. The summary and stats commands are components of the detailed command, but they all show the same general information. This is one of the rare circumstances where the running-config is the best way to verify EFP state. The only exception is the packet counters shown below, which can be valuable when troubleshooting tag mismatches.

```
R6#show ethernet service instance id 4600 interface gigabitEthernet 2
Identifier   Type      Interface           State    CE-Vlans
4600         Static    GigabitEthernet2    Up

R6#show ethernet service instance id 4600 interface gigabitEthernet 2 stats
Port maximum number of service instances: 8000
Service Instance 4600, Interface GigabitEthernet2
  Pkts In   Bytes In   Pkts Out   Bytes Out
    296     34102     202       23230

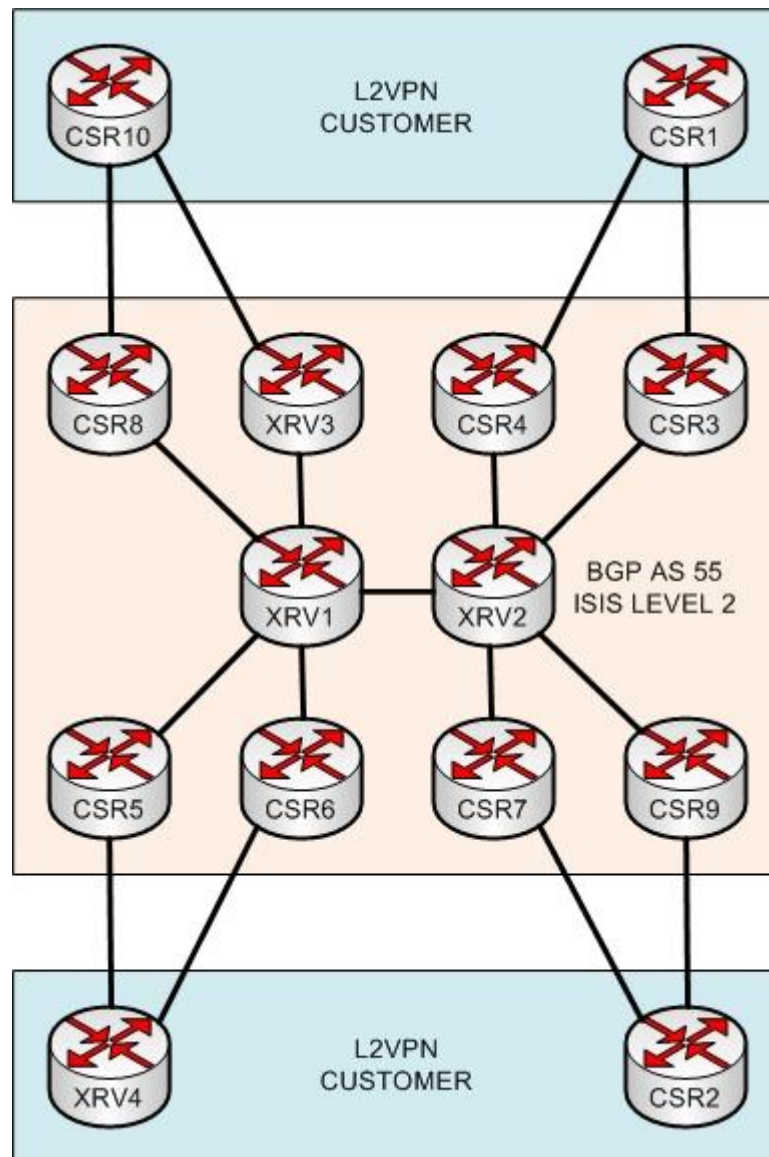
R6#show ethernet service instance id 4600 interface gigabitEthernet 2 detail
Service Instance ID: 4600
Service Instance Type: Static
Associated Interface: GigabitEthernet2
Associated EVC:
L2protocol drop
CE-Vlans:
Encapsulation: dot1q 3563 vlan protocol type 0x8100 second-dot1q 40-50 vlan
protocol type 0x8100 cos 0,6 payload etype ipv4 , ipv6
Rewrite: ingress tag translate 2-to-1 dot1ad 3544 vlan-type 0x88A8
Rewrite: egress tag push dot1q 3563 vlan-type 0x8100 second-dot1q 46 vlan-
type 0x8100
Interface Dot1q Tunnel Ethertype: 0x8100
State: Up
EFP Statistics:
  Pkts In   Bytes In   Pkts Out   Bytes Out
    296     34102     202       23230
```

Additional Reading - Reference configurations "l2vpn-mspw-rewrite"

32.1.2 BGP auto-discovery for VPWS/VPLS

This section uses the same basic setup as the static VPLS lab. The static VPWS (E-LINE) configuration is left in place just to show that all of the different PW options can coexist. VPWS autodiscovery is not specifically shown because it is identical to VPLS (E-LAN) with only two endpoints, and for efficiency, would have MAC address learning disabled on the corresponding bridge-domain (BD). The static VPLS (E-

LAN and E-TREE) configuration has been removed as those will be replaced with BGP auto-discovery mechanisms.



32.1.2.1 LDP signaling

LDP signaling is traditionally a Cisco mechanism for signaling BGP-discovered PWs within a VPLS instance. In a sentence, the VPLS endpoints are advertised into BGP, learned by other endpoints within the VPN, and then sessions are created between peers. This procedure relies on route-targets just like L3VPN, but there are some “easy” and automatic enhancements to it. E-LAN is configured on CSR4, CSR6, CSR7, and XRV3 just like last time, except now using LDP for signaling paired with BGP AD. The BD and EFP configurations are not shown again as they are independent of the VPLS configuration; they are identical to the static VPLS examples. CSR4 and CSR6 use the new syntax while CSR7 uses the legacy

syntax. Notice that none of the XE routers specify any RD/RT information, but XRv3 does. This is discussed in detail later.

```
! CSR4 and CSR6
l2vpn vfi context LAN
  vpn id 3467
  autodiscovery bgp signaling ldp

! CSR7
l2 vfi LAN autodiscovery
  vpn id 3467
  bridge-domain 3527

! XRv3
l2vpn
  logging
  pseudowire

bridge group LAN
  bridge-domain LAN
  interface GigabitEthernet0/0/0/0.503
  vfi LAN
    vpn-id 3467
    autodiscovery bgp
    rd auto
    route-target import 55:3467
    route-target export 55:3467
    signaling-protocol ldp
```

We also must configure BGP to exchange these L2VPN routes. XRv1 will be a RR for this AF and all other routers will peer with it. XR uses negative logic in its signaling mechanisms for BGP peers, which can be confusing. Rather than specify the desire to use BGP or LDP signaling, XR requires the administrator to specify which one NOT to use. In this case, we specify that BGP signaling should be disabled, which implies LDP signaling is enabled.

```
! XRv1
router bgp 55
  bgp cluster-id 55.0.0.11
  address-family l2vpn vpls-vpws

  af-group AF_L2_LDP address-family l2vpn vpls-vpws
    route-reflector-client
    Signalling bgp disable

  session-group SES_IBGP
    remote-as 55
    password encrypted 112B3E3528333E382C
    update-source Loopback0
```

```

    session-open-mode passive-only

neighbor-group NBR_L2_LDP
    use session-group SES_IBGP
    address-family l2vpn vpls-vpws
    use af-group AF_L2_LDP
neighbor 55.0.0.4
    use neighbor-group NBR_L2_LDP

neighbor 55.0.0.6
    use neighbor-group NBR_L2_LDP

neighbor 55.0.0.7
    use neighbor-group NBR_L2_LDP

neighbor 55.0.0.13
    use neighbor-group NBR_L2_LDP

! CSR4, CSR6, CSR7
router bgp 55
    bgp log-neighbor-changes
    no bgp default ipv4-unicast
    neighbor 55.0.0.11 remote-as 55
    neighbor 55.0.0.11 transport connection-mode active
    neighbor 55.0.0.11 password BGP_AUTH
    neighbor 55.0.0.11 update-source Loopback0
    address-family l2vpn vpls
        neighbor 55.0.0.11 activate

! Xrv3
router bgp 55
    address-family l2vpn vpls-vpws
    neighbor 55.0.0.11
        remote-as 55
        password encrypted 112B3E3528333E382C
        update-source Loopback0
        session-open-mode active-only
    address-family l2vpn vpls-vpws
        Signalling bgp disable

```

The first thing we notice is that the BGP sessions will not come up with the XE routers. CSR7, as an example, generates a strange log message alerting us to the problem. The session forms (so we know it isn't a password issue, for example) but as soon as Xrv1 receives a "route", it closes the session.

```

! CSR7
%BGP-5-ADJCHANGE: neighbor 55.0.0.11 Up
%BGP-5-NBR_RESET: Neighbor 55.0.0.11 reset (Peer closed the session)

```

```
%BGP-3-NOTIFICATION: received from neighbor 55.0.0.11 3/10 (illegal network)
1 bytes 60
%BGP_SESSION-5-ADJCHANGE: neighbor 55.0.0.11 L2VPN Vpls topology base removed
from session Peer closed the session
%BGP-5-ADJCHANGE: neighbor 55.0.0.11 Down Peer closed the session

! XRv1
bgp[1052]: %ROUTING-BGP-5-ADJCHANGE : neighbor 55.0.0.7 Up (VRF: default)
(AS: 55)
bgp[1052]: %ROUTING-BGP-5-ADJCHANGE : neighbor 55.0.0.7 Down - BGP
Notification sent, illegal network (VRF: default) (AS: 55)
```

Debugging on CSR7 and XRv1, unfortunately, does not help much. XRv1 indicates that CSR7 sent a malformed message, but the low-level details prompt additional research.

```
! CSR7
BGP(9): 55.0.0.11 NEXT_HOP self is set for sourced RT Filter for net
55:3467:55.0.0.7/96
BGP(9): (base) 55.0.0.11 send UPDATE (format) 55:3467:55.0.0.7/96, next
55.0.0.7, metric 0, path Local, extended community RT:55:3467 L2VPN
AGI:55:3467

! XRv1
bgp[1052]: %ROUTING-BGP-3-MALFORM_UPDATE : Malformed UPDATE message received
from neighbor 55.0.0.7 (VRF: default) - message length 88 bytes, error flags
0x00000008, action taken "Reset". Error details: "Error 0x00000008, Field
"MP_REACH_NLRI", Attribute 14 (Flags 0x00, Length 0), Data [60]". NLRI is:
[IPv4 Unicast]
```

This is a strange issue where referencing several Cisco documents and RFCs is required. Section 7 of RFC 6074, "BGP-AD and VPLS-BGP Interoperability," states: *"Both BGP-AD and VPLS-BGP [RFC4761] use the same AFI/SAFI. In order for both BGP-AD and VPLS-BGP to co-exist, the NLRI length must be used as a demultiplexer. The BGP-AD NLRI has an NLRI length of 12 bytes, containing only an 8-byte RD and a 4-byte VSI-ID. VPLS-BGP [RFC4761] uses a 17-byte NLRI length. Therefore, implementations of BGP-AD must ignore NLRI that are greater than 12 bytes."*

RFC 4761 specifies BGP signaling (VPLS-BGP) and RFC 4762 specifies LDP signaling (BGP-AD). The feature naming is confusing since BGP-AD is always used for discovery, while the signaling protocol varies. In either case, the AFI/SAFI is the same despite the very different signaling protocols. The original implementation of Cisco BGP-AD only used one byte for the prefix length, but we must set it to 2 on all XE PEs. This is what XR expects and is not required for BGP sessions between XE peers.

```
! All XE PEs
router bgp 55
 address-family l2vpn vpls
  neighbor 55.0.0.11 prefix-length-size 2
```


Now, we can see the neighbor come up for this specific capability. Nowhere in the capability does it specify LDP or BGP signaling, which is why the previous command was necessary. We also check the RR to ensure all the other sessions are up.

```
R7#show bgp l2vpn vpls all neighbors 55.0.0.11 | include L2VPN
Address family L2VPN Vpls: advertised and received
For address family: L2VPN Vpls

RP/0/0/CPU0:XRv1#show bgp l2vpn vpls neighbors | utility egrep '^BGP|L2VPN'
BGP neighbor is 55.0.0.4
Address family L2VPN VPLS: advertised and received
For Address Family: L2VPN VPLS
BGP neighbor is 55.0.0.6
Address family L2VPN VPLS: advertised and received
For Address Family: L2VPN VPLS
BGP neighbor is 55.0.0.7
Address family L2VPN VPLS: advertised and received
For Address Family: L2VPN VPLS
BGP neighbor is 55.0.0.13
Address family L2VPN VPLS: advertised and received
For Address Family: L2VPN VPLS
```

Next, we can examine the routes. We will look at CSR6 first. We see the routes have a 96 bit length, which is the result of the RD (64 bit) and the IPv4 address (32) bit. We never configured an RD on any of the XE routers, so it automatically creates an RD as BGP_ASN:VPN_ID. Notice that XR required manual RD configuration; an “auto” feature exists but for consistency, I manually set the RD to be the same as the other VPLS endpoints. The address contained in the prefix header is the BGP next-hop, which is the Loopback0 in this case.

```
R6#show bgp l2vpn vpls all | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 55:3467
*>i 55:3467:55.0.0.4/96
      55.0.0.4      0      100      0 ?
*> 55:3467:55.0.0.6/96
      0.0.0.0      32768 ?
*>i 55:3467:55.0.0.7/96
      55.0.0.7      0      100      0 ?
*>i 55:3467:55.0.0.13/96
      55.0.0.13      100      0 i
```

Digging into the route from CSR4, we see some familiar components. The RT, for example, is set to 55:3467, which again appears automatic. There is an XE feature called “auto-route-target” which is on by default; it assumes that all VPLS endpoints in the same VPN-ID want to communicate with one another. This is a great assumption for E-LAN but not for E-TREE; we will look at disabling this and

manually defining RT import/export policies for E-TREE designs later. Again, XR needs the RT policies to be manually configured, as shown above. The AGI is the Attachment Group Identifier and is used to group forwarders together for signaling purposes. Essentially it defines endpoints that can be connected via pseudowires; it defines the VPN. The BGP details also show MPLS labels despite us explicitly using LDP. Remember that the exp-null label number is 0, so the label fields are all zeroes in the update, which explains the output.

```
R6#show bgp l2vpn vpls rd 55:3467 55.0.0.4
BGP routing table entry for 55:3467:55.0.0.4/96, version 5
Paths: (1 available, best #1, table L2VPN-VPLS-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    55.0.0.4 (metric 30) from 55.0.0.11 (55.0.0.11)
      Origin incomplete, metric 0, localpref 100, valid, internal, best, AGI
version(268435458)
      Extended Community: RT:55:3467 L2VPN AGI:55:3467
      Originator: 55.0.0.4, Cluster list: 55.0.0.11
      mpls labels in/out exp-null/exp-null
      rx pathid: 0, tx pathid: 0x0
```

The AGI is essentially the ASN concatenated with the VPN ID and is used to determine the VC number for the PWs that are dynamically created (constructed just like the RD and RT). **Regardless of the RT import/export policies, the VPN ID (AGI) must match for the VPN to form.** It is an extended community just like the RT and is not “imported” unless the AGI is consistent. Below, we see CSR6 sending an update with AGI 55:999 since I changed its VPN ID to 999 temporarily (not shown). When it tries to import CSR4’s route with AGI:55:3467, it is rejected, much like an L3VPN route with incorrect RTs.

```
! CSR6
BGP(9): (base) 55.0.0.11 send UPDATE (format) 55:999:55.0.0.6/96, next
55.0.0.6, metric 0, path Local, extended community RT:55:999 L2VPN AGI:55:999
BGP(9): 55.0.0.11 rcvd 55:3467:55.0.0.4/96 -- DENIED due to: extended
community not supported;
```

After correcting the error on CSR6, all the routes are imported again. The result of importing these routes instructs the VPLS process to create a PW towards each endpoint. Notice that all of the dynamic PWs have the same VCID, which is the VPN ID. For clarity, the VPN ID and AGI are not the same thing, but the AGI is derived from the VPN ID by default. The RD/RT are also derived from the VPN ID as seen earlier. The RD, RT, and AGI can all be manually adjusted despite this automatic inheritance, which is discussed later.

```
R6#show l2vpn atom vc vcid 3467
```

Interface	Peer ID	VC ID	Service Type	Name	Status
-----	-----	-----	-----	-----	-----

pw100018	55.0.0.4	3467	vfi	LAN	UP
pw100019	55.0.0.7	3467	vfi	LAN	UP
pw100020	55.0.0.13	3467	vfi	LAN	UP

To select a specific PW for analysis, you must specify the destination address followed by the VCID. We can look at the details for the connection to CSR4. We have our usual fields like VC ID, label stack, and signaling protocol, which are important to note. First interesting to notice is the AGI and AII fields. AII is the Attachment Individual Identifier and is meant to identify separate components within an AGI. In this case, the IP addresses are decoded from hex to show the local and remote AIs. Above that is the AGI in hex (XE does not politely decode this for us), which includes 0x37 (55) and 0xD8B (3467). We can see that status and the CW were negotiated as well. Interestingly, VCCV claims that is not capable of using the PW-ACH method (using the CW, even though it was negotiated). We will investigate that later.

```
R6#show l2vpn atom vc destination 55.0.0.4 vcid 3467 detail
pseudowire100018 is up, VC status is up PW type: Ethernet
  Create time: 00:02:40, last status change time: 00:02:39
  Last label FSM state change time: 00:02:39
  Destination address: 55.0.0.4 VC ID: 3467
  Output interface: Gi2.561, imposed label stack {91003 4016}
  Preferred path: not configured
  Default path: active
  Next hop: 55.6.11.11
  Member of vfi service LAN
  Bridge-Domain id: 3467
  Service id: 0xc1000002
  Signaling protocol: LDP, peer 55.0.0.4:0 up
    Targeted Hello: 55.0.0.6(LDP Id) -> 55.0.0.4, LDP is UP
    Graceful restart: not configured and not enabled
    Non stop routing: not configured and not enabled
    Generalized PWid FEC (129)
    AGI: type 1, len 8, 000A 0037 0000 0D8B
    Local AII: type 1, len 4, 3700 0006 (55.0.0.6)
    Remote AII: type 1, len 4, 3700 0004 (55.0.0.4)
    Status TLV support (local/remote) : enabled/supported
      LDP route watch : enabled
      Label/status state machine : established, LruRru
      Local dataplane status received : No fault
      BFD dataplane status received : Not sent
      BFD peer monitor status received : No fault
      Status received from access circuit : No fault
      Status sent to access circuit : No fault
      Status received from pseudowire i/f : No fault
      Status sent to network peer : No fault
      Status received from network peer : No fault
      Adjacency status of remote peer : No fault
  Sequencing: receive disabled, send disabled
  Bindings
    Parameter      Local      Remote
```

```

-----
Label          6002                      4016
Group ID       n/a                      n/a
Interface
MTU            1500                      1500
Control word on (configured: autosense) on
PW type        Ethernet                  Ethernet
VCCV CV type 0x02                      0x02
                LSPV [2]                  LSPV [2]
VCCV CC type 0x06                      0x06
                RA [2], TTL [3]            RA [2], TTL [3]
Status TLV     enabled                  supported
SSO Descriptor: 55.0.0.4/3467, local label: 6002
Dataplane:
  SSM segment/switch IDs: 41031/4160 (used), PWID: 5
Rx Counters
  35 input transit packets, 3500 bytes
  82 drops, 0 seq err
Tx Counters
  26 output transit packets, 3456 bytes
  0 drops

```

There is nothing particularly special about the LDP session. It is just like a normal PW, except it was dynamically discovered using some new BGP extended communities. The discovery mechanism is targeted because once the L2VPN VPLS routes are imported on a PE, the BGP next-hop becomes an endpoint for tLDP.

```

R6#show mpls ldp neighbor 55.0.0.4
  Peer LDP Ident: 55.0.0.4:0; Local LDP Ident 55.0.0.6:0
    TCP connection: 55.0.0.4.646 - 55.0.0.6.16520
    State: Oper; Msgs sent/rcvd: 24/23; Downstream
    Up time: 00:08:10
    LDP discovery sources:
      Targeted Hello 55.0.0.6 -> 55.0.0.4, active, passive
    Addresses bound to peer LDP Ident:
      55.0.0.4      10.35.1.204      55.4.12.4

```

As an aside, the L2VPN RID is derived from the “global MPLS RID” which doesn’t make sense, as there isn’t one. This is according to the IOS XE 3S command reference. I am assuming that it derives its address the same way more RID-aware processes do by checking for the highest loopback address. The BGP prefix itself contains the L2VPN RID, but the BGP next-hop remains the update-source specified in the BGP configuration. We change the L2VPN RID on CSR4 then check the changes on CSR6. The BGP next-hop is unchanged by the prefix itself now reflects the new L2VPN RID. This has no effect on connectivity because, as stated above, the PW is built towards the BGP next-hop, not the L2VPN RID.

```

! CSR4
l2vpn

```

```
router-id 55.4.4.4
```

```
R6#show bgp l2vpn vpls rd 55:3467 | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 55:3467
*> 55:3467:55.0.0.6/96
      0.0.0.0
      32768 ?
*>i 55:3467:55.0.0.7/96
      55.0.0.7
      0      100      0 ?
*>i 55:3467:55.0.0.13/96
      55.0.0.13
      100      0 i
*>i 55:3467:55.4.4.4/96
      55.0.0.4
      0      100      0 ?
```

Since there are no PW interfaces or manual xconnects to configure, we appear to have little control over the dynamic PWs. We can apply a template to the PWs to give them certain characteristics. On CSR4, I enable load-balancing without FAT-PW just to demonstrate the template feature.

```
! CSR4
template type pseudowire TMP_LDP_DYNAMIC
  encapsulation mpls
  load-balance flow

l2vpn vfi context LAN
  autodiscovery bgp signaling ldp template TMP_LDP_DYNAMIC
```

Since we know there are three other PWs, we can quickly check each destination to ensure the load balancing feature is enabled. This is basic inheritance from a template and introduce no new technologies.

```
R4#show l2vpn atom vc vcid 3467 detail | include Destination add|Load
  Destination address: 55.0.0.6 VC ID: 3467
  Load Balance: ECMP
  Destination address: 55.0.0.7 VC ID: 3467
  Load Balance: ECMP
  Destination address: 55.0.0.13 VC ID: 3467
  Load Balance: ECMP
```

The VPLS-ID is carried in the BGP update as well, but not as a new community. Configuring a VPLS-ID overwrites the L2VPN AGI parameter, and when this is mismatched, VPNs cannot form. It would have been more convenient if consistent naming conventions were used, but in short, the VPLS-ID and AGI are interchangeable terms. Notice the VPN ID of 3467 (yielding AGI 55:3467) and VPLS-ID 55:12321 (AGI 55:12321). The BGP route will take the VPLS-ID over the VPN ID. In this case, the VPLS instance is broken as CSR4 claims the other routers do not have a matching extended community despite the route-target being the same. The AGI is also an extended community which explains the debug error.

```

! CSR4
l2vpn vfi context LAN
vpn id 3467
autodiscovery bgp signaling ldp template TMP_LDP_DYNAMIC
vpls-id 55:12321

R4#show bgp l2vpn vpls all | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 55:3467
*> 55:3467:55.4.4.4/96
                                0.0.0.0                                32768 ?

! CSR4
BGP(9): 55.0.0.11 rcvd UPDATE w/ attr: nexthop 55.0.0.13, origin i, localpref
100, originator 55.0.0.13, clusterlist 55.0.0.11, extended community
RT:55:3467 L2VPN AGI:55:3467
BGP(9): 55.0.0.11 rcvd 55:3467:55.0.0.13/96 -- DENIED due to: extended
community not supported;

```

The VPLS-ID is typically used for Inter-AS MPLS Option B and will be discussed more in that section. For now, I configure it to be the same as the VPN ID, although leaving it out completely is ideal unless it's necessary.

```

! CSR4
l2vpn vfi context LAN
vpn id 3467
autodiscovery bgp signaling ldp template TMP_LDP_DYNAMIC
vpls-id 55:3467

R4#show bgp l2vpn vpls all | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 55:3467
*>i 55:3467:55.0.0.6/96
                                55.0.0.6                                0    100    0 ?
*>i 55:3467:55.0.0.7/96
                                55.0.0.7                                0    100    0 ?
*>i 55:3467:55.0.0.13/96
                                55.0.0.13                                100    0 i
*> 55:3467:55.4.4.4/96
                                0.0.0.0                                32768 ?

```

Routed VPLS (using BDI interface) is supported for all VPLS types, including static, LDP-signaled, and BGP-signaled. We can quickly verify OSPF neighbors from a few routers to verify this. Always ensure the BDI interface number matches the bridge domain, just like an SVI number must match the VLAN ID. The configuration is shown for CSR4 only as CSR7's BDI is almost identical. We can see that XRv4 is the DR and has full adjacencies to all nodes, while CSR2 is the BDR. CSR1 sees hellos from the BDIs on CSR4 and

CSR7. CSR7, in summary form, shows two full adjacencies (DR and BDR), two 2WAY adjacencies (CSR1 and CSR4), for a total of 4.

```
! CSR4
interface BDI3541
  vrf forwarding LAN
  ip address 10.0.0.4 255.255.255.0
  ip ospf priority 0
  ip ospf 1 area 0
  no shutdown
```

R1#show ip ospf neighbor

Neighbor ID	Pri	State	Dead Time	Address	Interface
4.4.4.4	0	2WAY/DROTHER	00:00:32	10.0.0.4	Gig2.541
7.7.7.7	0	2WAY/DROTHER	00:00:33	10.0.0.7	Gig2.541
10.0.0.2	1	FULL/BDR	00:00:35	10.0.0.2	Gig2.541
10.0.0.14	1	FULL/DR	00:00:31	10.0.0.14	Gig2.541

RP/0/0/CPU0:XRv4#show ospf vrf LAN neighbor | begin State

Neighbor ID	Pri	State	Dead Time	Address	Interface
4.4.4.4	0	FULL/DROTHER	00:00:31	10.0.0.4	Gig0/0/0/0.564
Neighbor is up for 00:01:33					
7.7.7.7	0	FULL/DROTHER	00:00:32	10.0.0.7	Gig0/0/0/0.564
Neighbor is up for 00:00:54					
10.0.0.1	1	FULL/DROTHER	00:00:37	10.0.0.1	Gig0/0/0/0.564
Neighbor is up for 16:44:48					
10.0.0.2	1	FULL/BDR	00:00:35	10.0.0.2	GigabitEthernet0/0/0/0.564
Neighbor is up for 1d11h					

R7#show ip ospf neighbor summary | include FULL|2WAY|Total

```
2WAY      2
FULL      2
Total count 4      (Undergoing NSF 0)
```

Although relevant for all VPLS signaling types, we will briefly examine QoS for VPLS. The MPLS-core related QoS models are covered in detail in a dedicated section, but QoS on the EFPs is slightly different. It is common for VPLS carriers to police multicast and broadcast traffic aggressively, or charge more money for it, since it cannot be intelligently routed. As discussed earlier, it is highly inefficient to transport multicast over VPLS, so carriers can discourage customers from doing so. On CSR6, we will police multicast/broadcast traffic to 16 kbps using a single rate, two-color policer; exceeding traffic is dropped. The aggregate rate of remaining unicast traffic is limited to 48 kbps where exceeding traffic is remarked to EXP0. The EXP2 marking is applied to all conforming unicast traffic. Obviously this policy is sloppy since this will catch signaling multicast traffic like OSPF and PIM, but this is for demonstration only. As a general QoS consideration for VPLS, the 802.1P bits (CoS) are automatically copied into the MPLS EXP bits on all labels at the ingress PE, assuming the customer has marked them.

```

! CSR6
ip access-list extended ACL_BCAST_MCAST
  permit ip any host 255.255.255.255
  permit ip any host 10.0.0.255
  permit ip any 224.0.0.0 15.255.255.255

class-map match-all CMAP_BCAST_MCAST
  match access-group name ACL_BCAST_MCAST

policy-map PMAP_VPLS_IN
  class CMAP_BCAST_MCAST
    police cir 16000
  class class-default
    police cir 48000
    conform-action set-mpls-exp-imposition-transmit 2
    exceed-action set-mpls-exp-imposition-transmit 0

interface GigabitEthernet2
  service instance 3564 ethernet
  service-policy input PMAP_VPLS_IN

```

To view the policy, we cannot use “show policy-map interface” anymore. We have to use EFP-aware show commands. In an idle state, we see some broadcast/multicast traffic which probably represents the OSPF hellos. We don’t see any unicast yet since we haven’t sent any user traffic.

```

R6#show ethernet service instance policy-map
GigabitEthernet2: EFP 3564

```

```

Service-policy input: PMAP_VPLS_IN

```

```

Class-map: CMAP_BCAST_MCAST (match-all)
  14 packets, 1540 bytes
  5 minute offered rate 0000 bps, drop rate 0000 bps
  Match: access-group name ACL_BCAST_MCAST
  police:
    cir 16000 bps, bc 1500 bytes
    conformed 14 packets, 1540 bytes; actions:
      transmit
    exceeded 0 packets, 0 bytes; actions:
      drop
    conformed 0000 bps, exceeded 0000 bps

Class-map: class-default (match-any)
  0 packets, 0 bytes
  5 minute offered rate 0000 bps, drop rate 0000 bps
  Match: any
  police:

```



```
    cir 48000 bps, bc 1500 bytes
conformed 0 packets, 0 bytes; actions:
    set-mps-exp-imposition-transmit 2
exceeded 0 packets, 0 bytes; actions:
    set-mps-exp-imposition-transmit 0
conformed 0000 bps, exceeded 0000 bps
```

Clearing OSPF neighbors on XRv4 will generate a burst of traffic that is both unicast and multicast.

```
RP/0/0/CPU0:XRv4#clear ospf 1 vrf LAN process
Reset VRF LAN in OSPF process 1? [no]: y
```

```
R6#show ethernet service instance policy-map | include Class-map|packets
Class-map: CMAP_BCAST_MCAST (match-all)
    53 packets, 5898 bytes
Match: access-group name ACL_BCAST_MCAST
    conformed 53 packets, 5898 bytes; actions:
    exceeded 0 packets, 0 bytes; actions:
Class-map: class-default (match-any)
    13 packets, 1406 bytes
    conformed 13 packets, 1406 bytes; actions:
    exceeded 0 packets, 0 bytes; actions:
```

With OSPF converged, there is no more unicast on the network. Sending N packets from XRv4 to any VPLS endpoint should increase it by exactly N. We send 100 packets to CSR1 and CSR2 and expect to see 200 more unicast packets matched (100 to each destination).

```
RP/0/0/CPU0:XRv4#ping vrf LAN 10.0.0.1 repeat 100
Type escape sequence to abort.
Sending 100, 100-byte ICMP Echos to 10.0.0.1, timeout is 2 seconds:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Success rate is 100 percent (100/100), round-trip min/avg/max = 1/19/29 ms
```

```
RP/0/0/CPU0:XRv4#ping vrf LAN 10.0.0.2 repeat 100
Type escape sequence to abort.
Sending 100, 100-byte ICMP Echos to 10.0.0.2, timeout is 2 seconds:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Success rate is 100 percent (100/100), round-trip min/avg/max = 1/19/49 ms
```

```
R6#show ethernet service instance policy-map | include Class-map|packets
Class-map: CMAP_BCAST_MCAST (match-all)
    73 packets, 8098 bytes
    conformed 73 packets, 8098 bytes; actions:
    exceeded 0 packets, 0 bytes; actions:
Class-map: class-default (match-any)
    213 packets, 25006 bytes
```

```
conformed 213 packets, 25006 bytes; actions:
exceeded 0 packets, 0 bytes; actions:
```

We will create a flood of multicast pings from XRv4. We need to enable multicast routing first or else XR cannot originate this traffic.

```
RP/0/0/CPU0:XRv4#ping vrf LAN 225.0.0.1 size 1450 repeat 50 timeout 0
Mdef cons get failed for VRF 0x60000002 - No such process
```

```
! XRv4
multicast-routing
vrf LAN
address-family ipv4
interface all enable
```

We can now originate the traffic. XR warns us that such a low timeout is a bad idea, but we don't expect it to work anyway. We can see 48/50 of those packets were marked as exceeding and dropped.

```
RP/0/0/CPU0:XRv4#ping vrf LAN 225.0.0.1 size 1450 repeat 50 timeout 0
pings with timeout=0 may result in system instability and
control protocol flaps resulting in traffic impact.
Do you really want to continue[confirm with only 'y' or 'n'] [y/n] :y
Type escape sequence to abort.
Sending 50, 1450-byte ICMP Echos to 225.0.0.1, timeout is 0 seconds:
.....
```

```
R6#show ethernet service instance policy-map | include Class-map|packets
Class-map: CMAP_BCAST_MCAST (match-all)
  164 packets, 85286 bytes
    conformed 116 packets, 14822 bytes; actions:
    exceeded 48 packets, 70464 bytes; actions:
Class-map: class-default (match-any)
  218 packets, 25596 bytes
    conformed 218 packets, 25596 bytes; actions:
    exceeded 0 packets, 0 bytes; actions:
```

We cannot test broadcast from this node as XR does not support it. Assuming the MQC classification ACL was correct, broadcast traffic would also be policed similarly to multicast traffic.

```
RP/0/0/CPU0:XRv4#ping vrf LAN 255.255.255.255 size 1450 repeat 50 timeout 0
% Ping to Broadcast ipaddress is not supported
```

We will quickly test our unicast policer to ensure it marks traffic accordingly. XRv4 will generate traffic while CSR6 will mark conforming traffic to EXP2 and exceeding traffic to EXP0. The incoming label is 0xFB8 which is 4024 in decimal. We quickly verify this is the correct PW label. In green, I highlight the EXP/S-bit nibble. The left-most 3 bits represent EXP, so a value of 000 is EXP0 and 010 is EXP2. The last bit is always 1 since we are not using FAT-PW and this is the PW label. Thus, EXP0 is represented as 0x1

(0001) and EXP2 is represented as 0x5 (0101). The last byte is TTL and is not highlighted here. We can see the combination of EXP2 and EXP0 given the policer BC/BE values, which we did not explicitly configure. Seeing both values as part of the same flow (and only ms apart) is a good indication that the policer remarking technique is functioning correct.

```
R4#show l2vpn atom binding 55.0.0.6 | include Label
Local Label: 4024
Remote Label: 6016
```

```
R4#show monitor capture CAP buffer detailed
 1 1490      0.053006  00:50:56:A9:0E:6F -> 00:50:56:A9:2C:57 MPLS unicast
0000: 005056A9 2C570050 56A90E6F 8100DD6 .PV.,W.PV..o....
0010: 884700FB 85FD0000 00000050 56A91AAA .G.....PV...
0020: 005056A9 862A0800 450005AA 00000000 .PV...*..E.....
0030: FF01A244 0A00000E 0A000001 0800AF2A ...D.....*

 2 1490      0.120996  00:50:56:A9:0E:6F -> 00:50:56:A9:2C:57 MPLS unicast
0000: 005056A9 2C570050 56A90E6F 8100DD6 .PV.,W.PV..o....
0010: 884700FB 81FD0000 00000050 56A91AAA .G.....PV...
0020: 005056A9 862A0800 450005AA 00010000 .PV...*..E.....
0030: FF01A243 0A00000E 0A000001 0800AF29 ...C.....)

 3 1490      0.357983  00:50:56:A9:0E:6F -> 00:50:56:A9:2C:57 MPLS unicast
0000: 005056A9 2C570050 56A90E6F 8100DD6 .PV.,W.PV..o....
0010: 884700FB 81FD0000 00000050 56A91AAA .G.....PV...
0020: 005056A9 862A0800 450005AA 00020000 .PV...*..E.....
0030: FF01A242 0A00000E 0A000001 0800AF28 ...B.....(

 4 1490      0.602981  00:50:56:A9:0E:6F -> 00:50:56:A9:2C:57 MPLS unicast
0000: 005056A9 2C570050 56A90E6F 8100DD6 .PV.,W.PV..o....
0010: 884700FB 85FD0000 00000050 56A91AAA .G.....PV...
0020: 005056A9 862A0800 450005AA 00030000 .PV...*..E.....
0030: FF01A241 0A00000E 0A000001 0800AF27 ...A.....'
```

Running this test several times shows that about 40% of the unicast traffic was in excess of 48 kbps, and was subsequently remarked to EXP0.

```
R6#show ethernet service instance policy-map | include Class-map|packets
Class-map: CMAP_BCAST_MCAST (match-all)
 166 packets, 16924 bytes
   conformed 167 packets, 17034 bytes; actions:
   exceeded 0 packets, 0 bytes; actions:
Class-map: class-default (match-any)
 256 packets, 367650 bytes
   conformed 156 packets, 220850 bytes; actions:
   exceeded 100 packets, 146800 bytes; actions:
```

The configurations for the LDP signaling test are after the next section since both are part of the BGP auto-discovery setup.

32.1.2.2 BGP signaling

This section is a continuation from the last. The BGP signaling feature of VPLS with auto-discovery uses BGP to carry the PW label-related information. This tends to scale better as there are not $N * (N-1) / 2$ LDP sessions being built to support E-LAN. We will use BGP signaling to build an E-TREE between CSR3, CSR8, CSR5, and CSR9. Like the static configurations, CSR8 and CSR3 are roots while CSR5 and CSR9 are leaves. In E-LAN, we allowed the VPN-ID to be copied into the RT field so that everyone would automatically import/export that RT, creating a full mesh of PWs. For E-TREE, that is undesirable, and we require more control over the VPN endpoints. We can disable auto-route-target and perform this process manually, which is identical to L3VPN. The roots will import all RTs and the leaves only import root RTs. I used a separate RT on each node for flexibility and testing, but in real life, only two unique RTs are needed: one root RT and one leaf RT. The VPLS endpoint (VE) configurations are described shortly. The BD and EFP configurations are not shown as they identical to the static VPLS example.

Note: Unlike LDP signaling, BGP signaling is only supported with the new “l2vpn” syntax, so there are no legacy configurations in this model. This feature also does not appear to support CW or FAT-PW, at all. These CW/FAT-PW signaling are bits carried in an extended community (L2VPN INFO) but are always zero. There is no obvious mechanism in XE to modify it.

```
! CSR8
l2vpn vfi context TREE
vpn id 3589
autodiscovery bgp signaling bgp
ve id 10
ve range 11
route-target import 3589:2
route-target import 3589:14
route-target export 3589:10
route-target import 3589:1
no auto-route-target
```

```
! CSR3
l2vpn vfi context TREE
vpn id 3589
autodiscovery bgp signaling bgp
ve id 1
ve range 11
route-target import 3589:2
route-target import 3589:14
route-target import 3589:10
route-target export 3589:1
no auto-route-target
```

```
! CSR5
l2vpn vfi context TREE
```

```

vpn id 3589
autodiscovery bgp signaling bgp
ve id 5
route-target import 3589:1
route-target import 3589:10
route-target export 3589:14
no auto-route-target

! CSR9
l2vpn vfi context TREE
vpn id 3589
autodiscovery bgp signaling bgp
ve id 9
ve range 11
route-target import 3589:1
route-target import 3589:10
route-target export 3589:2
no auto-route-target

```

Next, we need to configure BGP similarly for BGP signaling as we did for LDP signaling. This is where the XR AF-groups come into play. We can duplicate the RR configuration except suppress LDP versus BGP for signaling. The XR negative logic is that LDP signaling should be disabled, implying that BGP signaling should be enabled. XE must specifically suppress the LDP signaling to use BGP signaling.

```

! XRv1
router bgp 55
af-group AF_L2_BGP address-family l2vpn vpls-vpws
route-reflector-client
Signalling ldp disable

neighbor-group NBR_L2_BGP
use session-group SES_IBGP
address-family l2vpn vpls-vpws
use af-group AF_L2_BGP

neighbor 55.0.0.3
use neighbor-group NBR_L2_BGP

neighbor 55.0.0.5
use neighbor-group NBR_L2_BGP

neighbor 55.0.0.8
use neighbor-group NBR_L2_BGP

neighbor 55.0.0.9
use neighbor-group NBR_L2_BGP

! All XE routers

```

```

router bgp 55
  bgp log-neighbor-changes
  no bgp default ipv4-unicast
  neighbor 55.0.0.11 remote-as 55
  neighbor 55.0.0.11 transport connection-mode active
  neighbor 55.0.0.11 password BGP_AUTH
  neighbor 55.0.0.11 update-source Loopback0

address-family l2vpn vpls
  neighbor 55.0.0.11 activate
  neighbor 55.0.0.11 suppress-signaling-protocol ldp

```

We see no BGP “illegal network” errors as we did for LDP, since the prefix-length of 1 is expected for this mechanism of signaling. We can verify connectivity from the RR and ensure the SAFI was negotiated properly.

```

RP/0/0/CPU0:XRv1#show bgp l2vpn vpls neighbors 55.0.0.3 | include L2VPN
  Address family L2VPN VPLS: advertised and received
  For Address Family: L2VPN VPLS

```

```

RP/0/0/CPU0:XRv1#show bgp l2vpn vpls neighbors 55.0.0.5 | include L2VPN
  Address family L2VPN VPLS: advertised and received
  For Address Family: L2VPN VPLS

```

```

RP/0/0/CPU0:XRv1#show bgp l2vpn vpls neighbors 55.0.0.8 | include L2VPN
  Address family L2VPN VPLS: advertised and received
  For Address Family: L2VPN VPLS

```

```

RP/0/0/CPU0:XRv1#show bgp l2vpn vpls neighbors 55.0.0.9 | include L2VPN
  Address family L2VPN VPLS: advertised and received
  For Address Family: L2VPN VPLS

```

A quick look at the BGP table on CSR3, a root node, shows four routes. Upon cursory inspection, they look different than the LDP ones. The VE ID is the VPLS endpoint ID and must be configured on each PE. It is like a RID and represents each PE in the VPLS instance. This VE ID is also how you reference these prefixes to look inside them, so there isn’t a dotted-decimal network displayed. The “Blk” represents the block-offset which is discussed shortly.

```

R3#show bgp l2vpn vpls rd 55:3589 | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 55:3589
*>  55:3589:VEID-1:Blk-1/136
      0.0.0.0                      32768 ?
*>i  55:3589:VEID-5:Blk-1/136
      55.0.0.5                      0    100      0 ?
*>i  55:3589:VEID-9:Blk-1/136
      55.0.0.9                      0    100      0 ?

```

```
*>i 55:3589:VEID-10:Blk-1/136
55.0.0.8 0 100 0 ?
```

The route contains some new fields. Notice that exact label values were not exchanged in the traditional L2VPN manner. This makes sense because there are no peer-to-peer sessions between PEs. Since each PW requires a different set of labels, the router must create a mechanism for each PE to automatically and deterministically compute the local/remote labels. Label 9000 shown below is just the start of the peer's label base or label block (LB). The virtual block size (VBS) is 11 from this peer, meaning it can allocate up to 11 local labels to support this VPLS instance. Essentially, this sets a maximum of 11 dynamic PWs. CSR5 is the only router that did not specify a VBS and the default value is the minimum of 10; this is done to conserve local labels for small VPLS deployments.

```
R3#show bgp l2vpn vpls rd 55:3589 ve-id 9 block-offset 1
BGP routing table entry for 55:3589:VEID-9:Blk-1/136, version 11
Paths: (1 available, best #1, table L2VPN-VPLS-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    55.0.0.9 (metric 20) from 55.0.0.11 (55.0.0.11)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      AGI version(0), VE Block Size(11) Label Base(9000)
      Extended Community: RT:3589:2 L2VPN L2:0x0:MTU-1500
      Originator: 55.0.0.9, Cluster list: 55.0.0.11
      mpls labels in/out exp-null/9000
      rx pathid: 0, tx pathid: 0x0
```

BGP signaling introduces quite a bit of math. So far, we have defined the VE-ID, VBS, and LB. The virtual block offset (VBO) is computed by the router according to this formula: $VBO = \text{ROUND}(\text{VE-ID}/\text{VBS}) * \text{VBS}$, where the ROUND function discards the remainder and always rounds down. For this route, $VBO = \text{ROUND}(9/11) * 11 = 0$. When the VBO evaluates to 0, the router automatically uses a value of 1, so in this case, the VBO is actually 1. The VBO is used when a PE needs to allocate multiple different LBs in the form of multiple BGP routes. For now, we will leave the VBO as 1 for each node by ensuring the $\text{VE-ID} < \text{VBS}$ everywhere. The table below gives a quick summary of the routes and the previously defined fields. From the perspective of CSR9, only two PWs are created towards the roots, since it is a leaf in the E-TREE.

```
R9#show l2vpn signaling rib rd 55:3589
+- Origin of entry (i=iBGP/e=eBGP)
| +- Provisioned (Yes/No)?
| | +- Stale entry (Yes/No)?
| | |
v v v
O P S RD VE-ID VBO VBS LB Next-Hop
-+-+-----+-----+-----+-----+-----+
i Y N 55:3589 1 1 11 3000 55.0.0.3
```

```
i Y N 55:3589 10 1 11 8000 55.0.0.8
```

The detailed view provides additional information to include the MTU settings, route-targets, control-word applicability.

```
R9#show l2vpn signaling rib rd 55:3589 detail
Route 55:3589:1 (epoch:0) from iBGP peer 55.0.0.3
Provisioned (Y) Stale (N)
  Route-Target: 3589:1
NLRI [4000000A]
  VE-ID:1 VBO:1 VBS:11 LB:3000
  MTU: 1500 Control Word: off
RIB Filter [D4000008]
  Local RD: 55:3589
  VE-ID: 9, VBO: 1, VBS: 11 LB: 9030
Forwarder [8B000002] VFI TREE
```

```
Route 55:3589:10 (epoch:0) from iBGP peer 55.0.0.8
Provisioned (Y) Stale (N)
  Route-Target: 3589:10
NLRI [6300000B]
  VE-ID:10 VBO:1 VBS:11 LB:8000
  MTU: 1500 Control Word: off
RIB Filter [D4000008]
  Local RD: 55:3589
  VE-ID: 9, VBO: 1, VBS: 11 LB: 9030
Forwarder [8B000002] VFI TREE
```

So far, we have not seen BGP distribute any discrete labels for any PW on any node, yet somehow the VPLS has formed. This is because upon receiving these BGP VPLS routes, the receiving PE must select a label independently (and deterministically) from everyone else. There is a sanity check that is applied first to ensure the received VBO is within range of the local configuration. Using CSR9, the local VE-ID (on CSR9) is compared against the received VBO and VBO/VBS sum. CSR9 needs to ensure that its VE-ID is in between the received VBO (lower bound) and the received VBO + VBS (upper bound) using the formula $[\text{remote VBO} \leq \text{local VE-ID} < \text{remote VBO} + \text{VBS}]$. In this case, the formula evaluates to true: $1 \leq 9 < 1 + 11$. When there are multiple LBs, this is used as a prefix selection mechanism, discussed later.

```
R9#show bgp l2vpn vpls rd 55:3589 ve-id 1 block-offset 1
BGP routing table entry for 55:3589:VEID-1:Blk-1/136, version 3
Paths: (1 available, best #1, table L2VPN-VPLS-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    55.0.0.3 (metric 20) from 55.0.0.11 (55.0.0.11)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    AGI version(0), VE Block Size(11) Label Base(3000)
    Extended Community: RT:3589:1 L2VPN L2:0x0:MTU-1500
```



```

Originator: 55.0.0.3, Cluster list: 55.0.0.11
mpls labels in/out exp-null/3000
rx pathid: 0, tx pathid: 0x0

```

Next, CSR9 needs to select a label from this block. We already know it will be within the range of 3000 – 3010 inclusive, which includes 11 possible choices. The demultiplexer formula is used: $LB + VE-ID - VBO$, where the VE-ID is local to the receiver and the LB / VBO are contained in the received route. In this case, the formula evaluates: $3000 + 9 - 1 = 3008$.

```
R9#show l2vpn vfi name TREE
```

Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No

```
VFI name: TREE, state: up, type: multipoint, signaling: BGP
```

```
VPN ID: 3589, VE-ID: 9, VE-SIZE: 11
```

```
RD: 55:3589, RT: import 3589:1, import 3589:10, export 3589:2
```

```
Bridge-Domain 3589 attachment circuits:
```

```
Pseudo-port interface: pseudowire100008
```

Interface	Peer Address	VE-ID	Local Label	Remote Label	S
pseudowire100010	55.0.0.8	10	9039	8008	Y
pseudowire100009	55.0.0.3	1	9030	3008	Y

We also notice the local label of 9030 is used on the same PW. We can quickly do the math from CSR3's perspective to ensure that is correct. We can get all the information we need from the first command, but I show the BGP prefix for practice. The sanity check succeeds: $VBO \leq VE-ID < VBS + VBO$, which evaluates as $1 \leq 1 < 1 + 11$. Next, CSR3 selects a label: $LB + VE-ID - VBO$, which evaluates as $9030 + 1 - 1 = 9030$. When the local VE-ID equals the VBO advertised in a route, the first label in the LB is used.

```
R3#show l2vpn signaling rib rd 55:3589 detail | begin :9
```

```
Route 55:3589:9 (epoch:0) from iBGP peer 55.0.0.9
```

```
Provisioned (Y) Stale (N)
```

```
Route-Target: 3589:2
```

```
NLRI [7E00000B]
```

```
VE-ID:9 VBO:1 VBS:11 LB:9030
```

```
MTU: 1500 Control Word: off
```

```
RIB Filter [4F000004]
```

```
Local RD: 55:3589
```

```
VE-ID: 1, VBO: 1, VBS: 11 LB: 3000
```

```
Forwarder [5F000002] VFI TREE
```

```
[snip]
```

```
R3#show bgp l2vpn vpls rd 55:3589 ve-id 9 block-offset 1
```

```
BGP routing table entry for 55:3589:VEID-9:Blk-1/136, version 52
```

```
Paths: (1 available, best #1, table L2VPN-VPLS-BGP-Table)
```

```
Not advertised to any peer
```

```
Refresh Epoch 1
```

```
Local
```

```
55.0.0.9 (metric 20) from 55.0.0.11 (55.0.0.11)
```

```

Origin incomplete, metric 0, localpref 100, valid, internal, best
AGI version(0), VE Block Size(11) Label Base(9030)
Extended Community: RT:3589:2 L2VPN L2:0x0:MTU-1500
Originator: 55.0.0.9, Cluster list: 55.0.0.11
mpls labels in/out exp-null/9030
rx pathid: 0, tx pathid: 0x0

```

Rather than distribute labels within the VPLS instance per PW, BGP allocates a range and allows the receiver to choose that label by following a strict formula. This guarantees that both sides select the same labels so that PWs can form properly. The formula is also deterministic, unlike LDP which will allocate new labels every time the session is cleared. This is more complex to manually verify than LDP but scales better. We will do one more example for practice by tracing the PW signaling between CSR8 and CSR5. CSR5 performs a sanity check on the received route: $VBO \leq VE-ID < VBS + VBO$ evaluates as $1 \leq 5 < 10 + 1$. Then, CSR5 selects a label: $LB + VE-ID - VBO$ evaluates as $8000 + 5 - 1 = 8004$. Remember that when doing these sanity checks and label selections, the VE-ID is the local value, while the VBO, VBS, and LB are the received (remote) values.

```

R5#show l2vpn signaling rib rd 55:3589 detail | begin \.8$
Route 55:3589:10 (epoch:0) from iBGP peer 55.0.0.8
Provisioned (Y) Stale (N)
  Route-Target: 3589:10
NLRI [5D000002]
  VE-ID:10 VBO:1 VBS:11 LB:8000
  MTU: 1500 Control Word: off
RIB Filter [9B000002]
  Local RD: 55:3589
  VE-ID: 5, VBO: 1, VBS: 10 LB: 5000
Forwarder [FF000001] VFI TREE

```

Before looking at the actual labels, we will do the math for CSR8 also. We can technically do this on CSR5 since CSR5 is doing the math anyway to determine the local label. CSR5 cannot allocate a local label for the PW until it learns the BGP route as it must know the remote VE-ID. We can use the same output as above and highlighted different focus areas. The logic is reversed: for local label selection, the VE-ID is remote and the VBO, VBS, and LB are local. First, the sanity check is performed: $VBO \leq VE-ID < VBS + VBO$ evaluates as $1 \leq 10 < 10 + 1$. Then, the local label is selected: $LB + VE-ID - VBO$ evaluates as $5000 + 10 - 1 = 5009$.

```

R5#show l2vpn signaling rib rd 55:3589 detail | begin \.8$
Route 55:3589:10 (epoch:0) from iBGP peer 55.0.0.8
Provisioned (Y) Stale (N)
  Route-Target: 3589:10
NLRI [5D000002]
  VE-ID:10 VBO:1 VBS:11 LB:8000
  MTU: 1500 Control Word: off
RIB Filter [9B000002]
  Local RD: 55:3589

```

```
VE-ID: 5, VBO: 1, VBS: 10 LB: 5000
Forwarder [FF000001] VFI TREE
```

We computed the remote label of 8004 and local label of 5009 using a single command to view the L2VPN signaling. We can verify it by checking the VFI status.

```
R5#show l2vpn vfi name TREE
Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No
VFI name: TREE, state: up, type: multipoint, signaling: BGP
VPN ID: 3589, VE-ID: 5, VE-SIZE: 10
RD: 55:3589, RT: import 3589:1, import 3589:10, export 3589:14
Bridge-Domain 3589 attachment circuits:
Pseudo-port interface: pseudowire100001
Interface      Peer Address    VE-ID  Local Label  Remote Label  S
pseudowire100003 55.0.0.8       10     5009         8004          Y
pseudowire100002 55.0.0.3       1      5000         3004          Y
```

So far, all VBOs have been 1 since the VE-ID has always been less than the VBS. The VBO calculation has always resulted in zero, which gets automatically rounded to 1 as discussed earlier. On CSR9, we will set the VE-ID to 90 and adjust the VBS to 16. The new block offset should be: $VBO = \text{ROUND}(\text{VE-ID}/\text{VBS}) * \text{VBS}$ evaluates to $\text{ROUND}(90/16) * 16 = 5 * 16 = 80$. A quick look at CSR9 confirms this. Notice that there is also a VBO of 1 in addition to the new VBO of 80. This is how one advertises multiple LBs; the VBO serves as a differentiator for each BGP route.

```
! CSR9
l2vpn vfi context TREE
autodiscovery bgp signaling bgp
ve id 90
ve range 16
```

```
R9#show bgp l2vpn vpls rd 55:3589 | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 55:3589
*>i 55:3589:VEID-1:Blk-1/136
      55.0.0.3      0      100      0 ?
*>i 55:3589:VEID-1:Blk-88/136
      55.0.0.3      0      100      0 ?
*>i 55:3589:VEID-10:Blk-1/136
      55.0.0.8      0      100      0 ?
*>i 55:3589:VEID-10:Blk-88/136
      55.0.0.8      0      100      0 ?
*> 55:3589:VEID-90:Blk-1/136
      0.0.0.0      32768 ?
*> 55:3589:VEID-90:Blk-80/136
      0.0.0.0      32768 ?
```

Looking at the details of each locally originated route, the only significant difference is the LB. One of them starts at 9020, and one starts at 9036. Since the VBS is 16, 9020 – 9035 was the first available block of 16 contiguous, unallocated labels, and 9036 – 9051 was the next. The two blocks don't have to be contiguous themselves, but coincidentally the blocks are contiguous in this example.

```
R9#show bgp l2vpn vpls rd 55:3589 ve-id 90 block-offset 1
BGP routing table entry for 55:3589:VEID-90:Blk-1/136, version 16
Paths: (1 available, best #1, table L2VPN-VPLS-BGP-Table)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (55.0.0.9)
      Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
      AGI version(0), VE Block Size(16) Label Base(9036)
      Extended Community: RT:3589:2 L2VPN L2:0x0:MTU-1500
        L2VPN L2:0x0:MTU-1500
      mpls labels in/out exp-null/9036
      rx pathid: 0, tx pathid: 0x0
```

```
R9#show bgp l2vpn vpls rd 55:3589 ve-id 90 block-offset 80
BGP routing table entry for 55:3589:VEID-90:Blk-80/136, version 17
Paths: (1 available, best #1, table L2VPN-VPLS-BGP-Table)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local
    0.0.0.0 from 0.0.0.0 (55.0.0.9)
      Origin incomplete, localpref 100, weight 32768, valid, sourced, local,
best
      AGI version(0), VE Block Size(16) Label Base(9020)
      Extended Community: RT:3589:2 L2VPN L2:0x0:MTU-1500
      mpls labels in/out exp-null/9020
      rx pathid: 0, tx pathid: 0x0
```

Let's trace the PW label selection between CSR8 and CSR9. We will do all of the calculations from CSR9's perspective. CSR9 needs to identify the remote label to use when sending traffic towards CSR8 inside this VPLS instance; this uses the remote VBO, VBS, and LB, along with the local VE-ID. The sanity check occurs first using formula: $VBO \leq VE-ID < VBS + VBO$ evaluates as $88 \leq 90 < 88 + 11$. The remote VBO of 88 is a new number that has no explanation yet; we will revisit this shortly. Then, CSR9 determines the remote label using formula: $LB + VE-ID - VBO$ evaluates as $8022 + 90 - 88 = 8024$.

```
R9#show l2vpn signaling rib rd 55:3589 detail | begin \.8$
Route 55:3589:10 (epoch:0) from iBGP peer 55.0.0.8
Provisioned (Y) Stale (N)
  Route-Target: 3589:10
```

```

NLRI [76000004]
  VE-ID:10 VBO:88 VBS:11 LB:8022
  MTU: 1500 Control Word: off
RIB Filter [C3000004]
  Local RD: 55:3589
  VE-ID: 90, VBO: 1, VBS: 16 LB: 9036
Forwarder [88000001] VFI TREE

```

We will quickly complete the local label calculation then explain how remote VBO 88 was computed. The sanity check: $VBO \leq VE-ID < VBS + VBO$ evaluates as $1 \leq 10 < 16 + 1$. The label selection: $LB + VE-ID - VBO$ evaluates as $9036 + 10 - 1 = 9045$. We confirm these are the correct labels by checking the VFI members.

```

R9#show l2vpn signaling rib rd 55:3589 detail | begin \.8$
Route 55:3589:10 (epoch:0) from iBGP peer 55.0.0.8
Provisioned (Y) Stale (N)
  Route-Target: 3589:10
NLRI [76000004]
  VE-ID:10 VBO:88 VBS:11 LB:8022
  MTU: 1500 Control Word: off
RIB Filter [C3000004]
  Local RD: 55:3589
  VE-ID: 90, VBO: 1, VBS: 16 LB: 9036
Forwarder [88000001] VFI TREE

```

```

R9#show l2vpn vfi name TREE
Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No

```

```

VFI name: TREE, state: up, type: multipoint, signaling: BGP
  VPN ID: 3589, VE-ID: 90, VE-SIZE: 16
  RD: 55:3589, RT: import 3589:1, import 3589:10, export 3589:2
  Bridge-Domain 3589 attachment circuits:
  Pseudo-port interface: pseudowire100001

```

Interface	Peer Address	VE-ID	Local Label	Remote Label	S
pseudowire100003	55.0.0.8	10	9045	8024	Y
pseudowire100002	55.0.0.3	1	9036	3022	Y

To explain the VBO 88, we look back at the BGP output from earlier. I show it again below for reference. Notice that CSR8, without having any new configuration, decided to originate a new route with a new block offset of 88. CSR3 also did this, yet only CSR9 underwent a configuration change.

```

R9#show bgp l2vpn vpls rd 55:3589 | begin Network
  Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 55:3589
  *>i 55:3589:VEID-1:Blk-1/136
                                55.0.0.3          0      100      0 ?
  *>i 55:3589:VEID-1:Blk-88/136

```

```

55.0.0.3 0 100 0 ?
*>i 55:3589:VEID-10:Blk-1/136
55.0.0.8 0 100 0 ?
*>i 55:3589:VEID-10:Blk-88/136
55.0.0.8 0 100 0 ?
*> 55:3589:VEID-90:Blk-1/136
0.0.0.0 32768 ?
*> 55:3589:VEID-90:Blk-80/136
0.0.0.0 32768 ?

```

Before determining why, we can at least figure out where 88 came from. Using the normal VBO formula [VBO = ROUND(VE-ID/VBS) * VBS] on CSR8 and CSR3, we evaluate this as $\text{ROUND}(90/11) * 11 = 8 * 11 = 88$. In this case, the local VBS of 11 was used along with the remote VE-ID of CSR9. If the locally-configured VBS were 10 (default) and not 11, we would have ended up with 80 just as CSR9 did. Because CSR3 has the same VBS value of 11, it computed 88 as well, which shows that the local VE-ID was not used in this calculation. On CSR9, we can look into the route details and see two separate LBs for these. This means that CSR8 has allocated a total of (22) local labels for this VPLS instance. For this reason, the designer must take care when selecting VE-ID and VBS values as they will increase the scale (and LIB size) based on the VBO formula.

```

R9#show bgp l2vpn vpls rd 55:3589 ve-id 10 block-offset 1
BGP routing table entry for 55:3589:VEID-10:Blk-1/136, version 8
Paths: (1 available, best #1, table L2VPN-VPLS-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    55.0.0.8 (metric 30) from 55.0.0.11 (55.0.0.11)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      AGI version(0), VE Block Size(11) Label Base(8000)
      Extended Community: RT:3589:10 L2VPN L2:0x0:MTU-1500
      Originator: 55.0.0.8, Cluster list: 55.0.0.11
      mpls labels in/out exp-null/8000
      rx pathid: 0, tx pathid: 0x0

```

```

R9#show bgp l2vpn vpls rd 55:3589 ve-id 10 block-offset 88
BGP routing table entry for 55:3589:VEID-10:Blk-88/136, version 13
Paths: (1 available, best #1, table L2VPN-VPLS-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    55.0.0.8 (metric 30) from 55.0.0.11 (55.0.0.11)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      AGI version(0), VE Block Size(11) Label Base(8022)
      Extended Community: RT:3589:10 L2VPN L2:0x0:MTU-1500
      Originator: 55.0.0.8, Cluster list: 55.0.0.11
      mpls labels in/out exp-null/8022
      rx pathid: 0, tx pathid: 0x0

```

Let's assume that, for some reason, CSR8 did not originate this second route. Going back to our math, CSR9 would first perform the sanity check before selecting a label. Assuming the VBO=1 route from CSR8 is used and the VBO=88 route doesn't exist, the formula: $VBO \leq VE-ID < VBS + VBO$ evaluates as $1 \leq 90 < 11 + 1$, which is false. CSR9 has a VE-ID that is too large for the range offered by CSR8. Because CSR8 performs this exact same calculation to identify its local label, it is smart enough to know a new route must be created with a VBO of 88 based on the calculation above. This second route allows CSR8 to compute its local label and CSR9 to compute its remote label. Also notice that the 9000-series label came from the VBO=1 pool of CSR9, while the 8000-series label came from the VBO=88 pool of CSR8. These are just label blocks and there is no artificial coupling between them. The summary outputs below show the remote information for each PW; CSR8 selected a label from the LB 9036 with VBO 1 while CSR9 did the same using LB 8022 with VBO=88.

```
R8#show l2vpn signaling rib rd 55:3589
```

```

+- Origin of entry                (i=iBGP/e=eBGP)
| +- Provisioned                  (Yes/No) ?
| | +- Stale entry                (Yes/No) ?
| | |
v v v
O P S      RD          VE-ID    VBO    VBS      LB      Next-Hop
+---+---+-----+-----+-----+-----+-----+
i Y N  55:3589         1         1     11       3000     55.0.0.3
i Y N  55:3589         5         1     10       5000     55.0.0.5
i Y N  55:3589        90         1     16       9036     55.0.0.9

```

```
R9#show l2vpn signaling rib rd 55:3589
```

```

+- Origin of entry                (i=iBGP/e=eBGP)
| +- Provisioned                  (Yes/No) ?
| | +- Stale entry                (Yes/No) ?
| | |
v v v
O P S      RD          VE-ID    VBO    VBS      LB      Next-Hop
+---+---+-----+-----+-----+-----+-----+
i Y N  55:3589         1         88     11       3020     55.0.0.3
i Y N  55:3589        10         88     11       8022     55.0.0.8

```

We saw earlier that CSR3 did the same thing, and because its VBS was 11 just like CSR8, it also computed a VBO of 88. For additional testing and variety, we will update CSR3's VE parameters as well. The VE-ID is set to 300 and the VBS is set to 35.

```

! CSR3
l2vpn vfi context TREE
  autodiscovery bgp signaling bgp
    ve id 300
    ve range 45

```

The new VBO calculation is as follows: $VBO = \text{ROUND}(\text{VE-ID}/\text{VBS}) * \text{VBS}$ evaluates as $\text{ROUND}(300/45) * 45 = 270$. We can now see that many routers are originating three routes with three different VBOs. This is mostly due to the wide variance in VE-ID and VBS configurations across the VPLS PEs. We see that CSR5 originated a route with a new VBO also: $VBO = \text{ROUND}(\text{VE-ID}/\text{VBS}) * \text{VBS}$ evaluates as $\text{ROUND}(300/10) * 10 = 300$. This is done using the local VBS (default) value of 10 along with the remote VE-ID. CSR8 does the same thing: $VBO = \text{ROUND}(\text{VE-ID}/\text{VBS}) * \text{VBS}$ evaluates as $\text{ROUND}(300/11) * 11 = 297$. By dividing by a number, rounding down, then multiplying by that number, we introduce a tiny amount of variance into the result which gives slightly different block sizes, assuming the VBS is different between the nodes. We also see the routes with the block sizes that are about the same as before, but some have been adjusted based on CSR3's VE changes. These new ranges are shown in yellow below. We see that CSR8 and CSR9 still have VBOs of 88 and 80 respectively; this is a result of their local VBS values not changing. CSR3 used to have a block size of 88 when its VBS was 11. With a new VBS of 45, the math needs to be evaluated: $VBO = \text{ROUND}(\text{VE-ID}/\text{VBS}) * \text{VBS}$ evaluates as $\text{ROUND}(90/45) * 45 = 90$. These ranges are shown in green. Fortunately, all of this happens automatically so remembering every detail is less important than understanding the concepts.

```
R3#show bgp l2vpn vpls rd 55:3589 | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 55:3589
*>i 55:3589:VEID-5:Blk-1/136
      55.0.0.5      0      100      0 ?
*>i 55:3589:VEID-5:Blk-300/136
      55.0.0.5      0      100      0 ?
*>i 55:3589:VEID-10:Blk-1/136
      55.0.0.8      0      100      0 ?
*>i 55:3589:VEID-10:Blk-88/136
      55.0.0.8      0      100      0 ?
*>i 55:3589:VEID-10:Blk-297/136
      55.0.0.8      0      100      0 ?
*>i 55:3589:VEID-90:Blk-1/136
      55.0.0.9      0      100      0 ?
*>i 55:3589:VEID-90:Blk-80/136
      55.0.0.9      0      100      0 ?
*>i 55:3589:VEID-90:Blk-288/136
      55.0.0.9      0      100      0 ?
*> 55:3589:VEID-300:Blk-1/136
      0.0.0.0      32768 ?
*> 55:3589:VEID-300:Blk-90/136
      0.0.0.0      32768 ?
*> 55:3589:VEID-300:Blk-270/136
      0.0.0.0      32768 ?
```

The routers are smart enough to originate these routes when they determine that they are unable to select labels within the given range based on remote VE-IDs. This is the purpose of the “sanity check”. We will trace two PWs to verify this, starting with the PW from CSR3 to CSR5. We can derive both labels using show commands on CSR3. The sanity check takes place multiple times against all of the VBOs

offered by CSR5. We will evaluate them all. For the first test, we need to reference the output above since they will fail the sanity check. First, the formula $VBO \leq VE-ID < VBS + VBO$ evaluates as $1 \leq 300 < 10 + 1$, which is false. This is why the failed condition does not appear in the output below as it is not valid for PW signaling. The next test is successful: $VBO \leq VE-ID < VBS + VBO$ evaluates as $300 \leq 300 < 10 + 300$, which is true. Now, CSR3 can select a remote label using $LB + VE-ID - VBO$ which evaluates as $5020 + 300 - 300 = 5020$.

```
R3#show l2vpn signaling rib rd 55:3589 detail | begin \.5$
Route 55:3589:5 (epoch:0) from iBGP peer 55.0.0.5
Provisioned (Y) Stale (N)
  Route-Target: 3589:14
NLRI [28000005]
  VE-ID:5 VBO:300 VBS:10 LB:5020
  MTU: 1500 Control Word: off
RIB Filter [AC000005]
  Local RD: 55:3589
  VE-ID: 300, VBO: 1, VBS: 45 LB: 3076
Forwarder [F7000001] VFI TREE
[snip]
```

CSR3 also has to compute a local label for this PW using the opposite set of information highlighted below. CSR3 has three possible VBOs from CSR3 to use for this: 1, 90, and 270. We can see that VBO 1 was selected since the remote VE-ID is a small number, but we will evaluate the failed conditions for practice. This will clarify why the VBOs of 90 and 270 are not used for label selection. First, the formula $VBO \leq VE-ID < VBS + VBO$ evaluates as $90 \leq 5 < 45 + 90$, which is false. Second, the formula $VBO \leq VE-ID < VBS + VBO$ evaluates as $270 \leq 5 < 45 + 270$, which is also false. Finally, the formula $VBO \leq VE-ID < VBS + VBO$ evaluates as $1 \leq 5 < 45 + 1$, which is true. Now CSR3 can derive its local label for this PW: $LB + VE-ID - VBO$ evaluates as $3076 + 5 - 1 = 3080$. We confirm the label pair of local = 3080 and remote = 5020.

```
R3#show l2vpn signaling rib rd 55:3589 detail | begin \.5$
Route 55:3589:5 (epoch:0) from iBGP peer 55.0.0.5
Provisioned (Y) Stale (N)
  Route-Target: 3589:14
NLRI [28000005]
  VE-ID:5 VBO:300 VBS:10 LB:5020
  MTU: 1500 Control Word: off
RIB Filter [AC000005]
  Local RD: 55:3589
  VE-ID: 300, VBO: 1, VBS: 45 LB: 3076
Forwarder [F7000001] VFI TREE
[snip]
```

```
R3#show l2vpn vfi name TREE
Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No
```

```
VFI name: TREE, state: up, type: multipoint, signaling: BGP
VPN ID: 3589, VE-ID: 300, VE-SIZE: 45
RD: 55:3589, RT: import 3589:2, import 3589:14, import 3589:10,
export 3589:1
Bridge-Domain 3589 attachment circuits:
Pseudo-port interface: pseudowire100001
```

Interface	Peer Address	VE-ID	Local Label	Remote Label	S
pseudowire100005	55.0.0.9	90	3121	9064	Y
pseudowire100004	55.0.0.8	10	3085	8036	Y
pseudowire100003	55.0.0.5	5	3080	5020	Y

Last, we will trace the signaling from CSR3 to CSR9. We will use CSR3 again for all calculations. We can see that CSR3 is using the VBO=288 route from CSR9 and rejecting the VBO=1 and VBO=80 ones. This is because the sum of CSR9's VBS (which is constant) and those VBO values must be strictly greater than its local VE-ID. This is the second condition of the sanity check, and since $1 + 45$ and $80 + 45$ are both less than 300, the VBO=288 must be used. The passing sanity check is as follows: $VBO \leq VE-ID < VBS + VBO$ evaluates as $288 \leq 300 < 16 + 288$. Now, CSR3 can find the remote label: $LB + VE-ID - VBO$ evaluates as $9052 + 300 - 288 = 9064$.

```
R3#show l2vpn signaling rib rd 55:3589 detail | begin \.9$
Route 55:3589:90 (epoch:0) from iBGP peer 55.0.0.9
Provisioned (Y) Stale (N)
Route-Target: 3589:2
NLRI [B6000007]
VE-ID:90 VBO:288 VBS:16 LB:9052
MTU: 1500 Control Word: off
RIB Filter [9C000006]
Local RD: 55:3589
VE-ID: 300, VBO: 90, VBS: 45 LB: 3121
Forwarder [F7000001] VFI TREE
```

CSR3 must also compute a local label. We can use a specific show command (below) to look at this specific PW by specifying the next-hop rather than all PWs in the VFI. This uses the local VE-ID with the remote VBO, VBS, and LB values. Because CSR3 has three routes with varying VBOs (1, 90, 270), it selects the one that satisfies the sanity check. VBO=1 does not work because the sum of the VBS and VBO ($45 + 1$) is less than the VE-ID (90); the VBO is too small. VBO=270 does not work because this value is greater than the VE-ID (90); the VBO is too large. The final VBO value of 90 meets the condition: $VBO \leq VE-ID < VBS + VBO$ evaluates as $90 \leq 90 < 45 + 90$. The label is selected as follows: $LB + VE-ID - VBO$ evaluates as $3121 + 90 - 90 = 3121$. We confirm the label pair of local = 3121 and remote = 9064.

```
R3#show l2vpn signaling rib next-hop 55.0.0.9 detail
Route 55:3589:90 (epoch:0) from iBGP peer 55.0.0.9
Provisioned (Y) Stale (N)
Route-Target: 3589:2
NLRI [B6000007]
VE-ID:90 VBO:288 VBS:16 LB:9052
```

```

MTU: 1500 Control Word: off
RIB Filter [9C000006]
Local RD: 55:3589
VE-ID: 300, VBO: 90, VBS: 45 LB: 3121
Forwarder [F7000001] VFI TREE

R3#show l2vpn vfi name TREE
Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No

VFI name: TREE, state: up, type: multipoint, signaling: BGP
VPN ID: 3589, VE-ID: 300, VE-SIZE: 45
RD: 55:3589, RT: import 3589:2, import 3589:14, import 3589:10,
export 3589:1
Bridge-Domain 3589 attachment circuits:
Pseudo-port interface: pseudowire100001
Interface      Peer Address      VE-ID  Local Label  Remote Label  S
pseudowire100005  55.0.0.9          90     3121         9064          Y
pseudowire100004  55.0.0.8          10     3085         8036          Y
pseudowire100003  55.0.0.5          5      3080         5020          Y

```

With the signaling validated, we can verify the CE reachability and routing information. Just like the static E-TREE, we expect CSR3 and CSR8, as roots, to have full reachability. Rather than use exhaustive ping tests, we can check the CE IGP (EIGRP) for neighbors. Each root router sees all other nodes in the VPLS instance.

```

R1#show eigrp address-family ipv4 vrf TREE neighbors
EIGRP-IPv4 VR(TREE) Address-Family Neighbors for AS(3589)
VRF(TREE)
H   Address                Interface      Hold Uptime    SRTT    RTO   Q   Seq
                               (sec)          (ms)          Cnt  Num
2   10.0.0.10               Gi2.531       10 01:01:52    1599   5000   0   38
1   10.0.0.2                Gi2.531       11 01:01:52     75    450   0   36
0   10.0.0.14               Gi2.531       14 01:01:52     27    162   0   28

```

```

R10#show eigrp address-family ipv4 vrf TREE neighbors
EIGRP-IPv4 VR(TREE) Address-Family Neighbors for AS(3589)
VRF(TREE)
H   Address                Interface      Hold Uptime    SRTT    RTO   Q   Seq
                               (sec)          (ms)          Cnt  Num
0   10.0.0.1                Gi2.508       11 01:02:02     9    100   0   45
1   10.0.0.2                Gi2.508       14 12:16:16    17    102   0   34
2   10.0.0.14               Gi2.508       12 14:35:32    27    162   0   26

```

We expect that the spokes can only communicate to the hubs at layer 2. Adjusting IGP parameters, such as split-horizon for EIGRP/RIP, BGP route-reflection rules, and OSPF network types, it is possible to enable spoke-to-spoke communication. We could also adjust how IPv4 ARP and IPv6 ND resolve as seen in previous sections. However, the design of E-TREE was to prevent that in the first place. All leaves should have an identical set of neighbors which is the list of all roots in the E-TREE. CSR2 and XRv4 will

never form EIGRP neighbors with one another unless a PW is provisioned between those two routers. This is controlled by the BGP route-targets as discussed in detail earlier.

```
R2#show eigrp address-family ipv4 vrf TREE neighbors
EIGRP-IPv4 VR(TREE) Address-Family Neighbors for AS(3589)
VRF(TREE)
```

H	Address	Interface	Hold Uptime (sec)	SRTT (ms)	RTO	Q Cnt	Seq Num
0	10.0.0.1	Gi2.529	10 01:03:27	29	174	0	42
1	10.0.0.10	Gi2.529	13 12:17:41	17	102	0	36

```
RP/0/0/CPU0:XRv4#show eigrp vrf TREE neighbors
IPv4-EIGRP VR(TREE) Neighbors for AS(3589) VRF TREE
```

H	Address	Interface	Hold Uptime (sec)	SRTT (ms)	RTO	Q Cnt	Seq Num
0	10.0.0.1	Gi0/0/0/0.554	13 01:03:42	1575	5000	0	46
1	10.0.0.10	Gi0/0/0/0.554	12 14:37:12	1599	5000	0	34

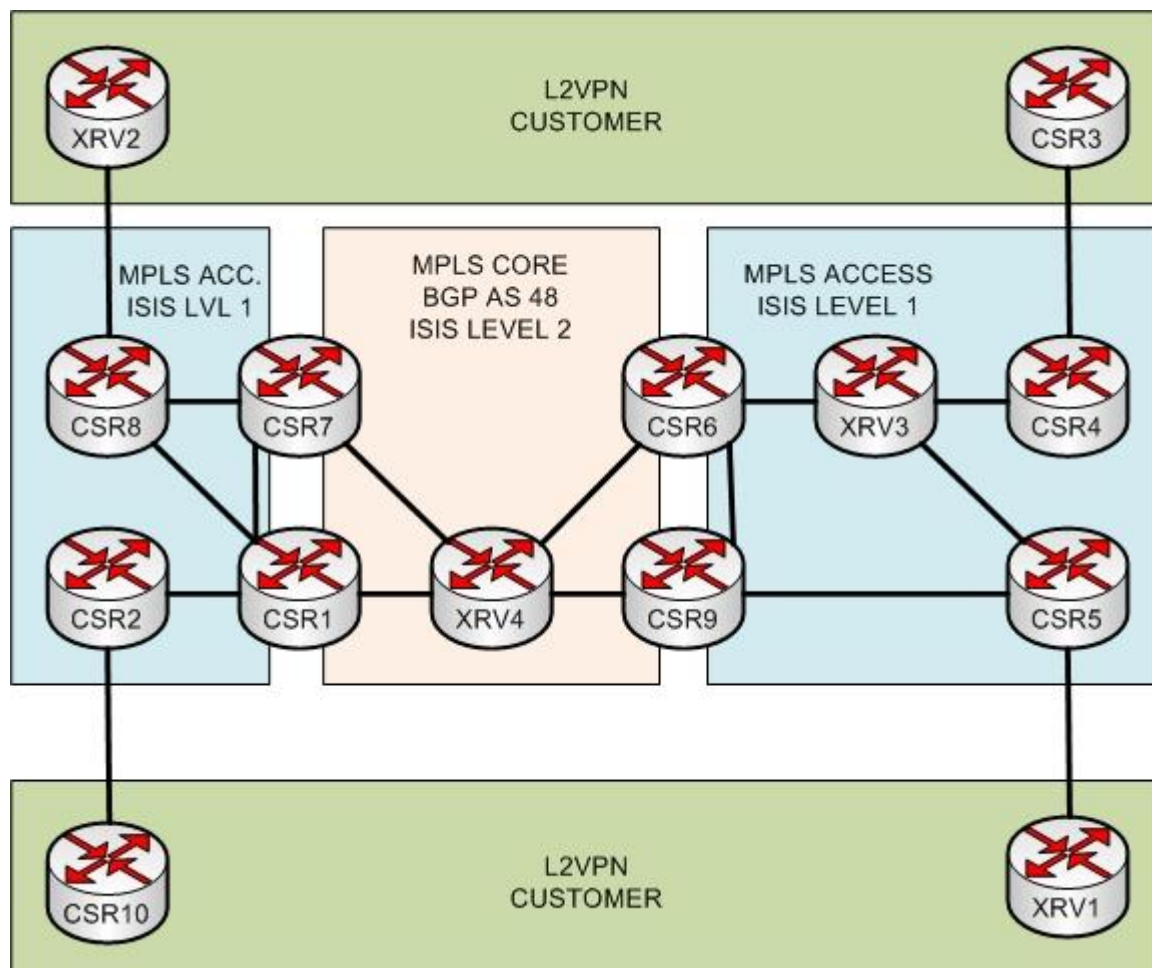
Additional Reading – Reference configurations "l2vpn-bgp-ad"

32.1.3 Hierarchical VPLS (H-VPLS)

H-VPLS is typically implemented using either MPLS or QinQ in the access network, with MPLS in the core network for both options. MPLS is naturally easier on the CSR1000vs since they are routers. QinQ requires some significant workarounds in a virtual environment by using routers for switches, but it is possible. QinQ can be simpler to configure and manage since there isn't a large IP/MPLS network, but MPLS scales better since QinQ switches still need to store all customer MAC addresses in CAM tables for the S-VLAN. MPLS also offers robust TE and FRR capabilities over QinQ. For router-to-router connections, MAC address learning is not a significant concern, but for data-center interconnect (DCI) applications with large layer 2 domains, the MPLS access networks scale better.

32.1.3.1 MPLS in the Access Network

One option to configure H-VPLS is to use MPLS in the access networks. This technique uses basic PWs from the user-facing PE (UPE) devices to the network-facing PE (NPE) devices. This ties the customers into the MPLS core. The NPEs have PWs between one another, typically a full-mesh, so that traffic arriving from an "access" PW can be transported to any other NPE for delivery to a CE. The "new" syntax makes this very straightforward as we do not have to explicitly disable split-horizon on the NPE-UPE circuits. The new syntax also supports sequencing over VPLS, plus FAT-PW and any other new features added in the future. All configurations in this test use the new syntax. The diagram below shows an MPLS core flanked by two MPLS access networks. The network is broken into IS-IS levels with a common area, which prevents the attached-bit from being set. None of the level-1 domains have a default route, but have the loopbacks of the L1/L2 routers to build PWs. This is done for additional scalability and works well with H-VPLS designs. CSR2, CSR8, CSR4, and CSR5 are the UPE devices which cross-connect EVCs into access PWs. CSR1, CSR6, CSR7, and CSR9 are NPEs that bridge VPLS instances with access PWs. XRv3 and XRv4 are P routers and the remaining devices outside of the SP network are CE devices.



H-VPLS can be very configuration intensive. BGP auto-discovery can be used for NPE pseudowires, but two restrictions apply. First, you must use LDP for signaling, as bridging different PW encapsulations or signaling strategies is not supported. This is true for manual or auto-discovered PWs. Second, you cannot use backup PWs in the access network when using any kind of auto-discovery. Below is an example of the first restriction where the parser does not allow the bridging between a P2P LDP PW and dynamically-discovered VPLS PW, signaled by BGP.

```
R1(config-bdomain)#member vfi VPLS_102
R1(config-bdomain)#member pseudowire 12102
% VFI is using BGP signaling protocol, manual pseudowires are not allowed.
```

In this example, there are two VPNs, referred to as 101 and 102 hereafter. Each CE has a VRF for each for testing purposes, and each UPE has an EVC for each to provide the Ethernet service. VPN 101 has a full mesh of manually configured PWs between all NPEs. CSR8 and CSR5 also have a pair of PWs going to each NPE within their access networks for redundancy; this redundancy forces the core to use manual PWs versus BGP auto-discovery. CSR2 and CSR4 do not have any redundant connections. For brevity, only parts of the configuration are shown, focusing on CSR8 and CSR7. CSR8 is the UPE which controls the redundancy; it has access PWs to two NPEs (CSR7 is primary) and binds these PWs to the EVC using

the L2VPN cross-connect. CSR7 is the NPE which has a full-mesh of network-facing PWs to CSR7, CSR6, and CSR9. These circuits inherit from a template and I enable many additional features for variety. It also terminates the access PW from CSR8; it does not control the redundancy, which means the configuration on CSR1 and CSR7 will be the same, minus circuit enumerations. The VPLS instance and the access PW are bridged in a bridge-domain, allowing layer 2 traffic to flow both ways.

```
! CSR8
interface GigabitEthernet2
  service instance 101 ethernet
    encapsulation dot1q 3582 second-dot1q 101
    rewrite ingress tag pop 2 symmetric

interface pseudowire78101
  description UPE VPLS 101 PRIMARY
  encapsulation mpls
  neighbor 48.0.0.7 78101

interface pseudowire18101
  description UPE VPLS 101 BACKUP
  encapsulation mpls
  neighbor 48.0.0.1 18101

l2vpn xconnect context VPLS_101
  member pseudowire78101 group VPLS_101
  member pseudowire18101 group VPLS_101 priority 1
  member GigabitEthernet2 service-instance 101

! CSR7
template type pseudowire TMP_NPE_PW
  encapsulation mpls
  sequencing both
  control-word include
  load-balance flow ip src-ip
  load-balance flow-label both

interface pseudowire17101
  description CSR1 TO CSR7 VPLS_101
  source template type pseudowire TMP_NPE_PW
  neighbor 48.0.0.1 17101

interface pseudowire67101
  description CSR6 TO CSR7 VPLS_101
  source template type pseudowire TMP_NPE_PW
  neighbor 48.0.0.6 67101

interface pseudowire78101
  description UPE VPLS 101 PRIMARY
  encapsulation mpls
```

```

neighbor 48.0.0.8 78101

interface pseudowire79101
description CSR7 TO CSR9 VPLS_101
source template type pseudowire TMP_NPE_PW
neighbor 48.0.0.9 79101

l2vpn vfi context VPLS_101
vpn id 101
member pseudowire79101
member pseudowire67101
member pseudowire17101

bridge-domain 101
member vfi VPLS_101
member pseudowire78101

```

The configuration of the CE devices is not shown. Each CE device sends traffic to the UPE with an inner VLAN tag of 101 or 102 and an outer VLAN tag of whatever metro VLAN was assigned by the provider (varies per site). EIGRP is running inside VPN 101 with OSPF running inside VPN 102. Each VPN has a loopback on the CE device to ensure routing works properly over the L2VPN. The P routers are just MPLS/LDP enabled and have no fancy configuration. We will begin in the core by verifying the PWs are operational. The output shows us all circuits within the L2VPN, even though we know 78101 is an access circuit not strictly part of the VPLS instance. The “detail” keyword shows us the MPLS labels also. Notice that CSR1 shows “standby” for the circuit to CSR8; this is the backup access PW, but CSR1 learned this standby information via status message, not local configuration. We will use a different verification command on every router for variety.

```

R7#show l2vpn service vfi name VPLS_101 detail | begin Interface

```

Interface	Group	Encapsulation	Prio	St	XC	St
-----	-----	-----	----	--	-----	
VPLS name: VPLS_101, State: UP						
pw100008		VPLS_101 (VFI)	0	UP	UP	
pw78101	core_pw	48.0.0.8:78101 (MPLS)	0	UP	UP	
		Local VC label 7004				
		Remote VC label 8005				
pw79101	core_pw	48.0.0.9:79101 (MPLS)	0	UP	UP	
		Local VC label 7013				
		Remote VC label 9002				
pw67101	core_pw	48.0.0.6:67101 (MPLS)	0	UP	UP	
		Local VC label 7007				
		Remote VC label 6010				
pw17101	core_pw	48.0.0.1:17101 (MPLS)	0	UP	UP	
		Local VC label 7038				

Remote VC label 1005

R1#show l2vpn service vfi name VPLS_101 | begin Interface

Interface	Group	Encapsulation	Prio	St	XC	St

VPLS name: VPLS_101, State: UP						
pw100001		VPLS_101 (VFI)	0	UP		UP
pw12101	core_pw	48.0.0.2:12101 (MPLS)	0	UP		UP
pw18101	core_pw	48.0.0.8:18101 (MPLS)	0	SB		SB
pw19101	core_pw	48.0.0.9:19101 (MPLS)	0	UP		UP
pw16101	core_pw	48.0.0.6:16101 (MPLS)	0	UP		UP
pw17101	core_pw	48.0.0.7:17101 (MPLS)	0	UP		UP

R7#show mpls l2transport vc 60000 80000

Local intf	Local circuit	Dest address	VC ID	Status

VFI VPLS_101	vfi	48.0.0.6	67101	UP
VFI VPLS_101	vfi	48.0.0.8	78101	UP
VFI VPLS_102	vfi	48.0.0.8	78102	UP
VFI VPLS_101	vfi	48.0.0.9	79101	UP

R9#show l2vpn atom vc service-name VPLS_101

Service					
Interface	Peer ID	VC ID	Type	Name	Status

pw19101	48.0.0.1	19101	vfi	VPLS_101	UP
pw59101	48.0.0.5	59101	vfi	VPLS_101	UP
pw69101	48.0.0.6	69101	vfi	VPLS_101	UP
pw79101	48.0.0.7	79101	vfi	VPLS_101	UP

We can also verify that the access PWs have split-horizon automatically disabled. When this feature is enabled (it is by default), traffic entering from one PW cannot exit through another PW; the expectation is that it is manually bridged elsewhere or piped to the AC for delivery. There are no ACs on the NPEs, so traffic entering from the access PW should be allowed to transit the network PWs, and vice versa. What remains disallowed is inter-network PW forwarding, which explains the need for a full-mesh of NPE routers like traditional VPLS requires for E-LAN service. The commands below verify that CSR7 and CSR1 both have split-horizon disabled on their access PWs. Split-horizon remains enabled for all network PWs as expected. CSR1 uses the new show command while CSR7 uses the legacy show command.

R1#show l2vpn vfi name VPLS_101

Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No

VFI name: VPLS_101, state: up, type: multipoint, signaling: LDP

VPN ID: 101

Bridge-Domain 101 attachment circuits:

Pseudo-port interface: pseudowire100001

Interface	Peer Address	VC ID	
pseudowire12101	48.0.0.2	12101	S
			N

pseudowire18101	48.0.0.8	18101	N
pseudowire19101	48.0.0.9	19101	Y
pseudowire16101	48.0.0.6	16101	Y
pseudowire17101	48.0.0.7	17101	Y

R7#show vfi name VPLS_101

Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No

VFI name: VPLS_101, state: up, type: multipoint, signaling: LDP

VPN ID: 101

Bridge-Domain 101 attachment circuits:

Neighbors connected via pseudowires:

Peer Address	VC ID	S
48.0.0.8	78101	N
48.0.0.9	79101	Y
48.0.0.6	67101	Y
48.0.0.1	17101	Y

To verify all of the connectivity and signaling is working as expected, we will trace the path from XRv2 to CSR3 within VPN 101. Traffic arrives at CSR8 with dot1q tags {3582 101} as we saw earlier. Both tags are popped by the EVC and the native Ethernet frame is piped to the access PW 78101 using PW label 7004. The transport label is implicit-null since the NPE is one hop away. The full label stack is simply 7004.

R8#show l2vpn atom vc service-name VPLS_101

Interface	Peer ID	VC ID	Service		Status
			Type	Name	
pw18101	48.0.0.1	18101	p2p	VPLS_101	STANDBY
pw78101	48.0.0.7	78101	p2p	VPLS_101	UP

R8#show l2vpn atom binding 78101 | include Remote

Remote Label: 7004

R8#show mpls ldp bindings 48.0.0.7 32 neighbor 48.0.0.7

lib entry: 48.0.0.7/32, rev 8

remote binding: lsr: 48.0.0.7:0, label: imp-null

R8#show l2vpn atom vc vcid 78101 detail | include label_stack

Output interface: Gi2.578, imposed label stack {7004}

When the MPLS packet arrives at CSR7, the LFIB shows all labels being removed with no obvious next-hop. This is because the PW terminates and is passed into a bridge-domain so the LFIB does not directly perform any label swaps on ingress. We see several MAC addresses learned by the bridge-domain, and identify XRv2 as being present. Normal switching rules apply at this point, and the destination of the Ethernet frame determines the egress PW. This is why the LFIB did not try to route the MPLS packet any further.

R7#show mpls forwarding-table labels 7004 detail

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Label	Outgoing interface	Next Hop
7004	No Label	l2ckt(6)	0		none	point2point

MAC/Encaps=0/0, MRU=0, Label Stack{}

No output feature configured

```
RP/0/0/CPU0:XRv2#show interfaces gig0/0/0/0 | include bia
Hardware is GigabitEthernet, address is 0050.56a9.0e6f (bia 0050.56a9.0e6f)
```

```
R7#show bridge-domain 101
Bridge-domain 101 (4 ports in all)
State: UP                               Mac learning: Enabled
Aging-Timer: 300 second(s)
```

```
vfi VPLS_101 neighbor 48.0.0.8 78101
vfi VPLS_101 neighbor 48.0.0.1 17101
vfi VPLS_101 neighbor 48.0.0.6 67101
vfi VPLS_101 neighbor 48.0.0.9 79101
```

AED	MAC	address	Policy	Tag	Age	Pseudoport
0	0050.56A9.9C60		forward	dynamic	297	VPLS_101.1004017
1	FFFF.FFFF.FFFF		flood	static	0	OLIST_PTR:0xe80a4420
0	0050.56A9.8CCF		forward	dynamic	297	VPLS_101.1004016
0	0050.56A9.0E6F		forward	dynamic	298	VPLS_101.1004014
0	0050.56A9.F961		forward	dynamic	296	VPLS_101.1004015

Assuming the destination MAC was CSR3 (MAC is below) the packet is routed out of a pseudo-port that is mapped to the PW connected to the NPE towards CSR3. In this case, it is CSR6. I have not been able to find a way to show the pseudo-port to PW mappings, so we will make the assumption without verifying as doing so appears impossible via show commands. The new label stack is {94008 6010} which includes an LDP transport label from XRv4 to reach CSR6's loopback, and the PW label for this circuit.

```
R3#show interfaces gigabitEthernet 2 | include bia
Hardware is CSR vNIC, address is 0050.56a9.8ccf (bia 0050.56a9.8ccf)
```

```
R7#show l2vpn atom vc vcid 67101 detail | include label_stack
Output interface: Gi2.574, imposed label stack {94008 6010}
```

XRv4 performs PHP and delivers packets with label 6010 to CSR6.

```
RP/0/0/CPU0:XRv4#show mpls forwarding labels 94008
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94008	Pop	48.0.0.6/32	Gi0/0/0/0.564	48.6.14.6	6216112

Just like CSR7, packets arriving on this PW are stripped of all labels and delivered to the bridge-domain. The LFIB cannot decide where the Ethernet frame should go, but the BD CAM table can. We can see CSR3's MAC address referencing a pseudo-port.

```
R6#show mpls forwarding-table labels 6010 detail
Local      Outgoing    Prefix          Bytes Label    Outgoing    Next Hop
Label      Label        or Tunnel Id    Switched       interface
6010       No Label     12ckt(6)        0              none         point2point
          MAC/Encaps=0/0, MRU=0, Label Stack{}
          No output feature configured
```

```
R6#show bridge-domain 101
Bridge-domain 101 (5 ports in all)
State: UP                               Mac learning: Enabled
Aging-Timer: 300 second(s)
vfi VPLS_101 neighbor 48.0.0.4 46101
vfi VPLS_101 neighbor 48.0.0.7 67101
vfi VPLS_101 neighbor 48.0.0.1 16101
vfi VPLS_101 neighbor 48.0.0.9 69101
vfi VPLS_101 neighbor 48.0.0.5 56101
AED MAC address      Policy Tag      Age Pseudoport
0  0050.56A9.9C60 forward dynamic 296 VPLS_101.100401d
1  FFFF.FFFF.FFFF flood  static  0  OLIST_PTR:0xe80d4850
0  0050.56A9.8CCF forward dynamic 297 VPLS_101.1004016
0  0050.56A9.0E6F forward dynamic 299 VPLS_101.100401e
0  0050.56A9.F961 forward dynamic 295 VPLS_101.100401f
```

Again, we cannot prove that the pseudo-port associated with CSR3's MAC address terminates on CSR4, but we will assume it does and continue. The PW label of 4000 is pushed to reference this circuit on CSR4 and the transport label of 93002 is pushed to reach CSR4's loopback.

```
R6#show l2vpn atom vc vcid 46101 detail | include label_stack
Output interface: Gi2.563, imposed label stack {93002 4000}
```

Like XRv4, XRv3 performs PHP to expose the PW label to CSR4.

```
RP/0/0/CPU0:XRv3#show mpls forwarding labels 93002
Local  Outgoing    Prefix          Outgoing    Next Hop    Bytes
Label  Label        or ID           Interface    Interface    Switched
-----
93002  Pop          48.0.0.4/32     Gi0/0/0/0.543 48.4.13.4    5408591
```

CSR4, as a UPE, has similar output as the NPEs CSR7 and CSR6 when packets arrive along a PW. The difference is that the outgoing interface specifies Gig2, which is correct, as there is an EVC mapped to this PW in the cross-connect process. Label 4000 maps to this cross-connect, and the outgoing dot1q encapsulation is shown as {3534 101} upon receiving the raw Ethernet frame.

```
R4#show mpls forwarding-table labels 4000 detail
Local      Outgoing    Prefix          Bytes Label    Outgoing    Next Hop
Label      Label        or Tunnel Id    Switched       interface
4000       No Label     12ckt(6)        0              none         point2point
          MAC/Encaps=0/0, MRU=0, Label Stack{}
          No output feature configured
```

```

4000      No Label      12ckt(1)      2501064      Gi2      point2point
MAC/Encaps=0/0, MRU=0, Label Stack{}
No output feature configured

```

```

R4#show l2vpn service xconnect name VPLS_101 detail | begin Interface
  Interface      Group      Encapsulation      Prio  St  XC St
  -----
VPWS name: VPLS_101, State: UP
  Gi2      Gi2:3534/101 (Eth VLAN)      0      UP  UP
            Interworking: ethernet
  pw46101  48.0.0.6:46101 (MPLS)      0      UP  UP
            Local VC label 4000
            Remote VC label 6003

```

Now we will verify the data-plane using EPC. EPC currently does not appear to support EVC monitoring, so we will limit our focus to the NPEs. Assuming the proper labels are imposed as traffic is bridged between different PWs, our control-plane verification shows the MPLS transport functions. Capturing inbound on CSR7 facing CSR8, and outbound on CSR7 facing XRv4, we can see label 7004 (0x1B5C) arriving and labels {94008 6010} (0x16F38, 0x177A) exiting. The FAT-PW label is also inserted beneath the PW label to achieve load-sharing on the network PW only (documented elsewhere) and the control-word is added last in green. The low-order 16 bits of the CW representing sequencing which is enabled on all network PWs, along with FAT-PW. CSR3's MAC is in cyan and is the destination MAC address in the customer frame. CSR3's IP address is in pink and is the destination IP address of 192.168.0.3.

```

R7#show monitor capture CAP buffer dump
2
0000: 005056A9 EA770050 56A9FB1C 81000DFA .PV..w.PV.....
0010: 884701B5 C1FF0000 00000050 56A98CCF .G.....PV...
0020: 005056A9 0E6F0800 450001F4 00000000 .PV..o..E.....
0030: FF0138A9 C0A8000C C0A80003 080065A0 ..8.....e.
0040: 30C50000 ABCDABCD ABCDABCD ABCDABCD 0.....

3
0000: 005056A9 862A0050 56A9EA77 81000DF6 .PV...*.PV..w....
0010: 884716F3 80FF0177 A0FF8000 01010000 .G.....w.....
0020: 24C10050 56A98CCF 005056A9 0E6F0800 $.PV....PV..o..
0030: 450001F4 00000000 FF0138A9 C0A8000C E.....8.....
0040: C0A80003 080065A0 30C50000 ABCDABCD .....e.0.....

```

We perform a similar capture on CSR6, this time capturing inbound from XRv4 and outbound towards XRv3. The colors used above are used again below and the only significant change is the label values. Label 6010 arrives, along with the FAT-PW label and CW. The frame is switched by the BD CAM table and gets labels {93002 4000} (0x16B4A, 0xFA0) imposed along the access PW towards CSR4.

```

R6#show monitor capture CAP buffer dump
10

```

```

0000: 005056A9 DE0D0050 56A9862A 81000DEC .PV....PV...*....
0010: 88470177 A0FE8000 01010000 253E0050 .G.w.....%>.P
0020: 56A98CCF 005056A9 0E6F0800 450001F4 V....PV..o..E...
0030: 00000000 FF0138A9 C0A8000C C0A80003 .....8.....
0040: 080045A0 50C50000 ABCDABCD ABCDABCD ..E.P.....

```

11

```

0000: 005056A9 EA540050 56A9DE0D 81000DEB .PV..T.PV.....
0010: 884716B4 A0FF00FA 01FF0000 00000050 .G.....P
0020: 56A98CCF 005056A9 0E6F0800 450001F4 V....PV..o..E...
0030: 00000000 FF0138A9 C0A8000C C0A80003 .....8.....
0040: 080045A0 50C50000 ABCDABCD ABCDABCD ..E.P.....

```

We can quickly verify that EIGRP is functioning properly inside the H-VPLS network without having to trace all LSPs individually. Since VPN 101 provides E-LAN service, we expect each router to have 3 EIGRP neighbors.

RP/0/0/CPU0:XRv2#show eigrp vrf 101 neighbors

IPv4-EIGRP VR(VPLS_101) Neighbors for AS(101) VRF 101

H	Address	Interface	Hold Uptime (sec)	SRTT (ms)	RTO	Q Cnt	Seq Num
1	192.168.101.10	Gi0/0/0/0.101	14 01:31:06	1	200	0	24
2	192.168.101.11	Gi0/0/0/0.101	14 12:15:05	24	200	0	19
0	192.168.101.3	Gi0/0/0/0.101	10 12:16:20	10	200	0	18

RP/0/0/CPU0:XRv1#show eigrp vrf 101 neighbors

IPv4-EIGRP VR(VPLS_101) Neighbors for AS(101) VRF 101

H	Address	Interface	Hold Uptime (sec)	SRTT (ms)	RTO	Q Cnt	Seq Num
0	192.168.101.10	Gi0/0/0/0.101	11 01:30:35	8	200	0	30
2	192.168.101.12	Gi0/0/0/0.101	14 12:15:22	344	2064	0	20
1	192.168.101.3	Gi0/0/0/0.101	11 12:15:22	13	200	0	17

R3#show eigrp address-family ipv4 vrf 101 neighbors

EIGRP-IPv4 VR(VPLS_101) Address-Family Neighbors for AS(101)
VRF(101)

H	Address	Interface	Hold Uptime (sec)	SRTT (ms)	RTO	Q Cnt	Seq Num
0	192.168.101.10	Gi2.101	12 01:30:53	57	342	0	27
2	192.168.101.11	Gi2.101	11 12:15:40	61	366	0	19
1	192.168.101.12	Gi2.101	10 12:16:55	35	210	0	20

R10#show eigrp address-family ipv4 vrf 101 neighbors

EIGRP-IPv4 VR(VPLS_101) Address-Family Neighbors for AS(101)
VRF(101)

H	Address	Interface	Hold Uptime (sec)	SRTT (ms)	RTO	Q Cnt	Seq Num
2	192.168.101.11	Gi2.101	14 01:31:04	68	408	0	19

1	192.168.101.3	Gi2.101	11 01:31:04	43	258	0	20
0	192.168.101.12	Gi2.101	14 01:31:52	30	180	0	20

We will also verify our backup PWs are functional on CSR8. To test it, we can shutdown the PW to CSR7 from the perspective of the UPE. Since the PWs are LDP-signaled and status is supported, the backup PW will come online and report this to the backup NPE, which is CSR1 in this case. The backup PW is pre-made and already signaled, so the LSPs should be functional. The PW shows “admin down” on CSR8, which means the PW to CSR1 moves from “standby” to “up”.

```
! CSR8
interface pseudowire78101
shutdown

R8#show l2vpn service xconnect name VPLS_101 | begin Interface
  Interface          Group          Encapsulation          Prio   St   XC   St
  -----          -
  VPWS name: VPLS_101, State: UP
  pw78101           VPLS_101      48.0.0.7:78101 (MPLS)    0      AD   AD
  pw18101           VPLS_101      48.0.0.1:18101 (MPLS)    1      UP   UP
  Gi2               Gi2:3582/101 (Eth VLAN) 0      UP   UP

! CSR7
%XCONNECT-5-PW_STATUS: MPLS peer 48.0.0.8 vcid 78101, VC state DOWN, PW Err

R7#show mpls l2transport vc 78101 detail | include Last_error
  Last error: Remote peer withdrew label

! CSR1
%XCONNECT-5-PW_STATUS: MPLS peer 48.0.0.8 vcid 18101, VC state UP
```

Checking the control-plane, we see that CSR8 will use PW label 1003 for the backup PW to CSR1. The transport label is implicit-null since CSR8 is the penultimate hop to CSR1 along this LSP.

```
R8#show l2vpn atom binding 18101 | include Remote
  Remote Label: 1003

R8#show mpls ldp bindings 48.0.0.1 32 neighbor 48.0.0.1
  lib entry: 48.0.0.1/32, rev 10
  remote binding: lsr: 48.0.0.1:0, label: imp-null

R8#show l2vpn atom vc vcid 18101 detail | include label_stack
  Output interface: Gi2.518, imposed label stack {1003}
```

CSR1 has CSR3’s MAC address in its BD CAM, and we assume that pseudo-port associated with the MAC address maps to the CSR6 network PW. This imposes labels {94008 6030}. The LDP label is the same one

used by CSR7 earlier to reach CSR6's loopback, which was allocated by XRv4. The PW label is specific to the network PW between CSR1 and CSR6.

```
R1#show bridge-domain 101
Bridge-domain 101 (5 ports in all)
State: UP                               Mac learning: Enabled
Aging-Timer: 300 second(s)
  vfi VPLS_101 neighbor 48.0.0.9 19101
  vfi VPLS_101 neighbor 48.0.0.2 12101
  vfi VPLS_101 neighbor 48.0.0.8 18101
  vfi VPLS_101 neighbor 48.0.0.6 16101
  vfi VPLS_101 neighbor 48.0.0.7 17101
AED MAC address      Policy Tag      Age Pseudoport
0  0050.56A9.9C60 forward dynamic 298 VPLS_101.1004015
1  FFFF.FFFF.FFFF flood  static  0  OLIST_PTR:0xe8057800
0  0050.56A9.8CCF forward dynamic 298 VPLS_101.1004018
0  0050.56A9.0E6F forward dynamic 296 VPLS_101.1004017
0  0050.56A9.F961 forward dynamic 296 VPLS_101.1004016

R1#show l2vpn atom vc vcid 16101 detail | include label_stack
Output interface: Gi2.514, imposed label stack {94008 6030}
```

XRv4 will pop label 94008 and deliver packets labeled 6030 to CSR6 (not shown again). CSR6 strips the PW label, delivers the frame to the BD CAM, which switches it towards CSR4. This has not changed from the previous example since there was no instability in the remote access MPLS network. A quick check on XRv2 shows that it still has all EIGRP neighbors, which implies connectivity is functional along the backup access PW.

```
RP/0/0/CPU0:XRv2#show eigrp vrf 101 neighbors
IPv4-EIGRP VR(VPLS_101) Neighbors for AS(101) VRF 101
```

H	Address	Interface	Hold Uptime (sec)	SRTT (ms)	RTO	Q Cnt	Seq Num
1	192.168.101.10	Gi0/0/0/0.101	14 01:47:02	1	200	0	24
2	192.168.101.11	Gi0/0/0/0.101	14 12:31:01	24	200	0	19
0	192.168.101.3	Gi0/0/0/0.101	11 12:32:16	10	200	0	18

Next, we will configure and test VPN 102. This will provide an E-TREE service and will use BGP auto-discovery with LDP signaling for NPE PWs. CSR6 and CSR9 are roots while CSR1 and CSR7 are leaves; this essentially means CSR1 and CSR7 will not have a PW between them. This implies that all UPEs homed into CSR1 or CSR7 will not be able to communicate with one another. This can be used to scale E-TREE architectures by segmenting groups of nodes, and treating those groups holistically as roots or leaves. In this case, the design will prohibit CSR10 and XRv2 from communicating. PW redundancy on the UPE is not supported with BGP auto-discovery, so there are no backup PWs for VPN 102. The UPE configuration is identical to VPN 101 (minus the backup BW), but the NPEs are more involved. We focus on CSR1 and

CSR9 for brevity. RT:48:1 represents the root RT and RT:48:2 represents the leaf RT. Roots import both RTs, while leaves only import the root RT. The NPE template is not shown again.

```
! CSR2
interface GigabitEthernet2
  service instance 102 ethernet
  encapsulation dot1q 3520 second-dot1q 102
  rewrite ingress tag pop 2 symmetric

interface pseudowire12102
  description UPE VPLS 102
  encapsulation mpls
  neighbor 48.0.0.1 12102

l2vpn xconnect context VPLS_102
  member GigabitEthernet2 service-instance 102
  member pseudowire12102

! CSR1
interface pseudowire12102
  description UPE VPLS_102
  encapsulation mpls
  neighbor 48.0.0.2 12102
  signaling protocol ldp

l2vpn vfi context VPLS_102
  vpn id 102
  autodiscovery bgp signaling ldp template TMP_NPE_PW
  route-target import 48:1
  route-target export 48:2
  no auto-route-target

bridge-domain 102
  member vfi VPLS_102
  member pseudowire12102

! CSR9
interface pseudowire59102
  description UPE VPLS 102
  encapsulation mpls
  neighbor 48.0.0.5 59102
  signaling protocol ldp

l2vpn vfi context VPLS_102
  vpn id 102
  autodiscovery bgp signaling ldp template TMP_NPE_PW
  route-target export 48:1
  route-target import 48:1
  route-target import 48:2
```



```

no auto-route-target

bridge-domain 102
member vfi VPLS_102
member pseudowire59102

```

We can quickly verify that the RT import/export policies were configured properly by checking the L2VPN VPLS prefixes exchanged between NPEs. CSR1 imports the routes from CSR6 and CSR9, but not CSR7. CSR9 imports them all.

```

R1#show bgp l2vpn vpls rd 48:102 | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 48:102
*> 48:102:48.0.0.1/96
                                0.0.0.0                                32768 ?
*>i 48:102:48.0.0.6/96
                                48.0.0.6                                0    100    0 ?
*>i 48:102:48.0.0.9/96
                                48.0.0.9                                0    100    0 ?

```

```

R9#show bgp l2vpn vpls rd 48:102 | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 48:102
*>i 48:102:48.0.0.1/96
                                48.0.0.1                                0    100    0 ?
*>i 48:102:48.0.0.6/96
                                48.0.0.6                                0    100    0 ?
*>i 48:102:48.0.0.7/96
                                48.0.0.7                                0    100    0 ?
*> 48:102:48.0.0.9/96
                                0.0.0.0                                32768 ?

```

Verifying the VFI configuration and auto-discovered PWs verifies which PWs were built based on the BGP L2VPN VPLS prefixes. We can see the RT import/export policies as well to validate them. CSR1 shows specific import/export RTs, while CSR9 does not specify a direction for RT:48:1, since it is both imported and exported. While we are here, we can also verify that split-horizon is disabled on the access PW, which allows traffic to transit between the NPE-signaled E-TREE and the access circuits to the UPEs.

```

R1#show l2vpn vfi name VPLS_102
Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No
VFI name: VPLS_102, state: up, type: multipoint, signaling: LDP
VPN ID: 102, VPLS-ID: 48:102
RD: 48:102, RT: import 48:1, export 48:2
Bridge-Domain 102 attachment circuits:
Pseudo-port interface: pseudowire100002
Interface      Peer Address      VC ID      Discovered Router ID      S
pseudowire100004  48.0.0.9          102        48.0.0.9                  Y

```

pseudowire100003	48.0.0.6	102	48.0.0.6	Y
pseudowire12102	48.0.0.2	12102	n/a	N

R9#show l2vpn vfi name VPLS_102

Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No

VFI name: VPLS_102, state: up, type: multipoint, signaling: LDP

VPN ID: 102, VPLS-ID: 48:102

RD: 48:102, RT: 48:1, import 48:2

Bridge-Domain 102 attachment circuits:

Pseudo-port interface: pseudowire100006

Interface	Peer Address	VC ID	Discovered Router ID	S
pseudowire100011	48.0.0.1	102	48.0.0.1	Y
pseudowire100009	48.0.0.7	102	48.0.0.7	Y
pseudowire100008	48.0.0.6	102	48.0.0.6	Y
pseudowire59102	48.0.0.5	59102	n/a	N

Next, we will verify the state of the PWs. We expect them all to be up, including the UPE circuits.

R1#show l2vpn service vfi name VPLS_102 | begin Interface

Interface	Group	Encapsulation	Prio	St	XC	St
-----	-----	-----	----	--	-----	
VPLS name: VPLS_102, State: UP						
pw100002		VPLS_102 (VFI)	0	UP	UP	
pw100004	core_pw	48.0.0.9:102 (MPLS)	0	UP	UP	
pw100003	core_pw	48.0.0.6:102 (MPLS)	0	UP	UP	
pw12102	core_pw	48.0.0.2:12102 (MPLS)	0	UP	UP	

R9#show l2vpn service vfi name VPLS_102 | begin Interface

Interface	Group	Encapsulation	Prio	St	XC	St
-----	-----	-----	----	--	-----	
VPLS name: VPLS_102, State: UP						
pw100006		VPLS_102 (VFI)	0	UP	UP	
pw100011	core_pw	48.0.0.1:102 (MPLS)	0	UP	UP	
pw100009	core_pw	48.0.0.7:102 (MPLS)	0	UP	UP	
pw100008	core_pw	48.0.0.6:102 (MPLS)	0	UP	UP	
pw59102	core_pw	48.0.0.5:59102 (MPLS)	0	UP	UP	

Next, we will quickly trace the LSP from CSR2 to CSR5, supporting customers CSR10 and XRv1. CSR2 receives packets in the EFP with dot1q tags {3520 102} and pops both on ingress (shown earlier). This is piped via access PW 12102 to CSR1 using label 1017. We prove all of this with one show command. We also lookup XRv1's MAC address so we know what to reference in the NPE BD CAM tables.

R2#show l2vpn service xconnect name VPLS_102 detail | begin Interface

Interface	Group	Encapsulation	Prio	St	XC	St
-----	-----	-----	----	--	-----	
VPWS name: VPLS_102, State: UP						
Gi2		Gi2:3520/102 (Eth VLAN)	0	UP	UP	
Interworking: ethernet						

```

pw12102                                48.0.0.1:12102 (MPLS)           0      UP    UP
                                         Local VC label 2001
                                         Remote VC label 1017

```

```

RP/0/0/CPU0:XRv1#show interfaces gigabitEthernet 0/0/0/0 | include bia
Hardware is GigabitEthernet, address is 0050.56a9.9c60 (bia 0050.56a9.9c60)

```

CSR1 bridges this PW to BD 102. The BD determines the outbound pseudo-port, which we assume maps to the NPE PW towards CSR9. CSR1 imposes labels {94006 9024}; the top most is an LDP label from XRv4 describing the path to CSR9, and the bottom is the PW label associated with the auto-discovered PW.

```

R1#show bridge-domain 102
Bridge-domain 102 (3 ports in all)
State: UP                               Mac learning: Enabled
Aging-Timer: 300 second(s)
  vfi VPLS_102 neighbor 48.0.0.6 102
  vfi VPLS_102 neighbor 48.0.0.9 102
  vfi VPLS_102 neighbor 48.0.0.2 12102
AED MAC address      Policy Tag        Age Pseudoport
0  0050.56A9.8CCF forward dynamic  290 VPLS_102.100401a
1  FFFF.FFFF.FFFF flood  static    0  OLIST_PTR:0xe8057830
0  0050.56A9.9C60 forward dynamic  295 VPLS_102.100401b
0  0050.56A9.F961 forward dynamic  296 VPLS_102.100401c

```

```

R1#show l2vpn atom vc destination 48.0.0.9 vcid 102 detail | include
label_stack
Output interface: Gi2.514, imposed label stack {94006 9024}

```

XRv4 performs a pop operation via PHP and delivers the packet with PW label exposed to CSR9.

```

RP/0/0/CPU0:XRv4#show mpls forwarding labels 94006
Local  Outgoing  Prefix          Outgoing      Next Hop      Bytes
Label  Label      or ID          Interface     Interface     Switched
-----
94006  Pop          48.0.0.9/32    Gi0/0/0/0.594 48.9.14.9    4168638

```

Like CSR1, CSR9 bridges this PW to BD 102. The BD determines the outbound pseudo-port, which we assume maps to the NPE PW towards CSR5. Notice that CSR9 has one more MAC in its BD CAM than CSR1 did; this is because CSR1 does not have the MAC address for XRv2, which is behind a leaf node in the E-TREE. CSR9 pushes label 5006 which is the PW label for the access circuit to the UPE. The transport label is implicit-null since CSR9 is the penultimate hop along this LSP.

```

R9#show bridge-domain 102
Bridge-domain 102 (4 ports in all)
State: UP                               Mac learning: Enabled
Aging-Timer: 300 second(s)

```

```

vfi VPLS_102 neighbor 48.0.0.5 59102
vfi VPLS_102 neighbor 48.0.0.1 102
vfi VPLS_102 neighbor 48.0.0.6 102
vfi VPLS_102 neighbor 48.0.0.7 102
AED MAC address      Policy Tag      Age Pseudoport
0 0050.56A9.0E6F forward dynamic 299 VPLS_102.100401a
0 0050.56A9.8CCF forward dynamic 295 VPLS_102.1004019
1 FFFF.FFFF.FFFF flood static 0 OLIST_PTR:0xe8147470
0 0050.56A9.9C60 forward dynamic 299 VPLS_102.1004017
0 0050.56A9.F961 forward dynamic 299 VPLS_102.100401c

```

```

R9#show l2vpn atom vc vcid 59102 detail | include label_stack
Output interface: Gi2.559, imposed label stack {5006}

```

CSR5 receives packets labeled with 5006 along VCID 59102 and pipes them to an EFP with dot1q encapsulation {3551 102} via cross-connect. This is all displayed in one show command.

```

R5#show l2vpn service xconnect name VPLS_102 detail | begin Interface
Interface          Group      Encapsulation          Prio  St  XC St
-----
VPWS name: VPLS_102, State: UP
Gi2                 Gi2:3551/102 (Eth VLAN) 0      UP  UP
                    Interworking: ethernet
pw59102             48.0.0.9:59102 (MPLS) 0      UP  UP
                    Local VC label 5006
                    Remote VC label 9023

```

Now we will verify the data-plane by capturing on CSR1 and CSR9. The first set of packets shows traffic entering CSR1 from the UPE with label 1017 (0x3F9) and exiting CSR1 towards the core with labels {94006 9024} (0x16F36, 0x2340). This does not include the FAT-PW label, and the CW isn't terribly important right now. The destination IP address in the VPN is 192.168.0.11. The color scheme is the same as before for consistency.

```

R1#show monitor capture CAP buffer dump
1
0000: 005056A9 1AAA0050 56A9BE8A 81000DB8 .PV....PV.....
0010: 8847003F 91FF0000 00000050 56A99C60 .G.?.....PV..`
0020: 005056A9 F9610800 450001F4 00440000 .PV..a..E....D..
0030: FF01385F C0A8000A C0A8000B 0800A25F ..8_....._
0040: 00120000 00000000 08899AA1 ABCDABCD .....

2
0000: 005056A9 862A0050 56A91AAA 81000DBA .PV...*.PV.....
0010: 884716F3 60FF0234 00FF8000 01010000 .G..`..4.....
0020: 03E80050 56A99C60 005056A9 F9610800 ...PV..`.PV..a..
0030: 450001F4 00440000 FF01385F C0A8000A E....D....8_....
0040: C0A8000B 0800A25F 00120000 00000000 ....._.....

```

```
0050: 08899AA1 ABCDABCD ABCDABCD ABCDABCD .....
```

We perform the same capture on CSR9, measuring inbound traffic from the core and outbound traffic towards the UPE, which is CSR5. Label 9024 arrives from the core, CSR9 bridges this to the access PW, and label 5006 (0x138E) is pushed. Nothing else changes except for the MPLS labels as the tunneled Ethernet frame, and higher layer protocols inside it, remain untouched.

```
R9#show monitor capture CAP buffer dump
```

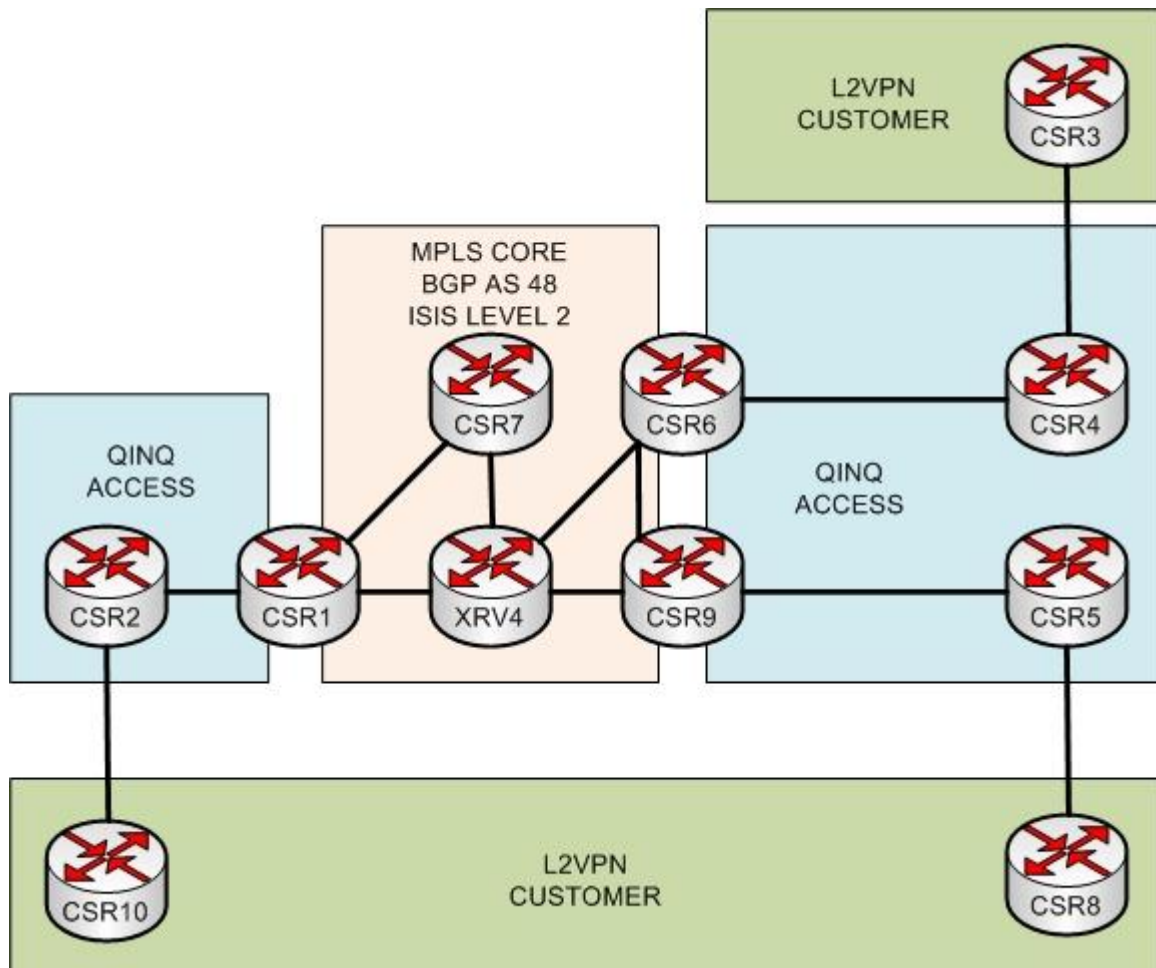
```
1
0000: 005056A9 D6720050 56A9862A 81000E0A .PV..r.PV..*....
0010: 88470234 00FE8000 01010000 04130050 .G.4.....P
0020: 56A99C60 005056A9 F9610800 450001F4 V..`.PV..a..E...
0030: 004E0000 FF013855 C0A8000A C0A8000B .N....8U.....
0040: 080053EC 00140000 00000000 088DE90E ..S.....
0050: ABCDABCD ABCDABCD ABCDABCD ABCDABCD .....

2
0000: 005056A9 DC630050 56A9D672 81000DE7 .PV..c.PV..r....
0010: 88470138 E1FF0000 00000050 56A99C60 .G.8.....PV..`
0020: 005056A9 F9610800 450001F4 004E0000 .PV..a..E....N..
0030: FF013855 C0A8000A C0A8000B 080053EC ..8U.....S.
0040: 00140000 00000000 088DE90E ABCDABCD .....
```

Additional Reading – Reference configurations "hvpls-mpls"

32.1.3.2 QinQ in the Access Network

The next test uses a smaller network and utilizes dot1q tunneling in the access networks. In real designs, large switched networks would make up the access networks by pushing another dot1q tag (metro or carrier tag) onto the C-VLANs the customer is trying to extend. As such, the devices in the access network do not have to run IP or MPLS at all; they just need to manipulate dot1q tags like switches. In this example, the UPEs are CSR2, CSR4, and CSR5. The NPEs are CSR1, CSR6, and CSR9. The P routers are CSR7 and XRv4. The customer routers are CSR10, CSR3, and CSR8. In order to create this port-based E-LAN service with total VLAN transparency, I created new vSwitches just to connect the CEs to the UPEs. This is because VLAN 101, for example, needs to be on different switches on both ends of the network, so within the scope of vSphere, this means different vSwitches. These new connections connect to Gig3 of the CE and UPE routers. The CEs are not using VRFs this time, and VLANs 101 and 102 are used for testing. Unlike the previous example, this time we offer an entirely transparent service so there is no reference to VLANs 101 or 102 anywhere in the SP network.



The CE configuration is not shown as it is just two dot1q interfaces in VLAN 101 and 102 on each router; no routing protocols exist in the customer network this time. The UPE devices accept all incoming traffic on Gig3 and push the carrier tag on top of the existing tag(s). They also pop it in the reverse direction (egress) so that the customer never sees the S- tag. The UPE bridges the incoming EFP where the tag was pushed to the outgoing EFP which performs no operation, just matches traffic with the proper carrier tag and any inner tag. The NPE (CSR1) pops the carrier tag on ingress (pushes it back on egress), which sustains the original VLAN ID in the tunneled frame. The NPE bridges the EFP and the VFI, consisting of PWs discovered and signaled using BGP for variety. This completes the H-VPLS architecture; the configuration on other NPEs and UPEs is nearly identical, except using different carrier tags and VE-IDs.

```
! CSR2
interface GigabitEthernet2
  service instance 3512 ethernet
  encapsulation dot1q 3512 second-dot1q 1-4094
  bridge-domain 3512

interface GigabitEthernet3
```

```

service instance 3512 ethernet
  encapsulation default
  rewrite ingress tag push dot1q 3512 symmetric
  bridge-domain 3512

! CSR1
interface GigabitEthernet2
  service instance 3512 ethernet
  encapsulation dot1q 3512 second-dot1q 1-4094
  rewrite ingress tag pop 1 symmetric

l2vpn vfi context VPLS
  vpn id 100
  autodiscovery bgp signaling bgp template TMP_NPE_PW
  ve id 1

bridge-domain 3512
  member GigabitEthernet2 service-instance 3512
  member vfi VPLS

```

We will trace the path from CSR10 to CSR3. First, we will record CSR3's MAC address so we know what to look for in the BD CAM tables. Packets arriving from CSR10 with this MAC address on Gig2's EFP 3512 are placed in the BD and sent upstream towards the NPE (CSR1). Packets arrive with tag X where X is any C-VLAN. The EFP pushes tag 3512 for a tag stack of {3512 X} transiting the access network. Imaginary witches in the transit path between the NPE and UPE would switch these frames according to the VLAN 3512 CAM, ignoring the inner tag.

```

R3#show interfaces gig3 | include bia
  Hardware is CSR vNIC, address is 0050.56a9.86c3 (bia 0050.56a9.86c3)

```

```

R2#show bridge-domain 3512
Bridge-domain 3512 (2 ports in all)
State: UP                               Mac learning: Enabled
Aging-Timer: 300 second(s)
  GigabitEthernet2 service instance 3512
  GigabitEthernet3 service instance 3512
AED MAC address    Policy Tag      Age  Pseudoport
0  0050.56A9.86C3 forward dynamic 298  GigabitEthernet2.EFP3512
0  0050.56A9.42EF forward dynamic 298  GigabitEthernet3.EFP3512
1  FFFF.FFFF.FFFF flood  static  0    OLIST_PTR:0xe8085800

```

Next, we will verify that CSR1 (NPE) received all the VPLS routes from BGP. We specifically would need to have the prefix from CSR6 at a minimum. Without this, the PWs cannot be built between NPEs which form the core of the VPLS instance.

```

R1#show bgp l2vpn vpls rd 48:100 | begin Network
      Network          Next Hop          Metric LocPrf Weight Path

```

```

Route Distinguisher: 48:100
*> 48:100:VEID-1:Blk-1/136
                                0.0.0.0                                32768 ?
*>i 48:100:VEID-6:Blk-1/136
                                48.0.0.6                                0      100      0 ?
*>i 48:100:VEID-9:Blk-1/136
                                48.0.0.9                                0      100      0 ?

```

Next, we can see which PWs were automatically created based on these BGP routes. Notice that the concept of split-horizon is moot for this design since there are no NPE-to-UPE PWs. The traffic arrives at the NPE on an EFP, not a PW, so there isn't any bridging of PWs requiring split-horizon to be disabled anymore. The entire access network is layer 2 only which alleviates the complexity of larger IP/MPLS and PW designs. We can also see the remote PW label is 6015, which was computed based on the VE-ID, VBO, and VBS (described in another section). Since we are using auto-route-target, the VPN-ID is used as the RT and is imported/exported by all nodes, creating a full-mesh of PWs fit for providing E-LAN service.

```

R1#show l2vpn vfi name VPLS
Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No
VFI name: VPLS, state: up, type: multipoint, signaling: BGP
VPN ID: 100, VE-ID: 1, VE-SIZE: 10
RD: 48:100, RT: 48:100
Bridge-Domain 3512 attachment circuits:
Pseudo-port interface: pseudowire100005
Interface      Peer Address    VE-ID  Local Label  Remote Label  S
pseudowire100007 48.0.0.9        9      1026         9013          Y
pseudowire100006 48.0.0.6        6      1023         6015          Y

```

We can check the VPLS instance to ensure both PWs are up. Specifically, we want to look at the PW with VE-ID 6 to see the label stack and verify the FEC.

```

R1#show l2vpn atom vc service-name VPLS

```

Interface	Peer ID	VC ID	Service Type	Name	Status
pw100006	6	100	vfi	VPLS	UP
pw100007	9	100	vfi	VPLS	UP

The route to CSR6 is via CSR7 due to IGP costs. We bind CSR7's LDP label to this prefix as a result, which is 7002. The full label stack is therefore {7002 6015}.

```

R1#show ip route 48.0.0.6
Routing entry for 48.0.0.6/32
  Known via "isis", distance 115, metric 28, type level-2
  Redistributing via isis 48
  Last update from 48.1.7.7 on GigabitEthernet2.517, 00:01:15 ago
  Routing Descriptor Blocks:

```



```
* 48.1.7.7, from 48.0.0.6, 00:01:15 ago, via GigabitEthernet2.517
Route metric is 28, traffic share count is 1
```

```
R1#show mpls ldp bindings 48.0.0.6 32 neighbor 48.0.0.7
lib entry: 48.0.0.6/32, rev 24
remote binding: lsr: 48.0.0.7:0, label: 7002
```

```
R1#show l2vpn atom vc ve-id 6 detail | include label_stack
Output interface: Gi2.517, imposed label stack {7002 6015}
```

We verify that CSR1 is actually receiving frames and learns the MAC address of CSR3 via the VPLS instance. The pseudo-port represents one of the PWs within that instance, and we assume this to be towards CSR6. This pseudo-port instructs the router to add MPLS encapsulation to the frame using the labels shown above. Frames received on the EFP have their top tag removed on ingress and only the inner VLAN is preserved end-to-end. The arriving tag stack is {3512 X} but only tag X is encapsulated inside MPLS inside the tunneled Ethernet frame header.

```
R1#show bridge-domain 3512
Bridge-domain 3512 (3 ports in all)
State: UP                               Mac learning: Enabled
Aging-Timer: 300 second(s)
GigabitEthernet2 service instance 3512
vfi VPLS neighbor 48.0.0.6 100
vfi VPLS neighbor 48.0.0.9 100
AED MAC address      Policy Tag      Age Pseudoport
0 0050.56A9.86C3 forward dynamic 205 VPLS.1004021
0 0050.56A9.42EF forward dynamic 206 GigabitEthernet2.EFP3512
1 FFFF.FFFF.FFFF flood static 0 OLIST_PTR:0xe8057850
```

As P routers, CSR7 and XRv4 perform swap and pop operations, respectively. This delivers the traffic to CSR6 with the PW label exposed.

```
R7#show mpls forwarding-table labels 7002
Local      Outgoing    Prefix      Bytes Label  Outgoing    Next Hop
Label      Label        or Tunnel Id  Switched     interface
7002      94008        48.0.0.6/32   990          Gi2.574     48.7.14.14
```

```
RP/0/0/CPU0:XRv4#show mpls forwarding labels 94008
Local  Outgoing    Prefix      Outgoing    Next Hop      Bytes
Label  Label        or ID       Interface    Interface     Switched
-----
94008  Pop          48.0.0.6/32  Gi0/0/0/0.564 48.6.14.6    15840
```

CSR6 cannot label switch this packet since the LFIB removes all labels. The packet is piped to the BD CAM so a switching decision can be made based on the destination MAC, which is highlighted. The BD CAM table shows this MAC address is accessible via an EFP that sends traffic into another access

network on the other side of the core. A new carrier tag is used simply because the virtualization environment limits, but it could have been the same if the access networks didn't share the same vSwitch. Traffic inside the MPLS packet has tag X and the NPE pushes the carrier tag 3546 before sending it into the access network. The tag stack becomes {3546 X}.

```
R6#show mpls forwarding-table labels 6015
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Label	Outgoing interface	Next Hop
6015	No Label	lbl-blk-id(2:0)	0		none	point2point

```
R6#show bridge-domain 3546
```

```
Bridge-domain 3546 (3 ports in all)
```

```
State: UP Mac learning: Enabled
```

```
Aging-Timer: 300 second(s)
```

```
GigabitEthernet2 service instance 3546
```

```
vfi VPLS neighbor 48.0.0.1 100
```

```
vfi VPLS neighbor 48.0.0.9 100
```

AED	MAC address	Policy	Tag	Age	Pseudoport
1	FFFF.FFFF.FFFF	flood	static	0	OLIST_PTR:0xe80d4840
0	0050.56A9.86C3	forward	dynamic	120	GigabitEthernet2.EFP3546
0	0050.56A9.42EF	forward	dynamic	120	VPLS.100402a

When the frame reaches CSR4, it is bridged from the ingress EFP to the egress EFP via the BD CAM table. Notice that the pseudo-port is an EFP from Gig3, which is our UPE-CE link. Traffic arrives with tag {3546 X} and is stripped on egress to simply carry tag X.

```
R4#show bridge-domain 3546
```

```
Bridge-domain 3546 (2 ports in all)
```

```
State: UP Mac learning: Enabled
```

```
Aging-Timer: 300 second(s)
```

```
GigabitEthernet2 service instance 3546
```

```
GigabitEthernet3 service instance 3546
```

AED	MAC address	Policy	Tag	Age	Pseudoport
1	FFFF.FFFF.FFFF	flood	static	0	OLIST_PTR:0xe80d4c20
0	0050.56A9.86C3	forward	dynamic	225	GigabitEthernet3.EFP3546
0	0050.56A9.42EF	forward	dynamic	43	GigabitEthernet2.EFP3546

Because we cannot use EPC on the EFPs, we will verify packets being MPLS-encapsulated on CSR1 towards CSR7, and likewise being MPLS-decapsulated on CSR6 from CSR7. We expect to see the tunneled VLAN ID inside the customer traffic, along with the expected label values revealed earlier. The first packet is sent from CSR1 to CSR7, and the second is a packet arriving on CSR6. BGP signaling does not appear to allow us to use FAT-PW or CW, so those components are missing from the capture. This restriction is specified in an extended community that IOS does not let us configure (L2VPN INFO community, which has the CW, sequencing, and FAT-PW bits set to zero unconditionally). Using the same colors as the previous section (labels are yellow, destination MAC is cyan, destination IP is pink), we can verify the packets are following the proper LSP. Packets are sent from CSR1 with labels {7002

6015} (0x1B5A, 0x177F) towards CSR7. CSR6 receives the packet only with label 6015 as expected. The difference with these captures, aside from missing a CW and FAT-PW label, is the presence of the dot1q C-VLAN tag (grey). In this case, it is VLAN 101 (0x65) but since the core is not matching this number anywhere, any VLANs could be used. This is also possible with MPLS in the access network but was not demonstrated last time.

```
R1#show monitor capture CAP buffer dump
```

```
2
```

```
0000: 005056A9 EA770050 56A91AAA 81000DBD .PV..w.PV.....
0010: 884701B5 A0FF0177 F1FF0050 56A986C3 .G.....w...PV...
0020: 005056A9 42EF8100 00650800 450001F4 .PV.B....e..E...
0030: 007B0000 FF016E2F C0A8650A C0A86503 .{....n/..e...e.
0040: 08004633 001D0000 00000000 092BF620 ..F3.....+.
0050: ABCDABCD ABCDABCD ABCDABCD ABCDABCD .....
```

```
R6#show monitor capture CAP buffer dump
```

```
1
```

```
0000: 005056A9 DE0D0050 56A9862A 81000DEC .PV....PV...*....
0010: 88470177 F1FD0050 56A986C3 005056A9 .G.w...PV....PV.
0020: 42EF8100 00650800 450001F4 007C0000 B....e..E....|..
0030: FF016E2E C0A8650A C0A86503 08004625 ..n...e...e...F%
0040: 001D0001 00000000 092BF62D ABCDABCD .....+.-....
```

Just to prove the transparency is functional, we quickly test another C-VLAN. The forwarding path, carrier tags, and LSPs do not change at all. The only difference is the dot1q C-VLAN tag, shown again in grey. This time, the customer is sending traffic on VLAN 102 (0x66). The other fields are highlighted as well for completeness; they are identical to the packets above.

```
R1#show monitor capture CAP buffer dump
```

```
1
```

```
0000: 005056A9 EA770050 56A91AAA 81000DBD .PV..w.PV.....
0010: 884701B5 A0FF0177 F1FF0050 56A986C3 .G.....w...PV...
0020: 005056A9 42EF8100 00660800 450001F4 .PV.B....f..E...
0030: 00850000 FF016C25 C0A8660A C0A86603 .....l%..f...f.
0040: 08007766 001F0000 00000000 0932C4E4 ..wf.....2..
0050: ABCDABCD ABCDABCD ABCDABCD ABCDABCD .....
```

```
R6#show monitor capture CAP buffer dump
```

```
0
```

```
0000: 005056A9 DE0D0050 56A9862A 81000DEC .PV....PV...*....
0010: 88470177 F1FD0050 56A986C3 005056A9 .G.w...PV....PV.
0020: 42EF8100 00660800 450001F4 00850000 B....f..E.....
0030: FF016C25 C0A8660A C0A86603 08007766 ..l%..f...f...wf
0040: 001F0000 00000000 09
32C4E4 ABCDABCD .....2.....
```

Additional Reading – Reference configurations "hvpls-qinq"

32.2 IP encapsulated L2VPN

Layer 2 tunneling protocol (L2TP) is another way to offer E-LINE, E-LAN, and E-TREE services when MPLS transport is not available. Like GRE, it is a form of IP encapsulation, but can carry many layer 2 protocols. L2TP, when used with xconnect, uses IP protocol 115 and is referenced in RFC 3931. L2TP can also use UDP encapsulation as seen with VPDN groups, which is beyond the scope of this section. The signaling for L2TP is quite complex and offers many verbose debugs. The network diagram is the same as the MPLS L2VPN tests, except MPLS is disabled everywhere, along with all of its supporting protocols (LDP, BGP, etc). L2TP can only be used in a P2P fashion as all VFI configurations reject all encapsulations other than MPLS. As an alternative to VPLS, we can use Overlay Transport Virtualization (OTV), a Cisco-proprietary protocol (for now) designed for data center interconnects (DCI) to built transparent LANs. Like DMVPN and LISP, it is more customer-oriented, but it is still technically a mechanism of providing E-LAN service, so I consider it in scope for the CCIE SP. Besides, a good service provider is aware of how customer-focused technologies work.

32.2.1 E-LINE with L2TP

Layer 2 tunnel protocol (L2TP) works like a P2P MPLS L2VPN in that it provides transparent layer 2 service across a service provider core. The difference is that it is IP-encapsulated, not requiring MPLS at all. L2TP is a very signaling-intensive protocol with several new messages described below (RFC 3931). Fortunately, the general logic follows a three-way handshake, much like TCP.

Start-Control-Connection-Request (SCCRQ): Initiates a control connection between two L2TP peers. Either router can initiate the session. It carries the L2TP router-ID, hostname, and other details. This is loosely analogous to a TCP SYN.

Start-Control-Connection-Request (SCCRP): Sent in response to a SCCRQ. This is an acknowledgement that the SCCRQ was accepted and setup should continue. This like a TCP SYN ACK.

Start-Control-Connection-Connected (SCCCN): Sent in response to a SCCRP to complete the control session setup. This is like the final TCP ACK. These three messages serve the same function as the TCP three-way handshake.

Stop-Control-Connection-Notification (StopCCN): Sent by either L2TP router that the control connection is being shut down. A result code is carried in the message to explain why. This is like a TCP RST.

Hello packet (HELLO): Control message sent by either L2TP router to serve as a keepalive.

Incoming-Call-Request (ICRQ): Like a SCCRQ, except happens after the session is established. It happens per call, not per session, and is mostly relevant in a dial-up deployment (BBA setup with LAC/LNS). This is also like a TCP SYN.

Incoming-Call-Request (ICRP): Like a SCCRP, except is sent in response to an ICRQ for a particular call. This is sent if the ICRQ parameters are accepted. This is also like a TCP SYN ACK.

Incoming-Call-Connected (ICCN): Like a SCCRCN, except is sent in response to an ICRP for a particular call. This finalizes the L2TP session for the call. This is also like a TCP ACK.

Outgoing-Call-Request (OCRQ): Like an ICRQ, except in the opposite direction. This is specific to calls from a LAC, which is not present in this test. This is also like a TCP SYN.

Outgoing-Call-Request (OCRP): Like an ICRP, except in the opposite direction. This is specific to calls from a LAC, which is not present in this test. This is also like a TCP SYN ACK.

Outgoing-Call-Connected (OCCN): Like an ICCN, except in the opposite direction. This is specific to calls from a LAC, which is not present in this test. This is also like a TCP ACK.

Call-Disconnect-Notify (CDN): Like a StopCCN, except specific for a call within the general L2TP control session. It disconnects the specific session but does not affect L2TP control.

WAN-Error-Notify (WEN): Sent from a LAC to the LNS to indicate an error condition in the WAN. Not present in this test.

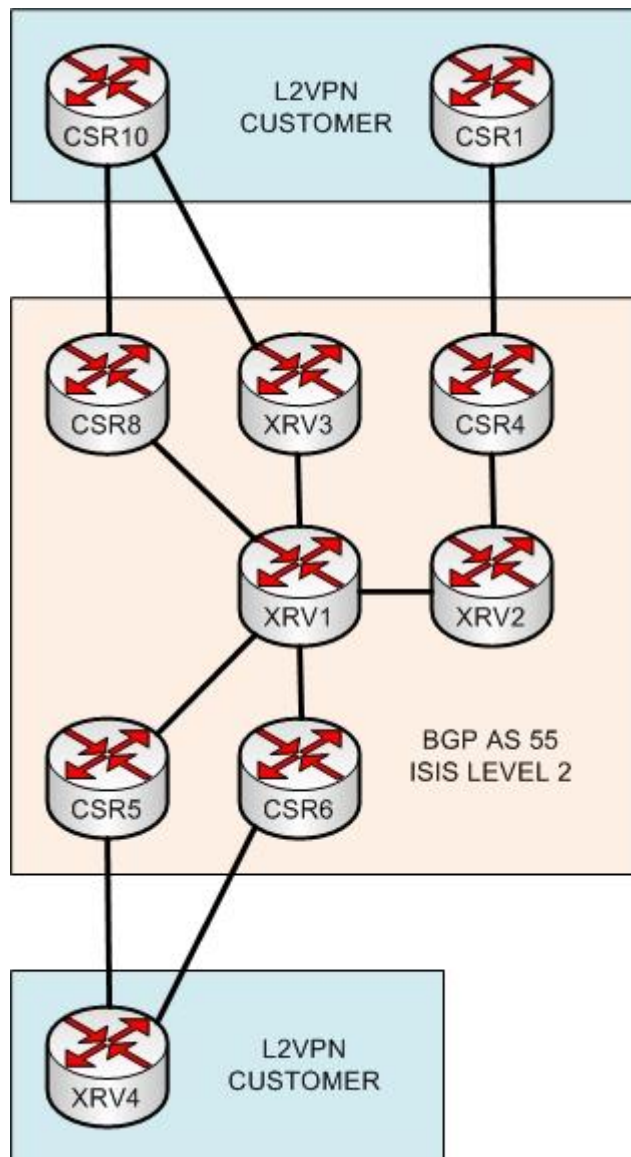
Set-Link-Info (SLI): This is used to send updated status reports across between L2TP speakers.

Explicit-Acknowledgement (EXP-ACK): Just acknowledges a message and is typically used with sequencing.

Zero-Length-Body (ZLB): A message received without any attribute-value pairs (AVPs). Serves the same function as the EXP-ACK. The RFC says it can only be used when control channel authentication is disabled, but I see it used all the time, with EXP-ACK never being used.

The normal P2P PW configuration is almost identical to the MPLS AToM section. The most significant change is using L2TPv3 encapsulation versus MPLS, along with some other parameters. The configurations are shown below. We will mirror the setup from earlier where CSR8 has two PWs in primary/backup operation to CSR5 and CSR6 respectively. There is no option for control-word, yet sequencing is available. We can also adjust the tunnel TOS, much like a GRE tunnel, either by reflecting the TOS inside the payload or hard-setting a value (CS1 in this case on the backup tunnel). We will explore these features later. We use the new L2VPN configuration on CSR8 and CSR5, and use the legacy xconnect configuration on CSR6. **Pitfall: L2TP is not supported with EFP/EVC in the CSR1000v. This means we must use a main interface or subinterface, but not a service instance.** Those interfaces don't need any configuration on them other than the encapsulation when using the new L2VPN configuration model (CSR5 and CSR8). When using the legacy model (CSR6), the xconnect goes under the subinterface. We will examine these configurations soon. The network is very similar to before by with some routers

removed since L2TP cannot support E-LAN or E-TREE services. There is no need to have a large network when only P2P connections are being tested.



```

! CSR8
template type pseudowire TMP_L2TP_PW
  encapsulation l2tpv3
  signaling protocol l2tpv3
  ip local interface Loopback0

interface pseudowire10141
  description PW TO CSR5
  source template type pseudowire TMP_L2TP_PW
  neighbor 55.0.0.5 10141
  
```

```

interface pseudowire10142
  description PW TO CSR6
  source template type pseudowire TMP_L2TP_PW
  neighbor 55.0.0.6 10142

interface GigabitEthernet2.3504
  encapsulation dot1Q 3580 second-dot1q 3504

l2vpn xconnect context XC_XRV14_R10
  member GigabitEthernet2.3504
  member pseudowire10141 group R10_XRV14 priority 3
  member pseudowire10142 group R10_XRV14 priority 6

! CSR5
interface pseudowire10141
  description PW TO CSR8
  encapsulation l2tpv3
  neighbor 55.0.0.8 10141
  signaling protocol l2tpv3
  ip local interface Loopback0

interface GigabitEthernet2.1014
  encapsulation dot1Q 3554 second-dot1q 1014

l2vpn xconnect context XC_XRV14_R10
  member GigabitEthernet2.1014
  member pseudowire10141

! CSR6
pseudowire-class PW_1014
  encapsulation l2tpv3
  ip local interface Loopback0

interface GigabitEthernet2.3546
  encapsulation dot1Q 3546
  xconnect 55.0.0.8 10142 encapsulation l2tpv3 pw-class PW_1014

```

We can verify the L2TP sessions are operational using some basic commands. We can see if the tunnel is up, and to which peer. We can also check the transport information, which shows the IP protocol number and local interface as well. The “tunnel” represents the control channel and all higher level protocols (PPP, xconnect, etc) all rely on this. Between a set of L2TP routers, there is a single tunnel, but multiple “calls” or “sessions”. This is why the SCC and IC messages are different.

```

R8#show l2tp tunnel
L2TP Tunnel Information Total tunnels 2 sessions 2
LocTunID  RemTunID  Remote Name  State  Remote Address  Sessn L2TP Class/
Count VPDN Group
1501687873 1414773374 R6          est    55.0.0.6        1      l2tp_default_cl
2718846397 3521913226 R5          est    55.0.0.5        1      l2tp_default_cl

```

```
R8#show l2tp tunnel transport
L2TP Tunnel Information Total tunnels 2 sessions 2
LocTunID   Type Prot  Local Address  Port  Remote Address  Port
1501687873 IP    115  55.0.0.8      0     55.0.0.6        0
2718846397 IP    115  55.0.0.8      0     55.0.0.5        0
```

We can also check the packets transported by L2TP. The difference between session and tunnel packets is subtle. “Session” packets only counts packets that occur within the given session (or “call”), so clearing the L2TP session or having the PW fail for any reason will reset these counters. “Tunnel” packets continue to increment even after PWs fail, provided the tunnel ID is not deallocated and the control-session doesn’t completely fail. Therefore, the tunnel packets will always be greater than or equal to the session packets since one is a subset of the other. Both commands reference tunnels by their tunnel-ID, which is a 32-bit unsigned number used to represent a connection. This used to be a 16-bit unsigned number in L2TPv2, but was expanded in to 32-bits in L2TPv3. Above, we see that the ID beginning with 150 corresponds to the backup connection to CSR6, while the one beginning with 271 corresponds to the primary connection to CSR5. That is why only one tunnel has packets associated.

```
R8#show l2tp session packets
L2TP Session Information Total tunnels 2 sessions 2
LocID      RemID      TunID      Pkts-In    Pkts-Out   Bytes-In   Bytes-Out
2927100785 1218283557 1501687873 0           0           0           0
3331724010 4005951606 2718846397 105         144         18990       22416
```

```
R8#show l2tp tunnel packets
L2TP Tunnel Information Total tunnels 2 sessions 2
LocTunID   Pkts-In    Pkts-Out   Bytes-In   Bytes-Out
1501687873 0           0           0           0
2718846397 126         186         32068      37198
```

After clearing the session and sending 100 packets through the tunnel, the session counts those, but the tunnel continues to increment from the value it had before the session cleared. The control-channel was reset but remained UP (in software, at least) but the “call” was re-established. This is a minor detail and is not terribly significant, but helps highlight the difference between a “tunnel” and a “session”.

```
R8#show l2tp session packets
L2TP Session Information Total tunnels 2 sessions 2
LocID      RemID      TunID      Pkts-In    Pkts-Out   Bytes-In   Bytes-Out
490257398  1184553043 1501687873 0           0           0           0
3918464858 50829490   2718846397 100         101         11800       12274
```

```
R8#show l2tp tunnel packets
L2TP Tunnel Information Total tunnels 2 sessions 2
LocTunID   Pkts-In    Pkts-Out   Bytes-In   Bytes-Out
1501687873 0           0           0           0
2718846397 226         288         43868      49546
```


Next, we verify the signaling packet flow. We will clear the backup tunnel then watch CSR6 renegotiate with CSR8. We see that CSR6 sends the initial SCCRP; we can confirm this by checking the tunnel details to ensure the tunnel is “locally” initiated. CSR8 responds with the SCCRP, and CSR6 confirms the session with the SCCN. After the control channel is established, CSR6 originates the call with an ICRQ message on behalf of the xconnect process. Only the “RQ” packets are highlighted during this exchange for brevity, but the debug clearly shows all of the message transactions. Once the session is established, the two routers then exchange status messages using the SLI message.

```
R6#debug l2tp packet brief
```

```
L2TP tnl 08213:4ED2CB86: Tx -> SCCRP loc 4ED2CB86 rem 00000000
L2TP tnl 08213:4ED2CB86: Rx <- SCCRP loc 4ED2CB86 rem 06244823
L2TP tnl 08213:4ED2CB86: Tx -> SCCCN loc 4ED2CB86 rem 06244823
L2TP ____:08213:29450B58: Tx -> ICRQ loc 29450B58 rem 00000000 ser 2D92DED3
L2TP ____:08213:29450B58: Rx <- ICRP loc 29450B58 rem 5700E5FC ser 2D92DED3
L2TP ____:08213:29450B58: Tx -> ZLB A loc 29450B58 rem 5700E5FC ser 2D92DED3
L2TP ____:08213:29450B58: Tx -> ICCN loc 29450B58 rem 5700E5FC ser 2D92DED3
L2TP ____:08213:29450B58: Tx -> SLI loc 29450B58 rem 5700E5FC ser 2D92DED3
L2TP ____:08213:29450B58: Tx -> ZLB A loc 29450B58 rem 5700E5FC ser 2D92DED3
L2TP ____:08213:29450B58: Rx <- SLI loc 29450B58 rem 5700E5FC ser 2D92DED3
```

```
R6#show l2tp tunnel state
```

```
L2TP Tunnel Information Total tunnels 1 sessions 1
```

LocTunID	RemTunID	Local Name	Remote Name	State	Last-Chg
1322437510	103041059	R6	R8	est	00:16:09

```
R6#show l2tp tunnel all id 1322437510 | include initiate
```

```
Locally initiated tunnel
```

We can also check the packet counters for control messages. We can see counters for the previously defined L2TP control messages.

```
R6#show l2tp counters tunnel
```

```
Global L2TP tunnel control message statistics:
```

	XMIT	RE-XMIT	RCVD	DROP
	=====	=====	=====	=====
ZLB	262	0	1174	1
SCCRQ	111	8	88	0
SCCRP	56	0	53	0
SCCCN	4	0	4	0
StopCCN	90	87	110	1
Hello	1124	0	88	16
OCRQ	0	0	0	0
OCRP	0	0	0	0
OCCN	0	0	0	0
ICRQ	19	0	11	0
ICRP	11	0	4	0

ICCN	4	0	11	0
CDN	0	0	25	0
WEN	0	0	0	0
SLI	15	0	38	0
EXP ACK	0	0	0	0
SRRQ	0	0	0	0
SRRP	0	0	0	0
CiscoACK	0	0	27	27
Total	1696	95	1633	45

Now, we will validate the detailed L2TP information. Using the tunnel ID beginning with 271, we can look deeper into this PW. First, we see the tunnel was “remotely” initiated in green, which makes sense since CSR6 claims to have initiated it. In a peer-to-peer setup like this, it doesn’t matter who really initiates the tunnel or not. Next, basic information about the tunnel endpoints, including IP addresses and hostnames, is displayed. The counters appear incorrect since, having cleared the tunnel, the top set of counters should be less than the bottom set, much like the “session” and “tunnel” packets seen earlier. Both of these fields represent the “tunnel” counters (yellow). The Ns, Nr, RWS, and congestion-control features are examined later (cyan). The next several lines deal with out of order packet drops, and the last line tells us we are not running authentication (pink). This is specific to the tunnel, not the session, which we look at later.

```
R8#show l2tp tunnel all id 2718846397
L2TP Tunnel Information Total tunnels 2 sessions 2
Tunnel id 2718846397 is up, remote id is 3521913226, 1 active sessions
Remotely initiated tunnel
Tunnel state is established, time since change 00:07:43
Tunnel transport is IP (115)
Remote tunnel name is R5
Internet Address 55.0.0.5, port 0
Local tunnel name is R8
Internet Address 55.0.0.8, port 0
L2TP class for tunnel is l2tp_default_class
Counters, taking last clear into account:
305 packets sent, 226 received
50844 bytes sent, 43868 received
Last clearing of counters never
Counters, ignoring last clear:
305 packets sent, 226 received
50844 bytes sent, 43868 received
Control Ns 61, Nr 54
Local RWS 1024 (default), Remote RWS 1024
Control channel Congestion Control is disabled
Tunnel PMTU checking disabled
Retransmission time 1, max 1 seconds
Unsent queuesize 0, max 0
Resend queuesize 0, max 3
Total resends 0, ZLB ACKs sent 43
```

```
Total out-of-order dropped pkts 0
Total out-of-order reorder pkts 0
Total peer authentication failures 0
Current no session pak queue check 0 of 5
Retransmit time distribution: 0 0 0 0 0 0 0 0 0
Control message authentication is disabled
```

The simplest way to begin adding complexity is to define an L2TP-class. This is a custom construct that is used to control advanced L2TP signaling parameters. This is not the same as a PW-template or PW-class as those contain general PW settings, such as the encapsulation type and neighbor address. The L2TP-class modifies the L2TP control-plane behavior. There are two ways to secure L2TP: L2TP control-channel authentication and L2TPv3 control message hashing. The first one is older and was carried over from L2TPv2 for interoperability with older versions. It behaves like the Challenge Handshake Authentication Protocol (CHAP) commonly used in PPP. The second is newer and uses the entire message in the hash, which is more secure. You can configure both at the same time as well. On CSR8, we will begin by enabling CHAP-style authentication in an L2TP-class and applying it to the backup PW towards CSR6. Note: In order to apply the L2TP-class, you must remove the signaling protocol and re-add it, which tears down the PW.

```
! CSR8
l2tp-class L2TP_CLASS_BACKUP
  authentication
  password L2TP_PASS

interface pseudowire10142
  no signaling protocol l2tpv3
  signaling protocol l2tpv3 L2TP_CLASS_BACKUP
```

We can verify the class configuration as well. This shows all the defaults within the specific class, too.

```
R8#show l2tp class | section BACKUP
class [L2TP_CLASS_BACKUP]
  is a statically configured class
  configuration:
    l2tp-class L2TP_CLASS_BACKUP
      authentication
      digest check
      hello 60
      password L2TP_PASS
      receive-window 1024
      retransmit retries 15
      retransmit timeout max 8
      retransmit timeout min 1
      retransmit initial retries 2
      retransmit initial timeout max 8
      retransmit initial timeout min 1
      timeout setup 300
```

As expected, our tunnel to CSR6 is down due to authentication issues as CSR6 has not been configured for authentication yet. The L2TP debugs are almost unreadable because they are so chatty. Sorting through them, we see a hint at the problem. The challenge was failed so the session cannot establish.

```
R8#show l2tp tunnel
```

```
%No active L2TP tunnels
```

```
R8#debug l2tp event
```

```
L2TP tnl 081F1:2B5DAEAF: Shutting down tunnel
```

```
L2TP tnl 081F1:2B5DAEAF: With 1 app
```

```
L2TP tnl 081F1:2B5DAEAF: Result Code
```

```
L2TP tnl 081F1:2B5DAEAF: Requester is not authorized to establish a  
control channel
```

```
L2TP tnl 081F1:2B5DAEAF: Error Code
```

```
L2TP tnl 081F1:2B5DAEAF: No error
```

```
L2TP tnl 081F1:2B5DAEAF: Vendor Error
```

```
L2TP tnl 081F1:2B5DAEAF: None
```

```
L2TP tnl 081F1:2B5DAEAF: Optional Message
```

```
L2TP tnl 081F1:2B5DAEAF: "process challenge failed "
```

On CSR6, we will correct the issue by creating an L2TP-class and applying it within the PW-class. We have to completely remove the xconnect from the subinterface before changing the signaling mechanism, but this is a simple step and is not shown. We intentionally configure the wrong password to see if the debug message reveals the mismatch.

```
! CSR6
```

```
l2tp-class L2TP_ADVANCED
```

```
authentication
```

```
password L2TP_PASS_WRONG
```

```
pseudowire-class PW_1014
```

```
protocol l2tpv3 L2TP_ADVANCED
```

As expected, debugging on CSR6 reveals that the tunnel still cannot come up. This time, the message is different and says the authentication is mismatched, versus having the “challenge” process totally fail.

```
R6#debug l2tp event
```

```
L2TP tnl 08138:F7AA6808:
```

```
L2TP tnl 08138:F7AA6808: Shutting down tunnel
```

```
L2TP tnl 08138:F7AA6808: Result Code
```

```
L2TP tnl 08138:F7AA6808: General error - refer to error code
```

```
L2TP tnl 08138:F7AA6808: Error Code
```

```
L2TP tnl 08138:F7AA6808: Vendor specific
```

```
L2TP tnl 08138:F7AA6808: Vendor Error
```

```
L2TP tnl 08138:F7AA6808: Authentication failed
```

```
L2TP tnl 08138:F7AA6808: Optional Message
```

```
L2TP tnl 08138:F7AA6808: "Tunnel auth failed for mismatch"
```

We quickly correct the password on CSR6 (not shown) and the PW immediately comes up. Unlike CHAP, we actually didn't create local usernames or a AAA server to authenticate the peer; L2TP will compare the hash from the shared secret and use that by itself, which is part of the reason why this is the "old" method. The hostnames could have been specified in the L2TP-class but, like PPP, will use the router hostname by default.

```
R6#debug l2tp event
L2TP tnl 0818E:09F70EBD: FSM-CC ev Sock-Ready
L2TP tnl 0818E:09F70EBD: FSM-CC Wt-Sock->Wt-SCCRP
L2TP tnl 0818E:09F70EBD: FSM-CC do Tx-SCCRQ
L2TP tnl 0818E:09F70EBD: FSM-CC ev Rx-SCCRP
L2TP tnl 0818E:09F70EBD: FSM-CC Wt-SCCRP->Proc-SCCRP
L2TP tnl 0818E:09F70EBD: FSM-CC do Rx-SCCRP
L2TP tnl 0818E:09F70EBD: Tunnel Authentication success
```

When we verify the details for this new tunnel, we see some interesting output/ CSR8 reports that the peer never had an authentication failure, and that authentication is not configured. Both of these claims are false; I assume that this status output only changes when the newer control-message hashing feature is enabled.

```
R8#show l2tp tunnel state | include R6
2400039996 2929395307 R8 R6 est 00:27:09
```

```
R8#show l2tp tunnel all id 2400039996 | include authen
Total peer authentication failures 0
Control message authentication is disabled
```

Next, we configure the "new" method of authentication on CSR8 and CSR5. The word "digest" houses many features, but when the "secret" option is used, this enables authentication and integrity checking. Again, I configure the password incorrectly on purpose on CSR5.

```
! CSR8
l2tp-class L2TP_CLASS_PRIMARY
digest secret 0 L2TP_DIGEST hash SHA1

interface pseudowire10141
signaling protocol l2tpv3 L2TP_CLASS_PRIMARY

! CSR5
interface pseudowire10141
signaling protocol l2tpv3 L2TP_ADVANCED

l2tp-class L2TP_ADVANCED
digest secret 0 L2TP_DIGEST_WRONG hash SHA1
```

As expected, the tunnel does not form due to the authentication mismatch. Like the control-channel authentication, the debug states this clearly.

```
! CSR5
L2TP tnl 10282:5CF2880D: Shutting down tunnel
L2TP tnl 10282:5CF2880D: Result Code
L2TP tnl 10282:5CF2880D: General error - refer to error code
L2TP tnl 10282:5CF2880D: Error Code
L2TP tnl 10282:5CF2880D: No error
L2TP tnl 10282:5CF2880D: Vendor Error
L2TP tnl 10282:5CF2880D: None
L2TP tnl 10282:5CF2880D: Optional Message
L2TP tnl 10282:5CF2880D: "SCCRQ authen failed"
L2TP tnl 10282:5CF2880D: ERROR: Validate message digest, received digest
mismatch with local
```

Once we correct the password on CSR5, we see the tunnel form immediately. Each control message is displayed in hex and should be followed by a message stating the digest was correct.

```
! CSR5
L2TP tnl 08325:116B1827: Control Message
C8 03 00 55 11 6B 18 27 00 03 00 05 80 08 00 00
00 00 00 10 80 1B 00 09 00 0C 01 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 80
0A 00 09 00 03 A1 8E 57 BE 80 0A 00 09 00 04 77
9B 9C 15 00 08 00 09 00 08 00 01 00 0A 00 09 00
72 00 00 00 00
L2TP tnl 08325:116B1827: Message digest match performed, passed.
```

Now, we can see CSR8 reporting authentication being enabled. The output here only updates when relating to the newer digest authentication and not the CHAP-style mechanism as we theorized earlier. You can also configure multiple digest secrets to perform password rollover.

```
R8#show l2tp tunnel state | include R5
4135688314 292231207 R8 R5 est 00:02:31

R8#show l2tp tunnel all id 4135688314 | include authen
Total peer authentication failures 0
Control message authentication is enabled with 1 digest secrets
Last control message authenticated with first digest secret
```

On CSR5, we will add a wrong password back in, but we will remove the right password so the wrong password is the “first” one. Now there are two passwords available. The session stays up because one is correct, but CSR5’s show command output differs from CSR8’s. CSR5 says there are two possible digests and the last message was authenticated using the second. Ideally, removing the “first” one, following a

password rollover, is a time-sensitive task. Cisco supports a maximum of 2 secrets for both IOS and XR, and the expectation is that L2TP would only have 2 passwords during the manual password rollover period.

```
! CSR5
l2tp-class L2TP_ADVANCED
  digest secret 0 L2TP_DIGEST_WRONG hash SHA1
  digest secret 0 L2TP_DIGEST hash SHA1

R5#show l2tp tunnel state | include R8
292231207 4135688314 R5 R8 est 00:05:02

R5#show l2tp tunnel all id 292231207 | include authen
Total peer authentication failures 0
Control message authentication is enabled with 2 digest secrets
Last control message authenticated with second digest secret
```

One last feature of the digest is disabling the digest check. It is enabled by default, and can only be disabled if digest authentication is disabled. Assuming CSR6 is an old router with poor performance, we can disable this feature to save some CPU cycles at the cost of security and message integrity. Other than verifying the L2TP-class, I have not found a way to verify that this is operational.

```
! CSR6
l2tp-class L2TP_ADVANCED
  no digest check

R6#show l2tp class | section ADVANCED
class [L2TP_ADVANCED]
  is a statically configured class
  configuration:
    l2tp-class L2TP_ADVANCED
      authentication
      no digest check
      hello 60
      [snip]
```

The HELLO mechanism is very simple and is similar to other protocols. The default hello timer is 60 seconds and we can see the packets exchanged using L2TP packet debugging. The difference is that L2TP receives a hello and replies with an acknowledgement (ZLB in our case). Peers must not expect hello messages according to a time interval, so there isn't a "dead time" concept. I think the parser just prints the ZLB before the hello, but the two happen at the same time. Notice the "loc" and "rem" fields represent the tunnel IDs in hex. We adjust the timer to 20 seconds on CSR6 only; the timers do not have to match. This will flap the session. This debug does not show received ZLB's since no processing is done on them.

```
! CSR6
```

```
l2tp-class L2TP_ADVANCED
hello 20
```

```
! CSR6
```

```
L2TP tnl 08217:50829095: Tx -> Hello loc 50829095 rem A50BA3D0
L2TP tnl 08217:50829095: Tx -> Hello loc 50829095 rem A50BA3D0
```

```
! CSR8
```

```
L2TP tnl 08542:A50BA3D0: Tx -> ZLB A loc A50BA3D0 rem 50829095
L2TP tnl 08542:A50BA3D0: Rx <- Hello loc A50BA3D0 rem 50829095
L2TP tnl 08542:A50BA3D0: Tx -> ZLB A loc A50BA3D0 rem 50829095
L2TP tnl 08542:A50BA3D0: Rx <- Hello loc A50BA3D0 rem 50829095
```

Next, we will examine L2TP sequencing. With AToM, this information was carried inside the control-word, which is a 4-byte field appended to the end of the MPLS label stack. L2TP handles it differently with a pair of numbers. The Ns and Nr fields represent sequence numbers: Ns is the number of the local message being sent and Nr is the expected number of the next packet being received.

```
! CSR5 and CSR8
```

```
interface pseudowire10141
sequencing both
```

We can verify the configuration was successful by checking the details of the session. The output is verbose but we only care about the sequencing part for now. To specify a session, we must provide the tunnel ID and the session ID, in that order. Alternatively, we can use the PW ID of 10141 which is simpler. We see that sequencing is enabled on CSR5 and CSR8. CSR5 is expecting to receive sequence number 16 (Nr), while CSR8 is using that number for the next packet to send (Ns). This one-way traffic is likely due to the RIP updates being sent by CSR10, since XRv4 is only passively listening.

```
R5#show l2tp session
```

```
L2TP Session Information Total tunnels 1 sessions 1
```

LocID	RemID	TunID	Username, Intf/ Vcid, Circuit	State	Last Chg	Uniq ID
1292798504	3934595965	292231207	10141, Gi2.1014:3554	est	00:06:31	0

```
R5#show l2tp session all tunnel id 292231207 1292798504 | section Sequenc
```

```
Sequencing is on
```

```
Ns 0, Nr 16, 0 out of order packets received
```

```
Packets switched/dropped by secondary path: Tx 0, Rx 0
```

```
R8#show l2tp session
```

```
L2TP Session Information Total tunnels 2 sessions 2
```

LocID	RemID	TunID	Username, Intf/ Vcid, Circuit	State	Last Chg	Uniq ID
1919779477	2135862268	2769003472	10142, Gi2.3504:3580	est	00:20:53	0
3934595965	1292798504	4135688314	10141, Gi2.3504:3580	est	00:07:59	0


```
R8#show l2tp session all vcid 10141 | section Sequenc
Sequencing is on
Ns 16, Nr 0, 0 out of order packets received
Packets switched/dropped by secondary path: Tx 0, Rx 0
```

XRv4 sends 100 pings to CSR10, who replies to them all. Just like with AToM sequencing, the numbers increase as expected. If packets were dropped due to being out of order, Ns/Nr would reveal it.

```
RP/0/0/CPU0:XRv4#ping vrf LINE 10.10.14.10 count 100
Type escape sequence to abort.
Sending 100, 100-byte ICMP Echos to 10.10.14.10, timeout is 2 seconds:
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Success rate is 100 percent (100/100), round-trip min/avg/max = 1/21/29 ms
```

```
R5#show l2tp session all tunnel id 292231207 1292798504 | section Sequenc
Sequencing is on
Ns 100, Nr 125, 0 out of order packets received
Packets switched/dropped by secondary path: Tx 0, Rx 0
```

```
R8#show l2tp session all vcid 10141 | section Sequenc
Sequencing is on
Ns 124, Nr 100, 0 out of order packets received
Packets switched/dropped by secondary path: Tx 0, Rx 0
```

The receive window size (RWS) indicates how many control packets can be queued for processing; smaller numbers are better for networks that cannot handle large bursts of traffic while large numbers allow more sessions (calls) to be processed faster. The default varies by platform, but is 1024 in the CSR1000v as the command shows this value as the “Remote RWS”. This value also does not need to match on both ends of the tunnel, so we configure 2048 on CSR5 while the peer uses the default value. As with all L2TP signaling changes in the L2TP-class, this will reset the session. Notice that we can also reference tunnels by “name”, which is the hostname. We specify the remote and local tunnel names versus the 32-bit unsigned integer tunnel IDs.

```
! CSR5
l2tp-class L2TP_ADVANCED
receive-window 2048
```

```
R5#show l2tp tunnel all remote-name R8 R5 | include RWS
Local RWS 2048, Remote RWS 1024
```

Another interesting component of L2TP sessions (calls) is interworking. With EVC an AToM, we could manipulate the VLAN tags with great flexibility. Without EVC, we can’t do much at all, which means interworking between disparate layer 2 technologies is important. The session type is “VLAN”; since the AC is running dot1q, the VLAN tags are carried across the PW. This is different than interworking for “Ethernet” which does not include the VLAN tags. The output below shows the session QinQ tag stack

{3554 1014}. This behaves a lot like VLAN interworking, but technically interworking is not configured. The detailed output does not make mention of interworking at all.

```
R5#show l2tp session interworking
L2TP Session Information Total tunnels 1 sessions 1
LocID      TunID      Peer-address      Type IWrk Username, Intf/
                                      Vcid, Circuit
3487589522 621690661 55.0.0.8          VLAN -           10141, Gi2.1014:3554/1014
```

```
R5#show l2tp session all vcid 10141 | section Ethernet Vlan
Session Layer 2 circuit, type is Ethernet Vlan, name is
GigabitEthernet2.1014:3554/1014
  Session vcid is 10141
  Circuit state is UP
    Local circuit state is UP
    Remote circuit state is UP
```

XRv4 will send small packets to CSR10 so we can keep the capture short. The VLAN tag stack of packets going from XRv4 to CSR10 (10.10.14.14 to 10.10.14.10 in yellow) is {3554 1014} in green as expected. The reply traffic is highlighted with the same colors and shows tag stack {3580 3504}. These are the correct VLAN encapsulations for each AC. L2TP is smart enough to remove those received tags before delivering the traffic to the forwarding process, which pushes the proper local AC tags on egress. With EFPs, we would normally have to explicitly pop or push the correct tags bidirectionally.

```
RP/0/0/CPU0:XRv4#ping vrf LINE 10.10.14.10 size 64
Type escape sequence to abort.
Sending 5, 64-byte ICMP Echos to 10.10.14.10, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/4/9 ms
```

```
R5#show monitor capture CAP buffer dump
1
 0000: 005056A9 9C600050 56A9DC63 81000DDF .PV..`.PV..c....
 0010: 08004500 00720418 00001073 37F53700 ..E..r.....s7.7.
 0020: 00053700 00088D20 EB664000 00730050 ..7.... .f@..s.P
 0030: 56A9F961 005056A9 862A8100 0DE28100 V..a.PV...*.....
 0040: 03F60800 45000040 00000000 FF018B91 ....E..@.....
 0050: 0A0A0E0E 0A0A0E0A 0800A2C3 40C60000 .....@...
 0060: ABCDABCD ABCDABCD ABCDABCD ABCDABCD .....
 0070: ABCDABCD ABCDABCD ABCDABCD ABCDABCD .....
 0080: ABCDABCD .....

2
 0000: 005056A9 DC630050 56A99C60 81000DDF .PV..c.PV..`. ....
 0010: 08004500 0072109B 40000773 F4713700 ..E..r..@..s.q7.
 0020: 00083700 0005CFE0 64924000 00900050 ..7.....d.@....P
 0030: 56A9862A 005056A9 F9618100 0DFC8100 V..*.PV..a.....
```

```

0040: 0DB00800 45000040 00000000 FF018B91  ....E...@.....
0050: 0A0A0E0A 0A0A0E0E 0000AAC3 40C60000  .....@...
0060: ABCDABCD ABCDABCD ABCDABCD ABCDABCD  .....
0070: ABCDABCD ABCDABCD ABCDABCD ABCDABCD  .....
0080: ABCDABCD  ....

```

Carrying VLAN tags was the default transport type to be negotiated, but there isn't a good reason to include VLAN tags in the tunnel. The interworking type must match between sites, but we can configure "Ethernet" interworking on all L2VPN contexts and PW classes to save 4-8 bytes of encapsulation. This configuration, like sequencing, is a session parameter, not a tunnel parameter, as this has nothing to do with the L2TP signaling.

```

! CSR5 and CSR8
l2vpn xconnect context XC_XRV14_R10
interworking ethernet

! CSR6
pseudowire-class PW_1014
interworking ethernet

```

Checking CSR5, the type is still VLAN, but interworking for Ethernet is now enabled.

```

R5#show l2tp session interworking
L2TP Session Information Total tunnels 1 sessions 1
LocID      TunID      Peer-address      Type IWrk Username, Intf/
                               Vcid, Circuit
980776166  621690661  55.0.0.8          VLAN ETH  10141, Gi2.1014:3554/1014

```

```

R5#show l2tp session all vcid 10141 | section Ethernet Vlan
Session Layer 2 circuit, type is Ethernet Vlan, name is
GigabitEthernet2.1014:3554/1014
Session vcid is 10141
Interworking type is Ethernet
Circuit state is UP
Local circuit state is UP
Remote circuit state is UP

```

The packet capture is the same as before, except with 8 bytes less overhead in each packet. Notice there are no dot1q tags present anywhere in the payload. The IP addresses are highlighted again, and the Ethertype for IP (0x0800) is all that remains. Interworking effectively removed the VLAN tags from the tunneled Ethernet frame as both L2TP endpoints agreed that this forwarding mechanism was supported.

```

R5#show monitor capture CAP buffer dump
0
0000: 005056A9 9C600050 56A9DC63 81000DDF  .PV...`.PV..c....
0010: 08004500 006A0427 00001073 37EE3700  ..E..j.'...s7.7.

```

```

0020: 00053700 00089A79 FE744000 000A0050 ..7....y.t@....P
0030: 56A9F961 005056A9 862A0800 45000040 V..a.PV...*..E..@
0040: 00000000 FF018B91 0A0A0E0E 0A0A0E0A .....
0050: 080072C3 70C60000 ABCDABCD ABCDABCD ..r.p.....
0060: ABCDABCD ABCDABCD ABCDABCD ABCDABCD .....
0070: ABCDABCD ABCDABCD ABCDABCD .....

```

1

```

0000: 005056A9 DC630050 56A99C60 81000DDF .PV..c.PV..`....
0010: 08004500 006A10C4 40000773 F4503700 ..E..j...@...s.P7.
0020: 00083700 00053A75 74E64000 00110050 ..7...:ut.@....P
0030: 56A9862A 005056A9 F9610800 45000040 V...*.PV..a..E..@
0040: 00000000 FF018B91 0A0A0E0A 0A0A0E0E .....
0050: 00007AC3 70C60000 ABCDABCD ABCDABCD ..z.p.....
0060: ABCDABCD ABCDABCD ABCDABCD ABCDABCD .....
0070: ABCDABCD ABCDABCD ABCDABCD .....

```

Other minor IP header options exist also, much like those found in a GRE tunnel configuration. Examples include TTL, TOS, and DF-bit handling. We can configure L2TP to reflect the tunneled TOS values into the L2TP header, or hardcode it to a specific value, where setting to 0 (clearing it completely) is the default behavior. The TTL represents the tunnel TTL which can be useful for limiting the “length” of a tunnel. The DF-bit prevents fragmentation of the post-encapsulated tunnel traffic in the network core. Fragmentation is generally undesirable and forcing the L2TP endpoints to fragment traffic will decrease performance significantly. These parameters do not need to match in either direction. CSR8 sets the TTL for all outgoing packets to 8 for both circuits, and sets the DF-bit. For CSR5, the TOS is reflected, and for CSR6 it is hard-coded to CS3. The remote L2TP routers use TTL 16 in the reverse direction, and CSR5 is the only router not setting the DF-bit. The QoS settings are matched both ways per PW.

```

! CSR8
template type pseudowire TMP_L2TP_PW
ip ttl 8
ip dfbit set

interface pseudowire10141
description PW TO CSR8
ip tos reflect

interface pseudowire10142
ip tos value 96

! CSR5
interface pseudowire10141
ip ttl 16
ip tos reflect

! CSR6
pseudowire-class PW_1014

```

```

ip dfbit set
ip tos value 96
ip ttl 16

```

Sending small packets from CSR10 to XRv4, we set the DSCP to CS1. This is reflected in the L2TP header as show below in yellow. The TTL arriving is 7 (green), where CSR8 initially set it to 8 and XRv1 decremented it. The TTL on the return packet is 0x10, which is 16 in decimal, as CSR5 is configured to use this value. Lastly, the DF-bit is set on traffic coming from CSR8 (0100 in binary) while it is cleared on packets coming from CSR5. Memorizing the exact position of the DF-bit in the IP header is unnecessary but I illustrate it here to prove that the configuration worked. The source/destination IP addresses inside the VPN are shown in pink.

```
R5#show monitor capture CAP buffer dump
```

```

0
0000: 005056A9 DC630050 56A99C60 81000DDF .PV..c.PV..`....
0010: 08004520 006A10F6 40000773 F3FE3700 ..E .j..@...s..7.
0020: 00083700 00053A75 74E64000 00430050 ..7...:ut.@..C.P
0030: 56A9862A 005056A9 F9610800 45200040 V..*.PV..a..E .@
0040: 00390000 FF018B38 0A0A0E0A 0A0A0E0E .9.....8.....
0050: 08006380 000C0000 00000000 09B32581 ..c.....%.
0060: ABCDABCD ABCDABCD ABCDABCD ABCDABCD .....
0070: ABCDABCD ABCDABCD ABCDABCD .....

1
0000: 005056A9 9C600050 56A9DC63 81000DDF .PV..`.PV..c....
0010: 08004520 006A0436 00001073 37BF3700 ..E .j.6...s7.7.
0020: 00053700 00089A79 FE744000 00190050 ..7...y.t@....P
0030: 56A9F961 005056A9 862A0800 45200040 V..a.PV..*..E .@
0040: 00390000 FF018B38 0A0A0E0E 0A0A0E0A .9.....8.....
0050: 00006B80 000C0000 00000000 09B32581 ..k.....%.
0060: ABCDABCD ABCDABCD ABCDABCD ABCDABCD .....
0070: ABCDABCD ABCDABCD ABCDABCD .....

```

We will shut down the PW on CSR5 so that the failover circuit to CSR6 is activated (not shown). We also switch over XRv4 to use this circuit by shuffling the IP addresses between interfaces (L2 bridging isn't supported, so it's an artificiality of the test). Capturing bidirectionally on CSR6, we see slightly different behavior in some places. Most notably, the L2TP header DSCP is now CS3 (hard coded) versus CS1 (original customer value). This is because the configuration specified a fixed value and is not reflecting TOS from the inner header to the outer header. The DF-bit is set in both directions, as configured (cyan). The asymmetrical TTL behavior is identical to before as the configurations on CSR5 and CSR6 are identical here.

```
R6#show monitor capture CAP buffer dump
```

```

1
0000: 005056A9 DE0D0050 56A99C60 81000DE9 .PV....PV..`....
0010: 08004560 00661118 40000773 F39F3700 ..E`.f..@...s..7.

```

```

0020: 00083700 00066861 496A0050 56A9862A ..7...haIj.PV...*
0030: 005056A9 F9610800 45200040 00480000 .PV...a..E .@.H..
0040: FF018B29 0A0A0E0A 0A0A0E0E 0800108F ...).....
0050: 000F0000 00000000 09B97869 ABCDABCD .....xi....
0060: ABCDABCD ABCDABCD ABCDABCD ABCDABCD .....
0070: ABCDABCD ABCDABCD .....

```

2

```

0000: 005056A9 9C600050 56A9DE0D 81000DE9 .PV...`.PV.....
0010: 08004560 0066000D 40001073 FBAA3700 ..E`.f...@...s..7.
0020: 00063700 0008F961 41D30050 56A9F961 ..7....aA..PV..a
0030: 005056A9 862A0800 45200040 00480000 .PV...*..E .@.H..
0040: FF018B29 0A0A0E0E 0A0A0E0A 0000188F ...).....
0050: 000F0000 00000000 09B97869 ABCDABCD .....xi....
0060: ABCDABCD ABCDABCD ABCDABCD ABCDABCD .....
0070: ABCDABCD ABCDABCD .....

```

Regarding status, you can configure CSR8 to show both the primary and secondary links up at the same time. This allows the head router, CSR8, to advertise UP status to CSR6 so that both tunnels can show up. The xconnect process ensures only the primary one forwards traffic. Normally, CSR6 shows the link as down due to the status signaling, even though the L2TP tunnel is up.

```

R6#show l2vpn service xconnect all | begin Interface
  Interface          Group          Encapsulation          Prio  St  XC  St
  -----          -
VPWS name: Gi2.3546-3546, State: DN
  Gi2.3546           left           Gi2.3546:3546(Eth VLAN)  0      UP  DN
  pw100002           right          55.0.0.8:10142(L2TP)    0      DN  DN

```

We will make the change on CSR8 so that it views the topology as dual-homed.

```

! CSR8
template type pseudowire TMP_L2TP_PW
  encapsulation l2tpv3
  status peer topology dual-homed

```

Now, CSR6 sees the link as UP and relies on the customer redundancy protocol.

```

R6#show l2vpn service xconnect all | begin Interface
  Interface          Group          Encapsulation          Prio  St  XC  St
  -----          -
VPWS name: Gi2.3546-3546, State: UP
  Gi2.3546           left           Gi2.3546:3546(Eth VLAN)  0      UP  UP
  pw100002           right          55.0.0.8:10142(L2TP)    0      UP  UP

```

CSR8 still treats this like a backup link, but allows CSR6 to remain online. The assumption is that there is some kind of redundancy protocol at the remote end; in our case, we are just changing interfaces for testing, but this is a valid configuration. The same holds true for AToM but is demonstrated here.

```
R8#show l2vpn service xconnect name XC_XRV14_R10 | begin Interface
```

Interface	Group	Encapsulation	Prio	St	XC	St
-----	-----	-----	----	--	-----	-----
VPWS name: XC_XRV14_R10, State: UP						
Gi2.3504		Gi2.3504:3580/3504 (Eth VLAN)	0	UP	UP	
pw10141	R10_XRV14	55.0.0.5:10141 (L2TP)	3	UP	UP	
pw10142	R10_XRV14	55.0.0.6:10142 (L2TP)	6	SB	IA	

Next, we will examine a static L2TP PW configuration. Just like with AToM, we can manually configure a PW between two sites but it requires the administrator to specify some low-level details. For AToM, this included local and remote PW labels. Here, it includes local and remote session IDs. The idea is the same as these IDs are normally negotiated when L2TP signaling is used. Additionally, we define local and remote “cookies”. These are additional guarantees that a packet along L2TP session is destined for the proper node. When packets are sent from one router to another, the remote session ID and remote cookie are included in the header. Below, I selected values easy to read in hex, making the EPC easier to comprehend. The session IDs are 0x4444 and 0x8888 for CSR4 and CSR8, respectively. The cookies are 0x8C8C and 0x4C4C for CSR8 and CSR4, respectively. I used untagged interfaces (dedicated) ports for this as static L2TP is not documented to work with VLAN mode. Interworking of any kind is also unsupported. CSR8 uses the new L2VPN CLI commands while CSR4 uses the legacy xconnect configuration.

```
! CSR8
interface pseudowire110
  description PW R10-R1
  encapsulation l2tpv3
  neighbor 55.0.0.4 110
  signaling protocol none
  ip local interface Loopback0
  l2tp id 34952 17476
  l2tp cookie local 4 35980
  l2tp cookie remote 4 19532

interface GigabitEthernet3
  description PW TO CSR1

l2vpn xconnect context XC_R10_R1
  member pseudowire110
  member GigabitEthernet3

! CSR4
pseudowire-class PW_L2TP_STATIC
  encapsulation l2tpv3
```

```

protocol none
ip local interface Loopback0

interface GigabitEthernet3
description PW TO CSR10
xconnect 55.0.0.8 110 encapsulation l2tpv3 manual pw-class PW_L2TP_STATIC
l2tp id 17476 34952
l2tp cookie local 4 19532
l2tp cookie remote 4 35980

```

Like AToM static PWs, these sessions are always UP, but there is no associated tunnel for signaling. CSR4 has no dynamic P2TP circuits, and thus has no tunnels. It does have a single session, though.

```

R4#show l2tp tunnel
%No active L2TP tunnels

```

```

R4#show l2tp session
L2TP Session Information Total tunnels 0 sessions 1
LocID      RemID      TunID      Username, Intf/      State  Last Chg Uniq ID
          Vcid, Circuit
17476      34952      n/a        110, Gi3              est    00:16:00 0

```

A quick checking of the interworking shows that it is disabled, but the session type is ETH and not VLAN. When shown on CSR8, the difference is clear, as the other circuits are of type VLAN but are using Ethernet interworking to strip dot1q tags from tunneled Ethernet frames, as discussed earlier. The type is shown as ETH because a dedicated physical interface is used for this test.

```

R8#show l2tp session interworking
L2TP Session Information Total tunnels 2 sessions 3
LocID      TunID      Peer-address  Type IWrk Username, Intf/
          Vcid, Circuit
1428121045 1008940922 55.0.0.6      VLAN ETH  10142, Gi2.3504:3580/3504
4015951127 1455401177 55.0.0.5      VLAN ETH  10141, Gi2.3504:3580/3504
34952      n/a        55.0.0.4      ETH -     110, Gi3

```

We can also check the session details. Specifically, we will look at the cookie and cached encapsulation information. We can see the 4-byte cookie values in hex, which is what we expect to see in the packet captures as well. The IP header encapsulation is shown below, which includes the remote session ID and remote cookie. Immediately following the remote cookie would be the payload; in this case, the beginning of the destination MAC address in the tunneled Ethernet frame. This can be very helpful for manually verifying captured packets via EPC.

```

R8#show l2tp session all vcid 110 | section cookie|cache
Session cookie information:
  local cookie, size 4 bytes, value 00 00 8c 8c
  remote cookie, size 4 bytes, value 00 00 4c 4c

```



```

FS cached header information:
  encaps size = 28 bytes
  45000014 00000000 ff734d6b 37000008
  37000004 00004444 00004c4c

```

As expected, we see no L2TP signaling packets of any kind. Since CSR4 has only one static PW, it has never participated in L2TP signaling, so all control counters are zero. This might be confusing or a false negative when troubleshooting L2TP if you don't understand how static PWs operate.

```

R4#show l2tp counters tunnel
Global L2TP tunnel control message statistics:

```

	XMIT =====	RE-XMIT =====	RCVD =====	DROP =====
ZLB	0	0	0	0
SCCRQ	0	0	0	0
SCCRP	0	0	0	0
SCCCN	0	0	0	0
StopCCN	0	0	0	0
Hello	0	0	0	0
OCRQ	0	0	0	0
OCRP	0	0	0	0
OCCN	0	0	0	0
ICRQ	0	0	0	0
ICRP	0	0	0	0
ICCN	0	0	0	0
CDN	0	0	0	0
WEN	0	0	0	0
SLI	0	0	0	0
EXP ACK	0	0	0	0
SRRQ	0	0	0	0
SRRP	0	0	0	0
CiscoACK	0	0	0	0
Total	0	0	0	0

Capturing bidirectionally on CSR4, we can see packets entering CSR4 with its local session ID and cookie values (0x4444 and 0x4C4C). When CSR4 sends traffic to CSR8, it uses the remotely configured session ID and cookie values (0x8888 and 0x8C8C). If any of these parameters do not match, the L2TP process discards the packets upon receipt. Just like PW labels, these unique values serve as demultiplexers so the egress PE knows how to map the packet to the proper customer/service. These session ID and cookie fields are shown in yellow. Notice that the traffic was sent with DSCP CS2, yet it was not copied into the L2TP header. This is because TOS reflection was not configured (it could have been) and the default action is to use DSCP 0 (green).

```

R10#ping vrf LINE 10.1.10.1 tos 64
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.1.10.1, timeout is 2 seconds:

```

!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 5/8/17 ms

R4#show monitor capture CAP buffer detailed

```
  2 160      0.146996    55.0.0.8          -> 55.0.0.4          L2TP
0000: 005056A9 2C570050 56A90E6F 81000DD6    .PV.,W.PV..o....
0010: 08004500 008E001F 0000FD73 4ED23700    ..E.....sN.7.
0020: 00083700 00040000 44440000 4C4C0050    ..7.....DD..LL.P
0030: 56A95796 005056A9 42EF0800 45400064    V.W..PV.B...E@d

  3 160      0.150002    55.0.0.4          -> 55.0.0.8          L2TP
0000: 005056A9 0E6F0050 56A92C57 81000DD6    .PV..o.PV.,W....
0010: 08004500 008E000F 0000FF73 4CE23700    ..E.....sL.7.
0020: 00043700 00080000 88880000 8C8C0050    ..7.....P
0030: 56A942EF 005056A9 57960800 45400064    V.B..PV.W...E@d
```

Recall that we have RIPv2 running across these P2P circuits, so CSR1 should have reachability to XRv4 and vice versa. Traffic hairpins through CSR10, but connectivity is achievable. For brevity, we will test it in one direction.

R1#show ip route vrf LINE 10.10.14.0

Routing Table: LINE

Routing entry for 10.10.14.0/24

Known via "rip", distance 120, metric 1

Redistributing via rip

Last update from 10.1.10.10 on GigabitEthernet3, 00:00:11 ago

Routing Descriptor Blocks:

* 10.1.10.10, from 10.1.10.10, 00:00:11 ago, via GigabitEthernet3

Route metric is 1, traffic share count is 1

R1#ping vrf LINE 10.10.14.14

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 10.10.14.14, timeout is 2 seconds:

!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 11/31/45 ms

R1#traceroute vrf LINE 10.10.14.14

Type escape sequence to abort.

Tracing the route to 10.10.14.14

VRF info: (vrf in name/id, vrf out name/id)

```
 1 10.1.10.10 11 msec 7 msec 7 msec
 2 10.10.14.14 10 msec 15 msec 34 msec
```

Last, we will quickly look at L2TP on XRv. We know L2VPN isn't going to work, but we expect the control-plane to converge as it did for MPLS-encapsulated PWs. The configuration itself is very straightforward, using the same general concepts as XE. CSR6 creates a dummy AC and PW to peer with XRv3.

! XRv3

```
interface GigabitEthernet0/0/0/0.503 l2transport
 encapsulation dot1q 3503 exact
```

```
l2vpn
 logging
 pseudowire
```

```
pw-class L2TP_CLASS
 encapsulation l2tpv3
 protocol l2tpv3
 ipv4 source 55.0.0.13
```

```
xconnect group L2TP
 p2p L2TP_CSR6
 interface GigabitEthernet0/0/0/0.503
 neighbor ipv4 55.0.0.6 pw-id 613
 pw-class L2TP_CLASS
 interworking ethernet
```

```
! CSR6
interface GigabitEthernet2.999
 description DUMMY AC
 encapsulation dot1Q 999
```

```
interface pseudowire613
 description DUMMY PW CSR6-XRV3
 encapsulation l2tpv3
 neighbor 55.0.0.13 613
 signaling protocol l2tpv3
 ip local interface Loopback0
```

```
l2vpn xconnect context XRV_TEST
 interworking ethernet
 member GigabitEthernet2.999
 member pseudowire613
```

IOS XR apparently does not support sequencing and interworking concurrently, although XE seems to support it just fine. The configuration is rejected when you try to commit. Interworking is left enabled and sequencing is disabled.

```
! XRV3
l2vpn
 xconnect group L2TP
 p2p L2TP_CSR6
 interworking ethernet
```

```
!!% Invalid argument: Sequencing not supported with interworking
```

Unfortunately, due to a Cisco bug, the session never establishes, even if the control tunnel does. L2TP event debugging on XRv3 reveals an issue with the ICRP message (incoming-call-reply) missing some AVPs. The bug **CSCus72718** clearly states: *“L2TPv3 session between 2 devices is stuck at pending state, wiccn and wicrp respectively. Following error might be observed in the [debug] logs of older release: Missing AVPs in ICRP Cisco Core AVP Pseudo Wire Type missing.*

```
! XRv3
l2tp_mgr[1176]: L2TP      :      :
l2tp_mgr[1176]: L2TP      :      : Shutting down session
l2tp_mgr[1176]: L2TP      :      : Result Code
l2tp_mgr[1176]: L2TP      :      : Reserved (0)
l2tp_mgr[1176]: L2TP      :      : Error Code
l2tp_mgr[1176]: L2TP      :      : Unknown AVP, sent as
mandatory (8)
l2tp_mgr[1176]: L2TP      :      : Vendor Error
l2tp_mgr[1176]: L2TP      :      : None (0)
l2tp_mgr[1176]: L2TP      :      : Optional Message
l2tp_mgr[1176]: L2TP      :      : "Missing AVPs in ICRP Cisco
Core AVP Pseudo Wire Type missing"
l2tp_mgr[1176]: L2TP tn1   0805D:5DA568FD: ERROR: ICRP Error parsing AVPs
```

I assume the ‘W’ before the ICCN and ICRP messages in the bug log stand for “waiting”. We can verify this by checking CSR6. Sure enough, we see exactly that message, which leads me to believe this is bug related and will not work on the current version of software. The bug specifically cites version 15.4(1)T2.2, which is an IOS issue, not an IOS-XR or XRv-specific issue. The version currently in use is shown below (newer), but this bug is recent and still open. The details of the L2TP session show a functioning control channel and a partially-built session.

```
R6#show version | include 15\.4
Cisco IOS Software, CSR1000V Software (X86_64_LINUX_IOSD-UNIVERSALK9-M),
Version 15.4(3)S2, RELEASE SOFTWARE (fc3)

R6#show l2tp session
L2TP Session Information Total tunnels 2 sessions 2
LocID      RemID      TunID      Username, Intf/      State  Last Chg Uniq ID
          Vcid, Circuit
3011146421 1323365658 182935741 613, Gi2.999:999    wiccn  00:07:01 0
4049220061 1428121045 3716203250 10142, Gi2.3546:3546 est    02:54:25 0

R6#show l2tp session all vcid 613
L2TP Session Information Total tunnels 2 sessions 2
Session id 3011146421 is down, logical session id 66238, tunnel id 182935741
Remote session id is 1323365658, remote tunnel id 1571121405
Remotely initiated session
Unique ID is 0
Session Layer 2 circuit, type is Ethernet Vlan, name is
GigabitEthernet2.999:999
```

```

Session vcid is 613
Interworking type is Ethernet
Circuit state is UP
  Local circuit state is UP
  Remote circuit state is UP
Call serial number is 927600002
Remote tunnel name is XRv3
  Internet address is 55.0.0.13
Local tunnel name is R6
  Internet address is 55.0.0.6
IP protocol 115
Session is L2TP signaled
Session state is Wt-Rx-ICCN, time since change 00:07:24
  0 Packets sent, 0 received
  0 Bytes sent, 0 received
[snip]

```

The show commands and outputs on XRv are almost identical. Some basic outputs are shown below. The tunnel is up, but the session is stuck. Notice that XRv3 is waiting on the ICRP (incoming-call-reply). This means that XRv3 sent the initial ICRQ (incoming-call-request) and thus initiated the session. It is waiting for the “SYN ACK” while CSR6 is waiting for the final “ACK”, to use TCP terms. We can prove it by checking the details and seeing the session is locally initiated.

```

RP/0/0/CPU0:XRv3#show l2tp tunnel brief
LocTunID  RemTunID  Remote Name  State  Vrf Name  Remote Address  Sessn
L2TP Class/
Count
VPDN Group
870710611  741828106  R6          est    55.0.0.6  1
l2tp_default_class

```

```

RP/0/0/CPU0:XRv3#show l2tp session brief
LocID      TunID      Peer-address  State      Username, Intf/
sess/cir   Vcid, Circuit
3614735818 870710611  55.0.0.6     wicrp,DN  613, Gi0/0/0/0.503

```

```

RP/0/0/CPU0:XRv3#show l2tp session detail
Session id 3614735818 is down, tunnel id 870710611, logical session id 32789
Remote session id is 0, remote tunnel id 741828106
Locally initiated session
Call serial number is 927600007
Remote tunnel name is R6
  Internet address is 55.0.0.6:1248
Local tunnel name is XRv3
  Internet address is 55.0.0.13:4211
IP protocol 115
Session is L2TP signaled
Session state is Wt-Rx-ICRP, time since change 00:06:29

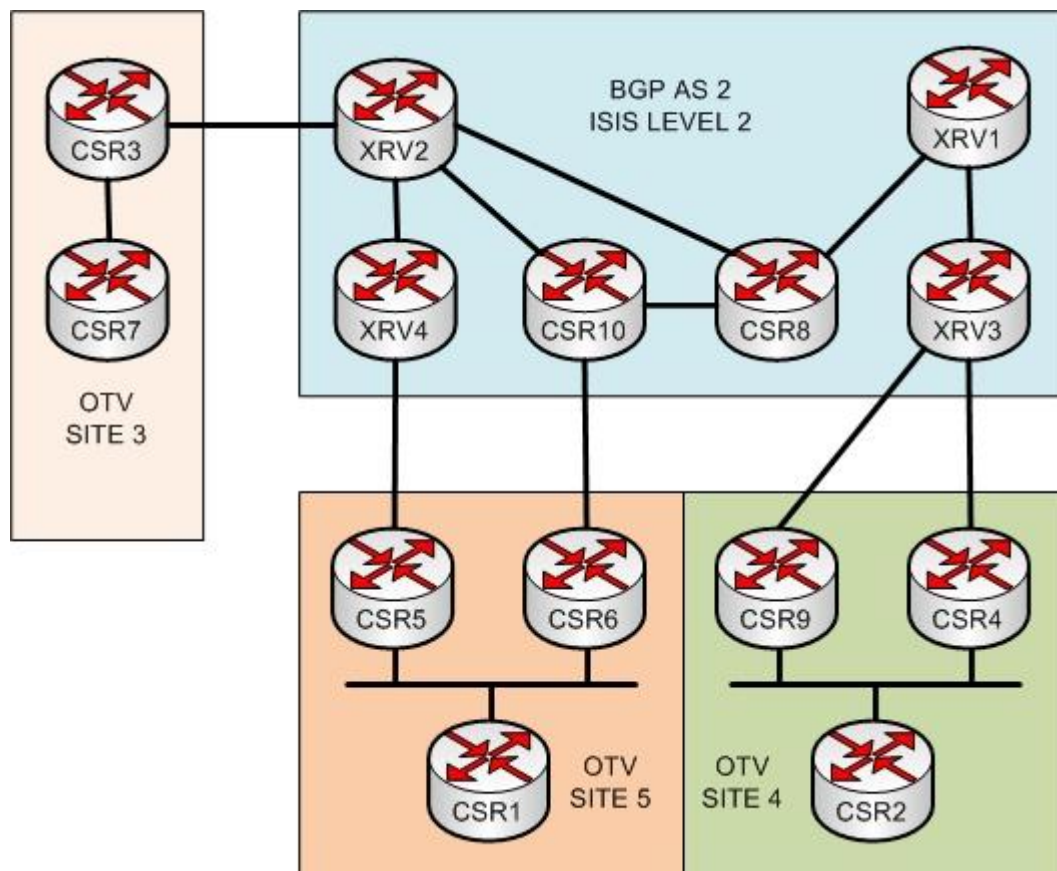
```

[snip]

Additional Reading – Reference configurations "l2vpn-l2tp"

32.2.2 E-LAN and E-TREE using OTV

Overlay Transport Virtualization (OTV) is a MP2MP L2VPN technology that is typically used as a “layer 2 stretch” for DCI (data center interconnection) designs. Unlike MPLS L2VPNs, this is a customer-side protocol, and cannot be enabled on a router with any MPLS features enabled. Like DMVPN, it is not a SP technology directly, but could be used by an SP if the CPE is SP owned/managed. OTV introduces many improvements over VPLS, including simplified OAM/provisioning, better handling of BUM traffic, and no requirement for MPLS transport. It is currently Cisco-proprietary but standardizing drafts are in progress. The diagram below shows a small SP network which uses draft Rosen (MVPN profile 0) for customer multicast transport; this is the only profile currently supported in XRv. There are three customer sites, two of which are dual-homed. The customer has an IP-only MPLS L3VPN. CSR1, CSR2, and CSR7 represent hosts on two different VLANs using QinQ for testing purposes. The goal is to stretch these two VLANs over the SP network between all sites.



The OTV configuration is very straightforward. Each site must have a site-ID which should be common for each OTV-speaker at the site, as well as a site bridge-domain (site VLAN in Nexus) which is used for intra-site communication. An Authoritative Edge Device (AED) is elected at each site and is loosely

analogous to a PIM Bidir designated forwarder (DF). It is responsible for sending/receiving traffic to/from the overlay. Other routers are in place for redundancy. The configuration on all OTV routers is almost identical and only CSR5 is shown for brevity. All routers must agree on the control-group within an overlay as this is used for dynamic discovery; XRv2 is the RP within the MVPN for VRF OTV. This is similar to the MDT default-group used with MVPN profile 0. The data-groups, much like MDT data-groups, do not have to match, and are spawned for non-LL multicast flows. EFPs configured on the LAN-side are configured similarly on the OTV interface and bridged together with BDs. The EFPs must ensure at least one VLAN tag remains on the frames so OTV can properly classify it; for this reason, we use QinQ and pop only the topmost tag. The site-ID and site-BD relate to the AED configuration; CSR6 and CSR5 will have identical values so the two can participate in the AED election. The OTV join-interface is the upstream, SP-facing link where IGMPv3 joins are issued. IGMPv3 technically isn't required unless you are using SSM, which we are not, but PIM passive is enabled to allow the router to process/forward multicast. Because there is no PE-CE routing, a static default is configured on each CE router.

```
! CSR5
otv site bridge-domain 3556
otv site-identifier 0000.0000.0005

interface GigabitEthernet2.554
 encapsulation dot1q 3554
 ip address 10.5.14.5 255.255.255.0
 ip pim passive
 ip igmp version 3

interface Overlay1
 no ip address
 otv control-group 239.0.0.1
 otv data-group 239.0.1.0/24
 otv join-interface GigabitEthernet2.554
 service instance 100 ethernet
  encapsulation dot1q 100
  bridge-domain 100

service instance 200 ethernet
 encapsulation dot1q 200
 bridge-domain 200

interface GigabitEthernet2
 service instance 100 ethernet
  encapsulation dot1q 3515 second-dot1q 100
  rewrite ingress tag pop 1 symmetric
  bridge-domain 100

service instance 200 ethernet
 encapsulation dot1q 3515 second-dot1q 200
 rewrite ingress tag pop 1 symmetric
 bridge-domain 200
```

```

service instance 3556 ethernet
  encapsulation dot1q 3556 exact
  bridge-domain 3556

ip route 0.0.0.0 0.0.0.0 10.5.15.15

```

As a quick note before starting with OTV, all of the flexibility with EFP/EVC VLAN manipulation appears to be supported. The configuration above is a simple approach; I use a more convoluted approach on CSR4/CSR9 to demonstrate the EVC CLI usefulness again. CSR2 is sending traffic to VLANs 101 and 102 when it should be 100 and 200. This is also inside QinQ; CSR4 and CSR9 can do a 2:1 mapping to change those tags to 100 or 200, which stages the proper frame for OTV to process. Provided the symmetric keyword is used, or the proper VLAN tags are pushed in the opposite direction, this approach works for VLAN translations between data centers in conjunction with OTV. The higher layer protocols (IP, IPv6, etc) must still be able to communicate, such as being in the same subnet, etc. The OTV site-BD (AED elections, etc) is still required, as always. CSR4 uses both the “symmetric” and manual strategies between the different “stretched” VLANs. The OTV BD and site-ID match between CSR4 and CSR9 as well, which is critical for the AED election.

```

! CSR4
otv site bridge-domain 3549
otv site-identifier 0000.0000.0004

interface GigabitEthernet2
  service instance 100 ethernet
  encapsulation dot1q 3529 second-dot1q 101
  rewrite ingress tag translate 2-to-1 dot1q 100
  rewrite egress tag translate 1-to-2 dot1q 3529 second-dot1q 101

service instance 200 ethernet
  encapsulation dot1q 3529 second-dot1q 102
  rewrite ingress tag translate 2-to-1 dot1q 200 symmetric
  bridge-domain 200

service instance 3549 ethernet
  encapsulation dot1q 3549 exact
  bridge-domain 3549

```

Assuming a near-identical configuration on the other OTV routers, we can verify the adjacencies are up. The internal system-ID is really an IS-IS NET which was auto-generated by the OTV process. The site-ID was manually configured and should be unique across sites, but common within a site. From CSR9’s perspective, we can see the other 4 OTV routers.

```

R9#show otv adjacency
Overlay Adjacency Database for overlay 1

```

Hostname	System-ID	Dest Addr	Site-ID	Up Time	State

R3	001e.f675.ce00	10.3.12.3	0000.0000.0003	00:20:30	UP
R6	001e.bd8f.6000	10.6.10.6	0000.0000.0005	00:20:31	UP
R4	001e.4977.7c00	10.4.13.4	0000.0000.0004	00:13:37	UP
R5	001e.14ff.9b00	10.5.14.5	0000.0000.0005	00:20:30	UP

We can change the IS-IS NET for routers to make the system-ID more readable, but since OTV carries the hostnames, it isn't really helpful. We also must shutdown the Overlay and the site BD to change the NET. We will also enable IS-IS adjacency logging, which is off by default, on CSR6 as an example. This only shows neighbors that are within the "Site" process which is used for AED elections and intra-site signaling, not overlay peers.

```
! CSR6
otv isis Site
 net 00.0000.beef.0006.00
 log-adjacency-changes all
```

```
%CLNS-5-ADJCHANGE: ISIS (Site): Adjacency to 001E.14FF.9B00 (OTV-Site) Up,
new adjacency
```

We can see the new NET by checking another router's adjacencies.

```
R3#show otv adjacency
Overlay Adjacency Database for overlay 1
```

Hostname	System-ID	Dest Addr	Site-ID	Up Time	State
R9	001e.e58c.1f00	10.9.13.9	0000.0000.0004	00:25:18	UP
R4	001e.4977.7c00	10.4.13.4	0000.0000.0004	00:18:24	UP
R5	001e.14ff.9b00	10.5.14.5	0000.0000.0005	14:05:37	UP
R6	0000.beef.0006	10.6.10.6	0000.0000.0005	00:01:26	UP

CSR5 and CSR6 will not see each other since the SP is not providing L3VPN service between them since they are in the same site. This does not affect OTV directly, provided CSR5 and CSR6 cannot be the ISIS DIS. Since they don't have direct connectivity to all nodes, they should never be elected DIS; this OTV design is similar to E-TREE where CSR5 and CSR6 are leaves. We can configure many ISIS parameters under the OTV interface, to include authentication. We can also remove the hello padding to save some WAN bandwidth cost. We will make these modifications quickly and verify them.

```
! CSR5 and CSR6 only
interface Overlay1
 otv isis priority 0
```

```
! All OTV routers
interface Overlay1
 no otv isis hello padding
 otv isis authentication mode md5
 otv isis authentication key-chain KC_ISIS_AUTH
 otv isis priority 0
```

We can see quite a bit of raw IS-IS information within the OTV process. Looking at the adjacencies, we have L1 adjacencies over the Overlay interfaces as well as the OTV-Site BD. In this case, we know CSR5 and CSR6 cannot form adjacencies over the Overlay due to lack of L3VPN connectivity, but within an entirely different IS-IS process, they form a neighbor to exchange AED information across the OTV BD. We can see both are broadcast networks since the Circuit-ID lists the IS-IS DIS (non-zero N-selector).

```
R5#show otv isis neighbors
```

Tag Overlay1:

System Id	Type	Interface	IP Address	State	Holdtime	Circuit Id
R4	L1	Ov1	10.4.13.4	UP	26	R3.01
R3	L1	Ov1	10.3.12.3	UP	7	R3.01
R6	L1	Ov1	10.6.10.6	INIT	29	R6.01
R9	L1	Ov1	10.9.13.9	UP	24	R3.01

Tag Site:

System Id	Type	Interface	IP Address	State	Holdtime	Circuit Id
0000.BEEF.0006	L1	OTV-Site		UP	26	R5.01

Checking the DIS, which is currently being represented as a pseudo-node on CSR3, we can see all connected IS-IS routers on the segment within the overlay.

```
R5#show otv isis database R3.01-00 detail
```

Tag Overlay1:

IS-IS Level-1 LSP R3.01-00

LSPID	LSP Seq Num	LSP Checksum	LSP Holdtime	ATT/P/OL
R3.01-00	0x0000006E	0x3544	962	0/0/0
Metric: 0	IS-Extended R3.00			
Metric: 0	IS-Extended R4.00			
Metric: 0	IS-Extended R6.00			
Metric: 0	IS-Extended R5.00			
Metric: 0	IS-Extended R9.00			

Dual-homed sites 4 and 5 will conduct an AED election on a per-VLAN basis. This allows for resiliency and load-sharing, assuming that each router selects half of the VLANs. The following debug on CSR5 summarizes the process; the routers compute hashes based on the VLAN ID and determine who will be the AED. In this case, CSR5 said that it is both the old and new AED, possibly because it was configured first. It seems that any custom NET will break the AED election. Neither router participates, and no one will become AED, even after a reload. This totally breaks connectivity, so I have removed the NET configuration on CSR6 as a workaround (not shown). Both routers claim to be inactive for all VLANs. We confirm the NET is no longer the custom string on CSR6.

```
R6#show otv isis protocol
```

Tag Overlay1:

IS-IS Router: Overlay1

```

System Id: 001E.BD8F.6000.00 IS-Type: level-1
Manual area address(es):
    00
Routing for area address(es):
    00
Interfaces supported by IS-IS:
    Overlay1

```

Tag Site:

```

IS-IS Router: Site
System Id: 001E.BD8F.6000.00 IS-Type: level-1
Manual area address(es):
    00
Routing for area address(es):
    00
Interfaces supported by IS-IS:
    OTV-Site

```

Now, we can examine the AED election. I've stripped some of the chatty debugs to focus on the important aspects.

```

CSR5#debug otv aed-svr-elec
CSR5#debug otv aed-svr-req
OTV-APP-AED-SVR-REQ: otv_aed_election_type_hash for vlan 100
OTV-APP-AED-SVR-REQ: otv_aed_election_type_hash for vlan 200
OTV-APP-AED-SVR-REQ: otv_validate_aed_svr_req_version Local BD:100 version
updated to:3 old-ver:2
OTV-APP-AED-SVR-REQ: otv_enforce_aed_svr_transaction: me old aed1, me old bkp
0, me new aed 1 me new bkp 0
OTV-APP-AED-SVR-REQ: otv_validate_aed_svr_req_version Local BD:200 version
updated to:3 old-ver:2
OTV-APP-AED-SVR-REQ: otv_enforce_aed_svr_transaction: me old aed1, me old bkp
0, me new aed 1 me new bkp 0

```

In case you miss the debugs, you can see the AED per VLAN after the election completes. In our case, we were unfortunate not to have any load sharing based on our VLAN IDs. CSR5 is the AED for both VLANs at site 5 which is why there is an asterisk next to its hostname. CSR9 is not the AED for either VLAN and its state is inactive; it cannot forward traffic into the overlay.

```

R5#show otv vlan
Key:  SI - Service Instance, NA - Non AED, NFC - Not Forward Capable.
Overlay 1 VLAN Configuration Information
  Inst VLAN BD   Auth ED           State           Site If(s)
  0    100  100  *R5             active         Gi2:SI100
  0    200  200  *R5             active         Gi2:SI200
Total VLAN(s): 2

R9#show otv vlan

```

Key: SI - Service Instance, NA - Non AED, NFC - Not Forward Capable.

Overlay 1 VLAN Configuration Information

Inst	VLAN	BD	Auth ED	State	Site If(s)
0	100	100	R4	inactive(NA)	Gi2:SI100
0	200	200	R4	inactive(NA)	Gi2:SI200
Total VLAN(s): 2					

We can verify that “MAC routing” is working correctly. CSR5 learns and installs all the remote MAC addresses since it is the AED for both VLANs. CSR6 only installs the remote routes. The output specifically says how the route was learned: IS-IS from a peer or a local BD EFP with service-instance number. We can see the AD of local routes is 40 and remote routes is 50, which can prevent suboptimal routing. Because VLAN 100 and VLAN 200 have the same MAC addresses, I just shown VLAN 100. For CSR6, it only has remote routes for VLAN 200. Since it isn’t the AED, it cannot learn local MAC addresses on its EFP.

```
R5#show otv route vlan 100
```

Codes: BD - Bridge-Domain, AD - Admin-Distance,
SI - Service Instance, * - Backup Route

OTV Unicast MAC Routing Table for Overlay1

Inst	VLAN	BD	MAC Address	AD	Owner	Next Hops(s)
0	100	100	001e.bd8f.60bc	40	BD Eng	Gi2:SI100
0	100	100	001e.e58c.1fbc	50	ISIS	R4
0	100	100	0050.56a9.be8a	50	ISIS	R4
0	100	100	0050.56a9.ea77	50	ISIS	R3

```
R6#show otv route vlan 200 | begin MAC
```

OTV Unicast MAC Routing Table for Overlay1

Inst	VLAN	BD	MAC Address	AD	Owner	Next Hops(s)
0	200	200	001e.e58c.1fbc	50	ISIS	R4
0	200	200	0050.56a9.be8a	50	ISIS	R4
0	200	200	0050.56a9.ea77	50	ISIS	R3

For additional detail, we can look at the IS-IS VLAN information. I removed the first column, which was “cluster-ID” and always zero. We saw the CSR6 NET earlier as 001E.BD8F.6000, which is “Backup AED” for many of these routes, in both VLANs, since it is not the AED for either (yellow). Below it, we see more backup paths but at site 4; we presume this is the non-AED router (green). We confirm that the system-ID belongs to CSR9 who is indeed not the AED, as we saw earlier. Most of the other entries in the table all show “active” as those are the AED-owned.

```
R5#show otv isis vlan status remote info
```

```
Tag Overlay1:OTV ISIS VLAN status remote info
```

```
[ Flags: A-AED, B-Backup AED, F-FwdReady,D- Delete,L-Local, R-Remote ]
```

Partition-ID	Device-ID	Site-ID	Flags	pri	vlan	version	site-index	updated(ago)
FFFF.FFFF.FFFF	0000.BEEF.0006	0000.0000.0005	----LR	0	100	0	0	00:09:46

```

FFFF.FFFF.FFFF 0000.BEEF.0006 0000.0000.0005 ----LR 0 200 0 0 00:09:46
FFFF.FFFF.FFFF 001E.4977.7C00 0000.0000.0004 FA--LR 0 100 3 1 00:25:45
FFFF.FFFF.FFFF 001E.4977.7C00 0000.0000.0004 FA--LR 0 200 3 1 00:25:45
FFFF.FFFF.FFFF 001E.BD8F.6000 0000.0000.0005 F-B-LR 0 100 9 0 00:05:58
FFFF.FFFF.FFFF 001E.BD8F.6000 0000.0000.0005 F-B-LR 0 200 9 0 00:05:58
FFFF.FFFF.FFFF 001E.E58C.1F00 0000.0000.0004 F-B-LR 0 100 3 1 00:25:45
FFFF.FFFF.FFFF 001E.E58C.1F00 0000.0000.0004 F-B-LR 0 200 3 1 00:25:45
FFFF.FFFF.FFFF 001E.F675.CE00 0000.0000.0003 FA--LR 0 100 1 2 00:25:45
FFFF.FFFF.FFFF 001E.F675.CE00 0000.0000.0003 FA--LR 0 200 1 2 00:25:45

```

```

R9#show otv isis protocol | include System
System Id: 001E.E58C.1F00.00 IS-Type: level-1
System Id: 001E.E58C.1F00.00 IS-Type: level-1

```

For even more detail, we can look at the raw IS-IS LSPs and see the TLVs carrying the MAC routes. From CSR4's perspective, we will examine CSR9's LSP for unicast routing. CSR4 is the AED for both VLANs, which is made clear in the LSP. The "ED" (end device) stanza lists the two nodes, CSR4 and CSR9, by their node ID. CSR9 is not the AED (AED: 0) but is the backup AED (Back-up AED: 1) for these VLANs. Like Segment Routing labels, contiguous VLANs can be grouped with step sizes to keep the LSPs smaller. The final AED stanza shows that CSR4 is the AED and CSR9 is the backup AED, once again.

```

R4#show otv isis hostname
Level  System ID      Dynamic Hostname  (Overlay1)
1       001E.BD8F.6000 R6
1       001E.F675.CE00 R3
* 001E.4977.7C00 R4
1       001E.E58C.1F00 R9
1       001E.14FF.9B00 R5

```

```

R4#show otv isis database R9.00-00 verbose
Tag Overlay1:
IS-IS Level-1 LSP R9.00-00
LSPID          LSP Seq Num  LSP Checksum  LSP Holdtime  ATT/P/OL
R9.00-00       0x00000022  0xFEB6       631           0/0/0
Area Address: 00
NLPID:        0xCC 0x8E
Hostname: R9
Metric: 10     IS-Extended R3.01
ED Summary:
Site ID: 0000.0000.0004
AED-Server-ID: 001E.E58C.1F00
Version: 22
Device ID: 001E.E58C.1F00, fwd_ready: 1
Device ID: 001E.4977.7C00, fwd_ready: 1
VLAN Status:
Site ID: 0000.0000.0004 Partition ID: FFFF.FFFF.FFFF
Device ID: 001E.E58C.1F00 Cluster-ID: 0
AED: 0 Back-up AED: 1 Fwd ready: 1 Priority 0
Delete: 0 Local: 1 Remote: 1 Range: 2 Version 3
Start-vlan:100 End-vlan: 100 Step: 1

```

```

      Start-vlan:200      End-vlan: 200      Step: 1
AED Server:
  Site ID: 0000.0000.0004 Partition ID: FFFF.FFFF.FFFF
  Device ID: 001E.E58C.1F00 Cluster-ID: 0
  Old-AED: 001E.E58C.1F00 New-AED: 001E.4977.7C00
  Old-Backup-AED: 0000.0000.0000 New-Backup-AED: 001E.E58C.1F00
  Delete: 0 Range: 2 Version 3
      Start-vlan:100      End-vlan: 100      Step: 1
      Start-vlan:200      End-vlan: 200      Step: 1

```

When we looked at VPLS, we ignored the AED column in the BD output when we looked at the MAC tables there. This time, the bit actually means something since the AED relates to OTV. We can see CSR5 is the AED for everything in VLAN 100, and can learn the local MACs via the EFPs. CSR6, on the other hand, marks AED for the broadcast address and the IS-IS routes, but not the local EFP ones. So technically, CSR6 does learn those MAC addresses, but the lack of the AED bit set means they cannot exist in the OTV routing table. CSR1's for example, has a MAC address ending in 1AAA. Only CSR5 can forward traffic for CSR1.

```

R5#show bridge-domain 100
Bridge-domain 100 (2 ports in all)
State: UP                               Mac learning: Enabled
Aging-Timer: 1800 second(s)
  GigabitEthernet2 service instance 100
  Overlay1 service instance 100
  AED MAC address      Policy Tag      Age Pseudoport
  1  0050.56A9.1AAA forward dynamic_c 1777 GigabitEthernet2.EFP100
  1  FFFF.FFFF.FFFF flood static 0 OLIST_PTR:0xe80ef400
  1  0050.56A9.BE8A forward static_r 0 OCE_PTR:0xe8366040
  1  0050.56A9.EA77 forward static_r 0 OCE_PTR:0xe8366080
  1  001E.BD8F.60BC forward dynamic_c 1788 GigabitEthernet2.EFP100

```

```

R6#show bridge-domain 100
Bridge-domain 100 (2 ports in all)
State: UP                               Mac learning: Enabled
Aging-Timer: 1800 second(s)
  GigabitEthernet2 service instance 100
  Overlay1 service instance 100
  AED MAC address      Policy Tag      Age Pseudoport
  0  001E.14FF.9BBC forward dynamic 1786 GigabitEthernet2.EFP100
  0  0050.56A9.1AAA forward dynamic 1756 GigabitEthernet2.EFP100
  1  FFFF.FFFF.FFFF flood static 0 OLIST_PTR:0xe80efc00
  0  0050.56A9.BE8A forward dynamic 1657 GigabitEthernet2.EFP100
  1  0050.56A9.BE8A forward static_r 0 OCE_PTR:0xe8367420
  0  0050.56A9.EA77 forward dynamic 1636 GigabitEthernet2.EFP100
  1  0050.56A9.EA77 forward static_r 0 OCE_PTR:0xe8367440

```

It is very rare that one would need to troubleshooting this level of detail with OTV. Below are some additional summary commands to help determine if OTV is working at a glance. Looking at the OTV VLANs, routes, and adjacencies are common and have already been examined. These additional commands show the state of the interfaces, multicast groups, and other basic information. Future versions of OTV will support UDP encapsulation as the current model is GRE. The “Fwd-capable” and “Fwd-ready” fields must be “Yes” or else OTV cannot work. Ensure the site-BD is up and the EFP is configured properly or else OTV cannot start. This is true even at single-homed sites.

```
R3#show otv
Overlay Interface Overlay1
  VPN name           : None
  VPN ID             : 5
  State              : UP
  Fwd-capable        : Yes
  Fwd-ready          : Yes
  AED-Server         : Yes
  Backup AED-Server  : No
  AED Capable        : Yes
  IPv4 control group : 239.0.0.1
  Mcast data group range(s): 239.0.1.0/24
  Join interface(s)  : GigabitEthernet2.532
  Join IPv4 address   : 10.3.12.3
  Tunnel interface(s) : Tunnel0
  Encapsulation format : GRE/IPv4
  Site Bridge-Domain : 3
  Capability          : Multicast-reachable
  Is Adjacency Server : No
  Adj Server Configured : No
  Prim/Sec Adj Svr(s) : None
```

```
R3#show otv summary
OTV Configuration Information, Site Bridge-Domain: 3
Overlay VPN Name      Control Group    Data Group(s)      Join Interface
State
1      None           239.0.0.1         239.0.1.0/24      Gi2.532          UP
Total Overlay(s): 1
```

You can also name an OTV instance with a VPN name to reference it later. The output is the same as “show vpn” but can be made specific when there are multiple overlays.

```
! CSR3
interface Overlay1
  otv vpn-name OTV_VPN

R3#show otv vpn OTV_VPN
Overlay Interface Overlay1
  VPN name           : OTV_VPN
```

```
VPN ID           : 5
State            : UP
[snip]
```

There is a Tunnel0 shown here despite us never configuring that. We can see the configuration of it and notice that it is a mGRE tunnel. The tunnel endpoints include all of the other sites, plus the multicast group used for OTV signaling. I assume this is how OTV is able to “send” multicast traffic without having to implement draft-Rosen under the hood.

```
R3#show derived-config interface tunnel0
interface Tunnel0
  ip unnumbered GigabitEthernet2.532
  no ip redirects
  tunnel source 10.3.12.3
  tunnel mode gre multipoint
```

```
R3#show tunnel endpoints tunnel 0
Tunnel0 running in multi-GRE/IP mode
Endpoint transport 10.4.13.4 Refcount 3 Base 0x7F59A9960710 Create Time
01:13:01
  overlay 10.4.13.4 Refcount 2 Parent 0x7F59A9960710 Create Time 01:13:01
Endpoint transport 10.5.14.5 Refcount 3 Base 0x7F59A9960A30 Create Time
00:38:50
  overlay 10.5.14.5 Refcount 2 Parent 0x7F59A9960A30 Create Time 00:38:50
Endpoint transport 10.6.10.6 Refcount 3 Base 0x7F59A9960BC0 Create Time
00:22:03
  overlay 10.6.10.6 Refcount 2 Parent 0x7F59A9960BC0 Create Time 00:22:03
Endpoint transport 10.9.13.9 Refcount 3 Base 0x7F59A99608A0 Create Time
01:19:55
  overlay 10.9.13.9 Refcount 2 Parent 0x7F59A99608A0 Create Time 01:19:55
Endpoint transport 239.0.0.1 Refcount 3 Base 0x7F59A9960D50 Create Time
23:41:31
  overlay 239.0.0.1 Refcount 2 Parent 0x7F59A9960D50 Create Time 23:41:31
```

The question then arises as to how OTV is able to use a single overlay to carry multiple VLANs. My first guess was to use the 4-byte GRE tunnel key as a multiplexer to encode the VLAN information, but the tunnel key feature is not enabled.

```
R3#show interface tunnel0 | section Tunnel proto
Tunnel protocol/transport multi-GRE/IP
  Key disabled, sequencing disabled
  Checksumming of packets disabled
```

The secret is using an MPLS label as a shim. This is the real reason why MPLS cannot be configured on an OTV router, for the “real” MPLS process would be confused. Reusing MPLS encapsulation was a very smart move so that a new shim did not need to be invented. We can see the encapsulation on CSR5 by

capturing outbound as CSR1 sends packets across VLAN 100 and VLAN 200 to CSR7. The first 18 bytes of both packets is identical and represents the destination and source MAC addresses of the frame (CSR5 to XRv4) inside VLAN 3554 (yellow). The next 20 bytes represents an outer IP header using the GRE IP addressing made by the dynamic tunnels (10.5.14.5 to 10.3.12.3) in green. The next 4 bytes is the GRE header which has no fancy options enable, but specifies the tunneled protocol as MPLS multicast in cyan (it obviously isn't multicast, though). The MPLS label values for the two packets are 132 and 232, respectively, corresponding to VLANs 100 and 200 (pink). I assume that the VLAN-to-label mapping just arithmetically adds 32 to the VLAN ID to derive a label value; likely to steer far away from the reserved MPLS labels (0 to 15). Inside MPLS, we have the original Ethernet frame with no dot1q tags as OTV has no reason to carry it, since the VLAN was signaled inside the MPLS label. This puts the VLAN closer to the "front" of the packet rather than making the router look into the GRE payload to find the first dot1q tag. The tunneled IP source/destination addresses evaluate to 10.0.100.1 to 10.0.100.7 for VLAN 100, and 10.0.200.1 to 10.0.200.7 for VLAN 200. In summary, OTV is using 42 bytes of encapsulation: 20 for IP, 4 for GRE, 4 for the MPLS shim, and 14 for the inner Ethernet header. This doesn't account for the outer-most layer 2 encapsulation which may vary. Of course, the MPLS PEs will impose more MPLS labels onto this traffic since it looks like regular IP unicast traffic, so MTU calculation is important.

```
R5#show monitor capture CAP buffer dump
```

```
0
```

```
0000: 005056A9 862A0050 56A9DC63 81000DE2 .PV...*.PV..c....
0010: 08004500 008E0038 4000FF2F 4CF90A05 ..E....8@.../L...
0020: 0E050A03 0C030000 88480008 41FF0050 .....H...A..P
0030: 56A9EA77 005056A9 1AAA0800 45000064 V..w.PV.....E..d
0040: 01300000 FF01DE60 0A006401 0A006407 .0.....`.d...d.
0050: 08009D49 00330000 00000000 1983C74A ...I.3.....J
0060: ABCDABCD ABCDABCD ABCDABCD ABCDABCD .....
```

```
6
```

```
0000: 005056A9 862A0050 56A9DC63 81000DE2 .PV...*.PV..c....
0010: 08004500 008E003E 4000FF2F 4CF30A05 ..E....>@.../L...
0020: 0E050A03 0C030000 8848000E 81FF0050 .....H.....P
0030: 56A9EA77 005056A9 1AAA0800 45000064 V..w.PV.....E..d
0040: 01360000 FF01165A 0A00C801 0A00C807 .6.....Z.....
0050: 08009983 00340001 00000000 1983CB0E .....4.....
0060: ABCDABCD ABCDABCD ABCDABCD ABCDABCD .....
```

Multicast over OTV is similar. Like draft-Rosen, it will encapsulate multicast into GRE but maintain a multicast destination using an address from its range of data-groups. We generate some bogus multicast traffic on CSR1 in both VLANs to test it. Right away, we see it doesn't work, as OTV is a smart multicast router. It has a simplified display for the MRIB, but it reads like the normal IOS MRIB. We can see that OTV is aware of a source on each VLAN, but no receivers, so it does not deliver the traffic anywhere. With VPLS, every node would have received those pings despite showing no interest.

```
R1#ping 225.1.2.3
```

```

R5#show otv mroute vlan 100 source 10.0.100.1 group 225.1.2.3
OTV Multicast Routing Table for Overlay1
Bridge-Domain = 100, s = 10.0.100.1, g = 225.1.2.3
Incoming interface list:
  Service Instance 100, GigabitEthernet2, 0050.56a9.1aaa
Incoming interface count = 1, Outgoing interface count = 0

R5#show otv mroute vlan 200 source 10.0.200.1 group 225.1.2.3
OTV Multicast Routing Table for Overlay1
Bridge-Domain = 200, s = 10.0.200.1, g = 225.1.2.3
Incoming interface list:
  Service Instance 200, GigabitEthernet2, 0050.56a9.1aaa
Incoming interface count = 1, Outgoing interface count = 0

```

We configure CSR7 to listen to this group on both VLAN 100 and 200. CSR5 learns this through OTV control messaging (IGMP snooping style). We can also verify the MRIB by VLAN or BD. Both are shown below.

```

R5#show otv mroute bridge-domain 100
[snip]
Bridge-Domain = 100, s = *, g = 225.1.2.3
Outgoing interface list:
  Overlay1, R3
Incoming interface count = 0, Outgoing interface count = 1

R5#show otv mroute vlan 200
[snip]
Bridge-Domain = 200, s = *, g = 225.1.2.3
Outgoing interface list:
  Overlay1, R3
Incoming interface count = 0, Outgoing interface count = 1

```

Because these are production flows, we expect that some multicast data-groups were allocated to support it. We can verify this within OTV as well. The output is very clean and shows the inner (active) source and destination followed by the tunnel (delivery) source and destination.

```

R5#show otv data-group
No remote data-group mappings
Flags:  D - Local active source dynamically detected
        S - Local active source statically configured
        J - Data group has been joined in the core
        U - Data group has not been joined in the core

```

```

Local Active Sources for Overlay1

```

BD	Active-Source	Active-Group	Delivery-Source	Delivery-Group	Flags
100	10.0.100.1	225.1.2.3	10.5.14.5	239.0.1.0	D
200	10.0.200.1	225.1.2.3	10.5.14.5	239.0.1.1	D

Below are two packets captured on egress from CSR5 for additional verification. For consistency, the same fields are highlighted the same colors as they were for the unicast flow. The outermost layer 2 header is yellow, the outer IP header is green, the GRE header is cyan and the MPLS shim header is cyan. All of this is the same as it was before with the exception of the destination GRE address. Two new data-groups have been spawned to support these multicast flows: 239.0.1.0 for VLAN 100 and 239.0.1.1 for VLAN 200.

R5#show monitor capture CAP buffer dump

```

3
0000: 01005E00 01000050 56A9DC63 81000DE2 ..^....PV..c....
0010: 08004500 008E008D 4000FF2F 72A90A05 ..E.....@.../r...
0020: 0E05EF00 01000000 88480008 41FF0100 .....H..A...
0030: 5E010203 005056A9 1AAA0800 45000064 ^....PV.....E..d
0040: 01620000 FF016931 0A006401 E1010203 .b....i1..d.....
0050: 0800C77F 00380007 00000000 199E9CED .....8.....
0060: ABCDABCD ABCDABCD ABCDABCD ABCDABCD .....

4
0000: 01005E00 01010050 56A9DC63 81000DE2 ..^....PV..c....
0010: 08004500 008E008E 4000FF2F 72A70A05 ..E.....@.../r...
0020: 0E05EF00 01010000 8848000E 81FF0100 .....H.....
0030: 5E010203 005056A9 1AAA0800 45000064 ^....PV.....E..d
0040: 01620000 FF010531 0A00C801 E1010203 .b.....1.....
0050: 0800C77F 00380007 00000000 199E9CED .....8.....
0060: ABCDABCD ABCDABCD ABCDABCD ABCDABCD .....

```

What is very interesting is that on CSR3, it sees data-groups allocated by CSR6 also. Although not an AED, CSR6 still allocates groups because it gives the remote routers time to join them. Just like all the OTV routers joined the control-group, CSR3 dynamically issue joins for OTV data-groups as it learns them via IS-IS.

R3#show otv data-group

```

Flags:  D - Local active source dynamically detected
        S - Local active source statically configured
        J - Data group has been joined in the core
        U - Data group has not been joined in the core

```

Remote Active Sources for Overlay1

BD	Active-Source	Active-Group	Delivery-Source	Delivery-Group	Flags
100	10.0.100.1	225.1.2.3	10.5.14.5	239.0.1.0	J
100	10.0.100.1	225.1.2.3	10.6.10.6	239.0.1.0	J
200	10.0.200.1	225.1.2.3	10.5.14.5	239.0.1.1	J
200	10.0.200.1	225.1.2.3	10.6.10.6	239.0.1.1	J

R3#show otv data-group detail

```

Flags:  D - Local active source dynamically detected
        S - Local active source statically configured
        J - Data group has been joined in the core

```

U - Data group has not been joined in the core

Remote Active Sources for Overlay1

BD	Active-Source	Active-Group	Delivery-Source	Delivery-Group	Flags	Listeners-Flags
100	10.0.100.1	225.1.2.3	10.5.14.5	239.0.1.0	J	(*,G)
100	10.0.100.1	225.1.2.3	10.6.10.6	239.0.1.0	J	(*,G)
200	10.0.200.1	225.1.2.3	10.5.14.5	239.0.1.1	J	(*,G)
200	10.0.200.1	225.1.2.3	10.6.10.6	239.0.1.1	J	(*,G)

If we check the group membership, we can see IGMPv3 INCLUDE joins are sent. This would work great for SSM, but ASM is also supported and is what we are currently using. It is good to know that this behavior is automatic, provided that IGMPv3 is enabled on the PE-CE link and OTV is configured properly. If the SP network suddenly decided that 239.0.0.0/8 was the SSM range, OTV would still work.

R3#show ip igmp groups 239.0.1.0 detail

Flags: L - Local, U - User, SG - Static Group, VG - Virtual Group,
SS - Static Source, VS - Virtual Source,
Ac - Group accounted towards access control limit

Interface: GigabitEthernet2.532

Group: 239.0.1.0

Flags:

Uptime: 00:03:43

Group mode: INCLUDE

Last reporter: 10.3.12.3

Group source list: (C - Cisco Src Report, U - URD, R - Remote, S - Static,
V - Virtual, M - SSM Mapping, L - Local,
Ac - Channel accounted towards access control limit)

Source Address	Uptime	v3 Exp	CSR Exp	Fwd	Flags
10.5.14.5	00:03:43	00:02:38	stopped	Yes	R
10.6.10.6	00:03:43	00:02:38	stopped	Yes	R

Interface: Tunnel0

Group: 239.0.1.0

Flags: VS

Uptime: 00:03:43

Group mode: INCLUDE

Last reporter: 10.3.12.3

Group source list: (C - Cisco Src Report, U - URD, R - Remote, S - Static,
V - Virtual, M - SSM Mapping, L - Local,
Ac - Channel accounted towards access control limit)

Source Address	Uptime	v3 Exp	CSR Exp	Fwd	Flags
10.5.14.5	00:03:43	00:02:38	stopped	Yes	RV
10.6.10.6	00:03:43	00:02:38	stopped	Yes	RV

The result of this join means that XRv2, as the RP, will be able to process register messages for this C(S,G) within the core. CSR5 as a source actually has a valid (S,G) entry with RPF towards the ingress PE (XRv4) and an OIL including the PE-CE link to CSR3. The second (S,G) relates to CSR6; even though the state is (S,G), a register was never received, and XR is just recording it this way due to the IGMPv3 join,

even if the group is not SSM. The “RR” flag means register received and this does not apply to CSR6, since no traffic was sourced from that node. The OIL is populated via the IGMPv3 join but nothing more. Again, this is OTV doing as much as possible to support HA for the overlay without wasting bandwidth.

```
RP/0/0/CPU0:XRv2#show pim vrf OTV topology 239.0.1.0
[snip]
(10.5.14.5,239.0.1.0)SPT SM Up: 00:05:14
JP: Join(00:00:29) RPF: mdtOTV,2.14.14.14 Flags: KAT(00:02:19) RA RR
(00:03:22)
    mdtOTV                                00:00:18   fwd AS(2.14.14.14,00:02:59)
    GigabitEthernet0/0/0/0.532           00:05:14   fwd LI LH

(10.6.10.6,239.0.1.0)SPT SM Up: 00:05:14
JP: Join(00:00:36) RPF: mdtOTV,2.10.10.10 Flags:
    GigabitEthernet0/0/0/0.532           00:05:14   fwd LI LH
```

OTV also introduces multicast LSPs within IS-IS to carry this information. We can also see the raw information in the IS-IS LSPDB. From CSR6, we check its local “mgroup” LSP. This information carries the VLAN, source, group, and some other data. Sources of 0.0.0.0 represent (*,G) state information; in this case, we see the AutoRP mapping-agent (*,G) state. Of greater importance are the (S,G) states for the VLAN 100 and VLAN 200 sources to group 225.1.2.3. The LSP also contains the “ds” and “dg”, or delivery-source and delivery-group, which is the post-encapsulation addresses. It would be the MDT addressing within the context of MVPN profile 0, as an example. This LSP is how CSR6 signals to CSR3 that it is also an “Active source” for this (S,G).

```
R6#show otv isis database mgroup R6.00-00 verbose
Tag Overlay1:
IS-IS Level-1 mgroup LSP R6.00-00
LSPID                LSP Seq Num  LSP Checksum  LSP Holdtime  ATT/P/OL
R6.00-00              * 0x00000017  0x799C        789           0/0/0
  Layer 2 Multicast Group
    Group IP Address: topoid 0, vlan 100, # of records 1
      Group address: 224.0.1.40, # of sources 1
        0.0.0.0
    Group IP Address: topoid 0, vlan 200, # of records 1
      Group address: 224.0.1.40, # of sources 1
        0.0.0.0
  Layer 2 Multicast Group Membership Active Source
    Group IP Address: vlan 100, ds 10.6.10.6, dg 239.0.1.0, # of records 1
      Group address: 225.1.2.3, # of sources 1
        10.0.100.1
    Group IP Address: vlan 200, ds 10.6.10.6, dg 239.0.1.1, # of records 1
      Group address: 225.1.2.3, # of sources 1
        10.0.200.1
```

CSR6 is also aware that CSR3 has receivers for group 225.1.2.3 in the first place. Otherwise, OTV will not send traffic into the overlay if there are no receivers, so the signaling works both ways. Like BGP c-multicast signaling in MVPN, OTV uses IS-IS for c-multicast signaling.

```
R6#show otv isis database mgroup R3.00-00 verbose
Tag Overlay1:
IS-IS Level-1 mgroup LSP R3.00-00
LSPID                LSP Seq Num  LSP Checksum  LSP Holdtime  ATT/P/OL
R3.00-00              0x00000017  0xBAC7        506           0/0/0

  Layer 2 Multicast Group
    Group IP Address: topoid 0, vlan 100, # of records 2
      Group address: 224.0.1.40, # of sources 1
        0.0.0.0
      Group address: 225.1.2.3, # of sources 1
        0.0.0.0
    Group IP Address: topoid 0, vlan 200, # of records 2
      Group address: 224.0.1.40, # of sources 1
        0.0.0.0
      Group address: 225.1.2.3, # of sources 1
        0.0.0.0
```

Having CSR6 signal the groups is good, but having it duplicate traffic is not. The control-plane seems to indicate traffic is not being duplicated. We can verify that only CSR5 is actually sending traffic into the core for additional certainty. CSR6 just shows some control traffic while CSR5 shows the data packets, which we analyzed earlier. This proves that the AED rule still applies, but for HA purposes, CSR6 can tell the remote node which data-groups to expect. Even if CSR3 is not receiving the traffic, at least we would know that the sending side functions properly. CSR6 is not sending that traffic since it is not the AED, which is ideal.

```
R5#show monitor capture CAP buffer brief
```

#	size	timestamp	source	destination	protocol
0	160	0.000000	10.5.14.5	-> 239.0.1.0	GRE
1	160	0.000992	10.5.14.5	-> 239.0.1.1	GRE
2	1460	0.526966	10.5.14.5	-> 239.0.0.1	GRE
3	160	2.004989	10.5.14.5	-> 239.0.1.0	GRE
4	160	2.004989	10.5.14.5	-> 239.0.1.1	GRE
5	160	4.006988	10.5.14.5	-> 239.0.1.0	GRE
6	160	4.007995	10.5.14.5	-> 239.0.1.1	GRE
7	160	6.008987	10.5.14.5	-> 239.0.1.0	GRE
8	160	6.008987	10.5.14.5	-> 239.0.1.1	GRE

```
R6#show monitor capture CAP buffer brief
```

#	size	timestamp	source	destination	protocol

0	188	0.000000	10.6.10.6	->	239.0.0.1	GRE
1	54	6.600027	10.6.10.6	->	224.0.0.1	IGMP
2	82	7.138045	10.6.10.6	->	224.0.0.22	IGMP
3	188	7.944995	10.6.10.6	->	239.0.0.1	GRE
4	188	17.173047	10.6.10.6	->	239.0.0.1	GRE

A final verification shows that traffic is being delivered to CSR3 only from CSR5, while the control traffic comes from all OTV routers in the overlay (including CSR6).

R3#show monitor capture CAP buffer brief

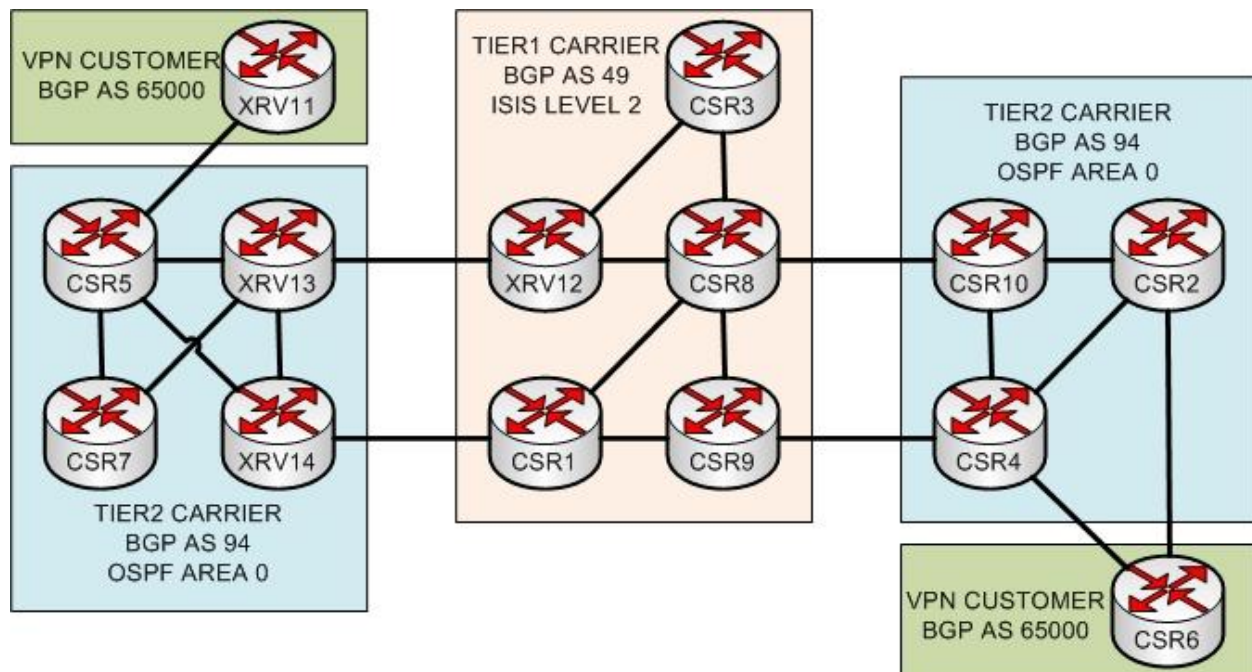
#	size	timestamp	source		destination	protocol
0	194	0.000000	10.9.13.9	->	239.0.0.1	GRE
1	160	1.124994	10.5.14.5	->	239.0.1.0	GRE
2	160	1.124994	10.5.14.5	->	239.0.1.1	GRE
3	188	1.499975	10.6.10.6	->	239.0.0.1	GRE
4	160	3.124994	10.5.14.5	->	239.0.1.0	GRE
5	160	3.126993	10.5.14.5	->	239.0.1.1	GRE

Additional Reading – Reference configurations "otv-mc"

33. Describe, implement, and troubleshoot Unified MPLS and CSC

33.1 Carrier Supporting Carrier (CSC)

Carrier Supporting Carrier (CSC) or Carrier's Carrier is another L3VPN variation. Similar to UMPLS, this relies on BGP labeled-unicast to provide an additional shim for transparent between a distinct "core" network and disparate "aggregation" networks. Unlike UMPLS, however, this is designed for multiple service provider interaction. If a tier 2 ISP (one without an infrastructure spanning end-to-end) requires transport between customers at remote sites, it can rely on a tier 1 ISP (one that does). Smaller ISPs may use this technique by essentially buying transit bandwidth from a tier 1 ISP and running their own MPLS services independent from the provider. The provider simply provides ordinary L3VPN transport for the tier 2 ISP, with the addition of BGP labeled-unicast in the customer VRF for additional label exchange. Like UMPLS, this supports both L3VPN and L2VPN. Only MVPN profile 0 is supported, and for variety, this design uses PIM SSM with that profile. This MVPN model works on XRv which is convenient as well. The diagram is shown below which shows the tier 1 and tier 2 ISP networks.



In terms of inter-AS transit links, these should be viewed just like normal L3VPN PE-CE customer links. All of the same concepts apply, such as BGP AS-override. At the 4 touch points, 3 of them use BGP + Label (XRV12, CSR8, and CSR9) and one of them uses OSPF+LDP (CSR1). BGP is by far the more popular and better choice in this kind of design, but I demonstrate IGP+LDP as well, which makes things more complex. First, we will examine the basic configuration in the core carrier, which is BGP AS 49 running IS-IS with MPLS-TE supported. The most basic L3VPN component configurations are not shown here. A quick check on CSR1 shows 5 routers in level-2, which is correct given the diagram.

```
CSR1#show isis database level-2
Tag 49:
IS-IS Level-2 Link State Database:
LSPID                LSP Seq Num  LSP Checksum  LSP Holdtime  ATT/P/OL
CSR1.00-00            * 0x000000DB  0x5D72        1189          0/0/0
CSR3.00-00            0x000000DC  0xC00A        588           0/0/0
CSR8.00-00            0x000000DF  0x821E        1060          0/0/0
CSR9.00-00            0x000000DA  0x2BCD        448           0/0/0
XRV12.00-00           0x000000E1  0xFB2C        386           0/0/0
```

We can see the affect of some link cost changes on CSR8 by checking the IS-IS topology; traffic from CSR8 to XRV12 prefers CSR3 and to CSR9 prefers CSR1. This is to have more “hops” in the core carrier for additional feature testing (TE, etc).

```
CSR8#show isis topology level-2
Tag 49:
IS-IS TID 0 paths to level-2 routers
System Id            Metric      Next-Hop  Interface  SNPA
```


CSR1	10	CSR1	Gi2.518	000c.29fb.a339
CSR3	10	CSR3	Gi2.538	000c.29d7.81fe
CSR8	--			
CSR9	20	CSR1	Gi2.518	000c.29fb.a339
XRv12	20	CSR3	Gi2.538	000c.29d7.81fe

The BGP topology is simple as XRv12 is a RR for the VPNv4 and IPv4 MDT AFIs within the core carrier's network. A single VPNv4 route (customer PE endpoint) and IPv4 MDT route (local PE endpoint) are learned from each peer.

```
RP/0/0/CPU0:XRv12#show bgp vpnv4 unicast summary | begin ^Neighbor
Neighbor      Spk    AS  MsgRcvd  MsgSent    TblVer  InQ  OutQ  Up/Down  St/PfxRcd
1.1.1.1        0    49    9383    8902        45    0    0  00:29:11      1
8.8.8.8        0    49    9366    8892        45    0    0  00:29:12      1
9.9.9.9        0    49     716     686        45    0    0  00:29:12      1
```

```
RP/0/0/CPU0:XRv12#show bgp ipv4 mdt summary | begin ^Neighbor
Neighbor      Spk    AS  MsgRcvd  MsgSent    TblVer  InQ  OutQ  Up/Down  St/PfxRcd
1.1.1.1        0    49    9393    8911        13    0    0  00:30:47      1
8.8.8.8        0    49    9376    8901        13    0    0  00:30:47      1
9.9.9.9        0    49     726     695        13    0    0  00:30:48      1
```

Looking at the VPNv4 routes, XRv12 can see the customer PE endpoints. It also learns an MDT endpoint for each of its local PEs as well. So far this looks like an ordinary L3VPN.

```
RP/0/0/CPU0:XRv12#show bgp vpnv4 unicast vrf T2 | begin Network
Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 49:12 (default for vrf T2)
*>i2.2.2.2/32      8.8.8.8          0    100      0 94 ?
*>i4.4.4.4/32      9.9.9.9          0    100      0 94 ?
*> 5.5.5.5/32      192.168.12.13    3          0 94 i
```

```
RP/0/0/CPU0:XRv12#show bgp ipv4 mdt | begin Network
Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 49:1
*>i1.1.1.1/96      1.1.1.1          0    100      0 ?
Route Distinguisher: 49:8
*>i8.8.8.8/96      8.8.8.8          0    100      0 ?
Route Distinguisher: 49:9
*>i9.9.9.9/96      9.9.9.9          0    100      0 ?
Route Distinguisher: 49:12
*> 12.12.12.12/96  0.0.0.0          0          0 i
```

The difference comes with the label exchange between core carrier PE (CSC-PE) and the customer carrier PE (CSC-CE). Looking at BGP, below are configurations on XRv12, XRv13, CSR8, and CSR10. Notice that AS-override is used on the CSC-PEs so that the CSC-CEs can accept routes from across the L3VPN originating in the same AS. XRv13 advertises CSR5's loopback via BGP locally, while CSR10 runs labeled-unicast directly with CSR2 and learns the route via BGP. This means that redistribution from BGP into

IGP on CSR10 is not required, since other routers on the customer network will learn the remote PEs via BGP. That is why XRv13 redistributes these prefixes from BGP into OSPF as an alternative; both techniques work. Personally, the BGP approach is more straightforward so that routes exchanged in BGP remain in BGP, which makes the CSC design more consistent with UMPLS.

```
! XRv12 (CSC-PE)
route-policy RPL_IN
  pass
end-policy

route-policy RPL_OUT
  pass
end-policy

router bgp 49
  vrf T2
    rd 49:12
    address-family ipv4 unicast
      allocate-label all

  neighbor 192.168.12.13
    remote-as 94
    address-family ipv4 labeled-unicast
      route-policy RPL_IN in
      route-policy RPL_OUT out
    as-override

! XRv13 (CSC-CE)
route-policy RPL_IN
  pass
end-policy

route-policy RPL_OUT
  pass
end-policy

router bgp 94
  address-family ipv4 unicast
    network 5.5.5.5/32
    allocate-label all

  neighbor 192.168.12.12
    remote-as 49
    address-family ipv4 labeled-unicast
      route-policy RPL_IN in
      route-policy RPL_OUT out

router ospf 94
```

```

redistribute bgp 94 route-policy RPL_IN

! CSR8 (CSC-PE)
router bgp 49
  address-family ipv4 vrf T2
    neighbor 192.168.8.10 remote-as 94
    neighbor 192.168.8.10 activate
    neighbor 192.168.8.10 as-override
    neighbor 192.168.8.10 send-label

! CSR10 (CSC-CE)
router bgp 94
  no bgp default ipv4-unicast
  neighbor 192.168.8.8 remote-as 49

  address-family ipv4
    neighbor 2.2.2.2 activate
    neighbor 2.2.2.2 next-hop-self
    neighbor 2.2.2.2 send-label
    neighbor 192.168.8.8 activate
    neighbor 192.168.8.8 as-override
    neighbor 192.168.8.8 send-label

```

We can also use IGP+LDP on one set of transit links instead of BGP. The configurations for XRv14 and CSR1 depict this. I apply a prefix list to only redistribute the proper routes from OSPF into BGP to prevent loops. For example, if CSR1 learned a route to CSR2's loopback via XRv14, this would be undesirable. I also adjust the AD for OSPF external routes on CSR1 within the VRF so that the iBGP routes learned from the core are preferred over OSPF external routes originated by XRv13. These enhancements are not SP-specific and core routing techniques. LDP is also enabled on the sub-interface, and this cannot be configured in conjunction with BGP. The two are mutually exclusive as a CSC-PE can only run one CSC protocol per customer interface.

```

! CSR1
interface GigabitEthernet2.514
  ip ospf network point-to-point
  ip ospf 94 area 0
  mpls ip

ip prefix-list PL_HOST_ROUTE seq 5 permit 5.5.5.5/32

route-map RM_OSPF_TO_BGP permit 10
  match ip address prefix-list PL_HOST_ROUTE

router ospf 94 vrf T2
  router-id 0.0.0.1
  prefix-suppression
  redistribute bgp 49 subnets
  distance ospf external 210

```

```
router bgp 49
 address-family ipv4 vrf T2
 redistribute ospf 94 route-map RM_OSPF_TO_BGP
```

Below are the errors you will see if you try to enable LDP to support CSC when BGP is already configured. I also show the opposite case of having LDP configured then trying to configure BGP. With this second case, you can technically run regular BGP but not labeled BGP. Notice that the basic neighbor commands are accepted, and the parser doesn't complain until we try to enable label exchange. In real life, there would probably never be a combination of BGP and LDP used as the CSC protocol for the same customer carrier.

```
CSR1(config-subif)#mpls ip
% Failed to set CSC protocol to LDP because BGP is already configured for VRF
T2
```

```
CSR1(config-router-af)#neighbor 192.168.1.14 remote-as 94
CSR1(config-router-af)#neighbor 192.168.1.14 activate
CSR1(config-router-af)#neighbor 192.168.1.14 send-label
% Failed to set CSC protocol to BGP because LDP is already configured for VRF
T2
```

Each OSPF island has been configured with MPLS TE as well. CSR4 shows 3 routers with 3 links each (2 transit links and one stub loopback). CSR5 shows 5 total routers, which includes the 4 within the customer network and the CSC-PE of CSR1. CSR1 has one transit link only, while the other routers have several additional links. Since prefix-suppression isn't supported for primary addresses with OSPFv2 on XR at this time, each transit link's stub network is included in the LSA1. This makes the LSA's appear "fatter" than they should be, but the output is otherwise correct.

```
CSR4#show ip ospf 94 database
      OSPF Router with ID (4.4.4.4) (Process ID 94)
      Router Link States (Area 0)

Link ID          ADV Router      Age           Seq#           Checksum Link count
2.2.2.2          2.2.2.2        78            0x80000036    0x005B2F  3
4.4.4.4          4.4.4.4        1319          0x80000033    0x009DD9  3
10.10.10.10     10.10.10.10    1826          0x80000034    0x00FE66  3
[snip]
```

```
CSR5#show ip ospf 94 database
      OSPF Router with ID (5.5.5.5) (Process ID 94)
      Router Link States (Area 0)

Link ID          ADV Router      Age           Seq#           Checksum Link count
0.0.0.1          0.0.0.1        1400          0x80000004    0x0017F5  1
5.5.5.5          5.5.5.5        1116          0x80000037    0x001568  4
```

7.7.7.7	7.7.7.7	1685	0x80000031	0x00A1BA	3
13.13.13.13	13.13.13.13	417	0x80000037	0x00E76A	7
14.14.14.14	14.14.14.14	1925	0x80000034	0x0037A0	7

[snip]

Looking at the routing on CSR2, it learns the route to 5.5.5.5/32 via BGP with a label from CSR10. This is because the ASBRs in the right side of the customer network are not redistributing the BGP routes into IGP, but are instead exchanging them via BGP. Because the route is BGP-learned, an LDP label is not used, so BGP must allocate a label for these prefixes for MPLS forwarding to work properly. This is similar to UMPLS, as discussed in the next section. We won't trace any LSPs quite yet; we see that the route to CSR10 is an IGP route, which means an additional LDP label will be pushed atop the label stack.

```
CSR2#show ip route 5.5.5.5
Routing entry for 5.5.5.5/32
  Known via "bgp 94", distance 200, metric 0
  Tag 49, type internal
  Last update from 10.10.10.10 00:53:20 ago
  Routing Descriptor Blocks:show bgp ip
    * 10.10.10.10, from 10.10.10.10, 00:53:20 ago
      Route metric is 0, traffic share count is 1
      AS Hops 1
      Route tag 49
      MPLS label: 10029
```

```
CSR2#show ip route 10.10.10.10
Routing entry for 10.10.10.10/32
  Known via "ospf 94", distance 110, metric 2, type intra area
  Last update from 94.2.10.10 on GigabitEthernet2.520, 1d02h ago
  Routing Descriptor Blocks:
    * 94.2.10.10, from 10.10.10.10, 1d02h ago, via GigabitEthernet2.520
      Route metric is 2, traffic share count is 1
```

On the other side of the customer network, CSR5 learns the route to 2.2.2.2/32 via OSPF as an external route. This is because CSR5 is not running IPv4 BGP since the ASBRs are redistributing this BGP-learned route into IGP. ECMP is enabled for this prefix since there are ECMP paths to the ASBRs. Similar to inter-AS MPLS option C, the ASBRs can either advertise the remote PE loopbacks via BGP labeled-unicast (right side) or BGP-to-IGP redistribution (left side).

```
CSR5#show ip route 2.2.2.2
Routing entry for 2.2.2.2/32
  Known via "ospf 94", distance 110, metric 1
  Tag 49, type extern 2, forward metric 2
  Last update from 94.5.14.14 on GigabitEthernet2.554, 00:50:42 ago
  Routing Descriptor Blocks:
    94.5.14.14, from 13.13.13.13, 00:50:42 ago, via GigabitEthernet2.554
      Route metric is 1, traffic share count is 1
      Route tag 49
```

```
* 94.5.7.7, from 13.13.13.13, 00:50:42 ago, via GigabitEthernet2.557
Route metric is 1, traffic share count is 1
Route tag 49
```

```
CSR5#show ip ospf border-routers
```

```
OSPF Router with ID (5.5.5.5) (Process ID 94)
```

```
Base Topology (MTID 0)
```

```
Internal Router Routing Table
```

```
Codes: i - Intra-area route, I - Inter-area route
```

```
i 0.0.0.1 [2] via 94.5.14.14, GigabitEthernet2.554, ABR/ASBR, Area 0, SPF 17
i 13.13.13.13 [2] via 94.5.14.14, GigabitEthernet2.554, ASBR, Area 0, SPF 17
i 13.13.13.13 [2] via 94.5.7.7, GigabitEthernet2.557, ASBR, Area 0, SPF 17
```

Last, we quickly verify the "service" BGP sessions (VPNv4, VPNv6, VPLS, etc) to service the customers. CSR2 is a route-reflector for those services and connects to CSR4 and CSR5. Since CSR4 does not run VPLS, that AF is not included in the route-reflection.

```
CSR2#show bgp vpnv4 unicast all summary | begin ^Neighbor
```

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
4.4.4.4	4	94	25462	25519	75	0	0	2d18h	2
5.5.5.5	4	94	14	30	75	0	0	00:00:11	2
192.168.102.6	4	65000	4732	4732	75	0	0	2d23h	1
192.168.202.6	4	65000	4722	4725	75	0	0	2d23h	1

```
CSR2#show bgp vpnv6 unicast all summary | begin ^Neighbor
```

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
4.4.4.4	4	94	25462	25519	269	0	0	2d18h	2
5.5.5.5	4	94	14	30	269	0	0	00:00:15	2
FD00:192:168:102::6	4	65000	4863	4731	269	0	0	2d23h	1
FD00:192:168:202::6	4	65000	4863	4724	269	0	0	2d23h	1

```
CSR2#show bgp l2vpn vpls all summary | begin ^Neighbor
```

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
5.5.5.5	4	94	15	31	19	0	0	00:00:20	1

CSR6 is a CE router which is multi-homed to two PE routers, CSR4 and CSR2. It uses local preference to select best paths for traffic sent towards the SP network. For simplicity, it prefers CSR2 for VRF U1 and CSR4 for VRF U2. This is true for both IPv4 and IPv6 AFIs. The other BGP configurations are omitted, but this path-selection modification is important.

```
! CSR6
route-map RM_LPREF_200 permit 10
set local-preference 200
```

```

router bgp 65000
  address-family ipv4 vrf U1
    neighbor 192.168.102.2 route-map RM_LPREF_200 in

  address-family ipv6 vrf U1
    neighbor FD00:192:168:102::2 route-map RM_LPREF_200 in

  address-family ipv4 vrf U2
    neighbor 192.168.204.4 route-map RM_LPREF_200 in

  address-family ipv6 vrf U2
    neighbor FD00:192:168:204::4 route-map RM_LPREF_200 in

```

33.1.1 L3VPN

Now, we can begin tracing some LSPs. We will begin with CSR5 inside VRF U1 using VPNv6. Looking at the VPNv6 route to CSR6, we see two BGP paths. One originates from CSR2 and is sent to CSR5 without having been reflected, while one originates from CSR4 and is reflected by CSR2. At a glance, it may appear that CSR5 selected CSR2 as the best-path due to having a shorter cluster list length. However, that is not true. It actually picked CSR2 because the BGP RID 2.2.2.2 is less than the originator ID 4.4.4.4. These two fields are directly compared first, and the cluster list length is only compared when the originator is the same. The label used is 2003.

```

CSR5#show bgp vpnv6 unicast vrf U2 ::2:6:6:6/128
BGP routing table entry for [49:205]::2:6:6:6/128, version 259
Paths: (2 available, best #2, table U2)
  Advertised to update-groups:
    3
  Refresh Epoch 2
  65000, imported path from [49:204]::2:6:6:6/128 (global)
    ::FFFF:4.4.4.4 (metric 1) (via default) from 2.2.2.2 (2.2.2.2)
    Origin incomplete, metric 0, localpref 100, valid, internal
    Extended Community: RT:49:204
    Originator: 4.4.4.4, Cluster list: 2.2.2.2
    Connector Attribute: count=1
      type 1 len 12 value 49:204:4.4.4.4
    mpls labels in/out nolabel/4033
    rx pathid: 0, tx pathid: 0
  Refresh Epoch 2
  65000, imported path from [49:202]::2:6:6:6/128 (global)
    ::FFFF:2.2.2.2 (metric 1) (via default) from 2.2.2.2 (2.2.2.2)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    Extended Community: RT:49:202
    Connector Attribute: count=1
      type 1 len 12 value 49:202:2.2.2.2
    mpls labels in/out nolabel/2003
    rx pathid: 0, tx pathid: 0x0

```

Although not relevant for CSC particularly, this is an interesting BGP best-path topic. I temporarily adjust CSR4's BGP RID to a number less than 2.2.2.2 to prove that BGP works as I explained above. Now, with RID 1.4.4.4, CSR4 is the best path.

```
! CSR4
router bgp 94
  bgp router-id 1.4.4.4

CSR5#show bgp vpnv6 unicast vrf U2 ::2:6:6:6/128
BGP routing table entry for [49:205]::2:6:6:6/128, version 272
Paths: (2 available, best #1, table U2)
  Advertised to update-groups:
    3
  Refresh Epoch 2
  65000, imported path from [49:204]::2:6:6:6/128 (global)
    ::FFFF:4.4.4.4 (metric 1) (via default) from 2.2.2.2 (2.2.2.2)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:49:204
      Originator: 1.4.4.4, Cluster list: 2.2.2.2
      Connector Attribute: count=1
        type 1 len 12 value 49:204:4.4.4.4
      mpls labels in/out nlabel/4023
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 2
  65000, imported path from [49:202]::2:6:6:6/128 (global)
    ::FFFF:2.2.2.2 (metric 1) (via default) from 2.2.2.2 (2.2.2.2)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Extended Community: RT:49:202
      Connector Attribute: count=1
        type 1 len 12 value 49:202:2.2.2.2
      mpls labels in/out nlabel/2003
      rx pathid: 0, tx pathid: 0
```

Before continuing, I remove this BGP RID modification from CSR4 (not shown) so CSR2 is the best path again. We proved earlier that this uses label 2003 as allocated by BGP VPNv6. The next-hop is 2.2.2.2, and we find a pair of OSPF routes to this. This implies we will use the LDP labels from CSR7 and XRv14 in load-sharing fashion. The LIB reveals labels 7003 and 94005, both of which are properly programmed into the FIB.

```
CSR5#show mpls ldp bindings 2.2.2.2 32
lib entry: 2.2.2.2/32, rev 147
  local binding:  label: 5007
  remote binding: lsr: 7.7.7.7:0, label: 7003
  remote binding: lsr: 13.13.13.13:0, label: 93005
  remote binding: lsr: 14.14.14.14:0, label: 94005
```



```

CSR5#show ip cef 2.2.2.2
2.2.2.2/32
  nexthop 94.5.7.7 GigabitEthernet2.557 label 7003
  nexthop 94.5.14.14 GigabitEthernet2.554 label 94005

```

We quickly query CEF to see which way the traffic will actually go from CSR5 to CSR2. CEF reports that traffic is being routed to XRv14, so we will continue our verification there. The label stack becomes {94005 2003}.

```

CSR5#show ip cef exact-route virtual 5.5.5.5 2.2.2.2
5.5.5.5 -> 2.2.2.2 => label 94005TAG adj out of GigabitEthernet2.554, addr
94.5.14.14

```

XRv14 is faced with another ECMP decision. The LFIB directs traffic towards CSR1 and XRv3 equally since the RIB shows ECMP paths for 2.2.2.2/32. Since both routes are IGP routes, we know the labels were allocated via LDP, so XRv14 is just a P router performing a label swap in either case.

```

RP/0/0/CPU0:XRv14#show mpls forwarding labels 94005

```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94005	1028	2.2.2.2/32	Gi0/0/0/0.514	192.168.1.1	172
	93005	2.2.2.2/32	Gi0/0/0/0.534	94.13.14.13	3932587

```

RP/0/0/CPU0:XRv14#show route ipv4 2.2.2.2
Routing entry for 2.2.2.2/32
  Known via "ospf 94", distance 110, metric 1
  Tag 3489660977, type extern 2
  Routing Descriptor Blocks
    192.168.1.1, from 0.0.0.1, via GigabitEthernet0/0/0/0.514
      Route metric is 1
    94.13.14.13, from 13.13.13.13, via GigabitEthernet0/0/0/0.534
      Route metric is 1

```

We can query the LFIB for the exact route based on the source/destination pair between CSR5 and CSR2. Traffic is being sent to XRv13 and the label stack becomes {93005 2003}.

```

RP/0/0/CPU0:XRv14#show mpls forwarding exact-route label 94005 ipv4 5.5.5.5
2.2.2.2

```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94005	93005	2.2.2.2/32	Gi0/0/0/0.534	94.13.14.13	N/A

```

  Via: Gi0/0/0/0.534, Next Hop: 94.13.14.13
  MAC/Encaps: 18/22, MTU: 1500
  Label Stack (Top -> Bottom): { 93005 }
  NHID: 0

```

When XRv13 receives the MPLS packets, it consults its LFIB and performs a swap operation. Looking at the RIB details, we can reveal that the route is BGP learned and see its corresponding label. The LFIB smartly connects the LDP LSP to the BGP LSP by swapping an LDP label with a BGP label. The label stack becomes {92004 2003} and traffic is forwarded into the core carrier PE, XRv12.

```
RP/0/0/CPU0:XRv13#show mpls forwarding labels 93005
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
93005	92004	2.2.2.2/32	Gi0/0/0/0.523	192.168.12.12	14402056

```
RP/0/0/CPU0:XRv13#show route ipv4 2.2.2.2 detail
```

```
Routing entry for 2.2.2.2/32
```

```
Known via "bgp 94", distance 20, metric 0, [ei]-bgp, labeled unicast (3107)
```

```
Tag 49, type external
```

```
Routing Descriptor Blocks
```

```
192.168.12.12, from 192.168.12.12, BGP external
```

```
Route metric is 0
```

```
Label: 0x16764 (92004)
```

```
Tunnel ID: None
```

```
Extended communities count: 0
```

```
NHID:0x0 (Ref:0)
```

```
[snip]
```

Since XRv12 connects to XRv13 inside a VRF, the label 92004 will be a VPNv4 label from the perspective of XRv12. Specifically, this is XRv12's local label for 2.2.2.2/32, which is just an ordinary L3VPN route. BGP indicates that the best-path is via CSR8 and uses label 8001. This implies that a swap operation will occur, changing the label stack to {8001 2003}.

```
RP/0/0/CPU0:XRv12#show bgp vpnv4 unicast vrf T2 2.2.2.2/32
```

```
BGP routing table entry for 2.2.2.2/32, Route Distinguisher: 49:12
```

```
Versions:
```

```
Process bRIB/RIB SendTblVer
```

```
Speaker 42 42
```

```
Local Label: 92004
```

```
Paths: (1 available, best #1)
```

```
Not advertised to any peer
```

```
Path #1: Received by speaker 0
```

```
Not advertised to any peer
```

```
94, (Received from a RR-client)
```

```
8.8.8.8 (metric 20) from 8.8.8.8 (8.8.8.8)
```

```
Received Label 8001
```

```
Origin incomplete, metric 0, localpref 100, valid, internal, best, group-best, import-candidate, imported
```

```
Received Path ID 0, Local Path ID 1, version 42
```

```
Extended community: RT:49:8
```

```
Connector: type: 1, Value:49:8:8.8.8.8
Source VRF: default, Source Route Distinguisher: 49:8
```

XRv12 may need to push additional labels depending on the route to 8.8.8.8, which is the BGP next-hop. The RIB indicates that the shortest path is via CSR3, which avoids the high-cost direct path. CSR3 allocates LDP label 3000, which will be pushed atop the label stack, resulting in {3000 8001 2003}. If we didn't modify the link cost, the direct path to CSR would have been chosen, which would have resulted in a label stack without 3000 due to having received implicit-null (which is less interesting).

```
RP/0/0/CPU0:XRv12#show route ipv4 8.8.8.8
Routing entry for 8.8.8.8/32
  Known via "isis 49", distance 115, metric 20, type level-2
  Routing Descriptor Blocks
    49.3.12.3, from 8.8.8.8, via GigabitEthernet0/0/0/0.532
    Route metric is 20
  No advertising protos.
```

```
RP/0/0/CPU0:XRv12#show mpls ldp bindings 8.8.8.8/32 neighbor 3.3.3.3
8.8.8.8/32, rev 134
  Local binding: label: 92003
  Remote bindings: (2 peers)
    Peer          Label
    -----
    3.3.3.3:0     3000
```

We can confirm this by checking the VRF-aware CEF table to see that the route to 2.2.2.2/32 includes the VPNv4 label (core carrier VPN label) and the transport label via LDP from CSR3. XRv12 is not aware of and does not care about the MPLS service label between CSR2 and CSR5.

```
RP/0/0/CPU0:XRv12#show cef vrf T2 ipv4 2.2.2.2/32
2.2.2.2/32, version 24, internal 0x1000001 0x0 (ptr 0xa13a7474) [1], 0x0
(0x0), 0x208 (0xa1476280)
Prefix Len 32, traffic index 0, precedence n/a, priority 3
via 8.8.8.8, 3 dependencies, recursive [flags 0x6000]
  path-idx 0 NHID 0x0 [0xa14dfaf4 0x0]
  recursion-via-/32
  next hop VRF - 'default', table - 0xe0000000
  next hop 8.8.8.8 via 92003/0/21
  local label 92004
  next hop 49.3.12.3/32 Gi0/0/0/0.532 labels imposed {3000 8001}
```

When CSR3 receives this packet with label 3000, it pops 3000 from the stack since CSR8 advertised implicit-null for 8.8.8.8/32 via LDP. The label stack becomes {8001 2003} and is forwarded to CSR8.

```
CSR3#show mpls forwarding-table labels 3000
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
-----
```

Label	Label	or Tunnel Id	Switched	interface	
3000	Pop Label	8.8.8.8/32	70866	Gi2.538	49.3.8.8

```

CSR3#show mpls ldp bindings 8.8.8.8 32 neighbor 8.8.8.8
  lib entry: 8.8.8.8/32, rev 8
    remote binding: lsr: 8.8.8.8:0, label: imp-null

```

When CSR8 receives the packet, it consults its LFIB and performs a label swap. Ignoring CSC for a moment, this is classic L3VPN behavior where the carrier VPN label of 8001 is exposed to the PE. The CSC enhancement shows the label swap, not pop, since the LSP continues towards the customer carrier. Normally the out-label is “no label” in an ordinary L3VPN design. The out-label is 10027 which is CSR10's local label for 2.2.2.2/32, allocated by BGP IPv4 labeled-unicast, which isn't in a VRF on CSR10. CSR8's VPNv4 entry shows the in/out labels for this prefix as seen by BGP. The label stack is now {10027 2003}.

```

CSR8#show mpls forwarding-table labels 8001
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id    Switched     interface
8001       10027      2.2.2.2/32 [V]  19374533     Gi2.580    192.168.8.10

```

```

CSR8#show bgp vpnv4 unicast vrf T2 2.2.2.2/32
BGP routing table entry for 49:8:2.2.2.2/32, version 13
Paths: (1 available, best #1, table T2)
  Advertised to update-groups:
    6
  Refresh Epoch 1
  94
    192.168.8.10 (via vrf T2) from 192.168.8.10 (10.10.10.10)
      Origin incomplete, localpref 100, valid, external, best
      Extended Community: RT:49:8
      mpls labels in/out 8001/10027
      rx pathid: 0, tx pathid: 0x0

```

When CSR10 receives the packet labeled 10027, it pops this label. The route to 2.2.2.2/32 is an OSPF route, which means an LDP label should be used. This happens to be implicit-null since CSR2 is the next-hop and is connected to 2.2.2.2/32. This is another example of how the MPLS process is smart enough to connect BGP labels to LDP labels to make a seamless LSP. The label stack is now {2003}.

```

CSR10#show mpls forwarding-table labels 10027
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id    Switched     interface
10027      Pop Label  2.2.2.2/32      25103370     Gi2.520    94.2.10.2

```

```

CSR10#show ip route 2.2.2.2
Routing entry for 2.2.2.2/32
  Known via "ospf 94", distance 110, metric 2, type intra area
  Last update from 94.2.10.2 on GigabitEthernet2.520, 3d21h ago
  Routing Descriptor Blocks:

```

```
* 94.2.10.2, from 2.2.2.2, 3d21h ago, via GigabitEthernet2.520
Route metric is 2, traffic share count is 1
```

```
CSR10#show mpls ldp bindings 2.2.2.2 32 neighbor 2.2.2.2
lib entry: 2.2.2.2/32, rev 190
remote binding: lsr: 2.2.2.2:0, label: imp-null
```

As the remote PE, CSR2 removes the label stack and delivers the traffic into the VRF. We can see the VPN route is in VF U2 and is for the customer prefix ::2:6:6:6/128. The out-label is “no label” since this is the end of the LSP. Using traceroute inside the VPN, we can see the full label stack for confirmation. The only discrepancy with this output is that CSR7 is the first MPLS hop, not XRv14. This is inconsistent with what CEF reported earlier, but rather than troubleshoot software-based ECMP mechanisms, we will ignore the minor discrepancy and continue.

```
CSR2#show mpls forwarding-table labels 2003 detail
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id Switched     interface
2003       No Label  ::2:6:6:6/128[V] 0          Gi2.202    FE80::6
MAC/Encaps=22/22, MRU=1504, Label Stack{}
000C2993FE00000C295CE1E981000DC6810000CA86DD
VPN route: U2
No output feature configured
```

```
RP/0/0/CPU0:XRv11#traceroute vrf U2 ::2:6:6:6 source ::2:11:11:11
Type escape sequence to abort.
Tracing the route to ::2:6:6:6
```

```
 1  fd00:192:168:205::5 19 msec 9 msec 0 msec
 2  ::ffff:94.5.7.7 [MPLS: Labels 7003/2003 Exp 0] 29 msec 39 msec 39 msec
 3  2094:94:7:13::13 [MPLS: Labels 93005/2003 Exp 0] 39 msec 39 msec 39 msec
 4  fd00:192:168:12::12 [MPLS: Labels 92004/2003 Exp 0] 39 msec 39 msec 49 msec
 5  ::ffff:49.3.12.3 [MPLS: Labels 3000/8001/2003 Exp 0] 39 msec 39 msec 39 msec
 6  ::ffff:49.3.8.8 [MPLS: Labels 8001/2003 Exp 0] 39 msec 39 msec 29 msec
 7  ::ffff:192.168.8.10 [MPLS: Labels 10027/2003 Exp 0] 39 msec 39 msec 39 msec
 8  fd00:192:168:202::2 [MPLS: Label 2003 Exp 0] 29 msec 49 msec 39 msec
 9  fd00:192:168:202::6 39 msec 39 msec 99 msec
```

Next, we will trace the same set of IPv6 prefixes in the reverse direction, which takes a different path. Because the CE (CSR6) is multi-homed, we begin there. CSR6 prefers the path via CSR4 due to local-preference which is the preferred ingress PE for VRF U2.

```
CSR6#show bgp vpnv6 unicast vrf U2 ::2:11:11:11/128
BGP routing table entry for [49:202]::2:11:11:11/128, version 396
Paths: (2 available, best #1, table U2)
Advertised to update-groups:
 2
Refresh Epoch 1
94 94
```

```

    FD00:192:168:204::4 (FE80::4) (via vrf U2) from FD00:192:168:204::4
(4.4.4.4)
    Origin incomplete, localpref 200, valid, external, best
    rx pathid: 0, tx pathid: 0x0
Refresh Epoch 1
94 94
    FD00:192:168:202::2 (FE80::2) (via vrf U2) from FD00:192:168:202::2
(2.2.2.2)
    Origin incomplete, localpref 100, valid, external
    rx pathid: 0, tx pathid: 0

```

Checking CSR4, we can see that CSR5 has allocated label 5009 for prefix. The next-hop is 5.5.5.5, which has a corresponding BGP route carrying label 9026. Since the next-hop for 5.5.5.5 is connected, the final label stack is {9026 5009} and the packet is sent towards CSR9.

```

CSR4#show bgp vpnv6 unicast vrf U2 ::2:11:11:11/128
BGP routing table entry for [49:204]::2:11:11:11/128, version 7
Paths: (1 available, best #1, table U2)
  Advertised to update-groups:
    13
Refresh Epoch 2
65000, imported path from [49:205]::2:11:11:11/128 (global)
  ::FFFF:5.5.5.5 (via default) from 2.2.2.2 (2.2.2.2)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    Extended Community: RT:49:205
    Originator: 5.5.5.5, Cluster list: 2.2.2.2
    Connector Attribute: count=1
      type 1 len 12 value 49:205:5.5.5.5
    mpls labels in/out nlabel/5009
    rx pathid: 0, tx pathid: 0x0

```

```

CSR4#show ip route 5.5.5.5
Routing entry for 5.5.5.5/32
  Known via "bgp 94", distance 20, metric 0
  Tag 49, type external
  Last update from 192.168.4.9 01:22:18 ago
  Routing Descriptor Blocks:
  * 192.168.4.9, from 192.168.4.9, 01:22:18 ago
    Route metric is 0, traffic share count is 1
    AS Hops 1
    Route tag 49
    MPLS label: 9026

```

```

CSR4#show ip route 192.168.4.9
Routing entry for 192.168.4.9/32
  Known via "connected", distance 0, metric 0 (connected, via interface)
  Redistributing via bgp 94
  Routing Descriptor Blocks:

```

```
* directly connected, via GigabitEthernet2.549
  Route metric is 0, traffic share count is 1
```

CSR9 consults its LFIB upon receipt of this packet. Label 9026 is swapped for label 1027 and packets are sent towards CSR1.

```
CSR9#show mpls forwarding-table labels 9026
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id    Switched     interface
9026       1027       5.5.5.5/32 [V]  4434         Gi2.519    49.1.9.1
```

Behind the scenes, there is more activity. CSR9 learns two VPNv4 routes for this customer prefix from CSR1 and XRv12. CSR1 is chosen as the best-path because it is closer than XRv12 from an IGP perspective, so BGP label 1027 is used instead of 92001.

```
CSR9#show bgp vpnv4 unicast vrf T2 5.5.5.5/32
BGP routing table entry for 49:9:5.5.5.5/32, version 14
Paths: (2 available, best #2, table T2)
  Advertised to update-groups:
    5
  Refresh Epoch 1
  94, imported path from 49:12:5.5.5.5/32 (global)
    12.12.12.12 (metric 40) (via default) from 12.12.12.12 (12.12.12.12)
      Origin IGP, metric 3, localpref 100, valid, internal
      Extended Community: RT:49:12
      Connector Attribute: count=1
        type 1 len 12 value 49:12:12.12.12.12
      mpls labels in/out 9026/92001
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  Local, imported path from 49:1:5.5.5.5/32 (global)
    1.1.1.1 (metric 10) (via default) from 12.12.12.12 (12.12.12.12)
      Origin incomplete, metric 3, localpref 100, valid, internal, best
      Extended Community: RT:49:1 OSPF DOMAIN ID:0x0005:0x00000005E0200
        OSPF RT:0.0.0.0:2:0 OSPF ROUTER ID:0.0.0.1:0
      Originator: 1.1.1.1, Cluster list: 12.12.12.12
      Connector Attribute: count=1
        type 1 len 12 value 49:1:1.1.1.1
      mpls labels in/out 9026/1027
      rx pathid: 0, tx pathid: 0x0
```

We confirm that the IGP cost to CSR1 is 10, and since the route is IS-IS, an LDP label will be pushed onto the label stack to reach this next-hop. CSR1 is advertising explicit-null, so the label stack becomes {0 1027 5009}.

```
CSR9#show ip route 1.1.1.1
Routing entry for 1.1.1.1/32
```

```
Known via "isis", distance 115, metric 10, type level-2
Redistributing via isis 49
Last update from 49.1.9.1 on GigabitEthernet2.519, 4d18h ago
Routing Descriptor Blocks:
* 49.1.9.1, from 1.1.1.1, 4d18h ago, via GigabitEthernet2.519
  Route metric is 10, traffic share count is 1
```

```
CSR9#show mpls ldp bindings 1.1.1.1 32 neighbor 1.1.1.1
lib entry: 1.1.1.1/32, rev 126
remote binding: lsr: 1.1.1.1:0, label: exp-null
```

CSR1 receives the packet and performs a lookup against label 0. This label is popped and the next-hop is identified as "input table", which means another lookup must occur in the same table in which the packet was received. Since the next label is 1027, the LFIB swaps 1027 for 94000. Normally, when verifying LSPs, you don't need to lookup label 0 since you already know the action is always pop and the follow-on action is another lookup.

```
CSR1#show mpls forwarding-table labels 0
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id  Switched     interface
explic     Pop Label  ipv4 exp-null  31937178     aggregate/<input table>

CSR1#show mpls forwarding-table labels 1027
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id  Switched     interface
1027       94000     5.5.5.5/32 [V]  16934232     Gi2.514   192.168.1.14
```

Unlike the other 3 CSC PE's, CSR1 is not running BGP IPv4 labeled-unicast with the customer carrier. When label 1027 arrives, BGP claims there is no outgoing label towards XRv14, which is true from a BGP-only perspective. This route is actually originated on CSR1 (not BGP learned), so this output makes sense; it also implies that CSR1's RIB doesn't have the BGP route installed. Instead, the route is learned via VRF-aware OSPF, which is an IGP route requiring an LDP label for transport. This hybrid design is very awkward but illustrates the CSC options.

```
CSR1#show bgp vpnv4 unicast vrf T2 5.5.5.5/32
BGP routing table entry for 49:1:5.5.5.5/32, version 9
Paths: (1 available, best #1, table T2)
  Advertised to update-groups:
    5
  Refresh Epoch 1
  Local
    192.168.1.14 (via vrf T2) from 0.0.0.0 (1.1.1.1)
      Origin incomplete, metric 3, localpref 100, weight 32768, valid,
sourced, best
      Extended Community: RT:49:1 OSPF DOMAIN ID:0x0005:0x00000005E0200
      OSPF RT:0.0.0.0:2:0 OSPF ROUTER ID:0.0.0.1:0
      mpls labels in/out 1027/nolabel
```



```
rx pathid: 0, tx pathid: 0x0
```

```
CSR1#show ip route vrf T2 5.5.5.5
Routing Table: T2
Routing entry for 5.5.5.5/32
  Known via "ospf 94", distance 110, metric 3, type intra area
  Redistributing via bgp 49
  Advertised by bgp 49 route-map RM_OSPF_TO_BGP
  Last update from 192.168.1.14 on GigabitEthernet2.514, 2d20h ago
  Routing Descriptor Blocks:
  * 192.168.1.14, from 5.5.5.5, 2d20h ago, via GigabitEthernet2.514
    Route metric is 3, traffic share count is 1
```

We can query the LIB to see the LDP bindings for 5.5.5.5/32 inside the VRF. This reveals label 94000 which the LFIB correctly identified as the out-label along this LSP. Again, the LFIB is smart enough to connect the VPNv4 label to the LDP label to support CSC. The label stack becomes {94000 5009}.

```
CSR1#show mpls ldp bindings vrf T2 5.5.5.5 32 neighbor 14.14.14.14
lib entry: 5.5.5.5/32, rev 4
remote binding: lsr: 14.14.14.14:0, label: 94000
```

When XRv14 receives this packet, it pops the topmost label to reveal label 5009 (VPNv6 label) to the remote PE, CSR5. CSR5 removes all labels from the stack and forwards the IPv6 packet to XRv11 inside VRF U2.

```
RP/0/0/CPU0:XRv14#show mpls forwarding labels 94000
Local   Outgoing   Prefix           Outgoing         Next Hop         Bytes
Label   Label       or ID            Interface        Next Hop         Switched
-----
94000   Pop         5.5.5.5/32      Gi0/0/0/0.554   94.5.14.5        14550528
```

```
CSR5#show mpls forwarding-table labels 5009 detail
Local   Outgoing   Prefix           Bytes Label      Outgoing   Next Hop
Label   Label      or Tunnel Id     Switched         interface
5009    No Label   ::2:11:11:11/128[V] \
                                     4896        Gi2.205     FE80::11
MAC/Encaps=22/22, MRU=1504, Label Stack{}
000C2950BAB7000C295F11A181000DDF810000CD86DD
VPN route: U2
No output feature configured
```

We can use traceroute to verify the label stacks. Explicit-null "confuses" traceroute and masks XRv14 as a hop in the path, but otherwise the labels appear correct. This confusion is due to the need for multiple LFIB lookups on a single router.

```
CSR6#traceroute vrf U2 ipv6
Target IPv6 address: ::2:11:11:11
```

```
Source address: ::2:6:6:6
[snip]
Tracing the route to ::2:11:11:11
```

```
 1 FD00:192:168:204::4 1 msec 0 msec 1 msec
 2 ::FFFF:192.168.4.9 [MPLS: Labels 9026/5009 Exp 0] 56 msec 41 msec 42 msec
 3 ::FFFF:49.1.9.1 [MPLS: Labels 0/1027/5009 Exp 0] 28 msec 25 msec 25 msec
 4 ::FFFF:49.1.9.1 [MPLS: Labels 0/1027/5009 Exp 0] 25 msec 33 msec 24 msec
 5 FD00:192:168:205::5 [MPLS: Label 5009 Exp 0] 28 msec 17 msec 25 msec
 6 ::2:11:11:11 [AS 94] 100 msec 59 msec 32 msec
```

33.1.2 L2VPN

This test uses BGP auto-discovery to build VPLS PWs along with BGP signaling for label exchange. As described in the L2VPN section, BGP signaling for VPLS doesn't actually exchange labels, but instead exchanges details about how the label should be derived. In the BGP AD route, we can see label 2020, but this is just the start of the label range allocated by CSR2; it may not actually be the exact label value in use.

```
CSR5#show bgp l2vpn vpls rd 94:567 ve-id 2 block-offset 1
BGP routing table entry for 94:567:VEID-2:Blk-1/136, version 20
Paths: (1 available, best #1, table L2VPN-VPLS-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    2.2.2.2 (metric 1) from 2.2.2.2 (2.2.2.2)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      AGI version(0), VE Block Size(10) Label Base(2020)
      Extended Community: RT:94:567 L2VPN L2:0x0:MTU-1500
      mpls labels in/out exp-null/2020
      rx pathid: 0, tx pathid: 0x0
```

```
CSR5#show l2vpn signaling rib rd 94:567 detail
Route 94:567:2 (epoch:0) from iBGP peer 2.2.2.2
Provisioned (Y) Stale (N)
  Route-Target: 94:567
NLRI [E9000007]
  VE-ID:2 VBO:1 VBS:10 LB:2020
  MTU: 1500 Control Word: off
RIB Filter [69000001]
  Local RD: 94:567
  VE-ID: 5, VBO: 1, VBS: 10 LB: 5023
Forwarder [CA000003] VFI VPLS
```

To quickly evaluate the math (described in detail in the L2VPN section), we first compute the VBO. This is defined as $VBO = \text{ROUND}(\text{VE-ID}/\text{VBS}) * \text{VBS}$ of the remote route, which evaluates as $\text{ROUND}(5/10) * 10 = 1$. The math isn't exactly 1, but 1 is considered the smallest usable block offset. Next, a sanity check is performed to ensure that the local VE-ID (5) is within the valid range given the remote VBO and VBS.

Specifically, the local VE-ID must evaluate as follows: $VBO \leq VE-ID < VBO + VBS$. In this case, the formula is true: $1 \leq 5 < 1 + 10$. Last, CSR5 must select a label from this block, and uses the following formula: $LB + VE-ID - VBO$. Again, the VE-ID is the local value while the VBO/VBS are for the received route. In this case, the label evaluates as follows: $2020 + 5 - 1 = 2024$. We confirm this value of 2024 by checking the L2VPN VFI details.

```
CSR5#show l2vpn vfi name VPLS
Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No
VFI name: VPLS, state: up, type: multipoint, signaling: BGP
  VPN ID: 567, VE-ID: 5, VE-SIZE: 10
  RD: 94:567, RT: 94:567
  Bridge-Domain 305 attachment circuits:
  Pseudo-port interface: pseudowire100016
  Interface          Peer Address      VE-ID  Local Label  Remote Label  S
  pseudowire100017   2.2.2.2           2      5024         2024         Y
```

Next, we need to find the route to the next-hop. This resolves to a pair of OSPF routes which is identical to what we saw earlier. Like UMPLS, CSC can support any MPLS service. What is odd is that the label stack, as confirmed by the PW details, is using CSR7 for next-hop recursion.

```
CSR5#show ip route 2.2.2.2
Routing entry for 2.2.2.2/32
  Known via "ospf 94", distance 110, metric 1
  Tag 49, type extern 2, forward metric 2
  Last update from 94.5.14.14 on GigabitEthernet2.554, 3d00h ago
  Routing Descriptor Blocks:
    94.5.14.14, from 13.13.13.13, 3d00h ago, via GigabitEthernet2.554
      Route metric is 1, traffic share count is 1
      Route tag 49
    * 94.5.7.7, from 13.13.13.13, 3d00h ago, via GigabitEthernet2.557
      Route metric is 1, traffic share count is 1
      Route tag 49
```

```
CSR5#show l2vpn atom vc vcid 567 2 detail | include label_stack
Output interface: Gi2.557, imposed label stack {7003 2024}
```

I am unable to find any documentation or show command to verify why this is the case, since CEF directs traffic between 5.5.5.5 and 2.2.2.2 towards XRv14. At least we can confirm that label 7003 is the proper LDP label allocated by CSR7 to represent 2.2.2.2/32. The label stack becomes {7003 2024} as indicated above. This is the same discrepancy noted in the L3VPN LSP trace.

```
CSR5#show ip cef exact-route virtual 5.5.5.5 2.2.2.2
5.5.5.5 -> 2.2.2.2 => label 94005 TAG adj out of GigabitEthernet2.554, addr
94.5.14.14
```

```
CSR5#show mpls ldp bindings 2.2.2.2 32 neighbor 7.7.7.7
```

```
lib entry: 2.2.2.2/32, rev 147
remote binding: lsr: 7.7.7.7:0, label: 7003
```

At this point, we might as well trace the new LSP. When CSR7 receives the packet, it swaps label 7003 for 93005 as a basic P router running LDP. The label stack becomes {93005 2024} and is delivered to XRv13.

```
CSR7#show mpls forwarding-table labels 7003
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id    Switched     interface
7003       93005      2.2.2.2/32      12769639     Gi2.573   94.7.13.13
```

When XRv13 receives the packet, the LDP label 93005 is swapped for the BGP IPv4 labeled-unicast value of 92004 at the CSC CE. The route is BGP-learned, which implies the label is BGP learned as well. The label stack becomes {92004 2024} and is delivered to XRv12.

```
RP/0/0/CPU0:XRv13#show mpls forwarding labels 93005
Local  Outgoing  Prefix          Outgoing  Next Hop      Bytes
Label  Label      or ID           Interface  Next Hop      Switched
-----
93005  92004      2.2.2.2/32      Gi0/0/0/0.523 192.168.12.12 15666830
```

```
RP/0/0/CPU0:XRv13#show route ipv4 unicast 2.2.2.2/32
Routing entry for 2.2.2.2/32
  Known via "bgp 94", distance 20, metric 0, [ei]-bgp, labeled unicast (3107)
  Tag 49, type external
  Routing Descriptor Blocks
    192.168.12.12, from 192.168.12.12, BGP external
    Route metric is 0
  No advertising protos.
```

```
RP/0/0/CPU0:XRv13#show bgp ipv4 labeled-unicast labels | include 2.2.2
*> 2.2.2.2/32          192.168.12.12    92004          93005
```

XRv12 performs an LFIB lookup and swaps label 92004 for label 8001. These are the CSC VPNv4 labels exchanged to provide IP reachability between the CSC customer sites. The route is BGP learned and the BGP label swapping occurs due to this. The transport LSP is identical to the one traced for L3VPN as only the MPLS service (L3VPN versus L2VPN) has changed.

```
RP/0/0/CPU0:XRv12#show mpls forwarding labels 92004
Local  Outgoing  Prefix          Outgoing  Next Hop      Bytes
Label  Label      or ID           Interface  Next Hop      Switched
-----
92004  8001      2.2.2.2/32 [V]          8.8.8.8    16365406
```

```
RP/0/0/CPU0:XRv12#show bgp vpnv4 unicast vrf T2 2.2.2.2/32
```

```

BGP routing table entry for 2.2.2.2/32, Route Distinguisher: 49:12
Versions:
  Process          bRIB/RIB  SendTblVer
  Speaker          42        42
    Local Label: 92004
Paths: (1 available, best #1)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  94, (Received from a RR-client)
    8.8.8.8 (metric 20) from 8.8.8.8 (8.8.8.8)
      Received Label 8001
        Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, imported
        Received Path ID 0, Local Path ID 1, version 42
        Extended community: RT:49:8
        Connector: type: 1, Value:49:8:8.8.8.8
        Source VRF: default, Source Route Distinguisher: 49:8

```

XRv12 also must push an additional label to reach CSR8. Since the route is IGP learned, the LDP label of 3000 is used. The label stack is now {3000 8001 2024}. Notice that this label stack is identical to the LSP we traced earlier, service (bottom) label notwithstanding. At this point, we will stop tracing this LSP for the sake of brevity since we already did it once.

```

RP/0/0/CPU0:XRv12#show route ipv4 8.8.8.8
Routing entry for 8.8.8.8/32
  Known via "isis 49", distance 115, metric 20, type level-2
  Routing Descriptor Blocks
    49.3.12.3, from 8.8.8.8, via GigabitEthernet0/0/0/0.532
      Route metric is 20
  No advertising protos.

```

```

RP/0/0/CPU0:XRv12#show mpls ldp bindings 8.8.8.8/32 neighbor 3.3.3.3
8.8.8.8/32, rev 134
  Local binding: label: 92003
  Remote bindings: (2 peers)
    Peer          Label
    -----
    3.3.3.3:0     3000

```

We can quickly verify the LSP using traceroute. Notice I use a probe count of 1, and then run the command a few times until I see CSR7 as the first hop since XRv14 has a 50-50 chance of being selected. This “chance” is the result of the self-generated traceroute probes being processed switched, and thus load-balanced per-packet in software. We run this on the PE routers in the customer carrier since this would not yield valuable results inside the VPN.

```

CSR5#traceroute 2.2.2.2 source 5.5.5.5 probe 1

```

```
Type escape sequence to abort.
Tracing the route to 2.2.2.2
VRF info: (vrf in name/id, vrf out name/id)
 1 94.5.7.7 [MPLS: Label 7003 Exp 0] 35 msec
 2 94.13.14.13 [MPLS: Label 93005 Exp 0] 33 msec
 3 192.168.12.12 [MPLS: Label 92004 Exp 0] 25 msec
 4 49.3.12.3 [MPLS: Labels 3000/8001 Exp 0] 33 msec
 5 192.168.8.8 [MPLS: Label 8001 Exp 0] 41 msec
 6 192.168.8.10 [MPLS: Label 10027 Exp 0] 33 msec
 7 94.2.10.2 25 msec
```

We will jump to CSR2 to verify the first few hops of its LSP as well for completeness. First, we need to do the VPLS math to determine the PW label. The BGP route is learned from 5.5.5.5 and carries a VBS of 10 and an LB of 5023. CSR2's local VE-ID is 2.

```
CSR2#show bgp l2vpn vpls rd 94:567 ve-id 5 block-offset 1
BGP routing table entry for 94:567:VEID-5:Blk-1/136, version 19
Paths: (1 available, best #1, table L2VPN-VPLS-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    5.5.5.5 (metric 2) from 5.5.5.5 (5.5.5.5)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      AGI version(0), VE Block Size(10) Label Base(5023)
      Extended Community: RT:94:567 L2VPN L2:0x0:MTU-1500
      mpls labels in/out exp-null/5023
      rx pathid: 0, tx pathid: 0x0
```

```
CSR2#show l2vpn signaling rib rd 94:567 detail
Route 94:567:5 (epoch:0) from iBGP peer 5.5.5.5
Provisioned (Y) Stale (N)
  Route-Target: 94:567
NLRI [7E000007]
  VE-ID:5 VBO:1 VBS:10 LB:5023
  MTU: 1500 Control Word: off
RIB Filter [23000001]
  Local RD: 94:567
  VE-ID: 2, VBO: 1, VBS: 10 LB: 2020
Forwarder [35000003] VFI VPLS
```

First, we evaluate the VBO: $VBO = \text{ROUND}(\text{VE-ID}/\text{VBS}) * \text{VBS} = \text{ROUND}(2/10) * 10 = 1$. This is displayed above but we verify it for completeness. Next, we perform a sanity check to ensure the router is looking at the proper block of labels for this PW based on the local VE-ID. Evaluate: $VBO \leq \text{VE-ID} < VBO + \text{VBS}$ translates into $1 \leq 2 < 1 + 10$, which is true. CSR2 then selects a label from this block using the formula $\text{LB} + \text{VE-ID} - \text{VBO}$. This evaluates as $5023 + 2 - 1 = 5024$. We can confirm this label by checking the L2VPN VFI summary or the PW details; both are shown below.

```

CSR2#show l2vpn vfi name VPLS
Legend: RT=Route-target, S=Split-horizon, Y=Yes, N=No
VFI name: VPLS, state: up, type: multipoint, signaling: BGP
  VPN ID: 567, VE-ID: 2, VE-SIZE: 10
  RD: 94:567, RT: 94:567
  Bridge-Domain 302 attachment circuits:
  Pseudo-port interface: pseudowire100018
  Interface          Peer Address      VE-ID  Local Label  Remote Label  S
  pseudowire100019   5.5.5.5           5      2024         5024          Y

CSR2#show l2vpn atom vc vcid 567 2 detail | include label_stack
  Output interface: Gi2.520, imposed label stack {10029 5024}

```

We also see label 10029 on the label stack from the PW details. The route to 5.5.5.5 is a BGP route that also carries a label; unlike CSR5, the route to the BGP next-hop is another BGP route since CSR10 is not performing BGP-to-IGP redistribution. Potentially, a third label may be needed to reach 10.10.10.10, which is the BGP next-hop for the labeled-unicast route. Since this is an IGP route directly connected to CSR10, LDP implicit-null is used, so the label stack remains {10029 5024}.

```

CSR2#show ip route 5.5.5.5
Routing entry for 5.5.5.5/32
  Known via "bgp 94", distance 200, metric 0
  Tag 49, type internal
  Last update from 10.10.10.10 3d01h ago
  Routing Descriptor Blocks:
  * 10.10.10.10, from 10.10.10.10, 3d01h ago
    Route metric is 0, traffic share count is 1
    AS Hops 1
    Route tag 49
    MPLS label: 10029

CSR2#show ip route 10.10.10.10
Routing entry for 10.10.10.10/32
  Known via "ospf 94", distance 110, metric 2, type intra area
  Last update from 94.2.10.10 on GigabitEthernet2.520, 4d03h ago
  Routing Descriptor Blocks:
  * 94.2.10.10, from 10.10.10.10, 4d03h ago, via GigabitEthernet2.520
    Route metric is 2, traffic share count is 1

CSR2#show mpls ldp bindings 10.10.10.10 32 neighbor 10.10.10.10
  lib entry: 10.10.10.10/32, rev 124
    remote binding: lsr: 10.10.10.10:0, label: imp-null

```

When CSR10 receives this packet, it swaps label 10029 for label 8028. Since the route to 5.5.5.5/32 is a BGP route, the BGP label of 8028 is used, and we can see how the LFIB connects the LDP and BGP labels seamlessly. The label stack becomes {8028 5024}.

```
CSR10#show mpls forwarding-table labels 10029
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
10029	8028	5.5.5.5/32	18219161	Gi2.580	192.168.8.8

```
CSR10#show ip route 5.5.5.5
Routing entry for 5.5.5.5/32
  Known via "bgp 94", distance 20, metric 0
  Tag 49, type external
  Last update from 192.168.8.8 3d01h ago
  Routing Descriptor Blocks:
    * 192.168.8.8, from 192.168.8.8, 3d01h ago
      Route metric is 0, traffic share count is 1
      AS Hops 1
      Route tag 49
      MPLS label: 8028
```

When CSR8 receives the packet, it consults its LFIB which shows a label swap to 1027. This is the VPNv4 label learned from CSR1. We can see the route is installed in the VRF routing table as a BGP route with an MPLS label and a flag indicating that MPLS must be used for transport. This is the result of the CSC architecture.

```
CSR8#show mpls forwarding-table labels 8028
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
8028	1027	5.5.5.5/32 [V]	18923482	Gi2.518	49.1.8.1

```
CSR8#show ip route vrf T2 5.5.5.5
Routing Table: T2
Routing entry for 5.5.5.5/32
  Known via "bgp 49", distance 200, metric 3, type internal
  Last update from 1.1.1.1 3d01h ago
  Routing Descriptor Blocks:
    * 1.1.1.1 (default), from 12.12.12.12, 3d01h ago
      Route metric is 3, traffic share count is 1
      AS Hops 0
      MPLS label: 1027
      MPLS Flags: MPLS Required
```

For completeness, we can see that CSR8 learned two BGP routes for 5.5.5.5/32 inside the VPN. The route to CSR1 was chosen because the AS path to CSR1 is shorter than the one to Xrv12.

```
CSR8#show bgp vpnv4 unicast vrf T2 5.5.5.5/32
BGP routing table entry for 49:8:5.5.5.5/32, version 23
Paths: (2 available, best #2, table T2)
  Advertised to update-groups:
    3
  Refresh Epoch 1
```



```

94, imported path from 49:12:5.5.5.5/32 (global)
12.12.12.12 (metric 20) (via default) from 12.12.12.12 (12.12.12.12)
  Origin IGP, metric 3, localpref 100, valid, internal
  Extended Community: RT:49:12
  Connector Attribute: count=1
    type 1 len 12 value 49:12:12.12.12.12
  mpls labels in/out 8028/92001
  rx pathid: 0, tx pathid: 0
Refresh Epoch 1
Local, imported path from 49:1:5.5.5.5/32 (global)
1.1.1.1.1 (metric 10) (via default) from 12.12.12.12 (12.12.12.12)
  Origin incomplete, metric 3, localpref 100, valid, internal, best
  Extended Community: RT:49:1 OSPF DOMAIN ID:0x0005:0x00000005E0200
    OSPF RT:0.0.0.0:2:0 OSPF ROUTER ID:0.0.0.1:0
  Originator: 1.1.1.1, Cluster list: 12.12.12.12
  Connector Attribute: count=1
    type 1 len 12 value 49:1:1.1.1.1
  mpls labels in/out 8028/1027
  rx pathid: 0, tx pathid: 0x0

```

CSR8 also may need to push a null label to reach CSR1. In this case, explicit-null is added to the stack, resulting in {0 1027 5024}.

```

CSR8#show mpls ldp bindings 1.1.1.1 32 neighbor 1.1.1.1
  lib entry: 1.1.1.1/32, rev 136
    remote binding: lsr: 1.1.1.1:0, label: exp-null

```

CSR1 performs two lookups in quick succession: one to remove the explicit-null label (typically used for QoS uniform and long-pipe modes), and one to observe the BGP transport label of 1027. This is swapped for label 94000 which was allocated by XRv14.

```

CSR1#show mpls forwarding-table labels 0
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id  Switched     interface
explicit   Pop Label   ipv4 exp-null 33242335     aggregate/<input table>

CSR1#show mpls forwarding-table labels 1027
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id  Switched     interface
1027       94000      5.5.5.5/32 [V] 18231011     Gi2.514    192.168.1.14

```

Since the route to 5.5.5.5/32 is an IGP route, CSR1 will use an LDP label to reach it. XRv14 allocated 94000 as its local LDP label for this prefix. CSR1 connects the VPNv4 label (CSC core carrier VPN label) with the LDP label (CSC customer transport label) within the VRF and sends the packet to XRv14. The label stack is now {94000 5024}.

```

CSR1#show ip route vrf T2 5.5.5.5

```

Routing Table: T2

Routing entry for 5.5.5.5/32

Known via "ospf 94", distance 110, metric 3, type intra area

Redistributing via bgp 49

Advertised by bgp 49 route-map RM_OSPF_TO_BGP

Last update from 192.168.1.14 on GigabitEthernet2.514, 3d01h ago

Routing Descriptor Blocks:

* 192.168.1.14, from 5.5.5.5, 3d01h ago, via GigabitEthernet2.514

Route metric is 3, traffic share count is 1

CSR1#show mpls ldp bindings vrf T2 5.5.5.5 32 neighbor 14.14.14.14

lib entry: 5.5.5.5/32, rev 4

remote binding: lsr: 14.14.14.14:0, label: 94000

XRv14 receives the packet, pops the topmost label, and forwards the packet only with label 5024 towards CSR5. CSR5 delivers this traffic to the L2VPN as expected.

RP/0/0/CPU0:XRv14#show mpls forwarding labels 94000

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94000	Pop	5.5.5.5/32	Gi0/0/0/0.554	94.5.14.5	15659423

CSR5#show mpls forwarding-table labels 5024

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Outgoing interface	Next Hop
5024	No Label	lbl-blk-id(1:1)	1112176	none	point2point

We confirm the LSP using traceroute from the customer PE. Explicit-null masks the XRv14 label of 94000 but the stack is otherwise correct. We also check the EIGRP neighbors for IPv4 and IPv6 on CSR6 to ensure it has valid neighbor relationships with XRv11 for both AFIs.

CSR2#traceroute 5.5.5.5 source 2.2.2.2

Type escape sequence to abort.

Tracing the route to 5.5.5.5

VRF info: (vrf in name/id, vrf out name/id)

```
 1 94.2.10.10 [MPLS: Label 10029 Exp 0] 69 msec 33 msec 25 msec
 2 192.168.8.8 [MPLS: Label 8028 Exp 0] 17 msec 16 msec 17 msec
 3 192.168.1.1 [MPLS: Labels 0/1027 Exp 0] 38 msec 17 msec 17 msec
 4 192.168.1.1 [MPLS: Labels 0/1027 Exp 0] 17 msec 24 msec 17 msec
 5 94.5.14.5 37 msec * 23 msec
```

CSR6#show eigrp address-family ipv4 vrf U3 neighbors

EIGRP-IPv4 VR(VPLS) Address-Family Neighbors for AS(65000)

VRF (U3)

H	Address	Interface	Hold Uptime (sec)	SRTT (ms)	RTO	Q Cnt	Seq Num
0	192.168.0.11	Gi2.302	10 3d01h	72	432	0	18

```

CSR6#show eigrp address-family ipv6 vrf U3 neighbors
EIGRP-IPv6 VR(VPLS) Address-Family Neighbors for AS(65000)
      VRF(U3)
H   Address                Interface      Hold Uptime    SRTT    RTO  Q   Seq
                               (sec)          (ms)          Cnt  Num
0   Link-local address:      Gi2.302        12 3d01h      104    624  0   22
    FE80::11

```

33.1.3 MVPN (Profile 0 with SSM)

To test multicast over CSC, we will use PIM/GRE since XRv actually supports that fully. Cisco documentation suggests that only MVPN profile 0 is supported for CSC as well. Both CSR2 and CSR5 use identical MDT configurations for VRFs U1 and U2. SSM is enabled for all multicast transport in the customer and core carrier networks for simplicity.

```

! CSR2 and CSR5
vrf definition U1
  address-family ipv4
    mdt default 232.94.94.251
    mdt data 232.94.41.0 0.0.0.127

  address-family ipv6
    mdt default 232.94.94.251
    mdt data 232.94.61.0 0.0.0.127

vrf definition U2
  address-family ipv4
    mdt default 232.94.94.252
    mdt data 232.94.42.0 0.0.0.127

  address-family ipv6
    mdt default 232.94.94.252
    mdt data 232.94.62.0 0.0.0.127

```

As mentioned earlier, IPv4 MDT is used to exchange the MDT information between the customer PE endpoints. Looking at the MDT route from CSR5 inside VRF U1 on CSR2, we can see the route carries the PE loopback address and the MDT default group, which is used for PIM c-mcast signaling. We also verify the opposite on CSR5 to ensure it learns CSR2's route with the matching MDT address.

```

CSR2#show bgp ipv4 mdt rd 49:105 5.5.5.5/32
BGP routing table entry for 49:105:5.5.5.5/32          version 10
Paths: (1 available, best #1, table IPv4-MDT-BGP-Table)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local, (Received from a RR-client)
    5.5.5.5 from 5.5.5.5 (5.5.5.5)

```

```
Origin incomplete, metric 0, localpref 100, valid, internal, best,
MDT group address: 232.94.94.251
rx pathid: 0, tx pathid: 0x0
```

```
CSR5#show bgp ipv4 mdt rd 49:102 2.2.2.2/32
BGP routing table entry for 49:102:2.2.2.2/32          version 14
Paths: (1 available, best #1, table IPv4-MDT-BGP-Table)
  Not advertised to any peer
  Refresh Epoch 3
  Local
    2.2.2.2 from 2.2.2.2 (2.2.2.2)
    Origin incomplete, metric 0, localpref 100, valid, internal, best,
    MDT group address: 232.94.94.251
    rx pathid: 0, tx pathid: 0x0
```

The customer PEs won't actually form PIM neighbors until multicast transport is provided between the two endpoints. This is the responsibility of the core carrier, which could use any other kind of MVPN transport, but in this case also uses PIM/GRE with SSM. XRv12 is the route-reflector that learns the MDT routes from all core carrier PEs. Since L3VPN routes are only exchanged east-to-west (that is, CSR8/CSR9 and XRv12/CSR1 don't exchange routes), the MVPN cannot loop as RPF will fail. Looking into the specific route exchange between CSR8 and XRv12, we can see they share an MDT default address for the T2 VRF.

```
RP/0/0/CPU0:XRv12#show bgp ipv4 mdt | begin Network
  Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 49:1
*>i1.1.1.1/96      1.1.1.1              0      100      0 ?
Route Distinguisher: 49:8
*>i8.8.8.8/96      8.8.8.8              0      100      0 ?
Route Distinguisher: 49:9
*>i9.9.9.9/96      9.9.9.9              0      100      0 ?
Route Distinguisher: 49:12
*> 12.12.12.12/96  0.0.0.0              0      100      0 i
```

```
RP/0/0/CPU0:XRv12#show bgp ipv4 mdt rd 49:8 8.8.8.8
BGP routing table entry for 8.8.8.8/96, Route Distinguisher: 49:8
Versions:
  Process          bRIB/RIB  SendTblVer
  Speaker          12        12
Paths: (1 available, best #1)
  Advertised to update-groups (with more than one peer):
    0.1
  Path #1: Received by speaker 0
  Advertised to update-groups (with more than one peer):
    0.1
  Local, (Received from a RR-client)
    8.8.8.8 (metric 20) from 8.8.8.8 (8.8.8.8)
```

```
Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best
```

```
Received Path ID 0, Local Path ID 1, version 12
```

```
MDT group address: 232.49.49.255
```

```
CSR8#show bgp ipv4 mdt rd 49:12 12.12.12.12
```

```
BGP routing table entry for 49:12:12.12.12.12/32          version 5
```

```
Paths: (1 available, best #1, table IPv4-MDT-BGP-Table)
```

```
Not advertised to any peer
```

```
Refresh Epoch 1
```

```
Local
```

```
12.12.12.12 from 12.12.12.12 (12.12.12.12)
```

```
Origin IGP, localpref 100, valid, internal, best,
```

```
MDT group address: 232.49.49.255
```

```
rx pathid: 0, tx pathid: 0x0
```

We quickly verify that the MDT tunnel P(S,G) state exists on XRv12 and CSR8. We can see that each router has joined the SSM SPT of every other PE in this MVPN instance.

```
RP/0/0/CPU0:XRv12#show pim topology 232.49.49.255 | begin 232
```

```
(1.1.1.1,232.49.49.255)SPT SSM Up: 3d18h
```

```
JP: Join(00:00:09) RPF: GigabitEthernet0/0/0/0.532,49.3.12.3 Flags:
```

```
Loopback0          3d18h          fwd LI LH
```

```
(8.8.8.8,232.49.49.255)SPT SSM Up: 3d18h
```

```
JP: Join(00:00:09) RPF: GigabitEthernet0/0/0/0.532,49.3.12.3 Flags:
```

```
Loopback0          3d18h          fwd LI LH
```

```
(9.9.9.9,232.49.49.255)SPT SSM Up: 3d18h
```

```
JP: Join(00:00:09) RPF: GigabitEthernet0/0/0/0.532,49.3.12.3 Flags:
```

```
Loopback0          3d18h          fwd LI LH
```

```
(12.12.12.12,232.49.49.255)SPT SSM Up: 3d20h
```

```
JP: Join(never) RPF: Loopback0,12.12.12.12* Flags:
```

```
Loopback0          3d20h          fwd LI LH
```

```
GigabitEthernet0/0/0/0.532 23:20:27 fwd Join(00:02:30)
```

```
CSR8#show ip mroute 232.49.49.255 | begin \
```

```
(9.9.9.9, 232.49.49.255), 3d19h/00:02:52, flags: sTIZ
```

```
Incoming interface: GigabitEthernet2.518, RPF nbr 49.1.8.1
```

```
Outgoing interface list:
```

```
GigabitEthernet2.538, Forward/Sparse, 23:21:00/00:02:52
```

```
MVRF T2, Forward/Sparse, 3d18h/00:02:49
```

```
(8.8.8.8, 232.49.49.255), 3d19h/00:02:42, flags: sT
```

```
Incoming interface: Loopback0, RPF nbr 0.0.0.0
```

```
Outgoing interface list:
```

```
GigabitEthernet2.538, Forward/Sparse, 23:21:00/00:02:42
```

```
GigabitEthernet2.518, Forward/Sparse, 3d18h/00:02:33
```

```
(1.1.1.1, 232.49.49.255), 3d19h/00:03:15, flags: sTIZ
Incoming interface: GigabitEthernet2.518, RPF nbr 49.1.8.1
Outgoing interface list:
GigabitEthernet2.538, Forward/Sparse, 23:21:00/00:03:15
MVRF T2, Forward/Sparse, 3d18h/00:02:49
```

```
(12.12.12.12, 232.49.49.255), 3d20h/00:03:09, flags: sTIZ
Incoming interface: GigabitEthernet2.538, RPF nbr 49.3.8.3
Outgoing interface list:
MVRF T2, Forward/Sparse, 3d18h/00:02:49
GigabitEthernet2.518, Forward/Sparse, 3d18h/00:03:09
```

PIM also must be enabled on any P routers as well. Looking at CSR3, we can see it is in the transit path for all 4 of the SPT trees; this is a result of the link cost adjustments since the P(S,G) joins travel up the reverse path towards the source.

```
CSR3#show ip mroute ssm
```

```
(12.12.12.12, 232.49.49.255), 23:21:31/00:03:23, flags: sT
Incoming interface: GigabitEthernet2.532, RPF nbr 49.3.12.12
Outgoing interface list:
GigabitEthernet2.538, Forward/Sparse, 23:21:31/00:03:23
```

```
(8.8.8.8, 232.49.49.255), 23:21:47/00:02:44, flags: sT
Incoming interface: GigabitEthernet2.538, RPF nbr 49.3.8.8
Outgoing interface list:
GigabitEthernet2.532, Forward/Sparse, 23:21:47/00:02:44
```

```
(1.1.1.1, 232.49.49.255), 23:21:47/00:02:44, flags: sT
Incoming interface: GigabitEthernet2.538, RPF nbr 49.3.8.8
Outgoing interface list:
GigabitEthernet2.532, Forward/Sparse, 23:21:47/00:02:44
```

```
(9.9.9.9, 232.49.49.255), 23:21:47/00:02:44, flags: sT
Incoming interface: GigabitEthernet2.538, RPF nbr 49.3.8.8
Outgoing interface list:
GigabitEthernet2.532, Forward/Sparse, 23:21:47/00:02:44
```

With the MDTs properly built, we can verify that there are PIM neighbors inside the T2 VRF. This will provide multicast transport across the core carrier for the customer carrier.

```
RP/0/0/CPU0:XRv12#show pim vrf T2 neighbor | begin Neighbor
```

Neighbor Address	Interface	Uptime	Expires	DR pri	Flags
192.168.12.12*	GigabitEthernet0/0/0/0.523	3d20h	00:01:34	1	B P E
192.168.12.13	GigabitEthernet0/0/0/0.523	3d19h	00:01:32	1	(DR) B P
1.1.1.1	mdtT2	3d19h	00:01:26	1	P
8.8.8.8	mdtT2	3d19h	00:01:16	1	P
9.9.9.9	mdtT2	3d19h	00:01:17	1	P

```
12.12.12.12*      mdtT2                3d20h      00:01:22 1 (DR) P
```

```
CSR8#show ip pim vrf T2 neighbor | begin ^Neighbor
```

Neighbor Address	Interface	Uptime/Expires	Ver	DR Prio/Mode
1.1.1.1	Tunnel0	3d18h/00:01:37	v2	1 / S P G
9.9.9.9	Tunnel0	3d18h/00:01:28	v2	1 / S P G
12.12.12.12	Tunnel0	3d19h/00:01:33	v2	1 / DR P G
192.168.8.10	GigabitEthernet2.580	3d19h/00:01:27	v2	1 / DR S P G

Now, we can backtrack to the customer PEs to see if PIM neighbors have formed there. If they have, this is an indication that the customer carrier default MDT has end-to-end connectivity, which means the core carrier MVPN transport is also working. We will verify PIM neighbors inside VRF U1 on CSR2 and CSR5 for brevity.

```
CSR5#show ip pim vrf U1 neighbor | begin ^Neighbor
```

Neighbor Address	Interface	Uptime/Expires	Ver	DR Prio/Mode
192.168.105.11	GigabitEthernet2.105	3d19h/00:01:36	v2	1 / DR P G
4.4.4.4	Tunnel3	3d18h/00:01:22	v2	1 / S P G
2.2.2.2	Tunnel3	3d19h/00:01:26	v2	1 / S P G

```
CSR2#show ip pim vrf U1 neighbor | begin ^Neighbor
```

Neighbor Address	Interface	Uptime/Expires	Ver	DR Prio/Mode
5.5.5.5	Tunnel3	3d19h/00:01:42	v2	1 / DR S P G
4.4.4.4	Tunnel3	3d19h/00:01:43	v2	1 / S P G
192.168.102.6	GigabitEthernet2.102	3d18h/00:01:29	v2	1 / DR S P G

To test it, we will configure some IGMPv3 SSM joins on XRv11 within both VRF U1 and U2. Since XRv11 is running PIM with CSR5, the joins are configured on the loopback, which means CSR5 should receive C(S,G) joins. Before continuing, we verify that the IGMP joins are seen on XRv11's loopbacks.

```
! XRv11
router igmp
vrf U1
  interface Loopback1
    join-group 232.1.1.1 1.6.6.6

vrf U2
  interface Loopback2
    join-group 232.2.2.2 2.6.6.6
```

```
RP/0/0/CPU0:XRv11#show igmp vrf U1 groups 232.1.1.1
```

```
IGMP Connected Group Membership
```

Group Address	Interface	Uptime	Expires	Last Reporter

232.1.1.1	Loopback1	3d19h	never	1.11.11.11

```
RP/0/0/CPU0:XRv11#show igmp vrf U2 groups 232.2.2.2
```

IGMP Connected Group Membership

Group Address	Interface	Uptime	Expires	Last Reporter
232.2.2.2	Loopback2	3d19h	never	2.11.11.11

Next, we verify the PIM C(S,G) state on XRv11 and CSR5. We can see XRv11 receives traffic from CSR5 and CSR5 receives traffic from the MDT tunnel. We can see that a data MDT has been spawned for this group for P(S,G) transport and uses the MDT data group 232.94.41.0. The big 'Y' indicates that this multicast state entry receives traffic from an MDT data group (as opposed to the default group). CSR2, the source-connected PE, originates these data-MDT TLVs inside the MDT default tree to signal receivers that a more optimal group should be used.

```
RP/0/0/CPU0:XRv11#show pim vrf U1 topology 232.1.1.1 1.6.6.6 | begin 232
(1.6.6.6,232.1.1.1)SPT SSM Up: 3d19h
```

```
JP: Join(00:00:06) RPF: GigabitEthernet0/0/0/0.105,192.168.105.5 Flags:
Loopback1 3d19h fwd LI II LH
```

```
CSR5#show ip mroute vrf U1 232.1.1.1 1.6.6.6 | begin \
(1.6.6.6, 232.1.1.1), 3d19h/00:02:39, flags: sTY
```

```
Incoming interface: Tunnel3, RPF nbr 2.2.2.2,
MDT: [2.2.2.2,232.94.41.0]/00:02:01
```

Outgoing interface list:

```
GigabitEthernet2.105, Forward/Sparse, 1d00h/00:02:39
```

If we check the MDT data group in the global table, we can see it is an MDT tunnel (big 'Z') and that it supports VRF U1. Traffic is sourced from 2.2.2.2 which is the remote tunnel endpoint. CSR5 sends the P(S,G) join to XRv14 since it has a higher IP address than CSR7 and is therefore the RPF neighbor towards 2.2.2.2. XRv14 forwards it onto CSR1 which also has the higher neighbor IP address as the ECMP tie breaker for RPF.

```
CSR5#show ip mroute 232.94.41.0 2.2.2.2 | begin \
(2.2.2.2, 232.94.41.0), 1d00h/stopped, flags: sTIZ
```

```
Incoming interface: GigabitEthernet2.554, RPF nbr 94.5.14.14
```

Outgoing interface list:

```
MVRF U1, Forward/Sparse, 1d00h/00:01:50
```

```
RP/0/0/CPU0:XRv14#show pim topology 232.94.41.0 2.2.2.2 | begin 232
(2.2.2.2,232.94.41.0)SPT SSM Up: 1d00h
```

```
JP: Join(00:00:45) RPF: GigabitEthernet0/0/0/0.514,192.168.1.1 Flags:
GigabitEthernet0/0/0/0.554 1d00h fwd Join(00:02:59)
```


To answer the question of why CSR5 selected CSR2 as the remote PE and not CSR4, we can look at BGP. We examined this earlier; CSR2 is the bestpath because the BGP RID 2.2.2.2 is less than the originator ID of 4.4.4.4.

```
CSR5#show bgp vpvv4 unicast vrf U1 1.6.6.6
BGP routing table entry for 49:105:1.6.6.6/32, version 80
Paths: (2 available, best #2, table U1)
  Advertised to update-groups:
    3
  Refresh Epoch 2
  65000, imported path from 49:104:1.6.6.6/32 (global)
    4.4.4.4 (metric 1) (via default) from 2.2.2.2 (2.2.2.2)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Extended Community: RT:49:104
      Originator: 4.4.4.4, Cluster list: 2.2.2.2
      Connector Attribute: count=1
        type 1 len 12 value 49:104:4.4.4.4
      mpls labels in/out nlabel/4003
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 2
  65000, imported path from 49:102:1.6.6.6/32 (global)
    2.2.2.2 (metric 1) (via default) from 2.2.2.2 (2.2.2.2)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:49:102
      Connector Attribute: count=1
        type 1 len 12 value 49:102:2.2.2.2
      mpls labels in/out nlabel/2010
      rx pathid: 0, tx pathid: 0x0
```

CSR1 sees this join inside of VRF T2, which looks like an ordinary MVPN profile 0 customer. The customer-facing interface is added to the OIL as expected. Just like CSR5, this is mapped to an MDT data group for efficient transport across the core. For lack of a better term, I will call this second layer of MDT the CSC(S,G) state information, which wraps the P(S,G) encapsulation, which wraps the C(S,G) native traffic. We can see that this CSC(S,G) join is sent towards CSR8 because CSR9 is only originating the path towards 4.4.4.4, which isn't the tunnel endpoint. The big 'Y' and 'Z' represent reception of traffic on an MDT data group and an MDT tunnel state entry, respectively.

```
CSR1#show ip mroute vrf T2 232.94.41.0 2.2.2.2 | begin \
(2.2.2.2, 232.94.41.0), 1d00h/00:02:59, flags: sTY
  Incoming interface: Tunnel0, RPF nbr 8.8.8.8,
MDT: [8.8.8.8,232.49.4.3]/00:02:48
  Outgoing interface list:
    GigabitEthernet2.514, Forward/Sparse, 1d00h/00:02:59

CSR1#show ip mroute 232.49.4.3 8.8.8.8 | begin \
(8.8.8.8, 232.49.4.3), 1d00h/stopped, flags: sTIZ
  Incoming interface: GigabitEthernet2.518, RPF nbr 49.1.8.8
```

Outgoing interface list:

MVRF T2, Forward/Sparse, 1d00h/00:00:08

```
CSR1#show bgp vpnv4 unicast vrf T2 2.2.2.2/32
BGP routing table entry for 49:1:2.2.2.2/32, version 6
Paths: (1 available, best #1, table T2)
  Not advertised to any peer
  Refresh Epoch 1
  94, imported path from 49:8:2.2.2.2/32 (global)
    8.8.8.8 (metric 10) (via default) from 12.12.12.12 (12.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:49:8
      Originator: 8.8.8.8, Cluster list: 12.12.12.12
      Connector Attribute: count=1
        type 1 len 12 value 49:8:8.8.8.8
      mpls labels in/out nlabel/8001
      rx pathid: 0, tx pathid: 0x0
```

When CSR8 receives the join, it sees that it is the root of the tree. Looking at VRF T2, we can see that this P(S,G) group is mapped to the CSC(S,G) group we just verified. Notice that these two entries have little 'z' and little 'y'; this indicates sending to an MDT data group, which makes sense since CSR8 is the ingress CSC PE, not the egress one. CSR8 indicates sending the P(S,G) join to CSR10.

```
CSR8#show ip mroute 232.49.4.3 | begin \
(8.8.8.8, 232.49.4.3), 1d00h/00:02:56, flags: sTz
  Incoming interface: Loopback0, RPF nbr 0.0.0.0
  Outgoing interface list:
    GigabitEthernet2.518, Forward/Sparse, 1d00h/00:02:56

CSR8#show ip mroute vrf T2 232.94.41.0 2.2.2.2 | begin \
(2.2.2.2, 232.94.41.0), 1d00h/00:02:38, flags: sTy
  Incoming interface: GigabitEthernet2.580, RPF nbr 192.168.8.10
  Outgoing interface list:
    Tunnel0, Forward/Sparse, 1d00h/00:02:38
```

When CSR10 receives it, it forwards the P(S,G) state to CSR2. CSR2 is the root of the tree and identifies itself as a sending for MDT data within the provider network (little 'z'). Using the verbose option on CSR2 inside VRF U1, we can see the original C(S,G) state is mapped to the P(S,G) state we just verified. This VRF mroute entry has a little 'y' to indicate sending to an MDT data group within the tier 2 provider network.

```
CSR10#show ip mroute 232.94.41.0 2.2.2.2 | begin \
(2.2.2.2, 232.94.41.0), 1d00h/00:02:54, flags: sT
  Incoming interface: GigabitEthernet2.520, RPF nbr 94.2.10.2
  Outgoing interface list:
    GigabitEthernet2.580, Forward/Sparse, 1d00h/00:02:54
```

```

CSR2#show ip mroute 232.94.41.0 2.2.2.2 | begin \
(2.2.2.2, 232.94.41.0), 1d00h/00:03:10, flags: sTz
  Incoming interface: Loopback0, RPF nbr 0.0.0.0
  Outgoing interface list:
    GigabitEthernet2.520, Forward/Sparse, 1d00h/00:03:10

CSR2#show ip mroute vrf U1 232.1.1.1 1.6.6.6 verbose | begin \
(1.6.6.6, 232.1.1.1), 1d00h/00:03:11, flags: sTyp
  Incoming interface: GigabitEthernet2.102, RPF nbr 192.168.102.6
  Outgoing interface list:
    Tunnel3, GRE MDT: 232.94.41.0 (data), Forward/Sparse, 1d00h/00:03:11, p

```

Last, CSR2 sends the C(S,G) join to CSR6 who is ultimately the root of the customer tree. Traffic is "incoming" from loopback1, so we will use this for our tests. The entire path, from source to receiver, is CSR6 > CSR2 > CSR10 > CSR8 > CSR1 > XRv14 > CSR5 > XRv11. A quick traceroute reveals this, except for the presence of explicit-null masks XRv14 in the path.

```

CSR6#show ip mroute vrf U1 232.1.1.1 1.6.6.6 | begin \
(1.6.6.6, 232.1.1.1), 3d19h/00:03:08, flags: sT
  Incoming interface: Loopback1, RPF nbr 0.0.0.0
  Outgoing interface list:
    GigabitEthernet2.102, Forward/Sparse, 1d00h/00:03:08

CSR6#traceroute vrf U1 1.11.11.11 source 1.6.6.6
Type escape sequence to abort.
Tracing the route to 1.11.11.11
VRF info: (vrf in name/id, vrf out name/id)
 1 192.168.102.2 1 msec 0 msec 1 msec
 2 94.2.10.10 [MPLS: Labels 10029/5014 Exp 0] 30 msec 42 msec 30 msec
 3 192.168.8.8 [MPLS: Labels 8028/5014 Exp 0] 36 msec 33 msec 34 msec
 4 49.1.8.1 [MPLS: Labels 0/1027/5014 Exp 0] 33 msec 33 msec 34 msec
 5 49.1.8.1 [MPLS: Labels 0/1027/5014 Exp 0] 33 msec 33 msec 34 msec
 6 192.168.105.5 [MPLS: Label 5014 Exp 0] 18 msec 17 msec 16 msec
 7 192.168.105.11 29 msec * 27 msec

```

We can use EPC to verify all the layers of encapsulation to ensure our control-plane verifications were correct. We will capture inbound on CSR1 (facing the core) and send traffic from CSR6. I will use 500-byte packets marked with DSCP CS1 to make them easier to identify.

```

CSR6#ping vrf U1
Protocol [ip]:
Target IP address: 232.1.1.1
Repeat count [1]: 10
Datagram size [100]: 500
Timeout in seconds [2]: 1
Extended commands [n]: y
Interface [All]: loopback1

```

```

Time to live [255]:
Source address or interface: loopback1
Type of service [0]: 32
[snip]
Type escape sequence to abort.
Sending 10, 500-byte ICMP Echos to 232.1.1.1, timeout is 1 seconds:
Packet sent with a source address of 1.6.6.6

```

```

Reply to request 0 from 1.11.11.11, 22 ms
[snip]
Reply to request 9 from 1.11.11.11, 21 ms

```

Below is the packet dump from CSR1. First, we see the multicast destination MAC address of 0100.5e31.0403, which includes the low-order 23 bits of the multicast destination address (yellow). The CSC multicast group for MDT data transfer was 232.49.4.3, and we can see the hexadecimal representations of 49, 4, and 3 in the MAC address highlighted MAC address (0x310403). The payload of this frame is IP (0x0800 also in yellow), not MPLS, since we are using GRE encapsulation for MDT transport. Next, we can see the beginning of the outermost IP header with TOS set to 0x20, which is 00100000 in binary, or 32 in decimal (DSCP CS1) and is colored green. The destination IP address is 232.49.4.3 which is the CSC MDT data group, and is shown in cyan. The next IP header is the customer carrier MDT data group which also has DSCP CS1 (green) and has destination IP address 232.94.41.0 (pink). The third IP header is the original customer multicast packet that also has DSCP CS1 (green) with multicast destination 232.1.1.1 (grey). In this way, we can see three layers of GRE encapsulation in a CSC MVPN profile 0 design.

```

CSR1#show monitor capture CAP buffer dump
0
0000: 01005E31 0403000C 296AFFD5 81000DBE ..^1....)j.....
0010: 08004520 0224C3F3 0000FF2F F9520808 ..E $...../.R..
0020: 0808E831 04030000 08004520 020C000C ...1.....E ....
0030: 0000FD2F A6340202 0202E85E 29000000 .../.4.....^)...
0040: 08004520 01F4000C 0000FD01 CBCE0106 ..E .....
0050: 0606E801 01010800 2B720001 00020000 .....+r.....
0060: 000037A8 BBB00000 ABCD0000 ABCD0000 ..7.....

```

33.1.4 TE and TE-FRR

TE isn't very exciting for CSC because there isn't a way to maintain end-to-end TE paths between the customer carrier. Each individual island can run the full set of TE features as one would expect, but they generally don't cross between domains. First, we will create a basic TE tunnel from CSR2 to CSR10 that transits CSR4. We will remove the need for an LDP label to reach the BGP next-hop in this way, and expect the LFIB to automatically connect the RSVP-TE and BGP LSPs. A quick check on CSR2 shows that all three routers are in the TE topology, and all of their OSPF area 0 links are included. The routers are basically in a triangle with both links TE-enabled.

```

CSR2#show mpls traffic-eng topology brief | include IGP_Id

```

```

IGP Id: 2.2.2.2, MPLS TE Id:2.2.2.2 Router Node (ospf 94 area 0)
  link[0]: Point-to-Point, Nbr IGP Id: 10.10.10.10, nbr_node_id:11, gen:35
  link[1]: Point-to-Point, Nbr IGP Id: 4.4.4.4, nbr_node_id:12, gen:35
IGP Id: 4.4.4.4, MPLS TE Id:4.4.4.4 Router Node (ospf 94 area 0)
  link[0]: Point-to-Point, Nbr IGP Id: 10.10.10.10, nbr_node_id:11, gen:39
  link[1]: Point-to-Point, Nbr IGP Id: 2.2.2.2, nbr_node_id:10, gen:39
IGP Id: 10.10.10.10, MPLS TE Id:10.10.10.10 Router Node (ospf 94 area 0)
  link[0]: Point-to-Point, Nbr IGP Id: 2.2.2.2, nbr_node_id:10, gen:40
  link[1]: Point-to-Point, Nbr IGP Id: 4.4.4.4, nbr_node_id:12, gen:40

```

A basic tunnel configuration is shown below which tunnels traffic through CSR4. We check to ensure the tunnel comes up.

```

! CSR2
ip explicit-path name EP_2_4_10 enable
  next-address 4.4.4.4
  next-address 10.10.10.10

interface Tunnel200
  description BASIC TE TO CSR10
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 10.10.10.10
  tunnel mpls traffic-eng autoroute announce
  tunnel mpls traffic-eng path-option 10 explicit name EP_2_4_10

CSR2#show mpls traffic-eng tunnels tunnel 200 | section RSVP Path
  RSVP Path Info:
    My Address: 94.2.4.2
    Explicit Route: 94.2.4.4 94.4.10.10 10.10.10.10
    Record Route: NONE
    Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits

```

Following the route recursion, CSR2 will receive IPv4 traffic within VRF U1 from CSR6 to XRv11 and use label 5014. CSR2 has a route to CSR5's loopback (VPNv4 next-hop) via 10.10.10.10, which hasn't changed. This includes BGP label 10029. The route to 10.10.10.10 is now via the TE tunnel (thanks to autoroute) which imposes label 4019 from RSVP-TE. Having BGP here obviates the need for LDP to be enabled on this tunnel, since label 10029 directs us to the remote PE. Normally tLDP is required for this in intra-AS TE designs without BGP labeled-unicast. Since the BGP label will be exposed correctly to the tunnel tail-end, tLDP is not required. The label stack becomes {4019 10029 5014} towards CSR4.

```

CSR2#show bgp vpnv4 unicast vrf U1 1.11.11.11
BGP routing table entry for 49:102:1.11.11.11/32, version 74
Paths: (1 available, best #1, table U1)
  Advertised to update-groups:
    4
  Refresh Epoch 1

```

65000, (Received from a RR-client), imported path from 49:105:1.11.11.11/32 (global)

5.5.5.5 (metric 2) (via default) from 5.5.5.5 (5.5.5.5)
Origin incomplete, metric 0, localpref 100, valid, internal, best
Extended Community: RT:49:105
Connector Attribute: count=1
type 1 len 12 value 49:105:5.5.5.5
mpls labels in/out nolaabel/5014
rx pathid: 0, tx pathid: 0x0

CSR2#show ip route 5.5.5.5

Routing entry for 5.5.5.5/32

Known via "bgp 94", distance 200, metric 0
Tag 49, type internal
Last update from 10.10.10.10 3d20h ago
Routing Descriptor Blocks:
* 10.10.10.10, from 10.10.10.10, 3d20h ago
Route metric is 0, traffic share count is 1
AS Hops 1
Route tag 49
MPLS label: 10029

CSR2#show ip route 10.10.10.10

Routing entry for 10.10.10.10/32

Known via "ospf 94", distance 110, metric 2, type intra area
Last update from 10.10.10.10 on Tunnel200, 00:00:38 ago
Routing Descriptor Blocks:
* 10.10.10.10, from 10.10.10.10, 00:00:38 ago, via Tunnel200
Route metric is 2, traffic share count is 1

CSR2#show mpls traffic-eng tunnels tunnel 200 | include Label

InLabel : -
OutLabel : GigabitEthernet2.524, 4019

When CSR4 receives this packet, it pops the top-most label so the stack becomes {10029 5014}. This packet is then sent towards CSR10.

CSR4#show mpls traffic-eng tunnels destination 10.10.10.10 | include Label

InLabel : GigabitEthernet2.524, 4019
OutLabel : GigabitEthernet2.540, implicit-null

CSR8 performs a BGP label swap to use CSR8's label towards 5.5.5.5/32 and sends the packet to CSR8. At this point, the LSP is as we traced it before.

CSR10#show mpls forwarding-table labels 10029

Local	Outgoing	Prefix	Bytes	Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched		interface	
10029	8028	5.5.5.5/32	22914981		Gi2.580	192.168.8.8

```

CSR10#show bgp ipv4 unicast 5.5.5.5/32
BGP routing table entry for 5.5.5.5/32, version 10
Paths: (1 available, best #1, table default)
  Advertised to update-groups:
    3
  Refresh Epoch 1
  49
    192.168.8.8 from 192.168.8.8 (8.8.8.8)
      Origin incomplete, localpref 100, valid, external, best
      mpls labels in/out 10029/8028
      rx pathid: 0, tx pathid: 0x0

```

A quick traceroute shows us that the traffic still works from within the VPN, except now we see a 3-label stack at CSR4, which normally wasn't in the transit path. This isn't CSC-specific but the point is to show that CSC architectures support TE for intra-AS transport within the tier 2 customer carrier islands.

```

CSR6#traceroute vrf U1 ip
Target IP address: 1.11.11.11
Source address: 1.6.6.6
[snip]
Tracing the route to 1.11.11.11
VRF info: (vrf in name/id, vrf out name/id)
 1 192.168.102.2 1 msec 0 msec 0 msec
 2 94.2.4.4 [MPLS: Labels 4019/10029/5014 Exp 0] 20 msec 17 msec 16 msec
 3 94.4.10.10 [MPLS: Labels 10029/5014 Exp 0] 27 msec 33 msec 30 msec
 4 192.168.8.8 [MPLS: Labels 8028/5014 Exp 0] 62 msec 41 msec 34 msec
 5 49.1.8.1 [MPLS: Labels 0/1027/5014 Exp 0] 33 msec 55 msec 50 msec
 6 49.1.8.1 [MPLS: Labels 0/1027/5014 Exp 0] 45 msec 33 msec 34 msec
 7 192.168.105.5 [MPLS: Label 5014 Exp 0] 17 msec 17 msec 17 msec
 8 192.168.105.11 24 msec * 39 msec

```

One of the issues with this approach is that PIM-based MVPN is broken to CSR2. During this test, CSR2 shows many angry syslog messages to show that the VRF-aware PIM sessions for all MVPN instances have failed. Checking CSR2 and CSR5, we see a strange asymmetry of PIM neighbors. CSR5 sees CSR2 as a neighbor but not vice versa.

```

! CSR2
%PIM-5-NBRCHG: VRF U1: neighbor 5.5.5.5 DOWN on interface Tunnel3 DR
%PIM-5-NBRCHG: VRF U2: neighbor 5.5.5.5 DOWN on interface Tunnel4 DR

```

```

CSR5#show ip pim vrf U1 neighbor | begin ^Neighbor
Neighbor          Interface          Uptime/Expires    Ver    DR
Address
192.168.105.11    GigabitEthernet2.105 3d21h/00:01:40    v2     1 / DR P G
4.4.4.4           Tunnel3             3d20h/00:01:43    v2     1 / S P G
2.2.2.2           Tunnel3             3d21h/00:01:15    v2     1 / S P G

```

```

CSR2#show ip pim vrf U1 neighbor | begin ^Neighbor
Neighbor          Interface          Uptime/Expires    Ver    DR
Address
192.168.102.6      GigabitEthernet2.102  3d20h/00:01:25    v2     1 / DR S P G

```

Similar issues were observed in the inter-AS MPLS and MVPN sections. The reason for these errors is a newly-introduced RPF failure due to our TE tunnel. CSR2 can still send traffic to CSR5, who receives the tunneled PIM hellos, which is sufficient to declare a PIM neighbor up. However, CSR2 drops the ingress PIM hellos due to RPF failure. This is because the unicast route to 5.5.5.5 is via the TE tunnel, which is not PIM enabled. It isn't a VRF multicast issue since there is still a PIM tunnel within the VRF (despite the PIM neighbor being broken at the moment). Earlier we saw that, once the route recursion unwinds, the path to 5.5.5.5 is via the TE tunnel.

```

CSR2#show ip rpf 5.5.5.5
failed, no route exists

```

```

CSR2#show ip rpf vrf U1 1.11.11.11
RPF information for ? (1.11.11.11)
RPF interface: Tunnel3
RPF neighbor: ? (5.5.5.5)
RPF route/mask: 1.11.11.11/32
RPF type: unicast (bgp 94)
Doing distance-preferred lookups across tables
BGP originator: 5.5.5.5
RPF topology: ipv4 multicast base, originated from ipv4 unicast base

```

We can't realistically enable PIM on a TE tunnel because that logically makes no sense (it only makes sense in passive mode for P2MP TE, which we are not doing here). Instead, we can tell OSPF to ignore these TE tunnels as RPF interfaces. The unicast routing for MPLS LSPs still uses the TE tunnels, but MVPN profile 0 GRE encapsulation can use whatever interface OSPF would have normally selected without considering TE. Now, RPF passes and the PIM neighbors work again. CEF still directs unicast traffic into the TE tunnel as expected which means the unicast LSP remains the same.

```

! CSR2
router ospf 94
mpls traffic-eng multicast-intact

```

```

CSR2#show ip rpf 5.5.5.5
RPF information for ? (5.5.5.5)
RPF interface: GigabitEthernet2.520
RPF neighbor: ? (94.2.10.10)
RPF route/mask: 5.5.5.5/32
RPF type: unicast (bgp 94)
Doing distance-preferred lookups across tables
RPF topology: ipv4 multicast base, originated from ipv4 unicast base

```



```
CSR2#show ip cef 5.5.5.5
5.5.5.5/32
  nexthop 10.10.10.10 Tunnel200 label 10029
```

We quickly test multicast again to ensure it works while coexisting with the TE tunnel. Even though we are sending traffic from right to left, the joins (signaling) traversed from left to right, which implies that the RPF path from CSR2's perspective is functional.

```
CSR6#ping vrf U1
Protocol [ip]:
Target IP address: 232.1.1.1
Repeat count [1]: 10
Datagram size [100]:
Timeout in seconds [2]:
Extended commands [n]: y
Interface [All]: loopback1
Time to live [255]:
Source address or interface: loopback1
[snip]
Sending 10, 100-byte ICMP Echos to 232.1.1.1, timeout is 2 seconds:
Packet sent with a source address of 1.6.6.6
```

```
Reply to request 0 from 1.11.11.11, 26 ms
[snip]
Reply to request 9 from 1.11.11.11, 21 ms
```

We can also test TE in the core carrier, and for variety, I couple it with TE-FRR. To make this interesting and complex, I will use inbound local preference on XRv12 in the core carrier to draw traffic out of the AS using XRv12. Specifically, this will change how AS 49 reaches 5.5.5.5/32 within VRF T2. The current best-path is selected as CSR1 due to a shorter AS path length, so adjusting MED won't help. We will use this as a temporary measure to create a longer LSP through the core carrier, which makes TE-FRR more interesting.

```
! XRv12
route-policy RPL_IN
  set local-preference 200
end-policy
```

Now, CSR9 selects XRv12 as the next-hop for 5.5.5.5/32 inside the CSC VPN. This makes the LSP much longer since CSR1, CSR8, and CSR3 are now in the transit path assuming we use LDP transport. A quick OAM check from CSR9 to XRv12 confirms this long dynamic path.

```
CSR9#show bgp vpnv4 unicast vrf T2 5.5.5.5/32
BGP routing table entry for 49:9:5.5.5.5/32, version 31
Paths: (2 available, best #1, table T2)
```

Advertised to update-groups:

5

Refresh Epoch 1

94, imported path from 49:12:5.5.5.5/32 (global)

12.12.12.12 (metric 40) (via default) from 12.12.12.12 (12.12.12.12)

Origin IGP, metric 3, localpref 200, valid, internal, best

Extended Community: RT:49:12

Connector Attribute: count=1

type 1 len 12 value 49:12:12.12.12.12

mpls labels in/out 9004/92001

rx pathid: 0, tx pathid: 0x0

Refresh Epoch 1

Local, imported path from 49:1:5.5.5.5/32 (global)

1.1.1.1 (metric 10) (via default) from 12.12.12.12 (12.12.12.12)

Origin incomplete, metric 3, localpref 100, valid, internal

Extended Community: RT:49:1 OSPF DOMAIN ID:0x0005:0x00000005E0200

OSPF RT:0.0.0.0:2:0 OSPF ROUTER ID:0.0.0.1:0

Originator: 1.1.1.1, Cluster list: 12.12.12.12

Connector Attribute: count=1

type 1 len 12 value 49:1:1.1.1.1

mpls labels in/out 9004/1027

rx pathid: 0, tx pathid: 0

CSR9#traceroute mpls ipv4 12.12.12.12/32 source 9.9.9.9

more work needed here to demux the tfs subtlv and to display the right output
[snip]

Type escape sequence to abort.

```
0 49.1.9.9 MRU 1500 [Labels: 1019 Exp: 0]
L 1 49.1.9.1 MRU 1500 [Labels: 8014 Exp: 0] 2 ms
L 2 49.1.8.8 MRU 1500 [Labels: 3001 Exp: 0] 28 ms
L 3 49.3.8.3 MRU 1500 [Labels: implicit-null Exp: 0] 26 ms
! 4 49.3.12.12 29 ms
```

Before building any TE tunnels, we will do a quick TED check to ensure all core links are TE-enabled. We can see all five routers, each with their full complement of links. This quick check can alleviate complicated troubleshooting later.

CSR8#show mpls traffic-eng topology brief | include IGP Id

```
IGP Id: 0000.0000.0001.00, MPLS TE Id:1.1.1.1 Router Node (isis level-2)
  link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0009.00, nbr_node_id:22, gen:107
  link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0008.00, nbr_node_id:21, gen:107
IGP Id: 0000.0000.0003.00, MPLS TE Id:3.3.3.3 Router Node (isis level-2)
  link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0008.00, nbr_node_id:21, gen:110
  link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0012.00, nbr_node_id:28, gen:110
IGP Id: 0000.0000.0008.00, MPLS TE Id:8.8.8.8 Router Node (isis level-2)
  link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0009.00, nbr_node_id:22, gen:108
  link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0001.00, nbr_node_id:23, gen:108
  link[2]: Point-to-Point, Nbr IGP Id: 0000.0000.0003.00, nbr_node_id:30, gen:108
```

```

link[3]: Point-to-Point, Nbr IGP Id: 0000.0000.0012.00, nbr_node_id:28, gen:108
IGP Id: 0000.0000.0009.00, MPLS TE Id:9.9.9.9 Router Node (isis level-2)
link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0001.00, nbr_node_id:23, gen:109
link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0008.00, nbr_node_id:21, gen:109
IGP Id: 0000.0000.0012.00, MPLS TE Id:12.12.12.12 Router Node (isis level-2)
link[0]: Point-to-Point, Nbr IGP Id: 0000.0000.0003.00, nbr_node_id:30, gen:105
link[1]: Point-to-Point, Nbr IGP Id: 0000.0000.0008.00, nbr_node_id:21, gen:105

```

Next, we will build a TE tunnel from CSR9 to CSR3 following the path via CSR8 (high-cost link). The tunnel also requests 4 Mbps of bandwidth, although this isn't specific to CSC. We confirm the tunnel comes up and recurses over the proper path. The TE tunnel requests FRR protection but there are no backup tunnels yet. We also have to run LDP on this tunnel because it is PE-P, so CSR9 needs to push CSR3's LDP label for 12.12.12.12/32 before the TE transport label. We could potentially use BGP labeled-unicast for this as an extreme workaround, but there is no good reason to run this in the core carrier when LDP can do the job with a targeted session.

```

! CSR9
ip explicit-path name EP_9_8_3 enable
  next-address 8.8.8.8
  next-address 3.3.3.3

interface Tunnel201
  description BASIC TE TO CSR3
  ip unnumbered Loopback0
  mpls ip
  tunnel mode mpls traffic-eng
  tunnel destination 3.3.3.3
  tunnel mpls traffic-eng autoroute announce
  tunnel mpls traffic-eng priority 7 7
  tunnel mpls traffic-eng bandwidth 4000
  tunnel mpls traffic-eng path-option 10 explicit name EP_9_8_3
  tunnel mpls traffic-eng fast-reroute

CSR9#show mpls traffic-eng tunnels tunnel 201 | section RSVP Path
  RSVP Path Info:
    My Address: 49.8.9.9
    Explicit Route: 49.8.9.8 49.3.8.3 3.3.3.3
    Record Route: NONE
    Tspec: ave rate=4000 kbits, burst=1000 bytes, peak rate=4000 kbits

```

Before continuing, we already know that this TE tunnel is going to break the MDT traffic since the RPF process won't have a route to 12.12.12.12 via a PIM enabled interface. We can instruct IS-IS to ignore the TE tunnel when doing RPF checks just like we did for OSPF in the customer carrier; the command is identical.

```

! CSR9
router isis 49

```

```
mpls traffic-eng multicast-intact
```

CSR9 will use auto-tunnel backup to automatically build NHOP tunnels for TE LSPs traversing through it that request FRR protection. We can see that tunnel210 was spawned to protect the original LSP. CSR9 is also the tunnel head, but we want to protect the high-cost link so we can use FRR via CSR1 to reach CSR8. If FRR is in use, label 1000 is pushed atop the stack to use CSR1 as a backup path.

```
! CSR9
mpls traffic-eng auto-tunnel backup nhop-only
mpls traffic-eng auto-tunnel backup tunnel-num min 210 max 219
```

```
CSR9#show mpls traffic-eng tunnels backup
CSR9_t210
  LSP Head, Admin: up, Oper: up
  Tun ID: 210, LSP ID: 1, Source: 9.9.9.9
  Destination: 8.8.8.8
  Fast Reroute Backup Provided:
    Protected i/fs: Gi2.589
    Protected LSPs/Sub-LSPs: 1, Active: 0
  Backup BW: any pool unlimited; inuse: 4000 kbps
  Backup flags: 0x0
```

We can confirm this on CSR9 by checking the FRR database. NHOP protection is available and the router is the tunnel head as well. We can see the original TE label of 8017 and the FRR label of 1000.

```
CSR9#show mpls traffic-eng fast-reroute database role head state ready
P2P Headend FRR information:
Protected tunnel      In-label Out intf/label    FRR intf/label    Status
-----
Tunnel201             Tun hd   Gi2.589:8017        Tu210:8017        ready
```

```
CSR9#show mpls traffic-eng tunnels tunnel 201 | include Label
InLabel   : -
OutLabel  : GigabitEthernet2.589, 8017
FRR OutLabel : Tunnel210, 8017
```

```
CSR9#show mpls traffic-eng tunnels tunnel 210 | include Label
InLabel   : -
OutLabel  : GigabitEthernet2.519, 1000
```

We can't use the RSVP RESV RRO to confirm this since the protection is local, but we can check the protected tunnel. We can see NHOP protection, along with the appropriate labels.

```
CSR9#show mpls traffic-eng tunnels tunnel 201 protection
BASIC TE TO CSR3
  LSP Head, Tunnel201, Admin: up, Oper: up
  Src 9.9.9.9, Dest 3.3.3.3, Instance 1
```

```

Fast Reroute Protection: Requested
Outbound: FRR Ready
Backup Tu210 to LSP nhop
Tu210: out i/f: Gi2.519, label: 1000
LSP signalling info:
Original: out i/f: Gi2.589, label: 8017, nhop: 49.8.9.8
With FRR: out i/f: Tu210, label: 8017
LSP bw: 4000 kbps, Backup level: any-unlim, type: any pool
Path Protection: None

```

We will quickly trace the transport label stack within the core. The route to 12.12.12.12 is learned via a TE tunnel via IS-IS, but since this is not the tunnel endpoint, CSR9 must push CSR3's label for 12.12.12.12, which is 3001. Then, the route to 3.3.3.3 is via a TE tunnel and is the destination, so the RSVP label is used. The stack becomes {8017 3001}, where 8017 is the tunnel label we saw earlier.

```

CSR9#show ip route 12.12.12.12
Routing entry for 12.12.12.12/32
Known via "isis", distance 115, metric 40, type level-2
Redistributing via isis 49
Last update from 3.3.3.3 on Tunnel201, 00:13:00 ago
Routing Descriptor Blocks:
* 3.3.3.3, from 12.12.12.12, 00:13:00 ago, via Tunnel201
  Route metric is 40, traffic share count is 1

CSR9#show mpls ldp bindings 12.12.12.12 32 neighbor 3.3.3.3
lib entry: 12.12.12.12/32, rev 145
remote binding: lsr: 3.3.3.3:0, label: 3001

CSR9#show ip route 3.3.3.3
Routing entry for 3.3.3.3/32
Known via "isis", distance 115, metric 30, type level-2
Redistributing via isis 49
Last update from 3.3.3.3 on Tunnel201, 00:13:03 ago
Routing Descriptor Blocks:
* 3.3.3.3, from 3.3.3.3, 00:13:03 ago, via Tunnel201
  Route metric is 30, traffic share count is 1

```

We can confirm the additional push operation of label 3001 by checking the FIB. The LFIB shows both labels explicitly, and traceroute proves it further.

```

CSR9#show ip cef 12.12.12.12
12.12.12.12/32
  nexthop 3.3.3.3 Tunnel201 label 3001

CSR9#show mpls forwarding-table 12.12.12.12 32 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
9033       3001      12.12.12.12/32  0            Tu201      point2point

```

```
MAC/Encaps=18/26, MRU=1496, Label Stack{8017 3001}, via Gi2.589
000C296AFFD5000C29E04F8481000E058847 01F5800000BB9000
No output feature configured
```

```
CSR9#traceroute 12.12.12.12 source 9.9.9.9
Type escape sequence to abort.
Tracing the route to 12.12.12.12
VRF info: (vrf in name/id, vrf out name/id)
 1 49.8.9.8 [MPLS: Labels 8017/3001 Exp 0] 11 msec 8 msec 11 msec
 2 49.3.8.3 [MPLS: Label 3001 Exp 0] 21 msec 20 msec 20 msec
 3 49.3.12.12 20 msec * 11 msec
```

Before continuing, we perform a quick traceroute inside VRF U1 from CSR6 to XRv11. We can see up to 4 labels in the stack in the core carrier as a result of the PE-P tunnel. Normally, there were only 3 labels, where the third label was the CSC transport label bound by LDP. The fourth label is a result of using a PE-P tunnel, requiring an LDP label plus a TE label.

```
CSR6#traceroute vrf U2 2.11.11.11 source 2.6.6.6
Type escape sequence to abort.
Tracing the route to 2.11.11.11
VRF info: (vrf in name/id, vrf out name/id)
 1 192.168.204.4 1 msec 0 msec 1 msec
 2 192.168.4.9 [MPLS: Labels 9034/5001 Exp 0] 32 msec 33 msec 33 msec
 3 49.8.9.8 [MPLS: Labels 8017/3001/92001/5001 Exp 0] 33 msec 33 msec 33msec
 4 49.3.8.3 [MPLS: Labels 3001/92001/5001 Exp 0] 42 msec 33 msec 33 msec
 5 49.3.12.12 [MPLS: Labels 92001/5001 Exp 0] 42 msec 33 msec 34 msec
 6 192.168.12.13 [MPLS: Labels 93000/5001 Exp 0] 50 msec 50 msec 50 msec
 7 94.7.13.7 [MPLS: Labels 7011/5001 Exp 0] 41 msec 42 msec 64 msec
 8 192.168.205.5 [MPLS: Label 5001 Exp 0] 36 msec 33 msec 33 msec
 9 192.168.205.11 33 msec * 30 msec
```

Before breaking the link between CSR8 and CSR9, we will quickly check to ensure BFD is enabled with TE-FRR bound to it. RSVP is configured to rely on BFD to determine when TE tunnels should be re-routed, which is critical for TE-FRR to work properly.

```
CSR9#show bfd neighbors ipv4 49.8.9.8 details | include Regist
Registered protocols: ISIS CEF TE/FRR FRR
```

```
CSR9#show ip rsvp hello
Hello:
  RSVP Hello for Fast-Reroute/Reroute: Disabled
  Statistics: Disabled
  BFD for Fast-Reroute/Reroute: Enabled
  RSVP Hello for Graceful Restart: Disabled
```

To break the link, I change the VLAN encapsulation on CSR8 so the link fails (not shown). CSR9 only has one explicit path option, so it can never recompute a new path, which gives us time to verify FRR. CSR9

shows the entry for tunnel201 as "active" which means FRR is in use; the tunnel protection details for the original LSP also show this. The backup tunnel label of 1000 is added to the existing label stack, which allows for NHOP facility backup (N:1) through CSR1. This tunnels traffic around the failed link to provide the original 4-label stack to CSR8.

```
CSR9#show mpls traffic-eng fast-reroute database role head state active
P2P Headend FRR information:
Protected tunnel      In-label Out intf/label    FRR intf/label    Status
-----
Tunnel201             Tun  hd   Gi2.589:8017      Tu210:8017        active
```

```
CSR9#show mpls traffic-eng tunnels tunnel 201 protection
BASIC TE TO CSR3
  LSP Head, Tunnel201, Admin: up, Oper: up
  Src 9.9.9.9, Dest 3.3.3.3, Instance 14
  Fast Reroute Protection: Requested
  Outbound: FRR Active
  Backup Tu210 to LSP nhop
  Tu210: out i/f: Gi2.519, label: 1000
  LSP signalling info:
    Original: out i/f: Gi2.589, label: 8017, nhop: 8.8.8.8
    With FRR: out i/f: Tu210, label: 8017
  LSP bw: 4000 kbps, Backup level: any-unlim, type: any pool
  Path Protection: None
```

Verifying traceroute from inside the VPN, we can see the full label stacks. When we activate TE-FRR, this will become 5 labels at CSR1, which is the deepest stack we have seen so far. From inner-most to outermost, I list them below as a general summary. I also color-code each label description to match the color of the label in the stack for clarity.

1. **VPNv4 label**: allocated by CSR5 to describe the route to 2.11.11.11. Stays the same for entire LSP.
2. **BGP label**: allocated by XRv12 to describe the route to 5.5.5.5. Appears as a VPNv4 label on the CSC PEs but uses IPv4 labeled-unicast to exchange it with the CSC CEs.
3. **LDP label**: allocated by CSR3 to describe the route to 12.12.12.12 through the TE tunnel.
4. **TE label**: allocated by CSR8 (RSVP-TE) to describe the route to 3.3.3.3, the TE tunnel destination.
5. **FRR label**: allocated by CSR1 (RSVP-TE) to describe the route to 8.8.8.8 for NHOP protection.

```
CSR6#traceroute vrf U2 2.11.11.11 source 2.6.6.6
Type escape sequence to abort.
Tracing the route to 2.11.11.11
VRF info: (vrf in name/id, vrf out name/id)
  1 192.168.204.4 1 msec 1 msec 0 msec
  2 192.168.4.9 [MPLS: Labels 9034/5001 Exp 0] 35 msec 33 msec 33 msec
  3 49.1.9.1 [MPLS: Labels 1000/8017/3001/92001/5001 Exp 0] 41 msec 37 msec
37 msec
```

```

4 49.1.8.8 [MPLS: Labels 8017/3001/92001/5001 Exp 0] 46 msec 27 msec 29
msec
5 49.3.8.3 [MPLS: Labels 3001/92001/5001 Exp 0] 36 msec 35 msec 33 msec
6 49.3.12.12 [MPLS: Labels 92001/5001 Exp 0] 33 msec 34 msec 50 msec
7 192.168.12.13 [MPLS: Labels 93000/5001 Exp 0] 64 msec 58 msec 68 msec
8 94.7.13.7 [MPLS: Labels 7011/5001 Exp 0] 48 msec 49 msec 50 msec
9 192.168.205.5 [MPLS: Label 5001 Exp 0] 35 msec 33 msec 41 msec
10 192.168.205.11 34 msec * 38 msec

```

Additional Reading – Reference configurations "csc"

33.2 Unified (seamless) MPLS

Unified MPLS is a technique to create end-to-end LSPs across different OSPF areas, different IS-IS levels, or totally independent IGPs without having to leak PE loopbacks between the disparate domains. For the purpose of this section an “end-to-end LSP” is an LSP whereby the bottom-most service label does not change. To achieve this (let’s use L3VPN as an example), the VPN routes must never have their BGP next-hops changed; doing so causes VPNv4/v6 BGP to allocate a new VPN label which terminates the LSP. Traffic can still be label-switched end-to-end, but having a consistent LSP simplifies things like MVPN (obviates need for PIM vector and BGP connector mechanisms). ***This technique DOES NOT introduce entirely new technologies, just new ideas with minor configuration modifications.***

Leaking loopbacks between areas, levels, or totally different IGPs within a domain is difficult to manage and scales poorly. Instead, IPv4 labeled-unicast BGP is used to transport these loopbacks between IGP islands so that specific route leaking is not necessary. The IGP border routers will be route-reflectors for this AFI/SAFI to service all PEs within the IGP island; P routers do not need to run IPv4 labeled-unicast as it is expected that LSPs will be tunneled across them using transport labels from RSVP-TE, LDP, or even Segment Routing.

The key is ensuring the border routers set “next-hop-self” on iBGP-reflected routes so that PEs in the IGP islands have reachability to it. The VPNv4 next-hops, however, should never be changed. We are essentially using this extra BGP-bound label to transport traffic to the remote PE and relying on the ABRs to swap this middle label.

Unified MPLS works on very old code, such as 12.4T which can be emulated in GNS3. The trick on old code is to manually use a route-map to set the next-hop to yourself (in some versions, “peer-address” also works) but also set the “mpls-label”. Because IOS platforms do not have a separate configuration construct for the IPv4 labeled-unicast SAFI, applying an outbound route-map for filtering in the IPv4 unicast SAFI also filters labels unless they are explicitly permitted. The “send-label” command just activates the IPv4 labeled-unicast SAFI but outbound filters may prevent those labels from being sent to a given neighbor. Without setting the “mpls-label”, the routes will appear everywhere, but the IPv4 labeled-unicast routes will look like regular IPv4 unicast routes without MPLS labels.

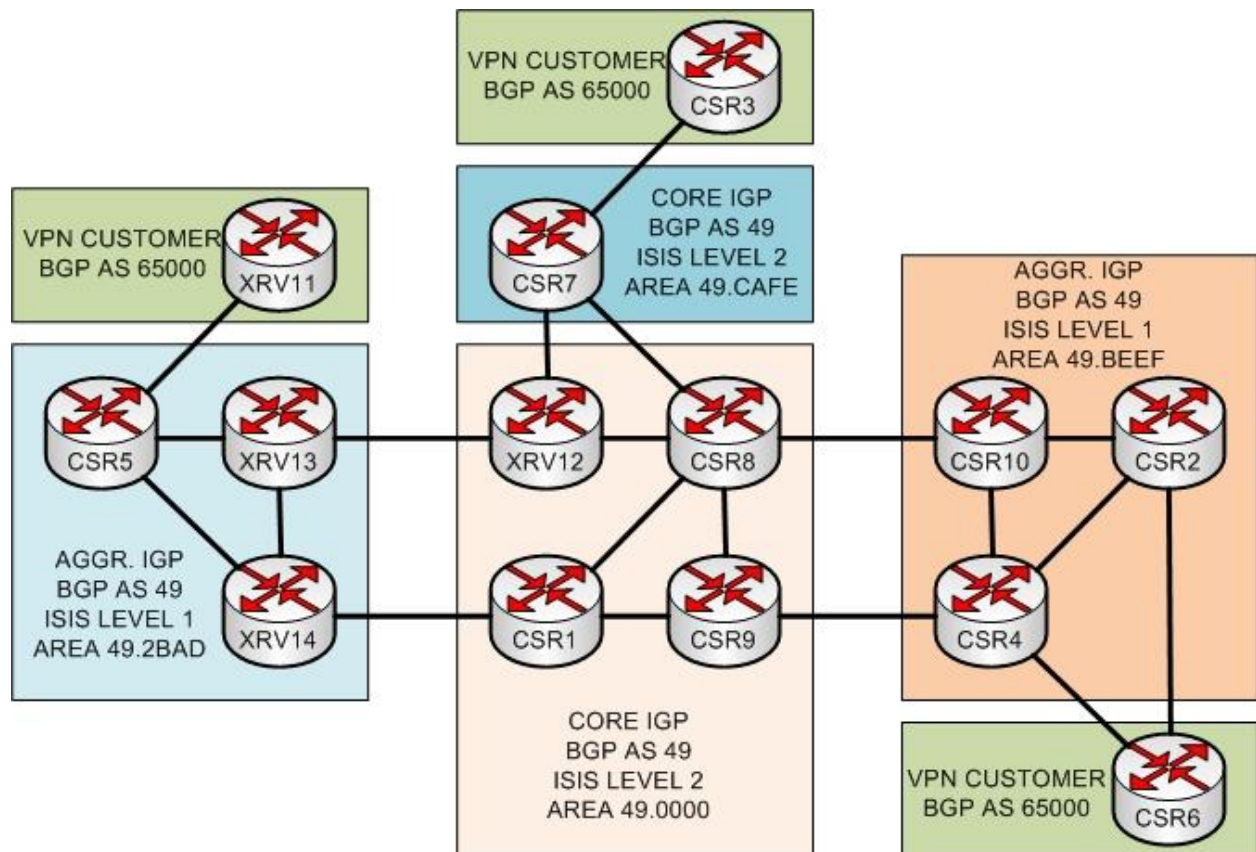
A typical label stack at imposition would look something like this:

1. Topmost: LDP or RSVP-TE allocated label to reach the ABR (changes hop by hop)
2. Middle: BGP IPv4 labeled-unicast label to reach the remote PE (changes at the ABRs)
3. Bottom: BGP VPNv4 label to reach the final destination (never changes)

Key notes:

- In IOS, use “next-hop-self all” on the ABRs, which are RRs, for the IPv4 labeled-unicast SAFI.
 - A route-map manually setting the next-hop to the peer-address or static IPv4 address also works as discussed earlier.
 - A route-map manually setting a static IPv4 address works too, same pitfall as above.
- In XR, use “ibgp policy out enforce-modifications” on the ABRs.
 - If you don’t, nothing effects the reflected iBGP prefixes, not even a manual RPL.
 - Once this is done, use ordinary “next-hop-self” on the iBGP peers.
 - An RPL with “next-hop self” also works.
 - Because XR has an explicit IPv4 labeled-unicast SAFI, you don’t need to manually set label allocation in an RPL match stanza unless you want to specify the label value (can’t do that in IOS or XE).
 - A manual cluster-ID appears necessary for the IPv4 labeled-unicast and VPNv4 SAFIs in order for RRs to operate. The BGP update debugs indicate a lack of a cluster ID otherwise.
- Like a classic L3VPN, only the true PEs need to run the VPN signaling protocol:
 - VPNv4, VPNv6, L2VPN variants, pseudo-wire, etc
- The ABRs must be RRs and must apply “next-hop-self” for IPv4 labeled-unicast SAFI
 - This should be in the both directions. We assume IGP routes are not leaked either direction. The examples below show some alternative interpretations of this rule.
- Mild leaking might be necessary, but it’s essentially one route per ABR, not per PE
 - ABR loopback in the IGP island should be leaked into the higher layer for BGP peering reachability
 - ABR loopback in the higher layer should be leaked into the IGP island, same reason
 - Recommend “redistribute connected” or a direct IGP advertisement
 - Redistributing from a protocol could be dangerous if the filter is wrong

Below is a network diagram for the IS-IS portion of the lab. The configurations shown below are universal (generally speaking) for both IS-IS or OSPF, so they are verified first.



The configurations are very extensive for this setup, so the key parts are summarized. The BGP topology is generally the same regardless of IGP, so configurations for that are shown first. Different "types" of UMPLS are tested here which explains the variance in configuration. The "service" topologies, such as L3VPN and L2VPN for customers, can be completely different than the transport topologies such as LDP and BGP IPv4 labeled-unicast. Below is the RR configuration on the L1L2 routers XRV13 and XRV14; reflected routes have their next-hops changed in both directions, which ensures traffic in both directions will transit these routers. It also guarantees the allocation of a local label for BGP IPv4 labeled-unicast, which is important for UMPLS scalability.

```
! XRV13 and XRV14
route-policy RPL_ADVERTISE_FROM_R5
  if next-hop in (5.5.5.5) then
    pass
  endif
end-policy

router bgp 49
  ibgp policy out enforce-modifications
  address-family ipv4 unicast
    network 13.13.13.13/32
    allocate-label all
```

```

af-group IPV4_LUCAST address-family ipv4 labeled-unicast
  route-policy RPL_ADVERTISE_FROM_R5 out
  next-hop-self

af-group IPV4_LUCAST_CLIENT address-family ipv4 labeled-unicast
  route-reflector-client
  next-hop-self

session-group IBGP_SESSION
  remote-as 49
  timers 10 40
  password encrypted 032D792C363000797A21
  update-source Loopback0

neighbor 5.5.5.5
  use session-group IBGP_SESSION
  address-family ipv4 labeled-unicast
  use af-group IPV4_LUCAST_CLIENT

neighbor 8.8.8.8
  use session-group IBGP_SESSION
  address-family ipv4 labeled-unicast
  use af-group IPV4_LUCAST

neighbor 12.12.12.12
  use session-group IBGP_SESSION
  address-family ipv4 labeled-unicast
  use af-group IPV4_LUCAST

```

The other set of ABRs have a slightly different (simpler but less scalable) policy. They do not change the next-hops when reflecting routes to the "core" RRs, which means they do not allocate new labels for these prefixes. For variety, I also show an alternative method to setting the next-hop by using a route-map. The pitfall discussed earlier would be to omit the "set mpls-label" command, which would allow the prefix to be processed by the route-map (permitted) but no label assigned. Minor differences appear between CSR4 and CSR10; since CSR4 is a PE also, its loopback is redistributed into BGP, but CSR10's is not. CSR4 uses the new "next-hop-self all" command rather than the clunky route-map, which was introduced specifically to support UMPLS in newer code releases. It always adjusts the BGP next-hop, even for reflected routes.

```

! CSR10
ip prefix-list PL_HOST_ROUTE seq 5 permit 0.0.0.0/0 ge 32

route-map RM_NHOP_SELF permit 10
  match ip address prefix-list PL_HOST_ROUTE
  set mpls-label
  set ip next-hop 10.10.10.10

```

```

router bgp 49
  template peer-policy IPV4_LUCAST
    send-community both
    send-label

  template peer-policy IPV4_LUCAST_CLIENT
    route-map RM_NHOP_SELF out
    route-reflector-client
    inherit peer-policy IPV4_LUCAST 10

  template peer-session IBGP_SESSION
    remote-as 49
    password IBGP_AUTH
    update-source Loopback0
    timers 10 40

  no bgp default ipv4-unicast
  neighbor 2.2.2.2 inherit peer-session IBGP_SESSION
  neighbor 8.8.8.8 inherit peer-session IBGP_SESSION
  neighbor 12.12.12.12 inherit peer-session IBGP_SESSION

  address-family ipv4
    neighbor 2.2.2.2 activate
    neighbor 2.2.2.2 inherit peer-policy IPV4_LUCAST_CLIENT
    neighbor 8.8.8.8 activate
    neighbor 8.8.8.8 inherit peer-policy IPV4_LUCAST
    neighbor 12.12.12.12 activate
    neighbor 12.12.12.12 inherit peer-policy IPV4_LUCAST

! CSR4
router bgp 49
  template peer-policy IPV4_LUCAST
    send-community both
    send-label

  template peer-policy IPV4_LUCAST_CLIENT
    route-reflector-client
    next-hop-self all
    inherit peer-policy IPV4_LUCAST 10

```

The "service" topology is very simple as CSR7 is the RR for VPNv4, VPNv6, IPv4 MVPN, IPv6 MVPN, and L2VPN VPLS. I omit the L3VPN BGP PE configurations since those are simple. I compress this configuration since it is highly redundant. CSR7 peers with the core RRs as well so it can advertise its loopback into BGP IPv4 labeled-unicast for UMPLS transport. This mechanism is the same technique used on CSR2, CSR4, and CSR5 as PE routers. The difference is that those only peer with CSR7 but negotiate the same AFls. CSR4 does not run VPLS, but otherwise, everything is the same across all PE routers.

```

! CSR7
route-map RM_CONN_TO_BGP permit 10
  match interface Loopback0

router bgp 49
  template peer-policy IPV4_LUCAST
    send-community both
    send-label

  template peer-session IBGP_SESSION
    remote-as 49
    password IBGP_AUTH
    update-source Loopback0
    timers 10 40

  no bgp default ipv4-unicast
  neighbor [CSR2, CSR4, CSR5, CSR8, and CSR12] inherit peer-session
  IBGP_SESSION

  address-family ipv4
    redistribute connected route-map RM_CONN_TO_BGP
    neighbor [CSR8 and CSR12] activate
    neighbor [CSR8 and CSR12] inherit peer-policy IPV4_LUCAST

  address-family ipv4 mvpn
    neighbor [CSR2, CSR4, and CSR5] activate
    neighbor [CSR2, CSR4, and CSR5] route-reflector-client

  address-family vpnv4
    neighbor [CSR2, CSR4, and CSR5] activate
    neighbor [CSR2, CSR4, and CSR5] route-reflector-client

  address-family ipv6 mvpn
    neighbor [CSR2, CSR4, and CSR5] activate
    neighbor [CSR2, CSR4, and CSR5] route-reflector-client

  address-family vpnv6
    neighbor [CSR2, CSR4, and CSR5] activate
    neighbor [CSR2, CSR4, and CSR5] route-reflector-client

  address-family l2vpn vpls
    neighbor [CSR2 and CSR5] activate
    neighbor [CSR2 and CSR5] activate

```

The core RRs peer with all ABRs in the topology, at a minimum. Due to the MVPN design, I also include the other core routers so that they have reachability to the P2MP roots (PE routers) in different areas. They would need BGP connectivity in order to see these loopbacks in a proper UMPLS design. In this case, I demonstrate policy inheritance in the event that a non-RR neighbor needs to be added in the

future. The configuration is very similar on XRv12 as well, except that the core routers CSR1 and CSR9 are not peered to it. For simplicity, I only configured them to peer with CSR8 as a quick way to repair MVPN P2MP tree issues.

```
! CSR8
route-map RM_CONN_TO_BGP permit 10
  match interface Loopback0

router bgp 49
  template peer-policy IPV4_LUCAST
    send-community both
    send-label

  template peer-policy IPV4_LUCAST_CLIENT
    route-reflector-client
    inherit peer-policy IPV4_LUCAST 10

  template peer-session IBGP_SESSION
    remote-as 49
    password IBGP_AUTH
    update-source Loopback0
    timers 10 40

  no bgp default ipv4-unicast
  neighbor [CSR1, CSR4, CSR7, CSR9, CSR10, XRv13, XRv14] inherit peer-session
  IBGP_SESSION

  address-family ipv4
    redistribute connected route-map RM_CONN_TO_BGP
    neighbor [CSR1, CSR4, CSR7, CSR9, CSR10, XRv13, XRv14] activate
    neighbor [CSR1, CSR4, CSR7, CSR9, CSR10, XRv13, XRv14] inherit peer-policy
    IPV4_LUCAST_CLIENT

! XRv12
router bgp 49
  address-family ipv4 unicast
    network 12.12.12.12/32
    allocate-label all

  af-group IPV4_LUCAST_CLIENT address-family ipv4 labeled-unicast
    route-reflector-client

  session-group IBGP_SESSION
    remote-as 49
    timers 10 40
    password encrypted 032D792C363000797A21
    update-source Loopback0

  neighbor [CSR4, CSR7, CSR10, XRv13, XRv14]
```

```

use session-group IBGP_SESSION
address-family ipv4 labeled-unicast
use af-group IPV4_LUCAST_CLIENT

```

CSR6 is a CE router which is multi-homed to two PE routers, CSR4 and CSR2. It uses local preference to select best paths for traffic sent towards the SP network. For simplicity, it prefers CSR2 for VRF U1 and CSR4 for VRF U2. This is true for both IPv4 and IPv6 AFIs. The other BGP configurations are omitted, but this path-selection modification is important. This was shown in the CSC section but is repeated for clarity.

```

! CSR6
route-map RM_LPREF_200 permit 10
set local-preference 200

router bgp 65000
address-family ipv4 vrf U1
neighbor 192.168.102.2 route-map RM_LPREF_200 in

address-family ipv6 vrf U1
neighbor FD00:192:168:102::2 route-map RM_LPREF_200 in

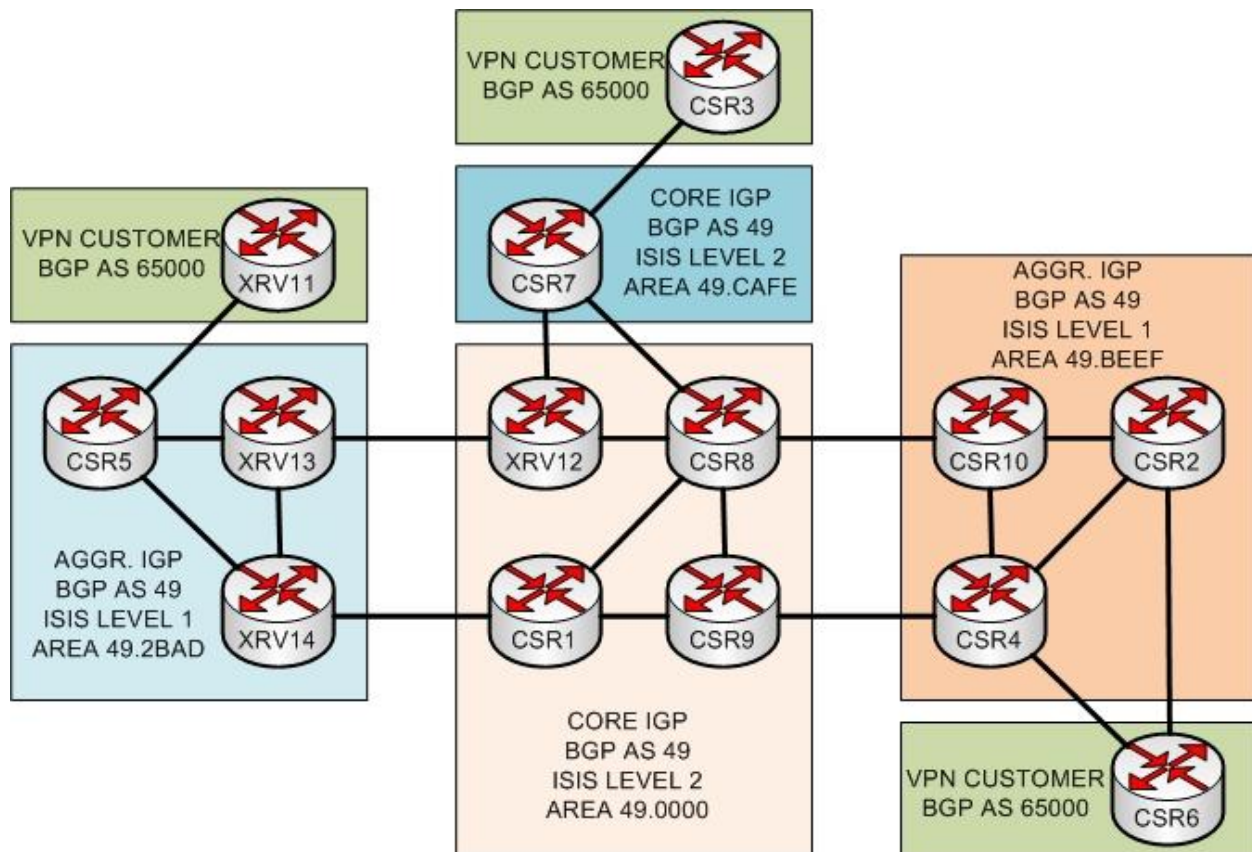
address-family ipv4 vrf U2
neighbor 192.168.204.4 route-map RM_LPREF_200 in

address-family ipv6 vrf U2
neighbor FD00:192:168:204::4 route-map RM_LPREF_200 in

```

33.2.1 IS-IS

This large demonstration shows many MPLS services over unified/seamless MPLS. The network is split into different IS-IS areas/levels with various levels of filtering at the L1/L2 routers. CSR5, CSR7, CSR2, and CSR4 are the PEs that offer VPNv4, VPNv6, VPLS, and MVPN services. The lab is very configuration-intensive so only certain pieces of configuration are shown. The network diagram is shown below, which is identical to the one used for the basic UMPLS verifications above.



Because this is such a complex topology, we will verify all of the individual components before beginning as a best practice. A very fast way of checking IS-IS for configuration errors is to check the IS-IS data on an L1/L2 router at each area boundary. They all agree on the L2 topology (at least, the number of link-state packets). We can check this against the diagram along with the independent L1 topologies in each area. CSR7 is an L2-only router since there are no other areas in 49.CAFE in which to form an L1 adjacency. Each L1/L2 router shows the proper routers in each level, so we can continue. All we are doing is checking the router hostnames for this cursory check.

```
RP/0/0/CPU0:XRv13#show isis database
```

```
IS-IS 49 (Level-1) Link State Database
```

LSPID	LSP Seq Num	LSP Checksum	LSP Holdtime	ATT/P/OL
CSR5.00-00	0x000001d5	0x5af5	689	0/0/0
XRv13.00-00	* 0x000001cc	0x5b4e	1139	0/0/0
XRv14.00-00	0x000001c9	0x5b49	1118	0/0/0

Total Level-1 LSP count: 3

Local Level-1 LSP count: 1

```
IS-IS 49 (Level-2) Link State Database
```

LSPID	LSP Seq Num	LSP Checksum	LSP Holdtime	ATT/P/OL
CSR1.00-00	0x000001ce	0x8b22	512	0/0/0
CSR4.00-00	0x000001d7	0x4bb2	455	0/0/0
CSR7.00-00	0x000001cc	0xeb89	502	0/0/0

CSR8.00-00	0x000001de	0xdc94	613	0/0/0
CSR9.00-00	0x000001d7	0x945e	1160	0/0/0
CSR10.00-00	0x000001d5	0x4554	1112	0/0/0
XRv12.00-00	0x000001ca	0x1e3f	921	0/0/0
XRv13.00-00	* 0x000001cb	0x15d2	488	0/0/0
XRv14.00-00	0x000001cb	0xb168	563	0/0/0

Total Level-2 LSP count: 9 Local Level-2 LSP count: 1

CSR10#show isis database

Tag 49:

IS-IS Level-1 Link State Database:

LSPID	LSP Seq Num	LSP Checksum	LSP Holdtime	ATT/P/OL
CSR2.00-00	0x000001D8	0x893D	1059	0/0/0
CSR4.00-00	0x000001D4	0x196A	951	1/0/0
CSR10.00-00	* 0x000001D1	0x1B0F	749	1/0/0

IS-IS Level-2 Link State Database:

LSPID	LSP Seq Num	LSP Checksum	LSP Holdtime	ATT/P/OL
CSR1.00-00	0x000001CE	0x8B22	420	0/0/0
CSR4.00-00	0x000001D8	0x49B3	1192	0/0/0
CSR7.00-00	0x000001CC	0xEB89	409	0/0/0
CSR8.00-00	0x000001DE	0xDC94	522	0/0/0
CSR9.00-00	0x000001D7	0x945E	1069	0/0/0
CSR10.00-00	* 0x000001D5	0x4554	1024	0/0/0
XRv12.00-00	0x000001CA	0x1E3F	827	0/0/0
XRv13.00-00	0x000001CB	0x15D2	392	0/0/0
XRv14.00-00	0x000001CB	0xB168	472	0/0/0

We will check LDP adjacencies quickly by selecting three routers spread throughout the network. This is imprecise, but is a good spot check. CSR8 has many links and covers most of the core, while CSR2 and CSR5 exist at the network extremities. Since VPLS with LDP signaling is enabled, we get the added benefit of verifying targeted LDP sessions on CSR2 and CSR5, which connect to CSR7 in a triangle. Those sessions are highlighted for clarity. CSR4 does not participate in VPLS.

CSR8#show mpls ldp neighbor | include Peer

```

Peer LDP Ident: 9.9.9.9:0; Local LDP Ident 8.8.8.8:0
Peer LDP Ident: 1.1.1.1:0; Local LDP Ident 8.8.8.8:0
Peer LDP Ident: 7.7.7.7:0; Local LDP Ident 8.8.8.8:0
Peer LDP Ident: 12.12.12.12:0; Local LDP Ident 8.8.8.8:0
Peer LDP Ident: 10.10.10.10:0; Local LDP Ident 8.8.8.8:0

```

CSR5#show mpls ldp neighbor | include Peer

```

Peer LDP Ident: 13.13.13.13:0; Local LDP Ident 5.5.5.5:0
Peer LDP Ident: 14.14.14.14:0; Local LDP Ident 5.5.5.5:0
Peer LDP Ident: 7.7.7.7:0; Local LDP Ident 5.5.5.5:0
Peer LDP Ident: 2.2.2.2:0; Local LDP Ident 5.5.5.5:0

```

CSR2#show mpls ldp neighbor | include Peer

```

Peer LDP Ident: 4.4.4.4:0; Local LDP Ident 2.2.2.2:0
Peer LDP Ident: 10.10.10.10:0; Local LDP Ident 2.2.2.2:0
Peer LDP Ident: 7.7.7.7:0; Local LDP Ident 2.2.2.2:0
Peer LDP Ident: 5.5.5.5:0; Local LDP Ident 2.2.2.2:0

```

Next, we will check IPv4 labeled-unicast BGP. This was verified in the general UMPLS verification but we quickly check it again. This is a key component of UMPLS since it allows disjoint IGP islands to communicate with one another. Using BGP to distribute VPN next-hops scales better than IGP redistribution and comes at the low cost of one additional MPLS shim header. Since XRv2 and CSR8 are the RRs for this AFI, we will check their session counts. So long as those routers are receiving one prefix from each peer, we can assume things are working to some extent. CSR1 and CSR9 are just core routers but are running IPv4 labeled-unicast as well. This was done as a workaround for UMPLS mLDP and is discussed later and has to do with tree construction between the islands. The reason there is only 1 prefix is because there is only 1 PE per island in this design, which is the only loopback that needs to be advertised into the core. In a real design, there would be many more PEs since the primary use-case of UMPLS is high scalability. It is assumed that the core has reachability to the ABR loopbacks via the core IGP.

```

CSR8#show bgp ipv4 unicast summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ  Up/Down  State/PfxRcd
1.1.1.1        4      49    158    168     23    0     0 00:24:45      1
4.4.4.4        4      49   37338   37378     23    0     0 4d02h        1
7.7.7.7        4      49   37366   37384     23    0     0 4d02h        1
9.9.9.9        4      49    159    166     23    0     0 00:24:36      1
10.10.10.10    4      49   35446   35480     23    0     0 3d21h        1
13.13.13.13    4      49   35480   37370     23    0     0 4d02h        1
14.14.14.14    4      49   35478   37380     23    0     0 4d02h        1

```

```

RP/0/0/CPU0:XRv12#show bgp ipv4 labeled-unicast summary | begin ^Neighbor
Neighbor      Spk      AS MsgRcvd MsgSent  TblVer  InQ  OutQ  Up/Down  St/PfxRcd
4.4.4.4        0      49   37372   35512     29    0     0 3d21h        1
7.7.7.7        0      49   37364   35517     29    0     0 3d21h        1
10.10.10.10    0      49   35668   33895     29    0     0 3d21h        1
13.13.13.13    0      49   35492   35493     29    0     0 3d21h        1
14.14.14.14    0      49   35490   35490     29    0     0 3d21h        1

```

The labeled-unicast service is critical to the operation of UMPLS. Since XRv12 is running labeled-unicast as evidenced by the show command above, CSR8 may not be, so we can verify this negotiation was successful by checking the neighbor details.

```

CSR8#show bgp ipv4 unicast neighbors | include ^BGP|Label_cap
BGP neighbor is 1.1.1.1, remote AS 49, internal link
  ipv4 MPLS Label capability: advertised and received
BGP neighbor is 4.4.4.4, remote AS 49, internal link
  ipv4 MPLS Label capability: advertised and received
BGP neighbor is 7.7.7.7, remote AS 49, internal link
  ipv4 MPLS Label capability: advertised and received
BGP neighbor is 9.9.9.9, remote AS 49, internal link

```

```

    ipv4 MPLS Label capability: advertised and received
BGP neighbor is 10.10.10.10, remote AS 49, internal link
    ipv4 MPLS Label capability: advertised and received
BGP neighbor is 13.13.13.13, remote AS 49, internal link
    ipv4 MPLS Label capability: advertised and received
BGP neighbor is 14.14.14.14, remote AS 49, internal link
    ipv4 MPLS Label capability: advertised and received

```

Finally, we will review the MPLS service-related BGP configurations. CSR7 is the route-reflector for all of these AFIs, and we check the neighbor status quickly. CSR7 is connected to and receiving "routes" from all BGP peers in the relevant service AFIs. Another alternative (and a better design for scalability) is to have an RR in each IGP island. This means that the individual islands are "self-sufficient" since the RRs will peer together and still service their intra-island PEs. In this design, we minimize the number of RRs while increasing dependence on the core network.

```

CSR7#show bgp vpnv4 unicast all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State/PfxRcd
2.2.2.2       4       49   347   392    166   0   0 00:52:16      2
4.4.4.4       4       49   350   376    166   0   0 00:52:13      2
5.5.5.5       4       49   348   401    166   0   0 00:52:19      2
192.168.107.3 4      65000 6282  6310    166   0   0 3d23h         1
192.168.207.3 4      65000 6286  6319    166   0   0 3d23h         1

```

```

CSR7#show bgp vpnv6 unicast all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State/PfxRcd
2.2.2.2       4       49   347   392    402   0   0 00:52:19      2
4.4.4.4       4       49   350   377    402   0   0 00:52:17      2
5.5.5.5       4       49   348   402    402   0   0 00:52:22      2
FD00:192:168:107::3
                4      65000  6335  6311    402   0   0 3d23h         1
FD00:192:168:207::3
                4      65000  6337  6312    402   0   0 3d23h         1

```

```

CSR7#show bgp ipv4 mvpn all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State/PfxRcd
2.2.2.2       4       49   351   399     14   0   0 00:52:59      1
4.4.4.4       4       49   354   383     14   0   0 00:52:57      1
5.5.5.5       4       49   352   409     14   0   0 00:53:02      1

```

```

CSR7#show bgp ipv6 mvpn all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State/PfxRcd
2.2.2.2       4       49   352   399     16   0   0 00:53:02      1
4.4.4.4       4       49   354   383     16   0   0 00:53:00      1
5.5.5.5       4       49   353   409     16   0   0 00:53:05      1

```

```

CSR7#show bgp l2vpn vpls all summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down  State/PfxRcd
2.2.2.2       4       49   353   400     26   0   0 00:53:11      1
5.5.5.5       4       49   354   410     26   0   0 00:53:14      1

```

With the basic control-plane verifications complete, we will detail the UMPLS transport architecture. We will begin with CSR5. Examining the IS-IS LSP in detail, we can see that CSR5's IPv4 loopback is advertised into IGP. This means everyone in the IGP island (in this case, IS-IS level-1) will have reachability to it. It also means they should use an LDP label when sending MPLS-encapsulated traffic towards it. CSR5 allocates implicit-null for this prefix since it is directly connected.

```
CSR5#show isis database level-1 CSR5.00-00 detail | begin Hostname
Hostname: CSR5
Metric: 10          IS-Extended XRv13.00
Metric: 10          IS-Extended XRv14.00
Metric: 10          IS (MT-IPv6) XRv13.00
Metric: 10          IS (MT-IPv6) XRv14.00
IP Address: 5.5.5.5
Metric: 0           IP 5.5.5.5/32
```

```
CSR5#show mpls ldp bindings 5.5.5.5 32 local
lib entry: 5.5.5.5/32, rev 2
local binding: label: imp-null
```

Since the level-1 topology only contains two other connected routers, both XRv13 and XRv14 will use this label to reach 5.5.5.5. Breaking links or changing IGP costs within the L1 topology can change this behavior as we will see later.

```
RP/0/0/CPU0:XRv13#show cef ipv4 5.5.5.5 | utilit egrep 'via|labels'
via 49.5.13.5, GigabitEthernet0/0/0/0.553, 5 dependencies, weight 0, class
0 [flags 0x0]
local label 93000 labels imposed {ImplNull}
```

```
RP/0/0/CPU0:XRv14#show cef ipv4 5.5.5.5 | utilit egrep 'via|labels'
via 49.5.14.5, GigabitEthernet0/0/0/0.554, 5 dependencies, weight 0, class
0 [flags 0x0]
local label 94008 labels imposed {ImplNull}
```

CSR5 also advertises this loopback into BGP. Although the ABRs could have done this advertisement, doing it on the PEs is simpler to manage, and the ABR will prefer the IGP route over the iBGP route anyway. As with LDP, CSR5 allocates implicit-null for this prefix in BGP.

```
CSR5#show bgp ipv4 unicast 5.5.5.5/32
BGP routing table entry for 5.5.5.5/32, version 2
Paths: (1 available, best #1, table default)
Advertised to update-groups:
1
Refresh Epoch 1
Local
0.0.0.0 from 0.0.0.0 (5.5.5.5)
Origin incomplete, metric 0, localpref 100, weight 32768, valid,
sourced, best
```

```
mpls labels in/out imp-null/nolabel
rx pathid: 0, tx pathid: 0x0
```

We confirm that the ABRs learn this prefix and corresponding implicit-null label (value of 3). The reason XRv14 has alternate paths via CSR8 and XRv12 is because they are RRs. CSR8 selected XRv13 as the best path due to having a lower neighbor ID than XRv14, then reflected the best-path down to all clients. XRv12 selected XRv13 because the IGP cost to 13.13.13.13 was lower than 14.14.14.14. This isn't significant, but a good observation.

```
RP/0/0/CPU0:XRv13#show bgp ipv4 labeled-unicast 5.5.5.5/32 | begin Local,
Local, (Received from a RR-client)
5.5.5.5 (metric 10) from 5.5.5.5 (5.5.5.5)
Received Label 3
Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best
Received Path ID 0, Local Path ID 1, version 3
```

```
RP/0/0/CPU0:XRv14#show bgp ipv4 labeled-unicast 5.5.5.5/32 | begin Local,
Local, (Received from a RR-client)
5.5.5.5 (metric 10) from 5.5.5.5 (5.5.5.5)
Received Label 3
Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best
Received Path ID 0, Local Path ID 1, version 3
Path #2: Received by speaker 0
Not advertised to any peer
Local
13.13.13.13 (metric 10) from 8.8.8.8 (5.5.5.5)
Received Label 93000
Origin incomplete, metric 0, localpref 100, valid, internal
Received Path ID 0, Local Path ID 0, version 0
Originator: 5.5.5.5, Cluster list: 8.8.8.8, 13.13.13.13
Path #3: Received by speaker 0
Not advertised to any peer
Local
13.13.13.13 (metric 10) from 12.12.12.12 (5.5.5.5)
Received Label 93000
Origin incomplete, metric 0, localpref 100, valid, internal
Received Path ID 0, Local Path ID 0, version 0
Originator: 5.5.5.5, Cluster list: 12.12.12.12, 13.13.13.13
```

For completeness, though not directly related to UMPLS, we will verify the claims above. CSR8 doesn't tell us why it chose XRv13 over XRv14, but we can see its due to the lower neighbor ID. Normally, the RIDs would be compared, but when RRs exist (XRv13 and XRv14 are RRs for this AFI), the originator ID is compared, which is a tie. XRv12 clearly shows the difference in IGP metric and states so in plain English below the non-best-path.

```

CSR8#show bgp ipv4 unicast 5.5.5.5/32
BGP routing table entry for 5.5.5.5/32, version 18
Paths: (2 available, best #2, table default)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  Local, (Received from a RR-client)
    14.14.14.14 (metric 20) from 14.14.14.14 (14.14.14.14)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Originator: 5.5.5.5, Cluster list: 14.14.14.14
      mpls labels in/out 8004/94008
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  Local, (Received from a RR-client)
    13.13.13.13 (metric 20) from 13.13.13.13 (13.13.13.13)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Originator: 5.5.5.5, Cluster list: 13.13.13.13
      mpls labels in/out 8004/93000
      rx pathid: 0, tx pathid: 0x0

RP/0/0/CPU0:XRv12#show bgp ipv4 labeled-unicast 5.5.5.5/32 bestpath-compare |
begin Local,
  Local, (Received from a RR-client)
    13.13.13.13 (metric 10) from 13.13.13.13 (5.5.5.5)
      Received Label 93000
      Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best
      Received Path ID 0, Local Path ID 1, version 27
      Originator: 5.5.5.5, Cluster list: 13.13.13.13
      best of local AS, Overall best
  Path #2: Received by speaker 0
  Not advertised to any peer
  Local, (Received from a RR-client)
    14.14.14.14 (metric 30) from 14.14.14.14 (5.5.5.5)
      Received Label 94008
      Origin incomplete, metric 0, localpref 100, valid, internal
      Received Path ID 0, Local Path ID 0, version 0
      Originator: 5.5.5.5, Cluster list: 14.14.14.14
      Higher IGP metric than best path (path #1)

```

XRv13 and XRv14 implements the textbook-correct, high-scalability version of UMPLS where IGP routes between aggregation and core islands do not cross at all. This is not the default IS-IS behavior since all L1 routes are converted to L2 routes at the L1/L2 router. UMPLS suggests the “core” and “aggregation” IGPs must be totally separate with no redistribution between the two. Using OSPF areas or IS-IS levels actually complicates UMPLS since manual filtering must be applied to break inter-area routing. We can disable this behavior with a deny-all RPL to stop all L1 to L2 propagation. In normal circumstances, this would break L1 to L2 connectivity. Since CSR5's loopback is known via IGP within its island and BGP beyond its island, this is an acceptable strategy when coupled with MPLS in the data plane. Tunneling

inside MPLS implies the core routers don't need routes to these endpoints provided the traffic can be label-switched across them (at least for L3VPN and L2VPN). We can verify this on any L2 router by checking the LSPs of XRv13 and XRv14 to ensure 5.5.5.5/32 does not appear. In fact, only the local ABR loopbacks exist in the L2 topology since the ABRs could not originate 5.5.5.5/32 into L2.

```
! XRv13 and XRv14
route-policy RPL_DROP
  drop
end-policy
```

```
router isis 49
  address-family ipv4 unicast
    propagate level 1 into level 2 route-policy RPL_DROP
```

```
RP/0/0/CPU0:XRv13#show isis database level 2 XRv13.00-00 detail | begin
Hostname
```

```
  Hostname:      XRv13
  IP Address:    13.13.13.13
  Router ID:     13.13.13.13
  Metric: 10     IS-Extended XRv12.00
  Metric: 0      IP-Extended 13.13.13.13/32
  Metric: 10     MT (IPv6 Unicast) IS-Extended XRv12.00
```

```
RP/0/0/CPU0:XRv13#show isis database level 2 XRv14.00-00 detail | begin
Hostname
```

```
  Hostname:      XRv14
  IP Address:    14.14.14.14
  Router ID:     14.14.14.14
  Metric: 10     IS-Extended CSR1.00
  Metric: 0      IP-Extended 14.14.14.14/32
  Metric: 10     MT (IPv6 Unicast) IS-Extended CSR1.00
```

For even stricter control, we can clear the attached bit (ATT-bit) which allows L1 routers in an area to install a default route via the ABRs. This is somewhat unrelated to UMPLS since there is no label for the default route, but we will disable it anyway. In an IP routing environment, this would break connectivity as well.

```
! XRv13 and XRv14
router isis 49
  address-family ipv4 unicast
    attached-bit send never-set
```

A quick check on CSR5 shows the ATT bit is cleared for those two LSPs. This is shown in contrast to CSR2, where CSR4 and CSR10 are setting the ATT bit. These outputs were shown earlier during the initial IS-IS verification, but are verified again for completeness.

```
CSR5#show isis database level-1
```

```
Tag 49:
```

```
IS-IS Level-1 Link State Database:
```

LSPID	LSP Seq Num	LSP Checksum	LSP Holdtime	ATT/P/OL
CSR5.00-00	* 0x000001DA	0x50FA	813	0/0/0
XRv13.00-00	0x000001D1	0x4963	860	0/0/0
XRv14.00-00	0x000001CE	0x495E	1143	0/0/0

```
CSR2#show isis database level-1
```

```
Tag 49:
```

```
IS-IS Level-1 Link State Database:
```

LSPID	LSP Seq Num	LSP Checksum	LSP Holdtime	ATT/P/OL
CSR2.00-00	* 0x000001DC	0x8141	660	0/0/0
CSR4.00-00	0x000001D8	0x116E	633	1/0/0
CSR10.00-00	0x000001D6	0x1114	938	1/0/0

We verified that CSR8 and XRv12 had CSR5's loopback in BGP. Next, we verify the L1/L2 routers in area 49.BEEF (remote island) have these routes. These L1/L2 routers are the RRs that will reflect these iBGP routes to the PE routers. CSR4 happens to be both a PE and L1/L2 router. There is no reason for the core RRs to change the BGP next-hops for these routes since the L1/L2 routers have IGP routes to all other L1/L2 routers. Technically, the core RRs could do this, but it would be highly inefficient as those RRs would place themselves in the data plane, which is undesirable.

```
CSR4#show bgp ipv4 unicast 5.5.5.5/32
```

```
BGP routing table entry for 5.5.5.5/32, version 26
```

```
Paths: (2 available, best #2, table default)
```

```
Advertised to update-groups:
```

```
1
```

```
Refresh Epoch 1
```

```
Local
```

```
13.13.13.13 (metric 50) from 12.12.12.12 (12.12.12.12)
```

```
Origin incomplete, metric 0, localpref 100, valid, internal
```

```
Originator: 5.5.5.5, Cluster list: 12.12.12.12, 13.13.13.13
```

```
mpls labels in/out 4003/93000
```

```
rx pathid: 0, tx pathid: 0
```

```
Refresh Epoch 3
```

```
Local
```

```
13.13.13.13 (metric 50) from 8.8.8.8 (8.8.8.8)
```

```
Origin incomplete, metric 0, localpref 100, valid, internal, best
```

```
Originator: 5.5.5.5, Cluster list: 8.8.8.8, 13.13.13.13
```

```
mpls labels in/out 4003/93000
```

```
rx pathid: 0, tx pathid: 0x0
```

```
CSR10#show bgp ipv4 unicast 5.5.5.5/32
```

```
BGP routing table entry for 5.5.5.5/32, version 3
```

```
Paths: (2 available, best #2, table default)
```

```
Advertised to update-groups:
```

```
6
```


Refresh Epoch 1

Local

```
13.13.13.13 (metric 30) from 12.12.12.12 (12.12.12.12)
  Origin incomplete, metric 0, localpref 100, valid, internal
  Originator: 5.5.5.5, Cluster list: 12.12.12.12, 13.13.13.13
  mpls labels in/out 10010/93000
  rx pathid: 0, tx pathid: 0
```

Refresh Epoch 8

Local

```
13.13.13.13 (metric 30) from 8.8.8.8 (8.8.8.8)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  Originator: 5.5.5.5, Cluster list: 8.8.8.8, 13.13.13.13
  mpls labels in/out 10010/93000
  rx pathid: 0, tx pathid: 0x0
```

CSR2 must learn these routes from CSR4 and CSR10. Due to the link metric adjustments, CSR2 prefers the path via CSR10 (cost to BGP next hop). Notice that the next-hops have been changed to represent the L1/L2 routers, which is a fundamental concept of unified MPLS. Technically this would not be required in this case since CSR2 has BGP routers for CSR8 and XRv12, but this isn't a typical, and was done for flexible testing only. The output below shows the "textbook" unified MPLS setup.

```
CSR2#show bgp ipv4 unicast 5.5.5.5/32
```

```
BGP routing table entry for 5.5.5.5/32, version 51
```

```
Paths: (2 available, best #1, table default)
```

```
Not advertised to any peer
```

```
Refresh Epoch 3
```

Local

```
10.10.10.10 (metric 10) from 10.10.10.10 (10.10.10.10)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  Originator: 5.5.5.5, Cluster list: 10.10.10.10, 8.8.8.8, 13.13.13.13
  mpls labels in/out nolabel/10010
  rx pathid: 0, tx pathid: 0x0
```

Refresh Epoch 1

Local

```
4.4.4.4 (metric 50) from 4.4.4.4 (4.4.4.4)
  Origin incomplete, metric 0, localpref 100, valid, internal
  Originator: 5.5.5.5, Cluster list: 4.4.4.4, 8.8.8.8, 13.13.13.13
  mpls labels in/out nolabel/4003
  rx pathid: 0, tx pathid: 0
```

33.2.1.1 L3VPN

Now that the UMPLS building blocks have been verified, we will trace some L3VPN LSPs. The basic VRF U1 and VRF U2 configurations are shown below on CSR5. The L3VPNs are fully meshed with the exception of CSR2 and CSR4. Since they service the same customer, there is little reason for them to import one another's RTs. The RT values are self-documenting since the last digit represents the PE number.

```

! CSR5
vrf definition U1
  rd 49:105
  route-target export 49:105
  route-target import 49:102
  route-target import 49:107
  route-target import 49:104

  address-family ipv4
    mdt auto-discovery mldp
    mdt default mpls mldp p2mp

  address-family ipv6
    mdt auto-discovery mldp
    mdt default mpls mldp p2mp

vrf definition U2
  rd 49:205
  vpn id 49:222
  route-target export 49:205
  route-target import 49:202
  route-target import 49:207
  route-target import 49:204

  address-family ipv4
    mdt preference mldp
    mdt default mpls mldp 8.8.8.8

  address-family ipv6
    mdt preference mldp
    mdt default mpls mldp 8.8.8.8

```

Next, we will send traffic from CSR6 to XRv11 within VRF U1. This is classic MPLS L3VPN service wrapped inside UMPLS transport. CSR2 learns label 5012 from CSR5, the remote PE in a different IGP island, which is the bottom label. This label represents the final destination within the VPN and should never be swapped.

```

CSR2#show bgp vpnv4 unicast vrf U1 1.11.11.11/32
BGP routing table entry for 49:102:1.11.11.11/32, version 144
Paths: (1 available, best #1, table U1)
  Advertised to update-groups:
    6
  Refresh Epoch 2
  65000, imported path from 49:105:1.11.11.11/32 (global)
    5.5.5.5 (metric 10) (via default) from 7.7.7.7 (7.7.7.7)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:49:105
      Originator: 5.5.5.5, Cluster list: 7.7.7.7

```

```
mpls labels in/out nlabel/5012
rx pathid: 0, tx pathid: 0x0
```

The route to the VPNv4 next-hop also happens to be a BGP route. The routing table reveals the MPLS label used for this destination since labeled-unicast was negotiated for IPv4 BGP. This represents CSR10's label to reach 5.5.5.5/32, which is the remote PE.

```
CSR2#show ip route 5.5.5.5
Routing entry for 5.5.5.5/32
  Known via "bgp 49", distance 200, metric 0, type internal
  Last update from 10.10.10.10 4d21h ago
  Routing Descriptor Blocks:
    * 10.10.10.10, from 10.10.10.10, 4d21h ago
      Route metric is 0, traffic share count is 1
      AS Hops 0
      MPLS label: 10010
```

The route to the IPv4 labeled-unicast next-hop is an IS-IS route, which means an LDP label will be used (no TE in play, yet). Since CSR10 is directly connected, CSR2 uses implicit-null, which effectively means no third label is pushed in this case. CSR10 will receive a label stack of {10010 5012}.

```
CSR2#show ip route 10.10.10.10
Routing entry for 10.10.10.10/32
  Known via "isis", distance 115, metric 10, type level-1
  Redistributing via isis 49
  Last update from 49.2.10.10 on GigabitEthernet2.520, 4d21h ago
  Routing Descriptor Blocks:
    * 49.2.10.10, from 10.10.10.10, 4d21h ago, via GigabitEthernet2.520
      Route metric is 10, traffic share count is 1
```

```
CSR2#show mpls ldp bindings 10.10.10.10 32 neighbor 10.10.10.10
lib entry: 10.10.10.10/32, rev 135
remote binding: lsr: 10.10.10.10:0, label: imp-null
```

Upon receipt, CSR10 consults its LFIB since the ingress packet was MPLS encapsulated. We will examine the control-plane structure first for extra verification. Since label 10010 was a BGP-allocated label as CSR10 changed the next hop, it is assumed CSR10 will perform a swap operation on this label. CSR10's next-hop for 5.5.5.5/32 is XRv13 who allocated label 93000. This implies that CSR10 swaps 10010 for 93000 which directs traffic to XRv13. The label stack is now {93000 5012}.

```
CSR10#show bgp ipv4 unicast 5.5.5.5/32 bestpath
BGP routing table entry for 5.5.5.5/32, version 3
Paths: (2 available, best #2, table default)
  Advertised to update-groups:
    6
  Refresh Epoch 8
```

Local

```
13.13.13.13 (metric 30) from 8.8.8.8 (8.8.8.8)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  Originator: 5.5.5.5, Cluster list: 8.8.8.8, 13.13.13.13
  mpls labels in/out 10010/93000
  rx pathid: 0, tx pathid: 0x0
```

Next, CSR10 needs to determine the path to reach 13.13.13.13/32. This is an IGP route, which means CSR10 selects an LDP label. Looking at the RIB and FIB for brevity, we can see this IS-IS route points towards CSR8 using label 8006. CSR10 pushes this third label onto the stack, resulting in {8006 93000 5012}.

```
CSR10#show ip route 13.13.13.13
Routing entry for 13.13.13.13/32
  Known via "isis", distance 115, metric 30, type level-2
  Redistributing via isis 49
  Last update from 49.8.10.8 on GigabitEthernet2.580, 4d21h ago
  Routing Descriptor Blocks:
  * 49.8.10.8, from 13.13.13.13, 4d21h ago, via GigabitEthernet2.580
    Route metric is 30, traffic share count is 1
```

```
CSR10#show ip cef 13.13.13.13
13.13.13.13/32
  nexthop 49.8.10.8 GigabitEthernet2.580 label 8006
```

As a quick verification, we can examine the detailed LFIB information for incoming label 10010 to see the swap and push operations rolled into one show command.

```
CSR10#show mpls forwarding-table labels 10010 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
10010      93000     5.5.5.5/32      27936151    Gi2.580   49.8.10.8
           MAC/Encaps=18/26, MRU=1496, Label Stack{8006 93000}
           000C296AFFD5000C2949716481000DFC8847 01F4600016B48000
           No output feature configured
```

In the path to XRv13, CSR8 is a P router that performs a simple swap operation between LDP labels. The label stack becomes {92001 10010 5012}.

```
CSR8#show mpls forwarding-table labels 8006
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
8006      92001     13.13.13.13/32  28983561    Gi2.582   49.8.12.12
```

```
CSR8#show mpls ldp bindings 13.13.13.13 32 neighbor 12.12.12.12
lib entry: 13.13.13.13/32, rev 121
  remote binding: lsr: 12.12.12.12:0, label: 92001
```

Likewise, XRv12 is a P router that performs PHP before sending the packet to XRv13. This exposes the BGP labeled-unicast label to XRv13. The label stack becomes {93000 5012}.

```
RP/0/0/CPU0:XRv12#show mpls forwarding labels 92001
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
92001	Pop	13.13.13.13/32	Gi0/0/0/0.523	49.12.13.13	67581176

```
RP/0/0/CPU0:XRv12#show mpls ldp bindings 13.13.13.13/32 neighbor 13.13.13.13
13.13.13.13/32, rev 120
```

```
Local binding: label: 92001
```

```
Remote bindings: (3 peers)
```

Peer	Label
13.13.13.13:0	ImpNull

XRv13 receives the MPLS packet with BGP label exposed. BGP shows the local (incoming) label of 93000 with an LDP-learned received (outgoing) label of 3 towards CSR5. Label 3 is implicit-null, which implies XRv13 will perform a pop operation of the BGP label. This is important because although the BGP label would also be implicit-null, technically the LDP one is used since the route is learned via IGP. The combination of popping the BGP label and not adding an LDP label exposes the VPN label to CSR5. The label stack becomes {5012}.

```
RP/0/0/CPU0:XRv13#show route ipv4 5.5.5.5/32
```

```
Routing entry for 5.5.5.5/32
```

```
Known via "isis 49", distance 115, metric 10, type level-1
```

```
Routing Descriptor Blocks
```

```
49.5.13.5, from 5.5.5.5, via GigabitEthernet0/0/0/0.553
```

```
Route metric is 10
```

```
No advertising protos.
```

```
RP/0/0/CPU0:XRv13#show mpls ldp bindings 5.5.5.5/32 neighbor 5.5.5.5
5.5.5.5/32, rev 46
```

```
Local binding: label: 93000
```

```
Remote bindings: (2 peers)
```

Peer	Label
5.5.5.5:0	ImpNull

At this point, CSR5 performs basic MPLS L3VPN PE techniques of removing the VPN label, mapping it to a VRF, and forwarding the packet to the VPN customer. We verify VPNv4 BGP and LFIB tables for clarity.

```
CSR5#show bgp vpnv4 unicast vrf U1 1.11.11.11/32
```

```
BGP routing table entry for 49:105:1.11.11.11/32, version 20
```

```

Paths: (1 available, best #1, table U1)
  Advertised to update-groups:
    9
  Refresh Epoch 1
  65000
  192.168.105.11 (via vrf U1) from 192.168.105.11 (11.11.11.11)
    Origin incomplete, metric 0, localpref 100, valid, external, best
    Extended Community: RT:49:105
    mpls labels in/out 5012/nolabel
    rx pathid: 0, tx pathid: 0x0

```

```

CSR5#show mpls forwarding-table labels 5012
Local   Outgoing   Prefix           Bytes Label   Outgoing   Next Hop
Label   Label       or Tunnel Id     Switched      interface
5012    No Label    1.11.11.11/32[V] 18604         Gi2.105     192.168.105.11

```

As a final check, we traceroute from within the VPN on CSR6. The label stack at each hop matches what we manually verified above.

```

CSR6#traceroute vrf U1 1.11.11.11 source 1.6.6.6
Type escape sequence to abort.
Tracing the route to 1.11.11.11
VRF info: (vrf in name/id, vrf out name/id)
 1 192.168.102.2 0 msec 1 msec 0 msec
 2 49.2.10.10 [MPLS: Labels 10010/5012 Exp 0] 38 msec 33 msec 33 msec
 3 49.8.10.8 [MPLS: Labels 8006/93000/5012 Exp 0] 34 msec 33 msec 40 msec
 4 49.8.12.12 [MPLS: Labels 92001/93000/5012 Exp 0] 35 msec 33 msec 34 msec
 5 49.12.13.13 [MPLS: Labels 93000/5012 Exp 0] 33 msec 33 msec 34 msec
 6 192.168.105.5 [MPLS: Label 5012 Exp 0] 33 msec 40 msec 35 msec
 7 192.168.105.11 33 msec * 36 msec

```

We can make some minor adjustments to this topology to change the label stack slightly. We temporarily shut down the link between CSR5 and XRv13 (not shown). The IS-IS session will fail but the iBGP session will not, which means XRv13 will still be the preferred ingress point for traffic coming from the core. This is suboptimal, but for testing it is beneficial since we can see additional label impositions on XRv13. The label stack all the way up to XRv13 remains the same, but when XRv13 receives packets from the core labeled 93000, it performs a different operation. We can see that the BGP label is still popped, but a new LDP label must be pushed. XRv13 must route via XRv14 to reach CSR5 now, so XRv13 pushes XRv14's LDP label for 5.5.5.5/32 onto the stack. The label stack becomes {94008 5012}.

```

RP/0/0/CPU0:XRv13#show route ipv4 5.5.5.5
Routing entry for 5.5.5.5/32
  Known via "isis 49", distance 115, metric 20, type level-1
  Routing Descriptor Blocks
    49.13.14.14, from 5.5.5.5, via GigabitEthernet0/0/0/0.534
      Route metric is 20
  No advertising protos.

```

```
RP/0/0/CPU0:XRv13#show mpls ldp bindings 5.5.5.5/32 neighbor 14.14.14.14
5.5.5.5/32, rev 46
```

```
Local binding: label: 93000
```

```
Remote bindings: (1 peers)
```

Peer	Label
14.14.14.14:0	94008

XRv14 receives packets labeled 94008, pops this label, and delivers the singly-labeled packet to CSR5. CSR5's operation is not shown again since it did not change.

```
RP/0/0/CPU0:XRv14#show mpls forwarding labels 94008
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
94008	Pop	5.5.5.5/32	Gi0/0/0/0.554	49.5.14.5	7089319

Using traceroute, we can verify another hop in the network along with the additional label. Note that simply counting the number of labels is not a valid way to determine what type of label is used at each point in the stack. For example, hop 5 has a BGP-IPv4/BGP-VPNv4 label stack, while hop 6 has an LDP/BGP-VPNv4 label stack. RSVP-TE could be used in the network also which is tested later. One would have to actually know the network architecture to understand the source of each label seen in a traceroute.

```
CSR6#traceroute vrf U1 1.11.11.11 source 1.6.6.6
```

```
Type escape sequence to abort.
```

```
Tracing the route to 1.11.11.11
```

```
VRF info: (vrf in name/id, vrf out name/id)
```

```
 1 192.168.102.2 1 msec 0 msec 1 msec
 2 49.2.10.10 [MPLS: Labels 10010/5012 Exp 0] 42 msec 41 msec 42 msec
 3 49.8.10.8 [MPLS: Labels 8006/93000/5012 Exp 0] 33 msec 41 msec 42 msec
 4 49.8.12.12 [MPLS: Labels 92001/93000/5012 Exp 0] 50 msec 41 msec 59 msec
 5 49.12.13.13 [MPLS: Labels 93000/5012 Exp 0] 41 msec 42 msec 42 msec
 6 49.13.14.14 [MPLS: Labels 94008/5012 Exp 0] 41 msec 50 msec 42 msec
 7 192.168.105.5 [MPLS: Label 5012 Exp 0] 42 msec 41 msec 42 msec
 8 192.168.105.11 41 msec * 48 msec
```

Before continuing, we will restore the link between CSR5 and XRv13. We will now break the link between CSR2 and CSR10. This will cause CSR10 to maintain an iBGP session with CSR2, but traffic is forced over the high-cost link between CSR2 and CSR4. This changes the label stack once again. The VPNv4 route of 1.11.11.11/32 has not changed as 5.5.5.5 is still the next-hop using label 5012.

```
CSR2#show bgp vpnv4 unicast vrf U1 labels
```

Network	Next Hop	In label/Out label
Route Distinguisher: 49:102 (U1)		

1.3.3.3/32	7.7.7.7	nolabel/7011
1.6.6.6/32	192.168.102.6	2019/nolabel
1.11.11.11/32	5.5.5.5	nolabel/5012

Now, CSR5 prefers CSR4 as the BGP bestpath since the cost to CSR4 is less than to CSR10 (high speed link directly to CSR10 was shutdown).

```
CSR2#show ip route 5.5.5.5
Routing entry for 5.5.5.5/32
  Known via "bgp 49", distance 200, metric 0, type internal
  Last update from 4.4.4.4 00:04:15 ago
  Routing Descriptor Blocks:
  * 4.4.4.4, from 4.4.4.4, 00:04:15 ago
    Route metric is 0, traffic share count is 1
    AS Hops 0
    MPLS label: 4003
```

```
CSR2#show bgp ipv4 unicast 5.5.5.5
BGP routing table entry for 5.5.5.5/32, version 67
Paths: (2 available, best #2, table default)
  Not advertised to any peer
  Refresh Epoch 3
  Local
    10.10.10.10 (metric 60) from 10.10.10.10 (10.10.10.10)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Originator: 5.5.5.5, Cluster list: 10.10.10.10, 8.8.8.8, 13.13.13.13
      mpls labels in/out nolabel/10010
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  Local
    4.4.4.4 (metric 50) from 4.4.4.4 (4.4.4.4)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Originator: 5.5.5.5, Cluster list: 4.4.4.4, 8.8.8.8, 13.13.13.13
      mpls labels in/out nolabel/4003
      rx pathid: 0, tx pathid: 0x0
```

CSR2's route to 4.4.4.4/32 is via CSR4, and since CSR2 and CSR4 are directly connected, this results in an implicit-null label. No third label is pushed, so the label stack is now {4003 5012}.

```
CSR2#show ip route 4.4.4.4
Routing entry for 4.4.4.4/32
  Known via "isis", distance 115, metric 50, type level-1
  Redistributing via isis 49
  Last update from 49.2.4.4 on GigabitEthernet2.524, 00:06:43 ago
  Routing Descriptor Blocks:
  * 49.2.4.4, from 4.4.4.4, 00:06:43 ago, via GigabitEthernet2.524
    Route metric is 50, traffic share count is 1
```



```

CSR2#show mpls ldp bindings 4.4.4.4 32 neighbor 4.4.4.4
lib entry: 4.4.4.4/32, rev 106
remote binding: lsr: 4.4.4.4:0, label: imp-null

```

Upon receipt, CSR4 swaps 4003 for 93000, which is XRv13's local label for 5.5.5.5/32. This is the same label CSR10 used earlier, so at this point the LSPs appear similar from a BGP perspective.

```

CSR4#show bgp ipv4 unicast 5.5.5.5/32 bestpath
BGP routing table entry for 5.5.5.5/32, version 45
Paths: (2 available, best #2, table default)
  Advertised to update-groups:
    1
  Refresh Epoch 3
  Local
    13.13.13.13 (metric 40) from 8.8.8.8 (8.8.8.8)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Originator: 5.5.5.5, Cluster list: 8.8.8.8, 13.13.13.13
      mpls labels in/out 4003/93000
      rx pathid: 0, tx pathid: 0x0

```

It is interesting to note that CSR4's route to 13.13.13.13 has a next-hop via CSR10. This is because the cost through CSR10 is less than through CSR9 as it transits four IS-IS hops. Using CSR9, this would be five IS-IS hops with a path cost of 50. This implies that CSR4 must push CSR10's local label for 13.13.13.13/32 onto its stack, which is label 10004. The label stack becomes {10004 93000 5012}.

```

CSR4#show ip route 13.13.13.13
Routing entry for 13.13.13.13/32
  Known via "isis", distance 115, metric 40, type level-2
  Redistributing via isis 49
  Last update from 49.4.10.10 on GigabitEthernet2.540, 00:10:16 ago
  Routing Descriptor Blocks:
    * 49.4.10.10, from 13.13.13.13, 00:10:16 ago, via GigabitEthernet2.540
      Route metric is 40, traffic share count is 1

```

```

CSR4#show mpls ldp bindings 13.13.13.13 32 neighbor 10.10.10.10
lib entry: 13.13.13.13/32, rev 124
remote binding: lsr: 10.10.10.10:0, label: 10004

```

Following the LSP evaluated earlier, we expect CSR10 to switch this packet towards CSR8. CSR10 merges this with the LSP we saw earlier by swapping to label 8006 (also seen earlier), essentially putting the traffic back onto the same path through the core and remote IGP island as seen before.

```

CSR10#show mpls forwarding-table labels 10004
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
10004      8006      13.13.13.13/32  50008        Gi2.580    49.8.10.8

```

Rather than verify the same LSP again from CSR8 forward, we will confirm the new LSP via traceroute on CSR6. The label stacks at each hop match what we verified above.

```
CSR6#traceroute vrf U1 1.11.11.11 source 1.6.6.6
Type escape sequence to abort.
Tracing the route to 1.11.11.11
VRF info: (vrf in name/id, vrf out name/id)
 1 192.168.102.2 1 msec 0 msec 1 msec
 2 49.2.4.4 [MPLS: Labels 4003/5012 Exp 0] 47 msec 33 msec 33 msec
 3 49.4.10.10 [MPLS: Labels 10004/93000/5012 Exp 0] 34 msec 33 msec 33 msec
 4 49.8.10.8 [MPLS: Labels 8006/93000/5012 Exp 0] 33 msec 33 msec 33 msec
 5 49.8.12.12 [MPLS: Labels 92001/93000/5012 Exp 0] 33 msec 33 msec 33 msec
 6 49.12.13.13 [MPLS: Labels 93000/5012 Exp 0] 33 msec 34 msec 33 msec
 7 192.168.105.5 [MPLS: Label 5012 Exp 0] 33 msec 34 msec 33 msec
 8 192.168.105.11 75 msec * 42 msec
```

Traffic in the opposite direction would, in a real deployment, work exactly the same way. However, I demonstrate an alternative UMPLS architecture in the reverse direction. This introduces more variety in our testing and is more interesting than tracing the same thing twice. This is not the "correct" way to implement UMPLS since it is less scalable by leaking aggregation loopbacks into the core. CSR4 and CSR10 are L1/L2 routers for area 49.BEEF. We saw earlier that they changed the BGP next-hop to be themselves when advertising routes to CSR2 (island PE); towards the core RRs, they do not set the next-hop to themselves. This is because the L1-to-L2 propagation is not manually filtered as it was on XRv13 and XRv14; as such, all L2 routers will have IS-IS L2 routes for CSR2's loopback. This obviates the need for BGP to shim another label onto the stack for core transport since all L1/L2 routers can use LDP labels to reach the remote PEs across the core. To visualize the difference, we can check CSR8's route for 2.2.2.2/32. We compare it to the route to 5.5.5.5/32 seen earlier, since the routes to CSR5 have next-hops of the L1/L2 routers while CSR2's route is marked as a RIB-failure.

```
CSR8#show bgp ipv4 unicast 5.5.5.5
BGP routing table entry for 5.5.5.5/32, version 18
Paths: (2 available, best #2, table default)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  Local, (Received from a RR-client)
    14.14.14.14 (metric 20) from 14.14.14.14 (14.14.14.14)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Originator: 5.5.5.5, Cluster list: 14.14.14.14
      mpls labels in/out 8004/94008
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  Local, (Received from a RR-client)
    13.13.13.13 (metric 20) from 13.13.13.13 (13.13.13.13)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
```

```
Originator: 5.5.5.5, Cluster list: 13.13.13.13
mpls labels in/out 8004/93000
rx pathid: 0, tx pathid: 0x0
```

```
CSR8#show bgp ipv4 unicast 2.2.2.2
BGP routing table entry for 2.2.2.2/32, version 33
Paths: (2 available, best #2, table default, RIB-failure(17))
  Advertised to update-groups:
    2
  Refresh Epoch 3
  Local, (Received from a RR-client)
    2.2.2.2 (metric 20) from 10.10.10.10 (10.10.10.10)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Originator: 2.2.2.2, Cluster list: 10.10.10.10
      mpls labels in/out 8010/imp-null
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 4
  Local, (Received from a RR-client)
    2.2.2.2 (metric 20) from 4.4.4.4 (4.4.4.4)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Originator: 2.2.2.2, Cluster list: 4.4.4.4
      mpls labels in/out 8010/imp-null
      rx pathid: 0, tx pathid: 0x0
```

It's good that CSR2's route was marked as a RIB failure since the next-hop is the same as the prefix which would cause a recursive routing logic failure. We verify that CSR8 has an IS-IS route in the RIB, which explains the BGP RIB failure. We compare this to the BGP route to 5.5.5.5/32 seen earlier. CSR8 reaches 2.2.2.2/32 via CSR10 and binds label 10000 to this prefix, which is learned via LDP.

```
CSR8#show ip route 5.5.5.5
Routing entry for 5.5.5.5/32
  Known via "bgp 49", distance 200, metric 0, type internal
  Last update from 13.13.13.13 5d04h ago
  Routing Descriptor Blocks:
    * 13.13.13.13, from 13.13.13.13, 5d04h ago
      Route metric is 0, traffic share count is 1
      AS Hops 0
      MPLS label: 93000

CSR8#show ip route 2.2.2.2
Routing entry for 2.2.2.2/32
  Known via "isis", distance 115, metric 20, type level-2
  Redistributing via isis 49
  Last update from 49.8.10.10 on GigabitEthernet2.580, 01:32:53 ago
  Routing Descriptor Blocks:
    * 49.8.10.10, from 10.10.10.10, 01:32:53 ago, via GigabitEthernet2.580
      Route metric is 20, traffic share count is 1
```

```
CSR8#show mpls ldp bindings 2.2.2.2 32 neighbor 10.10.10.10
lib entry: 2.2.2.2/32, rev 130
remote binding: lsr: 10.10.10.10:0, label: 10000
```

CSR4 and CSR10 also do not clear the ATT-bit when sending their L1 LSPs into area 49.BEEF, which isn't terribly relevant for UMPLS but demonstrates that we don't have to modify IS-IS. We saw this earlier but it is shown again for clarity.

```
CSR2#show isis database level-1
Tag 49:
IS-IS Level-1 Link State Database:
LSPID                LSP Seq Num  LSP Checksum  LSP Holdtime  ATT/P/OL
CSR2.00-00            * 0x00000250  0x97B6        935           0/0/0
CSR4.00-00            0x0000024E  0x10C7        478           1/0/0
CSR10.00-00           0x0000024C  0x435E        985           1/0/0
```

Now that we have seen the difference, we will trace an LSP from XRv11 to CSR6 using VPNv6 inside VRF U2. Since CSR6 is multi-homed to two PEs (CSR2 and CSR4), CSR5 learns two VPNv6 routes. CSR2 wins due to having a lower originator-ID, which uses label 2007 and an IPv4 next-hop of 2.2.2.2/32.

```
CSR5#show bgp vpnv6 unicast vrf U2 ::2:6:6:6/128
BGP routing table entry for [49:205]::2:6:6:6/128, version 514
Paths: (2 available, best #2, table U2)
  Advertised to update-groups:
    3
  Refresh Epoch 2
  65000, imported path from [49:204]::2:6:6:6/128 (global)
    ::FFFF:4.4.4.4 (metric 10) (via default) from 7.7.7.7 (7.7.7.7)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Extended Community: RT:49:204
      Originator: 4.4.4.4, Cluster list: 7.7.7.7
      mpls labels in/out nolaabel/4010
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 2
  65000, imported path from [49:202]::2:6:6:6/128 (global)
    ::FFFF:2.2.2.2 (metric 10) (via default) from 7.7.7.7 (7.7.7.7)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:49:202
      Originator: 2.2.2.2, Cluster list: 7.7.7.7
      mpls labels in/out nolaabel/2007
      rx pathid: 0, tx pathid: 0x0
```

The route to CSR2 is via XRv13 learned using BGP. Quickly checking the BGP table, we see that XRv13 wins by having a lower neighbor ID. CSR5 uses BGP label 93008 to reach 2.2.2.2/32. The label stack is now {93008 2007}.

```
CSR5#show ip route 2.2.2.2
```

```

Routing entry for 2.2.2.2/32
  Known via "bgp 49", distance 200, metric 0, type internal
  Last update from 13.13.13.13 01:42:03 ago
  Routing Descriptor Blocks:
  * 13.13.13.13, from 13.13.13.13, 01:42:03 ago
    Route metric is 0, traffic share count is 1
    AS Hops 0
    MPLS label: 93008

CSR5#show bgp ipv4 unicast 2.2.2.2/32
BGP routing table entry for 2.2.2.2/32, version 65
Paths: (2 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    13.13.13.13 (metric 10) from 13.13.13.13 (13.13.13.13)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Originator: 2.2.2.2, Cluster list: 13.13.13.13, 8.8.8.8, 4.4.4.4
      mpls labels in/out nolabel/93008
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  Local
    14.14.14.14 (metric 10) from 14.14.14.14 (14.14.14.14)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Originator: 2.2.2.2, Cluster list: 14.14.14.14, 8.8.8.8, 4.4.4.4
      mpls labels in/out nolabel/94004
      rx pathid: 0, tx pathid: 0

```

Lastly, CSR5 needs to reach XRv3's loopback; it has an IGP route bound to an implicit-null label from XRv13. The label stack remains {93008 2007}.

```

CSR5#show ip route 13.13.13.13
Routing entry for 13.13.13.13/32
  Known via "isis", distance 115, metric 10, type level-1
  Redistributing via isis 49
  Last update from 49.5.13.13 on GigabitEthernet2.553, 02:05:22 ago
  Routing Descriptor Blocks:
  * 49.5.13.13, from 13.13.13.13, 02:05:22 ago, via GigabitEthernet2.553
    Route metric is 10, traffic share count is 1

CSR5#show mpls ldp bindings 13.13.13.13 32 neighbor 13.13.13.13
  lib entry: 13.13.13.13/32, rev 114
    remote binding: lsr: 13.13.13.13:0, label: imp-null

```

XRv13 performs a very interesting function. It's BGP route, regardless of which one is selected (CSR8 is best due to lower neighbor ID), is reachable via 2.2.2.2/32. Since this is just a copy of the same path that passed through CSR4, the label binding is also the same, which is implicit-null. This is because CSR4 did not change the next-hop when it advertised these routes to the core RRs, which means it also did not

change the label binding originally allocated by CSR2. Based on this logic, the label stack would become {2007} as label 93008 is popped in ingress. At a glance, this does not make sense.

```
RP/0/0/CPU0:XRv13#show bgp ipv4 unicast 2.2.2.2/32 | begin Paths
Paths: (2 available, best #1)
  Advertised to peers (in unique update groups):
    5.5.5.5
  Path #1: Received by speaker 0
  Advertised to peers (in unique update groups):
    5.5.5.5
  Local
    2.2.2.2 (metric 40) from 8.8.8.8 (2.2.2.2)
      Received Label 3
      Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best
      Received Path ID 0, Local Path ID 1, version 28
      Originator: 2.2.2.2, Cluster list: 8.8.8.8, 4.4.4.4
  Path #2: Received by speaker 0
  Not advertised to any peer
  Local
    2.2.2.2 (metric 40) from 12.12.12.12 (2.2.2.2)
      Received Label 3
      Origin incomplete, metric 0, localpref 100, valid, internal
      Received Path ID 0, Local Path ID 0, version 0
      Originator: 2.2.2.2, Cluster list: 12.12.12.12, 4.4.4.4
```

Exposing the VPN label to the core is going to break the LSP. Continuing the RIB lookup process, XRv13 performs a route lookup for 2.2.2.2 and finds an IS-IS route via XRv13. Again, this is not customary for UMPLS, but since the area 49.BEEF L1/L2 routers propagated L1 routes into L2, all L2 routers have IS-IS routes for those remote loopbacks. Checking the LIB, XRv13 binds label 92007 to CSR2's loopback prefix and pushes this onto the stack, which becomes {92007 2007}. The BGP route shown above is not used at all for label operations on XRv13 for this particular LSP, but it can be a challenging distraction.

```
RP/0/0/CPU0:XRv13#show route ipv4 2.2.2.2/32
Routing entry for 2.2.2.2/32
  Known via "isis 49", distance 115, metric 40, type level-2
  Routing Descriptor Blocks
    49.12.13.12, from 10.10.10.10, via GigabitEthernet0/0/0/0.523
      Route metric is 40
  No advertising protos.
```

```
RP/0/0/CPU0:XRv13#show mpls ldp bindings 2.2.2.2/32 neighbor 12.12.12.12
2.2.2.2/32, rev 76
  Local binding: label: 93008
  Remote bindings: (2 peers)
    Peer          Label
    -----
    -----
```

12.12.12.12:0 92007

The LFIB evaluates this operation as a "swap" operation. This is essentially how a BGP label can be swapped for an LDP label with no change in label stack depth. Looking only at the BGP labels, though, the swap is not obvious since BGP isn't aware of the full picture.

```
RP/0/0/CPU0:XRv13#show mpls forwarding labels 93008
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
93008	92007	2.2.2.2/32	Gi0/0/0/0.523	49.12.13.12	24291332

Since XRv13 has a route to the remote PE, the entire remainder of the LSP is simply LDP-based label swapping. I quickly show the hops along the remainder of the path. The path is as follows: XRv12, CSR8, CSR10 (PHP), and CSR2. CSR2 receives label 2007 and delivers it into VRF U2 for the specific destination prefix.

```
RP/0/0/CPU0:XRv12#show mpls forwarding labels 92007
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
92007	8010	2.2.2.2/32	Gi0/0/0/0.582	49.8.12.8	25373558

```
CSR8#show mpls forwarding-table labels 8010
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
8010	10000	2.2.2.2/32	64675692	Gi2.580	49.8.10.10

```
CSR10#show mpls forwarding-table labels 10000
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
10000	Pop Label	2.2.2.2/32	1143321	Gi2.520	49.2.10.2

```
CSR2#show mpls forwarding-table labels 2007 detail
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
2007	No Label	::2:6:6:6/128[V]	0	Gi2.202	FE80::6

MAC/Encaps=0/0, MRU=1504, Label Stack{}

VPN route: U2

No output feature configured

The traceroute on XRv11 within the VPN is shown below. The only interesting part is the junction between hop 1 and hop 2, since the BGP-allocated label of 93008 is swapped directly to an LDP label 92007.

```
RP/0/0/CPU0:XRv11#traceroute vrf U2 ::2:6:6:6 source ::2:11:11:11
```

Type escape sequence to abort.
Tracing the route to ::2:6:6:6

```
 1  fd00:192:168:205::5 19 msec 9 msec 9 msec
 2  2049:49:5:13::13 [MPLS: Labels 93008/2007 Exp 0] 59 msec 69 msec 49 msec
 3  2049:49:12:13::12 [MPLS: Labels 92007/2007 Exp 0] 79 msec 79 msec 129
msec
 4  ::ffff:49.8.12.8 [MPLS: Labels 8010/2007 Exp 0] 69 msec 79 msec 49 msec
 5  ::ffff:49.8.10.10 [MPLS: Labels 10000/2007 Exp 0] 59 msec 49 msec 49 msec
 6  fd00:192:168:202::2 [MPLS: Label 2007 Exp 0] 49 msec 49 msec 69 msec
 7  fd00:192:168:202::6 49 msec 39 msec 49 msec
```

33.2.1.2 L2VPN

The basic L2VPN configuration consists of a QinQ access circuit where both tags are pushed and popped symmetrically. BGP auto-discovery is used with LDP signaling to build a full-mesh of PWs between CSR2, CSR5, and CSR7. CSR4 does not participate in VPLS. Only CSR5 is shown for brevity since the configuration related to L2VPN is identical on all PEs. The control-word and sequencing are enabled as best practices for L2VPN designs.

```
! CSR5
interface GigabitEthernet2
 service instance 305 ethernet
 encapsulation dot1q 3551 second-dot1q 305
 rewrite ingress tag pop 2 symmetric

template type pseudowire VPLS_TEMPLATE
 encapsulation mpls
 sequencing both
 control-word include

l2vpn vfi context VPLS
 vpn id 567
 autodiscovery bgp signaling ldp template VPLS_TEMPLATE

router bgp 49
 address-family l2vpn vpls
 neighbor 7.7.7.7 activate
```

Because L2VPN is using the same LDP/BGP labeled-unicast paths as all other MPLS services, tracing the LSP will be identical as it is for L3VPN. The only difference is the PW label. We can reveal the PW label of 5019 using a number of commands, two of which are shown below. The purpose of demonstrating L2VPN is to show that UMPLS is not limited to only L3VPN (IP-based) services.

```
CSR2#show l2vpn atom vc vcid 567 destination 5.5.5.5 detail | include
label_stack
  Output interface: Gi2.520, imposed label stack {10010 5019}
```



```

CSR2#show l2vpn atom binding 5.5.5.5 | section 567
Destination Address: 5.5.5.5, VC ID: 567
Local Label: 2016
    Cbit: 1,    VC Type: Ethernet,    GroupID: n/a
    MTU: 1500,    Interface Desc: n/a
    VCCV: CC Type: RA [2], TTL [3]
    CV Type: LSPV [2]
Remote Label: 5019
    Cbit: 1,    VC Type: Ethernet,    GroupID: n/a
    MTU: 1500,    Interface Desc: n/a
    VCCV: CC Type: RA [2], TTL [3]
    CV Type: LSPV [2]

```

Tracing L2VPN from the customer's perspective won't reveal any of the MPLS details, so we will use OAM instead. Combining an ordinary traceroute from the PE with PW-aware OAM reveals all of the labels which reveals the same LSP traced earlier. The PW label is 5019 which is the bottom-most label. Appending this to the bottom of the label stacks seen in the traceroute, we can have the full stack. I use an 800-byte packet so that it stands out inside the EPC capture happening next.

```

CSR2#traceroute 5.5.5.5 source 2.2.2.2
Type escape sequence to abort.
Tracing the route to 5.5.5.5
VRF info: (vrf in name/id, vrf out name/id)
 1 49.2.10.10 [MPLS: Label 10010 Exp 0] 39 msec 41 msec 28 msec
 2 49.8.10.8  [MPLS: Labels 8006/93000 Exp 0] 34 msec 50 msec 55 msec
 3 49.8.12.12 [MPLS: Labels 92001/93000 Exp 0] 33 msec 28 msec 33 msec
 4 49.12.13.13 [MPLS: Label 93000 Exp 0] 42 msec 33 msec 34 msec
 5 49.5.13.5 50 msec * 30 msec

```

```

CSR2#ping mpls pseudowire 5.5.5.5 567 repeat 1 verbose size 800
Sending 1, 800-byte MPLS Echos to 5.5.5.5,
    timeout is 2 seconds, send interval is 0 msec:
[snip]
Type escape sequence to abort.
!    size 800, reply addr 49.5.13.5, return code 3[Labels: 5019 Exp: 0]
    Rx Interface: 0x31050D05
    local 2.2.2.2 remote 5.5.5.5 vc id 567

```

```

Success rate is 100 percent (1/1), round-trip min/avg/max = 37/37/37 ms
Total Time Elapsed 37 ms

```

As a quick verification of all L2VPN components, including the control-word, we use EPC outbound on CSR10 to capture this OAM ping. The label stack {8006 93000 5019} is highlighted in yellow and the control-word is highlighted in green. The hex values of 0x1F46, 0x16B48, and 0x139B represent the correct labels. The beginning of the CW is 0x1, which is 0001 in binary, representing a PW-ACH which is specific to MPLS OAM, and the trailing 0x21 represents an IPv4 OAM packet.

```

CSR10#show monitor capture CAP buffer dump
3
0000: 000C296A FFD5000C 29497164 81000DFC ..)j....)Iqd....
0010: 884701F4 60FE16B4 80FE0139 B1011000 .G..`.....9....
0020: 00214600 0320F347 40000111 6B7C0202 .!F.. .G@...k|..
0030: 02027F00 00019404 00000DAF 0DAF0308 .....
0040: E8F20001 00000102 00002A25 71160000 .....*%q...
0050: 0001D9D2 25EAA937 4BC60000 00000000 ....%..7K.....
0060: 0000FC00 000C0000 00090001 00040000 .....
0070: 00040001 0024000B 00200202 02020505 .....$... ..
0080: 05050005 0108000A 00310000 02370104 .....1...7..
0090: 02020202 01040505 05050003 02A401AB .....
[snip]

```

As another quick test, we will look at the path from CSR3 to XRv11. The L2VPN show command reveals the entire label stack at imposition, which is 3 labels deep as expected. This single command is very useful for verifying the individual MPLS forwarding components.

```

CSR7#show l2vpn atom vc vcid 567 destination 5.5.5.5 detail | include
label_stack
Output interface: Gi2.572, imposed label stack {92001 93000 5004}

```

For additional verification, I also verify the basis of each individual label below. The PW label of 5004 is allocated by LDP and is seen in the AToM bindings, which describes the MPLS service. The middle transport label of 93000 is allocated by BGP labeled-unicast to describe the path to CSR5 via XRv13. The route to XRv13 is an IGP route, so the LDP label 92001 is used.

```

CSR7#show l2vpn atom binding 5.5.5.5 | section 567
Destination Address: 5.5.5.5, VC ID: 567
Local Label: 7004
Cbit: 1, VC Type: Ethernet, GroupID: n/a
MTU: 1500, Interface Desc: n/a
VCCV: CC Type: RA [2], TTL [3]
CV Type: LSPV [2]
Remote Label: 5004
Cbit: 1, VC Type: Ethernet, GroupID: n/a
MTU: 1500, Interface Desc: n/a
VCCV: CC Type: RA [2], TTL [3]
CV Type: LSPV [2]

```

```

CSR7#show ip route 5.5.5.5
Routing entry for 5.5.5.5/32
Known via "bgp 49", distance 200, metric 0, type internal
Last update from 13.13.13.13 6d00h ago
Routing Descriptor Blocks:
* 13.13.13.13, from 8.8.8.8, 6d00h ago
Route metric is 0, traffic share count is 1

```

```
AS Hops 0
MPLS label: 93000
```

```
CSR7#show ip route 13.13.13.13
Routing entry for 13.13.13.13/32
  Known via "isis", distance 115, metric 20, type level-2
  Redistributing via isis 49
  Last update from 49.7.12.12 on GigabitEthernet2.572, 6d01h ago
  Routing Descriptor Blocks:
  * 49.7.12.12, from 13.13.13.13, 6d01h ago, via GigabitEthernet2.572
    Route metric is 20, traffic share count is 1
```

```
CSR7#show mpls ldp bindings 13.13.13.13 32 neighbor 12.12.12.12
lib entry: 13.13.13.13/32, rev 28
remote binding: lsr: 12.12.12.12:0, label: 92001
```

We can also verify the label stack using global traceroute (transport labels) and MPLS OAM (PW label).

```
CSR7#traceroute 5.5.5.5 source 7.7.7.7
Type escape sequence to abort.
Tracing the route to 5.5.5.5
VRF info: (vrf in name/id, vrf out name/id)
 1 49.7.12.12 [MPLS: Labels 92001/93000 Exp 0] 29 msec 34 msec 33 msec
 2 49.12.13.13 [MPLS: Label 93000 Exp 0] 33 msec 34 msec 33 msec
 3 49.5.13.5 25 msec * 31 msec
```

```
CSR7#ping mpls pseudowire 5.5.5.5 567 size 800 repeat 1 verbose
Sending 1, 800-byte MPLS Echos to 5.5.5.5,
  timeout is 2 seconds, send interval is 0 msec:
[snip]
```

```
Type escape sequence to abort.
! size 800, reply addr 49.5.13.5, return code 3[Labels: 5004 Exp: 0]
Rx Interface: 0x31050D05
local 7.7.7.7 remote 5.5.5.5 vc id 567
```

```
Success rate is 100 percent (1/1), round-trip min/avg/max = 49/49/49 ms
Total Time Elapsed 50 ms
```

Finally, we perform outbound EPC on CSR7 to verify the label stack and CW. To ensure sequencing works, I will send some large pings inside the L2VPN from CSR2 to XRv11. Using this technique versus OAM allows us to verify that CW sequencing is working. The label stack {92001 93000 5004} is shown in yellow with the CW shown in green. Notice that the CW begins with 0x0, which means this is a regular AToM packet as opposed to an OAM packet (no PW-ACH indications here), and the low-order 16 bits represents the sequence number which increments as follows: 0x2DED, 0x2DEE, and 0x2DEF.

```
CSR7#show monitor capture CAP buffer detailed
```

```

2 848 0.860948 00:0C:29:66:4C:2C -> 00:0C:29:87:63:7A MPLS unicast
0000: 000C2987 637A000C 29664C2C 81000DF4 ..).cz..)fL,....
0010: 88471676 10FF16B4 80FF0138 C1FF0000 .G.v.....8....
0020: 2DED000C 2950BAB7 000C29D7 81FE0800 -...)P....).....
0030: 45000320 000A0000 FF013774 C0A80003 E.. .....7t....

3 848 0.897949 00:0C:29:66:4C:2C -> 00:0C:29:87:63:7A MPLS unicast
0000: 000C2987 637A000C 29664C2C 81000DF4 ..).cz..)fL,....
0010: 88471676 10FF16B4 80FF0138 C1FF0000 .G.v.....8....
0020: 2DEE000C 2950BAB7 000C29D7 81FE0800 -...)P....).....
0030: 45000320 000B0000 FF013773 C0A80003 E.. .....7s....

4 848 0.981004 00:0C:29:66:4C:2C -> 00:0C:29:87:63:7A MPLS unicast
0000: 000C2987 637A000C 29664C2C 81000DF4 ..).cz..)fL,....
0010: 88471676 10FF16B4 80FF0138 C1FF0000 .G.v.....8....
0020: 2DEF000C 2950BAB7 000C29D7 81FE0800 -...)P....).....
0030: 45000320 000C0000 FF013772 C0A80003 E.. .....7r....

```

33.2.1.3 MVPN (mLDP profiles 1 and 17)

Next, we will examine two flavors of mLDP in the network. VRF U1 uses BGP-AD to build P2MP trees for the default MDT (profile 17) while VRF U2 uses a statically configured MP2MP default MDT (profile 1). Since only MVPN profile 0 works for CSC, we will use other profiles for variety. A quick verification on CSR7 shows the trees. The P2MP trees with GID 65536 are IPv4 MDTs and there is one rooted in each PE per MVPN instance. The second set of P2MP trees with GID 131072 are for IPv6 within the same MVPN. There is a single MP2MP tree for the second MVPN instance which is shared by for IPv4 and IPv6. The IPv4 GIDs are shown in yellow with IPv6 GIDs shown in green. IPv4 trees appear to always end with 0x10000 (65536 in decimal) while IPv6 doubles that number to 0x20000 (131072). The single MP2MP tree represents the VPN U2 default DTM.

```
CSR7#show mpls mldp database summary
```

LSM ID	Type	Root	Decoded Opaque Value	Client
Cnt.				
3	P2MP	2.2.2.2	[gid 65536 (0x00010000)]	1
5	P2MP	4.4.4.4	[gid 65536 (0x00010000)]	1
7	P2MP	5.5.5.5	[gid 65536 (0x00010000)]	1
1	P2MP	7.7.7.7	[gid 65536 (0x00010000)]	3
4	P2MP	2.2.2.2	[gid 131072 (0x00020000)]	1
6	P2MP	4.4.4.4	[gid 131072 (0x00020000)]	1
8	P2MP	5.5.5.5	[gid 131072 (0x00020000)]	1
2	P2MP	7.7.7.7	[gid 131072 (0x00020000)]	3
B	MP2MP	8.8.8.8	[mdt 49:222 0]	1

We will perform a basic verification of the mLDP control-plane for both VRFs, beginning with U2 since it's much simpler. When CSR7 has to send LSM traffic to other nodes in the MVPN, it sends traffic towards the MP2MP root. The MP2MP is bidirectional, so there is an upstream label to send traffic

towards the root, and a downstream label to receive traffic along the same tree. In this case, label 8025 is used to send traffic upstream towards CSR8.

```
CSR7#show mpls mldp database id B | section Upstream
Upstream client(s) :
  8.8.8.8:0      [Active]
    Expires      : Never          Path Set ID   : B
    Out Label (U) : 8025          Interface    : GigabitEthernet2.578*
    Local Label (D) : 7016        Next Hop     : 49.7.8.8
```

When CSR8 receives it, it must replicate the traffic in several directions towards the remote PEs. In this case, traffic arrives with local label 8025 from CSR7, so it is distributed to CSR10 and XRv12, which are in the paths towards CSR2 and CSR5, respectively. We also verify this in the LFIB which shows the LSM replication.

```
CSR8#show mpls mldp database opaque_type mdt 49:222 0 | begin Replicat
Replication client(s):
  10.10.10.10:0
    Uptime       : 2d03h          Path Set ID   : E
    Out label (D) : 10019         Interface    : GigabitEthernet2.580*
    Local label (U) : 8022        Next Hop     : 49.8.10.10
  7.7.7.7:0
    Uptime       : 2d03h          Path Set ID   : 11
    Out label (D) : 7016          Interface    : GigabitEthernet2.578*
    Local label (U) : 8025        Next Hop     : 49.7.8.7
  12.12.12.12:0
    Uptime       : 1d02h          Path Set ID   : 14
    Out label (D) : 92019         Interface    : GigabitEthernet2.582*
    Local label (U) : 8021        Next Hop     : 49.8.12.12
```

```
CSR8#show mpls forwarding-table labels 8025
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id Switched     interface
8025       10019      [mdt 49:222 0] 1144008     Gi2.580   49.8.10.10
           92019      [mdt 49:222 0] 619936     Gi2.582   49.8.12.12
```

First, we will follow the path to CSR2 and CSR4. CSR10 receives packets labeled 10019 from CSR8, which is for traffic heading down the MDT. This downstream label is swapped for 2017 and forwarded to CSR2 as shown in the mLDP database and LFIB. The same action occurs for label 4014 towards CSR4.

```
CSR10#show mpls mldp database opaque_type mdt 49:222 0 | begin Upstream
Upstream client(s) :
  8.8.8.8:0      [Active]
    Expires      : Never          Path Set ID   : 9
    Out Label (U) : 8022          Interface    : GigabitEthernet2.580*
    Local Label (D) : 10019       Next Hop     : 49.8.10.8
Replication client(s):
```

```

4.4.4.4:0
  Uptime           : 1d02h           Path Set ID   : D
  Out label (D)    : 4014           Interface    : GigabitEthernet2.540*
  Local label (U)  : 10020          Next Hop     : 49.4.10.4
2.2.2.2:0
  Uptime           : 00:42:07        Path Set ID   : 11
  Out label (D)    : 2017           Interface    : GigabitEthernet2.520*
  Local label (U)  : 10018          Next Hop     : 49.2.10.2

```

```

CSR10#show mpls forwarding-table labels 10019
Local      Outgoing   Prefix           Bytes Label   Outgoing   Next Hop
Label      Label      or Tunnel Id     Switched      interface
10019      4014          [mdt 49:222 0]  1238248      Gi2.540    49.4.10.4
           2017          [mdt 49:222 0]  33408        Gi2.520    49.2.10.2

```

CSR4 receives packets with label 4014 and replicated it to the LSM virtual interface bound to VRF U2. The LFIB also shows this label being removed with traffic associated with the proper VRF and LSP-VIF interface.

```

CSR4#show mpls mldp database opaque_type mdt 49:222 0 | begin Upstream
Upstream client(s) :
  10.10.10.10:0    [Active]
    Expires        : Never           Path Set ID   : 9
    Out Label (U)  : 10020          Interface    : GigabitEthernet2.540*
    Local Label (D): 4014           Next Hop     : 49.4.10.10
Replication client(s):
  MDT (VRF U2)
    Uptime         : 2d03h           Path Set ID   : A
    Interface      : Lspvif1

```

```

CSR4#show mpls forwarding-table labels 4014 detail
Local      Outgoing   Prefix           Bytes Label   Outgoing   Next Hop
Label      Label      or Tunnel Id     Switched      interface
4014       No Label    [mdt 49:222 0][V]  \
                                           0          aggregate/U2
MAC/Encaps=0/0, MRU=0, Label Stack{}, via Ls1
VPN route: U2
No output feature configured
Broadcast

```

For completeness, CSR2 does the exact same thing as CSR4. Label 2017 arrives, is popped, and the tunneled packet is delivered to VRF U2.

```

CSR2#show mpls mldp database opaque_type mdt 49:222 0 | begin Upstream
Upstream client(s) :
  10.10.10.10:0    [Active]
    Expires        : Never           Path Set ID   : 9
    Out Label (U)  : 10018          Interface    : GigabitEthernet2.520*

```

```

Local Label (D): 2017           Next Hop       : 49.2.10.10
Replication client(s):
MDT (VRF U2)
  Uptime           : 2d03h       Path Set ID   : A
  Interface        : Lspvif1

```

```

CSR2#show mpls forwarding-table labels 2017 detail
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id  Switched      interface
2017       No Label    [mdt 49:222 0] [V]    \
                                           0
                                           aggregate/U2
MAC/Encaps=0/0, MRU=0, Label Stack{}, via Ls1
VPN route: U2
No output feature configured
Broadcast

```

Going back to CSR8 (MP2MP root), packets also travel towards CSR5. XRv12 receives packets labeled 92019 from CSR8 and swaps this for label 93019 towards XRv13. The 0x00008 is the LSM-ID which is automatically generated when database state is created. Unlike XE's LFIB, XR's LFIB does not show the opaque information, but the LSM-ID instead. The LSM-ID isn't terribly significant except for cross-referencing different show/debug command outputs.

```

RP/0/0/CPU0:XRv12#show mpls mldp database opaquetype mdt 49:222
mLDP database
LSM-ID: 0x00008    Type: MP2MP    Uptime: 1d02h
FEC Root           : 8.8.8.8
Opaque decoded      : [mdt 49:222 0]
Upstream neighbor(s) :
  8.8.8.8:0 [Active] Uptime: 1d02h
    Next Hop       : 49.8.12.8
    Interface      : GigabitEthernet0/0/0/0.582
    Local Label (D) : 92019           Remote Label (U): 8021
Downstream client(s):
  LDP 13.13.13.13:0 Uptime: 1d02h
    Next Hop       : 49.12.13.13
    Interface      : GigabitEthernet0/0/0/0.523
    Remote label (D) : 93019           Local label (U) : 92018

```

```

RP/0/0/CPU0:XRv12#show mpls forwarding labels 92019
Local  Outgoing  Prefix      Outgoing  Next Hop      Bytes
Label  Label      or ID        Interface  Hop           Switched
-----
92019  93019        MLDP: 0x00008  Gi0/0/0/0.523  49.12.13.13    0

```

XRv3 performs a similar action by swapping 93019 for 5009 and forwarding the packet to CSR5.

```

RP/0/0/CPU0:XRv13#show mpls mldp database opaquetype mdt 49:222 0 | begin
Upstream

```

```

Upstream neighbor(s) :
  12.12.12.12:0 [Active] Uptime: 1d02h
    Next Hop      : 49.12.13.12
    Interface     : GigabitEthernet0/0/0/0.523
    Local Label (D) : 93019
    Remote Label (U) : 92018
Downstream client(s):
  LDP 5.5.5.5:0      Uptime: 1d02h
    Next Hop      : 49.5.13.5
    Interface     : GigabitEthernet0/0/0/0.553
    Remote label (D) : 5009
    Local label (U) : 93018

```

```

RP/0/0/CPU0:XRv13#show mpls forwarding labels 93019
Local   Outgoing   Prefix           Outgoing   Next Hop   Bytes
Label   Label      or ID           Interface  Next Hop   Switched
-----
93019   5009          MLDP: 0x00008   Gi0/0/0/0.553 49.5.13.5   0

```

Finally, CSR5 receives packets labeled 5009. The label is popped and the multicast traffic is delivered to the VRF customer.

```

CSR5#show mpls mldp database opaque_type mdt 49:222 0 | begin Upstream
Upstream client(s) :
  13.13.13.13:0    [Active]
    Expires        : Never
    Out Label (U)   : 93018
    Local Label (D) : 5009
    Path Set ID    : B
    Interface      : GigabitEthernet2.553*
    Next Hop       : 49.5.13.13
Replication client(s):
  MDT (VRF U2)
    Uptime         : 2d03h
    Path Set ID    : C
    Interface      : Lspvif1

```

```

CSR5#show mpls forwarding-table labels 5009 detail
Local   Outgoing   Prefix           Bytes Label   Outgoing   Next Hop
Label   Label      or Tunnel Id     Switched      interface
5009    No Label    [mdt 49:222 0] [V]  \
                                         0          aggregate/U2
      MAC/Encaps=0/0, MRU=0, Label Stack{}, via Ls1
      VPN route: U2
      No output feature configured
Broadcast

```

Using OAM, we can verify the LSPs are functional from CSR7. All three of the remote PEs are responding as expected. We can see that UMPLS did not introduce any snags into the mLDP design, but this success is somewhat coincidental. Like inter-AS option C, a mechanism may be necessary for core routers to have build MDTs properly. The PIM vector may be used in some designs (MVPN profile 0) so that core routers can resolve RPF in the “direction” of the remote PE. In this network, although not seen yet, BGP

is running on all core routers. While not a common design, this quickly fixes the issue of mLDP tree construction.

```
CSR7#ping mpls mldp mp2mp 8.8.8.8 mdt 49:222 0 verbose
mp2mp Root node addr 8.8.8.8
Opaque type MDT, oui:index 0x49:0222, mdtnum 0
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
    timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]
```

Type escape sequence to abort.

```
Request #1
!    size 72, reply addr 49.2.10.2, return code 3
!    size 72, reply addr 49.5.13.5, return code 3
!    size 72, reply addr 49.4.10.4, return code 3
```

```
Round-trip min/avg/max = 17/44/61 ms
Received 3 replies
```

Next, we will verify VRF U1 multicast connectivity. Because there are many different (but very similar) P2MP trees, we will only examine the IPv4 tree rooted at CSR2. We will verify this from tail to head, which is technically how mLDP LSPs are built, which means the signaling begins on CSR7 and CSR5. Focusing on CSR7, it receives an MVPN Type-1 route (I-PMSI) from CSR2 which describes the default MDT tunnel attributes. We can see it is an mLDP P2MP tunnel (tunnel type 2) rooted at 2.2.2.2. Based on this, CSR7 creates the tree upstream towards the root by advertising downstream labels in that direction. In this case, the label mapping message is sent to CSR8. Notice that there is no upstream label since P2MP trees are always unidirectional, so there is never a case where CSR7 can send traffic to CSR2 along this tree. Instead, CSR7 can only receive downstream traffic using label 7018 and deliver it to its MVPN customer.

```
CSR7#show bgp ipv4 mvpn vrf U1 route-type 1 2.2.2.2
BGP routing table entry for [1][49:107][2.2.2.2]/12, version 20
Paths: (1 available, best #1, table MVPNV4-BGP-Table, not advertised to EBGp
peer)
    Advertised to update-groups:
        2
    Refresh Epoch 1
    Local, (Received from a RR-client), imported path from
[1][49:102][2.2.2.2]/12 (global)
    2.2.2.2 (metric 30) from 2.2.2.2 (2.2.2.2)
        Origin incomplete, metric 0, localpref 100, valid, internal, best
        Community: no-export
        Extended Community: RT:49:102
        PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 17, label: exp-null,
tunnel parameters: 0600 0104 0202 0202 0007 0100 0400 0100 00
```

```
rx pathid: 0, tx pathid: 0x0
```

```
CSR7#show mpls mldp database id 3 | begin Upstream
Upstream client(s) :
  8.8.8.8:0      [Active]
    Expires      : Never          Path Set ID   : 3
    Out Label (U) : None          Interface    : GigabitEthernet2.578*
    Local Label (D): 7018         Next Hop     : 49.7.8.8
Replication client(s):
  MDT (VRF U1)
    Uptime       : 2d04h          Path Set ID   : None
    Interface    : Lspvif0
```

CSR5 does the exact same thing. It has the same BGP type-1 route from CSR2 (not shown again) and sends a label mapping message to XRv13. XRv13 sends a label mapping message to XRv12, who sends one to CSR8. Those steps are highly redundant and not shown. CSR5 is allowed to receive downstream traffic from XRv3 only.

```
CSR5#show mpls mldp database id 3 | begin Upstream
Upstream client(s) :
  13.13.13.13:0  [Active]
    Expires      : Never          Path Set ID   : 3
    Out Label (U) : None          Interface    : GigabitEthernet2.553*
    Local Label (D): 5008         Next Hop     : 49.5.13.13
Replication client(s):
  MDT (VRF U1)
    Uptime       : 2d04h          Path Set ID   : None
    Interface    : Lspvif0
```

CSR8 receives label mapping messages from CSR7 and XRv12, making it a branch point for LSM. It sends a label mapping to CSR10 with label 8001 to receive traffic, and replicates downstream traffic to CSR7 and XRv12 using labels 7018 and 92010, respectively.

```
CSR8#show mpls mldp database id 1 | begin Upstream
Upstream client(s) :
  10.10.10.10:0  [Active]
    Expires      : Never          Path Set ID   : 1
    Out Label (U) : None          Interface    : GigabitEthernet2.580*
    Local Label (D): 8001         Next Hop     : 49.8.10.10
Replication client(s):
  7.7.7.7:0
    Uptime       : 2d04h          Path Set ID   : None
    Out label (D) : 7018          Interface    : GigabitEthernet2.578*
    Local label (U): None         Next Hop     : 49.7.8.7
  12.12.12.12:0
    Uptime       : 1d03h          Path Set ID   : None
    Out label (D) : 92010         Interface    : GigabitEthernet2.582*
```

Local label (U): None Next Hop : 49.8.12.12

CSR10 sends a label mapping message to CSR2 with downstream label 10011. CSR2 is the root of the tree and has no upstream neighbors, so it programs label 10011 from CSR10 as the first hop in the delivery tree. Notice that CSR4 did not join this P2MP tree; this is because the VPN route-targets were not exchanged by CSR2 and CSR4. This was done intentionally to show that mLDP is smart enough to build the trees in accordance with the RT policies as the BGP AD routes carry the RTs.

```
CSR10#show mpls mldp database id 1 | begin Upstream
Upstream client(s) :
  2.2.2.2:0        [Active]
    Expires                : Never                      Path Set ID    : 1
    Out Label (U)        : None                      Interface      : GigabitEthernet2.520*
    Local Label (D) : 10011                      Next Hop       : 49.2.10.2
Replication client(s):
  8.8.8.8:0
    Uptime                : 2d04h                      Path Set ID    : None
    Out label (D)        : 8001                      Interface      : GigabitEthernet2.580*
    Local label (U) : None                      Next Hop       : 49.8.10.8
```

```
CSR2#show mpls mldp database id 1 | section Replic
Replication client(s):
  MDT (VRF U1)
    Uptime                : 2d04h                      Path Set ID    : None
    Interface             : Lspvif0
  10.10.10.10:0
    Uptime                : 1d03h                      Path Set ID    : None
    Out label (D)        : 10011                      Interface      : GigabitEthernet2.520*
    Local label (U) : None                      Next Hop       : 49.2.10.10
```

Using OAM, we verify this LSP. Traffic can only flow in one direction in an mLDP P2MP tree, so the ping must be originated at the root of the tree. Again, we see no significant impact due to UMPLS. XE doesn't have a built in code for auto-discovered P2MP trees within the OAM command syntax tree, but it is 0x1 as discussed in the MVPN section.

```
CSR2#ping mpls mldp p2mp 2.2.2.2 hex 0x01 00010000 verbose
p2mp Root node addr 2.2.2.2
Opaque type hex value (0x1), num hex digits 4
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
  timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]
```

Type escape sequence to abort.

```
Request #1
!     size 72, reply addr 49.7.8.7, return code 3
```

```
!      size 72, reply addr 49.5.13.5, return code 3
```

33.2.1.4 Inter-area TE and TE-FRR

Next, we will look at using RSVP-TE in conjunction with UMPLS LSPs. This is a good way to demonstrate inter-area TE techniques (or inter-level TE for ISIS). The most obvious method to do inter-area TE is to stitch tunnels together at the ABRs or L1/L2 routers. This is configuration intensive but offers the most control over paths in each area/level. It also implies that some tunnels may require LDP label exchange inside of them since the tunnels are P-P and PE-P. All LSRs are configured to accept targeted-LDP sessions as well as allow mLDP to use TE tunnels (not shown). Currently, CSR2 is the head-end of a TE tunnel to CSR10 [ID 100]. Normally these stitched tunnels require LDP, but in this case, LDP is not required. The real reason for needing LDP on stitched tunnels is to learn the label for the remote PE, but CSR2 is already learning it via BGP; for this reason, LDP is not needed on CSR2's tunnel. There is a second tunnel from CSR10 to CSR7 which does not require LDP for unicast connectivity since the tunnel destination is the PE. The goal is to ensure these two tunnels are stitched together properly.

```
! CSR2
ip explicit-path name EP_4_10 enable
  next-address 4.4.4.4
  next-address 10.10.10.10

interface Tunnel100
  description PATH TO R10 VIA R4
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 10.10.10.10
  tunnel mpls traffic-eng autoroute announce
  tunnel mpls traffic-eng path-option 10 explicit name EP_4_10

! CSR10
ip explicit-path name EP_8_12_7 enable
  next-address 8.8.8.8
  next-address 12.12.12.12
  next-address 7.7.7.7

interface Tunnel100
  description PATH TO R7 VIA R8 AND XRV12
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 7.7.7.7
  tunnel mpls traffic-eng autoroute announce
  tunnel mpls traffic-eng path-option 10 explicit name EP_8_12_7
```

We quickly verify that CSR2's tunnel is up to CSR10. The path cost of 60 suggests that the high-cost link to CSR4 is in use, as expected. We also include the label value of 4008 which is used to send traffic to CSR4.

```

CSR2#show mpls traffic-eng tunnels tunnel 100 | section Status|Label
Status:
  Admin: up      Oper: up      Path: valid      Signalling: connected
  path option 10, type explicit EP_4_10 (Basis for Setup, path weight 60)

InLabel  : -
OutLabel : GigabitEthernet2.524, 4008

```

The recursive routing lookups dictate the label usage. CSR2 still learns the route to CSR7 via BGP, which has a next-hop of the L1/L2 router and exiting BGP label of 10006. This is the reason LDP is not needed; CSR7 is already being provided the "middle" label so that the bottom label is not exposed to the tunnel tail. Since CSR10 is the next-hop of this BGP route and also the tunnel tail, the proper label is exposed to CSR10 and the LSP should function. The route to CSR10 points of a TE tunnel, which means the RSVP label of 4008 is used. The label stack is now {4008 10006}.

```

CSR2#show ip route 7.7.7.7
Routing entry for 7.7.7.7/32
  Known via "bgp 49", distance 200, metric 0, type internal
  Last update from 10.10.10.10 00:22:09 ago
  Routing Descriptor Blocks:
  * 10.10.10.10, from 10.10.10.10, 00:22:09 ago
    Route metric is 0, traffic share count is 1
    AS Hops 0
    MPLS label: 10006

```

```

CSR2#show ip route 10.10.10.10
Routing entry for 10.10.10.10/32
  Known via "isis", distance 115, metric 10, type level-1
  Redistributing via isis 49
  Last update from 10.10.10.10 on Tunnel100, 00:19:38 ago
  Routing Descriptor Blocks:
  * 10.10.10.10, from 10.10.10.10, 00:19:38 ago, via Tunnel100
    Route metric is 10, traffic share count is 1

```

CSR4 pops label 4008 on ingress to expose the next label, which is 10006, to CSR10.

```

CSR4#show mpls forwarding-table labels 4008
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
4008       Pop Label  2.2.2.2 100 [5]  257066      Gi2.540     49.4.10.10

```

The label operations on CSR10 are tricky. CSR10 does not install the BGP route to 7.7.7.7/32 since it has an IGP route for it via a TE tunnel. This means the TE label should be used. When merging BGP and LDP paths, we saw that the LFIB cleanly associated the labels in a swap operation. When interworking between BGP and TE labels, the LFIB is less clear. The BGP label is popped and the TE label is pushed, and the LFIB shows this correctly but only in the "detail" output. Label 8023 is the TE label from CSR8

which is effectively swapped onto the stack at CSR10. With BGP/LDP connections in the LFIB, this was more fluid.

```
CSR10#show bgp ipv4 unicast 7.7.7.7/32
BGP routing table entry for 7.7.7.7/32, version 4
Paths: (2 available, best #2, table default, RIB-failure(17))
  Advertised to update-groups:
    8
  Refresh Epoch 1
  Local
    7.7.7.7 (metric 20) from 12.12.12.12 (12.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Originator: 7.7.7.7, Cluster list: 12.12.12.12
      mpls labels in/out 10006/imp-null
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 8
  Local
    7.7.7.7 (metric 20) from 8.8.8.8 (8.8.8.8)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Originator: 7.7.7.7, Cluster list: 8.8.8.8
      mpls labels in/out 10006/imp-null
      rx pathid: 0, tx pathid: 0x0
```

```
CSR10#show ip route 7.7.7.7
Routing entry for 7.7.7.7/32
  Known via "isis", distance 115, metric 20, type level-2
  Redistributing via isis 49
  Last update from 7.7.7.7 on Tunnel100, 00:22:14 ago
  Routing Descriptor Blocks:
  * 7.7.7.7, from 7.7.7.7, 00:22:14 ago, via Tunnel100
    Route metric is 20, traffic share count is 1
```

```
CSR10#show mpls traffic-eng tunnels tunnel100 | include Label
InLabel   : -
OutLabel  : GigabitEthernet2.580, 8023
```

```
CSR10#show mpls forwarding-table labels 10006 detail
Local      Outgoing   Prefix          Bytes Label  Outgoing   Next Hop
Label      Label       or Tunnel Id    Switched     interface
10006      Pop Label   7.7.7.7/32      141850      Tu100      point2point
          MAC/Encaps=18/22, MRU=1500, Label Stack{8023}, via Gi2.580
          000C296AFFD5000C2949716481000DFC8847 01F57000
          No output feature configured
```

CSR8 and XRv12 perform basic label swap and pop (PHP) operations, respectively. This reveals the MPLS service label to the PE, CSR7. In our case, we are not tracing a particular MPLS service, so the IPv4 packet would be revealed.

```
CSR8#show mpls forwarding-table labels 8023
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Outgoing interface	Next Hop
8023	92020	10.10.10.10 100	[6] \ 155887	Gi2.582	49.8.12.12

```
RP/0/0/CPU0:XRv12#show mpls forwarding labels 92020
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
92020	Pop	100	Gi0/0/0/0.572	49.7.12.7	131093

We can verify the fully stitched LSP using OAM. The command output needs improvement, says the Cisco developer, as there wouldn't have been another third label in this stack on the first hop (even if it were not implicit-null). Ignoring the implicit-nulls that don't belong, the label stack is as we described it.

```
CSR2#traceroute mpls ipv4 7.7.7.7/32
```

more work needed here to demux the tfs subtlv and to display the right output
[snip]

Type escape sequence to abort.

```
0 49.2.4.2 MRU 1500 [Labels: 4008/implicit-null/10006 Exp: 0/0/0]
L 1 49.2.4.4 MRU 1500 [Labels: 10006 Exp: 0] 1 ms
L 2 49.4.10.10 MRU 1500 [Labels: 8023/implicit-null Exp: 0/0] 2 ms
L 3 49.8.10.8 MRU 1500 [Labels: 92020 Exp: 0] 5 ms
L 4 49.8.12.12 MRU 1500 [Labels: implicit-null Exp: 0] 35 ms
! 5 49.7.12.7 25 ms
```

There is an alternative method to inter-area TE which is a relatively new feature. While tunnel stitching works, it is configuration-intensive and requires configuring many devices. Loose-hop expansion was designed to carry a TE tunnel through all the ABRs in the network by only configuring the head-end. The PCALC expansion algorithm comes into play which essentially computes paths between the loose hops based on cost, which means the administrator has less control. This is the same technique used for inter-AS MPLS option C except the tunnel is crossing IGP boundaries, not AS boundaries. We will configure this feature on CSR5 building a path to CSR2.

```
! CSR5
ip explicit-path name EP_14_10_2_LOOSE enable
  next-address loose 14.14.14.14
  next-address loose 10.10.10.10
  next-address loose 2.2.2.2
```

```
interface Tunnel101
  description PATH TO R2 VIA XRV14 AND R10
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 2.2.2.2
```

```
tunnel mpls traffic-eng path-option 10 explicit name EP_14_10_2_LOOSE
```

To see this tunnel in action, we will enable debugging on CSR5 (head), XRv14 (local L1/L2), and CSR10 (remote L1/L2). Each one of these routers have special responsibilities for ensuring the path is built between areas/levels since the L1-only and L2-only routers have limited visibility.

```
! CSR5, CSR10, and XRv14
debug mpls traffic-eng path lookup
```

CSR5's output begins with a traditional PCALC invocation with an explicit-path. The tunnel destination node of 5.5.5.5 cannot be found in the local L1 topology, so PCALC evaluates the first loose hop. This first loose hop of 14.14.14.14 does exist in the local L1 topology and is an L1/L2 router. As a result, CSR5 computes a path to XRv14 within the L1 topology. Notice that CSR5 expands the ERO in the PATH message to include the strict hops within the L1 topology to reach XRv14, and uses this new ERO for signaling.

```
! CSR5
TE-PCALC-API: 5.5.5.5_1->2.2.2.2_101 {7}: P2P LSP Path Lookup called
TE-PCALC: 5.5.5.5_1->2.2.2.2_101 {7}: Path Request Info
  Flags:  IP_EXPLICIT_PATH  METRIC_TE
  IP explicit-path: Supplied
    14.14.14.14 Loose
    10.10.10.10 Loose
    2.2.2.2 Loose
  bw 0, min_bw 0, metric: 0
  setup_pri 7, hold_pri 7
  affinity_bits 0x0, affinity_mask 0xFFFF
TE-PCALC-PATH: 5.5.5.5_1->2.2.2.2_101 {7}: Area (isis level-1) Path Lookup
begin
TE-PCALC-PATH: Area (isis level-1): Dest ip addr 2.2.2.2 not found
TE-PCALC-PATH: lsr_exists:first Loose Hop is to addr 14.14.14.14
TE-PCALC-PATH:Path from 0000.0000.0005.00 -> 0000.0000.0014.00:
  49.5.14.5->49.5.14.14 (admin_weight=10):
    num_hops 2, accumulated_aw 10, min_bw 100000
TE-PCALC-PATH: 5.5.5.5_1->2.2.2.2_101 {7}: Freeing rrr_path_setup_t
TE-PCALC-PATH: 5.5.5.5_1->2.2.2.2_101 {7}: Free all paths in path tree
TE-PCALC: Verify Path Lookup: 5.5.5.5_1->2.2.2.2_101 {7}: (protocol nil area
nil)
  Flags:  METRIC_TE
  Last Strict Router: 14.14.14.14
  sub-lsp weight:0 (Total LSP weight:10)
  Hop List:
    49.5.14.14
    14.14.14.14
    10.10.10.10 Loose
    2.2.2.2 Loose
TE-PCALC-VERIFY: VERIFY to 14.14.14.14 BEGIN:
```



```

TE-PCALC-VERIFY: Verify:
TE-PCALC-VERIFY: 0000.0000.0005.00, 5.5.5.5 points to
TE-PCALC-VERIFY: 0000.0000.0014.00, 49.5.14.14
TE-PCALC-VERIFY: VERIFY to 14.14.14.14 PASSED
TE-PCALC-PATH: 5.5.5.5_1->2.2.2.2_101 {7}: Area (isis level-1) Path Lookup
end: path found
TE-PCALC-API: 5.5.5.5_1->2.2.2.2_101 {7}: P2P LSP Path Lookup result: success

```

```

CSR5#show ip rsvp sender detail filter session-type 7 destination 2.2.2.2 |
section outgoing

```

```

ERO: (outgoing)
  49.5.14.14 (Strict IPv4 Prefix, 8 bytes, /32)
  14.14.14.14 (Strict IPv4 Prefix, 8 bytes, /32)
  10.10.10.10 (Loose IPv4 Prefix, 8 bytes, /32)
  2.2.2.2 (Loose IPv4 Prefix, 8 bytes, /32)

```

As a mechanism of loop prevention, the RSVP PATH RRO is introduced. This is similar to the RSVP RESV RRO except is used to record PHOPs along the path so that downstream routers don't accidentally compute paths through those nodes again. When using loose-hop expansion, multiple routers are running PCALC, which is not common for TE; normally the head-end runs PCALC, encodes the TE path in an RSVP PATH ERO, and signals the path using RSVP. When XRv14 receives this PATH message, it can see that it needs to compute a path to CSR10, the first loose hop in the ERO. It calls the PCALC "expand" mechanism which computes the path to this loose hop using any dynamic path that satisfies the tunnel constraints. XRv14 selects the path from CSR1, to CSR8 to CSR10. The ERO shown below shows CSR10 as a loose hop, and the outgoing path identifies all the L2 topology hops and retrains CSR2 as the last loose-hop. Technically this last loose-hop for the tunnel destination is not required, but is added for clarity. XRv14 also excludes CSR5 from the computation (doesn't matter since the computation is happening in L2, so 5.5.5.5 is an unknown node) because it is carried in the RSVP PATH RRO shown below. The XR debug is verbose but unedited since this is the only place where it is shown. The details are highlighted but I leave all of the output for reference.

```

! XRv14
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_show_path:8987 (Tunnel ID:
101): Tunnel101 Path Setup [0000.0000.0000.00] 10.10.10.10: (No IGP
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_show_path:8998 (Tunnel ID:
101): bw 0, rev bw 0, min_bw 0, metric: 0
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_show_path:9002 (Tunnel ID:
101): setup_pri 7, hold_pri 7
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_show_path:9007 (Tunnel ID:
101): affinity_bits 0x0, affinity_mask 0xffff
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_show_path:9011 (Tunnel ID:
101): path type unidirectional
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_show_path:9016 (Tunnel ID:
101): endpoint switching capability psc1
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_show_path:9019 (Tunnel ID:
101): endpoint encoding packet

```

```

te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_show_path:9023 (Tunnel ID:
101): transit switching capability psc1,
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_show_path:9026 (Tunnel ID:
101): transit encoding packet
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_lsr_expand:17597 (Tunnel ID:
101): TE-PCALC_PATH: exclude_address[0] = 5.5.5.5
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_reset_exclude_list:9775: TE-
PCALC_SPF: ##### new gen 14
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_reset_exclude_list:9775: TE-
PCALC_SPF: ##### new gen 15
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_alloc_hop_data:528 (Tunnel
ID: 101): alloc_hop_data: 0x106e4370
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_exclude_from_spf:9938 (Tunnel
ID: 101): TE-PCALC_PATH: exclude_from_spf checking area IS-IS 49 level-2
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_exclude_from_spf:9955 (Tunnel
ID: 101): hop_type = 2
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_exclude_from_spf:9973 (Tunnel
ID: 101): TE-PCALC_PATH: exclude_path: ipaddr 5.5.5.5 area IS-IS 49 level-2
not known!
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_free_hoplist:550 (Tunnel ID:
101): free_hop_data: 0x106e4370
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_get_path_internal:9525
(Tunnel ID: 101): TE-PCALC_PATH:Path from 0000.0000.0014.00 ->
0000.0000.0010.00:
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_get_path_internal:9559
(Tunnel ID: 101): TE-PCALC_PATH: 49.8.10.8->49.8.10.10
(admin_weight=30):
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_get_path_internal:9559
(Tunnel ID: 101): TE-PCALC_PATH: 49.1.8.1->49.1.8.8 (admin_weight=20):
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_get_path_internal:9559
(Tunnel ID: 101): TE-PCALC_PATH: 49.1.14.14->49.1.14.1
(admin_weight=10):
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_get_path_internal:9594
(Tunnel ID: 101): TE-PCALC_PATH: Create a path structure for new path
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_alloc_path_setup_data:501
(Tunnel ID: 101): alloc_path_setup: 0x106ecb8c
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_alloc_hop_data:528 (Tunnel
ID: 101): alloc_hop_data: 0x106e4370
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_alloc_hop_data:528 (Tunnel
ID: 101): alloc_hop_data: 0x106e426c
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_alloc_hop_data:528 (Tunnel
ID: 101): alloc_hop_data: 0x106e4308
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_alloc_hop_data:528 (Tunnel
ID: 101): alloc_hop_data: 0x106e433c
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_get_path_internal:9713
(Tunnel ID: 101): TE-PCALC_PATH: num_hops 4, accumulated_aw 30, min_bw
100000
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_lsr_expand:17783 (Tunnel ID:
101): Loose 10.10.10.10 expands to:

```

```

te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_lsr_expand:17793 (Tunnel ID:
101): 49.1.14.1
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_lsr_expand:17793 (Tunnel ID:
101): 49.1.8.8
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_lsr_expand:17793 (Tunnel ID:
101): 49.8.10.10
te_control[1044]: DBG-PATH-LOOKUP[1]: rrr_pcalc_lsr_expand:17793 (Tunnel ID:
101): 10.10.10.10
te_control[1044]: DBG-PATH-LOOKUP[1]: te_s2l_expand_ero_h:1197
(T:101,L:1,P:2147483647,E:5.5.5.5,SI:0,SO:0.0.0.0,S:5.5.5.5,D:2.2.2.2,CT:7,V:
1610612736): expanded path found, bw: 0, min_bw: 100000

```

```

RP/0/0/CPU0:XRv14#show mpls traffic-eng tunnels 101 | begin Expansion
    ERO Expansion Info:
    IS-IS 49 level-2, Metric 30 (TE), Affinity 0x0, Mask 0xffff, Queries 16
    Last Better Path Query was received 00:08:28 ago

    Outgoing:
    Explicit Route:
    Strict, 49.1.14.1
    Strict, 49.1.8.8
    Strict, 49.8.10.10
    Strict, 10.10.10.10
    Loose, 2.2.2.2
    Record Route:
    IPv4 49.5.14.5, flags 0x0
    [snip]

```

CSR10 receives the PATH message from CSR8 which contains CSR2 as a loose hop. Like XRv14, CSR10 invokes the LSP path expansion process to determine a dynamic path to CSR2. Also note that existing transit routers, such as CSR8, CSR1, and XRv14, are removed from consideration. It also tries to exclude CSR5 but cannot see its TE RID as it resides in a remote L1 topology. Again, CSR10 knows about these previous hops since they are contained in the RSVP PATH RRO. CSR2 is directly connected to CSR10, which is selected as the shortest path.

```

! CSR10
TE-PCALC-API: 5.5.5.5_1->2.2.2.2_101 {7}: LSP Path Expand called
TE-PCALC: 5.5.5.5_1->2.2.2.2_101 {7}: Path Request Info
  Flags: END_SWCAP_UNKNOWN
  IP explicit-path: None (dynamic)
  bw 0, min_bw 0, metric: 0
  setup_pri 7, hold_pri 7
  affinity_bits 0x0, affinity_mask 0x0
TE-PCALC-PATH: 5.5.5.5_1->2.2.2.2_101 {7}: rrr_pcalc_lsr_expand: Exclude
node: 8.8.8.8 (intf: 49.8.10.8)
TE-PCALC-PATH: 5.5.5.5_1->2.2.2.2_101 {7}: rrr_pcalc_lsr_expand: Exclude
node: 1.1.1.1 (intf: 49.1.8.1)

```

```

TE-PCALC-PATH: 5.5.5.5_1->2.2.2.2_101 {7}: rrr_pcalc_lsr_expand: Exclude
node: 14.14.14.14 (intf: 49.1.14.14)
TE-PCALC-PATH: 5.5.5.5_1->2.2.2.2_101 {7}: rrr_pcalc_lsr_expand: Can't get
router ID addr for 49.5.14.5
TE-PCALC-PATH: 5.5.5.5_1->2.2.2.2_101 {7}: Area (isis level-1) Path Lookup
begin
TE-PCALC-PATH: exclude_path: system_id 0-0-0-0-0-0-0 not known!
TE-PCALC-PATH: exclude_path: system_id 0-0-0-0-0-0-0 not known!
TE-PCALC-PATH: exclude_path: system_id 0-0-0-0-0-0-0 not known!
TE-PCALC-PATH: exclude_path: system_id 0-0-0-0-0-0-0 not known!
TE-PCALC-PATH: Path from 0000.0000.0010.00 -> 0000.0000.0002.00:
  49.2.10.10->49.2.10.2 (admin_weight=10):
    num_hops 2, accumulated_aw 10, min_bw 100000
TE-PCALC-PATH: 10.10.10.10_1->2.2.2.2_101 {7}: Area (isis level-1) Path
Lookup end: path found
2.2.2.2 expands to:
49.2.10.2
2.2.2.2
TE-PCALC-API: 10.10.10.10_1->2.2.2.2_101 {7}: LSP Path Expand result: success
TE-PCALC-PATH: 10.10.10.10_1->2.2.2.2_101 {7}: Freeing rrr_path_setup_t

CSR10#show ip rsvp sender detail filter session-type 7 destination 2.2.2.2 |
section [ER]RO
  ERO: (incoming)
    49.8.10.10 (Strict IPv4 Prefix, 8 bytes, /32)
    10.10.10.10 (Strict IPv4 Prefix, 8 bytes, /32)
    2.2.2.2 (Loose IPv4 Prefix, 8 bytes, /32)
  ERO: (outgoing)
    49.2.10.2 (Strict IPv4 Prefix, 8 bytes, /32)
    2.2.2.2 (Strict IPv4 Prefix, 8 bytes, /32)
  RRO:
    49.8.10.8/32, Flags:0x0 (No Local Protection)
    49.1.8.1/32, Flags:0x0 (No Local Protection)
    49.1.14.14/32, Flags:0x0 (No Local Protection)
    49.5.14.5/32, Flags:0x0 (No Local Protection)

```

For extra debugging, we can look at the RSVP dump messages to see the incoming PATH message on CSR10. This reveals the ERO and RRO, both of which are interest for loose-hop expansion.

```

! CSR10
Incoming Path:
  version:1 flags:0000 cksum:89E2 ttl:252 reserved:0 length:248
  SESSION          type 7 length 16:
    Tun Dest:      2.2.2.2  Tun ID: 101  Ext Tun ID: 5.5.5.5
  HOP              type 1 length 12:
    Hop Addr: 49.8.10.8 LIH: 0x0000000F
  TIME_VALUES      type 1 length 8 :
    Refresh Period (msec): 30000
  EXPLICIT_ROUTE   type 1 length 28:

```

```
49.8.10.10 (Strict IPv4 Prefix, 8 bytes, /32)
10.10.10.10 (Strict IPv4 Prefix, 8 bytes, /32)
2.2.2.2 (Loose IPv4 Prefix, 8 bytes, /32)
[snip]
```

```
RECORD_ROUTE          type 1 length 36:
49.8.10.8/32, Flags:0x0 (No Local Protection)
49.1.8.1/32, Flags:0x0 (No Local Protection)
49.1.14.14/32, Flags:0x0 (No Local Protection)
49.5.14.5/32, Flags:0x0 (No Local Protection)
```

Because the RRO is included in the PATH message, this implies that the tail-end router (CSR2) will be able to see every PHOP along the path, not just the local PHOP.

```
CSR2#show ip rsvp sender detail filter session-type 7 destination 2.2.2.2 |
section RRO
```

```
RRO:
49.2.10.10/32, Flags:0x0 (No Local Protection)
49.8.10.8/32, Flags:0x0 (No Local Protection)
49.1.8.1/32, Flags:0x0 (No Local Protection)
49.1.14.14/32, Flags:0x0 (No Local Protection)
49.5.14.5/32, Flags:0x0 (No Local Protection)
```

Unlike tunnel stitching, we can use OAM to test the entire length of the tunnel, versus each segment. From the head-end, we verify the entire path.

```
CSR5#traceroute mpls traffic-eng tunnel 101
Tracing MPLS TE Label Switched Path on Tunnel101, timeout is 2 seconds
[snip]
```

Type escape sequence to abort.

```
0 49.5.14.5 MRU 1500 [Labels: 94011 Exp: 0]
L 1 49.5.14.14 MRU 1500 [Labels: 1012 Exp: 0] 1 ms
L 2 49.1.14.1 MRU 1500 [Labels: 8018 Exp: 0] 50 ms
L 3 49.1.8.8 MRU 1500 [Labels: 10015 Exp: 0] 24 ms
L 4 49.8.10.10 MRU 1500 [Labels: implicit-null Exp: 0] 24 ms
! 5 49.2.10.2 24 ms
```

When we configured this tunnel, we did not configure a mechanism for routing. Auto-route announce and forwarding adjacency cannot work on these tunnels; notice that the destination TE ID cannot be seen by the head-end router, which breaks the auto-route announce logic. Forwarding-adjacency also would not make sense since it builds a link in the graph, which is not suitable between OSPF areas or IS-IS levels.

```
! CSR5
interface Tunnel101
 tunnel mpls traffic-eng autoroute announce
```

```

CSR5#show mpls traffic-eng autoroute
MPLS TE autorouting enabled
  destination 0-0-0-0-0-0-0, area isis level-1, has 1 tunnels
    Tunnel101 (load balancing metric 0, nexthop 2.2.2.2)
              (flags: Announce)

```

Instead, we can use autoroute destination on this tunnel since the destination actually is the real tail end. This strategy does not work on stitched tunnels, though. This means that CSR5 and CSR2 have a complete, end-to-end connection. This means that UMPLS architecture is totally bypassed; we no longer need BGP IPv4 labeled-unicast. The downside is that this solution defeats the purpose of UMPLS, which is to achieve better scalability. Stitched TE tunnels between ABRs in the core, or between ABRs/PEs in the aggregation islands, may scale better than PE to PE tunnels in this design. TE also may have negative effects on MVPN, which has been discussed many times in many other chapters.

```

! CSR5
interface Tunnel101
  tunnel mpls traffic-eng autoroute destination

```

Verifying L3VPN connectivity, we can see traffic from CSR5 to CSR2 has only two labels: the bottom BGP VPNv6 label and the top RSVP-TE label.

```

RP/0/0/CPU0:XRv11#traceroute vrf U2 ::2:6:6:6 source ::2:11:11:11
Type escape sequence to abort.
Tracing the route to ::2:6:6:6

 1  fd00:192:168:205::5 0 msec 0 msec 0 msec
 2  2049:49:5:14::14 [MPLS: Labels 94011/2007 Exp 0] 29 msec 89 msec 49 msec
 3  ::ffff:49.1.14.1 [MPLS: Labels 1012/2007 Exp 0] 69 msec 29 msec 49 msec
 4  ::ffff:49.1.8.8 [MPLS: Labels 8018/2007 Exp 0] 49 msec 49 msec 59 msec
 5  ::ffff:49.8.10.10 [MPLS: Labels 10015/2007 Exp 0] 49 msec 39 msec 39 msec
 6  fd00:192:168:202::2 [MPLS: Label 2007 Exp 0] 79 msec 69 msec 89 msec
 7  fd00:192:168:202::6 89 msec 49 msec 49 msec

```

Next, we will examine TE-FRR with UMPLS. CSR1, CSR9, CSR8, CSR4, and CSR10 are running BFD between all links upon which they connect. BFD is disabled on all other routers. The goal will be to provide NHOP and NNHOP protection in various points in the network. Since tunnel stitching is just two ordinary TE tunnels, providing FRR for them is nothing new. Instead, we will configure the inter-area tunnel with the headend at CSR5 to request FRR. We can quickly confirm this is successful by verifying label bindings for each hop in the RSVP RESV RRO. Additionally, we will configure a bandwidth reservation to ensure bandwidth can be backed up between areas/levels.

```

! CSR5
interface Tunnel101
  tunnel mpls traffic-eng fast-reroute
  tunnel mpls traffic-eng bandwidth 5000

```

```
CSR5#show ip rsvp reservation filter session-type 7 destination 2.2.2.2
Destination      Tun Sender      TunID LSPID Next Hop      I/F      Fi Serv BPS
2.2.2.2          5.5.5.5         101   19    49.5.14.14      Gi2.554  SE LOAD 5M
```

```
CSR5#show ip rsvp reservation detail filter session-type 7 destination
2.2.2.2 | begin RRO
```

RRO:

```
14.14.14.14/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 94010
49.5.14.14/32, Flags:0x0 (No Local Protection)
  Label subobject: Flags 0x1, C-Type 1, Label 94010
1.1.1.1/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 1017
49.1.8.1/32, Flags:0x0 (No Local Protection)
  Label subobject: Flags 0x1, C-Type 1, Label 1017
8.8.8.8/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 8024
49.8.10.8/32, Flags:0x0 (No Local Protection)
  Label subobject: Flags 0x1, C-Type 1, Label 8024
10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 10023
49.2.10.10/32, Flags:0x0 (No Local Protection)
  Label subobject: Flags 0x1, C-Type 1, Label 10023
2.2.2.2/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 3
49.2.10.2/32, Flags:0x0 (No Local Protection)
  Label subobject: Flags 0x1, C-Type 1, Label 3
```

Status:

Policy: Accepted. Policy source(s): MPLS/TE

First, we will provide basic NHOP protection in the core to protect the link from CSR1 to CSR8. CSR1 creates this NHOP tunnel and acts as the PLR. It will backup 7 Mbps of bandwidth from any pool. We quickly verify the FRR database on the PLR and the RSVP RESV RRO on the headend to ensure the tunnel is protected. We can use CSR1's local label shown in the RRO to see the specific FRR database entry.

```
! CSR1
ip explicit-path name EP_AVOID_R1_R8_LINK enable
  exclude-address 49.1.8.1
  exclude-address 49.1.8.8

interface Tunnel110
  description INTER-AREA NHOP PROTECTION
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 8.8.8.8
  tunnel mpls traffic-eng backup-bw 7000
  tunnel mpls traffic-eng path-option 10 explicit name EP_AVOID_R1_R8_LINK
```

```

interface GigabitEthernet2.518
  mpls traffic-eng backup-path Tunnel110

CSR1#show mpls traffic-eng tunnels role head destination 8.8.8.8 | section
RSVP Path
  RSVP Path Info:
    My Address: 49.1.9.1
    Explicit Route: 49.1.9.9 49.8.9.8 8.8.8.8
    Record Route: NONE
    Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits

CSR5#show ip rsvp reservation detail filter session-type 7 destination
2.2.2.2 | section RRO:
  RRO:
    14.14.14.14/32, Flags:0x20 (No Local Protection, Node-id)
      Label subobject: Flags 0x1, C-Type 1, Label 94010
    49.5.14.14/32, Flags:0x0 (No Local Protection)
      Label subobject: Flags 0x1, C-Type 1, Label 94010
    1.1.1.1/32, Flags:0x25 (Local Prot Avail/Has BW/to NHOP, Node-id)
      Label subobject: Flags 0x1, C-Type 1, Label 1019
    49.1.8.1/32, Flags:0x5 (Local Prot Avail/Has BW/to NHOP)
      Label subobject: Flags 0x1, C-Type 1, Label 1019
    8.8.8.8/32, Flags:0x20 (No Local Protection, Node-id)
      Label subobject: Flags 0x1, C-Type 1, Label 8018
    49.8.10.8/32, Flags:0x0 (No Local Protection)
      Label subobject: Flags 0x1, C-Type 1, Label 8018
    10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
      Label subobject: Flags 0x1, C-Type 1, Label 10021
    49.2.10.10/32, Flags:0x0 (No Local Protection)
      Label subobject: Flags 0x1, C-Type 1, Label 10021
    2.2.2.2/32, Flags:0x20 (No Local Protection, Node-id)
      Label subobject: Flags 0x1, C-Type 1, Label 3
    49.2.10.2/32, Flags:0x0 (No Local Protection)
      Label subobject: Flags 0x1, C-Type 1, Label 3

```

```

R1#show mpls traffic-eng fast-reroute database labels 1019 | begin P2P LSP
P2P LSP midpoint frr information:
LSP identifier          In-label Out intf/label    FRR intf/label    Status
-----
5.5.5.5 101 [21]       1019     Gi2.518:8018       Tu110:8018       ready

```

We also quickly verify that the backup bandwidth is working. The RRO indicates that the NHOP protection "has BW" but we can verify it on the PLR as well. For extra verification, we use OAM to verify the path avoids the link between CSR1 and CSR8. This is also shown in the RSVP PATH ERO shown above.

```

CSR1#show mpls traffic-eng tunnels tunnel 110 backup
INTER-AREA NHOP PROTECTION

```



```
LSP Head, Admin: up, Oper: up
Tun ID: 110, LSP ID: 1, Source: 1.1.1.1
Destination: 8.8.8.8
Fast Reroute Backup Provided:
  Protected i/fs: Gi2.518
  Protected LSPs/Sub-LSPs: 1, Active: 0
  Backup BW: any pool; limit: 7000 kbps, inuse: 5000 kbps (BWP inuse: 0
kbps)
  Backup flags: 0x0
```

```
CSR1#traceroute mpls traffic-eng tunnel 110
Tracing MPLS TE Label Switched Path on Tunnel110, timeout is 2 seconds
[snip]
```

```
Type escape sequence to abort.
 0 49.1.9.1 MRU 1500 [Labels: 9014 Exp: 0]
L 1 49.1.9.9 MRU 1500 [Labels: implicit-null Exp: 0] 3 ms
! 2 49.8.9.8 34 ms
```

We can also configure NNHOP protection across IS-IS level boundaries using loose-hop expansion. We will configure this tunnel on CSR8 going to CSR2, effectively bypassing CSR10. This tunnel will use a combination of strict and loose hops to reach the final destination. We cannot simply exclude CSR10 since CSR8 won't find an intra-area/level path to CSR2. We ensure the tunnel comes up and avoids CSR10 completely using a more explicit method.

```
! CSR8
ip explicit-path name EP_AVOID_CSR10 enable
  next-address 9.9.9.9
  next-address loose 4.4.4.4
  next-address loose 2.2.2.2

interface Tunnel111
  description INTER_AREA NNHOP PROTECTION
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 2.2.2.2
  tunnel mpls traffic-eng backup-bw 9000
  tunnel mpls traffic-eng path-option 10 explicit name EP_AVOID_CSR10

interface GigabitEthernet2.580
  mpls traffic-eng backup-path Tunnel111
```

```
CSR8#show mpls traffic-eng tunnels tunnel 111 | section RSVP Path
  RSVP Path Info:
    My Address: 49.8.9.8
    Explicit Route: 49.8.9.9 49.4.9.4 4.4.4.4 2.2.2.2*
    Record Route:
    Tspec: ave rate=0 kbits, burst=1000 bytes, peak rate=0 kbits
```

First, we check the RSVP RESV RRO on the headend to ensure the NNHOP tunnel backs up the TE tunnel, also offering bandwidth. The PLR shows the FRR database entry as well, and we reference it by local label as derived by the RRO. RSVP is smart enough to know that a tunnel offers NHOP protection when its destination is the NHOP. Likewise, when a tunnel's destination is the NNHOP, regardless of how the tunnel was configured, RSVP considers it an NNHOP protection tunnel.

```
CSR5#show ip rsvp reservation detail filter session-type 7 destination
2.2.2.2 | section RRO:
```

RRO:

```
14.14.14.14/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 94010
49.5.14.14/32, Flags:0x0 (No Local Protection)
  Label subobject: Flags 0x1, C-Type 1, Label 94010
1.1.1.1/32, Flags:0x25 (Local Prot Avail/Has BW/to NHOP, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 1019
49.1.8.1/32, Flags:0x5 (Local Prot Avail/Has BW/to NHOP)
  Label subobject: Flags 0x1, C-Type 1, Label 1019
8.8.8.8/32, Flags:0x2D (Local Prot Avail/Has BW/to NNHOP, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 8018
49.8.10.8/32, Flags:0xD (Local Prot Avail/Has BW/to NNHOP)
  Label subobject: Flags 0x1, C-Type 1, Label 8018
10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 10021
49.2.10.10/32, Flags:0x0 (No Local Protection)
  Label subobject: Flags 0x1, C-Type 1, Label 10021
2.2.2.2/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 3
49.2.10.2/32, Flags:0x0 (No Local Protection)
  Label subobject: Flags 0x1, C-Type 1, Label 3
```

```
CSR8#show mpls traffic-eng fast-reroute database labels 8018 | begin P2P LSP
P2P LSP midpoint frr information:
```

LSP identifier	In-label	Out intf/label	FRR intf/label	Status
5.5.5.5 101 [21]	8018	Gi2.580:10021	Tu111:implicit-n ready	

For completeness, we quickly check the backup bandwidth and use OAM to verify the data plane. Of the 9 Mbps available, 5 Mbps are allocated for the tunnel configured on CSR5.

```
CSR8#show mpls traffic-eng tunnels tunnel 111 backup
INTER_AREA NNHOP PROTECTION
LSP Head, Admin: up, Oper: up
Tun ID: 111, LSP ID: 3, Source: 8.8.8.8
Destination: 2.2.2.2
Fast Reroute Backup Provided:
Protected i/fs: Gi2.580
```

```
Protected LSPs/Sub-LSPs: 1, Active: 0
Backup BW: any pool; limit: 9000 kbps, inuse: 5000 kbps (BWP inuse: 0
kbps)
Backup flags: 0x0
```

```
CSR8#traceroute mpls traffic-eng tunnel 111
Tracing MPLS TE Label Switched Path on Tunnel111, timeout is 2 seconds
[snip]
```

```
Type escape sequence to abort.
 0 49.8.9.8 MRU 1500 [Labels: 9017 Exp: 0]
L 1 49.8.9.9 MRU 1500 [Labels: 4001 Exp: 0] 1 ms
L 2 49.4.9.4 MRU 1500 [Labels: 10022 Exp: 0] 3 ms
L 3 49.4.10.10 MRU 1500 [Labels: implicit-null Exp: 0] 21 ms
! 4 49.2.10.2 21 ms
```

First, we will test the NHOP tunnel. The FRR label that CSR1 (PLR) pushes to tunnel traffic through CSR9 is 9014 as shown below. We use this, along with label 8018, to see traffic in the FRR tunnel using EPC.

```
CSR1#show mpls traffic-eng tunnels tunnel 110 | include Label
InLabel   : -
OutLabel  : GigabitEthernet2.519, 9014
```

BFD takes about 2.7 seconds to detect a failure in this setup. We change the encapsulation on CSR8 towards CSR1 after starting a ping within the L3VPN. After those 3 seconds, the packets are forwarded in the TE tunnel for a short time (only one packet in this instance, since the tunnel head is very close to the PLR). The single packet that ended up in the TE-FRR tunnel is highlighted green in the ping output. The label stack is {9014 8018 2007} as highlighted in cyan. This shows that NHOP protection works as expected when using inter-area tunnels. The reason we cannot reasonably use traceroute is because convergence happens so fast with dynamic or loose-explicit paths.

```
RP/0/0/CPU0:XRv11#ping vrf U2 ::2:6:6:6 source ::2:11:11:11 size 500 count
1000
Type escape sequence to abort.
Sending 1000, 500-byte ICMP Echos to ::2:6:6:6, timeout is 2 seconds:
!!!...! [snip]
```

```
CSR9#show monitor capture CAP buffer detailed | begin 534
 13 534      4.955005  00:0C:29:FB:A3:39 -> 00:0C:29:E0:4F:84 MPLS unicast
0000:  000C29E0 4F84000C 29FBA339 81000DBF  ..).O...)..9....
0010:  88470233 603901F5 2039007D 713B6000  .G.3`9...9.}q;`.
0020:  000001CC 3A3B0000 00000000 00000002  ....:;.....
0030:  00110011 00110000 00000000 00000002  .....
```

We will bring the CSR1-CSR8 link back up, reoptimize the tunnel, and test NNHOP protection next. This is a little more interesting since both the protected and protecting tunnels are inter-area with

loose-hops, but the RSVP signaling process is still the same. We will break the link between CSR10 and CSR8, then check the RSVP information. We don't need to use EPC because I managed to quickly capture the relevant information via show-commands (with the NHOP test above, I wasn't fast enough). CSR5 shows the NNHOP tunnel actually "in use". The NHOP tunnel is still available but is no longer "in use". This shows that the head-end was still notified by the PLR that FRR was activated. We won't do all of the TE-FRR verification details as this is sufficient to show that loose-hop tunnels can be both protected and protecting tunnels. Additional TE verification commands are show in the dedicated TE section.

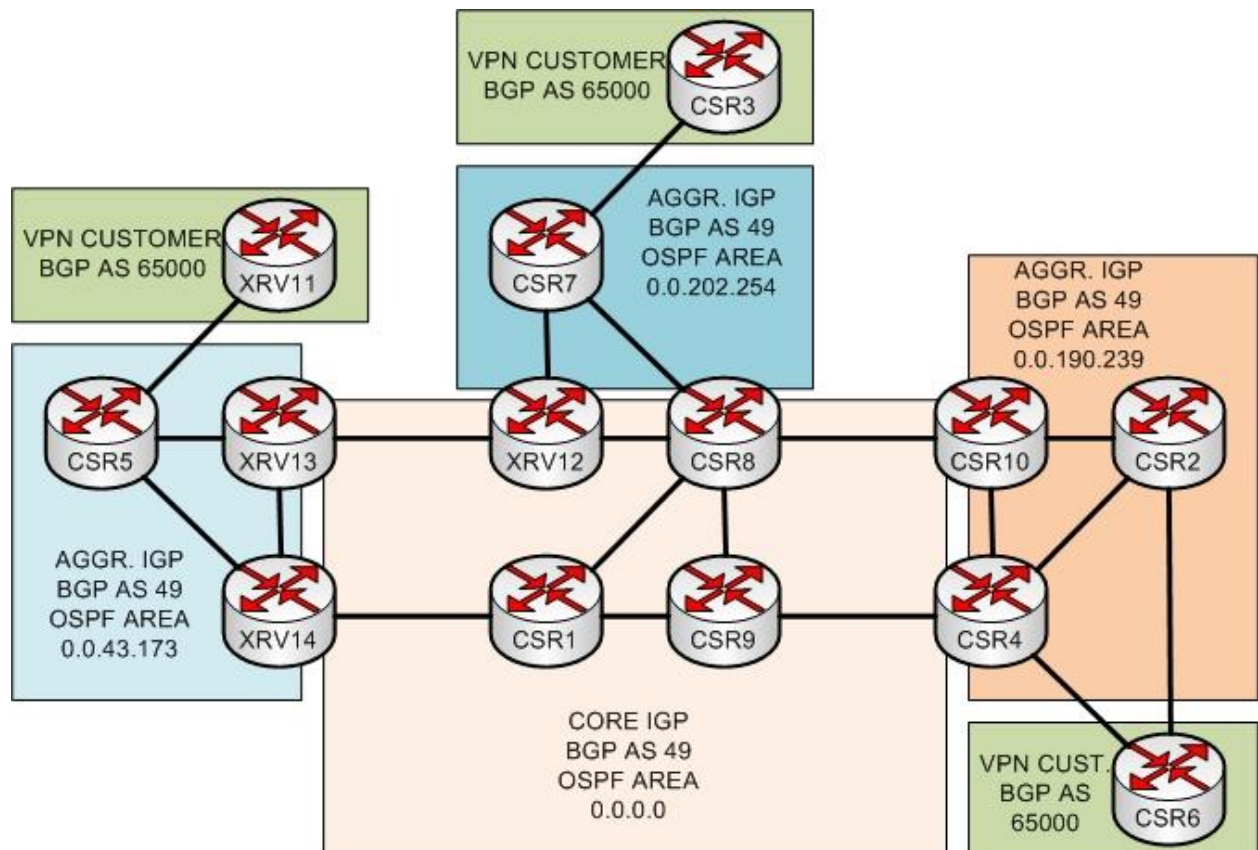
```
CSR5#show ip rsvp reservation detail filter session-type 7 destination
2.2.2.2 | section RRO
RRO:
 14.14.14.14/32, Flags:0x20 (No Local Protection, Node-id)
   Label subobject: Flags 0x1, C-Type 1, Label 94010
 49.5.14.14/32, Flags:0x0 (No Local Protection)
   Label subobject: Flags 0x1, C-Type 1, Label 94010
 1.1.1.1/32, Flags:0x25 (Local Prot Avail/Has BW/to NHOP, Node-id)
   Label subobject: Flags 0x1, C-Type 1, Label 1021
 49.1.8.1/32, Flags:0x5 (Local Prot Avail/Has BW/to NHOP)
   Label subobject: Flags 0x1, C-Type 1, Label 1021
 8.8.8.8/32, Flags:0x2F (Local Prot Avail/In Use/Has BW/to NNHOP, Node-id)
   Label subobject: Flags 0x1, C-Type 1, Label 8024
 49.8.9.8/32, Flags:0xF (Local Prot Avail/In Use/Has BW/to NNHOP)
   Label subobject: Flags 0x1, C-Type 1, Label 8024
 10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
   Label subobject: Flags 0x1, C-Type 1, Label 10015
 49.2.10.10/32, Flags:0x0 (No Local Protection)
   Label subobject: Flags 0x1, C-Type 1, Label 10015
 2.2.2.2/32, Flags:0x20 (No Local Protection, Node-id)
   Label subobject: Flags 0x1, C-Type 1, Label 3
 49.2.10.2/32, Flags:0x0 (No Local Protection)
   Label subobject: Flags 0x1, C-Type 1, Label 3
```

Additional Reading – Reference configurations "umpls-isis"

33.2.2 OSPF (summarized)

We also analyze UMPLS using OSPF instead of IS-IS. The concepts are all identical, so this section is summarized to highlight the differences. The area boundaries are in the same places, so the routers that were L1/L2 routers for IS-IS are now ABRs for OSPF. XRv21 and CSR8 are also ABRs now; despite CSR7 being in a different IS-IS area earlier, it was still a level-2 router, so there was no additional complexity involved; UMPLS did not come into play. In this case, the topology is a little more complex as a result of added ABRs. There are different area types in use here mostly to demonstrate that UMPLS can work with any area type. Unlike IS-IS, we have to carefully select how to advertise loopback interfaces in areas. I use a combination of aggregation area and core area advertisement, along with LSA3 filtering to allow routes to leak. Area 0.0.43.173 and area 0.0.202.254 follow a classic UMPLS model where the ABRs don't allow routes to leak between areas and set next-hop-self on reflected BGP routes for IPv4

labeled-unicast. Area 0.0.190.239 is a relaxed variant where the OSPF inter-area routes leak into area 0 (as with a normal OSPF design) so the LDP LSP can be used. The BGP verifications are the same and are not repeated, since we will see them when verifying the MPLS services. The area 0.0.43.173 ABR configuration points of interest are shown below; notice that the loopbacks are advertised into the NSSA (not traditional) and are leaked into area 0.



```
! XRV13 (XRV14 is identical with different RPL)
route-policy RPL_LEAK_LOOPBACK
  if destination in (13.13.13.13/32) then
    pass
  endif
end-policy

router ospf 49
 network point-to-point
 area 0
  route-policy RPL_LEAK_LOOPBACK in

 area 0.0.43.173
  nssa no-summary
  interface Loopback0
  passive enable
```

The ABRs for area 0.0.202.254 are slightly different. The loopbacks are advertised into area 0 and are leaked outbound into all non-zero areas. This is similar logic as above except in the reverse direction. Additionally, all LSA3s are filtered from non-zero areas into area 0, which means BGP will be adjusting the next-hop. Setting the next-hop in an RPL doesn't appear to work for iBGP, even with iBGP modifications enforced, so we ignore this on XRv12. On CSR8, the OSPF filtering logic is identical with different syntax.

```
! XRv12
route-policy RPL_DROP
  drop
end-policy

route-policy RPL_LEAK_LOOPBACK
  if destination in (12.12.12.12/32) then
    pass
  endif
end-policy

router ospf 49
  network point-to-point
  area 0
    route-policy RPL_DROP in
    route-policy RPL_LEAK_LOOPBACK out
  interface Loopback0
    passive enable

area 0.0.202.254

! CSR8
interface Loopback0
  ip ospf 49 area 0

ip prefix-list PL_DROP seq 5 deny 0.0.0.0/0 le 32
ip prefix-list PL_LEAK_LOOPBACK seq 5 permit 8.8.8.8/32

router ospf 49
  area 0 filter-list prefix PL_DROP in
  area 0 filter-list prefix PL_LEAK_LOOPBACK out
```

The ABRs for area 0.0.190.239 has no filtering applied, other than the stub area configuration. This means all routers inside the non-zero areas can flood into area 0, implying that the ABRs do not need to adjust the BGP next-hop when advertising IPv4 labeled-unicast prefixes into the core. Since the ABRs are only advertising default routes, they must adjust the BGP next-hop when advertising routes to the PEs inside area 0.0.190.239 (the only PE is CSR2).

```
! CSR4 and CSR10
interface Loopback0
```

```
ip ospf 49 area 0.0.190.239

router ospf 49
area 0.0.190.239 stub no-summary
```

33.2.2.1 L3VPN

We will trace some L3VPN LSPs to ensure UMPLS is working correctly. First, we will test VRF U2 connectivity between XRv11 and CSR3 for IPv6. CSR5 has a VPNv6 route for ::2:3:3:3/128 with label 7010. This is the VPN label which, in UMPLS, never changes. The BGP next-hop is an IPv4 address of 7.7.7.7.

```
CSR5#show bgp vpnv6 unicast vrf U2 ::2:3:3:3/128
BGP routing table entry for [49:205]::2:3:3:3/128, version 232
Paths: (1 available, best #1, table U2)
  Advertised to update-groups:
    1
  Refresh Epoch 2
  65000, imported path from [49:207]::2:3:3:3/128 (global)
    ::FFFF:7.7.7.7 (metric 2) (via default) from 7.7.7.7 (7.7.7.7)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:49:207
      mpls labels in/out nolabel/7010
      rx pathid: 0, tx pathid: 0x0
```

CSR5's route to 7.7.7.7/32 is via XRv14. I adjusted local-preference on XRv14 to 200 earlier (it was not shown for brevity, though), because I wanted to use XRv14 as well, since XRv13 was in the transit path for all the IS-IS tests. The label for this destination is 94004, which was allocated by XRv14. Its next-hop is 14.14.14.14 for which CSR5 should have an IGP route and LDP label.

```
CSR5#show ip route 7.7.7.7
Routing entry for 7.7.7.7/32
  Known via "bgp 49", distance 200, metric 0, type internal
  Last update from 14.14.14.14 00:20:16 ago
  Routing Descriptor Blocks:
    * 14.14.14.14, from 14.14.14.14, 00:20:16 ago
      Route metric is 0, traffic share count is 1
      AS Hops 0
      MPLS label: 94004
```

```
CSR5#show bgp ipv4 unicast 7.7.7.7/32
BGP routing table entry for 7.7.7.7/32, version 162
Paths: (2 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    14.14.14.14 (metric 2) from 14.14.14.14 (14.14.14.14)
      Origin incomplete, metric 0, localpref 200, valid, internal, best
```

```

    Originator: 7.7.7.7, Cluster list: 14.14.14.14, 8.8.8.8
    mpls labels in/out nolabel/94004
    rx pathid: 0, tx pathid: 0x0
Refresh Epoch 1
Local
  13.13.13.13 (metric 2) from 13.13.13.13 (13.13.13.13)
    Origin incomplete, metric 0, localpref 100, valid, internal
    Originator: 7.7.7.7, Cluster list: 13.13.13.13, 8.8.8.8
    mpls labels in/out nolabel/93003
    rx pathid: 0, tx pathid: 0

```

The route to XRv14's loopback is an OSPF route via a non-TE tunnel interface, which means an LDP label will be used as we predicted above. Since XRv14 is directly connected, that label value is implicit-null. The label stack is now {94004 7010}.

```

CSR5#show ip route 14.14.14.14
Routing entry for 14.14.14.14/32
  Known via "ospf 49", distance 110, metric 2, type intra area
  Last update from 49.5.14.14 on GigabitEthernet2.554, 01:30:45 ago
  Routing Descriptor Blocks:
  * 49.5.14.14, from 14.14.14.14, 01:30:45 ago, via GigabitEthernet2.554
    Route metric is 2, traffic share count is 1

CSR5#show mpls ldp bindings 14.14.14.14 32 neighbor 14.14.14.14
  lib entry: 14.14.14.14/32, rev 58
    remote binding: lsr: 14.14.14.14:0, label: imp-null

```

When XRv14 receives this packet, it swaps the label from 94004 to 8002. The reason the path via XRv12 is invalid is because there is no route to 7.7.7.7/32 without causing a recursive routing loop. XR appears incapable of adjusting next-hops in RPLs for iBGP neighbors, which we attempted to configure on XRv12. “next-hop-self” works when applied to neighbors but not inside a custom RPL. As such, CSR8 is the ingress point for area 0.0.202.254. The label stack is now {8002 7010}.

```

RP/0/0/CPU0:XRv14#show bgp ipv4 labeled-unicast 7.7.7.7/32
[snip]
Paths: (2 available, best #1)
  Advertised to peers (in unique update groups):
    5.5.5.5
  Path #1: Received by speaker 0
  Advertised to peers (in unique update groups):
    5.5.5.5
Local
  8.8.8.8 (metric 3) from 8.8.8.8 (7.7.7.7)
    Received Label 8002
    Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best
    Received Path ID 0, Local Path ID 1, version 122

```



```

    Originator: 7.7.7.7, Cluster list: 8.8.8.8
    Path #2: Received by speaker 0
    Not advertised to any peer
    Local
    7.7.7.7 (metric 3) from 12.12.12.12 (7.7.7.7)
    Received Label 3
    Origin incomplete, metric 0, localpref 100, valid, internal
    Received Path ID 0, Local Path ID 0, version 0
    Originator: 7.7.7.7, Cluster list: 12.12.12.12

```

Next, XRv14 needs to lookup the route for the BGP next-hop, which is 8.8.8.8 via OSPF towards CSR1. The corresponding LDP label from CSR1 is 1016. The label stack is now {1016 8002 7010}.

```

RP/0/0/CPU0:XRv14#show route ipv4 8.8.8.8
Routing entry for 8.8.8.8/32
  Known via "ospf 49", distance 110, metric 3, type intra area
  Routing Descriptor Blocks
    49.1.14.1, from 8.8.8.8, via GigabitEthernet0/0/0/0.514
    Route metric is 3
  No advertising protos.

RP/0/0/CPU0:XRv14#show mpls ldp bindings 8.8.8.8/32 neighbor 1.1.1.1
8.8.8.8/32, rev 68
  Local binding: label: 94002
  Remote bindings: (2 peers)
    Peer          Label
    -----
    1.1.1.1:0     1016

```

CSR1 is a pure P router and only looks at the top-most label since there are no ECMP paths. OSPF says the next-hop is towards CSR8 directly, so implicit-null is advertised by CSR8, forcing CSR1 to pop the top-most label. The label stack is now {8002 7010}.

```

CSR1#show ip route 8.8.8.8
Routing entry for 8.8.8.8/32
  Known via "ospf 49", distance 110, metric 2, type intra area
  Last update from 49.1.8.8 on GigabitEthernet2.518, 01:44:40 ago
  Routing Descriptor Blocks:
    * 49.1.8.8, from 8.8.8.8, 01:44:40 ago, via GigabitEthernet2.518
    Route metric is 2, traffic share count is 1

```

```

CSR1#show mpls forwarding-table labels 1016
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id  Switched     interface
1016      Pop Label  8.8.8.8/32   80067        Gi2.518   49.1.8.8

```

When CSR8 receives the packets, the BGP label is removed. Notice that the BGP route is not in the routing table due to a RIB failure; this is because CSR8 has an OSPF route to 7.7.7.7 as they share an area. CSR7 allocates implicit-null via LDP for prefix 7.7.7.7/32, causing CSR8 to pop the BGP label. The label stack is now 7010, which reveals the VPN label of 7010 to CSR7. Although BGP and LDP both dictate that the out-label operation is “pop”, the LDP implicit-null is the one that is actually processed.

```
CSR8#show bgp ipv4 unicast 7.7.7.7/32
BGP routing table entry for 7.7.7.7/32, version 75
Paths: (1 available, best #1, table default, RIB-failure(17))
  Advertised to update-groups:
    6
  Refresh Epoch 7
  Local, (Received from a RR-client)
    7.7.7.7 (metric 2) from 7.7.7.7 (7.7.7.7)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      mpls labels in/out 8002/imp-null
      rx pathid: 0, tx pathid: 0x0
```

```
CSR8#show ip route 7.7.7.7
Routing entry for 7.7.7.7/32
  Known via "ospf 49", distance 110, metric 2, type intra area
  Last update from 49.7.8.7 on GigabitEthernet2.578, 01:47:55 ago
  Routing Descriptor Blocks:
    * 49.7.8.7, from 7.7.7.7, 01:47:55 ago, via GigabitEthernet2.578
      Route metric is 2, traffic share count is 1
```

```
CSR8#show mpls ldp bindings 7.7.7.7 32 neighbor 7.7.7.7
lib entry: 7.7.7.7/32, rev 22
  remote binding: lsr: 7.7.7.7:0, label: imp-null
```

On CSR7, the LFIB removes all labels and delivers the IPv6 traffic to CSR3 inside VRF U2.

```
CSR7#show mpls forwarding-table labels 7010 detail
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id  Switched     interface
7010       No Label  ::2:3:3:3/128[V] 7498         Gi2.207     FE80::3
MAC/Encaps=22/22, MRU=1504, Label Stack{}
000C29D781FE000C29664C2C81000DD1810000CF86DD
VPN route: U2
No output feature configured
```

We can confirm the label stack at each hop using traceroute as shown below.

```
RP/0/0/CPU0:XRv11#traceroute vrf U2 ::2:3:3:3 source ::2:11:11:11
Type escape sequence to abort.
Tracing the route to ::2:3:3:3
```

```

1  fd00:192:168:205::5 9 msec 9 msec 119 msec
2  2049:49:5:14::14 [MPLS: Labels 94004/7010 Exp 0] 139 msec 49 msec 39 msec
3  ::ffff:49.1.14.1 [MPLS: Labels 1016/8002/7010 Exp 0] 39 msec 39 msec 39
msec
4  ::ffff:49.1.8.8 [MPLS: Labels 8002/7010 Exp 0] 39 msec 29 msec 59 msec
5  fd00:192:168:207::7 [MPLS: Label 7010 Exp 0] 39 msec 39 msec 39 msec
6  fd00:192:168:207::3 39 msec 39 msec 39 msec

```

Next, we will trace the LSP from CSR6 to XRv11 inside VRF U2. We will notice some asymmetry in the forwarding path due to BGP adjustments along the way. First, we verify that CSR6 (CE) is sending traffic to 2.11.11.11 via CSR4 versus CSR2. This is a result of the BGP local-preference changes shown earlier.

```

CSR6#show bgp vpnv4 unicast vrf U2 2.11.11.11/32
BGP routing table entry for 49:202:2.11.11.11/32, version 76
Paths: (2 available, best #2, table U2)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  49 49
    192.168.202.2 (via vrf U2) from 192.168.202.2 (2.2.2.2)
      Origin incomplete, localpref 100, valid, external
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  49 49
    192.168.204.4 (via vrf U2) from 192.168.204.4 (4.4.4.4)
      Origin incomplete, localpref 200, valid, external, best
      rx pathid: 0, tx pathid: 0x0

```

The same command on CSR4 shows us the VPN label allocations. The remote PE (CSR5) allocates label 5002) for this customer prefix. This label never changes.

```

CSR4#show bgp vpnv4 unicast vrf U2 2.11.11.11/32
BGP routing table entry for 49:204:2.11.11.11/32, version 142
Paths: (1 available, best #1, table U2)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  65000, imported path from 49:205:2.11.11.11/32 (global)
    5.5.5.5 (metric 6) (via default) from 7.7.7.7 (7.7.7.7)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:49:205
      Originator: 5.5.5.5, Cluster list: 7.7.7.7
      mpls labels in/out nolaabel/5002
      rx pathid: 0, tx pathid: 0x0

```

The route to 5.5.5.5, the VPNv4 next-hop, is a BGP route which carries label 93000. This is allocated by XRv13, the remote ABR. XRv13 is performing the “correct” role of a seamless MPLS ABR by adjusting the BGP labeled-unicast next-hop to itself.

```
CSR4#show ip route 5.5.5.5
Routing entry for 5.5.5.5/32
  Known via "bgp 49", distance 200, metric 0, type internal
  Last update from 13.13.13.13 00:45:45 ago
  Routing Descriptor Blocks:
    * 13.13.13.13, from 8.8.8.8, 00:45:45 ago
      Route metric is 0, traffic share count is 1
      AS Hops 0
      MPLS label: 93000
```

The route to 13.13.13.13 is an OSPF route via CSR9, which is bound to LDP label 9003. The label stack is now {9003 93000 5002}.

```
CSR4#show ip route 13.13.13.13
Routing entry for 13.13.13.13/32
  Known via "ospf 49", distance 110, metric 6, type inter area
  Last update from 49.4.9.9 on GigabitEthernet2.549, 01:25:21 ago
  Routing Descriptor Blocks:
    * 49.4.9.9, from 13.13.13.13, 01:25:21 ago, via GigabitEthernet2.549
      Route metric is 6, traffic share count is 1
```

```
CSR4#show mpls ldp bindings 13.13.13.13 32 neighbor 9.9.9.9
lib entry: 13.13.13.13/32, rev 96
remote binding: lsr: 9.9.9.9:0, label: 9003
```

All of the core routers are involved in this LSP. CSR9, CSR1, and CSR8 all are P routers along this path and perform basic label swaps of LDP labels. XRv12 is the PHP router along the LDP LSP and pops the LDP label to reveal the IPv4 labeled-unicast label of 93000 to XRv13. The show commands are displayed below, and the label stack becomes {93000 5002}.

```
CSR9#show mpls forwarding-table labels 9003
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id  Switched     interface
9003      1001      13.13.13.13/32  1458         Gi2.519   49.1.9.1
```

```
CSR1#show mpls forwarding-table labels 1001
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id  Switched     interface
1001      8000      13.13.13.13/32  1884         Gi2.518   49.1.8.8
```

```
CSR8#show mpls forwarding-table labels 8000
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id  Switched     interface
```

```
8000      92002      13.13.13.13/32      286255      Gi2.582      49.8.12.12
```

```
RP/0/0/CPU0:XRv12#show mpls forwarding labels 92002
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
92002	Pop	13.13.13.13/32	Gi0/0/0/0.523	49.12.13.13	764630

When XRv13 receives packets with label 93000, it pops the label. The "Received Label" of 3 means implicit-null but does not apply here since XRv13 has an OSPF route for 5.5.5.5/32, thus using the LDP label. It is also implicit-null, and as we have seen many times before, can be tricky to understand which "implicit-null" is being used. This reveals label 5002 to the remote PE. I constantly show this as a "trick" because the in-label is BGP allocated, but the out-label cannot be used since the route is not installed via BGP. In this case, the LDP and BGP labels are both 3, but this is not always the case.

```
RP/0/0/CPU0:XRv13#show bgp ipv4 labeled-unicast 5.5.5.5/32
```

```
[snip]
```

```
Paths: (1 available, best #1)
```

```
  Advertised to update-groups (with more than one peer):
```

```
    0.1
```

```
  Path #1: Received by speaker 0
```

```
  Advertised to update-groups (with more than one peer):
```

```
    0.1
```

```
  Local, (Received from a RR-client)
```

```
    5.5.5.5 (metric 2) from 5.5.5.5 (5.5.5.5)
```

```
      Received Label 3
```

```
      Origin incomplete, metric 0, localpref 100, valid, internal, best,  
group-best
```

```
      Received Path ID 0, Local Path ID 1, version 69
```

```
RP/0/0/CPU0:XRv13#show route ipv4 5.5.5.5/32
```

```
Routing entry for 5.5.5.5/32
```

```
  Known via "ospf 49", distance 110, metric 2, type intra area
```

```
  Routing Descriptor Blocks
```

```
    49.5.13.5, from 5.5.5.5, via GigabitEthernet0/0/0/0.553
```

```
      Route metric is 2
```

```
  No advertising protos.
```

```
RP/0/0/CPU0:XRv13#show mpls ldp bindings 5.5.5.5/32 neighbor 5.5.5.5
```

```
5.5.5.5/32, rev 43
```

```
  Local binding: label: 93000
```

```
  Remote bindings: (2 peers)
```

Peer	Label
5.5.5.5:0	ImpNull

When CSR5 receives packets with label 5002, it delivers them to XRv11 inside VRF U2 for prefix 2.11.11.11/32 as expected.

```
CSR5#show mpls forwarding-table labels 5002 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
5002       No Label   2.11.11.11/32[V] 4328         Gi2.205    192.168.205.11
MAC/Encaps=22/22, MRU=1504, Label Stack{}
000C2950BAB7000C295F11A181000DDF810000CD0800
VPN route: U2
No output feature configured
```

As usual, we can confirm this LSP using traceroute below. The CEF table on the ingress PE within the VPN also shows the imposition label stack for quick verification.

```
CSR4#show ip cef vrf U2 2.11.11.11
2.11.11.11/32
  nexthop 49.4.9.9 GigabitEthernet2.549 label 9003 93000 5002
```

```
CSR6#traceroute vrf U2 2.11.11.11 source 2.6.6.6
Type escape sequence to abort.
Tracing the route to 2.11.11.11
VRF info: (vrf in name/id, vrf out name/id)
 1 192.168.204.4 1 msec 1 msec 0 msec
 2 49.4.9.9 [MPLS: Labels 9003/93000/5002 Exp 0] 121 msec 33 msec 33 msec
 3 49.1.9.1 [MPLS: Labels 1001/93000/5002 Exp 0] 31 msec 27 msec 32 msec
 4 49.1.8.8 [MPLS: Labels 8000/93000/5002 Exp 0] 34 msec 34 msec 32 msec
 5 49.8.12.12 [MPLS: Labels 92002/93000/5002 Exp 0] 26 msec 32 msec 26 msec
 6 49.12.13.13 [MPLS: Labels 93000/5002 Exp 0] 34 msec 33 msec 33 msec
 7 192.168.205.5 [MPLS: Label 5002 Exp 0] 32 msec 26 msec 31 msec
 8 192.168.205.11 26 msec * 30 msec
```

33.2.2.2 L2VPN

Like IS-IS, the verification for L2VPN is based mostly on the transport path plus the binding of the PW labels. Tracing the LSP from CSR6 to CSR3 inside the L2VPN, we can determine the PW label by checking the AToM bindings. In this case, the remote PW label is 7034.

```
CSR2#show l2vpn atom binding 7.7.7.7
Destination Address: 7.7.7.7, VC ID: 567
Local Label: 2018
  Cbit: 1, VC Type: Ethernet, GroupID: n/a
  MTU: 1500, Interface Desc: n/a
  VCCV: CC Type: RA [2], TTL [3]
  CV Type: LSPV [2]
Remote Label: 7034
  Cbit: 1, VC Type: Ethernet, GroupID: n/a
  MTU: 1500, Interface Desc: n/a
```

```
VCCV: CC Type: RA [2], TTL [3]
      CV Type: LSPV [2]
```

We will quickly determine the transport label stack as well. The route to CSR7, the remote PW endpoint, is a BGP route with label 10029 via CSR10. The route to 10.10.10.10 is an OSPF route bound to implicit-null, also via CSR10. The label stack is now {10029 7034}.

```
CSR2#show ip route 7.7.7.7
Routing entry for 7.7.7.7/32
  Known via "bgp 49", distance 200, metric 0, type internal
  Last update from 10.10.10.10 01:05:16 ago
  Routing Descriptor Blocks:
  * 10.10.10.10, from 10.10.10.10, 01:05:16 ago
    Route metric is 0, traffic share count is 1
    AS Hops 0
    MPLS label: 10029

CSR2#show ip route 10.10.10.10
Routing entry for 10.10.10.10/32
  Known via "ospf 49", distance 110, metric 2, type intra area
  Last update from 49.2.10.10 on GigabitEthernet2.520, 01:55:27 ago
  Routing Descriptor Blocks:
  * 49.2.10.10, from 10.10.10.10, 01:55:27 ago, via GigabitEthernet2.520
    Route metric is 2, traffic share count is 1

CSR2#show mpls ldp bindings 10.10.10.10 32 neighbor 10.10.10.10
lib entry: 10.10.10.10/32, rev 61
remote binding: lsr: 10.10.10.10:0, label: imp-null
```

CSR10 selects CSR8 as the BGP bestpath towards 7.7.7.7/32 due to having a lower neighbor ID (worst tie breaker). As seen earlier, the route via XRv12 is invalid due to the infinite recursive lookups that would occur. This swaps label 10029 for 8002; we confirm this in the LFIB also. The label stack is now {8002 7034}.

```
CSR10#show bgp ipv4 unicast 7.7.7.7/32
BGP routing table entry for 7.7.7.7/32, version 82
Paths: (2 available, best #2, table default)
  Advertised to update-groups:
    4
  Refresh Epoch 1
  Local
    7.7.7.7 (metric 2) from 12.12.12.12 (12.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Originator: 7.7.7.7, Cluster list: 12.12.12.12
      mpls labels in/out 10029/imp-null
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 5
```

Local

```
8.8.8.8 (metric 2) from 8.8.8.8 (8.8.8.8)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  Originator: 7.7.7.7, Cluster list: 8.8.8.8
  mpls labels in/out 10029/8002
  rx pathid: 0, tx pathid: 0x0
```

CSR10#show mpls forwarding-table labels 10029

Local	Outgoing	Prefix	Bytes Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched	interface	
10029	8002	7.7.7.7/32	338367	Gi2.580	49.8.10.8

As seen earlier in the L3VPN verification, CSR8 pops this BGP label due to receiving an LDP-bound implicit-null for 7.7.7.7/32. The LDP label is used because the route to prefix 7.7.7.7/32 is an IGP route. This reveals the PW label of 7034 to CSR7.

CSR8#show ip route 7.7.7.7

Routing entry for 7.7.7.7/32

Known via "ospf 49", distance 110, metric 2, type intra area

Last update from 49.7.8.7 on GigabitEthernet2.578, 01:47:55 ago

Routing Descriptor Blocks:

* 49.7.8.7, from 7.7.7.7, 01:47:55 ago, via GigabitEthernet2.578

Route metric is 2, traffic share count is 1

CSR8#show mpls ldp bindings 7.7.7.7 32 neighbor 7.7.7.7

lib entry: 7.7.7.7/32, rev 22

remote binding: lsr: 7.7.7.7:0, label: imp-null

We can verify the transport label path using MPLS OAM. Ignoring the extraneous "implicit-nulls", the label stack is as we verified it above. Looking at the L2VPN PW details, we can see the label stack including the PW label as well.

CSR2#traceroute mpls ipv4 7.7.7.7/32 source 2.2.2.2

more work needed here to demux the tfs subtlv and to display the right output
[snip]

Type escape sequence to abort.

```
0 49.2.10.2 MRU 1500 [Labels: implicit-null/10029 Exp: 0/0]
L 1 49.2.10.10 MRU 1500 [Labels: 8002 Exp: 0] 1 ms
L 2 49.8.10.8 MRU 1500 [Labels: implicit-null Exp: 0] 29 ms
! 3 49.7.8.7 20 ms
```

CSR2#show l2vpn atom vc vcid 567 destination 7.7.7.7 detail | include label_stack

Output interface: Gi2.520, imposed label stack {10029 7034}

We will also verify the PW from CSR7 to CSR5 for extra practice. The PW label is 5009 as shown by the AToM bindings below.

```
CSR7#show l2vpn atom binding 5.5.5.5 | include Label
Local Label: 7021
Remote Label: 5009
```

The route to the PW endpoint of 5.5.5.5 is a BGP route, implying that we use the BGP label to reach it. Notice that the reason CSR8 was selected was due to having a lower neighbor ID.

```
CSR7#show ip route 5.5.5.5
Routing entry for 5.5.5.5/32
  Known via "bgp 49", distance 200, metric 0, type internal
  Last update from 8.8.8.8 01:14:05 ago
  Routing Descriptor Blocks:
    * 8.8.8.8, from 8.8.8.8, 01:14:05 ago
      Route metric is 0, traffic share count is 1
      AS Hops 0
      MPLS label: 8004
```

```
CSR7#show bgp ipv4 unicast 5.5.5.5/32
BGP routing table entry for 5.5.5.5/32, version 143
Paths: (2 available, best #2, table default)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    12.12.12.12 (metric 2) from 12.12.12.12 (12.12.12.12)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Originator: 5.5.5.5, Cluster list: 12.12.12.12, 13.13.13.13
      mpls labels in/out nolabel/92012
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 3
  Local
    8.8.8.8 (metric 2) from 8.8.8.8 (8.8.8.8)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Originator: 5.5.5.5, Cluster list: 8.8.8.8, 13.13.13.13
      mpls labels in/out nolabel/8004
      rx pathid: 0, tx pathid: 0x0
```

The route to the BGP next-hop of 8.8.8.8 is an OSPF route, implying that the LDP label should be pushed atop the BGP one. The LDP label is implicit-null, so the label stack becomes {8004 5009}.

```
CSR7#show ip route 8.8.8.8
Routing entry for 8.8.8.8/32
  Known via "ospf 49", distance 110, metric 2, type inter area
  Last update from 49.7.8.8 on GigabitEthernet2.578, 01:17:49 ago
  Routing Descriptor Blocks:
```

```
* 49.7.8.8, from 8.8.8.8, 01:17:49 ago, via GigabitEthernet2.578
Route metric is 2, traffic share count is 1
```

```
CSR7#show mpls ldp bindings 8.8.8.8 32 neighbor 8.8.8.8
lib entry: 8.8.8.8/32, rev 145
remote binding: lsr: 8.8.8.8:0, label: imp-null
```

Checking the LFIB, we can see that label 8004 is swapped for label 93000, with label 92002 pushed on top of that. The swap is conducted by BGP since the remote label from either XRv13 or XR14 was allocated by BGP also. CSR5 selects XRv13 as the bestpath due to the peer having a lower neighbor ID.

```
CSR8#show mpls forwarding-table labels 8004 detail
Local      Outgoing   Prefix      Bytes Label  Outgoing   Next Hop
Label      Label      or Tunnel Id Switched     interface
8004       93000      5.5.5.5/32  519868      Gi2.582    49.8.12.12
MAC/Encaps=18/26, MRU=1496, Label Stack{92002 93000}
000C2987637A000C296AFFD581000DFE8847 1676200016B48000
No output feature configured
```

```
CSR8#show ip route 5.5.5.5
Routing entry for 5.5.5.5/32
Known via "bgp 49", distance 200, metric 0, type internal
Last update from 13.13.13.13 01:55:05 ago
Routing Descriptor Blocks:
* 13.13.13.13, from 13.13.13.13, 01:55:05 ago
Route metric is 0, traffic share count is 1
AS Hops 0
MPLS label: 93000
```

```
CSR8#show bgp ipv4 unicast 5.5.5.5/32
BGP routing table entry for 5.5.5.5/32, version 69
Paths: (2 available, best #2, table default)
Advertised to update-groups:
 5          6
Refresh Epoch 1
Local, (Received from a RR-client)
14.14.14.14 (metric 3) from 14.14.14.14 (14.14.14.14)
Origin incomplete, metric 0, localpref 100, valid, internal
Originator: 5.5.5.5, Cluster list: 14.14.14.14
mpls labels in/out 8004/94008
rx pathid: 0, tx pathid: 0
Refresh Epoch 1
Local, (Received from a RR-client)
13.13.13.13 (metric 3) from 13.13.13.13 (13.13.13.13)
Origin incomplete, metric 0, localpref 100, valid, internal, best
Originator: 5.5.5.5, Cluster list: 13.13.13.13
mpls labels in/out 8004/93000
rx pathid: 0, tx pathid: 0x0
```

The push operation occurs because CSR8 has to reach XRv13 across the MPLS core, so it pushes an LDP label from XRv12 for 13.13.13.13/32. The label stack is now {92002 93000 5009}.

```
CSR8#show ip route 13.13.13.13
Routing entry for 13.13.13.13/32
  Known via "ospf 49", distance 110, metric 3, type inter area
  Last update from 49.8.12.12 on GigabitEthernet2.582, 01:53:55 ago
  Routing Descriptor Blocks:
  * 49.8.12.12, from 13.13.13.13, 01:53:55 ago, via GigabitEthernet2.582
    Route metric is 3, traffic share count is 1
```

```
CSR8#show mpls ldp bindings 13.13.13.13 32 neighbor 12.12.12.12
  lib entry: 13.13.13.13/32, rev 90
    remote binding: lsr: 12.12.12.12:0, label: 92002
```

This same in-label was seen earlier in the L3VPN section, but we verify again. The LDP label is popped revealing the BGP label to the ABR, XRv13.

```
RP/0/0/CPU0:XRv12#show mpls forwarding labels 92002
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
92002	Pop	13.13.13.13/32	Gi0/0/0/0.523	49.12.13.13	1032102

XRv13 pops the BGP label since the path to 5.5.5.5/32 is an IGP route that has an implicit-null LDP label. This reveals the PW label of 5009 to CSR5.

```
RP/0/0/CPU0:XRv13#show route ipv4 5.5.5.5/32
Routing entry for 5.5.5.5/32
  Known via "ospf 49", distance 110, metric 2, type intra area
  Routing Descriptor Blocks
  49.5.13.5, from 5.5.5.5, via GigabitEthernet0/0/0/0.553
    Route metric is 2
  No advertising protos.
```

```
RP/0/0/CPU0:XRv13#show mpls ldp bindings 5.5.5.5/32 neighbor 5.5.5.5
5.5.5.5/32, rev 43
  Local binding: label: 93000
  Remote bindings: (2 peers)
    Peer          Label
    -----
    5.5.5.5:0     ImpNull
```

MPLS traceroute and L2VPN detailed outputs confirm the LSP we verified.

```
CSR7#show l2vpn atom vc vcid 567 destination 5.5.5.5 detail | include label_stack
```

```
Output interface: Gi2.578, imposed label stack {8004 5009}
```

```
CSR7#traceroute mpls ipv4 5.5.5.5/32 source 7.7.7.7
```

more work needed here to demux the tfs subtlv and to display the right output [snip]

Type escape sequence to abort.

```
0 49.7.8.7 MRU 1500 [Labels: implicit-null/8004 Exp: 0/0]
L 1 49.7.8.8 MRU 1500 [Labels: 92002/93000 Exp: 0/0] 2 ms
L 2 49.8.12.12 MRU 1500 [Labels: implicit-null/93000 Exp: 0/0] 8 ms
L 3 49.12.13.13 MRU 1500 [Labels: implicit-null Exp: 0] 39 ms
! 4 49.5.13.5 40 ms
```

A quick verification on CSR3 shows that the L2VPN E-LAN is working. There is a full-mesh of EIGRP neighbors for both IPv4 and IPv6, as expected. Since the Q-count is zero, we can assume that multicast and unicast connectivity has been achieved bidirectionally across the L2VPN.

```
CSR3#show eigrp address-family ipv4 vrf U3 neighbors
EIGRP-IPv4 VR(VPLS) Address-Family Neighbors for AS(65000)
VRF(U3)
```

H	Address	Interface	Hold Uptime (sec)	SRTT (ms)	RTO	Q Cnt	Seq Num
1	192.168.0.6	Gi2.307	10 01:25:14	258	1548	0	46
0	192.168.0.11	Gi2.307	14 01:25:24	152	912	0	40

```
CSR3#show eigrp address-family ipv6 vrf U3 neighbors
EIGRP-IPv6 VR(VPLS) Address-Family Neighbors for AS(65000)
VRF(U3)
```

H	Address	Interface	Hold Uptime (sec)	SRTT (ms)	RTO	Q Cnt	Seq Num
1	Link-local address: FE80::11	Gi2.307	10 01:25:19	149	894	0	41
0	Link-local address: FE80::6	Gi2.307	14 01:25:26	25	150	0	43

33.2.2.3 MVPN (mLDP profiles 1 and 17)

The mLDP connectivity works identically regardless of the IGPs in use. We quickly verify the tree construction on CSR2 and CSR5 as examples. Because CSR2 and CSR4 do not import one another's RTs, the MVPN BGP AD type-1 routers are not imported, so they don't join one another's P2MP trees. That is why CSR2 is not joined to a P2MP rooted at CSR4, but CSR5 is. As seen earlier, IPv4 P2MP trees are highlighted in yellow while IPv6 P2MP trees are highlighted in green.

```
CSR2#show mpls mldp database summary
```

LSM ID	Type	Root	Decoded Opaque Value	Client
Cnt.				

11	P2MP	2.2.2.2	[gid 65536 (0x00010000)]	2
21	P2MP	5.5.5.5	[gid 65536 (0x00010000)]	1
22	P2MP	7.7.7.7	[gid 65536 (0x00010000)]	1
12	P2MP	2.2.2.2	[gid 131072 (0x00020000)]	2
1F	P2MP	5.5.5.5	[gid 131072 (0x00020000)]	1
20	P2MP	7.7.7.7	[gid 131072 (0x00020000)]	1
13	MP2MP	8.8.8.8	[mdt 49:222 0]	1

CSR5#show mpls mldp database summary

LSM ID	Type	Root	Decoded Opaque Value	Client
Cnt.				
2C	P2MP	2.2.2.2	[gid 65536 (0x00010000)]	1
27	P2MP	4.4.4.4	[gid 65536 (0x00010000)]	1
13	P2MP	5.5.5.5	[gid 65536 (0x00010000)]	2
28	P2MP	7.7.7.7	[gid 65536 (0x00010000)]	1
2B	P2MP	2.2.2.2	[gid 131072 (0x00020000)]	1
29	P2MP	4.4.4.4	[gid 131072 (0x00020000)]	1
14	P2MP	5.5.5.5	[gid 131072 (0x00020000)]	2
2A	P2MP	7.7.7.7	[gid 131072 (0x00020000)]	1
15	MP2MP	8.8.8.8	[mdt 49:222 0]	1

A quick OAM ping on CSR5 shows that the MP2MP tree is working properly. Traffic can flow bidirectionally which means that the OAM ping can be originated from any PE in the MVPN instance.

```
CSR5#ping mpls mldp mp2mp 8.8.8.8 mdt 49:222 0
mp2mp Root node addr 8.8.8.8
Opaque type MDT, oui:index 0x49:0222, mdtnum 0
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
    timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]
```

```
Request #1
! reply addr 49.4.9.4
! reply addr 49.7.8.7
! reply addr 49.2.10.2
```

```
Round-trip min/avg/max = 172/200/222 ms
Received 3 replies
```

We can also use OAM to check the P2MP trees from the root of that tree. In this case, we test the MVPNv6 tree from CSR5 to see that CSR2, CSR4, and CSR7 join. We also run the same test from CSR4 and only see responses from CSR5 and CSR7. This is expected because CSR2 did not join this P2MP tree.

```
CSR5#ping mpls mldp p2mp 5.5.5.5 hex 0x1 00020000
p2mp Root node addr 5.5.5.5
Opaque type hex value (0x1), num hex digits 4
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
```

```
        timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]
```

```
Request #1
! reply addr 49.2.10.2
! reply addr 49.4.9.4
! reply addr 49.7.8.7
```

```
Round-trip min/avg/max = 20/115/181 ms
Received 3 replies
```

```
CSR4#ping mpls mldp p2mp 4.4.4.4 hex 0x1 00020000
p2mp Root node addr 4.4.4.4
Opaque type hex value (0x1), num hex digits 4
Sending 1, 72-byte MPLS Echos to Target FEC Stack TLV descriptor,
        timeout is 2.2 seconds, send interval is 0 msec, jitter value is 200
msec:
[snip]
```

```
Request #1
! reply addr 49.7.8.7
! reply addr 49.5.14.5
```

```
Round-trip min/avg/max = 91/123/155 ms
Received 2 replies
```

For extra verification, we can see the BGP MVPNV6 routes on CSR4. Only CSR5 and CSR7 routes exist (plus the local one from CSR4), which is additional proof that CSR2 and CSR4 are not exchanging MVPN routes. The details of the CSR7 BGP AD Type-1 (I-PMSI) route are shown below. The format is identical for MVPNV4 and MVPNV6, which is discussed in detail in the MVPN section. This concludes the verification for this section; the OAM checks were sufficient to ensure the MVPN, from an SP perspective, is functional.

```
CSR4#show bgp ipv6 mvpn vrf U1 | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 49:104 (default for vrf U1)
*> [1][49:104][4.4.4.4]/12
      ::                                32768 ?
* i [1][49:104][5.5.5.5]/12
      5.5.5.5                        0    100    0 ?
*>i      5.5.5.5                        0    100    0 ?
*>i [1][49:104][7.7.7.7]/12
      7.7.7.7                        0    100    0 ?

CSR4#show bgp ipv6 mvpn vrf U1 route-type 1 7.7.7.7
BGP routing table entry for [1][49:104][7.7.7.7]/12, version 121
```

```

Paths: (1 available, best #1, table MVPNV6-BGP-Table, not advertised to EBGp
peer)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from [1][49:107][7.7.7.7]/12 (global)
    7.7.7.7 (metric 4) from 7.7.7.7 (7.7.7.7)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Community: no-export
      Extended Community: RT:49:107
      PMSI Attribute: Flags: 0x0, Tunnel type: 2, length 17, label: exp-null,
tunnel parameters: 0600 0104 0707 0707 0007 0100 0400 0200 00
      rx pathid: 0, tx pathid: 0x0

```

33.2.2.4 MPLS TE and TE-FRR

Inter-area TE is similar between IS-IS levels and OSPF areas. The same constraints and techniques apply. We will quickly test both the tunnel stitching and loose-hop expansion methods. First, we have a stitched LSP from CSR2 to CSR7. For variety, we will adjust the tunnel in area 0.0.190.239 to terminate on CSR4; this is one hop away, which makes it a primary one-hop tunnel as well.

```

! CSR2
ip explicit-path name EP_4_DIRECT enable
  next-address 49.2.4.4
  next-address 4.4.4.4

interface Tunnel100
  description PATH TO R4 (DIRECT)
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 4.4.4.4
  tunnel mpls traffic-eng autoroute announce
  tunnel mpls traffic-eng path-option 10 explicit name EP_4_DIRECT

```

Unlike IS-IS, this area boundary with CSR7 actually matters to OSPF. IS-IS considered the inter-area link level-2, so the fact that CSR7 was in a separate area made no difference to IS-IS in that specific design. The tunnel originating from CSR4 cannot terminate on CSR7 for this reason. Instead, we terminate it on CSR8 by looping through other core routers; because this was the cost-based dynamic best-path, we don't make an explicit-path for simplicity. We verify this path with OAM.

```

! CSR4
interface Tunnel100
  description PATH TO CSR8 (DYNAMIC)
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 8.8.8.8
  tunnel mpls traffic-eng autoroute announce
  tunnel mpls traffic-eng path-option 10 dynamic

```

```
CSR4#trace mpls traffic-eng tunnel 100
Tracing MPLS TE Label Switched Path on Tunnel100, timeout is 2 seconds
[snip]
```

Type escape sequence to abort.

```
0 49.4.9.4 MRU 1500 [Labels: 9019 Exp: 0]
L 1 49.4.9.9 MRU 1500 [Labels: 1004 Exp: 0] 2 ms
L 2 49.1.9.1 MRU 1500 [Labels: implicit-null Exp: 0] 9 ms
! 3 49.1.8.8 13 ms
```

Rather than configure a tunnel in area 0.0.202.254, we will merge the TE path with the LDP path. We don't necessarily need to stitch tunnels everywhere, so we can continue to use LDP towards CSR7 from the ABRs servicing that area. First, we have several routing issues to fix. On CSR2, the route to 7.7.7.7 ultimately recurses towards CSR10 outside of the TE tunnel. We can fix this with a static route, or the more elegant approach; adjust BGP.

```
CSR2#show ip route 7.7.7.7
Routing entry for 7.7.7.7/32
  Known via "bgp 49", distance 200, metric 0, type internal
  Last update from 10.10.10.10 00:08:20 ago
  Routing Descriptor Blocks:
    * 10.10.10.10, from 10.10.10.10, 00:08:20 ago
      Route metric is 0, traffic share count is 1
      AS Hops 0
      MPLS label: 10029
```

```
CSR2#show ip route 10.10.10.10
Routing entry for 10.10.10.10/32
  Known via "ospf 49", distance 110, metric 2, type intra area
  Last update from 49.2.10.10 on GigabitEthernet2.520, 00:06:37 ago
  Routing Descriptor Blocks:
    * 49.2.10.10, from 10.10.10.10, 00:06:37 ago, via GigabitEthernet2.520
      Route metric is 2, traffic share count is 1
```

CSR2 has selected CSR10 as the best-path for the route to 7.7.7.7/32 due to it being "closer" to CSR7 than CSR4 is. This generally makes sense and is the reason why BGP considers the IGP metric to the BGP next-hop in the first place.

```
CSR2#show bgp ipv4 unicast 7.7.7.7/32
BGP routing table entry for 7.7.7.7/32, version 111
Paths: (2 available, best #2, table default)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    4.4.4.4 (metric 3) from 4.4.4.4 (4.4.4.4)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Originator: 7.7.7.7, Cluster list: 4.4.4.4, 8.8.8.8
```



```

mpls labels in/out nolabel/4024
rx pathid: 0, tx pathid: 0
Refresh Epoch 1
Local
10.10.10.10 (metric 2) from 10.10.10.10 (10.10.10.10)
Origin incomplete, metric 0, localpref 100, valid, internal, best
Originator: 7.7.7.7, Cluster list: 10.10.10.10, 8.8.8.8
mpls labels in/out nolabel/10029
rx pathid: 0, tx pathid: 0x0

```

We temporarily increase CSR10's cost on its loopback to 2. This means that CSR4 will become the winner based on having the lower neighbor ID. If we used a number greater than 2, CSR4 would still win, but as a result of having a lower IGP metric to the BGP next-hop. I form an IGP metric tie in this case just for fun. Label 4024 is now used within the LSP. Because CSR4 is the tail end of the tunnel, the next label in the stack should be a local label of that tail-end router. Using CSR10's local label would break the LSP as CSR4 would not know how to label-switch those 10000-series labels.

```

! CSR10
interface Loopback0
ip ospf cost 2

CSR2#show bgp ipv4 unicast 7.7.7.7/32
BGP routing table entry for 7.7.7.7/32, version 116
Paths: (2 available, best #1, table default)
Not advertised to any peer
Refresh Epoch 1
Local
4.4.4.4 (metric 3) from 4.4.4.4 (4.4.4.4)
Origin incomplete, metric 0, localpref 100, valid, internal, best
Originator: 7.7.7.7, Cluster list: 4.4.4.4, 8.8.8.8
mpls labels in/out nolabel/4024
rx pathid: 0, tx pathid: 0x0
Refresh Epoch 2
Local
10.10.10.10 (metric 3) from 10.10.10.10 (10.10.10.10)
Origin incomplete, metric 0, localpref 100, valid, internal
Originator: 7.7.7.7, Cluster list: 10.10.10.10, 8.8.8.8
mpls labels in/out nolabel/10029
rx pathid: 0, tx pathid: 0

```

To verify that this works, we follow the route lookups on CSR2. Now that 4.4.4.4 is the BGP next-hop for 7.7.7.7, the traffic is routed into the tunnel via auto-route. Ignoring MPLS service labels (L3VPN, L2VPN, etc), the label stack is now {4024}. Also, ignoring mLDP requirements, LDP is not required on this tunnel despite it not being end-to-end because BGP can be used for the intermediate label bindings normally required for PE-P or P-P tunnels.

```

CSR2#show ip route 7.7.7.7

```

Routing entry for 7.7.7.7/32

Known via "bgp 49", distance 200, metric 0, type internal

Last update from 4.4.4.4 00:00:14 ago

Routing Descriptor Blocks:

* 4.4.4.4, from 4.4.4.4, 00:00:14 ago

Route metric is 0, traffic share count is 1

AS Hops 0

MPLS label: 4024

CSR2#show ip route 4.4.4.4

Routing entry for 4.4.4.4/32

Known via "ospf 49", distance 110, metric 3, type intra area

Last update from 4.4.4.4 on Tunnel100, 00:11:21 ago

Routing Descriptor Blocks:

* 4.4.4.4, from 4.4.4.4, 00:11:21 ago, via Tunnel100

Route metric is 3, traffic share count is 1

CSR2#show mpls traffic-eng tunnels tunnel 100 | include Label

InLabel : -

OutLabel : GigabitEthernet2.524, implicit-null

The same logic is true on CSR4. Checking the route recursion, 7.7.7.7 is reachable via 8.8.8.8, which is reachable via the TE tunnel. The label stack becomes {9019 8002}. In short, creating P-P to PE-P tunnels that terminate on routers that swap BGP labels (UMPLS ABRs, for example) is an acceptable alternative to using tLDP.

CSR4#show ip route 7.7.7.7

Routing entry for 7.7.7.7/32

Known via "bgp 49", distance 200, metric 0, type internal

Last update from 8.8.8.8 3d07h ago

Routing Descriptor Blocks:

* 8.8.8.8, from 8.8.8.8, 3d07h ago

Route metric is 0, traffic share count is 1

AS Hops 0

MPLS label: 8002

CSR4#show ip route 8.8.8.8

Routing entry for 8.8.8.8/32

Known via "ospf 49", distance 110, metric 4, type intra area

Last update from 8.8.8.8 on Tunnel100, 00:09:53 ago

Routing Descriptor Blocks:

* 8.8.8.8, from 8.8.8.8, 00:09:53 ago, via Tunnel100

Route metric is 4, traffic share count is 1

CSR4#show mpls traffic-eng tunnels tunnel 100 | include Label

InLabel : -

OutLabel : GigabitEthernet2.549, 9019

BGP details this operation as well. When label 4024 arrives, it is swapped to another BGP label of 8002, which is CSR8's local label for 7.7.7.7/32. Then, as seen above, traffic to 8.8.8.8 is tunneled inside MPLS using RSVP-bound labels. The LFIB details show these swap and push operations, which occur in that specific sequence. As expected, BGP doesn't have the "full picture" of other things happening beyond its purview, but the LFIB does.

```
CSR4#show bgp ipv4 unicast 7.7.7.7/32 bestpath
BGP routing table entry for 7.7.7.7/32, version 97
Paths: (2 available, best #2, table default)
  Advertised to update-groups:
    3
  Refresh Epoch 4
  Local
    8.8.8.8 (metric 4) from 8.8.8.8 (8.8.8.8)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Originator: 7.7.7.7, Cluster list: 8.8.8.8
      mpls labels in/out 4024/8002
      rx pathid: 0, tx pathid: 0x0

CSR4#show mpls forwarding-table labels 4024 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
4024       8002      7.7.7.7/32      41231        Tu100     point2point
MAC/Encaps=18/26, MRU=1496, Label Stack{9019 8002}, via Gi2.549
000C29E04F84000C29D8E5F581000DDD8847 0233B00001F42000
No output feature configured
```

CSR9 and CSR1 are just P routers and perform ordinary label swap and PHP operations, respectively. The label stack becomes {1004 8002} and {8002} as it transits the core.

```
CSR9#show mpls forwarding-table labels 9019
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
9019       1004      4.4.4.4 100 [1] 97530        Gi2.519    49.1.9.1

CSR1#show mpls forwarding-table labels 1004
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
1004       Pop Label  4.4.4.4 100 [1] 95886        Gi2.518    49.1.8.8
```

CSR8 receives packets with only label 8002. Because CSR8's next-hop is CSR7, which was the original next-hop, the BGP label is removed which reveals the service label, if it exists. Beware of the BGP output; the BGP route is not installed in the RIB, which means CSR8 is learning the route another way. Specifically, it is an OSPF route and has an LDP binding for implicit-null. It just happens to be the same label binding as the BGP route, but technically, the implicit-nulls are different and mean different things about the forwarding path.

```

CSR8#show bgp ipv4 unicast 7.7.7.7/32
BGP routing table entry for 7.7.7.7/32, version 75
Paths: (1 available, best #1, table default, RIB-failure(17))
  Advertised to update-groups:
    6
  Refresh Epoch 7
  Local, (Received from a RR-client)
    7.7.7.7 (metric 2) from 7.7.7.7 (7.7.7.7)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      mpls labels in/out 8002/imp-null
      rx pathid: 0, tx pathid: 0x0

```

```

CSR8#show ip route 7.7.7.7
Routing entry for 7.7.7.7/32
  Known via "ospf 49", distance 110, metric 2, type intra area
  Last update from 49.7.8.7 on GigabitEthernet2.578, 3d08h ago
  Routing Descriptor Blocks:
    * 49.7.8.7, from 7.7.7.7, 3d08h ago, via GigabitEthernet2.578
      Route metric is 2, traffic share count is 1

```

```

CSR8#show mpls ldp bindings 7.7.7.7 32 neighbor 7.7.7.7
lib entry: 7.7.7.7/32, rev 22
  remote binding: lsr: 7.7.7.7:0, label: imp-null

```

We can verify the label stack using OAM, ignoring the implicit-nulls that don't belong.

```

CSR2#traceroute mpls ipv4 7.7.7.7/32 source 2.2.2.2
more work needed here to demux the tfs subtlv and to display the right output
[snip]

```

```

Type escape sequence to abort.
  0 49.2.4.2 MRU 1500 [Labels: implicit-null/implicit-null/4024 Exp: 0/0/0]
L 1 49.2.4.4 MRU 1500 [Labels: 9019/8002 Exp: 0/0] 2 ms
L 2 49.4.9.9 MRU 1500 [Labels: 1004/8002 Exp: 0/0] 9 ms
L 3 49.1.9.1 MRU 1500 [Labels: 8002 Exp: 0] 13 ms
L 4 49.1.8.8 MRU 1500 [Labels: implicit-null Exp: 0] 35 ms
! 5 49.7.8.7 21 ms

```

A quick MPLS service check for L3VPN indicates that the tunnel works. The transport labels from LDP, RSVP, and BGP are all the same, with the additional VPN label at the bottom. We confirm this value is 7007 by checking BGP VPNv4, then use traceroute to verify the rest of the label stack. The key point is that targeted LDP was not enabled on either tunnel, yet they were successfully stitched together. Although this tunneling design causes issues with MVPN as described in other sections section, it does work well with UMPLS.

```

CSR2#show bgp vpnv4 unicast vrf U1 1.3.3.3/32

```

```

BGP routing table entry for 49:102:1.3.3.3/32, version 128
Paths: (1 available, best #1, table U1)
  Advertised to update-groups:
    1
  Refresh Epoch 2
  65000, imported path from 49:107:1.3.3.3/32 (global)
    7.7.7.7 (metric 3) (via default) from 7.7.7.7 (7.7.7.7)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:49:107
      mpls labels in/out nolaabel/7007
      rx pathid: 0, tx pathid: 0x0

```

```

CSR6#traceroute vrf U1 1.3.3.3 source 1.6.6.6
Type escape sequence to abort.
Tracing the route to 1.3.3.3
VRF info: (vrf in name/id, vrf out name/id)
 1 192.168.102.2 1 msec 1 msec 1 msec
 2 49.2.4.4 [MPLS: Labels 4024/7007 Exp 0] 2 msec 2 msec 10 msec
 3 49.4.9.9 [MPLS: Labels 9019/8002/7007 Exp 0] 32 msec 32 msec 32 msec
 4 49.1.9.1 [MPLS: Labels 1004/8002/7007 Exp 0] 32 msec 32 msec 32 msec
 5 49.1.8.8 [MPLS: Labels 8002/7007 Exp 0] 32 msec 31 msec 37 msec
 6 192.168.107.7 [MPLS: Label 7007 Exp 0] 26 msec 27 msec 20 msec
 7 192.168.107.3 20 msec * 3 msec

```

The loose-expansion tunnel works exactly the same in OSPF as it does for IS-IS. For this reason, we don't have to change the tunnel configuration on CSR5 at all for tunnel101. The configuration is shown again for reference but remains unchanged.

```

! CSR5
ip explicit-path name EP_14_10_2_LOOSE enable
  next-address loose 14.14.14.14
  next-address loose 10.10.10.10
  next-address loose 2.2.2.2

interface Tunnel101
  description PATH TO R2 VIA XRV14 AND R10
  ip unnumbered Loopback0
  tunnel mode mpls traffic-eng
  tunnel destination 2.2.2.2
  tunnel mpls traffic-eng autoroute destination
  tunnel mpls traffic-eng priority 7 7
  tunnel mpls traffic-eng bandwidth 5000
  tunnel mpls traffic-eng path-option 10 explicit name EP_14_10_2_LOOSE
  tunnel mpls traffic-eng fast-reroute

```

First, CSR5 computes a strict path to the local ABR which is XRV14. This uses the directly connected interface between CSR5 and XRV14.

```
CSR5#show ip rsvp sender detail filter session-type 7 destination 2.2.2.2 |
section ERO
```

```
ERO: (incoming)
  5.5.5.5 (Strict IPv4 Prefix, 8 bytes, /32)
  49.5.14.14 (Strict IPv4 Prefix, 8 bytes, /32)
  14.14.14.14 (Strict IPv4 Prefix, 8 bytes, /32)
  10.10.10.10 (Loose IPv4 Prefix, 8 bytes, /32)
  2.2.2.2 (Loose IPv4 Prefix, 8 bytes, /32)
ERO: (outgoing)
  49.5.14.14 (Strict IPv4 Prefix, 8 bytes, /32)
  14.14.14.14 (Strict IPv4 Prefix, 8 bytes, /32)
  10.10.10.10 (Loose IPv4 Prefix, 8 bytes, /32)
  2.2.2.2 (Loose IPv4 Prefix, 8 bytes, /32)
```

When XRv14 receives the PATH message, it runs the PCALC expansion process to compute a strict path to the first loose hop, which is CSR10. This path routes via CSR1 and CSR8.

```
RP/0/0/CPU0:XRv14#show rsvp sender session-type lsp-p2p destination 2.2.2.2
detail | begin Explicit
```

```
Explicit Route (Incoming):
  Strict, 49.5.14.14/32
  Strict, 14.14.14.14/32
  Loose, 10.10.10.10/32
  Loose, 2.2.2.2/32
Explicit Route (Outgoing):
  Strict, 49.1.14.1/32
  Strict, 49.1.8.8/32
  Strict, 49.8.10.10/32
  Strict, 10.10.10.10/32
  Loose, 2.2.2.2/32
```

When CSR10 receives, the PATH message, it also runs the PCALC expand algorithm to evaluate a strict path to the first loose hop, which is CSR2.

```
CSR10#show ip rsvp sender detail filter session-type 7 sender 5.5.5.5 |
section ERO
```

```
ERO: (incoming)
  49.8.10.10 (Strict IPv4 Prefix, 8 bytes, /32)
  10.10.10.10 (Strict IPv4 Prefix, 8 bytes, /32)
  2.2.2.2 (Loose IPv4 Prefix, 8 bytes, /32)
ERO: (outgoing)
  49.2.10.2 (Strict IPv4 Prefix, 8 bytes, /32)
  2.2.2.2 (Strict IPv4 Prefix, 8 bytes, /32)
```

Since each router in the path appended its local interface to the RSVP PATH RRO, CSR2 can see the full path back to source, to include inter-area hops. The RRO also shows available TE-FRR protection along the way, which is something normally the tail-end does not see. These TE-FRR backup tunnels were

configured in the previous section and remain intact here. No labels are included in the PATH RRO since there is no MPLS forwarding along this path (it is not an LSP), which is different than the RESV RRO also shown below. This is the RRO we are used to seeing, which includes the per-hop RSVP-allocated labels along with any protection capabilities along the LSP. The labels, in sequence, are 94018 > 1032 > 8006 > 10021 > implicit-null.

```
CSR2#show ip rsvp sender detail filter session-type 7 sender 5.5.5.5 |
section RRO
```

RRO:

```
49.2.10.10/32, Flags:0x0 (No Local Protection)
49.8.10.8/32, Flags:0xD (Local Prot Avail/Has BW/to NNHOP)
49.1.8.1/32, Flags:0x5 (Local Prot Avail/Has BW/to NHOP)
49.1.14.14/32, Flags:0x0 (No Local Protection)
49.5.14.5/32, Flags:0x0 (No Local Protection)
```

```
CSR5#show ip rsvp reservation detail filter session-type 7 destination
2.2.2.2 | section RRO
```

RRO:

```
14.14.14.14/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 94018
49.5.14.14/32, Flags:0x0 (No Local Protection)
  Label subobject: Flags 0x1, C-Type 1, Label 94018
1.1.1.1/32, Flags:0x25 (Local Prot Avail/Has BW/to NHOP, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 1032
49.1.8.1/32, Flags:0x5 (Local Prot Avail/Has BW/to NHOP)
  Label subobject: Flags 0x1, C-Type 1, Label 1032
8.8.8.8/32, Flags:0x2D (Local Prot Avail/Has BW/to NNHOP, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 8006
49.8.10.8/32, Flags:0xD (Local Prot Avail/Has BW/to NNHOP)
  Label subobject: Flags 0x1, C-Type 1, Label 8006
10.10.10.10/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 10021
49.2.10.10/32, Flags:0x0 (No Local Protection)
  Label subobject: Flags 0x1, C-Type 1, Label 10021
2.2.2.2/32, Flags:0x20 (No Local Protection, Node-id)
  Label subobject: Flags 0x1, C-Type 1, Label 3
49.2.10.2/32, Flags:0x0 (No Local Protection)
  Label subobject: Flags 0x1, C-Type 1, Label 3
```

We quickly verify the LSP is operational using OAM from CSR5 and verify the labels at the same time. Notice that this technique, unlike tunnel stitching, does not rely on an additional (third) BGP label in the stack to achieve scalability. An end-to-end TE design like this defeats the purpose of UMPLS because it scales poorly, but is a valid technique.

```
CSR5#traceroute mpls traffic-eng tunnel 101
Tracing MPLS TE Label Switched Path on Tunnel101, timeout is 2 seconds
[snip]
```

Type escape sequence to abort.

```
0 49.5.14.5 MRU 1500 [Labels: 94018 Exp: 0]
L 1 49.5.14.14 MRU 1500 [Labels: 1032 Exp: 0] 2 ms
L 2 49.1.14.1 MRU 1500 [Labels: 8006 Exp: 0] 56 ms
L 3 49.1.8.8 MRU 1500 [Labels: 10021 Exp: 0] 26 ms
L 4 49.8.10.10 MRU 1500 [Labels: implicit-null Exp: 0] 24 ms
! 5 49.2.10.2 40 ms
```

We also use traceroute within the VPN on XRv11 to test VPNv6 reachability to CSR6. CSR5 selects the path via CSR2 as best due to having a lower originator ID, so the VPN label of 2002 is used. The top-most transport labels are the same RSVP-allocated labels we verified with OAM.

```
CSR5#show bgp vpnv6 unicast vrf U2 ::2:6:6:6/128
BGP routing table entry for [49:205]::2:6:6:6/128, version 470
Paths: (2 available, best #1, table U2)
  Advertised to update-groups:
    1
  Refresh Epoch 2
  65000, imported path from [49:202]::2:6:6:6/128 (global)
    ::FFFF:2.2.2.2 (via default) from 7.7.7.7 (7.7.7.7)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:49:202
      Originator: 2.2.2.2, Cluster list: 7.7.7.7
      mpls labels in/out nlabel/2004
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 2
  65000, imported path from [49:204]::2:6:6:6/128 (global)
    ::FFFF:4.4.4.4 (metric 2) (via default) from 7.7.7.7 (7.7.7.7)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Extended Community: RT:49:204
      Originator: 4.4.4.4, Cluster list: 7.7.7.7
      mpls labels in/out nlabel/4006
      rx pathid: 0, tx pathid: 0
```

```
RP/0/0/CPU0:XRv11#traceroute vrf U2 ::2:6:6:6 source ::2:11:11:11
```

Type escape sequence to abort.

Tracing the route to ::2:6:6:6

```
1  fd00:192:168:205::5 19 msec 9 msec 0 msec
2  2049:49:5:14::14 [MPLS: Labels 94018/2004 Exp 0] 59 msec 39 msec 39 msec
3  ::ffff:49.1.14.1 [MPLS: Labels 1032/2004 Exp 0] 39 msec 39 msec 39 msec
4  ::ffff:49.1.8.8 [MPLS: Labels 8006/2004 Exp 0] 39 msec 39 msec 39 msec
5  ::ffff:49.8.10.10 [MPLS: Labels 10021/2004 Exp 0] 39 msec 39 msec 39 msec
6  fd00:192:168:202::2 [MPLS: Label 2004 Exp 0] 39 msec 39 msec 39 msec
7  fd00:192:168:202::6 49 msec 59 msec 49 msec
```


We saw earlier that this inter-area tunnel was protected by FRR as evidenced by the PATH and RESV RROs. CSR1 offers NHOP protected and is protecting the LSP with in-label 1032. It provides 7 Mbps bandwidth backup for any pool.

```
CSR1#show mpls traffic-eng fast-reroute database labels 1032 | begin P2P LSP
P2P LSP midpoint frr information:
LSP identifier          In-label Out intf/label  FRR intf/label  Status
-----
5.5.5.5 101 [81]       1032     Gi2.518:8006      Tu110:8006      ready
```

```
CSR1#show mpls traffic-eng tunnels tunnel 110 backup
INTER-AREA NHOP PROTECTION
  LSP Head, Admin: up, Oper: up
  Tun ID: 110, LSP ID: 66, Source: 1.1.1.1
  Destination: 8.8.8.8
  Fast Reroute Backup Provided:
    Protected i/fs: Gi2.518
    Protected LSPs/Sub-LSPs: 1, Active: 0
    Backup BW: any pool; limit: 7000 kbps, inuse: 5000 kbps (BWP inuse: 0
kbps)
    Backup flags: 0x0
```

Additionally, CSR8 provides NNHOP protection to completely avoid CSR10. This is also an inter-area tunnel using loose-hop expansion, which is valid and was verified in detail earlier. The protected LSP has a local label of 8020 and since the NNHOP is the tunnel destination, the FRR label becomes implicit-null since there isn't a need to instruct the merge point to forward the packet further (MP is the tail end). We can also see the backup bandwidth which is available for the protected LSP.

```
CSR8#show mpls traffic-eng fast-reroute database labels 8020 | begin P2P LSP
P2P LSP midpoint frr information:
LSP identifier          In-label Out intf/label  FRR intf/label  Status
-----
5.5.5.5 101 [83]       8020     Gi2.580:10012    Tu111:implicit-n ready
```

```
CSR8#show mpls traffic-eng tunnels tunnel 111 backup
INTER_AREA NNHOP PROTECTION
  LSP Head, Admin: up, Oper: up
  Tun ID: 111, LSP ID: 195, Source: 8.8.8.8
  Destination: 2.2.2.2
  Fast Reroute Backup Provided:
    Protected i/fs: Gi2.580
    Protected LSPs/Sub-LSPs: 1, Active: 0
    Backup BW: any pool; limit: 9000 kbps, inuse: 5000 kbps (BWP inuse: 0
kbps)
    Backup flags: 0x0
```

Additional Reading – Reference configurations "umpls-ospf"

34. Describe, implement, and troubleshoot LISP

Locator/ID Separation Protocol (LISP) is less of a routing protocol and more of a dynamic tunneling protocol, somewhat similar to MPLS and mGRE encapsulations. It uses UDP encapsulation on port 4341 for data traffic and UDP port 4342 for LISP control traffic. It was developed by Cisco and is currently an experimental RFC within the IETF (RFC 6830). The basic premise of LISP is, as the name states, to separate WHERE a host is from WHO it is. IP routing relies on a host being part of a network (subnet) which is both its location on the network and its identity concurrently. LISP does not re-invent IP routing nor does it advertise IP routes. It uses routing locations (RLOC) to specify WHERE a host is and endpoint-IDs (EID) to specify WHO the host is. This is basic tunneling; the RLOCs are the outer encapsulation in the tunnel and are expected to be reachable via some existing IP routing protocol, just like normal IP tunneling mechanisms. The EIDs don't have to be routed between sites with a dedicated routing protocols provided they are registered to a central location where LISP speakers can determine where the EIDs are. This EID resolution is similar to DNS where a set of central devices can map EIDs to RLOCs.

LISP decouples the roles of resolving EID-to-RLOC mappings and mapping registration into two components:

MS: The central mechanism used for RLOC/EID registration is known as a map server (MS). ETRs (discussed later) register their EID-to-RLOC mappings with the MS. This process is loosely analogous to the NHRP registration process of mapping overlay (VPN tunnel) addresses to underlay (NBMA) addresses. Map-requests are forwarded from the MR (discussed later) to the MS, who will perform an EID-database lookup. Assuming there is a match, it will forward the map-request directly to the ETR who registered the EID; the ETR is responsible for sending the map-reply to the original ITR.

MR: The map resolver (MR) is the target of all map-request messages. When an ITR (discussed later) sends a map-request to the MR, the request is forwarded to the MS over the ALT (discussed later). The MR does very little work in that its entire purpose is to receive map-requests and forward them to the MS, who actually has the mappings. In order to actually forward map-requests to the MS, the MR needs to have BGP routes inside the ALT for the EID destinations; this is how the MR knows which MS has the correct mappings if multiple MS devices exist.

ALT: Not a component of LISP so much as it is a virtual network used for inter MR/MS communication. It relies on VRFs, GRE, and BGP so that the MR and MS can exist on separate devices. Although uncommon, the ALT would be required when the MR and MS are separated, which might be good for availability/scalability. **The MR and MS roles are commonly combined on a single router for simplicity, known as the MR/MS.**

Two types of LISP tunneling routers are used at the customer sites for encapsulation and decapsulation:

ITR: An ingress tunnel router (ITR) is used for communicating with the MR to finding EID-to-RLOC mappings based on traffic being sent towards the provider (that is to say, traffic outbound from the

customer). The ITR role is typically assigned to a CE and runs on a Cisco ISR or ASR, for example. When a packet arrives on the LAN side on an ITR (inside EID space), the ITR sends a map-request to the MR. As discussed earlier, the MR forwards the map-request to the MS. If the MS has a mapping for this EID, the map-request is forwarded to the ETR that registered the EID. If not, the MS sends a map-reply that indicates that the destination of the original packet is not an EID (no mapping). The ITR then creates a negative cache entry to blackhole traffic.

ETR: An egress tunnel router (ETR) communicates with the MS to register EID-to-RLOC mappings. These mappings are sent in map-register messages. When LISP-encapsulated packets arrive on an ETR destined to one of its RLOCs (incoming traffic from the provider), verifies that the inner-header is destined to one of its registered EIDs, and forwards the packet natively on the LAN-side (inside EID space). Upon receiving a forwarded map-request from the MS, it verifies that it has the EID being requested (it should, otherwise the MS wouldn't have sent it, but this is a sanity check). It replies directly to the ITR that originated the map-request using the ITR RLOCs listed in the message.

The ITR and ETR roles are commonly combined on a single router for simplicity, known as the **xTR**.

To enable communication with non-LISP domains, two additional LISP router variants are introduced:

PITR: The Proxy ITR performs ITR functions on behalf of non-LISP sites. These are typically deployed at Internet exchange points to provide interworking from non-LISP to LISP sites. Non-LISP sites will route traffic towards the PITR for encapsulation and forwarding to LISP sites. For scalability, aggregation should be used so that the PITR does not have to advertise many specific routes. No additional configuration is needed on the ITRs when the PITR is deployed.

PETR: The Proxy ETR performs ETR functions on behalf of LISP sites. The PETR is deployed similarly to the PITR, except is used to provide interworking from LISP to non-LISP. This would normally be used when the SP will not allow private (EID) space to be routed over its core. ETRs must specify the PETR address to use the PETR service.

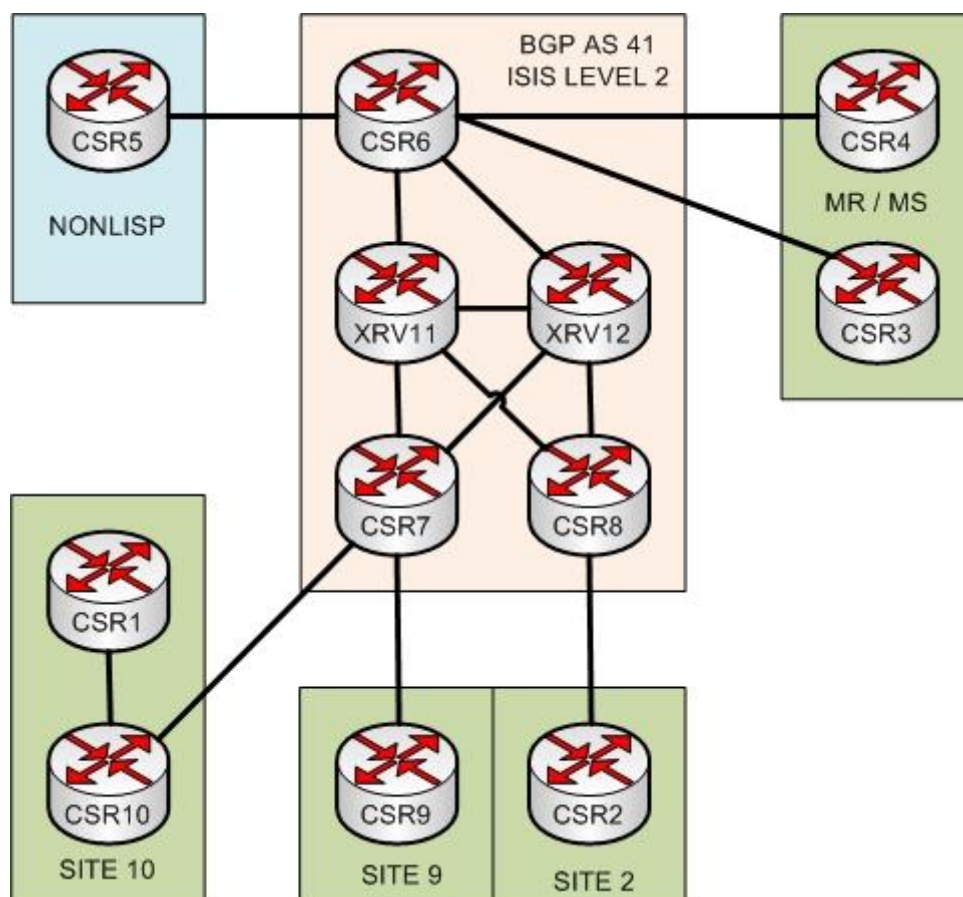
The PITR and PETR roles are commonly combined on a single router for simplicity, known as the **PxTR**.

LISP supports two main variations of virtualization:

Shared Model: EID spaces exist in varying VRFs with a single upstream RLOC table. For example, CSR9 has its RLOC in global space while its EIDs are in varying VRFs. Each EID VRF has its own instance ID (IID) specified on the ETR and MS so that the EID-to-RLOC mappings are kept separate. Just like BGP VPNv4/v6, overlap can exist in different VRFs, so this IID has a similar function to the BGP RD. CSR10 also uses this model except the RLOC is in a VRF, much like a front-door VRF (FVRF) model commonly used in overlay networking.

Parallel Model: EID spaces are in N VRFs with N upstream RLOCs which correspond to each different VRF. This is a more traditional model of splitting the router into VRFs and not having any inter-VRF communication between EID and RLOC namespaces. This generally means a different layer 3 interface per EID VRF towards the provider. This is implemented on CSR2, where there are two subinterfaces in two different VRFs to service two different EID spaces. This also requires two separate LISP processes.

The network diagram below shows an SP core inside AS 41 that provides IPv4/v6 transport between the PE-CE connected routes only. There is no PE-CE protocol exchange anywhere. There are three customer sites that communicate using LISP (CSR10, CSR9, and CSR2). These sites use a variety of LISP virtualization models discussed later; many of the test endpoints are loopbacks on these routers for simplicity. CSR3 is an MR/MS and CSR4 is an MR only; the two communicate over an ALT. CSR5 is a public Internet site which uses the PxTR configured on CSR6 to communicate to and from the LISP customer sites. To test multicast, CSR1 is an ordinary end host and multicast source. CSR1 is also used to test mobility by moving between LANs hosted on CSR10 (home) and CSR9 (foreign); this particular feature is deemed out of scope for CCIE SP and is not tested here.



The relevant configurations for CSR9 and CSR10 are shown below. We will begin with the basic components to understand the control-plane mechanisms. CSR9 has both RLOC and EIDs in the global table (no virtualization yet) while CSR10 has the RLOC in a front-door VRF (shared virtualization model).

Each of them register their loopbacks with the MS, which is 10.3.6.3 (CSR3). The MR is 10.4.6.4 (CSR4) which is separated from the MS.

```
! CSR9
interface GigabitEthernet2.579
 ip address 10.7.9.9 255.255.255.0

router lisp 1
 eid-table default instance-id 0
  database-mapping 10.9.9.9/32 IPv4-interface GigabitEthernet2.579 priority
10 weight 10
  ipv4 itr map-resolver 10.4.6.4
  ipv4 etr map-server 10.3.6.3 key CSR9

ip route 0.0.0.0 0.0.0.0 10.7.9.7

! CSR10
interface GigabitEthernet2.570
 vrf forwarding UP
 ip address 10.7.10.10 255.255.255.0

router lisp 1
 locator-table vrf UP
 eid-table default instance-id 0
  database-mapping 10.10.10.10/32 IPv4-interface GigabitEthernet2.570
priority 10 weight 10
  ipv4 itr map-resolver 10.4.6.4
  ipv4 etr map-server 10.3.6.3 key CSR10

ip route vrf UP 0.0.0.0 0.0.0.0 10.7.10.7
```

Before any traffic can flow, the ETRs must register their EID-to-RLOC mappings with the MS (CSR3). Both CSR10 and CSR9 must do this; notice that the map-register message also contains the IID so the MS knows in which table to store the mappings. This IID serves as a method to distinguished EID routes between customers, much like the BGP RD does for MPLS VPNs. IID 0 represents the global table, which is how we can sort through the debugs to eliminate other duplicate prefixes of 10.10.10.10/32 and 10.9.9.9/32 in other VRFs. LISP debugs are some of the best debugs in IOS; very detailed with many granular options. Notice that the map-register messages use UDP port 4342 as this is used only for LISP signaling. This registration occurs once every 60 seconds.

```
! CSR9
CSR9#debug lisp control-plane etr-map-server

LISP-1: IPv4 Map Server IID 0 10.3.6.3, Sending map-register (src_rloc
10.7.9.9) nonce 0x960C2F3D-0xB25541FE.

! CSR3
```

```
CSR3#debug lisp control-plane map-server-registration
```

```
LISP: Processing received Map-Register(3) message on GigabitEthernet2.536
from 10.7.10.10:4342 to 10.3.6.3:4342
LISP: Processing Map-Register no proxy, map-notify, no merge, no security, no
mobile-node, not to-RTR, no fast-map-register, no EID-notify, ID-included, 1
record, nonce 0xE9B119D3-0xA5B3EAFD, key-id 1, auth-data-len 20, hash-
function sha1, xTR-ID 0x86671414-0xB88BC82F-0x428A5BA4-0xB567A9B7, site-ID
unspecified
LISP: Processing Map-Register mapping record for IID 0 10.10.10.10/32, ttl
1440, action none, authoritative, 1 locator
      10.7.10.10 pri/wei=10/10 LpR
LISP-0: MS registration IID 0 prefix 10.10.10.10/32 10.7.10.10 site CSR10,
Updating.
```

```
LISP: Processing received Map-Register(3) message on GigabitEthernet2.536
from 10.7.9.9:4342 to 10.3.6.3:4342
LISP: Processing Map-Register no proxy, map-notify, no merge, no security, no
mobile-node, not to-RTR, no fast-map-register, no EID-notify, ID-included, 1
record, nonce 0x960C2F3D-0xB25541FE, key-id 1, auth-data-len 20, hash-
function sha1, xTR-ID 0xDF137D40-0xE6306DF2-0xB055C530-0xD3D28EB2, site-ID
unspecified
LISP: Processing Map-Register mapping record for IID 0 10.9.9.9/32, ttl 1440,
action none, authoritative, 1 locator
      10.7.9.9 pri/wei=10/10 LpR
LISP-0: MS registration IID 0 prefix 10.9.9.9/32 10.7.9.9 site CSR9,
Updating.
```

A summary view of this IID 0 table shows the following registered prefixes with the MS. Looking into the details of 10.9.9.9/32, most of this information is the same as what we saw in the debug message. This is useful for troubleshooting more advanced LISP functionality like PxTR, load sharing, and dynamic EID creation. The priority/weight values are discussed later.

```
CSR3#show lisp site instance-id 0
LISP Site Registration Information
* = Some locators are down or unreachable
```

Site Name	Last Register	Up	Who Last Registered	Inst ID	EID Prefix
CSR10	00:00:11	yes	10.7.10.10	0	10.10.10.10/32
CSR9	00:00:01	yes	10.7.9.9	0	10.9.9.9/32

```
CSR3#show lisp site 10.9.9.9/32 instance-id 0
LISP Site Registration Information
Site name: CSR9
Allowed configured locators: any
Requested EID-prefix:
      EID-prefix: 10.9.9.9/32 instance-id 0
```

```

First registered:      3d00h
Routing table tag:    0
Origin:               Configuration
Merge active:         No
Proxy reply:          No
TTL:                  1d00h
State:                complete
Registration errors:
  Authentication failures: 0
  Allowed locators mismatch: 0
ETR 10.7.9.9, last registered 00:00:12, no proxy-reply, map-notify
  TTL 1d00h, no merge, hash-function sha1, nonce 0x960C2F3D-
0xB25541FE
  state complete, no security-capability
  xTR-ID 0xDF137D40-0xE6306DF2-0xB055C530-0xD3D28EB2
  site-ID unspecified
Locator  Local  State      Pri/Wgt  Scope
10.7.9.9  yes    up        10/10   IPv4 none

```

Because CSR3 is not an MR for IID 0, it needs to somehow receive map-requests from CSR4, which is the MR for IID 0. This is doing using the ALT; there is a P2P GRE tunnel between CSR3 and CSR4 inside of a VRF called LISPALT. BGP is running inside this VRF to connect the two sites using the IPv4 AF within the VRF. The significant configuration snippets are shown below and are explained next.

```

! CSR3
interface Tunnel34
  vrf forwarding LISPALT
  ip address 10.3.4.3 255.255.255.0
  ipv6 address FE80::3 link-local
  ipv6 address 2001:10:3:4::3/64
  tunnel source 10.3.6.3
  tunnel destination 10.4.6.4

router lisp
  eid-table default instance-id 0
  ipv4 alt-vrf LISPALT
  ipv6 alt-vrf LISPALT

site CSR10
  authentication-key CSR10
  eid-prefix 10.10.10.10/32

site CSR9
  authentication-key CSR9
  eid-prefix 10.9.9.9/32

router bgp 34

```

```

address-family ipv4 vrf LISPALT
  redistribute lisp
  neighbor 10.3.4.4 remote-as 34
  neighbor 10.3.4.4 activate

address-family ipv6 vrf LISPALT
  redistribute lisp
  neighbor 2001:10:3:4::4 remote-as 34
  neighbor 2001:10:3:4::4 activate

! CSR4
interface Tunnel34
  vrf forwarding LISPALT
  ip address 10.3.4.4 255.255.255.0
  ipv6 address FE80::4 link-local
  ipv6 address 2001:10:3:4::4/64
  tunnel source 10.4.6.4
  tunnel destination 10.3.6.3

router lisp
  ipv4 map-resolver
  ipv4 alt-vrf LISPALT

router bgp 34
  address-family ipv4 vrf LISPALT
    neighbor 10.3.4.3 remote-as 34
    neighbor 10.3.4.3 activate

  address-family ipv6 vrf LISPALT
    neighbor 2001:10:3:4::3 remote-as 34
    neighbor 2001:10:3:4::3 activate

```

When CSR3 receives map-registrations, it originates an exact-match route into the ALT for advertisement to the MRs. CSR3 has originated routes for both 10.9.9.9/32 and 10.10.10.10/32 as LISP routes with AD 10 pointing to Null0. In this way, they are present in the RIB for the ALT VRF. This allows them to be redistributed into BGP for advertisement over the ALT. The routes don't direct traffic but are important to facilitate communication between disparate MR and MS devices.

```
CSR3#show vrf LISPALT
```

Name	Default RD	Protocols	Interfaces
LISPALT	34:3	ipv4, ipv6	Tu34

```
CSR3#show ip route vrf LISPALT lisp | begin Gateway
```

```
Gateway of last resort is not set
```

```

      10.0.0.0/8 is variably subnetted, 4 subnets, 2 masks
1       10.9.9.9/32 [10/1], 3d00h, Null0
1       10.10.10.10/32 [10/1], 4d19h, Null0

```



```

CSR3#show bgp vpnv4 unicast vrf LISPALT | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 34:3 (default for vrf LISPALT)
*> 10.9.9.9/32          0.0.0.0          1          32768 ?
*> 10.10.10.10/32       0.0.0.0          1          32768 ?

```

CSR4 is the MR for IID 0 and needs to know where to forward map-requests it receives from ITRs. Without these BGP routes, the MR has no idea where to forward the map-requests and the ITR will never be able to LISP-encapsulate any packets. A quick check on CSR4 shows that the routes were learned from CSR4 via iBGP through the P2P GRE tunnel.

```

CSR4#show vrf LISPALT
      Name          Default RD          Protocols  Interfaces
      LISPALT        34:4          ipv4,ipv6  Tu34

CSR4#show bgp vpnv4 unicast vrf LISPALT | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 34:4 (default for vrf LISPALT)
*>i 10.9.9.9/32        10.3.4.3          1    100      0 ?
*>i 10.10.10.10/32     10.3.4.3          1    100      0 ?

```

Now, we will send traffic from CSR9 to CSR10. LSIP control-plane debugging is enabled on each device so each step can be visualized. First, traffic is sourced from EID space on CSR10 destined to EID space on CSR9. As an ITR, CSR10 issues a map-request to the MR which is CSR4 (10.4.6.4). CSR10 has no idea how to reach 10.9.9.9/32 as there is no EID-to-RLOC mapping for it currently. The ITR also includes all of its RLOCs in this message (10.7.10.10 and 2001:10:7:10::10), which is why ITR RLOCs reads as 2. This is included so the ETR knows how to respond to the map-request. Notice the timestamp in the comment.

```
! CSR10, time 15:10:58.345
```

```

LISP-1: IID 0 Request processing of remote EID prefix map requests to IPv4.
LISP: Send map request type remote EID prefix
LISP: Send map request for EID prefix IID 0 10.9.9.9/32
LISP-1: Remote EID IID 0 prefix 10.9.9.9/32, Send map request (1) (sources:
<signal>, state: incomplete, rlocs: 0).
LISP-1: EID-AF IPv4, Sending map-request from 10.9.9.9 to 10.9.9.9 for EID
10.9.9.9/32, ITR-RLOCs 2, nonce 0x5AB0F557-0xEEBD265C (encap src 10.7.10.10,
dst 10.4.6.4).

```

When the MR (CSR4) receives the map-request from the ITR (CSR9), it immediately forwards it towards to MS (CSR3) inside the ALT. The reason the MR even knows about the EID prefix 10.9.9.9/32 in the first place is because of BGP in the ALT, which we saw earlier. Without this BGP route, the MR cannot sufficiently resolve the request as it is not aware of any map-servers that are authoritative for this EID. The debug says it is forwarding the map-request “to 10.9.9.9”, but resolving this recursively really means 10.3.4.4 as the BGP routes above reveal.

```

! CSR4
LISP: Processing received Map-Request(1) message on GigabitEthernet2.546 from
10.9.9.9:4342 to 10.9.9.9:4342
LISP: Received map request for IID 0 10.9.9.9/32, source_eid IID 0
10.10.10.10, ITR-RLOCs: 10.7.10.10 2001:10:7:10::10, records 1, nonce
0x5AB0F557-0xEEBD265C
LISP-0: AF IID 0 IPv4, Forwarding map request to 10.9.9.9 on the ALT.

```

When the MS receives it from the MR, it checks its IID 0 database and sees RLOC 10.7.9.9 for the EID prefix 10.9.9.9/32. There were the RLOCs that were included in the map-register message initially sent to the MS by CSR9 (as an ETR). The MS forwards the map-request down to the ETR who send the map-register, which is CSR9, using the ETR RLOC specified in the database.

```

! CSR3
LISP: Processing received Map-Request(1) message on Tunnel34 from
10.9.9.9:4342 to 10.9.9.9:4342
LISP: Received map request for IID 0 10.9.9.9/32, source_eid IID 0
10.10.10.10, ITR-RLOCs: 10.7.10.10 2001:10:7:10::10, records 1, nonce
0x5AB0F557-0xEEBD265C
LISP-0: MS EID IID 0 prefix 10.9.9.9/32 site CSR9, Forwarding map request to
ETR RLOC 10.7.9.9.

```

When the ETR receives it from the MS, the ETR verifies that this map-request is actually for an EID that it has locally registered and was destined to a valid RLOC. It sends a map-reply directly to the "better" ITR RLOC which is determined by priorities. Prioritization is discussed later, but in this case it selected the IPv4 RLOC over the IPv6 RLOC.

```

! CSR10
LISP: Processing received Map-Request(1) message on GigabitEthernet2.579 from
10.9.9.9:4342 to 10.9.9.9:4342
LISP: Received map request for IID 0 10.9.9.9/32, source_eid IID 0
10.10.10.10, ITR-RLOCs: 10.7.10.10 2001:10:7:10::10, records 1, nonce
0x5AB0F557-0xEEBD265C
LISP: Processing map request record for EID prefix IID 0 10.9.9.9/32
LISP-1: Sending map-reply from 10.7.9.9 to 10.7.10.10.

```

When the ITR receives the final map-reply from the ETR, it effectively learns the remote RLOC (LISP tunnel destination) it needs to use for encapsulation. Most of the debug below includes low-level table updates, but the first few lines show the remote RLOC clearly. The RTT is also shown, and although I have removed the timestamps, the entire process from ITR map-request to ETR map-reply was exactly 26ms. I included timestamps in the comment below for comparison against the timestamp shown above.

```

! CSR10, time 15:10:58.371
LISP: Received map reply nonce 0x5AB0F557-0xEEBD265C, records 1

```

```

LISP: Processing Map-Reply mapping record for IID 0 10.9.9.9/32, ttl 1440,
action none, authoritative, 1 locator
      10.7.9.9 pri/wei=10/10 LpR
LISP-1: Map Request IID 0 prefix 10.9.9.9/32 remote EID prefix[LL], Received
reply with rtt 26ms.
LISP: Processing mapping information for EID prefix IID 0 10.9.9.9/32
LISP-1: Remote EID IID 0 prefix 10.9.9.9/32, Change state to reused (sources:
<map-rep>, state: incomplete, rlocs: 0).
LISP-1: Remote EID IID 0 prefix 10.9.9.9/32, Starting idle timer (delay
00:02:30) (sources: <map-rep>, state: reused, rlocs: 0).
LISP-1: AF IID 0 IPv4, Persistent db: ignore writing request, disabled.
LISP-1: Remote EID IID 0 prefix 10.9.9.9/32, Change state to complete
(sources: <map-rep>, state: reused, rlocs: 0).

```

This entire process happens in the reverse direction since bidirectional communication is the goal. CSR9 would be the ITR and CSR10 would be the ETR for this process. The process is identical except reversed and is not shown again. To prove connectivity is working, we will send packets from CSR10 to CSR9 and use EPC in both directions on CSR9. Two packets are shown below. The first packet has an outer UDP (0x11, protocol 17, green) header and sends traffic from 10.7.10.10 to 10.7.9.9 (yellow) using port 4341 (0x10F5, cyan). This is the LISP data packet encapsulation and uses IP addressing from the RLOC space. The inner packet is ICMP (0x01, protocol 1, green) from 10.10.10.10 to 10.9.9.9 (yellow), which is in EID space. The second packet is the reply; the same fields are highlighted in the same colors for the return traffic.

```

CSR9#show monitor capture CAP buffer dump
0000: 000C29E0 4F84000C 29664C2C 81000DFB  ..) .O...) fL, ....
0010: 08004500 0088124A 4000FE11 42FA0A07  ..E....J@...B...
0020: 0A0A0A07 09090600 10F50074 00004000  .....t...@.
0030: 00000000 00014500 0064004A 0000FF01  .....E..d.J....
0040: 94290A0A 0A0A0A09 09090800 D2280010  .) ..... (..
0050: 00000000 000056CC 5545ABCD ABCDABCD  .....V.UE.....
[snip]

0000: 000C2966 4C2C000C 29E04F84 81000DFB  ..) fL, ..) .O.....
0010: 08004500 0088352C 4000FF11 1F180A07  ..E...5,@.....
0020: 09090A07 0A0A0600 10F50074 00004000  .....t...@.
0030: 00000000 00014500 0064004A 0000FF01  .....E..d.J....
0040: 94290A09 09090A0A 0A0A0000 DA280010  .) ..... (..
0050: 00000000 000056CC 5545ABCD ABCDABCD  .....V.UE.....
[snip]

```

After connectivity is validated, we check the LISP mapping tables on both routers to see their new entries. Each of them has the other router's EID prefix along with a valid RLOC to reach it. Both of these entries were learned via the dynamic map-request/map-reply process.

```

CSR9#show ip lisp 1 map-cache

```

LISP IPv4 Mapping Cache for EID-table default (IID 0), 2 entries

0.0.0.0/0, uptime: 2d19h, expires: never, via static send map-request

Negative cache entry, action: send-map-request

10.10.10.10/32, uptime: 00:00:31, expires: 23:59:28, via map-reply, complete

Locator	Uptime	State	Pri/Wgt
10.7.10.10	00:00:31	up	10/10

CSR10#show ip lisp 1 map-cache

LISP IPv4 Mapping Cache for EID-table default (IID 0), 2 entries

0.0.0.0/0, uptime: 4d20h, expires: never, via static send map-request

Negative cache entry, action: send-map-request

10.9.9.9/32, uptime: 00:10:08, expires: 23:49:51, via map-reply, complete

Locator	Uptime	State	Pri/Wgt
10.7.9.9	00:10:08	up	10/10

If we look at the FIB, we see interesting output. The first command is misleading because we have an exact match /32 but it says no route. Normally the lack of a route would show "0.0.0.0/0" followed by "no route". Looking at the details, we see that LISP is working behind the scenes and performing "action encap" to get the traffic over the network. The reason it says "no route" is because the RLOC space is in a VRF on CSR10; it's a shared front-door VRF used by all EID spaces across all tables. On CSR10, the EID namespace has 10.10.10.10/32 in the global table which has no routes aside from this loopback (shown in the routing table summary). The lack of any routes, or only having a default, is required to trigger LISP mapping resolution. Ultimately, the FIB recurses to having no route and should be dropped, but LISP steps in and performs the encapsulation. This is somewhat similar to the DMVPN phase 3 operation where the NHRP cache augments the FIB.

CSR10#show ip cef 10.9.9.9

10.9.9.9/32

no route

CSR10#show ip cef 10.9.9.9 detail

10.9.9.9/32, epoch 2, flags [default route handler, subtree context, check lisp eligibility, default route]

SC owned, sourced: LISP remote EID - locator status bits 0x00000001

LISP remote EID: 28 packets 2596 bytes fwd action encap

LISP source path list

nexthop 10.7.9.9 LISP1

2 IPL sources [unresolved, active source]

Dependent covered prefix type inherit, cover 0.0.0.0/0

recursive via 0.0.0.0/0

no route

CSR10#show ip route summary

IP routing table name is default (0x0)

IP routing table maximum-paths is 32

Route Source (bytes)	Networks	Subnets	Replicates	Overhead	Memory
application	0	0	0	0	0
connected	0	1	0	96	288
static	0	0	0	0	0
lisp	0	0	0	0	0
internal	1				368
Total	1	1	0	96	656

On CSR9, the output is a little different. Because both the EID prefix and RLOC namespace are in the global table, the FIB does recurse to a valid route since there is a default route in the global table, just like there is a default route in the FVRF on CSR10. The behavior is exactly the same in terms of LISP encapsulation, but the outputs are slightly different due to route recursion. LISP still trumps the FIB. There is still a specific entry for the EID prefix learned from LISP, and the action remains "action encap". Notice that the next-hops shown with "LISP1" are IPv4 addresses of the remote RLOC in use.

```
CSR9#show ip cef 10.10.10.10
10.10.10.10/32
  nexthop 10.7.9.7 GigabitEthernet2.579
```

```
CSR9#show ip cef 10.10.10.10 detail
10.10.10.10/32, epoch 2, flags [subtree context, check lisp eligibility,
default route]
  SC owned, sourced: LISP remote EID - locator status bits 0x00000001
  LISP remote EID: 19 packets 1900 bytes fwd action encap
  LISP source path list
    nexthop 10.7.10.10 LISP1
  2 IPL sources [active source]
    Dependent covered prefix type inherit, cover 0.0.0.0/0
    recursive via 0.0.0.0/0
    recursive via 10.7.9.7
    attached to GigabitEthernet2.579
```

```
CSR9#show ip route 0.0.0.0
Routing entry for 0.0.0.0/0, supernet
  Known via "static", distance 1, metric 0, candidate default path
  Routing Descriptor Blocks:
  * 10.7.9.7
    Route metric is 0, traffic share count is 1
```

LISP does an excellent job of tracking statistics. We will look at CSR9 and CSR3 as examples. Notice the high number of map-registers sent from the ETR and received on the MS. The MS also shows the map-requests forwarded. Note: When using LISP show commands, you often need to include the process number. Notice on CSR9, I must include the "1" or else the show command does not work. On CSR3, I use the general LISP process with no PID.

```
CSR9#show ip lisp 1 statistics instance-id 0 | section Map-Re
```

```

Map-Requests in/out: 1/1
  Encapsulated Map-Requests in/out: 1/1
  RLOC-probe Map-Requests in/out: 0/0
  SMR-based Map-Requests in/out: 0/0
  Map-Requests expired on-queue/no-reply 0/0
  Map-Resolver Map-Requests forwarded: 0
  Map-Server Map-Requests forwarded: 0
Map-Reply records in/out: 1/1
  Authoritative records in/out: 1/1
  Non-authoritative records in/out: 0/0
  Negative records in/out: 0/0
  RLOC-probe records in/out: 0/0
  Map-Server Proxy-Reply records out: 0
Map-Register records in/out: 0/4492
  Map-Server AF disabled: 0
  Authentication failures: 0
  Map-Reply deferred/dropped: 0/0
  MR negative Map-Reply deferred/dropped: 0/0
  MR Map-Request fwd deferred/dropped: 0/0
  MS Map-Request fwd deferred/dropped: 0/0
  MS proxy Map-Reply deferred/dropped: 0/0
  RTR Map-Register fwd deferred/dropped: 0/0
[snip]

```

CSR3#show ip lisp statistics instance-id 0 | section Map-Re

```

Map-Requests in/out: 4/4
  Encapsulated Map-Requests in/out: 0/4
  RLOC-probe Map-Requests in/out: 0/0
  SMR-based Map-Requests in/out: 0/0
  Map-Requests expired on-queue/no-reply 0/0
  Map-Resolver Map-Requests forwarded: 0
  Map-Server Map-Requests forwarded: 4
Map-Reply records in/out: 0/0
  Authoritative records in/out: 0/0
  Non-authoritative records in/out: 0/0
  Negative records in/out: 0/0
  RLOC-probe records in/out: 0/0
  Map-Server Proxy-Reply records out: 0
Map-Register records in/out: 14219/0
  Map-Server AF disabled: 0
  Authentication failures: 0
  Map-Reply deferred/dropped: 0/0
  MR negative Map-Reply deferred/dropped: 0/0
  MR Map-Request fwd deferred/dropped: 0/0
  MS Map-Request fwd deferred/dropped: 0/0
  MS proxy Map-Reply deferred/dropped: 0/0
  RTR Map-Register fwd deferred/dropped: 0/0
[snip]

```

In many environments, the MR/MS are consolidated onto a single device. Although the MR and MS roles remain the same, the actual forwarding of the map-request from MR to MS occurs in software within the router. That is, there is no need for an ALT in this model. At present, the ALT feature only appears to support EIDs in IID 0, so for testing EID prefixes inside VRFs, we consolidate the MR/MS services on CSR3. When configuring EID prefixes not within the default IID, you must explicitly specify the IID when defining the EID prefix. All three sites have prefixes in both IID 101 and 102 as shown below. The “accept-more-specifics” option allows a more generic registration to occur where the MS does not need to enumerate individual prefixes that are contiguous. I also introduce IPv6, but LISP does an excellent job of providing feature and configuration parity between IPv4 and IPv6. The concepts are identical.

```
! CSR3
router lisp
site CSR10
  authentication-key CSR10
  eid-prefix instance-id 101 10.1.10.0/23 accept-more-specifics
  eid-prefix instance-id 102 10.10.10.10/32
  eid-prefix instance-id 102 2001:10:10:10::10/128

site CSR2
  authentication-key CSR2
  eid-prefix instance-id 101 10.2.2.2/32
  eid-prefix instance-id 102 10.2.2.2/32
  eid-prefix instance-id 102 2001:10:2:2::2/127 accept-more-specifics

site CSR9
  authentication-key CSR9
  eid-prefix instance-id 101 10.1.9.0/24
  eid-prefix instance-id 101 10.9.9.9/32
  eid-prefix instance-id 102 10.9.9.9/32
  eid-prefix instance-id 102 2001:10:9:9::9/128

ipv4 map-server
ipv4 map-resolver
ipv6 map-server
ipv6 map-resolver
```

We also introduce CSR2 using the parallel virtualization model. The relevant configurations for that are below. The EID prefixes in VRF C1 and C2 are not included as those are simply endpoints. Each LISP process maps to an IID (VRF) because each LISP process can only specify a single “locator-table”. The configurations are largely redundant but this creates additional isolation between LISP virtual instances on a since xTR.

```
! CSR2
interface GigabitEthernet2.520
  vrf forwarding C2
  ip address 10.8.2.2 255.255.255.0
```

```

ipv6 address 2001:10:8:2::2/64

interface GigabitEthernet2.528
 vrf forwarding C1
 ip address 10.2.8.2 255.255.255.0
 ipv6 address 2001:10:2:8::2/64

router lisp 1
 locator-table vrf C1
 eid-table vrf C1 instance-id 101
 database-mapping 10.2.2.2/32 IPv4-interface GigabitEthernet2.528 priority
10 weight 10
 ipv4 itr map-resolver 10.3.6.3
 ipv4 itr
 ipv4 etr map-server 10.3.6.3 key CSR2
 ipv4 etr

router lisp 2
 locator-table vrf C2
 eid-table vrf C2 instance-id 102
 database-mapping 10.2.2.2/32 IPv4-interface GigabitEthernet2.520 priority
20 weight 10
 database-mapping 10.2.2.2/32 IPv6-interface GigabitEthernet2.520 priority
15 weight 10
 database-mapping 2001:10:2:2::2/128 IPv4-interface GigabitEthernet2.520
priority 10 weight 10
 database-mapping 2001:10:2:2::2/128 IPv6-interface GigabitEthernet2.520
priority 15 weight 10
 ipv4 itr map-resolver 10.3.6.3
 ipv4 itr
 ipv4 etr map-server 10.3.6.3 key CSR2
 ipv4 etr
 ipv6 itr map-resolver 10.3.6.3
 ipv6 itr
 ipv6 etr map-server 10.3.6.3 key CSR2
 ipv6 etr

```

For practice, we will trace the LISP control-plane process again, except this time we will send traffic between CSR10 and CSR2 within a customer's VRF EID space (VRF C2). This VRF uses IID 102. As a quick check, we can verify that the MS knows about the 10.10.10.10/32 and 10.2.2.2/32 EID prefixes from each site within IID 102.

```

CSR3#show lisp site instance-id 102
LISP Site Registration Information
* = Some locators are down or unreachable
Site Name      Last      Up    Who Last      Inst      EID Prefix
                Register    Registered    ID
CSR10          00:00:26  yes   10.7.10.10   102       10.10.10.10/32

```


	00:00:07	yes	10.7.10.10	102	2001:10:10:10::10/128
CSR2	00:00:17	yes	10.8.2.2	102	10.2.2.2/32
	never	no	--	102	2001:10:2:2::2/127
	00:00:43	yes	10.8.2.2	102	2001:10:2:2::2/128
CSR9	00:00:51	yes	10.7.9.9	102	10.9.9.9/32
	00:00:50	yes	10.7.9.9	102	2001:10:9:9::9/128

A quick debug on CSR3 shows CSR10 and CSR2 registering their IID 102 prefixes. Notice that each prefix has an IPv4 and IPv6 RLOC.

```

! CSR3
CSR3#debug lisp control-plane map-server-registration
LISP: Processing received Map-Register(3) message on GigabitEthernet2.536
from 10.7.10.10:4342 to 10.3.6.3:4342
LISP: Processing Map-Register no proxy, map-notify, no merge, no security, no
mobile-node, not to-RTR, no fast-map-register, no EID-notify, ID-included, 1
record, nonce 0x23C5A837-0x5B4889E2, key-id 1, auth-data-len 20, hash-
function sha1, xTR-ID 0x86671414-0xB88BC82F-0x428A5BA4-0xB567A9B7, site-ID
unspecified
LISP: Processing Map-Register mapping record for IID 102 10.10.10.10/32, ttl
1440, action none, authoritative, 2 locators
10.7.10.10 pri/wei=10/10 LpR
2001:10:7:10::10 pri/wei=15/10 LpR
LISP-0: MS registration IID 102 prefix 10.10.10.10/32 10.7.10.10 site CSR10,
Updating.

LISP: Processing received Map-Register(3) message on GigabitEthernet2.536
from 10.8.2.2:4342 to 10.3.6.3:4342
LISP: Processing Map-Register no proxy, map-notify, no merge, no security, no
mobile-node, not to-RTR, no fast-map-register, no EID-notify, ID-included, 1
record, nonce 0xA97E340B-0xA2AA248E, key-id 1, auth-data-len 20, hash-
function sha1, xTR-ID 0xB5353617-0x22428435-0xF93808BB-0x80454A5A, site-ID
unspecified
LISP: Processing Map-Register mapping record for IID 102 10.2.2.2/32, ttl
1440, action none, authoritative, 2 locators
10.8.2.2 pri/wei=10/10 LpR
2001:10:8:2::2 pri/wei=15/10 LpR
LISP-0: MS registration IID 102 prefix 10.2.2.2/32 10.8.2.2 site CSR2,
Updating.

```

For another test, we originate traffic on CSR2, which uses the parallel model for virtualization. This means there is a direct link to the SP inside this VRF, which implies there is also a default route if LISP is going to be used for lookups. We confirm this before continuing.

```

CSR2#show ip route vrf C2 0.0.0.0 0.0.0.0
Routing Table: C2
Routing entry for 0.0.0.0/0, supernet
Known via "static", distance 1, metric 0, candidate default path

```

Routing Descriptor Blocks:

* 10.8.2.8

Route metric is 0, traffic share count is 1

When CSR2 sends traffic (sourced from within the proper VRF), it issues a map-request to the MR, which is CSR3 for IID 102. This is CSR2's way of asking for the valid RLOCs for EID prefix 10.10.10.10/32 within the context of IID 102.

! CSR2, time 17:15:25.650

LISP-2: IID 102 Request processing of remote EID prefix map requests to IPv4.

LISP: Send map request type remote EID prefix

LISP: Send map request for EID prefix IID 102 10.10.10.10/32

LISP-2: Remote EID IID 102 prefix 10.10.10.10/32, Send map request (1)
(sources: <signal>, state: incomplete, rlocs: 0).

LISP-2: EID-AF IPv4, Sending map-request from 10.10.10.10 to 10.10.10.10 for
EID 10.10.10.10/32, ITR-RLOCs 2, nonce 0x9701B20F-0x8488F010 (encap src
10.8.2.2, dst 10.3.6.3).

When CSR3 receives the map-request (as the MR), it knows that it's also an MS so there is no need to consult the ALT for a BGP route. Instead, the local database on the MS is checked against the current EID-to-RLOC mappings. The check is successful, and the ETR who registered this mapping is CSR10. The MS forwards the map-request for CSR10.

! CSR3

LISP: Processing received Map-Request(1) message on GigabitEthernet2.536 from
10.10.10.10:4342 to 10.10.10.10:4342

LISP: Received map request for IID 102 10.10.10.10/32, source_eid IID 102
10.2.2.2, ITR-RLOCs: 10.8.2.2 2001:10:8:2::2, records 1, nonce 0x9701B20F-
0x8488F010

LISP-0: MS EID IID 102 prefix 10.10.10.10/32 site CSR10, Forwarding map
request to ETR RLOC 10.7.10.10.

When CSR10 receives the map-request (as the ETR), it performs a sanity check to ensure the EID-to-RLOC mappings are correct, then sends the map-reply directly to the ITR, which is CSR2. Interestingly, CSR10 uses the IPv6 RLOCs for this map-reply for the transport mechanism.

! CSR10

LISP: Processing received Map-Request(1) message on GigabitEthernet2.570 from
10.10.10.10:4342 to 10.10.10.10:4342

LISP: Received map request for IID 102 10.10.10.10/32, source_eid IID 102
10.2.2.2, ITR-RLOCs: 10.8.2.2 2001:10:8:2::2, records 1, nonce 0x9701B20F-
0x8488F010

LISP: Processing map request record for EID prefix IID 102 10.10.10.10/32

LISP-1: Sending map-reply from 2001:10:7:10::10 to 2001:10:8:2::2.

When CSR10 receives the map-reply (as the ITR), it measures the time for the LISP signaling process to occur, which was 33 ms. CSR2 then installs this EID-to-RLOC mapping as valid and uses these mappings to augment the FIB. The debug didn't show the RLOC mappings here; I cannot explain why.

```
! CSR2, time 17:15:25.683
LISP-2: Map Request IID 102 prefix 10.10.10.10/32 remote EID prefix[LL],
Received reply with rtt 33ms.
LISP: Processing mapping information for EID prefix IID 102 10.10.10.10/32
LISP-2: Remote EID IID 102 prefix 10.10.10.10/32, Change state to reused
(sources: <map-rep>, state: incomplete, rlocs: 0).
LISP-2: Remote EID IID 102 prefix 10.10.10.10/32, Starting idle timer (delay
00:02:30) (sources: <map-rep>, state: reused, rlocs: 0).
LISP-2: AF IID 102 IPv4, Persistent db: ignore writing request, disabled.
LISP-2: Remote EID IID 102 prefix 10.10.10.10/32, Change state to complete
(sources: <map-rep>, state: reused, rlocs: 0).
```

Again, we use EPC to validate the traffic flow on CSR7, this time facing CSR10. Two packets are shown below. The first packet is UDP (0x11, protocol 17, green) and sends traffic from 10.8.2.2 to 10.7.10.10 (yellow) using port 4341 (0x10F5, cyan). This is the LISP encapsulation and uses IP addressing from the RLOC space. The inner packet is ICMP (0x01, protocol 1, green) from 10.2.2.2 to 10.10.10.10 (yellow), which is in EID space. The second packet is the reply; the same fields are highlighted in the same colors for the return traffic.

```
CSR7#show monitor capture CAP buffer dump
0000: 000C2949 7164000C 29664C2C 81000DF2 ..) Iqd..) fL,....
0010: 08004500 00880046 4000FC11 5E040A08 ..E.....F@...^...
0020: 02020A07 0A0A0D00 10F50074 00004800 .....t..H.
0030: 00000000 66034500 006400AF 0000FF01 ....f.E..d.....
0040: 9AD20A02 02020A0A 0A0A0800 50820027 .....P..'
0050: 00000000 00005710 D690ABCD ABCDABCD .....W.....
[snip]

0000: 000C2966 4C2C000C 29497164 81000DF2 ..) fL,..) Iqd....
0010: 08004500 008812A8 4000FF11 48A20A07 ..E.....@...H...
0020: 0A0A0A08 02020D00 10F50074 00004800 .....t..H.
0030: 00000000 66034500 006400AF 0000FF01 ....f.E..d.....
0040: 9AD20A0A 0A0A0A02 02020000 58820027 .....X..'
0050: 00000000 00005710 D690ABCD ABCDABCD .....W.....
[snip]
```

Earlier we saw that both of these prefixes had multiple RLOCs. Specifically, one was IPv4 and one was IPv6. The lower priorities are preferred; right now the IPv4 RLOCs have priority 10 while IPv6 RLOCs have 15. IPv6 RLOCs are defined identically to the IPv4 ones and are not detailed in the inline configuration snippets. After increasing the IPv4 RLOC priorities to 20 on both CSR2 and CSR10, we still have reachability. The EPC shows that the LISP encapsulation (RLOC space) is using IPv6 while the inner

packet (EID space) still uses IPv4. This permits any combination of IPvX in IPvX tunneling. Before sending traffic, we can validate that the MS knows about these RLOC priority changes.

```
CSR3#show lisp site 10.2.2.2/32 instance-id 102 | begin Locator
Locator      Local  State  Pri/Wgt  Scope
10.8.2.2     yes   up     20/10    IPv4 none
2001:10:8:2::2 yes   up     15/10    IPv6 none
```

```
CSR3#show lisp site 10.10.10.10/32 instance-id 102 | begin Locator
Locator      Local  State  Pri/Wgt  Scope
10.7.10.10   yes   up     20/10    IPv4 none
2001:10:7:10::10 yes   up     15/10    IPv6 none
```

First, we need to clear the map-cache entries on both CSR2 and CSR10 so they can re-signal the LISP path with the MR/MS.

```
CSR2#clear ip lisp 2 map-cache *
CSR10#clear ip lisp 1 map-cache *
```

The EPC is shown again below after sending traffic from CSR2 to CSR10. We can immediately see the packet is significantly larger, which hints towards the usage of IPv6. Two packets are shown below. The first packet has an outer IP header which wraps UDP (0x11, protocol 17, green) and sends traffic from 2001:10:8:2::2 to 2001:10:7:10::10 (yellow) using port 4341 (0x10F5, cyan). This is the LISP encapsulation and uses IP addressing from the RLOC space. The inner packet is ICMP (0x01, protocol 1, green) from 10.2.2.2 to 10.10.10.10 (yellow), which is in EID space. The second packet is the reply; the same fields are highlighted in the same colors for the return traffic.

```
CSR7#show monitor capture CAP buffer dump
0000: 000C2949 7164000C 29664C2C 81000DF2 ..) Iqd..) fL,....
0010: 86DD6000 00000074 11FC2001 00100008 ..`....t.. ..
0020: 00020000 00000000 00022001 00100007 .....
0030: 00100000 00000000 00100D00 10F50074 .....t
0040: 00004800 00000000 66034500 006400C3 ..H....f.E..d..
0050: 0000FF01 9ABE0A02 02020A0A 0A0A0800 .....
0060: F15E002B 00000000 00005722 359EABCD .^.+.....W"5...
[snip]

0000: 000C2966 4C2C000C 29497164 81000DF2 ..) fL, ..) Iqd....
0010: 86DD6000 00000074 11FE2001 00100007 ..`....t.. ..
0020: 00100000 00000000 00102001 00100008 .....
0030: 00020000 00000000 00020D00 10F50074 .....t
0040: 00004800 00000000 66034500 006400C3 ..H....f.E..d..
0050: 0000FF01 9ABE0A0A 0A0A0A02 02020000 .....
0060: F95E002B 00000000 00005722 359EABCD .^.+.....W"5...
[snip]
```

We can verify the map-cache entries on both CSR2 and CSR10. We use the summary view on CSR2 and the detailed view on CSR10 for variety. Note that the xTR maintains all of the RLOCs assigned to an EID prefix, and locally selects the best one based on the advertised priorities. It is not prescribed by a remote LISP node; the remote node influences the decision via the priority settings.

```
CSR2#show ip lisp 2 instance-id 102 map-cache
LISP IPv4 Mapping Cache for LISP 2 EID-table vrf C2 (IID 102), 2 entries
0.0.0.0/0, uptime: 00:15:36, expires: never, via static send map-request
Negative cache entry, action: send-map-request
10.10.10.10/32, uptime: 00:15:34, expires: 23:44:25, via map-reply, complete
Locator      Uptime      State      Pri/Wgt
10.7.10.10    00:15:34    up         20/10
2001:10:7:10::10 00:15:34    up         15/10
```

```
CSR10#show ip lisp 1 instance-id 102 map-cache 10.2.2.2/32
LISP IPv4 Mapping Cache for EID-table vrf C2 (IID 102), 2 entries
10.2.2.2/32, uptime: 00:16:15, expires: 23:43:45, via map-reply, complete
Sources: map-reply
State: complete, last modified: 00:16:15, map-source: 10.8.2.2
Idle, Packets out: 8 (~ 00:15:05 ago)
Locator      Uptime      State      Pri/Wgt
10.8.2.2     00:16:15    up         20/10
  Last up-down state change:      00:16:15, state change count: 1
  Last route reachability change:  00:16:15, state change count: 1
  Last priority / weight change:   never/never
  RLOC-probing loc-status algorithm:
    Last RLOC-probe sent:         never
2001:10:8:2::2 00:16:15    up         15/10
  Last up-down state change:      00:16:15, state change count: 1
  Last route reachability change:  00:16:15, state change count: 1
  Last priority / weight change:   never/never
  RLOC-probing loc-status algorithm:
    Last RLOC-probe sent:         never
```

Verifying the FIB, we see the specific EID prefix with "action encaps" on both sides. Notice that the LISP next-hops are the IPv6 RLOCs now since it was the "better" RLOC per our manual priority modifications. Due to virtualization in play, the "LISP" interfaces have subinterfaces with a number matching the IID.

```
CSR2#show ip cef vrf C2 10.10.10.10 detail
10.10.10.10/32, epoch 0, flags [subtree context, check lisp eligibility,
default route]
SC owned, sourced: LISP remote EID - locator status bits 0x00000003
LISP remote EID: 18 packets 1666 bytes fwd action encaps
LISP source path list
  nexthop 2001:10:7:10::10 LISP2.102
2 IPL sources [active source]
  Dependent covered prefix type inherit, cover 0.0.0.0/0
recursive via 0.0.0.0/0
```

```

recursive via 10.8.2.8
attached to GigabitEthernet2.520

CSR10#show ip cef vrf C2 10.2.2.2 detail
10.2.2.2/32, epoch 0, flags [default route handler, subtree context, check
lisp eligibility, default route]
  SC owned, sourced: LISP remote EID - locator status bits 0x00000003
  LISP remote EID: 8 packets 800 bytes fwd action encaps
  LISP source path list
    nexthop 2001:10:8:2::2 LISP1.102
  2 IPL sources [unresolved, active source]
    Dependent covered prefix type inherit, cover 0.0.0.0/0
recursive via 0.0.0.0/0
no route

```

A more common usage of IPv6 with LISP would be to tunnel it inside IPv4 to connect two disjoint IPv6 "islands". We just demonstrated the opposite which, while possible, has much less utility. Assuming the SP did not provide VPNv6 transport, we can use LISP as an IPv6 transition/tunneling mechanism. We won't trace all the signaling debugs since the IPv4/IPv6 RLOCs are carried in the same LISP mapping messages. First, we check the MS (CSR3) to ensure it knows about CSR2 and CSR10 IID 102 IPv6 prefixes. These are 2001:10:2:2::2/128 and 2001:10:10:10::10/128, respectively. Notice that there is an entry for 2001:10:2:2::2/127 also (less specific); I configured the MS for the /127 but told it to accept longer-matches, which is why the /128 was able to register successfully. The configuration snippets shown above demonstrated this option.

```

CSR3#show lisp site instance-id 102
LISP Site Registration Information
* = Some locators are down or unreachable

```

Site Name	Last Register	Up	Who Last Registered	Inst ID	EID Prefix
CSR10	00:00:02	yes	10.7.10.10	102	10.10.10.10/32
	00:00:53	yes	10.7.10.10	102	2001:10:10:10::10/128
CSR2	00:00:59	yes	10.8.2.2	102	10.2.2.2/32
	never	no	--	102	2001:10:2:2::2/127
CSR9	00:00:35	yes	10.8.2.2	102	2001:10:2:2::2/128
	00:00:27	yes	10.7.9.9	102	10.9.9.9/32
	00:00:40	yes	10.7.9.9	102	2001:10:9:9::9/128

The MS also knows the priorities of the different RLOCs. In this case, IPv4 RLOCs have a lower (better) priority, so we expect those to be used for traffic forwarding. This is the opposite test of what we just did; we transition from IPv4-in-IPv6 to IPv6-in-IPv4. We simply change the priority of the IPv4 RLOCs back to 10 to re-baseline the network.

```

CSR3#show lisp site 2001:10:2:2::2/128 instance-id 102 | begin Locator
Locator      Local  State  Pri/Wgt  Scope
10.8.2.2     yes    up     10/10    IPv4 none

```

```
2001:10:8:2::2  yes      up          15/10    IPv6 none
```

```
CSR3#show lisp site 2001:10:10:10::10/128 instance-id 102 | begin Locator
```

Locator	Local	State	Pri/Wgt	Scope
10.7.10.10	yes	up	10/10	IPv4 none
2001:10:7:10::10	yes	up	15/10	IPv6 none

When we send traffic between the two sites, we can look for the IPv6-in-IPv4 encapsulation on CSR7 outbound towards CSR10. Two packets are shown below. The first packet has an outer UDP protocol type (0x11, protocol 17, green) and sends traffic from 10.8.2.2 to 10.7.10.10 (yellow) using port 4341 (0x10F5, cyan). This is the LISP encapsulation and uses IP addressing from the RLOC space. The inner packet is IPv6-ICMP (0x3A, protocol 58, green) from 2001:10:2:2::2 to 2001:10:10:10::10 (yellow), which is in EID space. The second packet is the reply; the same fields are highlighted in the same colors for the return traffic.

```
CSR7#show monitor capture CAP buffer dump
```

```
0000: 000C2949 7164000C 29664C2C 81000DF2  ..) Iqd..) fL,....
0010: 08004500 0088005E 4000FC11 5DEC0A08  ..E....^@...]...
0020: 02020A07 0A0A0454 10F50074 00004800  ....T...t..H.
0030: 00000000 66036000 0000003C 3A402001  ....f.`....<:@ .
0040: 00100002 00020000 00000000 00022001  .....
0050: 00100010 00100000 00000000 00108000  .....
0060: AF420348 00000001 02030405 06070809  .B.H.....
[snip]
```

```
0000: 000C2966 4C2C000C 29497164 81000DF2  ..) fL,..) Iqd....
0010: 08004500 008812F4 4000FF11 48560A07  ..E.....@...HV..
0020: 0A0A0A08 02020454 10F50074 00004800  ....T...t..H.
0030: 00000000 66036000 0000003C 3A402001  ....f.`....<:@ .
0040: 00100010 00100000 00000000 00102001  .....
0050: 00100002 00020000 00000000 00028100  .....
0060: AE420348 00000001 02030405 06070809  .B.H.....
[snip]
```

The data-plane is working, so let's verify more control-plane components. The map-cache on both CSR10 (summary) and CSR2 (detailed) shows all available RLOCs along with their priorities. The IPv4 RLOCs are used due to having lower priority values.

```
CSR10#show ipv6 lisp 1 instance-id 102 map-cache
```

```
LISP IPv6 Mapping Cache for EID-table vrf C2 (IID 102), 2 entries
```

```
::/0, uptime: 3d04h, expires: never, via static send map-request
```

```
Negative cache entry, action: send-map-request
```

```
2001:10:2:2::2/128, uptime: 00:05:04, expires: 23:54:55, via map-reply,
complete
```

Locator	Uptime	State	Pri/Wgt
10.8.2.2	00:05:04	up	10/10
2001:10:8:2::2	00:05:04	up	15/10

```
CSR2#show ipv6 lisp 2 instance-id 102 map-cache 2001:10:10:10::10/128
LISP IPv6 Mapping Cache for LISP 2 EID-table vrf C2 (IID 102), 2 entries
2001:10:10:10::10/128, uptime: 00:05:33, expires: 23:54:26, via map-reply,
complete
```

Sources: map-reply

State: complete, last modified: 00:05:33, map-source: 2001:10:7:10::10
Idle, Packets out: 28 (~ 00:04:54 ago)

Locator	Uptime	State	Pri/Wgt
10.7.10.10	00:05:33	up	10/10

Last up-down state change: 00:05:33, state change count: 1

Last route reachability change: 00:38:11, state change count: 1

Last priority / weight change: never/never

RLOC-probing loc-status algorithm:

Last RLOC-probe sent: never

2001:10:7:10::10	00:05:33	up	15/10

Last up-down state change: 00:05:33, state change count: 1

Last route reachability change: 00:38:11, state change count: 1

Last priority / weight change: never/never

RLOC-probing loc-status algorithm:

Last RLOC-probe sent: never

We can verify the outer (LISP) tunneling encapsulation by checking the IPv6 FIB on both routers. The LISP path is an IPv4 RLOC in both cases.

```
CSR2#show ipv6 cef vrf C2 2001:10:10:10::10/128 detail
2001:10:10:10::10/128, epoch 0, flags [subtree context, check lisp
eligibility, default route]
SC owned, sourced: LISP remote EID - locator status bits 0x00000003
LISP remote EID: 28 packets 2596 bytes fwd action encap
LISP source path list
  nexthop 10.7.10.10 LISP2.102
2 IPL sources [active source]
  Dependent covered prefix type inherit, cover ::/0
recursive via ::/0
  recursive via 2001:10:8:2::8
  attached to GigabitEthernet2.520
```

```
CSR10#show ipv6 cef vrf C2 2001:10:2:2::2/128 detail
2001:10:2:2::2/128, epoch 0, flags [default route handler, subtree context,
check lisp eligibility, default route]
SC owned, sourced: LISP remote EID - locator status bits 0x00000003
LISP remote EID: 26 packets 1898 bytes fwd action encap
LISP source path list
  nexthop 10.8.2.2 LISP1.102
2 IPL sources [unresolved, active source]
  Dependent covered prefix type inherit, cover ::/0
recursive via ::/0
```


The minor difference in outputs is due to CSR2 having a default route within the VRF (required for parallel model virtualization) while CSR10 has no routes within the VRF C2 other than the connected/local loopbacks. This has no effect on connectivity.

```
CSR2#show ipv6 route vrf C2 ::/0
Routing entry for ::/0
  Known via "static", distance 1, metric 0
  Route count is 1/1, share count 0
  Routing paths:
    2001:10:8:2::8
    Last updated 1w0d ago
```

```
CSR10#show ipv6 route vrf C2 summary
IPv6 routing table name is C2(1E000003) global scope - 2 entries
IPv6 routing table default maximum-paths is 16
Route Source      Networks      Overhead      Memory (bytes)
connected         0             0              0
local             2            288           416
application       0             0              0
ND                0             0              0
  Default: 0 Prefix: 0 Destination: 0 Redirect: 0
static            0             0              0
  Static: 0 Per-user static: 0
lisp              0             0              0
  LISP alt: 0 LISP site-registrations: 0 LISP dyn-eid: 0
Total            2            288           416
```

```
Number of prefixes:
  /8: 1, /128: 1
no route
```

Next, we will look at using CSR6 as a Pitr to provide interworking from a non-LISP site (CSR5) to a LISP site (CSR9) within IID 101 specifically. The Pitr provides the opposite, giving access from LISP to non-LISP sites. Because we are using virtualization and the VRF on CSR9 has no routes, we need the Pitr also. The xTr has no idea about the Pitr but must be configured to use the Pitr. The PxTr services can exist inside a VRF, and since CSR6 is the MPLS L3VPN PE, the PxTr address will be 22.5.6.6 within VRF NONLISP.

```
! CSR6
interface GigabitEthernet2.556
  vrf forwarding NONLISP
  ip address 22.5.6.6 255.255.255.0
  ipv6 address 2001:22:5:6::6/64

router lisp
  locator-table vrf NONLISP
```

```

eid-table vrf NONLISP instance-id 101
  map-cache 10.1.10.0/24 map-request
  map-cache 10.2.2.2/32 map-request
  map-cache 10.9.9.9/32 map-request
ipv4 proxy-etr
ipv4 proxy-itr 22.5.6.6
ipv4 itr map-resolver 10.3.6.3

! CSR9
router lisp 1
  eid-table vrf C1 instance-id 101
  ipv4 use-petr 22.5.6.6

```

CSR9, as an ITR, sends a normal map-request to the MR. The MR is CSR3 for IID 101 and should receive the map-request from CSR9 once the customer in the VRF starts sending traffic. CSR9 is looking for the public address 22.5.6.5 which is in RLOC space, not EID space.

```

! CSR9, time 14:19:07.186
LISP-1: IID 101 Request processing of remote EID prefix map requests to IPv4.
LISP: Send map request type remote EID prefix
LISP: Send map request for EID prefix IID 101 22.5.6.5/32
LISP-1: Remote EID IID 101 prefix 22.5.6.5/32, Send map request (1) (sources:
<signal>, state: incomplete, rlocs: 0).
LISP-1: EID-AF IPv4, Sending map-request from 22.5.6.5 to 22.5.6.5 for EID
22.5.6.5/32, ITR-RLOCs 2, nonce 0xCE3DD210-0x48740DDB (encap src 10.7.9.9,
dst 10.3.6.3).

```

The MR/MS receives the map-request from the ETR (CSR9) and sees that it does not have any EID-to-RLOC mappings for this prefix within IID 101. It returns a "negative" reply with a course aggregate of 16.0.0.0/4 (which covers 22.5.6.5).

```

! CSR3
LISP: Processing received Map-Request(1) message on GigabitEthernet2.536 from
22.5.6.5:4342 to 22.5.6.5:4342
LISP: Received map request for IID 101 22.5.6.5/32, source_eid IID 101
10.9.9.9, ITR-RLOCs: 10.7.9.9 2001:10:7:9::9, records 1, nonce 0xCE3DD210-
0x48740DDB
LISP-0: sending negative map-reply from 10.3.6.3 to 10.7.9.9 for IID 101
16.0.0.0/4.

```

The ETR (CSR9) receives the negative reply directly from the MS (CSR3) and adds an entry to its LISP map-cache and FIB for 16.0.0.0/4. We examine this in more detail once the LISP control-plane converges. The lack of an RLOC means that LISP has no choice but to forward the traffic natively, at least so far. Notice the time difference was about ~22 ms.

```

! CSR9, time 14:19:07.209

```

```

LISP: Received map reply nonce 0xCE3DD210-0x48740DDB, records 1
LISP: Processing Map-Reply mapping record for IID 101 16.0.0.0/4, ttl 15,
action forward, authoritative, 0 locators
LISP-1: Map Request IID 101 prefix 22.5.6.5/32 remote EID prefix[LL],
Received reply with rtt 22ms.
LISP: Processing mapping information for EID prefix IID 101 16.0.0.0/4
LISP-1: AF IID 101 IPv4, Persistent db: ignore writing request, disabled.
LISP-1: Remote EID IID 101 prefix 16.0.0.0/4, Change state to forward-native
(sources: <map-reply>, state: unknown, rlocs: 0).

```

At this point, CSR9 is sending encapsulated LISP packets to the PETR at 22.5.6.6 (CSR6). Without the PETR, the traffic would have actually been natively forwarded, but the whole purpose of the PETR is to provide an ETR service for CSR9 when talking to a non-LISP site. The ITR (CSR9) will have a CEF entry for this /4 prefix. The outputs between the LISP map-cache and FIB appear inconsistent. The FIB says "action encap" while the map-cache says forward-native. The action is actually encap; at the bottom of the map-cache output, it mentions that LISP encap is actually being used to the PETR, so the FIB resolves this and presents the final result as we expect. The LISP-encapsulated next-hop is the PETR IP address.

```

CSR9#show ip cef vrf C1 22.5.6.5 detail
16.0.0.0/4, epoch 0, flags [default route handler, subtree context, check
lisp eligibility, default route]
  SC owned,sourced: LISP remote EID - locator status bits 0x00000000
  LISP remote EID: 18 packets 1674 bytes fwd action encap
  LISP source path list
    nexthop 22.5.6.6 LISP1.101
  2 IPL sources [unresolved, active source]
    Dependent covered prefix type inherit, cover 0.0.0.0/0
  recursive via 0.0.0.0/0
  no route

CSR9#show ip lisp 1 map-cache 16.0.0.0/4 instance-id 101
LISP IPv4 Mapping Cache for EID-table vrf C1 (IID 101), 4 entries
16.0.0.0/4, uptime: 00:00:36, expires: 00:14:24, via map-reply, forward-
native
  Sources: map-reply
  State: forward-native, last modified: 00:00:36, map-source: 2001:10:3:6::3
  Active, Packets out: 0
  Encapsulating to proxy ETR

```

The PITR has not been in play yet since it has not received any traffic inbound from the non-LISP site. A quick EPC is used to verify packets are, at this point, getting to the non-LISP destination at 22.5.6.5 (CSR5). The capture from CSR7 inbound (facing CSR9) is shown below. Notice it is a LISP-encapsulated packet destined for the PETR's IP address (22.5.6.6) as CSR9's FIB directs. The PETR will decapsulate the packet and send the native ICMP packet to CSR5. Two captures are shown below: the first is CSR7 inbound (LISP) and the second is CSR5 inbound (ICMP). The first packet dump shows the full IP packet inside LISP for proof. The inner packet is ICMP (0x01, protocol 1, green) from 10.9.9.9 to 22.5.6.5, where

the source is in EID space and the destination is in RLOC space. The outer encapsulation is UDP (0x11, protocol 17, green) and sends traffic from 10.7.9.9 to 22.5.6.6 (yellow) using port 4341 (0x10F5, cyan). The destination of the LISP frame is the PETR address as discussed earlier. CSR6 decapsulated those packets and forwards them to CSR5 as native ICMP.

```
CSR7#show monitor capture CAP buffer dump
0
0000: 000C2966 4C2C000C 29E04F84 81000DFB  ..)fL,..).O.....
0010: 08004500 00884629 4000FF11 06210A07  ..E...F)@....!...
0020: 09091605 06061600 10F50074 00004800  .....t...H.
0030: 00000000 65014500 00640098 0000FF01  ....e.E..d.....
0040: 8BE50A09 09091605 06050800 D0A50025  .....%
0050: 00000000 000014F4 988BABCD ABCDABCD  .....
[snip]
```

```
CSR5#show monitor capture CAP buffer detailed
0 118 0.000000 10.9.9.9 -> 22.5.6.5 ICMP
0000: 000C295F 11A1000C 2993FE00 81000DE4  ..)_....).....
0010: 08004500 00640098 0000FC01 8EE50A09  ..E..d.....
0020: 09091605 06050800 D0A50025 00000000  .....%....
0030: 000014F4 988BABCD ABCDABCD ABCDABCD  .....
```

When CSR5 goes to respond towards 10.9.9.9, it follows its default route. The SP has no idea how to reach 10.9.9.9 since this exists in EID space only and is not part of any VPNv4 BGP table. This is where the PITR is used (interworking from non-LISP to LISP). The PITR is statically configured to issue map-requests for a number of ranges, including 10.9.9.9/32, and it does so just like a normal ITR. The fact that it is a PITR is specified in the map-request.

```
! CSR6, time 14:19:03.302
LISP-0: IID 101 Request processing of remote EID prefix map requests to IPv4.
LISP: Send map request type remote EID prefix
LISP: Send map request for EID prefix IID 101 10.9.9.9/32
LISP-0: Remote EID IID 101 prefix 10.9.9.9/32, Send map request (1) (sources:
<signal|send map-req>, state:incomplete, rlocs: 0).
LISP-0: EID-AF IPv4, Sending map-request from 10.9.9.9 to 10.9.9.9 for EID
10.9.9.9/32, ITR-RLOCs 1, nonce 0x6A7FF8D5-0x207A06EF (encap src 22.5.6.6,
dst 10.3.6.3), FromPITR.
```

When this map-request gets to the MR/MS (CSR3), it sees a valid EID-to-RLOC mapping and forwards this down to the ETR that registered the EID-to-RLOC mapping (CSR9).

```
! CSR3
LISP: Processing received Map-Request(1) message on GigabitEthernet2.536 from
10.9.9.9:4342 to 10.9.9.9:4342
```

```
LISP: Received map request for IID 101 10.9.9.9/32, source_eid IID 101
22.5.6.5, ITR-RLOCs: 22.5.6.6, records 1, nonce 0x6A7FF8D5-0x207A06EF,
FromPITR
LISP-0: MS EID IID 101 prefix 10.9.9.9/32 site CSR9, Forwarding map request
to ETR RLOC 10.7.9.9.
```

When this map-request gets to the ETR (CSR9), it validates that the EID was originated from itself and replies directly to the PITR (CSR6) to the map-request with a map-reply. This is what it would normally do for a regular ITR.

```
! CSR9
LISP: Processing received Map-Request(1) message on GigabitEthernet2.579 from
10.9.9.9:4342 to 10.9.9.9:4342
LISP: Received map request for IID 101 10.9.9.9/32, source_eid IID 101
22.5.6.5, ITR-RLOCs: 22.5.6.6, records 1, nonce 0x6A7FF8D5-0x207A06EF,
FromPITR
LISP: Processing map request record for EID prefix IID 101 10.9.9.9/32
LISP-1: Sending map-reply from 10.7.9.9 to 22.5.6.6
```

When the map-reply from the ETR (CSR9) reaches the PITR (CSR6), it performs the ordinary ITR duties of adding LISP encapsulation to traffic from CSR5 destined to LISP sites. Notice the time for this second map-request/map-reply process took about ~19 ms.

```
! CSR6, time 14:19:03.321
LISP: Processing received Map-Reply(2) message on GigabitEthernet2.561 from
10.7.9.9:4342 to 22.5.6.6:4342
LISP: Received map reply nonce 0x6A7FF8D5-0x207A06EF, records 1
LISP: Processing Map-Reply mapping record for IID 101 10.9.9.9/32, ttl 1440,
action none, authoritative, 1 locator
ei=10/10 LpR
LISP-0: Map Request IID 101 prefix 10.9.9.9/32 remote EID prefix[LL],
Received reply with rtt 19ms.
LISP: Processing mapping information for EID prefix IID 101 10.9.9.9/32
LISP-0: Remote EID IID 101 prefix 10.9.9.9/32, Change state to reused
(sources: <map-rep|send map-req>, state: incomplete, rlocs: 0).
LISP-0: AF IID 101 IPv4, Persistent db: ignore writing request, disabled.
LISP-0: Remote EID IID 101 prefix 10.9.9.9/32, Change state to complete
(sources: <map-rep|send map-req>, state: reused, rlocs: 0).
```

At this point, the PITR is performing ITR duties on behalf of the non-LISP site and the PETR is performing ETR duties on behalf of the LISP site. An outbound capture on CSR7's interface facing CSR9 shows the returning traffic flow. We already saw the original flow earlier when examining the PETR behavior. The inner packet is ICMP (0x01, protocol 1, green) from 22.5.6.5 to 10.9.9.9, where the source is in RLOC space and the destination is in EID space. The outer encapsulation has a UDP (0x11, protocol 17, green) header and sends traffic from 22.5.6.6 to 10.7.9.9 (yellow) using port 4341 (0x10F5, cyan). This source is

the P1TR address. CSR9 will perform local ETR duties and decapsulate these LISP packets, and then deliver them to the VRF C1 customer inside IID 101 (loopback1).

```
CSR7#show monitor capture CAP buffer dump
0
 0000: 000C29E0 4F84000C 29664C2C 81000DFB  ..).O...)fL,....
 0010: 08004500 00880034 4000FC11 4F161605  ..E....4@...O...
 0020: 06060A07 09091600 10F50074 00000800  .....t....
 0030: 00000000 65004500 0064009E 0000FE01  ....e.E..d.....
 0040: 8CDF1605 06050A09 09090000 69D10026  .....i..&
 0050: 00010000 00001508 074AABCD ABCDABCD  .....J.....
[snip]
```

Next, we examine LISP multicast. Only static RP is supported, and only PIM-SSM and PIM-ASM are supported for transport. CSR2 is the RP and also has an ASM receiver on group 225.1.1.1. CSR9 has an SSM receiver on group 232.1.1.1 from source 10.1.10.1 (CSR1). CSR1 is just a test host and has no interesting configuration. CSR2 and CSR9 have identical configurations except for the IGMP group joined. CSR2 requests ASM while CSR9 requests SSM from CSR1 explicitly. CSR10 just enables PIM on the LISP and LAN interfaces, along with the basic RP/SSM information.

```
! CSR2 and CSR9
interface Loopback1
 ip pim sparse-mode

interface LISP1.101
 ip pim sparse-mode

ip pim vrf C1 rp-address 10.2.2.2
ip pim vrf C1 ssm default

! CSR9 only
interface Loopback1
 ip igmp join-225.1.1.1

! CSR9 only
interface Loopback1
 ip igmp join-group 232.1.1.1 source 10.1.10.1

! CSR10
interface GigabitEthernet2.510
 ip pim sparse-mode

interface LISP1.101
 ip pim sparse-mode

ip pim vrf C1 rp-address 10.2.2.2
ip pim vrf C1 ssm default
```

Examining SSM first, we see that CSR9 has the (S,G) configured on its loopback and sends the (S,G) PIM join towards CSR10. The RPF interface on CSR9's (S,G) entry specifies that it should arrive via the LISP1.101 interface, with 101 representing the IID.

```
! CSR9
PIM(1): Insert (10.1.10.1,232.1.1.1) join in nbr 10.7.10.10's queue
PIM(1): Building Join/Prune packet for nbr 10.7.10.10
PIM(1): Adding v2 (10.1.10.1/32, 232.1.1.1), S-bit Join
PIM(1): Send v2 join/prune to 10.7.10.10 (LISP1.101)
```

```
CSR9#show ip mroute vrf C1 232.1.1.1 10.1.10.1 | begin \
(10.1.10.1, 232.1.1.1), 4d05h/stopped, flags: sLTI
Incoming interface: LISP1.101, RPF nbr 10.7.10.10
Outgoing interface list:
Loopback1, Forward/Sparse, 4d05h/00:00:49
```

The packet travels from CSR9's RLOC to CSR10's RLOC as a LISP unicast packet (yellow addresses, cyan port number). This means that CSR10 and CSR9 already mapped one another's EID prefixes to RLOCs for multicast to work. I'm not going to dig into all the details of the packet, but below is a PIM packet (protocol 0x67, 103, green) inside of LISP that contains 232.1.1.1 and 10.1.10.1 (pink) within the packet. I will assume this is the (S,G) join. LISP is intelligent enough to know the unicast RLOC destination for PIM joins based on the RPF interface. All we had to do was configure PIM on the LISP virtual interface, which makes LISP multicast very easy.

```
CSR7#show monitor capture CAP buffer dump
0
0000: 000C2966 4C2C000C 29E04F84 81000DFB ..)fL,..).O.....
0010: 08004500 0064491D 4000FF11 0B4B0A07 ..E..dI.@....K..
0020: 09090A07 0A0A840D 10F50050 00004800 .....P..H.
0030: 00000000 65014500 0040B5D4 0000FF67 ....e.E..@.....g
0040: 12630A09 0909E000 000D2300 A57C0101 .c.....#...|..
0050: 0A070A0A 05010146 05010A07 09090001 .....F.....
0060: 00D20100 0020E801 01010001 00000100 .....
0070: 04200A01 0A01
```

CSR10 receives the packet, performs the ETR functions to decapsulate it, and then processes the PIM join. CSR10 directs traffic out towards the LISP process using EID-to-RLOC mappings found inside IID 101. Because the PIM neighbor address is included with the OIL line entry, we know that the LISP virtual interface is operating in PIM NBMA mode despite it not being explicitly configured. This makes sense because, like DMVPN, LISP is treating each neighbor as a separate unicast peer.

```
! CSR10
PIM(3): Received v2 Join/Prune on LISP1.101 from 10.9.9.9, to us
PIM(3): Join-list: (10.1.10.1/32, 232.1.1.1), S-bit set
```

```
PIM(3): Update LISP1.101/10.7.9.9 to (10.1.10.1, 232.1.1.1), Forward state,
by PIM SG Join
```

All possible debugging has been turned off

```
CSR10#show ip mroute vrf C1 232.1.1.1 10.1.10.1 | begin \((
(10.1.10.1, 232.1.1.1), 4d05h/00:03:19, flags: sT
Incoming interface: GigabitEthernet2.510, RPF nbr 0.0.0.0
Outgoing interface list:
LISP1.101, 10.7.9.9, Forward/Sparse, 1d04h/00:03:19
```

When CSR1 starts sending multicast EID traffic, CSR10 encapsulates it inside unicast LISP packets and sends it towards the receiver, CSR9. We can see traffic being sent by looking at counters on both CSR9 and CSR10. A quick EPC verification outbound on CSR7 confirms this. The outer encapsulation has a UDP (0x11, protocol 17, green) header and sends traffic from 10.7.10.10 to 10.7.9.9 (yellow) using port 4341 (0x10F5, cyan) in RLOC space. The inner packet is ICMP (0x01, protocol 1, green) from 10.1.10.1 to 232.1.1.1 (yellow) in EID space. Like most tunneling techniques, LISP transports multicast inside unicast, which is inefficient.

```
CSR9#show ip mroute vrf C1 232.1.1.1 10.1.10.1 count | begin ^Group
Group: 232.1.1.1, Source count: 1, Packets forwarded: 17, Packets received:
17
Source: 10.1.10.1/32, Forwarding: 17/0/154/0, Other: 17/0/0
```

```
CSR10#show ip mroute vrf C1 232.1.1.1 10.1.10.1 count | begin ^Group
Group: 232.1.1.1, Source count: 1, Packets forwarded: 25, Packets received:
25
Source: 10.1.10.1/32, Forwarding: 25/0/118/0, Other: 25/0/0
```

```
CSR7#show monitor capture CAP buffer dump
0
0000: 000C29E0 4F84000C 29664C2C 81000DFB ..).O...)fL,....
0010: 08004500 00881946 4000FD11 3CFE0A07 ..E....F@...<...
0020: 0A0A0A07 0909EC00 10F50074 00004800 .....t...H.
0030: 00000000 65014500 00640091 0000FE01 ....e.E..d.....
0040: BF030A01 0A01E801 01010800 0A1E0013 .....
0050: 001D0000 00005CCB 1731ABCD ABCDABCD .....\\..1.....
[snip]
```

Last, we will examine ASM over LISP. CSR2 is both the RP and the receiver, so the (*,G) join doesn't transit the list network. CSR10 should send a register message to CSR2 when it receives traffic from CSR1; this message will have LISP and PIM encapsulation. Because this is an ordinary unicast packet, we won't capture it on EPC. Upon receipt, CSR2 generates an (S,G) join back towards the newly-discovered source.

```
! CSR10
PIM(3): Check RP 10.2.2.2 into the (*, 225.1.1.1) entry
```


PIM(3): Adding register encap tunnel (Tunnel0) as forwarding interface of (10.1.10.1, 225.1.1.1).

! CSR2

PIM(2): Received v2 Register on LISP1.101 from 10.1.10.10
for 10.1.10.1, group 225.1.1.1

PIM(2): Adding register decap tunnel (Tunnel1) as accepting interface of (10.1.10.1, 225.1.1.1).

PIM(2): Insert (10.1.10.1,225.1.1.1) join in nbr 10.7.10.10's queue

PIM(2): Building Join/Prune packet for nbr 10.7.10.10

PIM(2): Adding v2 (10.1.10.1/32, 225.1.1.1), S-bit Join

PIM(2): Send v2 join/prune to 10.7.10.10 (LISP1.101)

When CSR10 receives traffic down the shared tree, it sends a (*,G) prune to the RP for CSR1 as the specific source. It also receives the (S,G) join from CSR9 and updates it MRIB entry.

! CSR10

PIM(3): Building Triggered (*,G) Join / (S,G,RP-bit) Prune message for 225.1.1.1

PIM(3): Join-list: (10.1.10.1/32, 225.1.1.1), S-bit set

PIM(3): Add LISP1.101/10.2.8.2 to (10.1.10.1, 225.1.1.1), Forward state, by PIM SG Join

CSR10 continues to register the group until told to stop by the RP. Once the registration process is complete and the (S,G) tree is built, the PIM control-plane converges and native multicast flows without issue. The multicast traffic is still unicast-encapsulated via LISP (unicast), so it introduces transport inefficiencies. The PIM ASM source discovery and tree construction process is identical here as it is in a non-LISP network.

! CSR2

PIM(2): Received v2 Register on LISP1.101 from 10.1.10.10
for 10.1.10.1, group 225.1.1.1

PIM(2): Removing register decap tunnel (Tunnel1) as accepting interface of (10.1.10.1, 225.1.1.1).

PIM(2): Installing LISP1.101 as accepting interface for (10.1.10.1, 225.1.1.1).

PIM(2): Insert (10.1.10.1,225.1.1.1) join in nbr 10.7.10.10's queue

PIM(2): Building Join/Prune packet for nbr 10.7.10.10

PIM(2): Adding v2 (10.1.10.1/32, 225.1.1.1), S-bit Join

PIM(2): Send v2 join/prune to 10.7.10.10 (LISP1.101)

PIM(2): Send v2 Register-Stop to 10.1.10.10 for 10.1.10.1, group 225.1.1.1

! CSR10

PIM(3): Received v2 Register-Stop on LISP1.101 from 10.2.2.2

PIM(3): for source 10.1.10.1, group 225.1.1.1

PIM(3): Removing register encap tunnel (Tunnel0) as forwarding interface of (10.1.10.1, 225.1.1.1).

PIM(3): Clear Registering flag to 10.2.2.2 for (10.1.10.1/32, 225.1.1.1)

Quickly checking the MRIB counters on both routers confirms proper multicast operation.

```
CSR10#show ip mroute vrf C1 225.1.1.1 10.1.10.1 count | begin ^Group
Group: 225.1.1.1, Source count: 1, Packets forwarded: 90, Packets received:
90
    Source: 10.1.10.1/32, Forwarding: 90/0/117/0, Other: 90/0/0

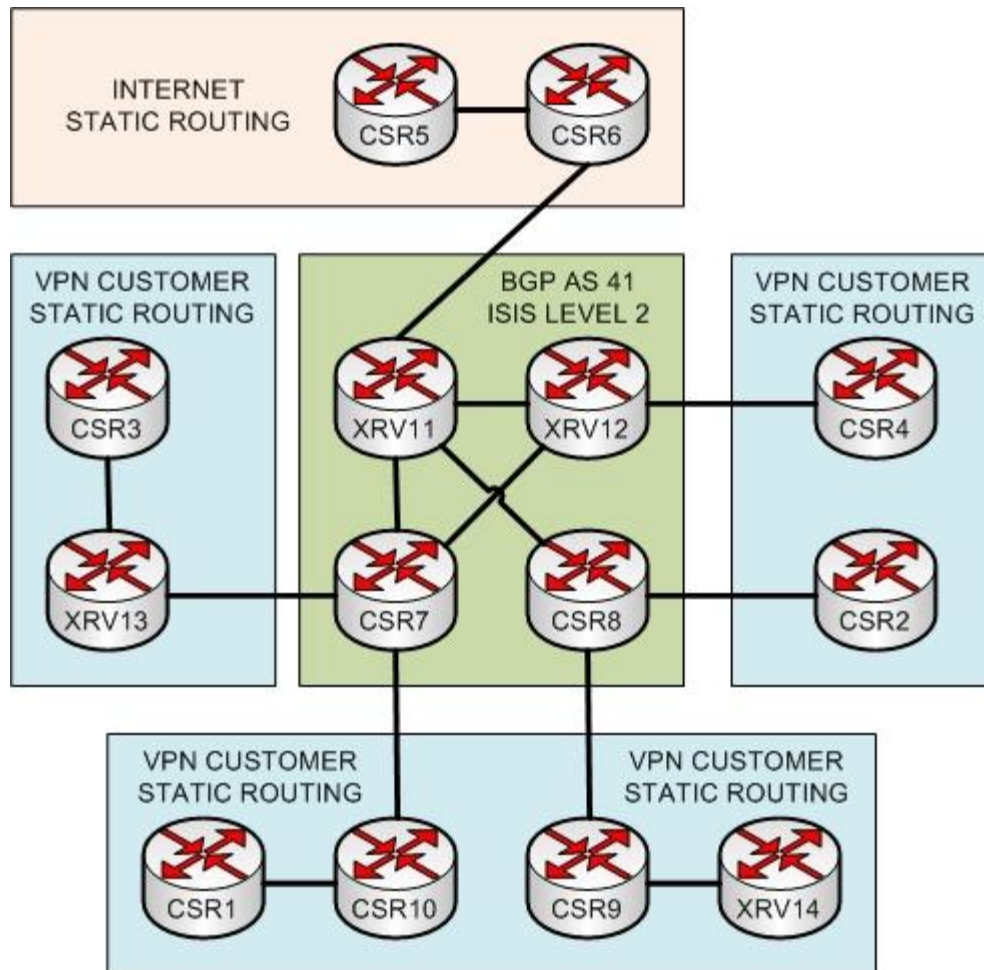
CSR2#show ip mroute vrf C1 225.1.1.1 10.1.10.1 count | begin ^Group
Group: 225.1.1.1, Source count: 1, Packets forwarded: 90, Packets received:
91
    Source: 10.1.10.1/32, Forwarding: 89/0/153/0, Other: 90/1/0
```

Additional Reading – Reference configurations "lisp"

35. Describe, implement, and troubleshoot GRE and mGRE-based VPN

35.1 P2P GRE tunneling and GRE features

GRE is a commonly used mechanism of tunneling various layer 3 protocols over IPv4 or IPv6. It is more flexible than IP-in-IP or IPv6-in-IP tunnels and can carry pretty much anything (IPv4, IPv6, IPX, CLNS, AppleTalk, etc). The basics of GRE configuration are not covered here, but the GRE header has many additional options which are worth evaluating. The current architecture is built on a hub-and-spoke architecture. CSR4 and XRV13 are hubs and CSR10, CSR9, and CSR2 are spokes. The hubs have individual P2P GRE tunnels to all spokes. Spoke to spoke traffic is routed through the hubs in this design. The collection of P2P tunnels means that we can test different features on different tunnels.



The relevant configurations are shown below on the hubs only. These are basic tunnels that require no explanation. The SP core MPLS L3VPN configuration, along with the customer IGP (OSPF) over tunnels, is not shown. These features are verified in detail later.

```
! XRV13
interface tunnel-ip113
  description P2P GRE XRV13 TO CSR10
  ipv4 unnumbered Loopback0
  tunnel source 10.11.13.13
  tunnel destination 10.7.10.10

interface tunnel-ip213
  description P2P GRE XRV13 TO CSR2
  ipv4 address 10.2.13.13 255.255.255.0
  tunnel source 10.11.13.13
  tunnel destination 10.8.2.2

interface tunnel-ip413
  description P2P GRE XRV13 TO CSR4
```

```
ipv4 address 10.4.13.13 255.255.255.0
tunnel source 10.11.13.13
tunnel destination 10.4.12.4
```

```
! CSR4
interface Tunnel24
description P2P GRE CSR4 TO CSR2
ip address 10.2.4.4 255.255.255.0
tunnel source 10.4.12.4
tunnel destination 10.8.2.2
```

```
interface Tunnel49
description P2P GRE CSR4 TO CSR9
ip address 10.4.9.4 255.255.255.0
tunnel source 10.4.12.4
tunnel destination 10.7.9.9
```

From a network perspective, each router has a default route towards the tunnel endpoints. This is the best match they have. For example, building CSR9's tunnel to CSR4 requires underlay connectivity to CSR4's tunnel source. This is ultimately achieved through the default route. Designs like this are valid since the private WAN traffic (OSPF routes) are routed through the tunnel, but all other "Internet" traffic is routed natively to the SP. This means that when routers need to reach CSR5, the Internet router, the traffic is never GRE-encapsulated. If the Internet access points aren't through the corporate WAN hub, this design is very efficient.

```
CSR9#show ip cef 10.4.12.4 detail
10.4.12.4/32, epoch 2, flags [default route]
  1 RR source [active source]
    Dependent covered prefix type rr, cover 0.0.0.0/0
    recursive via 0.0.0.0/0
      recursive via 10.7.9.7
        attached to GigabitEthernet2.579
```

```
CSR9#show ip cef 22.5.6.5 detail
0.0.0.0/0, epoch 2, flags [cover dependents, default route]
  Covered dependent prefixes: 2
    notify cover updated: 2
      recursive via 10.7.9.7
        attached to GigabitEthernet2.579
```

```
CSR9#show ip cef 4.4.4.4 detail
4.4.4.4/32, epoch 2
  nexthop 10.4.9.4 Tunnel49
```

Before looking at GRE features, we examine some basic techniques often used on GRE tunnels. IP unnumbered is often used on P2P links to conserve IP addressing since it is very rare that traffic is destined to these transit links. The tunnel between XRv13 and CSR10 shows this. The OSPF database's

point-to-point entry shows the interface number as being the router interface address. A common misconception is that the last octet of the “Link Data” comes from the SNMP ifmib; it is actually the software interface descriptor block (IDB). The output below illustrates this clearly, although it isn’t terribly relevant.

```
CSR10#show ip ospf 1 0 database router self-originate
```

```
OSPF Router with ID (10.10.10.10) (Process ID 1)
```

```
Router Link States (Area 0)
```

```
LS age: 29
Options: (No TOS-capability, DC)
LS Type: Router Links
Link State ID: 10.10.10.10
Advertising Router: 10.10.10.10
LS Seq Number: 80000006
Checksum: 0x1E48
Length: 60
Area Border Router
Number of Links: 3
```

```
Link connected to: another Router (point-to-point)
```

```
(Link ID) Neighboring Router ID: 13.13.13.13
```

```
(Link Data) Router Interface address: 0.0.0.24
```

```
Number of MTID metrics: 0
```

```
TOS 0 Metrics: 1000
```

```
[snip]
```

```
CSR10#show ifnum list | include Tu113
```

```
HWIDB    15      ACTIVE    Tu113          20000000 00000000 00000000
00060000 00000000 00000000 00000071 00000000
SWIDB    24      ACTIVE    Tu113          28000000 00000000 00000000
00060000 00000000 00000000 00000071 00000000
```

```
CSR10#show snmp mib ifmib ifindex Tunnel113
```

```
Interface = Tunnel113, Ifindex = 16
```

We see similar output on XRv13. This router is a hub so it has many more P2P tunnels than CSR10 (7 versus 3). I have not found a command to display the XRv IDB information, but I assume it works similarly on all Cisco platforms. The number 25 is likely allocated by this IDB process.

```
RP/0/0/CPU0:XRv13#show ospf 1 0 database router self-originate
```

```
OSPF Router with ID (13.13.13.13) (Process ID 1)
```

```
Router Link States (Area 0)
```

```
LS age: 172
Options: (No TOS-capability, DC)
LS Type: Router Links
Link State ID: 13.13.13.13
Advertising Router: 13.13.13.13
LS Seq Number: 80000009
Checksum: 0x99e
Length: 108
Area Border Router
Number of Links: 7
```

```
Link connected to: another Router (point-to-point)
(Link ID) Neighboring Router ID: 10.10.10.10
(Link Data) Router Interface address: 0.0.0.25
Number of TOS metrics: 0
TOS 0 Metrics: 1000
```

[snip]

Another approach specific to IS-IS and OSPF is to strip the IP routing component for that subnet from the link-state information. IS-IS does this by explicitly advertising only passive interface IP prefixes in its LSPs. OSPF removes the stub link-state information from its LSA-1 to achieve the same effect. XRv3 only appears to support OSPFv2 prefix-suppression for secondary addresses, so we will demonstrate this on XE routers. The tunnel between CSR4 and CSR2 is removed from the LSA-1, so no one will have an OSPF-learned route for 10.2.4.0/24.

```
! CSR2 and CSR4
interface Tunnel24
 ip ospf prefix-suppression
```

Looking at CSR10, it has no route for this prefix and when examining the LSA-1s from CSR2 and CSR4, the prefix is not listed as a stub network. CSR2 and CSR4 still have connected routes, of course, which is necessary for route recursion. This feature should be mutually exclusive with IP unnumbered; unnumbered links have no stub network to suppress in the first place. The output is long so we can be thorough in verifying that the stub networks were properly suppressed. Even outside of GRE, this feature is valuable on any transit links where there are no end hosts.

```
CSR10#show ip route 10.2.4.0
% Subnet not in table
```

```
CSR10#show ip ospf 1 0 database router 2.2.2.2
```

```
OSPF Router with ID (10.10.10.10) (Process ID 1)
```

```
Router Link States (Area 0)
```

```
LS age: 52
```

Options: (No TOS-capability, DC)
LS Type: Router Links
Link State ID: 2.2.2.2
Advertising Router: 2.2.2.2
LS Seq Number: 80000007
Checksum: 0xDFB9
Length: 72
Number of Links: 4

Link connected to: a Stub Network
(Link ID) Network/subnet number: 2.2.2.2
(Link Data) Network Mask: 255.255.255.255
Number of MTID metrics: 0
TOS 0 Metrics: 1

Link connected to: another Router (point-to-point)
(Link ID) Neighboring Router ID: 13.13.13.13
(Link Data) Router Interface address: 10.2.13.2
Number of MTID metrics: 0
TOS 0 Metrics: 1000

Link connected to: a Stub Network
(Link ID) Network/subnet number: 10.2.13.0
(Link Data) Network Mask: 255.255.255.0
Number of MTID metrics: 0
TOS 0 Metrics: 1000

Link connected to: another Router (point-to-point)
(Link ID) Neighboring Router ID: 4.4.4.4
(Link Data) Router Interface address: 10.2.4.2
Number of MTID metrics: 0
TOS 0 Metrics: 1000

CSR10#show ip ospf 1 0 database router 4.4.4.4

OSPF Router with ID (10.10.10.10) (Process ID 1)

Router Link States (Area 0)

LS age: 67
Options: (No TOS-capability, DC)
LS Type: Router Links
Link State ID: 4.4.4.4
Advertising Router: 4.4.4.4
LS Seq Number: 8000000F
Checksum: 0xBF3B
Length: 108
Number of Links: 7

Link connected to: another Router (point-to-point)
(Link ID) Neighboring Router ID: 10.9.9.9
(Link Data) Router Interface address: 10.4.9.4
Number of MTID metrics: 0
TOS 0 Metrics: 1000

Link connected to: a Stub Network
(Link ID) Network/subnet number: 10.4.9.0
(Link Data) Network Mask: 255.255.255.0
Number of MTID metrics: 0
TOS 0 Metrics: 1000

Link connected to: another Router (point-to-point)
(Link ID) Neighboring Router ID: 10.10.10.10
(Link Data) Router Interface address: 10.4.10.4
Number of MTID metrics: 0
TOS 0 Metrics: 1000

Link connected to: a Stub Network
(Link ID) Network/subnet number: 10.4.10.0
(Link Data) Network Mask: 255.255.255.0
Number of MTID metrics: 0
TOS 0 Metrics: 1000

Link connected to: another Router (point-to-point)
(Link ID) Neighboring Router ID: 13.13.13.13
(Link Data) Router Interface address: 10.4.13.4
Number of MTID metrics: 0
TOS 0 Metrics: 1000

Link connected to: a Stub Network
(Link ID) Network/subnet number: 10.4.13.0
(Link Data) Network Mask: 255.255.255.0
Number of MTID metrics: 0
TOS 0 Metrics: 1000

Link connected to: another Router (point-to-point)
(Link ID) Neighboring Router ID: 2.2.2.2
(Link Data) Router Interface address: 10.2.4.4
Number of MTID metrics: 0
TOS 0 Metrics: 1000

Tunnels can also exist in different VRFs than their endpoints. This allows a tunnel to have a front-door VRF (FVRF) for its endpoints but still run IGP in the global table. CSR2 uses this technique; this is a great way to ensure there are no tunnel recursion errors as the tunnel endpoints and the tunnel itself are in different VRFs. Be sure to update the underlay routing to be VRF-aware; in this case, we update the static route towards the tunnel endpoint.


```

! CSR2
vrf definition FVRF
  address-family ipv4

interface Tunnel24
  tunnel vrf FVRF

ip route vrf FVRF 10.0.0.0 255.0.0.0 10.8.2.8

```

We can verify this using a simple show command as seen below.

```

CSR2#show ip interface tunnel24 | include VRF
Tunnel VPN Routing/Forwarding "FVRF"

```

The smallest GRE header, not counting the IP encapsulation component, is 4 bytes long. The first 4 bits can tell how long the header will be; 1 of the bits is unused, but the other 3 represent additional 4-byte features. For example, if 2 of the bits are set, the GRE header will be 12 bytes long (original 4 plus 8 more). The header also carries the protocol type (like an ethertype) along with the version. This allows GRE to carry multiple protocols. As an example, the tunnel between CSR2 and XRv3 has IPv6 enabled with LL addressing (not shown). Capturing inbound on CSR8, we see IPv4 packets with GRE-type 0800 and IPv6 packets with GRE-type 0x86DD. The IPv4 packet happens to be an OSPFv2 hello and the IPv6 packet is IPv6-ICMP, but that is beyond the scope of this test. The first 16 bits (4 hex digits) are all zero, indicating no special GRE extensions are in use. Simply being able to carry a protocol type in the GRE header is what gives GRE its flexibility.

```

CSR8#show monitor capture CAP buffer detailed

```

```

-----
#    size    timestamp      source                destination    protocol
-----
  0    122      0.000000    10.8.2.2             -> 10.11.13.13    GRE
0000:  000C296A  FFD5000C  295CE1E9  81000DC0    ..)j....)\.....
0010:  080045C0  006805A5  0000FF2F  91E00A08    ..E..h...../....
0020:  02020A0B  0D0D0000  080045C0  0050AA3A    .....E..P.:
0030:  00000159  17520A02  0D02E000  00050201    ...Y.R.....

  1    142      0.221988    10.8.2.2             -> 10.11.13.13    GRE
0000:  000C296A  FFD5000C  295CE1E9  81000DC0    ..)j....)\.....
0010:  08004500  007C05A6  0000FF2F  928B0A08    ..E..|...../....
0020:  02020A0B  0D0D0000  86DD6000  0000003C    .....`.....<
0030:  3A40FE80  00000000  00000000  00000000    :@.....

```

GRE has many interesting features beyond basic tunneling. First, we will examine the GRE tunnel key. This is a 32-bit unsigned integer that differentiates tunnels with the same source and destination, and is commonly used in networks with many overlays and a single underlay. It is also a form of weak security used to prevent against misconfigurations or from accepting stray packets. To make it easy to see this in

the packet capture, we will use the decimal key 3203386110 (0xBEEFCAFE) on the tunnel between CSR2 and XRv13.

```
! CSR2
interface Tunnel213
 tunnel key 3203386110

! XRv13
interface tunnel-ip213
 tunnel key 3203386110
```

If the tunnel keys do not match, "debug tunnel keepalive" will reveal the fault. To test it, I temporarily change one of the keys to 11111 (not shown) to ensure the key is being checked.

```
! CSR2
Tunnel: Drop, Failed to find tunnel for GRE/IPv4 packet 10.11.13.13->10.8.2.2
(tbl=4, "FVRF" len=104, ttl=252)
```

After capturing on CSR8, we can see the packet is 4 bytes larger (122 to 126) and contains our new 4-byte key. The original 4 byte header is shown in yellow with the new GRE tunnel key in green. The beginning of the header changed also; the first hex digit is 0010 in binary, and the 1 represents the flag indicating that the tunnel key is present.

CSR8#show monitor capture CAP buffer detailed

#	size	timestamp	source	destination	protocol
0	126	0.000000	10.8.2.2	-> 10.11.13.13	GRE
0000:	000C296A	FFD5000C	295CE1E9	81000DC0	..)j....)\.....
0010:	080045C0	006C060C	0000FF2F	91750A08	..E..l...../..u..
0020:	02020A0B	0D0D2000	0800BEEF	CAFE45C0	E.
0030:	0050AB62	00000159	162A0A02	0D02E000	.P.b...Y.*.....

Next, we enable the GRE checksum feature. This is a 16-bit field, where the low-order 16-bits are always set to 0. This used for validating the packet for corruption. This is not supported on XR, but can be enabled unidirectionally.

```
! CSR2
interface Tunnel213
 tunnel checksum
```

Notice that packet has grown by another 4 bytes to 130. The checksum is 0xCE10 and the next 16 bits are set to 0 (cyan). The GRE key follows that. Notice the very first hex digit of the GRE header is 0xA, which is 1010. We conclude that the leftmost bit represents the checksum option. Note: The rightmost

bit represents the sequence numbers field. GRE sequence numbers, the third GRE option, do not appear to be supported on XE or XR, so they are not tested.

```
CSR8#show monitor capture CAP buffer detailed
 1 130 23.000000 10.8.2.2 -> 10.11.13.13 GRE
0000: 000C296A FFD5000C 295CE1E9 81000DC0 ..)j....)\.....
0010: 080045C0 0070069F 0000FF2F 90DE0A08 ..E..p...../....
0020: 02020A0B 0D0DA000 0800CE10 0000BEEF .....
0030: CAFE45C0 0050ACE7 00000159 14A50A02 ..E..P.....Y....
```

Although not a header extension, GRE can also issue regular keepalives. This is supported on P2P tunnels only, but it's essentially an empty GRE packet. We enable this on the CSR2-XRv13 tunnel using the basic "keepalive" command.

```
! CSR2
interface Tunnel213
 keepalive 3 9

! XRv13
interface tunnel-ip213
 keepalive 3 9
```

Keepalives can be enabled unidirectionally since it works like BFD echo-mode. Inside the GRE packet from CSR2 to XRv13, there is another GRE packet which has the source/destination headers reversed. This allows XRv13 to respond normally. Notice the GRE header at the very end of the packet. The protocol number is 0x0000 and the reserved field that was previously 0x0000 is now 0x0001. I assume this low-level detail signals a keepalive to the GRE process, although technically it is part of the sequence number field. The debug below also shows the keepalive being sent and received, but notice the direction is represented by the inner packet (from the destination to the source). I used the same colors as above to show the GRE basic header (yellow), checksum (cyan), and key (green). I use grey to show the inner GRE packet with reversed IP addresses.

```
CSR8#show monitor capture CAP buffer dump
0
0000: 000C296A FFD5000C 295CE1E9 81000DC0 ..)j....)\.....
0010: 080045C0 00400707 0000FF2F 90A60A08 ..E..@...../....
0020: 02020A0B 0D0DA000 0800CE10 0000BEEF .....
0030: CAFE45C0 0020000F 0000FF2F 97BE0A0B ..E.. ...../....
0040: 0D0D0A08 0202A000 0000D60F 0001BEEF .....
0050: CAFE
```

```
Tunnel213: sending keepalive, 10.11.13.13->10.8.2.2 (len=32 ttl=255),
counter=1
Tunnel213: keepalive received, 10.11.13.13->10.8.2.2 (len=32 ttl=251),
resetting counterund all
```

When we enable it on XRv13, there does not appear to be a debug that shows the keepalives. However, outbound EPC on CSR8 reveals those packets. Notice that XRv13 is not using checksums, which explains the different in length (74 versus 82, account for two checksums).

```
CSR8#show monitor capture CAP buffer dump
5
0000: 000C295C E1E9000C 296AFFD5 81000DC0 ..)\....)j.....
0010: 080045C0 00380000 4000FD2F 59B50A0B ..E..8..@../Y...
0020: 0D0D0A08 02022000 0800BEEF CAFE45C0 .....E.
0030: 001C0000 0000FF2F 97D10A08 02020A0B ...../.....
0040: 0D0D2000 0000BEEF CAFE .. .....
```

You can see a quick summary of the GRE special features using the command below.

```
CSR2#show interfaces tunnel213 | section GRE|Keepalive
Description: P2P GRE CSR2 TO XRv13
Keepalive set (3 sec), retries 9
Tunnel protocol/transport GRE/IP
Key 0xBEEFCAFE, sequencing disabled
Checksumming of packets enabled

RP/0/0/CPU0:XRv13#show interfaces tunnel-ip213 | include is enabled
Keepalive is enabled, interval 3 seconds, maximum retry 9
Key is enabled, key value 3203386110
```

The tunnel TOS and TTL parameters are straightforward and self-explanatory. These set the TOS and TTL of the outer IP header only. We demonstrate this on CSR9 sending traffic to CSR4 and capturing inbound on CSR7. CSR9 will set the outer IP header TOS to DSCP CS3 rather than reflect the tunneled value. The tunnels default TTL is 255 but we can shorten the “length” of the tunnel by selecting a smaller value.

```
! CSR9
interface Tunnel49
 tunnel tos 96
 tunnel ttl 8
```

The OSPF hellos have DSCP CS6 set, but the tunnel overrides it to DSCP CS3. The tunnel TTL has been reduced to 8 for additional security. Comparing it to the packets going to XRv3 (a different and unmodified tunnel), the differences become clear. The TTLs are highlighted in yellow and the TOS values in green. Failing to specify a tunnel TOS means that the value is copied from the inner IP header; the packets going to XRv3 have DSCP CS6 which was copied from the OSPFv2 hellos inside. Failing to specify a TTL means that a value of 255 is always used.

```
CSR7#show monitor capture CAP buffer detail
0 122 0.000000 10.7.9.9 -> 10.11.13.13 GRE
0000: 000C2966 4C2C000C 29E04F84 81000DFB ..)fL,..).O.....
```

```

0010: 080045C0 0068087B 0000FF2F 88040A07  ..E..h.{.../....
0020: 09090A0B 0D0D0000 080045C0 0050A615  .....E..P..
0030: 00000159 1B690A09 0D09E000 00050201  ...Y.i.....

  1 122      4.775021    10.7.9.9      -> 10.4.12.4      GRE
0000: 000C2966 4C2C000C 29E04F84 81000DFB  ..)fL,..).O.....
0010: 08004560 0068087C 0000082F 80740A07  ..E`.h.|.../.t..
0020: 09090A04 0C040000 080045C0 0050A616  .....E..P..
0030: 00000159 1F6D0A04 0909E000 00050201  ...Y.m.....

```

The “flow egress-records” command appears to do nothing. Whether it is enabled or not, egress Netflow records are always captured. FNF is enabled on CSR10 to show this.

```

! CSR10
flow monitor FLOW_MON_IPV4
record netflow-original

interface Tunnel410
ip flow monitor FLOW_MON_IPV4 output
tunnel flow egress-records

CSR10#show interfaces tunnel410 | include flow
Egress flow records enabled

CSR10#show flow monitor FLOW_MON_IPV4 cache
[snip]

IPV4 SOURCE ADDRESS:      10.1.10.1
IPV4 DESTINATION ADDRESS: 4.4.4.4
TRNS SOURCE PORT:        0
TRNS DESTINATION PORT:    2048
INTERFACE INPUT:         Gi2.510
FLOW SAMPLER ID:         0
IP TOS:                   0x00
IP PROTOCOL:              1
[snip]

```

When the command is removed and the cache is cleared, we see the same effect. I also tried putting it inbound only on the LAN to see if I would get duplicate records for outbound packets, but I did not. We will stop investigating the feature as it appears to have limited utility.

```

CSR10#show interfaces tunnel410 | include flow
[no output]

CSR10#show flow monitor FLOW_MON_IPV4 cache
[snip]

IPV4 SOURCE ADDRESS:      10.1.10.1

```

```
IPV4 DESTINATION ADDRESS: 4.4.4.4
TRNS SOURCE PORT: 0
TRNS DESTINATION PORT: 2048
INTERFACE INPUT: Gi2.510
FLOW SAMPLER ID: 0
IP TOS: 0x00
IP PROTOCOL: 1
[snip]
```

The tunnel bandwidth feature is another command on a GRE tunnel of limited value. I believe it is informational only and is meant to be used for Layer1 transports that don't have the same bandwidth in both directions, like satellite links using Rate-Based Satellite Control Protocol (RBSCP) tunnels. I assume this information is communicated to upper-layer protocols as it is not a policer for the tunnel. The default is 8000 kbps and this value also doesn't seem to affect IGP cost calculation. This feature is probably not relevant for GRE tunnels at all, but we will investigate it briefly.

```
CSR9#show interfaces tunnel913 | include bandwidth
Tunnel transmit bandwidth 8000 (kbps)
Tunnel receive bandwidth 8000 (kbps)

CSR9#show ip ospf interface brief | include Tu913
Tu913      1      0      10.9.13.9/24      1000      P2P      1/1
```

We modify the tunnel on CSR9 going to XRv3 to be TX 4 Mbps and RX 1 Mbps. There is no change in the IGP metric.

```
! CSR9
interface Tunnel913
 tunnel bandwidth transmit 4096
 tunnel bandwidth receive 1024

CSR9#show interfaces tunnel913 | include bandwidth
Tunnel transmit bandwidth 4096 (kbps)
Tunnel receive bandwidth 1024 (kbps)

CSR9#show ip ospf interface brief | include Tu913
Tu913      1      0      10.9.13.9/24      1000      P2P      1/1
```

Only the traditional "bandwidth" command has an effect on the IGP metric. There is also a "bandwidth receive" command. Having 4 different bandwidth commands on a single interface can be very confusing. I've set the bandwidth to 16 Mbps and the RX BW to 2048, making all 4 values are different.

```
! CSR9
interface Tunnel913
 bandwidth 16384
 bandwidth receive 2048
```

The 16 Mbps value is used for the OSPF cost calculation now. The "tunnel bandwidth receive" is deprecated according to Cisco documentation as this was once used for RBSCP. RBSCP is beyond the scope of this document, but only the "transmit" command is used now.

```
CSR9#show interfaces tunnel 913 | include bandwidth|BW
MTU 9976 bytes, BW 16384 Kbit/sec, RxBW 2048 Kbit/sec, DLY 50000 usec,
Tunnel transmit bandwidth 4096 (kbps)
Tunnel receive bandwidth 1024 (kbps)
```

```
CSR9#show ip ospf interface brief | include Tu913
Tu913      1      0      10.9.13.9/24      6      P2P      1/1
```

Tunnel entropy uses a hash value for the lower-order 8 bits of the tunnel key to introduce more randomness into the packet header. It is a 6-tuple combining src/dst IP, src/dst port, VRF, and IP protocol. The goal of this is better load sharing assuming routers in the core do their ECMP-hash-based forwarding using the tunnel key. This also means the GRE key must be 24 bits or less, so using 0xBEEFCAFE again won't work. We will enable this on the CSR2-CSR4 tunnel, and we must enable entropy before adding the key.

```
CSR2(config-if)#tunnel key ?
<0-4294967295> key
```

When entropy is enabled, the key can only be 24 bits. Using a small key 0xBEEF (48879) will work. We configure this on both ends of the tunnel then verify it was successful. Both ends must have a matching key with the entropy option enabled. To enable entropy, you must configure entropy first and then the tunnel key; order of operations is significant.

```
! CSR2 and CSR4
interface Tunnel24
 tunnel entropy
 tunnel key 48879
```

```
CSR2(config-if)#tunnel key ?
<0-16777215> key
```

```
CSR2#show interfaces tunnel24 | include Key
Key 0xBEEF, sequencing disabled
Tunnel Entropy Calculation Enabled (24-bit Key)
```

```
CSR4#show interfaces tunnel24 | include Key
Key 0xBEEF, sequencing disabled
Tunnel Entropy Calculation Enabled (24-bit Key)
```

Looking into the packet by capturing outbound on CSR8, we can see the lower-order 8 bits of the key are no longer part of the "key" used for security purposes between the endpoints. The top flow is a telnet session and has entry entropy value 0xBE, and the bottom flow is an ICMP flow with entropy value 0x7C. The protocol numbers 6 and 1 for TCP and ICMP, respectively, are highlighted in pink. Because every flow is likely to have a different GRE key (256 total combinations), if the underlying architecture can do hashing based on the GRE key rather than the src/dst IP information, better load sharing can be achieved. I cannot find an authoritative source that tells me whether XE/XR routers can do load sharing based on the GRE tunnel key, nor can I find a mechanism to configure it. The concept is very clever and could be very useful for load sharing in an IP-based core network.

```
CSR8#show monitor capture CAP buffer detailed
 3  88    0.558021  10.4.12.4      -> 10.8.2.2      GRE
0000:  000C295C E1E9000C 296AFFD5 81000DC0  ..)\....)j.....
0010:  080045C0 0046196C 0000FD2F 814B0A04  ..E..F.l.../.K..
0020:  0C040A08 02022000 080000BE EFBE45C0  .....E.
0030:  002A5977 0000FF06 458D0A02 04040A02  .*Yw....E.....

 4 146    3.018004  10.4.12.4      -> 10.8.2.2      GRE
0000:  000C295C E1E9000C 296AFFD5 81000DC0  ..)\....)j.....
0010:  08004500 008018FC 0000FD2F 82410A04  ..E...../.A..
0020:  0C040A08 02022000 080000BE EF7C4500  .....|E.
0030:  00640017 0000FF01 9F780A02 04040A02  .d.....x.....
```

Additional Reading – Reference configurations "gre-p2p"

35.2 Dynamic Multipoint VPN (DMVPN) basics

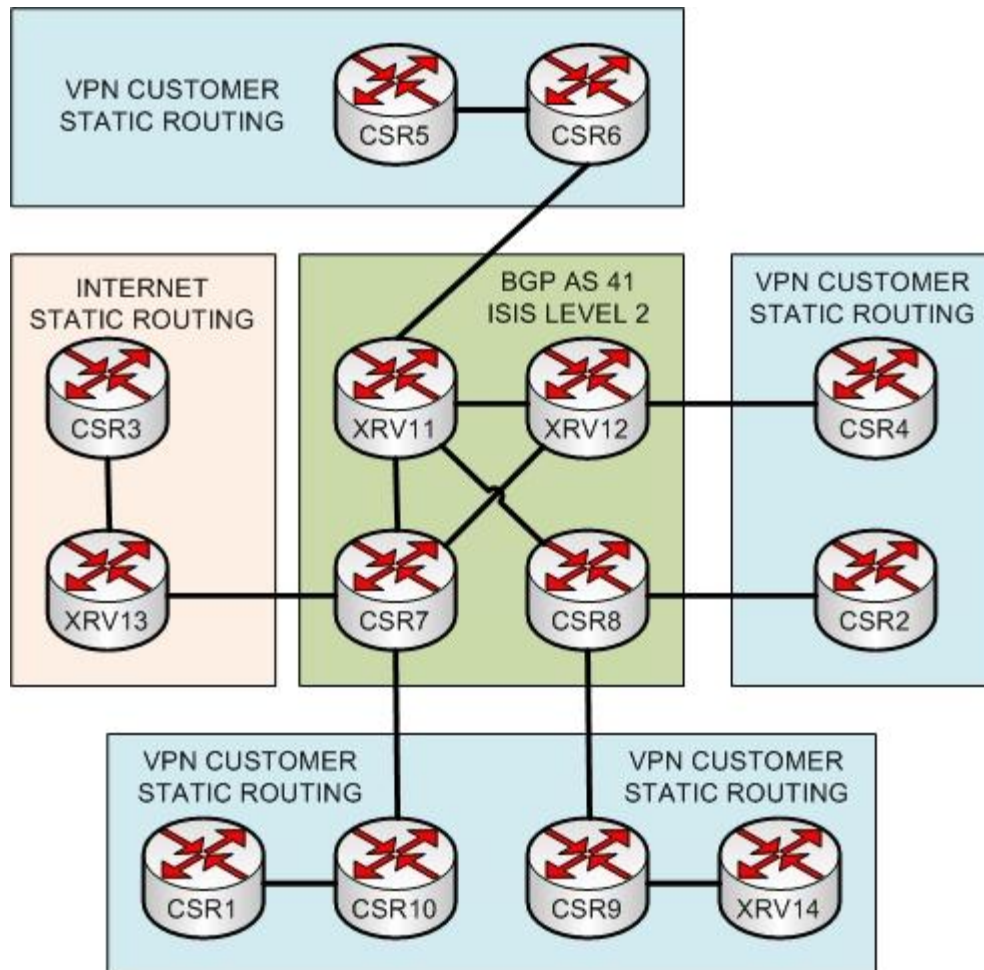
DMVPN isn't really an SP technology but is very commonly used in enterprise networks to connect branch sites to corporate HQ sites in hub-spoke fashion using multipoint GRE (mGRE) tunnel interfaces. Since mGRE is within the blueprint, we examine DMVPN also; understanding what enterprise customers are doing over the SP network is valuable for customer interactions and joint troubleshooting. Encryption via IPSec can optionally be added, but this is beyond the scope of the SP blueprint. Spokes register to the hubs using Next Hop Resolution Protocol (NHRP) so that hubs can dynamically discover hubs, which simplifies provisioning for new spokes. The registration message contains a mapping between the VPN addresses (the typically-private tunnel IP) and the NBMA addresses (the typically-public address, AKA the tunnel endpoints). The process is somewhat similar to LISP where ETRs register EID-to-RLOC mappings to the MS. In most deployments the hubs dynamically create downstream multicast mappings based on spoke registrations. DMVPN comes in three variations:

1. **Phase 1:** Still takes advantage of mGRE tunnels, but does not provide any direct spoke-to-spoke communication. All spoke-to-spoke traffic is hair-pinned through the hub. In single-hub deployments, spokes can use a P2P GRE tunnel construct with some additional configuration for simplicity. In dual-hub deployments, spokes can use the mGRE tunnel construct similar to the hubs. Phase 1 is really only recommended for small, single-hub deployments with little to no spoke-to-spoke traffic.

2. **Phase 2:** Uses mGRE tunnel interfaces everywhere as it provides dynamic spoke-to-spoke tunnel creation. When a spoke needs to reach a remote spoke, it queries the hub requesting the NBMA (underlay) address for a given VPN (overlay) address. The hub resolves the remote spoke's NBMA address (processed detailed later) and the local spoke can directly encapsulate traffic with a GRE destination of the remote spoke's NBMA address. So long as the underlay is intelligently designed to provide spoke-to-spoke access (such as the Internet or SP WAN links), this technology works well.

3. **Phase 3:** Similar to phase 2 but it enhances scalability. Using new NHRP redirect messages from the hub and CEF shortcuts on the spokes, a hierarchical hub architecture can be designed to further regionalize the network. All of the routers still share a tunnel mesh (and thus an IP subnet), but the IGP neighbor discovery process can be bounded by the hub hierarchy. It also allows the hub routers to advertise very coarse aggregate routes, like defaults, and not provide more specific matches. This is not possible in Phase 2 since the spokes still need to know the actual VPN next-hops for every destination, even other spokes.

The diagram below shows a network logically similar to the P2P GRE architecture. The main difference from the P2P GRE setup is that XRv3 is no longer a "hub" since it does not support DMVPN at all. It will be the "Internet" router and CSR5 will be a hub, alongside CSR4. CSR10, CSR9, and CSR2 remain spokes. This lab does not detail advanced DMVPN designs with respect to IGP, multicast, etc. The focus is on basic DMVPN constructs, detailed debugging, and control-plane verification.



General restrictions:

- GRE keepalives only work on tunnels that have explicit destinations configured. Even on mGRE spokes with NHRP static mappings, GRE keepalives are not sent. Recall that keepalives can be configured unidirectionally, so a Phase 1 spoke in a single-hub deployment, for example, can use GRE keepalives to probe the hub for reachability.

- Multicast NBMA designs apply. For Phase 1, both PIM (*,G) and (S,G) joins can be enabled since the next-hop for all routes is the hub. For Phase 2, only PIM (*,G) joins are valid since the RPF neighbor will appear spoke-to-spoke (directly connected), which is not supported for LL multicast. There are no PIM neighbors laterally across any flavor of DMVPN. For Phase 3, (S,G) joins can work, but only if the NHO process is in effect. NHRP (H) routes are used for RPF and break multicast when they exist. These H routes are common when performing summarization on spokes.

35.2.1 Phase 1

In this setup, CSR4/CSR5 are DMVPN phase 1 hubs. CSR9 and CSR10 use mGRE interfaces as spokes to connect to both hubs. They use static mappings towards each hub for unicast and multicast. CSR2 demonstrates using a P2P tunnel to CSR4, which would be common in a single-hub deployment. To

connect to CSR5, both CSR2 and CSR5 must provision a new, separate P2P GRE tunnel since CSR2 cannot use the same subnet on multiple interfaces in the same VRF. For phase 1, this doesn't affect forwarding, but complicates the design. The OSPF network type should be P2MP on the hubs and either P2MP or P2P on the spokes depending on the design.

```
! Generic hub (CSR4)
interface Tunnel100
  description DMVPN PHASE 1 HUB
  ip address 10.0.100.4 255.255.255.0
  no ip redirects
  ip nhrp authentication NHRPAUTH
  ip nhrp map multicast dynamic
  ip nhrp network-id 100
  ip ospf network point-to-multipoint
  ip ospf hello-interval 10
  ip ospf 1 area 0
  tunnel source 10.4.12.4
  tunnel mode gre multipoint
```

```
! Generic mGRE spoke (CSR9)
interface Tunnel100
  description DMVPN PHASE 1 - SPOKE
  ip address 10.0.100.9 255.255.255.0
  no ip redirects
  ip nhrp authentication NHRPAUTH
  ip nhrp map 10.0.100.4 10.4.12.4
  ip nhrp map multicast 10.4.12.4
  ip nhrp map 10.0.100.5 10.5.6.5
  ip nhrp map multicast 10.5.6.5
  ip nhrp network-id 100
  ip nhrp nhs 10.0.100.4
  ip nhrp nhs 10.0.100.5
  ip ospf network point-to-multipoint
  ip ospf hello-interval 10
  ip ospf 1 area 0
  tunnel source 10.7.9.9
  tunnel mode gre multipoint
```

```
! P2P GRE spoke (CSR2)
interface Tunnel100
  description DMVPN PHASE 1 - HUB CSR4
  ip address 10.0.100.2 255.255.255.0
  ip nhrp authentication NHRPAUTH
  ip nhrp map 10.0.100.4 10.4.12.4
  ip nhrp map multicast 10.4.12.4
  ip nhrp network-id 100
  ip nhrp nhs 10.0.100.4
  ip ospf network point-to-point
```

```

ip ospf 1 area 0
keepalive 3 3
tunnel source 10.8.2.2
tunnel destination 10.4.12.4
tunnel vrf FVRF

```

We will watch the NHRP registration process in detail from the perspective of a spoke (CSR9) and a hub (CSR5). The debugs are verbose so I have stripped out the unnecessary parts. First, CSR9's tunnel comes up and adds the static mappings to the NHRP cache. Technically, those static mappings count as NHRP entries also. The NHRP process iterates over the configured hub mappings in series. I also show the NHRP cache at present to show the mappings were added. The command used is "debug nhrp".

```

! CSR9
NHRP: if up: Tunnel100 proto 'NHRP_IPv4'
NHRP: Adding all static maps to cache

NHRP: Adding multicast map entry to multicast list10.5.6.5
NHRP: Adding Tunnel Endpoints (VPN: 10.0.100.5, NBMA: 10.5.6.5)
NHRP: Successfully attached NHRP subblock for Tunnel Endpoints (VPN:
10.0.100.5, NBMA: 10.5.6.5)
NHRP: No peer data updated in NHRP subblock for Tunnel Endpoints (VPN:
10.0.100.5, NBMA: 10.5.6.5)

NHRP: Adding multicast map entry to multicast list10.4.12.4
NHRP: Adding Tunnel Endpoints (VPN: 10.0.100.4, NBMA: 10.4.12.4)
NHRP: Successfully attached NHRP subblock for Tunnel Endpoints (VPN:
10.0.100.4, NBMA: 10.4.12.4)
NHRP: No peer data updated in NHRP subblock for Tunnel Endpoints (VPN:
10.0.100.4, NBMA: 10.4.12.4)

CSR9#show ip nhrp brief
      Target          Via          NBMA          Mode    Intfc    Claimed
      10.0.100.4/32  10.0.100.4      10.4.12.4      static    Tu100    < >
      10.0.100.5/32  10.0.100.5      10.5.6.5       static    Tu100    < >

CSR9#show ip nhrp multicast
      I/F      NBMA address
Tunnel100    10.4.12.4      Flags: static      (Enabled)
Tunnel100    10.5.6.5       Flags: static      (Enabled)

```

Next, CSR9 will attempt to register its VPN/NBMA mappings to each NHS in series. Priority values can determine whether NHS' are more preferred than others (0 is the highest priority) according to Cisco documentation, but this is used in specific HA architectures. NHRP "clusters" can be used for failover scenarios; these features are not examined in great detail here. Below, the debugs from CSR9 show the registration packets being sent, and the replies being received from each NHS. The transition from 'E' to 'RE' means that the NHS entry went from "expecting replies" to "expecting replies and receiving them". This means registration was successful and each NHS is marked as "up".

```

! CSR9
NHRP: Multicast enabled for dst 10.4.12.4
NHRP: NHS 10.0.100.4 Tunnel100 vrf 0 Cluster 0 Priority 0 Transitioned to 'E'
from ' '
NHRP: Attempting to send packet through interface Tunnel100 via DEST dst
10.0.100.4
NHRP: Send Registration Request via Tunnel100 vrf 0, packet size: 108
src: 10.0.100.9, dst: 10.0.100.4
NHRP: Encapsulation succeeded. Sending NHRP Control Packet NBMA Address:
10.4.12.4
NHRP: 132 bytes out Tunnel100
NHRP: Receive Registration Reply via Tunnel100 vrf 0, packet size: 128
NHRP: NHS 10.0.100.4 Tunnel100 vrf 0 Cluster 0 Priority 0 Transitioned to
'RE' from 'E'
NHRP: NHS-UP: 10.0.100.4

```

```

NHRP: Multicast enabled for dst 10.5.6.5
NHRP: NHS 10.0.100.5 Tunnel100 vrf 0 Cluster 0 Priority 0 Transitioned to 'E'
from ' '
NHRP: Attempting to send packet through interface Tunnel100 via DEST dst
10.0.100.5
NHRP: Send Registration Request via Tunnel100 vrf 0, packet size: 108
src: 10.0.100.9, dst: 10.0.100.5
NHRP: Encapsulation succeeded. Sending NHRP Control Packet NBMA Address:
10.5.6.5
NHRP: 132 bytes out Tunnel100
NHRP: Receive Registration Reply via Tunnel100 vrf 0, packet size: 128
NHRP: NHS 10.0.100.5 Tunnel100 vrf 0 Cluster 0 Priority 0 Transitioned to
'RE' from 'E'
NHRP: NHS-UP: 10.0.100.5

```

```

CSR9#show ip nhrp nhs detail
Legend: E=Expecting replies, R=Responding, W=Waiting
Tunnel100:
10.0.100.4 RE priority = 0 cluster = 0 req-sent 4 req-failed 0 repl-recv
4 (00:16:55 ago)
10.0.100.5 RE priority = 0 cluster = 0 req-sent 4 req-failed 0 repl-recv
4 (00:16:55 ago)

```

Viewing the process from the hub's perspective, we see the registration requests come in from the spoke NBMA addresses. Inside that message are the VPN/NBMA mappings, and this allows the hub to create a dynamic NHRP entry for both multicast and unicast traffic. The hub sends the registration reply so the spoke knows its request was processed.

```

! CSR5
NHRP: Receive Registration Request via Tunnel100 vrf 0, packet size: 108
NHRP: Tunnels gave us pak src: 10.7.9.9

```

```

NHRP: Adding Tunnel Endpoints (VPN: 10.0.100.9, NBMA: 10.7.9.9)
NHRP: NHRP subblock already exists for Tunnel Endpoints (VPN: 10.0.100.9,
NBMA: 10.7.9.9)
NHRP: Cache already has a subblock node attached for Tunnel Endpoints (VPN:
10.0.100.9, NBMA: 10.7.9.9)
NHRP: Attempting to send packet through interface Tunnel100 via DEST dst
10.0.100.9
NHRP: Send Registration Reply via Tunnel100 vrf 0, packet size: 128
src: 10.0.100.5, dst: 10.0.100.9
NHRP: Encapsulation succeeded. Sending NHRP Control Packet NBMA Address:
10.7.9.9
NHRP: 152 bytes out Tunnel100

```

```
CSR5#show ip nhrp brief
```

Target	Via	NBMA	Mode	Intfc	Claimed
10.0.100.9/32	10.0.100.9	10.7.9.9	dynamic	Tu100	< >
10.0.100.10/32	10.0.100.10	10.7.10.10	dynamic	Tu100	< >

```
CSR5#show ip nhrp multicast
```

I/F	NBMA address	Flags	
Tunnel100	10.7.10.10	dynamic	(Enabled)
Tunnel100	10.7.9.9	dynamic	(Enabled)

Because NHRP is a component of DMVPN, we can also use the DMVPN-derived debugging. These provide significant additional detail over the NHRP debugs. The same process is shown again, and I've focused on the actual NHRP registration packet. The details of the registration packet are displayed using this debug, so checking for authentication mismatches or NAT-traversal issues (with IPSec) is simplified. The command used is "debug dmvpn all nhrp". The new F, M, and C fields represent several new concepts. "F" is the fixed field, specifying address-family, version, and type. "M" is the mandatory information field, primarily used by the hub to construct dynamic mappings to spokes that register. "C" is the client information element (CIE) field that can be added as NHS speakers need to communicate information back and forth. These three fields are shown in green just for identification purposes. CSR9 sends a registration request to CSR5, and CSR5 responds with a registration reply.

```

! CSR9
NHRP: Attempting to send packet through interface Tunnel100 via DEST dst
10.0.100.5
NHRP: Send Registration Request via Tunnel100 vrf 0, packet size: 108
src: 10.0.100.9, dst: 10.0.100.5
(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
shtl: 4(NSAP), sstl: 0(NSAP)
pktsz: 108 extoff: 52
(M) flags: "unique nat ", reqid: 70
src NBMA: 10.7.9.9
src protocol: 10.0.100.9, dst protocol: 10.0.100.5
(C-1) code: no error(0)
prefix: 32, mtu: 9976, hd_time: 7200

```

```

    addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
Responder Address Extension(3):
Forward Transit NHS Record Extension(4):
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
    type:Cleartext(1), data:NHRPAUTH
NAT address Extension(9):
    (C-1) code: no error(0)
        prefix: 32, mtu: 9976, hd_time: 0
        addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
        client NBMA: 10.5.6.5
        client protocol: 10.0.100.5
NHRP: Setting 'used' flag on cache entry with nhop: 10.0.100.5
NHRP: NHRP successfully mapped '10.0.100.5' to NBMA 10.5.6.5
NHRP: Encapsulation succeeded.  Sending NHRP Control Packet  NBMA Address:
10.5.6.5
NHRP: 132 bytes out Tunnel100

! CSR5
NHRP: Receive Registration Request via Tunnel100 vrf 0, packet size: 108
    (F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
        shtl: 4(NSAP), sstl: 0(NSAP)
        pktsz: 108 extoff: 52
    (M) flags: "unique nat ", reqid: 72
        src NBMA: 10.7.9.9
        src protocol: 10.0.100.9, dst protocol: 10.0.100.5
    (C-1) code: no error(0)
        prefix: 32, mtu: 9976, hd_time: 7200
        addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
Responder Address Extension(3):
Forward Transit NHS Record Extension(4):
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
    type:Cleartext(1), data:NHRPAUTH
NAT address Extension(9):
    (C-1) code: no error(0)
        prefix: 32, mtu: 9976, hd_time: 0
        addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
        client NBMA: 10.5.6.5
        client protocol: 10.0.100.5
NHRP: netid_in = 100, to_us = 1
NHRP: Tunnels gave us pak src: 10.7.9.9
NHRP: Tunnel100: Cache update for target 10.0.100.9/32 next-hop 10.0.100.9
10.7.9.9
NHRP: Adding Tunnel Endpoints (VPN: 10.0.100.9, NBMA: 10.7.9.9)
NHRP: NHRP subblock already exists for Tunnel Endpoints (VPN: 10.0.100.9,
NBMA: 10.7.9.9)
NHRP: Cache already has a subblock node attached for Tunnel Endpoints (VPN:
10.0.100.9, NBMA: 10.7.9.9)

```

```

NHRP: Updating our cache with NBMA: 10.5.6.5, NBMA_ALT: 10.5.6.5
NHRP: New mandatory length: 32
NHRP: Attempting to send packet through interface Tunnel100 via DEST dst
10.0.100.9
NHRP: Send Registration Reply via Tunnel100 vrf 0, packet size: 128
src: 10.0.100.5, dst: 10.0.100.9
(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 128 extoff: 52
(M) flags: "unique nat ", reqid: 72
    src NBMA: 10.7.9.9
    src protocol: 10.0.100.9, dst protocol: 10.0.100.5
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
Responder Address Extension(3):
(C) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.5.6.5
    client protocol: 10.0.100.5
Forward Transit NHS Record Extension(4):
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
    type: Cleartext(1), data: NHRPAUTH
NAT address Extension(9):
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 0
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.5.6.5
    client protocol: 10.0.100.5
NHRP: Setting 'used' flag on cache entry with nhop: 10.0.100.9
NHRP: NHRP successfully mapped '10.0.100.9' to NBMA 10.7.9.9
NHRP: Encapsulation succeeded. Sending NHRP Control Packet NBMA Address:
10.7.9.9
NHRP: 152 bytes out Tunnel100

```

Below is an example of mismatched NHRP authentication. The NHS doesn't actually send a message back to say the authentication failed, but it fails to send a registration response. CSR9 continues to retransmit the message, which could hint at an authentication issue. Before continuing, we correct the authentication issue. This test was simply to show that the debug can help debug those issues.

```

! CSR9
NHRP: Send Registration Request via Tunnel100 vrf 0, packet size: 107
src: 10.0.100.9, dst: 10.0.100.4
(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 107 extoff: 52

```



```

(M) flags: "unique nat ", reqid: 77
    src NBMA: 10.7.9.9
    src protocol: 10.0.100.9, dst protocol: 10.0.100.4
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
Responder Address Extension(3):
Forward Transit NHS Record Extension(4):
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
    type:Cleartext(1), data:BADAUTH
NAT address Extension(9):
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 0
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.4.12.4
    client protocol: 10.0.100.4
NHRP: NHRP successfully mapped '10.0.100.4' to NBMA 10.4.12.4
NHRP: Encapsulation succeeded.  Sending NHRP Control Packet  NBMA Address:
10.4.12.4
NHRP: 131 bytes out Tunnel100
NHRP-RATE: Retransmitting Registration Request for 10.0.100.4, reqid 77,
(retrans ivl 8 sec)

! CSR5
NHRP: Receive Registration Request via Tunnel100 vrf 0, packet size: 107
(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 107 extoff: 52
(M) flags: "unique nat ", reqid: 78
    src NBMA: 10.7.9.9
    src protocol: 10.0.100.9, dst protocol: 10.0.100.5
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
Responder Address Extension(3):
Forward Transit NHS Record Extension(4):
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
    type:Cleartext(1), data:BADAUTH
NAT address Extension(9):
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 0
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.5.6.5
    client protocol: 10.0.100.5

```

We have also configured IPv6 in the overlay also. DMVPN can be an IPv6 transition strategy to connect separated IPv6 islands. The NHRP syntax is nearly identical for IPv6 as it is for IPv4 and is shown below;

only the additional components for IPv6 are shown. Some commands require netmask specification, unlike IPv4. EIGRP is used for IPv6 unicast routing. There are two significant configurations for EIGRP. Disabling split-horizon on the hubs, which are NBMA interfaces, is important to distribute updates to spokes. Making the spokes EIGRP stubs is a good design practice which prevents them from being transit routers and ensures they can never receive EIGRP queries for active prefixes.

```
! Generic hub (CSR5)
interface Tunnel100
  ipv6 address FE80::5 link-local
  ipv6 address 2001:10:0:100::5/64
  ipv6 nhrp authentication NHRPAUTH
  ipv6 nhrp map multicast dynamic
  ipv6 nhrp network-id 100

router eigrp VPN
  address-family ipv6 unicast autonomous-system 65000
    af-interface Tunnel100
      no split-horizon

! Generic spoke (CSR9)
interface Tunnel100
  ipv6 address FE80::9 link-local
  ipv6 address 2001:10:0:100::9/64
  ipv6 nhrp authentication NHRPAUTH
  ipv6 nhrp map 2001:10:0:100::4/128 10.4.12.4
  ipv6 nhrp map 2001:10:0:100::5/128 10.5.6.5
  ipv6 nhrp map multicast 10.4.12.4
  ipv6 nhrp map multicast 10.5.6.5
  ipv6 nhrp network-id 100
  ipv6 nhrp nhs 2001:10:0:100::4 priority 5
  ipv6 nhrp nhs 2001:10:0:100::5 priority 2
```

In cases where routers are behind the "stub", like CSR1 and XRv4, you can either leak routes through or summarize on the stub router. CSR10 uses the leak method while CSR9 uses the summary method. CSR2 only advertises connected routes as it has nothing to summarize nor any downstream EIGRP neighbors. This is a quick side-note and is not directly related to DMVPN, but general EIGRP design over NBMA architectures. These same concepts apply to IPv4 as well.

```
! CSR10
ipv6 prefix-list PL_IPV6_ALL seq 5 permit ::/0 le 128

route-map RM_EIGRP_LEAK_IPV6 permit 10
  match ipv6 address prefix-list PL_IPV6_ALL

router eigrp VPN
  address-family ipv6 unicast autonomous-system 65000
    eigrp stub connected leak-map RM_EIGRP_LEAK_IPV6
```

```

! CSR9
router eigrp VPN
 address-family ipv6 unicast autonomous-system 65000
   eigrp stub connected summary
 af-interface Tunnel100
   summary-address ::14:14:14:0/112
 af-interface GigabitEthernet2.594
   summary-address ::/0

```

```

! CSR2
router eigrp VPN
 address-family ipv6 unicast autonomous-system 65000
   eigrp stub connected

```

The debugs for NHRP are identical between IPv4 and IPv6. CSR9 includes its public IPv6 address in the "M" stanza and its LL address in the first "C" stanza.

```

! CSR9
NHRP: Attempting to send packet through interface Tunnel100 via DEST dst
2001:10:0:100::5
NHRP: Send Registration Request via Tunnel100 vrf 0, packet size: 164
src: 2001:10:0:100::9, dst: 2001:10:0:100::5
(F) afn: AF_IP(1), type: IPv6(86DD), hop: 255, ver: 1
  shtl: 4(NSAP), sstl: 0(NSAP)
  pktsz: 164 extoff: 96
(M) flags: "unique nat ", reqid: 94
  src NBMA: 10.7.9.9
  src protocol: 2001:10:0:100::9, dst protocol: 2001:10:0:100::5
(C-1) code: no error(0)
  prefix: 128, mtu: 9976, hd_time: 7200
  addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 16, pref: 0
  client NBMA: 10.7.9.9
  client protocol: FE80::9
Responder Address Extension(3):
Forward Transit NHS Record Extension(4):
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
  type:Cleartext(1), data:NHRPAUTH
NAT address Extension(9):
(C-1) code: no error(0)
  prefix: 128, mtu: 9976, hd_time: 0
  addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 16, pref: 0
  client NBMA: 10.5.6.5
  client protocol: 2001:10:0:100::5

```

The hub receives this request and programs it in its NHRP cache.

```

! CSR5
NHRP: Receive Registration Request via Tunnel100 vrf 0, packet size: 164
(F) afn: AF_IP(1), type: IPv6(86DD), hop: 255, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 164 extoff: 96
(M) flags: "unique nat ", reqid: 94
    src NBMA: 10.7.9.9
    src protocol: 2001:10:0:100::9, dst protocol: 2001:10:0:100::5
(C-1) code: no error(0)
    prefix: 128, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 16, pref: 0
    client NBMA: 10.7.9.9
    client protocol: FE80::9
[snip, same as above]

```

```

CSR5#show ipv6 nhrp 2001:10:0:100::9/128
2001:10:0:100::9/128 via 2001:10:0:100::9
    Tunnel100 created 00:10:21, expire 01:53:32
    Type: dynamic, Flags: unique registered used nhop
    NBMA address: 10.7.9.9

```

The hub sends a reply message back towards the spoke. The "M" stanza remains unchanged, but there are some additional mappings inserted. The spoke NBMA is mapped to its LL IPv6 address, as is the hubs. We did not see this in IPv4 as LL addressing isn't used the same way.

```

! CSR5
NHRP: Send Registration Reply via Tunnel100 vrf 0, packet size: 228
src: 2001:10:0:100::5, dst: 2001:10:0:100::9
(F) afn: AF_IP(1), type: IPv6(86DD), hop: 255, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 228 extoff: 128
(M) flags: "unique nat ", reqid: 94
    src NBMA: 10.7.9.9
    src protocol: 2001:10:0:100::9, dst protocol: 2001:10:0:100::5
(C-1) code: no error(0)
    prefix: 128, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 16, pref: 0
    client NBMA: 10.7.9.9
    client protocol: FE80::9
(C-2) code: no error(0)
    prefix: 128, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 16, pref: 0
    client NBMA: 10.5.6.5
    client protocol: FE80::5
Responder Address Extension(3):
(C) code: no error(0)
    prefix: 128, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 16, pref: 0

```

```

client NBMA: 10.5.6.5
client protocol: 2001:10:0:100::5
Forward Transit NHS Record Extension(4):
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
  type: Cleartext(1), data: NHRPAUTH
NAT address Extension(9):
  (C-1) code: no error(0)
        prefix: 128, mtu: 9976, hd_time: 0
        addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 16, pref: 0
        client NBMA: 10.5.6.5
        client protocol: 2001:10:0:100::5

```

CSR9 receives the reply, accepts it, and programs it to its NHRP cache. It creates two entries, one for CSR5's real VPN address, and one for its LL address, as both are mapped to the same IPv4 NBMA address. The LL address has a special NHRP flag to identify it as such. It is interesting to note that the "type" of the LL mapping is static, although it was learned. Since it is bound to a hub that was a static mapping, the two are considered equivalent enough to share these attributes.

```

! CSR9
NHRP: Receive Registration Reply via Tunnel100 vrf 0, packet size: 228
(F) afn: AF_IP(1), type: IPv6(86DD), hop: 255, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 228 extoff: 128
(M) flags: "unique nat ", reqid: 90
    src NBMA: 10.7.9.9
    src protocol: 2001:10:0:100::9, dst protocol: 2001:10:0:100::5
[snip, same as above]

```

```

CSR9#show ipv6 nhrp 2001:10:0:100::5/128
2001:10:0:100::5/128 via 2001:10:0:100::5
  Tunnel100 created 00:10:22, never expire
  Type: static, Flags: used
  NBMA address: 10.5.6.5

```

```

CSR9#show ipv6 nhrp FE80::5/128
FE80::5/128 via FE80::5
  Tunnel100 created 00:10:28, never expire
  Type: static, Flags: nhs-ll
  NBMA address: 10.5.6.5

```

At this point, the IPv6 capability in the second hub at CSR4 has not been configured yet. Below is an example of output that indicates one fully functional hub, and one dysfunctional one. The NHS at CSR4, from CSR9's perspective, is expecting replies but not receiving them; 6 requests were sent but 0 received.

```

CSR9#show ipv6 nhrp nhs detail

```

Legend: E=Expecting replies, R=Responding, W=Waiting

Tunnel100:

2001:10:0:100::4 E priority = 5 cluster = 0 req-sent 6 req-failed 0
repl-recv 0

2001:10:0:100::5 RE priority = 2 cluster = 0 req-sent 1 req-failed 0
repl-recv 1 (00:00:45 ago)

Pending Registration Requests:

Registration Request: Reqid 85, Ret 32 NHS 2001:10:0:100::4 expired (Tu100)

Quickly adding the IPv6 NHRP hub configuration to CSR4 (similar to CSR5 and not shown), we can see the registration is successful on CSR9. Also notice that CSR5 has a better (lower) priority than CSR4, which should come into play with phases 2 and 3.

CSR9#show ipv6 nhrp nhs detail

Legend: E=Expecting replies, R=Responding, W=Waiting

Tunnel100:

2001:10:0:100::4 RE priority = 5 cluster = 0 req-sent 42 req-failed 0
repl-recv 1 (00:00:13 ago)

2001:10:0:100::5 RE priority = 2 cluster = 0 req-sent 1 req-failed 0
repl-recv 1 (00:34:37 ago)

Another way to quickly verify that all spokes have registered is by checking the IPv6 NHRP caches on the hub in brief form. CSR4 has 3 spokes while CSR5 only has two; recall that CSR2 is only in the DMVPN connected to CSR4, using a P2P tunnel for connectivity to CSR5. The hubs show the spoke LL addresses as dynamic mappings as they were contained inside the NHRP registration requests initially.

CSR5#show ipv6 nhrp brief

Target	Via		
2001:10:0:100::9/128	2001:10:0:100::9		
NBMA: 10.7.9.9	Mode: dynamic	Intfc: Tu100	Claimed: < >
2001:10:0:100::10/128	2001:10:0:100::10		
NBMA: 10.7.10.10	Mode: dynamic	Intfc: Tu100	Claimed: < >
FE80::9/128	2001:10:0:100::9		
NBMA: 10.7.9.9	Mode: dynamic	Intfc: Tu100	Claimed: < >
FE80::10/128	2001:10:0:100::10		
NBMA: 10.7.10.10	Mode: dynamic	Intfc: Tu100	Claimed: < >

CSR4#show ipv6 nhrp brief

Target	Via		
2001:10:0:100::2/128	2001:10:0:100::2		
NBMA: 10.8.2.2	Mode: dynamic	Intfc: Tu100	Claimed: < >
2001:10:0:100::9/128	2001:10:0:100::9		
NBMA: 10.7.9.9	Mode: dynamic	Intfc: Tu100	Claimed: < >
2001:10:0:100::10/128	2001:10:0:100::10		
NBMA: 10.7.10.10	Mode: dynamic	Intfc: Tu100	Claimed: < >
FE80::2/128	2001:10:0:100::2		
NBMA: 10.8.2.2	Mode: dynamic	Intfc: Tu100	Claimed: < >

FE80::9/128	2001:10:0:100::9			
NBMA: 10.7.9.9	Mode: dynamic	Intfc: Tu100	Claimed: <	>
FE80::10/128	2001:10:0:100::10			
NBMA: 10.7.10.10	Mode: dynamic	Intfc: Tu100	Claimed: <	>

The only time phase 1 NHRP resolution requests are issued by spokes is when they are trying to directly reach one another's VPN addresses when the route recursion points to a connected route. This also only occurs when there are multiple hubs and the spokes use mGRE interfaces (not P2P interfaces). For example, CSR9 tries to ping CSR10's VPN address along with its loopback. In IPv4, this VPN address is an OSPF /32 route given the P2MP network type used. The loopback is a normal OSPF /32 route. Both of these follow the regular routing rules through the hub, and there are no reachability issues. This is a very rare case, but the process is near-identical for other DMVPN phases, so we analyze it here.

```
CSR9#show ip cef 10.0.100.10
10.0.100.10/32
  nexthop 10.0.100.4 Tunnel100
  nexthop 10.0.100.5 Tunnel100
```

```
CSR9#show ip cef 10.10.10.10
10.10.10.10/32
  nexthop 10.0.100.4 Tunnel100
  nexthop 10.0.100.5 Tunnel100
```

```
CSR9#ping 10.0.100.10
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.100.10, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 29/34/42 ms
```

```
CSR9#ping 10.10.10.10
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.10.10.10, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 16/33/46 ms
```

Doing the same test to CSR2's VPN address and loopback is different. The path to the loopback is a normal OSPF /32 route, but the path to the VPN endpoint is the connected /24 route. Because of the NBMA network, CSR9 does not have direct connectivity to this. It needs to query the hub for resolution.

```
CSR9#show ip cef 10.0.100.2
10.0.100.0/24
  attached to Tunnel100
```

```
CSR9#show ip cef 2.2.2.2
2.2.2.2/32
  nexthop 10.0.100.4 Tunnel100
  nexthop 10.0.100.5 Tunnel100
```

When it queries the hub, only CSR4 can respond since CSR2 is not running DMVPN with CSR5 at all (uses P2P tunnel). The resolution request is shown below. This is CSR9 asking CSR4 "What is the NBMA address for VPN address 10.0.100.2"? I usually look at the "M" field first since it contains 3 of the 4 key pieces of information, and the resolution request seeks to find the fourth.

```
! CSR9
NHRP: Send Resolution Request via Tunnel100 vrf 0, packet size: 88
src: 10.0.100.9, dst: 10.0.100.2
(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 88 extoff: 52
(M) flags: "router auth src-stable nat ", reqid: 8
    src NBMA: 10.7.9.9
    src protocol: 10.0.100.9, dst protocol: 10.0.100.2
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
```

The hub receives the resolution request, performs a route lookup, and then forwards the request towards CSR2. This is similar to what a LISP MR/MS does when receiving map-requests from ITRs. The NHS inserts a "transit NHS record extension" into the packet which includes its local VPN/NBMA mapping. This simply means that an NHS was in the resolution transit path from source to destination (the path of the resolution request). Although technically CSR4 knows the NBMA mapping for 10.0.100.2, it forwards the request to CSR2 for authoritative resolution.

```
! CSR4
NHRP: Receive Resolution Request via Tunnel100 vrf 0, packet size: 88
(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 88 extoff: 52
(M) flags: "router auth src-stable nat ", reqid: 8
    src NBMA: 10.7.9.9
    src protocol: 10.0.100.9, dst protocol: 10.0.100.2
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
```

```
NHRP: Route lookup for destination 10.0.100.2 in (0x0) yielded interface
Tunnel100, prefixlen 24
```

```
NHRP-ATTR: In nhrp_rcv_resolution_request NHRP Resolution Request packet is
forwarded to 10.0.100.2
```

```
NHRP: Attempting to forward to destination: 10.0.100.2
```

```
NHRP: Forwarding: NHRP SAS picked source: 10.0.100.4 for destination:
10.0.100.2
```



```

NHRP: Attempting to send packet through interface Tunnel100 via DEST dst
10.0.100.2
NHRP: Forwarding Resolution Request via Tunnel100 vrf 0, packet size: 108
src: 10.0.100.4, dst: 10.0.100.2
(F) afn: AF_IP(1), type: IP(800), hop: 254, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 108 extoff: 52
(M) flags: "router auth src-stable nat ", reqid: 8
    src NBMA: 10.7.9.9
    src protocol: 10.0.100.9, dst protocol: 10.0.100.2
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
Responder Address Extension(3):
Forward Transit NHS Record Extension(4):
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.4.12.4
    client protocol: 10.0.100.4

```

CSR2 receives that modified resolution request from CSR9 (via CSR4 in the middle), performs a route lookup. CSR2 determines it's a local route and generates a resolution reply for CSR9, using the NHS as transit.

```

! CSR2
NHRP: Receive Resolution Request via Tunnel100 vrf 0, packet size: 108
(F) afn: AF_IP(1), type: IP(800), hop: 254, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 108 extoff: 52
(M) flags: "router auth src-stable nat ", reqid: 9
    src NBMA: 10.7.9.9
    src protocol: 10.0.100.9, dst protocol: 10.0.100.2
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
Responder Address Extension(3):
Forward Transit NHS Record Extension(4):
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.4.12.4
    client protocol: 10.0.100.4

NHRP: Route lookup for destination 10.0.100.2 in (0x0) yielded interface
Tunnel100, prefixlen
NHRP: Request was to us. Process the NHRP Resolution Request.

```

At this point, CSR2 has enough information to where it can add CSR9's NBMA address to its local cache. Unlike LISP, the entire resolution process does not have to happen in both directions. CSR2 marks this entry as "implicit" because an explicit resolution was not triggered; it was able to glean the information from a resolution request to which it was responding.

```
! CSR2
NHRP: Tunnel100: Cache update for target 10.0.100.9/32 next-hop 10.0.100.9
10.7.9.9

CSR2#show ip nhrp 10.0.100.9
10.0.100.9/32 via 10.0.100.9
Tunnel100 created 00:18:03, expire 01:58:31
Type: dynamic, Flags: router implicit nhop
NBMA address: 10.7.9.9
```

The first line of the debug below is interesting and details why Tunnel25 was not used for NHRP signaling; it is not part of the DMVPN network and should not be used for NHRP-resolved routes. CSR2 adds a new extension for "responder address", which makes sense since CSR2 originated the resolution reply. This is where the NBMA mapping is encoded and ultimately how CSR9's initial question is answered. Notice this information is in the first CIE field also.

```
! CSR2
NHRP: Pak out Tunnel25 would leave logical NBMA network
NHRP: Attempting to send packet via NHS 10.0.100.4
NHRP: Send Resolution Reply via Tunnel100 vrf 0, packet size: 136
src: 10.0.100.2, dst: 10.0.100.4
(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
shtl: 4(NSAP), sstl: 0(NSAP)
pktsz: 136 extoff: 60
(M) flags: "router auth dst-stable unique src-stable nat ", reqid: 9
src NBMA: 10.7.9.9
src protocol: 10.0.100.9, dst protocol: 10.0.100.2
(C-1) code: no error(0)
prefix: 32, mtu: 9976, hd_time: 7200
addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
client NBMA: 10.8.2.2
client protocol: 10.0.100.2
Responder Address Extension(3):
(C) code: no error(0)
prefix: 32, mtu: 9976, hd_time: 7200
addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
client NBMA: 10.8.2.2
client protocol: 10.0.100.2
Forward Transit NHS Record Extension(4):
(C-1) code: no error(0)
prefix: 32, mtu: 9976, hd_time: 7200
addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
```

```
client NBMA: 10.4.12.4
client protocol: 10.0.100.4
```

CSR4 receives the resolution reply from CSR2 and forwards it back to CSR9. It also adds a "reverse transit NHS record" to show that the NHS was also in the transit path going in the reverse (reply) direction also.

```
! CSR4
NHRP: Receive Resolution Reply via Tunnel100 vrf 0, packet size: 136
(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 136 extoff: 60
(M) flags: "router auth dst-stable unique src-stable nat ", reqid: 8
    src NBMA: 10.7.9.9
    src protocol: 10.0.100.9, dst protocol: 10.0.100.2
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.8.2.2
    client protocol: 10.0.100.2
Responder Address Extension(3):
(C) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.8.2.2
    client protocol: 10.0.100.2
Forward Transit NHS Record Extension(4):
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.4.12.4
    client protocol: 10.0.100.4

NHRP: Forwarding Resolution Reply via Tunnel100 vrf 0, packet size: 156
src: 10.0.100.4, dst: 10.0.100.9
(F) afn: AF_IP(1), type: IP(800), hop: 254, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 156 extoff: 60
(M) flags: "router auth dst-stable unique src-stable nat ", reqid: 8
    src NBMA: 10.7.9.9
    src protocol: 10.0.100.9, dst protocol: 10.0.100.2
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.8.2.2
    client protocol: 10.0.100.2
Responder Address Extension(3):
(C) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
```

```

    client NBMA: 10.8.2.2
    client protocol: 10.0.100.2
Forward Transit NHS Record Extension(4):
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.4.12.4
    client protocol: 10.0.100.4
Reverse Transit NHS Record Extension(5):
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.4.12.4
    client protocol: 10.0.100.4

```

CSR9 receives this packet which is full of information. The original 'M' stanza contains three out of the four key pieces of information: source VPN, source NBMA, and destination VPN. This was originated by CSR9 in the first place. The entire purpose of this entire process was to resolve the destination NBMA to reach CSR2. The following stanza (CIE) contains the "answer" to that query, which is 10.8.2.2 mapped to VPN address 10.0.100.2. The remaining fields are used internally by NHRP for tracking from which NHS this resolution was proxied, among other things.

```

! CSR9
NHRP: Receive Resolution Reply via Tunnel100 vrf 0, packet size: 156
(F) afn: AF_IP(1), type: IP(800), hop: 254, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 156 extoff: 60
(M) flags: "router auth dst-stable unique src-stable nat ", reqid: 8
    src NBMA: 10.7.9.9
    src protocol: 10.0.100.9, dst protocol: 10.0.100.2
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.8.2.2
    client protocol: 10.0.100.2
Responder Address Extension(3):
(C) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.8.2.2
    client protocol: 10.0.100.2
Forward Transit NHS Record Extension(4):
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.4.12.4
    client protocol: 10.0.100.4
Reverse Transit NHS Record Extension(5):

```

```
(C-1) code: no error(0)
      prefix: 32, mtu: 9976, hd_time: 7200
      addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
      client NBMA: 10.4.12.4
      client protocol: 10.0.100.4
```

CSR9 processes the received resolution reply and adds this information to its NHRP cache. Notice the word "implicit" is absent since CSR9 send an explicit resolution request to the NHS, CSR4.

```
! CSR9
NHRP: Tunnel100: Cache update for target 10.0.100.2/32 next-hop 10.0.100.2
      10.8.2.2
NHRP: Adding Tunnel Endpoints (VPN: 10.0.100.2, NBMA: 10.8.2.2)
NHRP: Dequeue request of type Resolution Request for 10.0.100.2, reqid 8,
netid 100
```

```
CSR9#show ip nhrp 10.0.100.2
10.0.100.2/32 via 10.0.100.2
      Tunnel100 created 00:00:33, expire 01:59:26
      Type: dynamic, Flags: router nhop
      NBMA address: 10.8.2.2
```

This is a very bad design because CSR2 is only a P2P tunnel, it can never install dynamic NHRP mappings on that interface as the destination is fixed. CSR9 can send traffic to 10.0.100.2, but the reply traffic will never work. In this way, phase 1 can be tricky to troubleshoot and understand. Even though the NHRP mappings appear correct, there is no connectivity between VPN addresses. Normally, you would have a single hub with all sites using P2P tunnels (with prefix-suppression of sorts, somewhat like CSR2), or multiple hubs plus all sites using mGRE (CSR9 and CSR10).

Next, we can begin data-plane testing; most reachability is achievable both within the VPN and to the Internet site, with the exception of some VPN addresses due to an intentionally poor design described above. Only connected networks (transit links) have Internet access, since NAT is not configured as it is beyond the scope of this test. Of greater interest is determining the routing path; a traceroute on CSR9 to CSR10 clearly shows the hub(s) in the transit path. The CEF table confirms this both for CSR10's loopback and the OSPF summary behind it. This is the main disadvantage of DMVPN phase 1 and the reason why it's rarely used.

```
CSR9#traceroute 10.10.10.10
Type escape sequence to abort.
Tracing the route to 10.10.10.10
VRF info: (vrf in name/id, vrf out name/id)
 1 10.0.100.4 17 msec
   10.0.100.5 25 msec
   10.0.100.4  8 msec
 2 10.0.100.10 33 msec *  64 msec
```

```
CSR9#show ip cef 10.10.10.10
10.10.10.10/32
  nexthop 10.0.100.4 Tunnel100
  nexthop 10.0.100.5 Tunnel100
```

```
CSR9#show ip cef 1.1.1.1
1.1.1.0/24
  nexthop 10.0.100.4 Tunnel100
  nexthop 10.0.100.5 Tunnel100
```

The same is true for IPv6 routing. Because of EIGRP stub routing, The hubs cannot reach one another without a P2P tunnel or static mappings on their mGRE interfaces. I add a P2P tunnel quickly to achieve reachability (not shown). CSR9's paths to other remote sites are shown also, and as expected, route through the hubs directly.

```
CSR5#show ipv6 cef ::4:4:4:4
::4:4:4:4/128
  nexthop FE80::4 Tunnel145
```

```
CSR9#show ipv6 cef ::1:1:1:1
::1:1:1:1/128
  nexthop FE80::4 Tunnel100
  nexthop FE80::5 Tunnel100
```

```
CSR9#show ipv6 cef ::2:2:2:2
::2:2:2:2/128
  nexthop FE80::4 Tunnel100
  nexthop FE80::5 Tunnel100
```

Additional Reading – Reference configurations "dmvpn1"

35.2.2 Phase 2

DMVPN phase 2 solves some of the problems associated with phase 1 by allowing spokes to send traffic directly to one another. This implies that all routers must have the same mGRE interface (no P2P links with static tunnel destinations) so that they can install NHRP mappings to remote NBMA addresses inside the same DMVPN. GRE keepalives are not supported at any node, hub or spoke, in this design. From a routing perspective, EIGRP must be configured with split-horizon disabled (like phase 1) and with next-hop-self disabled (specific to phase 2). This allows spoke VPN addresses for re-advertised routes to be maintained, which allows traffic to traverse the dynamic spoke-to-spoke tunnels. For OSPF, a multi-access network type should be used (broadcast or nonbroadcast). Typically broadcast is used as it simplifies the configuration, but the spokes must have their DR priorities set to 0 as only hub nodes have connectivity to all endpoints to distribute OSPF updates. We will use OSPFv2 for IPv4 and EIGRP for IPv6 as we did in phase 1. I have removed the P2P tunnel between CSR4/CSR5 and used a static mapping on CSR5 to build the linkage. CSR5 is a backup hub that references CSR4 as an NHS also, so within a local DMVPN phase 2 design, you can achieve a little bit of hierarchy this way. For brevity, only the relevant

configuration components are shown, since the basic configuration (addresses, tunnel source, etc) has not changed.

```
! CSR4, NHS only
interface Tunnel100
  description DMVPN PHASE 2 - HUB
  ip nhrp authentication NHRPAUTH
  ip nhrp map multicast dynamic
  ip nhrp network-id 100
  ip nhrp server-only
  ip ospf network broadcast
  ip ospf priority 5
  ip ospf 1 area 0
  ipv6 nhrp authentication NHRPAUTH
  ipv6 nhrp map multicast dynamic
  ipv6 nhrp network-id 100

! CSR5, NHS and client
interface Tunnel100
  description DMVPN PHASE 2 HUB/CLIENT
  ip nhrp authentication NHRPAUTH
  ip nhrp map multicast dynamic
  ip nhrp map multicast 10.4.12.4
  ip nhrp map 10.0.100.4 10.4.12.4
  ip nhrp network-id 100
  ip nhrp nhs 10.0.100.4
  ip ospf network broadcast
  ip ospf 1 area 0
  ipv6 nhrp authentication NHRPAUTH
  ipv6 nhrp map multicast dynamic
  ipv6 nhrp map multicast 10.4.12.4
  ipv6 nhrp map 2001:10:0:100::4/128 10.4.12.4
  ipv6 nhrp network-id 100
  ipv6 nhrp nhs 2001:10:0:100::4

! General NHS client (spoke)
interface Tunnel100
  description DMVPN PHASE 2 - SPOKE
  ip nhrp authentication NHRPAUTH
  ip nhrp map 10.0.100.4 10.4.12.4
  ip nhrp map multicast 10.4.12.4
  ip nhrp map 10.0.100.5 10.5.6.5
  ip nhrp map multicast 10.5.6.5
  ip nhrp network-id 100
  ip nhrp nhs 10.0.100.4
  ip nhrp nhs 10.0.100.5
  ip ospf network broadcast
  ip ospf priority 0
  ip ospf 1 area 0
```

```

ipv6 nhrp authentication NHRPAUTH
ipv6 nhrp map 2001:10:0:100::4/128 10.4.12.4
ipv6 nhrp map 2001:10:0:100::5/128 10.5.6.5
ipv6 nhrp map multicast 10.4.12.4
ipv6 nhrp map multicast 10.5.6.5
ipv6 nhrp network-id 100
ipv6 nhrp nhs 2001:10:0:100::4 priority 5
ipv6 nhrp nhs 2001:10:0:100::5 priority 2

```

Before starting, a quick check of NHRP registrations on the hubs is valuable to ensure the network has converged. Notice that CSR4 has all dynamic entries, where CSR5 has a static entry for CSR4. Then, checking OSPF neighbors confirms that LL multicast is functioning (assuming "broadcast" network type) and IGP can converge. The hubs should be candidate DRs while the spokes are forced to never be DRs (DROthers).

```
CSR4#show ip nhrp brief
```

Target	Via	NBMA	Mode	Intfc	Claimed
10.0.100.2/32	10.0.100.2	10.8.2.2	dynamic	Tu100	< >
10.0.100.5/32	10.0.100.5	10.5.6.5	dynamic	Tu100	< >
10.0.100.9/32	10.0.100.9	10.7.9.9	dynamic	Tu100	< >
10.0.100.10/32	10.0.100.10	10.7.10.10	dynamic	Tu100	< >

```
CSR5#show ip nhrp brief
```

Target	Via	NBMA	Mode	Intfc	Claimed
10.0.100.2/32	10.0.100.2	10.8.2.2	dynamic	Tu100	< >
10.0.100.4/32	10.0.100.4	10.4.12.4	static	Tu100	< >
10.0.100.9/32	10.0.100.9	10.7.9.9	dynamic	Tu100	< >
10.0.100.10/32	10.0.100.10	10.7.10.10	dynamic	Tu100	< >

```
CSR4#show ip ospf neighbor
```

Neighbor ID	Pri	State	Dead Time	Address	Interface
2.2.2.2	0	FULL/DROTHER	00:00:35	10.0.100.2	Tunnel100
5.5.5.5	1	FULL/BDR	00:00:32	10.0.100.5	Tunnel100
10.9.9.9	0	FULL/DROTHER	00:00:36	10.0.100.9	Tunnel100
10.10.10.10	0	FULL/DROTHER	00:00:37	10.0.100.10	Tunnel100

```
CSR5#show ip ospf neighbor
```

Neighbor ID	Pri	State	Dead Time	Address	Interface
2.2.2.2	0	FULL/DROTHER	00:00:32	10.0.100.2	Tunnel100
4.4.4.4	5	FULL/DR	00:00:34	10.0.100.4	Tunnel100
10.9.9.9	0	FULL/DROTHER	00:00:33	10.0.100.9	Tunnel100
10.10.10.10	0	FULL/DROTHER	00:00:35	10.0.100.10	Tunnel100

When CSR1 sends traffic to XRv14, CSR10 is the next router in the path. However, CSR10 does not know the NBMA address for 10.0.100.9, which is the next-hop of the route for 14.14.14.0/24. CSR10 also does not report having an NHRP mapping for this VPN address (IGP next-hop), which is loosely analogous to not having an ARP entry (MAC address) for a next-hop IP address.


```
CSR1#show ip cef 14.14.14.14
0.0.0.0/0
  nexthop 10.1.10.10 GigabitEthernet2.510
```

```
CSR10#show ip cef 14.14.14.14
14.14.14.0/24
  nexthop 10.0.100.9 Tunnel100
```

```
CSR10#show ip nhrp 10.0.100.9
[no output]
```

The lack of a valid NHRP mapping for 10.0.100.9 triggers an NHRP resolution request on CSR10. CSR10 sends this request to CSR4 only, presumably before it comes first in the configuration as the priorities are the same. The debug below is very chatty (I've reduced many parts), but it effectively says "I don't know the NBMA for 10.0.100.9, so I will ask CSR4. I have the NBMA/VPN mappings for the NHS, so I will send a resolution request for 10.0.100.9 to the NHS NBMA address".

```
! CSR10
NHRP: NHRP could not map 10.0.100.9 to NBMA, cache entry not found
NHRP: MACADDR: if_in GigabitEthernet2.510 netid-in 0 if_out Tunnel100 netid-
out 100
NHRP: Sending packet to NHS 10.0.100.4 on Tunnel100
NHRP: NHRP successfully mapped '10.0.100.4' to NBMA 10.4.12.4
NHRP: Tunnel100: Cache add for target 10.0.100.9/32 next-hop 10.0.100.9
      10.4.12.4
NHRP: Posted msg for temp cache installation - Tunnel100: dst 10.0.100.9, nhs
10.0.100.4(nbma 10.4.12.4),tableid 0, holdtime 185, afn 1
NHRP: Enqueued NHRP Resolution Request for destination: 10.0.100.9
NHRP: Tunnel100: Cache update for target 10.0.100.9/32 next-hop 10.0.100.9
      10.4.12.4
NHRP: Adding Tunnel Endpoints (VPN: 10.0.100.9, NBMA: 10.4.12.4)
NHRP: Successfully attached NHRP subblock for Tunnel Endpoints (VPN:
10.0.100.9, NBMA: 10.4.12.4)
NHRP: Send Resolution Request via Tunnel100 vrf 0, packet size: 88
      src: 10.0.100.10, dst: 10.0.100.9
      (F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
          shtl: 4(NSAP), sstl: 0(NSAP)
          pktsz: 88 extoff: 52
      (M) flags: "router auth src-stable nat ", reqid: 8
          src NBMA: 10.7.10.10
          src protocol: 10.0.100.10, dst protocol: 10.0.100.9
      (C-1) code: no error(0)
          prefix: 32, mtu: 9976, hd_time: 7200
          addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
Responder Address Extension(3):
Forward Transit NHS Record Extension(4):
Reverse Transit NHS Record Extension(5):
```

```

Authentication Extension(7):
  type:Cleartext(1), data:NHRPAUTH
NAT address Extension(9):
NHRP: NHRP successfully mapped '10.0.100.9' to NBMA 10.4.12.4
NHRP: Encapsulation succeeded.  Sending NHRP Control Packet  NBMA Address:
10.4.12.4
NHRP: 112 bytes out Tunnel100

```

CSR4 receives the resolution request from CSR10 attempting to resolve the NBMA address for CSR9. CSR4 does a route lookup for 10.0.100.9 and sees that it points out of Tunnel100, which means it's within the same DMVPN. Like a standard NHRP resolution, the NHS adds itself as the "forward transit NHS" inside the request before sending the traffic towards the destination.

```

! CSR4
NHRP: Receive Resolution Request via Tunnel100 vrf 0, packet size: 88
(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
  shtl: 4(NSAP), sstl: 0(NSAP)
  pktsz: 88 extoff: 52
(M) flags: "router auth src-stable nat ", reqid: 8
  src NBMA: 10.7.10.10
  src protocol: 10.0.100.10, dst protocol: 10.0.100.9
(C-1) code: no error(0)
  prefix: 32, mtu: 9976, hd_time: 7200
  addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
Responder Address Extension(3):
Forward Transit NHS Record Extension(4):
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
  type:Cleartext(1), data:NHRPAUTH
NAT address Extension(9):
NHRP: Route lookup for destination 10.0.100.9 in (0x0) yielded interface
Tunnel100, prefixlen 24
NHRP: Forwarding request due to authoritative request.
NHRP-ATTR: In nhrp_rcv_resolution_request NHRP Resolution Request packet is
forwarded to 10.0.100.9
NHRP: Attempting to forward to destination: 10.0.100.9
NHRP: Forwarding: NHRP SAS picked source: 10.0.100.4 for destination:
10.0.100.9
NHRP: Attempting to send packet through interface Tunnel100 via DEST  dst
10.0.100.9
NHRP: Forwarding Resolution Request via Tunnel100 vrf 0, packet size: 108
src: 10.0.100.4, dst: 10.0.100.9
(F) afn: AF_IP(1), type: IP(800), hop: 254, ver: 1
  shtl: 4(NSAP), sstl: 0(NSAP)
  pktsz: 108 extoff: 52
(M) flags: "router auth src-stable nat ", reqid: 8
  src NBMA: 10.7.10.10
  src protocol: 10.0.100.10, dst protocol: 10.0.100.9

```

```

(C-1) code: no error(0)
      prefix: 32, mtu: 9976, hd_time: 7200
      addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
Responder Address Extension(3):
Forward Transit NHS Record Extension(4):
(C-1) code: no error(0)
      prefix: 32, mtu: 9976, hd_time: 7200
      addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
      client NBMA: 10.4.12.4
      client protocol: 10.0.100.4
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
  type:Cleartext(1), data:NHRPAUTH
NAT address Extension(9):
NHRP: NHRP successfully mapped '10.0.100.9' to NBMA 10.7.9.9
NHRP: Encapsulation succeeded.  Sending NHRP Control Packet  NBMA Address:
10.7.9.9
NHRP: 132 bytes out Tunnel100

```

CSR9 receives the resolution request originated by CSR10 and forwarded by CSR4. CSR9 performs a route lookup for the destination and sees that it is a local interface, so the packet is processed locally. CSR9 also knows the NBMA/VPN mappings for the source, CSR10, as this information is contained in the original request. This is considered an "implicit" mapping and is valid for entry into CSR10's NHRP cache.

```

! CSR9
NHRP: Receive Resolution Request via Tunnel100 vrf 0, packet size: 108
(F) afn: AF_IP(1), type: IP(800), hop: 254, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 108 extoff: 52
(M) flags: "router auth src-stable nat ", reqid: 8
    src NBMA: 10.7.10.10
    src protocol: 10.0.100.10, dst protocol: 10.0.100.9
(C-1) code: no error(0)
      prefix: 32, mtu: 9976, hd_time: 7200
      addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
Responder Address Extension(3):
Forward Transit NHS Record Extension(4):
(C-1) code: no error(0)
      prefix: 32, mtu: 9976, hd_time: 7200
      addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
      client NBMA: 10.4.12.4
      client protocol: 10.0.100.4
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
  type:Cleartext(1), data:NHRPAUTH
NAT address Extension(9):
NHRP: Route lookup for destination 10.0.100.9 in (0x0) yielded interface
Tunnel100, prefixlen 24

```

```

NHRP-ATTR: smart spoke feature and attributes are not configured,
NHRP: Request was to us. Process the NHRP Resolution Request.
NHRP: Request was to us, responding with our address
NHRP: Checking for delayed event 10.0.100.10/10.0.100.9 on list (Tunnel100).
NHRP: No delayed event node found.
NHRP: No need to delay processing of resolution event nbma src:10.7.9.9 nbma
dst:10.7.10.10
NHRP: Tunnel100: Cache add for target 10.0.100.10/32 next-hop 10.0.100.10
10.7.10.10
NHRP: Adding Tunnel Endpoints (VPN: 10.0.100.10, NBMA: 10.7.10.10)
NHRP: Successfully attached NHRP subblock for Tunnel Endpoints (VPN:
10.0.100.10, NBMA: 10.7.10.10)
NHRP: Inserted subblock node for cache: Target 10.0.100.10/32 nhop
10.0.100.10
NHRP: Converted internal dynamic cache entry for 10.0.100.10/32 interface
Tunnel100 to external
NHRP: Tunnel100: Cache update for target 10.0.100.10/32 next-hop 10.0.100.10
10.7.10.10

```

Unlike the process we examined for phase 1, the resolution reply is sent directly back to CSR10, which means the NHS won't see it again. I presume this is because the IGP routing directs the traffic directly back to the spoke; with Phase 1, the hub was always in the transit path, even with the NHRP resolutions that occurred between VPN addresses. As expected, CSR9 adds the "responder address extension" field to the NHRP message so that the originator knows who responded.

```

! CSR9
NHRP: Attempting to send packet through interface Tunnel100 via DEST dst
10.0.100.10
NHRP: Send Resolution Reply via Tunnel100 vrf 0, packet size: 136
src: 10.0.100.9, dst: 10.0.100.10
(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
shtl: 4(NSAP), sstl: 0(NSAP)
pktsz: 136 extoff: 60
(M) flags: "router auth dst-stable unique src-stable nat ", reqid: 8
src NBMA: 10.7.10.10
src protocol: 10.0.100.10, dst protocol: 10.0.100.9
(C-1) code: no error(0)
prefix: 32, mtu: 9976, hd_time: 7200
addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
client NBMA: 10.7.9.9
client protocol: 10.0.100.9
Responder Address Extension(3):
(C) code: no error(0)
prefix: 32, mtu: 9976, hd_time: 7200
addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
client NBMA: 10.7.9.9
client protocol: 10.0.100.9
Forward Transit NHS Record Extension(4):

```

```

(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.4.12.4
    client protocol: 10.0.100.4
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
    type: Cleartext(1), data: NHRPAUTH
NAT address Extension(9):
NHRP: Setting 'used' flag on cache entry with nhop: 10.0.100.10
NHRP: NHRP successfully mapped '10.0.100.10' to NBMA 10.7.10.10
NHRP: Encapsulation succeeded. Sending NHRP Control Packet NBMA Address:
10.7.10.10
NHRP: 160 bytes out Tunnel100

```

The output below shows the entry that CSR9 adds upon receiving the resolution request from CSR10. Since the source NBMA was contained in the message, an "implicit" entry can be added since it's fair to assume that the return traffic should be sent towards the source NBMA address. After all, CSR9 just used this mapping to send a resolution reply directly back to CSR9, avoiding the transit NHS on the return path.

```

CSR9#show ip nhrp 10.0.100.10
10.0.100.10/32 via 10.0.100.10
    Tunnel100 created 00:36:11, expire 01:23:48
    Type: dynamic, Flags: router implicit used nhop
    NBMA address: 10.7.10.10

```

CSR10 receives the resolution reply from CSR9 which contains its NBMA address 10.7.9.9. CSR10 adds this to its NHRP cache and can directly encapsulate unicast traffic to this spoke inside mGRE. The debug reveals this new NBMA information in several different ways, which is why I have trimmed it down.

```

! CSR10
NHRP: Receive Resolution Reply via Tunnel100 vrf 0, packet size: 136
(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 136 extoff: 60
(M) flags: "router auth dst-stable unique src-stable nat ", reqid: 8
    src NBMA: 10.7.10.10
    src protocol: 10.0.100.10, dst protocol: 10.0.100.9
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.7.9.9
    client protocol: 10.0.100.9
Responder Address Extension(3):
(C) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200

```

```

    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.7.9.9
    client protocol: 10.0.100.9
Forward Transit NHS Record Extension(4):
  (C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.4.12.4
    client protocol: 10.0.100.4
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
  type: Cleartext(1), data: NHRPAUTH
NAT address Extension(9):
NHRP: nhrp_pak_reply_from_target: answer_nbma: 10.7.9.9, responder_nbma:
10.7.9.9
NHRP: Tunnels gave us pak src: 10.7.9.9
NHRP: No need to delay processing of resolution event nbma src:10.7.10.10
nbma dst:10.7.9.9
NHRP: Tunnel100: Cache update for target 10.0.100.9/32 next-hop 10.0.100.9
10.7.9.9
NHRP: Deleted subblock node associated with cache: Target 10.0.100.9/32 nhop
10.0.100.9
NHRP: Adding Tunnel Endpoints (VPN: 10.0.100.9, NBMA: 10.7.9.9)
NHRP: Successfully attached NHRP subblock for Tunnel Endpoints (VPN:
10.0.100.9, NBMA: 10.7.9.9)
NHRP: Tunnel100: Cache update for target 10.0.100.9/32 next-hop 10.0.100.9
10.7.9.9
NHRP: Adding Tunnel Endpoints (VPN: 10.0.100.9, NBMA: 10.7.9.9)
NHRP: Dequeue request of type Resolution Request for 10.0.100.9, reqid 8,
netid 100.

```

Below is the final result of the NHRP process. CSR10 adds this NBMA entry as a dynamic entry into its NHRP cache. Nothing changes in the RIB/FIB since the next-hop for the routes behind a spoke is the spoke's VPN address. As long as NHRP can provide the dynamic "tunnel destination", we can use traceroute to prove traffic routes directly between nodes; notice CSR4/CSR5 are not in the transit path.

```

CSR10#show ip nhrp 10.0.100.9
10.0.100.9/32 via 10.0.100.9
  Tunnel100 created 00:35:53, expire 01:24:06
  Type: dynamic, Flags: router nhop
  NBMA address: 10.7.9.9

```

```

CSR1#traceroute 14.14.14.14 source 1.1.1.1
Type escape sequence to abort.
Tracing the route to 14.14.14.14
VRF info: (vrf in name/id, vrf out name/id)
 1 10.1.10.10 0 msec 0 msec 1 msec
 2 10.0.100.9 1 msec 1 msec 5 msec

```

```
3 10.9.14.14 8 msec * 2 msec
```

We will quickly test IPv6 also without much debugging. The nature of EIGRP provides two identical routes to CSR10 for ::14:14:14:0/112 since the hubs are leaving the LL next-hop the same and simply re-advertising it (like a route-reflector). CSR10 does not have an NHRP mapping for this LL next-hop, which is essentially a VPN address for CSR9.

```
CSR1#show ipv6 cef ::14:14:14:14
::14:14:14:0/112
  nexthop FE80::10 GigabitEthernet2.510
```

```
CSR10#show ipv6 cef ::14:14:14:12
::14:14:14:0/112
  nexthop FE80::9 Tunnel100
  nexthop FE80::9 Tunnel100
```

```
CSR10#show ipv6 nhrp FE80::9/128
[no output]
```

When CSR1 sends traffic, the NHRP process takes place. CSR9 sends its request to CSR4 (small debug shown), which forwards it to CSR10. CSR10 then responds directly.

```
! CSR9
NHRP: NHRP successfully mapped '2001:10:0:100::4' to NBMA 10.4.12.4
NHRP: IPv6-resolution: Sending packet to NHS 2001:10:0:100::4/10.4.12.4 on
Tunnel100 - status: 1
```

When CSR9 receives the resolution reply, it installs the LL mapping into the NHRP cache. Even though it learns the publicly-routable VPN address of CSR9 via the "responder address extension", this is not relevant for routing in this case.

```
! CSR9
NHRP: Receive Resolution Reply via Tunnel100 vrf 0, packet size: 196
(F) afn: AF_IP(1), type: IPv6(86DD), hop: 255, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 196 extoff: 96
(M) flags: "router auth dst-stable unique src-stable nat ", reqid: 12
    src NBMA: 10.7.10.10
    src protocol: FE80::10, dst protocol: FE80::9
(C-1) code: no error(0)
    prefix: 128, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 16, pref: 0
    client NBMA: 10.7.9.9
    client protocol: FE80::9
Responder Address Extension(3):
(C) code: no error(0)
    prefix: 128, mtu: 9976, hd_time: 7200
```

```
addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 16, pref: 0
client NBMA: 10.7.9.9
client protocol: 2001:10:0:100::9
```

```
CSR10#show ipv6 nhrp FE80::9/128
FE80::9/128 via FE80::9
Tunnel100 created 00:00:54, expire 01:59:05
Type: dynamic, Flags: router nhop
NBMA address: 10.7.9.9
```

Traceroute confirms direct spoke-to-spoke connectivity.

```
CSR1#traceroute ipv6
Target IPv6 address: ::14:14:14:14
Source address: ::1:1:1:1
Type escape sequence to abort.
Tracing the route to ::14:14:14:14

 1 2001:10:1:10::10 1 msec 0 msec 1 msec
 2 2001:10:0:100::9 2 msec 1 msec 4 msec
 3 ::14:14:14:14 20 msec 13 msec 17 msec
```

Additional Reading – Reference configurations "dmvpn2"

35.2.3 Phase 3

DMVPN phase 3 achieves greater scalability than both phases 1 and 2, as mentioned briefly earlier. In phase 2, spokes still need the routes from every other spoke since the VPN next-hops must be preserved. Even if each node can summarize its back-end networks into a single summary, this could still mean thousands of routes in a large enterprise deployment with thousands of spokes. With phase 3, the hubs can originate default routes, which directs all traffic to them initially. Using new NHRP redirect messages (like ICMP redirects), the hubs can redirect follow-on packets to spokes directly. Spokes can install CEF shortcuts for one another's NBMA addresses to provide spoke-to-spoke connectivity. Hubs can also be stacked hierarchically to regionalize the DMVPN architecture, which is not examined here. The configuration requires one new command on hubs and one new command on spokes. Also, EIGRP still needs split-horizon disabled but the next-hops should be changed to be the hub (next-hop-self). To demonstrate the scalability improvements, as well as new NHRP route types, we will originate coarse aggregate routes from the hubs (2001::/16). The reason for using these is so that the enterprise WAN traffic (2001::/16) is kept inside the DMVPN, while default traffic (::/0) goes to the SP without encapsulation via existing default routes. OSPF P2MP network type should be used to allow the next-hops to be the hubs. This makes the phase 3 configuration a hybrid between phase 1 and phase 2 with respect to NHRP and IGP configurations.

```
! General NHS (hub)
interface Tunnel100
description DMVPN PHASE 3 - HUB
```



```

ip nhrp authentication NHRPAUTH
ip nhrp map multicast dynamic
ip nhrp network-id 100
ip nhrp server-only
ip nhrp redirect
ip ospf network point-to-multipoint
ip ospf 1 area 0
ipv6 nhrp authentication NHRPAUTH
ipv6 nhrp map multicast dynamic
ipv6 nhrp network-id 100
ipv6 nhrp redirect

router eigrp VPN
address-family ipv6 unicast autonomous-system 65000
af-interface Tunnel100
no split-horizon
next-hop-self
summary-address 2001::/16

! General NHS client (spoke)
interface Tunnel100
description DMVPN PHASE 3 - SPOKE
ip nhrp authentication NHRPAUTH
ip nhrp map 10.0.100.4 10.4.12.4
ip nhrp map multicast 10.4.12.4
ip nhrp map 10.0.100.5 10.5.6.5
ip nhrp map multicast 10.5.6.5
ip nhrp network-id 100
ip nhrp nhs 10.0.100.4
ip nhrp nhs 10.0.100.5
ip nhrp shortcut
ip ospf network point-to-multipoint
ip ospf 1 area 0
ipv6 nhrp authentication NHRPAUTH
ipv6 nhrp map 2001:10:0:100::4/128 10.4.12.4
ipv6 nhrp map 2001:10:0:100::5/128 10.5.6.5
ipv6 nhrp map multicast 10.4.12.4
ipv6 nhrp map multicast 10.5.6.5
ipv6 nhrp network-id 100
ipv6 nhrp nhs 2001:10:0:100::4 priority 5
ipv6 nhrp nhs 2001:10:0:100::5 priority 2
ipv6 nhrp shortcut

```

We quick check the hubs to ensure the NHRP registrations completed successfully. This is identical to other DMVPN phases. We also quickly test for OSPF/EIGRP neighbors.

```

CSR4#show ip nhrp brief

```

Target	Via	NBMA	Mode	Intfc	Claimed
10.0.100.2/32	10.0.100.2	10.8.2.2	dynamic	Tu100	< >

10.0.100.5/32	10.0.100.5	10.5.6.5	dynamic	Tu100	<	>
10.0.100.9/32	10.0.100.9	10.7.9.9	dynamic	Tu100	<	>
10.0.100.10/32	10.0.100.10	10.7.10.10	dynamic	Tu100	<	>

CSR5#show ip nhrp brief

Target	Via	NBMA	Mode	Intfc	Claimed
10.0.100.2/32	10.0.100.2	10.8.2.2	dynamic	Tu100	< >
10.0.100.4/32	10.0.100.4	10.4.12.4	static	Tu100	< >
10.0.100.9/32	10.0.100.9	10.7.9.9	dynamic	Tu100	< >
10.0.100.10/32	10.0.100.10	10.7.10.10	dynamic	Tu100	< >

CSR4#show ip ospf neighbor

Neighbor ID	Pri	State	Dead Time	Address	Interface
10.10.10.10	0	FULL/ -	00:01:36	10.0.100.10	Tunnel100
10.9.9.9	0	FULL/ -	00:01:36	10.0.100.9	Tunnel100
2.2.2.2	0	FULL/ -	00:01:53	10.0.100.2	Tunnel100
5.5.5.5	0	FULL/ -	00:01:39	10.0.100.5	Tunnel100

CSR5#show ip ospf neighbor

Neighbor ID	Pri	State	Dead Time	Address	Interface
10.10.10.10	0	FULL/ -	00:01:54	10.0.100.10	Tunnel100
10.9.9.9	0	FULL/ -	00:01:53	10.0.100.9	Tunnel100
2.2.2.2	0	FULL/ -	00:01:42	10.0.100.2	Tunnel100
4.4.4.4	0	FULL/ -	00:01:52	10.0.100.4	Tunnel100

CSR4#show eigrp address-family ipv6 neighbors

EIGRP-IPv6 VR(VPN) Address-Family Neighbors for AS(65000)

H	Address	Interface	Hold (sec)	Uptime	SRTT (ms)	RTO	Q Cnt	Seq Num
4	Link-local address: FE80::5	Tu100	10	00:01:48	190	5000	0	205
2	Link-local address: FE80::9	Tu100	12	00:02:41	133	5000	0	123
1	Link-local address: FE80::2	Tu100	13	00:02:41	170	5000	0	151
0	Link-local address: FE80::10	Tu100	12	00:02:41	204	5000	0	121
3	Link-local address: FE80::5	Tu45	13	1d21h	77	1476	0	206

CSR5#show eigrp address-family ipv6 neighbors

EIGRP-IPv6 VR(VPN) Address-Family Neighbors for AS(65000)

H	Address	Interface	Hold (sec)	Uptime	SRTT (ms)	RTO	Q Cnt	Seq Num
4	Link-local address: FE80::9	Tu100	14	00:01:59	102	5000	0	123
2	Link-local address: FE80::2	Tu100	11	00:01:59	113	5000	0	153
1	Link-local address: FE80::10	Tu100	14	00:01:59	84	5000	0	124
0	Link-local address: FE80::4	Tu100	14	00:02:00	139	5000	0	197

```
3   Link-local address:      Tu45          10 1d21h      21  5000  0  198
   FE80::4
```

The link-state nature of OSPF makes hub summarization impossible, so each spoke still has OSPF routes for each remote spoke since they share a common area. For this reason (and others), OSPF is generally a poor choice of IGP over DMVPN. For IPv6 routes learned via EIGRP, every spoke only has a pair of 2001::/16 routes towards the hubs. My original IPv6 addressing plan was poor in that the remote networks (loopbacks) did not summarize well; they have been updated to reflect good EIGRP design using the format 2001:x:x:x:x/128 (not shown). In this way, each spoke will have a very small number of routes regardless of how many spokes are added to the network.

```
CSR9#show ip route ospf | begin ^O
O IA      1.1.1.0 [110/2002] via 10.0.100.5, 00:15:46, Tunnel100
          [110/2002] via 10.0.100.4, 00:15:46, Tunnel100
          2.0.0.0/32 is subnetted, 1 subnets
O          2.2.2.2 [110/2001] via 10.0.100.5, 00:15:56, Tunnel100
          [110/2001] via 10.0.100.4, 00:15:56, Tunnel100
          4.0.0.0/32 is subnetted, 1 subnets
O          4.4.4.4 [110/1001] via 10.0.100.4, 02:55:26, Tunnel100
          5.0.0.0/32 is subnetted, 1 subnets
O          5.5.5.5 [110/1001] via 10.0.100.5, 1d19h, Tunnel100
          10.0.0.0/8 is variably subnetted, 12 subnets, 2 masks
O          10.0.100.2/32 [110/2000] via 10.0.100.5, 00:15:56, Tunnel100
          [110/2000] via 10.0.100.4, 00:15:56, Tunnel100

[snip]
```

```
CSR9#show ipv6 route eigrp | begin ^D
D   2001::/16 [90/76800640]
    via FE80::4, Tunnel100
    via FE80::5, Tunnel100
D   2001:14:14:14::14/128 [90/10752]
    via FE80::14, GigabitEthernet2.594
```

We will send traffic from CSR1's to CSR2's loopback using IPv4. Unlike phase 2, CSR10 does have a valid IP route for 2.2.2.2/32 via CSR4 and CSR5, implying that CSR10 does not send a resolution request to the NHS. To determine which coarse aggregate it uses for the initial packet, we check the FIB. CSR10 will encapsulate this packet and send it to CSR5 according to the output below. Including the "virtual" keyword will show the tunnel interface information (VPN address) in addition to the tunnel destination (NBMA address).

```
CSR10#show ip cef exact-route virtual 1.1.1.1 2.2.2.2
1.1.1.1 -> 2.2.2.2 => IP midchain out of Tunnel100, addr 10.0.100.5
7FE069004A78 => IP adj out of GigabitEthernet2.570, addr 10.7.10.7
```

When CSR5 receives this packet, it routes it normally out of the tunnel (hair-pin) and tells CSR10 "I will forward this packet, but you should send traffic straight to CSR2 in the future". This is what the NHRP

redirect message does. Although the message type is “Traffic Indication”, the code identified in the mandatory (M) field is “redirect”. Notice that it does not contain CSR2's NBMA address anywhere in the packet; this is just an indication for CSR10 to originate a traditional NHRP resolution request to resolve CSR2's NBMA. This packet goes from CSR5 directly back towards 1.1.1.1/32, where CSR10 is the DMVPN endpoint servicing that host.

```
! CSR5
NHRP: Attempting to Redirect, remote_nbma:10.7.10.10, dst:2.2.2.2
NHRP: inserting (10.7.10.10/2.2.2.2) in redirect table
NHRP: Attempting to send packet through interface Tunnel100 via DEST dst
1.1.1.1
NHRP: Send Traffic Indication via Tunnel100 vrf 0, packet size: 120
src: 10.0.100.5, dst: 1.1.1.1
(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 120 extoff: 88
(M) traffic code: redirect(0)
    src NBMA: 10.5.6.5
    src protocol: 10.0.100.5, dst protocol: 1.1.1.1
    Contents of nhrp traffic indication packet:
        45 00 00 64 01 8A 00 00 FD 01 B6 09 01 01 01 01
        02 02 02 02 08 00 B6 C8 00 27 00 00 00 00 00 00
        8A 46 3D 14 AB CD AB CD AB CD AB CD AB CD AB
Forward Transit NHS Record Extension(4):
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
    type:Cleartext(1), data:NHRPAUTH
NAT address Extension(9):
NHRP: Switching directly using pre-set NBMA 10.7.10.10
NHRP: Encapsulation succeeded. Sending NHRP Control Packet NBMA Address:
10.7.10.10
NHRP: 144 bytes out Tunnel100
```

CSR10 receives the traffic indication message (redirect) and attempts to resolve the NBMA mapping for 2.2.2.2. Unlike phase 2, phase 3 resolves NBMA/VPN mappings where the VPN address is the final destination, not the actual VPN address/next-hop. A poorly summarized network could result in a huge NHRP cache in this case; the reason the VPN address itself is not resolved is because the hub summarization has hidden the NBMA topology. The NBMA topology and IGP routing are totally decoupled with DMVPN phase 3, which allows it to scale (assuming IGP summarizes well). The resolution request contains the usual three parameters: source NBMA, source VPN, and destination VPN (final destination, in this case). This is sent towards CSR5, which is the NHS that originated the redirect.

```
! CSR10
NHRP: Receive Traffic Indication via Tunnel100 vrf 0, packet size: 120
(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
```

```

    pktsz: 120 extoff: 88
(M) traffic code: redirect(0)
    src NBMA: 10.5.6.5
    src protocol: 10.0.100.5, dst protocol: 1.1.1.1
    Contents of nhrp traffic indication packet:
        45 00 00 64 01 8A 00 00 FD 01 B6 09 01 01 01 01
        02 02 02 02 08 00 B6 C8 00 27 00 00 00 00 00 00
        8A 46 3D 14 AB CD AB CD AB CD AB CD AB CD AB
Forward Transit NHS Record Extension(4):
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
    type:Cleartext(1), data:NHRPAUTH
NAT address Extension(9):
NHRP: Enqueued NHRP Resolution Request for destination: 2.2.2.2
NHRP: Sending NHRP Resolution Request for dest: 2.2.2.2 to nexthop: 2.2.2.2
using our src: 10.0.100.10
NHRP: Attempting to send packet through interface Tunnel100 via DEST dst
2.2.2.2
NHRP: Send Resolution Request via Tunnel100 vrf 0, packet size: 88
    src: 10.0.100.10, dst: 2.2.2.2
    (F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
        shtl: 4(NSAP), sstl: 0(NSAP)
        pktsz: 88 extoff: 52
    (M) flags: "router auth src-stable nat ", reqid: 13
        src NBMA: 10.7.10.10
        src protocol: 10.0.100.10, dst protocol: 2.2.2.2
    (C-1) code: no error(0)
        prefix: 32, mtu: 9976, hd_time: 7200
        addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
Responder Address Extension(3):
Forward Transit NHS Record Extension(4):
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
    type:Cleartext(1), data:NHRPAUTH
NAT address Extension(9):
NHRP: NHRP successfully mapped '10.0.100.5' to NBMA 10.5.6.5
NHRP: Encapsulation succeeded. Sending NHRP Control Packet NBMA Address:
10.5.6.5
NHRP: 112 bytes out Tunnel100

```

CSR5 receives the resolution request, performs a route lookup, and forwards it back out of the tunnel towards CSR2. CSR5 inserts the "forward transit NHS record" into the packet so CSR2 knows which NHS forwarded the message. As an NHS, CSR5 knows the NBMA address of CSR2, which is 10.8.2.2. This is the first time we have seen that address in any debug message. As expected, the NHS does not respond with this directly to CSR10, but forwards the request towards the final destination, 2.2.2.2/32.

```

! CSR5
NHRP: Receive Resolution Request via Tunnel100 vrf 0, packet size: 88

```

```

(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 88 extoff: 52
(M) flags: "router auth src-stable nat ", reqid: 13
    src NBMA: 10.7.10.10
    src protocol: 10.0.100.10, dst protocol: 2.2.2.2
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
Responder Address Extension(3):
Forward Transit NHS Record Extension(4):
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
    type:Cleartext(1), data:NHRPAUTH
NAT address Extension(9):
NHRP: Route lookup for destination 2.2.2.2 in (0x0) yielded interface
Tunnel100, prefixlen 32
NHRP-ATTR: In nhrp_rcv_resolution_request NHRP Resolution Request packet is
forwarded to 2.2.2.2.
NHRP: Forwarding: NHRP SAS picked source: 10.0.100.5 for destination: 2.2.2.2
NHRP: Attempting to send packet through interface Tunnel100 via DEST dst
2.2.2.2
NHRP: Forwarding Resolution Request via Tunnel100 vrf 0, packet size: 108
src: 10.0.100.5, dst: 2.2.2.2
(F) afn: AF_IP(1), type: IP(800), hop: 254, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 108 extoff: 52
(M) flags: "router auth src-stable nat ", reqid: 13
    src NBMA: 10.7.10.10
    src protocol: 10.0.100.10, dst protocol: 2.2.2.2
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
Responder Address Extension(3):
Forward Transit NHS Record Extension(4):
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.5.6.5
    client protocol: 10.0.100.5
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
    type:Cleartext(1), data:NHRPAUTH
NAT address Extension(9):
NHRP: NHRP successfully mapped '10.0.100.2' to NBMA 10.8.2.2
NHRP: Encapsulation succeeded. Sending NHRP Control Packet NBMA Address:
10.8.2.2
NHRP: 132 bytes out Tunnel100

```

CSR2 receives the resolution request from CSR10 via the NHS CSR5. Although the packet isn't directed at the VPN address, CSR2 acknowledges that it is the "egress" router so it processes, and replies to, the received request. Inside the "responder address" field, CSR2 returns its NBMA address, which is the fundamental problem NHRP is meant to solve. Although CSR2 does know the NBMA address of CSR10 (the source), it sends its resolution reply to the NHS on the way back. I presume that the reason the NHS is used on the way back is because the route back to the "source protocol" isn't "connected" anymore, so NHS does not make assumptions about its whereabouts (even though it technically could, given the source NBMA address).

```
! CSR2
NHRP: Receive Resolution Request via Tunnel100 vrf 0, packet size: 108
  (F) afn: AF_IP(1), type: IP(800), hop: 254, ver: 1
      shtl: 4(NSAP), sstl: 0(NSAP)
      pktsz: 108 extoff: 52
  (M) flags: "router auth src-stable nat ", reqid: 13
      src NBMA: 10.7.10.10
      src protocol: 10.0.100.10, dst protocol: 2.2.2.2
  (C-1) code: no error(0)
      prefix: 32, mtu: 9976, hd_time: 7200
      addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 0, pref: 0
Responder Address Extension(3):
Forward Transit NHS Record Extension(4):
  (C-1) code: no error(0)
      prefix: 32, mtu: 9976, hd_time: 7200
      addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
      client NBMA: 10.5.6.5
      client protocol: 10.0.100.5
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
  type: Cleartext(1), data: NHRPAUTH
NAT address Extension(9):
NHRP: Route lookup for destination 2.2.2.2 in (0x0) yielded interface
Loopback0, prefixlen 32
NHRP: We are egress router. Process the NHRP Resolution Request.
NHRP: Tunnel100: Cache add for target 10.0.100.10/32 next-hop 10.0.100.10
10.7.10.10
NHRP: Adding Tunnel Endpoints (VPN: 10.0.100.10, NBMA: 10.7.10.10)
NHRP: Successfully attached NHRP subblock for Tunnel Endpoints (VPN:
10.0.100.10, NBMA: 10.7.10.10)
NHRP: Inserted subblock node for cache: Target 10.0.100.10/32 nhop
10.0.100.10
NHRP: Converted internal dynamic cache entry for 10.0.100.10/32 interface
Tunnel100 to external
NHRP: Tunnel100: Internal Cache add for target 2.2.2.2/32 next-hop 10.0.100.2
10.8.2.2
NHRP: Attempting to send packet through interface Tunnel100 via DEST dst
10.0.100.10
NHRP: Send Resolution Reply via Tunnel100 vrf 0, packet size: 136
```

```

src: 10.0.100.2, dst: 10.0.100.10
(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 136 extoff: 60
(M) flags: "router auth dst-stable unique src-stable nat ", reqid: 13
    src NBMA: 10.7.10.10
    src protocol: 10.0.100.10, dst protocol: 2.2.2.2
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.8.2.2
    client protocol: 10.0.100.2
Responder Address Extension(3):
(C) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.8.2.2
    client protocol: 10.0.100.2
Forward Transit NHS Record Extension(4):
(C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.5.6.5
    client protocol: 10.0.100.5
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
    type: Cleartext(1), data: NHRPAUTH
NAT address Extension(9):
NHRP: No NHRP subblock found in packet
NHRP: nhrp_ifcache: Avl Root: 7F52F862AF28
NHRP: NHRP successfully mapped '10.0.100.5' to NBMA 10.5.6.5
NHRP: Encapsulation succeeded. Sending NHRP Control Packet NBMA Address:
10.5.6.5
NHRP: 160 bytes out Tunnel100

```

When CSR5 receives the resolution reply from CSR2, it adds the "reverse transit NHS" field to show that the reply transited through this NHS. CSR5 was used for both the forward and reverse legs of the resolution, so its information appears in both fields.

```

! CSR5
NHRP: Receive Resolution Reply via Tunnel100 vrf 0, packet size: 136
(F) afn: AF_IP(1), type: IP(800), hop: 255, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 136 extoff: 60
(M) flags: "router auth dst-stable unique src-stable nat ", reqid: 13
    src NBMA: 10.7.10.10
    src protocol: 10.0.100.10, dst protocol: 2.2.2.2
(C-1) code: no error(0)

```



```

    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.8.2.2
    client protocol: 10.0.100.2
Responder Address Extension(3):
  (C) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.8.2.2
    client protocol: 10.0.100.2
Forward Transit NHS Record Extension(4):
  (C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.5.6.5
    client protocol: 10.0.100.5
Reverse Transit NHS Record Extension(5):
Authentication Extension(7):
  type: Cleartext(1), data: NHRPAUTH
NAT address Extension(9):
NHRP: netid_in = 0, to_us = 0
NHRP: No NHRP subblock found in packet
NHRP: Forwarding Resolution Reply to 10.0.100.10
NHRP: Attempting to forward to destination: 10.0.100.10
NHRP-MPLS: tableid: 0 vrf:
NHRP: nhrp_ifcache: Avl Root: 7F6270A9FFF0
NHRP: Forwarding: NHRP SAS picked source: 10.0.100.5 for destination:
10.0.100.10
NHRP: Attempting to send packet through interface Tunnel100 via DEST dst
10.0.100.10
NHRP: Forwarding Resolution Reply via Tunnel100 vrf 0, packet size: 156
src: 10.0.100.5, dst: 10.0.100.10
  (F) afn: AF_IP(1), type: IP(800), hop: 254, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 156 extoff: 60
  (M) flags: "router auth dst-stable unique src-stable nat ", reqid: 13
    src NBMA: 10.7.10.10
    src protocol: 10.0.100.10, dst protocol: 2.2.2.2
  (C-1) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.8.2.2
    client protocol: 10.0.100.2
Responder Address Extension(3):
  (C) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 10.8.2.2
    client protocol: 10.0.100.2

```

```

Forward Transit NHS Record Extension(4):
(C-1) code: no error(0)
      prefix: 32, mtu: 9976, hd_time: 7200
      addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
      client NBMA: 10.5.6.5
      client protocol: 10.0.100.5
Reverse Transit NHS Record Extension(5):
(C-1) code: no error(0)
      prefix: 32, mtu: 9976, hd_time: 7200
      addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
      client NBMA: 10.5.6.5
      client protocol: 10.0.100.5
Authentication Extension(7):
      type: Cleartext(1), data: NHRPAUTH
NAT address Extension(9):
NHRP: nhrp_ifcache: Avl Root: 7F6270A9FFF0
NHRP: No NHRP subblock found in packet
NHRP: nhrp_ifcache: Avl Root: 7F6270A9FFF0
NHRP: NHRP successfully mapped '10.0.100.10' to NBMA 10.7.10.10
NHRP: Encapsulation succeeded. Sending NHRP Control Packet NBMA Address:
10.7.10.10
NHRP: 180 bytes out Tunnel100

```

CSR10 receives the resolution reply, processes it, and adds an NHRP mapping for 2.2.2.2/32 to the NHRP cache. NHRP tries to add a new NHRP route to the RIB with AD 250 for 2.2.2.2/32. Since there is already an exact-match OSPF route with AD 110, the next-hop-override (NHO) process is triggered instead. This is a function of the CEF shortcut; we know the VPN next-hop for 2.2.2.2/32 is 10.0.100.2, so rather than use the hubs as the IGP next-hops, the router overwrites them. Ultimately, the FIB is augmented by the NHRP cache in this way. The routing table displays these entries with "%" symbols. The VPN address of CSR2 is also marked as an "nhop" and the final destination is not, which is self-explanatory.

```

! CSR10
NHRP: Receive Resolution Reply via Tunnel100 vrf 0, packet size: 156
(F) afn: AF_IP(1), type: IP(800), hop: 254, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 156 extoff: 60
(M) flags: "router auth dst-stable unique src-stable nat ", reqid: 13
    src NBMA: 10.7.10.10
    src protocol: 10.0.100.10, dst protocol: 2.2.2.2
(C-1) code: no error(0)
      prefix: 32, mtu: 9976, hd_time: 7200
      addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
      client NBMA: 10.8.2.2
      client protocol: 10.0.100.2
Responder Address Extension(3):
(C) code: no error(0)
    prefix: 32, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0

```

```

client NBMA: 10.8.2.2
client protocol: 10.0.100.2
Forward Transit NHS Record Extension(4):
(C-1) code: no error(0)
      prefix: 32, mtu: 9976, hd_time: 7200
      addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
      client NBMA: 10.5.6.5
      client protocol: 10.0.100.5
Reverse Transit NHS Record Extension(5):
(C-1) code: no error(0)
      prefix: 32, mtu: 9976, hd_time: 7200
      addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
      client NBMA: 10.5.6.5
      client protocol: 10.0.100.5
Authentication Extension(7):
  type: Cleartext(1), data: NHRPAUTH
NAT address Extension(9):
NHRP: netid_in = 0, to_us = 1
NHRP: This NHRP Resolution Reply is a forwarded packet
NHRP: Tunnel100: Cache update for target 2.2.2.2/32 next-hop 10.0.100.2
      10.8.2.2
NHRP: Adding Tunnel Endpoints (VPN: 10.0.100.2, NBMA: 10.8.2.2)
NHRP: Inserted subblock node for cache: Target 2.2.2.2/32 nhop 10.0.100.2
NHRP: Converted internal dynamic cache entry for 2.2.2.2/32 interface
Tunnel100 to external
NHRP: Adding route entry for 2.2.2.2/32 (Tunnel100) to RIB
NHRP: Route addition failed (admin-distance)
NHRP: nexthop-override added to RIB
NHRP: Adding route entry for 10.0.100.2/32 (Tunnel100) to RIB
NHRP: Route addition failed (admin-distance)
NHRP: nexthop-override added to RIB

CSR10#show ip cef 2.2.2.2
2.2.2.2/32
  nexthop 10.0.100.2 Tunnel100

CSR10#show ip route | include 2.2.2.2
O    %    2.2.2.2 [110/2001] via 10.0.100.5, 03:44:13, Tunnel100

CSR10#show ip route next-hop-override 2.2.2.2
Routing entry for 2.2.2.2/32
  Known via "ospf 1", distance 110, metric 2001, type intra area
  Redistributing via nhrp
  Last update from 10.0.100.4 on Tunnel100, 03:22:50 ago
  Routing Descriptor Blocks:
  * 10.0.100.5, from 2.2.2.2, 03:22:50 ago, via Tunnel100
    Route metric is 2001, traffic share count is 1
  10.0.100.4, from 2.2.2.2, 03:22:50 ago, via Tunnel100
    Route metric is 2001, traffic share count is 1

```

```
[NHO]10.0.100.2, from 10.0.100.2, 00:20:01 ago, via Tunnel100
Route metric is 1, traffic share count is 1
MPLS label: none
```

```
CSR10#show ip nhrp shortcut
2.2.2.2/32 via 10.0.100.2
Tunnel100 created 00:20:20, expire 01:39:39
Type: dynamic, Flags: router rib nho
NBMA address: 10.8.2.2
10.0.100.2/32 via 10.0.100.2
Tunnel100 created 00:20:20, expire 01:39:39
Type: dynamic, Flags: router nhop rib nho
NBMA address: 10.8.2.2
```

Unlike phase 2, there isn't an "implicit" NHRP entry on the remote side router. Even though CSR2 showed adding the tunnel endpoints from the original request, the redirect process still occurs. The entire process occurs again in the opposite direction so that CSR2 knows how to reach 1.1.1.1/32. Most of the debugs are not shown again, but there is one significant difference. On CSR10, the route 2.2.2.2/32 could not be added to the RIB because the OSPF route (AD 110) was better than the NHRP route (AD 250). Since CSR2 never had a route for the exact prefix 1.1.1.1/32, it is able to add this longer match. CSR2 only had a summary for 1.1.1.0/24, which was enough for routing to work, but means the NHO process need not occur. The NHRP entry does not specify "nho" for the 1.1.1.1/32 entry, but does for the VPN address of CSR10, which is expected. New NHRP routes are denoted by "H" in the routing table.

```
! CSR2
NHRP: Adding route entry for 1.1.1.1/32 (Tunnel100) to RIB
NHRP: Route addition to RIB Successful
```

```
CSR2#show ip cef 1.1.1.1
1.1.1.1/32
nexthop 10.0.100.10 Tunnel100
```

```
CSR2#show ip route 1.1.1.1
Routing entry for 1.1.1.1/32
Known via "nhrp", distance 250, metric 1
Last update from 10.0.100.10 on Tunnel100, 00:45:24 ago
Routing Descriptor Blocks:
* 10.0.100.10, from 10.0.100.10, 00:45:24 ago, via Tunnel100
Route metric is 1, traffic share count is 1
MPLS label: none
```

```
CSR2#show ip route nhrp | begin Gateway
Gateway of last resort is 10.2.8.8 to network 0.0.0.0
```

```
1.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
H 1.1.1.1/32 [250/1] via 10.0.100.10, 00:45:42, Tunnel100
```

```

CSR2#show ip nhrp shortcut
1.1.1.1/32 via 10.0.100.10
    Tunnel100 created 00:45:13, expire 01:14:46
    Type: dynamic, Flags: router rib
    NBMA address: 10.7.10.10
10.0.100.10/32 via 10.0.100.10
    Tunnel100 created 00:45:13, expire 01:14:46
    Type: dynamic, Flags: router nhop rib nho
    NBMA address: 10.7.10.10

```

The process for IPv6 is identical. Because phase 3 makes no assumptions about the VPN topology, having coarse aggregates pointing to the hub will always work. Before sending any traffic from CSR1 to CSR2, CSR10 only has aggregates for 2001::/16. This kind of summarization means that NHO will be non-existent for real destinations since there is never an exact-match route already known by the spokes. Before sending any traffic between spokes, CSR10 only has the hub-generated aggregate routes.

```

CSR10#show ipv6 cef 2001:2:2:2::2
2001::/16
    nexthop FE80::4 Tunnel100
    nexthop FE80::5 Tunnel100

```

When CSR10 receives IPv6 traffic from CSR1 towards CSR2, it initially sends the traffic along the regular routing path because it knows the NBMA address for both NHS', per the RIB and static NHRP mappings. Upon receipt, the hub returns the NHRP redirect message towards the sender, forwards the original packet back out of the tunnel, and expects to receive NHRP resolution from the sender soon. Because the process happens in both directions (no "implicit" mappings), the NHS tracks the redirects for a short time until they expire. After expiration, they can be regenerated if the spoke resolution requests fail for whatever reason. Below shows pending redirects from both CSR10 and CSR2 with 6 seconds left until they expire.

```

CSR4#show ipv6 nhrp redirect

```

I/F	NBMA address	Destination	Drop Count	Expiry
Tunnel100	10.7.10.10	2001:2:2:2::2	1	00:00:06
Tunnel100	10.8.2.2	2001:1:1:1::1	1	00:00:06

Upon receipt of the redirect message, CSR10 begins the usual phase 3 resolution process. In short, the process is as follows: CSR10 sends traffic to hub, hub sends redirect to CSR10, and then CSR10 issues a resolution request to follow the normal NHRP process. For these resolutions, the NHRP resolution reply does not transit the NHS and is sent directly back from the target spoke to the originating spoke. I speculate that this is different from the previous example because with OSPF P2MP, the routers had an OSPF /32 to each VPN endpoint where with EIGRPv6 they did not. Having the NHRP route (not an IGP route) to the VPN endpoint may enable DMVPN to bypass the NHS on the reverse path when sending the replies.

```

! CSR10
NHRP: Receive Resolution Reply via Tunnel100 vrf 0, packet size: 196
(F) afn: AF_IP(1), type: IPv6(86DD), hop: 255, ver: 1
    shtl: 4(NSAP), sstl: 0(NSAP)
    pktsz: 196 extoff: 96
(M) flags: "router auth dst-stable unique src-stable nat ", reqid: 19
    src NBMA: 10.7.10.10
    src protocol: 2001:10:0:100::10, dst protocol: 2001:2:2:2::2
(C-1) code: no error(0)
    prefix: 128, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 16, pref: 0
    client NBMA: 10.8.2.2
    client protocol: 2001:10:0:100::2
Responder Address Extension(3):
(C) code: no error(0)
    prefix: 128, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 16, pref: 0
    client NBMA: 10.8.2.2
    client protocol: 2001:10:0:100::2
Forward Transit NHS Record Extension(4):
(C-1) code: no error(0)
    prefix: 128, mtu: 9976, hd_time: 7200
    addr_len: 4(NSAP), subaddr_len: 0(NSAP), proto_len: 16, pref: 0
    client NBMA: 10.4.12.4
    client protocol: 2001:10:0:100::4
Reverse Transit NHS Record Extension(5):
[no output]

```

When the resolution completes, CSR10 has specific CEF entries for CSR2's loopback, which was the target address inside the VPN. The following outputs mirror the IPv4 outputs seen earlier for other tests. The "rib" flag signifies that there are exact-match routes for these entries in the RIB, but they don't necessarily have to be H-routes. If "rib" is present and "nho" is absent, the route is surely an H route. When both are present, the route is a non-H route; NHO and H-routes are mutually exclusive within the context of a single prefix.

```

CSR10#show ipv6 cef 2001:2:2:2::2
2001:2:2:2::2/128
    nexthop 2001:10:0:100::2 Tunnel100

CSR10#show ipv6 route 2001:2:2:2::2/128
Routing entry for 2001:2:2:2::2/128
    Known via "NHRP-IPv6", distance 250, metric 1
    Route count is 1/1, share count 0
    Routing paths:
        2001:10:0:100::2, Tunnel100
        Last updated 00:01:10 ago

```

```

CSR10#show ipv6 route nhrp | begin ^H
H   2001:2:2:2::2/128 [250/1]
    via 2001:10:0:100::2, Tunnel100
H   2001:10:0:100::2/128 [250/1]
    via 2001:10:0:100::2, Tunnel100

CSR10#show ipv6 nhrp shortcut
2001:2:2:2::2/128 via 2001:10:0:100::2
    Tunnel100 created 00:00:45, expire 01:59:14
    Type: dynamic, Flags: router rib
    NBMA address: 10.8.2.2
2001:10:0:100::2/128 via 2001:10:0:100::2
    Tunnel100 created 00:00:45, expire 01:59:14
    Type: dynamic, Flags: router used nhop rib
    NBMA address: 10.8.2.2

```

For completeness, the verification on CSR2 is shown below. As discussed earlier, the phase 3 resolution process happens in the reverse direction assuming traffic needs to flow bidirectionally.

```

CSR2#show ipv6 cef 2001:1:1:1::1
2001:1:1:1::1/128
    nexthop 2001:10:0:100::10 Tunnel100

CSR2#show ipv6 route 2001:1:1:1::1/128
Routing entry for 2001:1:1:1::1/128
    Known via "NHRP-IPv6", distance 250, metric 1
    Route count is 1/1, share count 0
    Routing paths:
        2001:10:0:100::10, Tunnel100
        Last updated 00:08:16 ago

CSR2#show ipv6 route nhrp | begin ^H
H   2001:1:1:1::1/128 [250/1]
    via 2001:10:0:100::10, Tunnel100
H   2001:10:0:100::10/128 [250/1]
    via 2001:10:0:100::10, Tunnel100

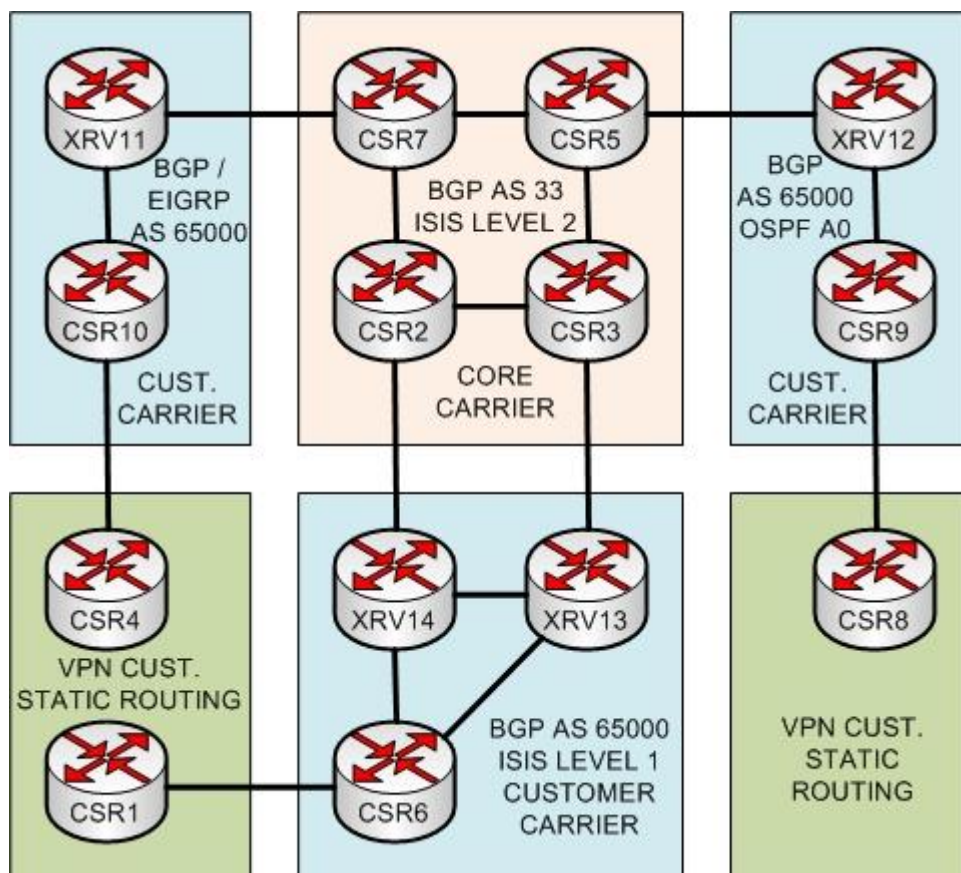
CSR2#show ipv6 nhrp shortcut
2001:1:1:1::1/128 via 2001:10:0:100::10
    Tunnel100 created 00:08:26, expire 01:51:33
    Type: dynamic, Flags: router rib
    NBMA address: 10.7.10.10
2001:10:0:100::10/128 via 2001:10:0:100::10
    Tunnel100 created 00:08:26, expire 01:51:33
    Type: dynamic, Flags: router nhop rib
    NBMA address: 10.7.10.10

```

Additional Reading – Reference configurations "dmvpn3"

35.3 mGRE-based L3VPN

The L3VPN-over-mGRE feature is used to dynamically connect customers within remote provider sites together. Imagine a CSC-like scenario where the core carrier refuses to provide a labeled-unicast service across two customer carrier islands. The connectivity between the customer carriers is IP only, which means MPLS transport doesn't work automatically. The L3VPNNoGRE feature fully automates the tunnel endpoint discovery and maintains the BGP VPNv4/v6 allocated labels between customer carrier sites. The feature also supports the MVPN profile 0 (PIM/GRE) implementation for efficient multicast transport; this is sensible since profile 0 works well over CSC architectures. The network used for this test is quite complex; it is a CSC scenario where the core carrier provides an IP-only VPN to three customer carrier sites. The core carrier uses mLDP to transport LSM for variety since the core and customer carriers can use entirely different MVPN techniques. The customer carrier sites must use profile 0 since their CSC service is IP only. CSR6, CSR9, and CSR10 are the customer PE routers that service the true customers, CSR1, CSR4, and CSR8. The XrV routers don't support the dynamic L3VPNNoGRE feature, and are P routers within the customer carriers. Different IGP's are used in different regions for variety. Before continuing with this test, ensure you understand MVPN/CSC technologies and techniques.



The configuration is rather involved, so the initial verifications are limited to basic reachability. Within the core, CSR2 is the RR for VPNv4/v6 AFs using the BGP listen command (dynamic peer groups). We quickly verify that the neighbors are up and we receive at least some routes. VPNv6 in the core carrier was a test for other things and is not directly related to this test.

```
CSR2#show bgp vpnv4 unicast all summary | begin Neighbor
Neighbor      V          AS MsgRcvd MsgSent  TblVer  InQ  OutQ  Up/Down  State/PfxRc
*3.3.3.3       4           33     304    337     32    0    0 04:21:39      1
*5.5.5.5       4           33     307    335     32    0    0 04:24:47      1
*7.7.7.7       4           33     309    334     32    0    0 04:24:54      1
10.255.14.14   4          65000    250    284     32    0    0 04:06:26      1
* Dynamically created based on a listen range command
Dynamically created neighbors: 3, Subnet ranges: 1
```

```
BGP peergroup IBGP listen range group members:
0.0.0.0/0
```

Total dynamically created neighbors: 3/(100 max), Subnet ranges: 1

```
CSR2#show bgp vpnv6 unicast all summary | begin Neighbor
Neighbor      V          AS MsgRcvd MsgSent  TblVer  InQ  OutQ  Up/Down  State/PfxRc
*3.3.3.3       4           33     313    346     36    0    0 04:29:17      1
*5.5.5.5       4           33     316    344     36    0    0 04:32:25      1
*7.7.7.7       4           33     317    344     36    0    0 04:32:32      1
FD00:10:255:14::14
                4          65000    259    289     36    0    0 04:14:04      1
* Dynamically created based on a listen range command
Dynamically created neighbors: 3, Subnet ranges: 1
```

```
BGP peergroup IBGP listen range group members:
0.0.0.0/0
```

Total dynamically created neighbors: 3/(100 max), Subnet ranges: 1

We also check the BGP RR for the customer carrier, which is CSR6. Rather than check the BGP summary for the neighbors, I will check the BGP tables for VPNv4, VPNv6, and IPv4 MDT. We learn the remote two VPNv4/v6 routes (one from each remote PE), as well as the MDT sources for MVPN transport within the IPv4 MDT AFI.

```
CSR6#show bgp vpnv4 unicast all | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 65000:6 (default for vrf VPN)
*> 192.168.1.0      0.0.0.0          0          32768 ?
*>i 192.168.4.0      10.10.10.10       0        100    0 ?
*>i 192.168.8.0      9.9.9.9          0        100    0 ?
Route Distinguisher: 65000:9
*>i 192.168.8.0      9.9.9.9          0        100    0 ?
Route Distinguisher: 65000:10
*>i 192.168.4.0      10.10.10.10       0        100    0 ?
```

```

CSR6#show bgp vpnv6 unicast all | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 65000:6 (default for vrf VPN)
*>  FD00:192:168:1::/64
      ::              0              32768 ?
*>i  FD00:192:168:4::/64
      ::FFFF:10.10.10.10
      0      100      0 ?
*>i  FD00:192:168:8::/64
      ::FFFF:9.9.9.9      0      100      0 ?
Route Distinguisher: 65000:9
*>i  FD00:192:168:8::/64
      ::FFFF:9.9.9.9      0      100      0 ?

CSR6#show bgp ipv4 mdt all | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 65000:6 (default for vrf VPN)
*>  6.6.6.6/32      0.0.0.0
      0      100      0 ?
Route Distinguisher: 65000:9
*>i  9.9.9.9/32      9.9.9.9      0      100      0 ?
Route Distinguisher: 65000:10
*>i  10.10.10.10/32  10.10.10.10      0      100      0 ?
Route Distinguisher: 65000:10
*>i  FD00:192:168:4::/64
      ::FFFF:10.10.10.10
      0      100      0 ?

```

Given this, we know the BGP control plane is properly built and there is IP reachability between CSR6, CSR9, and CSR10. If there were not, the BGP sessions between customer carrier PEs would not work. The LSP tracing verification procedure does not work, because there is no LSP between these PEs, despite them trying to run VPNv4/v6. We receive a BGP label from the remote PE (CSR10), but we have no transport label. XRv2 and XRv4 do not allocate labels because 10.10.10.10/32 was a BGP learned prefix from their perspective, and the provider never allocated a label for it via labeled-unicast.

```

CSR6#show bgp vpnv4 unicast vrf VPN 192.168.4.0
BGP routing table entry for 65000:6:192.168.4.0/24, version 49
Paths: (1 available, best #1, table VPN)
  Not advertised to any peer
  Refresh Epoch 2
  Local, (Received from a RR-client), imported path from
65000:10:192.168.4.0/24 (global)
  10.10.10.10 (metric 10) (via default) from 10.10.10.10 (10.10.10.10)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  Extended Community: RT:65000:10
  Connector Attribute: count=1
  type 1 len 12 value 65000:10:10.10.10.10
  mpls labels in/out nlabel/10006

```

```
rx pathid: 0, tx pathid: 0x0
```

```
CSR6#show ip route 10.10.10.10
Routing entry for 10.10.10.10/32
  Known via "isis", distance 115, metric 10, type level-1
  Redistributing via isis 65000
  Last update from 10.6.13.13 on GigabitEthernet2.563, 02:59:24 ago
  Routing Descriptor Blocks:
  * 10.6.14.14, from 14.14.14.14, 02:59:24 ago, via GigabitEthernet2.564
    Route metric is 10, traffic share count is 1
  10.6.13.13, from 13.13.13.13, 02:59:24 ago, via GigabitEthernet2.563
    Route metric is 10, traffic share count is 1
```

```
CSR6#show mpls forwarding-table 10.10.10.10
Local      Outgoing   Prefix      Bytes Label  Outgoing   Next Hop
Label      Label      or Tunnel Id Switched     interface
6009       No Label   10.10.10.10/32  0           Gi2.563    10.6.13.13
           No Label   10.10.10.10/32  0           Gi2.564    10.6.14.14
```

The solution for this problem involves creating an L3VPN encapsulation profile. Here, we specify tunnel template parameters, somewhat similar to auto-tunnel configuration in TE. We can specify the tunnel protocol (only IPv4/GRE is currently supported) along with optional tunnel source and key. We then create a generalized route-map for both VPNv4 and VPNv6 that changes the next-hop on ingress. The logic is that when BGP receives a VPNv4/v6 prefix, it looks at the next-hop and creates a GRE tunnel endpoint from it. This alleviates the need for DMVPN-style signaling and allows spokes to communicate directly without extra signaling. BGP already knows all the endpoints since these are also the VPNv4/v6 next-hops. Even though only IPv4 is supported for the outer encapsulation at this time, VPNv6 via 6VPE can function using this feature, so we also adjust the IPv6 next-hop.

```
! CSR6, CSR9, CSR10
l3vpn encapsulation ip L3VPN_PROFILE
  transport ipv4 source Loopback0
  protocol gre key 65000

route-map RM_VPN_NHOP_IN permit 10
  set ip next-hop encapsulate l3vpn L3VPN_PROFILE
  set ipv6 next-hop encapsulate l3vpn L3VPN_PROFILE

router bgp 65000
  address-family vpnv4 unicast
    neighbor IBGP route-map RM_VPN_NHOP_IN in
  address-family vpnv6 unicast
    neighbor IBGP route-map RM_VPN_NHOP_IN in
```

After applying this configuration to the customer carrier PEs, the BGP path now shows reachability to 10.10.10.10 via Tunnel0. This is a dynamic entity that serves as a template, and the tunnel configuration reflects the configured source address and key. Nothing has changed with the MPLS label bindings.

```

CSR6#show bgp vpnv4 unicast vrf VPN 192.168.4.0/24
BGP routing table entry for 65000:6:192.168.4.0/24, version 10
Paths: (1 available, best #1, table VPN)
  Not advertised to any peer
  Refresh Epoch 2
  Local, (Received from a RR-client), imported path from
65000:10:192.168.4.0/24 (global)
    10.10.10.10 (metric 10) (via default) (via Tunnel0) from 10.10.10.10
(10.10.10.10)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:65000:10
      Connector Attribute: count=1
        type 1 len 12 value 65000:10:10.10.10.10
        mpls labels in/out nlabel/10006
      rx pathid: 0, tx pathid: 0x0

```

```

CSR6#show derived-config interface tunnel0
Building configuration...

```

```

Derived configuration : 151 bytes

```

```

!
interface Tunnel0
  ip unnumbered Loopback0
  no ip redirects
  ipv6 enable
  tunnel source Loopback0
  tunnel mode gre multipoint
  tunnel key 65000
end

```

The routing table never changed, but now VPN traffic destined for those BGP next-hops is sent inside this tunnel using the L3VPN encapsulation profile. There is a new adjacency in the FIB that details these connections. One is listed as IP and one is listed as MPLS.

```

CSR6#show adjacency 10.10.10.10
Protocol Interface Address
IP Tunnel0 10.10.10.10 (4)
TAG Tunnel0 10.10.10.10 (4)

```

Digging into the encapsulation details, we see the only difference between these two entries is the protocol type (0x0800 for IP versus 0x8847 for MPLS). We expect the MPLS traffic inside this tunnel to have the VPN label directly beneath this layer of encapsulation, before the customer IP header. The tunnel key of 65000 (0xFDE8) marks the end of the GRE encapsulation. Immediately following this tunnel key, assuming the type is 0x8847, will be the customer VPN label.

```

CSR6#show adjacency 10.10.10.10 encapsulation

```

```

Protocol Interface      Address
IP      Tunnel0        10.10.10.10(4)
  Encap length 28
  4500000000000000FF2F9BAF06060606
  0A0A0A0A200008000000FDE8
  Provider: TUNNEL
  Protocol header count in macstring: 2
    HDR 0: ipv4
      dst: static, 10.10.10.10
      src: static, 6.6.6.6
      prot: static, 47
      ttl: static, 255
      df: static, cleared
      per packet fields: tos ident tl chksum
    HDR 1: gre
      prot: static, 0x800
      key: static, 65000
      per packet fields: none
TAG      Tunnel0        10.10.10.10(4)
  Encap length 28
  4500000000000000FF2F9BAF06060606
  0A0A0A0A200088470000FDE8
  Provider: TUNNEL
  Protocol header count in macstring: 2
    HDR 0: ipv4
      dst: static, 10.10.10.10
      src: static, 6.6.6.6
      prot: static, 47
      ToS: static, 0
      ttl: static, 255
      df: static, cleared
      per packet fields: ident tl chksum
    HDR 1: gre
      prot: static, 0x8847
      key: static, 65000
      per packet fields: none

```

We can also check the tunnel endpoints to see what was discovered. Because this tunnel is used for VPNv4 and VPNv6, both IPv4 and IPv6 BGP next-hops are bound to that transport address.

```

CSR6#show tunnel endpoints tunnel 0
Tunnel0 running in multi-GRE/IP mode

Endpoint transport 9.9.9.9 Refcount 4 Base 0x7FEC1811F158 Create Time 01:35:03
  overlay ::FFFF:9.9.9.9 Refcount 2 Parent 0x7FEC1811F158 Create Time 01:35:03
  overlay 9.9.9.9 Refcount 2 Parent 0x7FEC1811F158 Create Time 01:35:03
Endpoint transport 10.10.10.10 Refcount 4 Base 0x7FEC1811F2E8 Create Time 01:34:58
  overlay ::FFFF:10.10.10.10 Refcount 2 Parent 0x7FEC1811F2E8 Create Time 01:34:58
  overlay 10.10.10.10 Refcount 2 Parent 0x7FEC1811F2E8 Create Time 01:34:58

```

As an example, we will send traffic from CSR4 to CSR8 inside the customer network. The IGP cost between CSR7 and CSR5 in the core carrier is high cost, so capturing on CSR2's interface towards CSR7 should yield the most encapsulation. The significant components are highlighted in different colors and are shown below.

```
CSR4#ping 192.168.8.8
```

```
CSR2#show monitor capture CAP7 buffer dump
0000: 000C295C E1E9000C 29664C2C 81000DC7  ..)\....)fL,....
0010: 8847007D 80FD013B 91FD4500 00840010  .G.}...;..E....
0020: 0000FD2F 97150A0A 0A0A0909 09092000  .../.....
0030: 88470000 FDE80232 D1FF4500 00640030  .G.....2..E..d.0
0040: 0000FF01 2E05C0A8 040AC0A8 08090800  .....
0050: 7B3B000A 00010000 00009568 6D9BABCD  {;.....hm...
[snip]
```

The first 12 bytes are the destination and source MAC addresses, respectively. The next 4 bytes specify 802.1Q encapsulation with VLAN ID 0xDC7 (3527). 0x8847 specifies MPLS unicast, and 0x7D8 (2008) is the LDP label from CSR2 to reach CSR5, highlighted in yellow.

```
CSR7#show mpls ldp bindings 5.5.5.5 32
lib entry: 5.5.5.5/32, rev 8
  local binding:  label: 7000
  remote binding: lsr: 5.5.5.5:0, label: imp-null
  remote binding: lsr: 2.2.2.2:0, label: 2008
```

The next label is 0x13B9 (5049) which is the VPNv4 label from CSR5 describing reachability to CSR9. The S-bit is set, as expected. This is the bottom-most label and is also highlighted in yellow.

```
CSR7#show bgp vpnv4 unicast vrf CH labels
Network          Next Hop          In label/Out label
Route Distinguisher: 33:7 (CH)
  6.6.6.6/32      3.3.3.3           nolabel/3001
                  2.2.2.2           nolabel/2003
  9.9.9.9/32      5.5.5.5           nolabel/5049
  10.10.10.10/32  10.255.11.11      7010/nolabel
```

Next is an IP packet with protocol 0x2F (47) for GRE in cyan. Source and destination addresses are highlighted in green (10.10.10.10 -> 9.9.9.9). The payload of this GRE/IP packet is 0x8847 with a tunnel key of 0xFDE8 (65000). The next label is 0x232D (9005) which is the VPN label allocated by CSR9 describing reachability to the connected route with CSR8, highlighted in grey. This proves that the VPN label immediately follows the GRE header.

```
CSR10#show bgp vpnv4 unicast vrf VPN labels
```

Network	Next Hop	In label/Out label
Route Distinguisher: 65000:10 (VPN)		
192.168.1.0	6.6.6.6	nolabel/6002
192.168.4.0	0.0.0.0	10006/nolabel (VPN)
192.168.8.0	9.9.9.9	nolabel/9005

The final significant piece of this encapsulation is the final IP header, which is IP protocol 1 (ICMP), from 192.168.4.10 to 192.168.8.9 (in pink). This is the original VPN traffic. We can summarize the encapsulation for unicast traffic as follows:

1. Layer 2 encapsulation – Ethernet, FR, ATM, PPP, etc
2. MPLS transport core labels – LDP, RSVP-TE, TE-FRR, Segment Routing, etc
3. MPLS VPN core label – BGP VPNv4 always, since only IPv4 is supported for outer encapsulation
4. IP/GRE header (L3VPN encap profile) – 4 or 8 bytes depending on key, type is always 0x8847
5. MPLS VPN customer label – BGP VPNv4 or VPNv6 between customer PEs
6. Customer traffic – IPv4 or IPv6 customer traffic

Next, we will examine multicast transport. Both the customer carrier and final customers are using SSM only to simplify testing, so there are no RPs anywhere. The core carrier is PIM-free using mLDP with CSR3 as the root of the MP2MP tree (MVPN profile 1). There are no P2MP mLDP trees in this example, although there is nothing precluding them from existing. CSR1 and CSR8 are both receivers for group 232.4.4.4 and signal their interest to the PEs using IGMPv3. The joins include the source 192.168.4.4, which is the only sender. The customer PEs will encapsulate this multicast traffic into GRE multicast using the MDT default group 232.0.0.255 (MVPN profile 0). We will check CSR10 as a quickly to verify its MDT status. CSR10 has VRF-aware PIM neighbors within the customer VPN.

```
CSR10#show ip pim mdt
```

```
* implies mdt is the default MDT, # is (*,*) Wildcard,
> is non-(*,*) Wildcard
```

MDT Group/Num	Interface	Source	VRF
* 232.0.0.255	Tunnel1	Loopback0	VPN

```
CSR10#show ip pim vrf VPN neighbor
```

```
PIM Neighbor Table
```

```
Mode: B - Bidir Capable, DR - Designated Router, N - Default DR Priority,
      P - Proxy Capable, S - State Refresh Capable, G - GenID Capable,
      L - DR Load-balancing Capable
```

Neighbor Address	Interface	Uptime/Expires	Ver	DR Prio/Mode
9.9.9.9	Tunnel1	01:51:51/00:01:35	v2	1 / S P G
6.6.6.6	Tunnel1	03:33:13/00:01:35	v2	1 / S P G

XRv11 receives a P(S,G) join from CSR7 for the default MDT. Customer AS 65000 is PIM-enabled for MVPN transport, as expected, so XRv11's only job is to continue building the P(S,G) tree towards the source, CSR10. XRv11's P(S,G) state shown below forwards traffic along the SPT towards the core carrier.

For brevity, we do not trace the entire P(S,G) join process; if XRv11 has the join, it is indicative of a successful configuration. Additional MVPN verifications are detailed in the corresponding chapter.

```
RP/0/0/CPU0:XRv11#show pim topology 232.0.0.255 10.10.10.10 | begin 232
(10.10.10.10,232.0.0.255)SPT SSM Up: 01:19:08
JP: Join(00:00:41) RPF: GigabitEthernet0/0/0/0.510,10.10.11.10 Flags:
GigabitEthernet0/0/0/0.571 01:19:08 fwd Join(00:03:14)
```

A quick check of CSR7's MDT status shows all neighbors are up. MDT 0 within the context of mLDP means the default MDT (MP2MP delivery tree). Even without verifying this, we could assume it was working based on the P(S,G) join above. If the core carrier's MDT is broken, the customer MDTs relying on it are, by extension, also broken.

```
CSR7#show ip pim mdt
* implies mdt is the default MDT, # is (*,*) Wildcard,
> is non-(*,*) Wildcard
MDT Group/Num Interface Source VRF
* 0 Lspvif0 Loopback0 CH
```

```
CSR7#show ip pim vrf CH neighbor
PIM Neighbor Table
Mode: B - Bidir Capable, DR - Designated Router, N - Default DR Priority,
      P - Proxy Capable, S - State Refresh Capable, G - GenID Capable,
      L - DR Load-balancing Capable
Neighbor Interface Uptime/Expires Ver DR
Address Prio/Mode
10.255.11.11 GigabitEthernet2.571 03:41:22/00:01:36 v2 1 / DR P G
5.5.5.5 Lspvif0 03:37:09/00:01:36 v2 1 / S P G
3.3.3.3 Lspvif0 03:37:09/00:01:38 v2 1 / S P G
2.2.2.2 Lspvif0 03:37:09/00:01:31 v2 1 / S P G
```

When CSR4 starts sending traffic, CSR10 encapsulates it inside the default MDT and forwards it to XRv11. XRv11 forwards it to CSR7 so it enters the core carrier. These basic steps are not shown, and we skip to CSR7 for additional verification. Upon receipt, CSR7 encapsulates the traffic inside MPLS (LSM) with a single mLDP label allocated by CSR2. This is upstream towards the root, which is CSR3.

```
CSR7#show mpls mldp database opaque_type mdt 33:65000 0
LSM ID : 1 (RNR LSM ID: 2) Type: MP2MP Uptime : 03:38:27
FEC Root : 3.3.3.3
Opaque decoded : [mdt 33:65000 0]
Opaque length : 11 bytes
Opaque value : 02 000B 000033000650000000000000
RNR active LSP : (this entry)
Upstream client(s) :
2.2.2.2:0 [Active]
Expires : Never Path Set ID : 1
Out Label (U) : 2011 Interface : GigabitEthernet2.527*
```



```

Local Label (D): 7005           Next Hop       : 33.2.7.2
Replication client(s):
MDT (VRF CH)
Uptime           : 03:38:27       Path Set ID  : 2
Interface        : Lspvif0

```

At this time, we will enable the capture inbound on CSR2 and use that to view the encapsulation as we did for unicast. We expect to see a single transport label (mLDP) after the layer 2 encapsulation with no customer carrier MPLS encapsulation, since that carrier is using MVPN profile 0 (no LSM). The output is described in detail next.

```
CSR4#ping 232.4.4.4 repeat 5 timeout 0
```

```

CSR2#show monitor capture CAP7 buffer dump
0000: 000C295C E1E9000C 29664C2C 81000DC7   ..)\....)fL,....
0010: 8847007D B1FD4500 007C00AB 0000FD2F   .G.}...E...|.... /
0020: BF940A0A 0A0AE800 00FF0000 08004500   .....E.
0030: 00641178 0000FE01 FA6BC0A8 0404E804   .d.x.....k.....
0040: 04040800 ACF40027 00010000 00009669   .....'......i
0050: 3AC4ABCD ABCDABCD ABCDABCD ABCDABCD   :.....

```

Jumping straight to the first set of MPLS labels, we see value 0x7DB (2011) used to send traffic towards the mLDP root (CSR3), using an upstream mLDP label allocated by CSR2 (yellow). This is the only label, and the assertion of the S-bit proves this. Immediately following this is a GRE/IP header with source 10.10.10.10 and destination 232.0.0.255, which represents the MDT source and default group, respectively (green). Following that is another IP packet, the customer ICMP ping, from 192.168.4.4 to 232.4.4.4 (cyan). There is significantly less MPLS interaction here since the core carrier uses a single label to transport the multicast between PEs (no PHP) but the customer carrier doesn't use LSM at all. The point of this demonstration was to show that the L3VPN encapsulation profile supports both unicast and multicast transport. Like any good CSC solution, the customer and core carrier forwarding mechanisms can be entirely different as well.

The final set of tests will quickly evaluate IPv6 unicast and multicast across the network. IPv6 SSM is used again in the customer network as transport and is encapsulated inside IPv4 GRE multicast at the customer PEs. Since the P(S,G) signaled is still IPv4-based, we don't need to retrace the signaling path for the IPv6 MVPN flow. We will perform a unicast capture first, and then validate the packet contents against the control-plane show commands. The IPv6 header is much larger to account for larger addresses, but decoding it is easier because the numbers are hexadecimal in the first place.

```
CSR4#ping fd00:192:168:8::8
```

```

CSR2#show monitor capture CAP7 buffer dump
0000: 000C295C E1E9000C 29664C2C 81000DC7   ..)\....)fL,....
0010: 8847007D 80FD013B 91FD4500 008400C1   .G.}....;..E.....
0020: 0000FD2F 96640A0A 0A0A0909 09092000   .../.d.....

```

```

0030:  88470000 FDE80233 113F6000 0000003C   .G.....3.?`....<
0040:  3A3FFD00 01920168 00040000 00000000   :?.....h.....
0050:  0004FD00 01920168 00080000 00000000   .....h.....
0060:  00088000 9CC821D6 00020203 04050607   .....!.....
0070:  08090A0B 0C0D0E0F 10111213 14151617   .....

```

The core carrier label stack is {2008 5049} in yellow, the IP/GRE header is 10.10.10.10 -> 9.9.9.9 in cyan, the customer VPN label is 9009 in green, and the IPv6 header is FD00:192:168:4::4 -> FD00:192:168:8::8 in pink. We won't verify the ICMP ping because we initiated it and it's easily understood. We will verify it as quickly as possible using the most succinct show commands, for variation. The core carrier imposes two labels for traffic heading towards the remote PE, and this traffic was generated by the source PE sending traffic to the dynamic endpoints and imposing the VPN label for the customer route allocated by the remote PE (VPNv6). The colors in the EPC capture match the highlights below.

```

CSR7#show ip cef vrf CH 9.9.9.9
9.9.9.9/32
  nexthop 33.2.7.2 GigabitEthernet2.527 label 2008 5049

CSR10#show ipv6 cef vrf VPN fd00:192:168:8::8
FD00:192:168:8::/64
  nexthop ::FFFF:9.9.9.9 Tunnel0 label 9009

CSR10#show tunnel endpoints tunnel 0 | section 9\.9
Endpoint transport 9.9.9.9 Refcount 4 Base 0x7F57F6166E98 Create Time
01:50:44
  overlay ::FFFF:9.9.9.9 Refcount 2 Parent 0x7F57F6166E98 Create Time
01:50:44
    overlay 9.9.9.9 Refcount 2 Parent 0x7F57F6166E98 Create Time 01:50:44

```

Unfortunately, IPv6 multicast does not appear to work. There is an RPF logic problem. When the last-hop router (egress customer PE) receives the MLDv2 SSM join from the customer, it thinks the RPF interface is the L3VPN encapsulation tunnel and not the MDT tunnel. The remote end (CSR10) never receives the PIMv6 (S,G) join since the egress PE isn't able to send it properly. Below, all RPF/C-MFIB related commands indicate that the L3VPNv6 tunnel is the RPF interface (Tunnel0) while the PIM neighbors form over the default MDT (Tunnel1).

```

CSR6#show ipv6 rpf vrf VPN FD00:192:168:4::4
RPF information for FD00:192:168:4::4
  RPF interface: Tunnel0
  RPF neighbor: ::FFFF:10.10.10.10
  RPF route/mask: FD00:192:168:4::/64
  RPF type: Unicast
  RPF recursion count: 0
  Metric preference: 200
  Metric: 0
  Using Extranet RPF Rule: BGP Imported Route, RPF VRF:

```

```

CSR6#show ipv6 mroute vrf VPN FF33::4 FD00:192:168:4::4
[snip]
(FD00:192:168:4::4, FF33::4), 00:20:40/never, flags: sTI
  Incoming interface: Tunnel0
  RPF nbr: ::FFFF:10.10.10.10
  Immediate Outgoing interface list:
    GigabitEthernet2.516, Forward, 00:20:40/never

```

```

CSR6#show ipv6 pim vrf VPN neighbor
PIM Neighbor Table
Mode: B - Bidir Capable, G - GenID Capable
Neighbor Address      Interface      Uptime      Expires      Mode DR pri
::FFFF:9.9.9.9        Tunnel1       00:10:00    00:01:38    B G        1
::FFFF:10.10.10.10    Tunnel1       00:13:02    00:01:38    B G    DR 1

```

```

CSR10#show ipv6 mroute vrf VPN
No mroute entries found.

```

Quickly checking IPv4, we see this is not a problem since the router knows that the RPF interface for multicast traffic within the VPN should be the MDT "emulated LAN". I believe the IPv6 support isn't fully implemented yet, because we clearly saw IPv4 multicast working earlier. Since there isn't an actual Tunnel0 or Tunnel1 interface, correcting this problem appears impossible at this time (no ability to enable PIM, adjust mroutes, etc).

```

CSR6#show ip rpf vrf VPN 192.168.4.4
RPF information for ? (192.168.4.4)
  RPF interface: Tunnel1
  RPF neighbor: ? (10.10.10.10)
  RPF route/mask: 192.168.4.0/24
  RPF type: unicast (bgp 65000)
  Doing distance-preferred lookups across tables
  BGP originator: 10.10.10.10
  RPF topology: ipv4 multicast base, originated from ipv4 unicast base

```

```

CSR6#show ip mroute vrf VPN 232.4.4.4 192.168.4.4
[snip]
(192.168.4.4, 232.4.4.4), 04:32:28/00:02:05, flags: sTI
  Incoming interface: Tunnel1, RPF nbr 10.10.10.10
  Outgoing interface list:
    GigabitEthernet2.516, Forward/Sparse, 04:32:28/00:02:05

```

```

CSR6#show ip pim vrf VPN neighbor
PIM Neighbor Table
Mode: B - Bidir Capable, DR - Designated Router, N - Default DR Priority,
      P - Proxy Capable, S - State Refresh Capable, G - GenID Capable,
      L - DR Load-balancing Capable

```

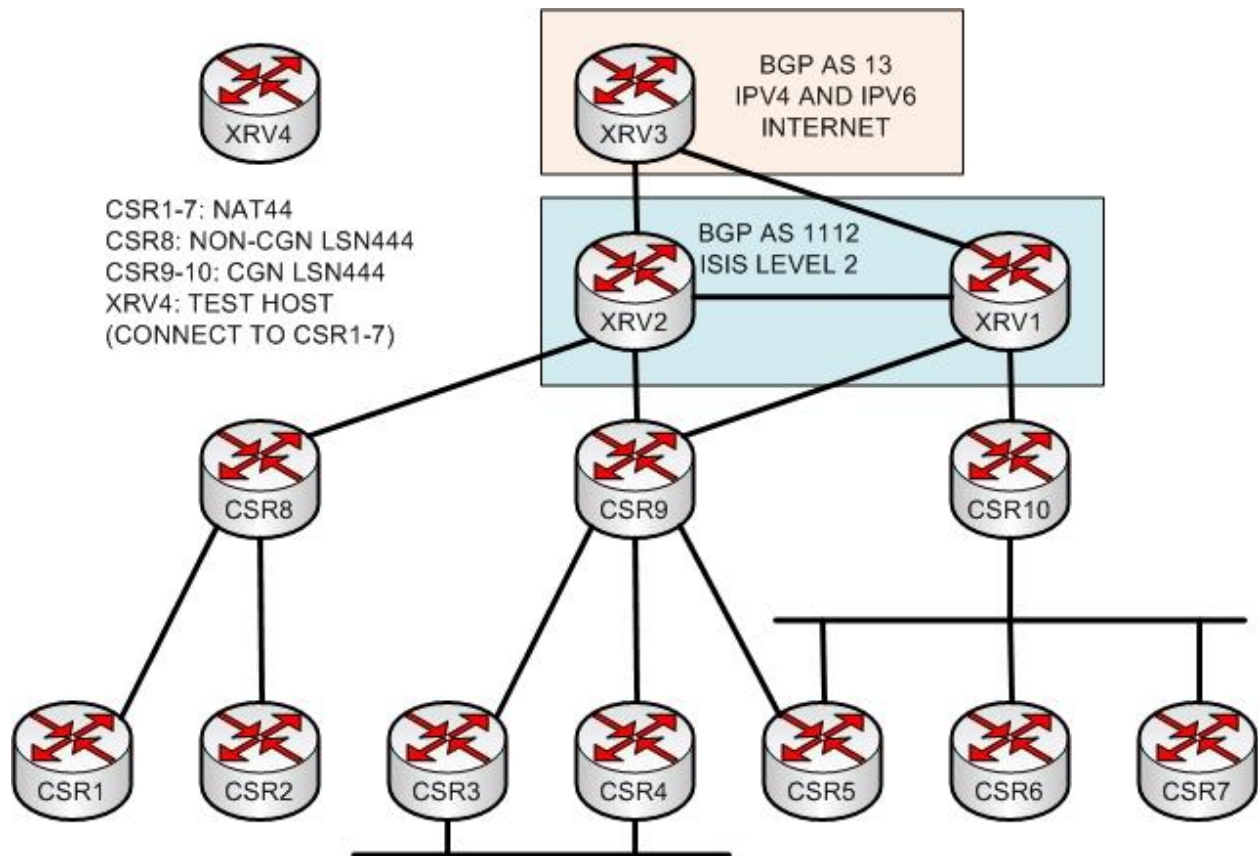
Neighbor Address	Interface	Uptime/Expires	Ver	DR Prio/Mode
9.9.9.9	Tunnel1	03:01:36/00:01:43	v2	1 / S P G
10.10.10.10	Tunnel1	04:42:57/00:01:39	v2	1 / DR SPG

Additional Reading – Reference configurations "l3vpn-mgre"

36. Describe, implement, and troubleshoot IPv6 transition mechanisms

36.1 NAT44 and NAT444

NAT44 is synonymous with ordinary IPv4 NAT which has been around for many years. It is more accurately called NAT44 nowadays, implying an IPv4-to-IPv4 translation. With other mechanisms in existence, such as NAT64 (formerly NAT-PT) and NAT66 (also called Network Prefix Translation v6 or NPTv6), there needed to be a way to distinguish between them, preferably without inventing different names for each one. NAT444 just means doing NAT44 twice; this is common in carrier-grade NAT (CGN) or large scale NAT (LSN) architectures. LSN is the newer term which is more accurate, since CGN implies that NAT444 is only relevant in SP networks, which is false. This document uses both the CGN and LSN terms interchangeably. The lab is generally focused on NAT444 since NAT44 is a subcomponent of it, and having a separate NAT44 lab adds no value. In this setup, CSR1 through CSR7 serve as CPE routers for small businesses or homes. In common LSN architectures, these nodes will perform NAT44 from a private IP range on the inside (locally administered) to another private IP range on the outside (SP administered). The reason is that NAT overload gives us some ~65,000 connections per IP address using TCP/UDP port numbers as multiplexers, and assigning a real global address to a CPE router is viewed as wasteful. If each CPE received a public IPv4 address, their overload utilization would be much lower, which is highly inefficient compared to LSN. The LSN nodes, CSR8 through CSR10, each have real public addresses. They perform NAT44 again from the SP administered private addressing on the inside to public addressing on the outside. XRv1 and XRv2 are BGP routers that interact with the IPv4/v6 Internet simulator, XRv3. XRv4 has several VRFs to represent customer hosts, like residential laptops, and is used for testing only.



Several different NAT44 variations are tested here since we are performing NAT44 on ten different nodes. A summary of the variations is below. CSR8 provides a direct Ethernet service to its CPEs, while CSR9 and CSR10 provide PPPoE variations for broadband aggregation. The CPE inside addressing is in the 192.168.0.0/16 range, the SP-administered private addressing is in the 10.0.0.0/8 range, and the public addresses are in the 209.19.85.8/30 range.

1. CSR1: ACL with NAT pool, no overload
2. CSR2: Route-map with NAT pool, reversible
3. CSR3: HSRP (active) and NAT
4. CSR4: HSRP (standby) and NAT
5. CSR5: Route-map per interface with PBR input, overload
6. CSR6: Static NAT
7. CSR7: ACL with NAT overload, VRF-aware
8. CSR8: Route-map with NAT overload, CGN disabled
9. CSR9: Route-map per interface, overload, CGN enabled
10. CSR10: Route-map with NAT overload, CGN enabled, VRF-aware

The IS-IS and BGP configurations are not shown in this lab as they are very basic to enable reachability from the Internet to the public post-NAT444 IP addresses on the LSN nodes. IS-IS is used between the

LSN and Internet ASBRs to exchange LSN and Internet routes back and forth. CSR1's NAT configuration is very basic. NAT inside is enabled on the CPE LAN with NAT outside facing the SP. The NAT pool contains the outside addresses assigned by the SP. Assigning 100 addresses to a single CPE may seem wasteful, but since they are private addresses, it matters less.

```
! CSR1
interface GigabitEthernet2.514
 ip nat inside

interface GigabitEthernet2.518
 ip nat outside

ip nat pool NAT_POOL 10.1.8.100 10.1.8.199 prefix-length 24
ip nat inside source list ACL_NAT pool NAT_POOL
ip access-list standard ACL_NAT
 permit 192.168.1.0 0.0.0.255
```

We can verify the proper interface and pool configuration using the show commands below. We don't care about the statistical values just yet, but only that the names are correct.

```
R1#show ip nat pool name NAT_POOL
NAT Pool Statistics
Pool name NAT_POOL, id 1
```

	Assigned	Available
Addresses	1	99
UDP Low Ports	0	51200
TCP Low Ports	0	51200
UDP High Ports	0	6451100
TCP High Ports	0	6451100

(Low ports are less than 1024. High ports are greater than or equal to 1024.)

```
R1#show ip nat statistics | section interface
Outside interfaces:
 GigabitEthernet2.518
Inside interfaces:
 GigabitEthernet2.514
```

When using ACL-based NAT without overloading, non-extended NAT entries are automatically created. That is, when a host initiates a protocol-specific session to a remote destination (ping, telnet, etc), that entry is accounted for with a specific port/protocol mapping. However, the inside local to inside global mapping is also stored in the NAT table, which allows unsolicited inbound traffic from the outside. XRv4 sends some traffic to CSR8, the LSN node. CSR1 performs NAT44 to change XRv4's address to an available 10.1.8.0/24 address from the specified pool. Two entries are added to the table; one is fully extended (yellow), which contains the port/protocol information. The other is non-extended (green) which associates 10.1.8.100 with 192.168.1.14.

```
R1#show ip nat translations
```

Pro	Inside global	Inside local	Outside local	Outside global
---	10.1.8.100	192.168.1.14	---	---
icmp	10.1.8.100:37041	192.168.1.14:37041	10.1.8.8:37041	10.1.8.8:37041

Because of this non-extended (green) entry, CSR8 can reach into the CPE network and ping XRv4 using 10.1.8.100. The NAT order of operations states that packets arriving on an inside interface are subject to routing first, then NAT. Packets arriving on an outside interface are subject to NAT first, then routing. Therefore, packets sent to 10.1.8.100 are translated to destination 192.168.1.14 first, then routed, which means CSR1 is not replying to the pings (XRv4 is). We can prove this by performing an EPC capture on CSR1's inside LAN. We see five ICMP packets sourced from 10.1.8.8 destined for 192.168.1.14, and the five replies coming back. It is clear that CSR8 initiated this ping flow and that XRv4 responded, which shows the outside-to-inside reachability. This is often undesirable but is the default behavior of using a NAT ACL with a NAT pool without overloading enabled.

```
R8#ping 10.1.8.100
```

```
Type escape sequence to abort.
```

```
Sending 5, 100-byte ICMP Echos to 10.1.8.100, timeout is 2 seconds:
```

```
!!!!
```

```
Success rate is 100 percent (5/5), round-trip min/avg/max = 3/5/16 ms
```

```
R1#show monitor capture CAP buffer brief
```

#	size	timestamp	source	destination	protocol
0	118	0.000000	10.1.8.8	-> 192.168.1.14	ICMP
1	118	0.001007	192.168.1.14	-> 10.1.8.8	ICMP
2	118	0.001999	10.1.8.8	-> 192.168.1.14	ICMP
3	118	0.003006	192.168.1.14	-> 10.1.8.8	ICMP
4	118	0.003997	10.1.8.8	-> 192.168.1.14	ICMP
5	118	0.005004	192.168.1.14	-> 10.1.8.8	ICMP
6	118	0.005996	10.1.8.8	-> 192.168.1.14	ICMP
7	118	0.005996	192.168.1.14	-> 10.1.8.8	ICMP
8	118	0.007995	10.1.8.8	-> 192.168.1.14	ICMP
9	118	0.016997	192.168.1.14	-> 10.1.8.8	ICMP

When using a route-map with a NAT pool, this outside-to-inside reachability is disabled by default. We will demonstrate this on CSR2. The configuration is very similar, using a slightly different NAT pool to account for a separate Ethernet link from CSR8. The NAT ACL is wrapped in a route-map, which is normally used for multi-homing, but can also be used to hide this "outside-in" functionality.

```
! CSR2
```

```
interface GigabitEthernet2.524
 ip nat inside
```

```
interface GigabitEthernet2.528
```

```

ip nat outside

ip access-list standard ACL_NAT
 permit 192.168.2.0 0.0.0.255

route-map RM_NAT permit 10
 match ip address ACL_NAT

ip nat pool NAT_POOL 10.2.8.100 10.2.8.199 prefix-length 24
ip nat inside source route-map RM_NAT pool NAT_POOL

```

Sending traffic from XRv4 only generates a single fully-extended NAT translation entry, which prevents inside access from CSR8. However, CSR8 still gets ICMP echo-replies, but since outside-to-inside NAT did not occur (no matching entry), CSR2 locally responded to those pings.

```

R2#show ip nat translations
Pro  Inside global      Inside local      Outside local      Outside global
icmp 10.2.8.100:45233   192.168.2.14:45233 10.2.8.8:45233    10.2.8.8:45233
Total number of translations: 1

```

```

R8#ping 10.2.8.100
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.2.8.100, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/6 ms

```

This ping response on CSR2 is the result of the dynamic IP alias created for the inside global address. On static NAT mappings, we can disable it, but not on dynamic ones. Packets arrived on CSR2's outside interface and were subject to NAT before routing. There was no matching NAT entry for the unsolicited ICMP flow from CSR8, so CSR2 performs a routing lookup and sees the dynamic alias effectively as a connected route.

```

R2#show ip alias
Address Type      IP Address      Port
Interface         10.2.8.2
Dynamic           10.2.8.100
Interface         192.168.1.2
Interface         192.168.2.2

```

Using EPC, we can prove these pings are going to XRv4, and that CSR2 is responding. We see no packets sent to the inside LAN when CSR8 initiates the pings, which validates the statements above.

```

R2#show monitor capture CAP buffer brief
-----
#   size  timestamp      source      destination  protocol
-----
[no output]

```


For consistency with CSR1, we can explicitly enable outside-in connectivity when using route-maps using the “reversible” keyword. We enable this on CSR2 and retest. Now, XRv4’s flow has a second entry to account for the outside-originated traffic (the non extended one).

```
! CSR2
ip nat inside source route-map RM_NAT pool NAT_POOL reversible

R2#show ip nat translations
Pro  Inside global      Inside local      Outside local      Outside global
---  10.2.8.100          192.168.2.14      ---              ---
icmp 10.2.8.100:41137   192.168.2.14:41137 10.2.8.8:41137    10.2.8.8:41137
```

Pinging from CSR8 now shows up in CSR2’s capture after modifying the NAT configuration. Since there is a non-extended NAT entry, NAT sees a match, performs a translation, then routing occurs. This behavior is operationally identical to CSR1.

```
R8#ping 10.2.8.100
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.2.8.100, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/140/671 ms
```

```
R2#show monitor capture CAP buffer brief
-----
#    size    timestamp      source          destination     protocol
-----
0    118      0.000000      10.2.8.8        -> 192.168.2.14    ICMP
1    118      0.001999      192.168.2.14    -> 10.2.8.8        ICMP
2    118      0.840005      10.2.8.8        -> 192.168.2.14    ICMP
3    118      0.843011      192.168.2.14    -> 10.2.8.8        ICMP
4    118      0.850014      10.2.8.8        -> 192.168.2.14    ICMP
5    118      0.852013      192.168.2.14    -> 10.2.8.8        ICMP
6    118      0.854012      10.2.8.8        -> 192.168.2.14    ICMP
7    118      0.856010      192.168.2.14    -> 10.2.8.8        ICMP
8    118      0.858009      10.2.8.8        -> 192.168.2.14    ICMP
9    118      0.860008      192.168.2.14    -> 10.2.8.8        ICMP
```

Next, we will configure CGN on CSR8. There is a specific “CGN” mode of NAT which is discussed later, but the term is misleading because there is nothing “carrier-grade” about it. It just modifies the NAT process slightly; we will evaluate this on CSR9 and CSR10 later. CSR8, however, will just do NAT44 again with CSR1 and CSR2 10.1.8.0/24 and 10.2.8.0/24 subnets as inside addresses. These subnets were the outside NAT pools on the CPEs, but are viewed as inside addresses from the perspective of CSR8. The public address 209.19.85.8/32 is configured on a loopback so as not to waste public addressing on transit links with XRv2. CSR8 uses a route-map with overload, which means reversible NAT (outside-in) is not supported at all.

```

! CSR8
interface GigabitEthernet2.518
 ip nat inside

interface GigabitEthernet2.528
 ip nat inside

interface GigabitEthernet2.582
 ip nat outside

ip nat inside source route-map RM_NAT interface Loopback209 overload

ip access-list standard ACL_NAT
 permit 10.1.8.0 0.0.0.255
 permit 10.2.8.0 0.0.0.255

route-map RM_NAT permit 10
 match ip address ACL_NAT

```

We will perform a basic routing check on CSR8 to make sure it has Internet connectivity. The Internet consists of routes from AS 13 (13.0.0.0/8) while XRv1 and XRv2 summarize this into IS-IS from BGP. We confirm that CSR8 has the routing information and reachability via ping.

```

R8#show ip route 13.0.0.0 255.0.0.0
Routing entry for 13.0.0.0/8
  Known via "isis", distance 115, metric 30, type level-2
  Redistributing via isis 1112
  Last update from 10.8.12.12 on GigabitEthernet2.582, 00:27:51 ago
  Routing Descriptor Blocks:
  * 10.8.12.12, from 0.0.0.0, 00:27:51 ago, via GigabitEthernet2.582
    Route metric is 30, traffic share count is 1

R8#ping 13.144.2.9 source loopback209
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 13.144.2.9, timeout is 2 seconds:
Packet sent with a source address of 209.19.85.8
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/147/724 ms

```

At this point, hosts behind CSR1 and CSR2 should have Internet reachability. Sending traffic from both LANs to the Internet, we check the NAT tables on CSR1, CSR2, and CSR8. Since CSR1 is using ACL-based NAT with pools, we see non-extended entries for outside-in traffic. Likewise on CSR2, using the route-map-based NAT pool technique with the “reversible” option, we see non-extended entries as well. CSR8 has two fully-extended entries that map the inside local addresses (inside global addresses from CSR1 and CSR2 perspectives) to the single public IP addressing using port overloading. Addresses are highlighted to show their position in the NAT tables. NAT444 derives its names from the original host being represented by three addresses, which implies NAT44 performed twice. In this case, 192.168.1.14

is also seen as 10.1.8.100 and 209.19.85.8. 192.168.2.14 is also seen as 10.2.8.100 and 209.19.85.8; the hierarchy of LSN allows better IPv4 address conservation.

```
R1#show ip nat translations
Pro  Inside global      Inside local      Outside local     Outside global
---  10.1.8.100           192.168.1.14     ---              ---
icmp 10.1.8.100:12465   192.168.1.14:12465 13.144.2.1:12465 13.144.2.1:12465
```

```
R2#show ip nat translations
Pro  Inside global      Inside local      Outside local     Outside global
---  10.2.8.100           192.168.2.14     ---              ---
icmp 10.2.8.100:16561  192.168.2.14:16561 13.144.2.1:16561 13.144.2.1:16561
Total number of translations: 2
```

```
R8#show ip nat translations
Pro  Inside global      Inside local      Outside local     Outside global
icmp 209.19.85.8:2     10.2.8.100:16561 13.144.2.1:16561 13.144.2.1:2
icmp 209.19.85.8:1     10.1.8.100:12465 13.144.2.1:12465 13.144.2.1:1
```

Next, we will configure CSR3 and CSR4 concurrently. Stateful NAT (SNAT) does not appear to be supported on the CSR1000v, which was the intent of this setup. We can configure a SNAT-like setup anyway. The XRv4 VRF host is on a shared LAN with both routers; CSR3 and CSR4 run HSRPv2 to determine default gateway status for XRv4. Each one tracks the presence of the default route, which is negotiated with IPCP (via PPPoE). CSR3 is active while CSR4 is standby based on priorities. The configuration is identical on CSR3 and CSR4 with the exception of the HSRP priority and interface enumerations, so only CSR3 is shown. Only pertinent configurations are shown; both routers use basic ACL NAT with overload, which means outside-in traffic is not permitted. For PPPoE technical details, review the specific chapter on this topic.

```
! CSR3
interface GigabitEthernet2.534
 ip nat inside
 standby version 2
 standby 344 ip 192.168.34.254
 standby 344 priority 105
 standby 344 preempt
 standby 344 name SITE34
 standby 344 track 9 decrement 10

interface Dialer3
 ip address negotiated
 ip nat outside
 ppp ipcp route default

interface GigabitEthernet2.539
 pppoe-client dial-pool-number 3

track 9 ip route 0.0.0.0 0.0.0.0 reachability
```

```
ip nat inside source list ACL_NAT interface Dialer3 overload
ip access-list standard ACL_NAT
 permit 192.168.34.0 0.0.0.255
```

In order for basic reachability to exist with the LSN node, we have to configure CSR9 also. CSR9 hands out IP addresses via IPCP using a local address pool. This keeps all of the PPPoE clients in the same “subnet”, but they cannot reach each other since PPP only installs host routes between connected endpoints. All of CSR9’s PPPoE clients are in different VLANs, so we can set very strict session limits for broadband aggregation. For now, we will not configure any NAT on CSR9 so that we can verify CSR3 and CSR4 in isolation.

```
! CSR9
bba-group pppoe PPPOE_LSN
 virtual-template 345
 sessions per-mac limit 1
 sessions per-vlan limit 1

interface Virtual-Template345
 peer default ip address pool PPPOE_POOL

interface GigabitEthernet2.539
 pppoe enable group PPPOE_LSN

interface GigabitEthernet2.549
 pppoe enable group PPPOE_LSN
```

After configuring this, CSR3 and CSR4 should establish PPPoE sessions with CSR9. They both install static default routes as supplied by IPCP, using the PPPoE server as a next-hop. For brevity, we will verify the client details on CSR3 only. Recursive routing lookups show that the PPPoE server IP address is known via a host route, also installed by PPP.

```
R3#show pppoe session
 1 client session
```

Uniq ID	PPPoE SID	RemMAC LocMAC	Port	VT	VA VA-st	State Type
N/A	26	0050.56a9.d672 0050.56a9.8ccf	Gi2.539	Di3	Vi1	UP

```
R3#show ip route 0.0.0.0
Routing entry for 0.0.0.0/0, supernet
  Known via "static", distance 1, metric 0, candidate default path
  Routing Descriptor Blocks:
    * 10.34.59.9
      Route metric is 0, traffic share count is 1
```

```
R3#show ip route 10.34.59.9
Routing entry for 10.34.59.9/32
```

Known via "connected", distance 0, metric 0 (connected, via interface)

Routing Descriptor Blocks:

* directly connected, via Dialer3

Route metric is 0, traffic share count is 1

CSR9 shows two PPPoE sessions, one to CSR3 and one to CSR4. Notice that the VLAN numbers help differentiate the connections. We can see that CSR9 has host routes to two local pool addresses, which are allocated to CSR3 and CSR4.

```
R9#show pppoe session
```

```
3 sessions in LOCALLY_TERMINATED (PTA) State
```

```
3 sessions total
```

Uniq ID	PPPoE SID	RemMAC LocMAC	Port	VT	VA VA-st	State Type
26	26	0050.56a9.8ccf	Gi2.539	345	Vi1.1	PTA
		0050.56a9.d672	VLAN:3539		UP	
35	35	0050.56a9.2c57	Gi2.549	345	Vi1.2	PTA
		0050.56a9.d672	VLAN:3549		UP	

```
R9#show ip route connected | include 59\.1[0-9]+
```

```
C 10.34.59.101/32 is directly connected, Virtual-Access1.1
```

```
C 10.34.59.110/32 is directly connected, Virtual-Access1.2
```

For clarity, we confirm that CSR3 is .101 and CSR4 is .110. We can also draw this conclusion by looking at the in-use pool addresses and mapping those to the virtual-access IDs shown in the PPPoE session table. Vi1.1 maps to CSR3 and Vi1.2 maps to CSR4.

```
R3#show ip interface brief dialer 3
```

Interface	IP-Address	OK?	Method	Status
Dialer3	10.34.59.101	YES	IPCP	up

```
R4#show ip interface brief dialer 4
```

Interface	IP-Address	OK?	Method	Status
Dialer4	10.34.59.110	YES	IPCP	up

```
R9#show ip local pool PPPoE_POOL | begin Inuse
```

```
Inuse addresses:
```

```
10.34.59.101 Vi1.1
```

```
10.34.59.110 Vi1.2
```

At this point, we will verify that HSRP has converged by checking both CSR3 and CSR4. CSR3 should be active while CSR4 is standby. Last, we double check our NAT configuration to ensure the interfaces are set up properly before testing.

```
R3#show standby brief
                P indicates configured to preempt.
                |
Interface   Grp  Pri P State   Active           Standby           Virtual IP
Gi2.534     344  105 P Active   local            192.168.34.4      192.168.34.254
```

```
R3#show ip nat statistics | section interface
```

```
Outside interfaces:
```

```
Dialer3, Virtual-Access1
```

```
Inside interfaces:
```

```
GigabitEthernet2.534
```

```
R4#show standby brief
```

```
                P indicates configured to preempt.
                |
Interface   Grp  Pri P State   Active           Standby           Virtual IP
Gi2.534     344  100 P Standby  192.168.34.3     local            192.168.34.254
```

```
R4#show ip nat statistics | section interface
```

```
Outside interfaces:
```

```
Dialer4, Virtual-Access2
```

```
Inside interfaces:
```

```
GigabitEthernet2.534
```

Now, we will send traffic from XRv4 to CSR9's PPPoE server address. We expect to see NAT entries on CSR3 only (no SNAT supported) since it is the active HSRP gateway. Notice that there is only one fully-extended entry; whether using an ACL or route-map, outside-in traffic is non-reversible when overloading is enabled. Non-extended entries don't exist since the NAT mappings happen at layer 4, not layer 3.

```
R3#show ip nat translations
Pro  Inside global      Inside local          Outside local          Outside global
icmp 10.34.59.101:1       192.168.34.14:24753  10.34.59.9:24753      10.34.59.9:1
Total number of translations: 1
```

```
R4#show ip nat translations
```

```
Total number of translations: 0
```

As a quick test of HSRP, we will break CSR3's PPPoE connection by shutting down its dialer interface. CSR4 becomes the active HSRP gateway since CSR3 loses its default route and decrements its HSRP priority to less than CSR4's value. We confirm this on CSR3 and CSR4. This proves that the default-route tracker works properly.

```
R3#show standby brief
```

```
                P indicates configured to preempt.
                |
Interface   Grp  Pri P State   Active           Standby           Virtual IP
Gi2.534     344  95  P Standby  192.168.34.4     local            192.168.34.254
```

```
R4#show standby brief
```

```
          P indicates configured to preempt.
```

Interface	Grp	Pri	P	State	Active	Standby	Virtual IP
Gi2.534	344	100	P	Active	local	192.168.34.3	192.168.34.254

When sending traffic to CSR9 from XRv4 again, now CSR4 handles the NAT. The NAT behavior is identical to CSR3 since both routers shared common NAT strategies.

```
R3#show ip nat translations
```

```
Total number of translations: 0
```

```
R4#show ip nat translations
```

Pro	Inside global	Inside local	Outside local	Outside global
icmp	10.34.59.110:1	192.168.34.14:28849	10.34.59.9:28849	10.34.59.9:1

```
Total number of translations: 1
```

Before continuing, we bring CSR3 back into the PPPoE network. CSR9 now allocates a new address of 10.34.59.103, which is relevant for NAT44 discussed next.

```
R3#show ip interface brief dialer 3
```

Interface	IP-Address	OK?	Method	Status
Protocol				
Dialer3	10.34.59.103	YES	IPCP	up

Next, we can configure NAT on CSR9. We will introduce a new feature; route-map multi-homing. Since CSR9 is connected to multiple BGP gateway routers, CEF will split traffic between the two assuming ECMP is in play. Using route-maps, we can reference the same inner ACL to match the inside local addresses (from the PPPoE local pool). CSR9 has two public IP addresses available: 209.19.85.9 and 209.19.85.11. Traffic routed to XRv1 (east) will use .9, while traffic routed to XRv2 (west) will use .11. CSR9 is not using policy-routing to force traffic in a given direction; this just specifies the post-NAT address, or packet source, based on the CEF load-sharing decision process. Since inside NAT happens before routing, the outgoing interface has already been identified, implying that the NAT process can use this as match criteria in the route-map.

```
! CSR9
```

```
interface Virtual-Template345
 ip nat inside
```

```
interface GigabitEthernet2.591
 ip nat outside
```

```
interface GigabitEthernet2.592
 ip nat outside
```

```
ip access-list standard ACL_NAT
 permit 10.34.59.0 0.0.0.255
```

```

route-map RM_NAT_EAST permit 10
  match ip address ACL_NAT
  match interface GigabitEthernet2.591

route-map RM_NAT_WEST permit 10
  match ip address ACL_NAT
  match interface GigabitEthernet2.592

ip nat inside source route-map RM_NAT_EAST interface Loopback209 overload
ip nat inside source route-map RM_NAT_WEST interface Loopback211 overload

```

We will also introduce the concept of CGN. There are two modes of NAT supported on the CSR1000v: traditional and CGN. Traditional NAT stores the outside local/global address per flow which determine the final destination an inside host is trying to reach. Conversely, it shows the source address of returning packets. CGN does not track this at all; this conserves memory, which allows NAT to scale better. The outside local/global address are irrelevant since a CGN device can look at the destination address/port of a returning packet and map it to an existing NAT entry. Enabling CGN automatically removes support for NAT outside mappings, so only the first command is required. The biggest downside of CGN in an enterprise environment is that the entire “ip nat outside” syntax tree disappears. Because there are no outside mappings, this feature is unsupported. Outside mappings are not commonly used anyway, and certainly not needed for LSN.

```

! CSR9
ip nat settings mode cgn
no ip nat settings support mapping outside

```

We expect traffic to be translated from 10.34.59.103 (CSR3’s PPPoE client address) to either 205.19.85.9 or .11 depending on the CEF decision. To demonstrate both NAT policies, we will destine traffic to two different Internet destinations from the same source. CSR9’s FIB says that traffic to 13.144.2.1 will be routed east to XRv1, while .9 will be routed west to XRv2. Yellow (east) and green (west) are used throughout this test to differentiate flows.

```

R9#show ip cef exact-route 10.34.59.103 13.144.2.1
10.34.59.103 -> 13.144.2.1 =>IP adj out of GigabitEthernet2.591, addr
10.9.11.11

```

```

R9#show ip cef exact-route 10.34.59.103 13.144.2.9
10.34.59.103 -> 13.144.2.9 =>IP adj out of GigabitEthernet2.592, addr
10.9.12.12

```

XRv4 will send pings to both destinations at the same time. We already know that CSR3 is going to translate both flows to an outside address of 10.34.59.103 based on its IPCP address; we do not verify this again. On CSR9, we see two separate NAT entries with those final Internet destinations (outside global) not tracked at all. CGN simply does not care about those Internet destinations. The traffic to 13.144.2.1 is source-translated to 209.19.85.9, while traffic to 13.144.2.9 is source-translated to

209.19.85.11. This is expected per the NAT policies; since the route-maps used interfaces as match criteria, different post-NAT addresses were used based on the routing decisions. The NAT statistical refcounts show that there is one translation entry per rule. This is a quick way to verify there is some kind of load sharing happening.

```
R9#show ip nat translations
Pro  Inside global      Inside local          Outside local          Outside global
icmp 209.19.85.11:1    10.34.59.103:4       ---                    ---
icmp 209.19.85.9:1    10.34.59.103:3       ---                    ---
Total number of translations: 2
```

```
R9#show ip nat statistics | include ref
[Id: 3] route-map RM_NAT_EAST interface Loopback209 refcount 1
[Id: 4] route-map RM_NAT_WEST interface Loopback211 refcount 1
```

We will use EPC to verify this by capturing outbound on CSR9 using both upstream interfaces. We look at the destination MAC address to verify the directionality from CSR9's perspective. The buffer brief shows the same information as the NAT table, where packets from different post-NAT addresses are used when traffic is destined differently. EPC happens after routing and NAT, which is why we see post-NAT sources.

```
R9#show monitor capture CAP buffer brief
```

#	size	timestamp	source	destination	protocol
2	118	2.249987	209.19.85.9	-> 13.144.2.1	ICMP
3	118	2.252977	209.19.85.9	-> 13.144.2.1	ICMP
4	118	2.254976	209.19.85.9	-> 13.144.2.1	ICMP
5	118	2.257982	209.19.85.9	-> 13.144.2.1	ICMP
6	118	2.267976	209.19.85.9	-> 13.144.2.1	ICMP
7	118	5.979996	209.19.85.11	-> 13.144.2.9	ICMP
8	118	5.982987	209.19.85.11	-> 13.144.2.9	ICMP
9	118	5.985993	209.19.85.11	-> 13.144.2.9	ICMP
10	118	5.988999	209.19.85.11	-> 13.144.2.9	ICMP
11	118	5.998001	209.19.85.11	-> 13.144.2.9	ICMP

The ARP entries are shown below for verifying the destination MAC addresses of XRv1 and XRv2 in the packet captures. We can confirm the statements above by verifying the MAC addresses in addition to the IP addresses. This is just to confirm that the packets are being sent out of the proper interfaces; only one of each packet is shown. The same two packets (#6 and #7) are highlighted again below.

```
R9#show ip arp 10.9.11.11
Protocol Address      Age (min)  Hardware Addr  Type   Interface
Internet 10.9.11.11      3         0050.56a9.9c60 ARPA   GigabitEth2.591
```

```
R9#show ip arp 10.9.12.12
Protocol Address      Age (min)  Hardware Addr  Type   Interface
```

```
Internet 10.9.12.12          190  0050.56a9.0e6f  ARPA  GigabitEth2.592
```

```
R9#show monitor capture CAP buffer detail
```

```
-----
#      size      timestamp      source      destination      protocol
-----
  6    118        2.267976      209.19.85.9      -> 13.144.2.1      ICMP
0000:  005056A9 9C600050 56A9D672 81000E07  .PV..`.PV..r....
0010:  08004500 00640004 0000FD01 87E7D113  ..E..d.....
0020:  55090D90 02010800 CF0E0001 0004ABCD  U.....
0030:  ABCDABCD ABCDABCD ABCDABCD ABCDABCD  .....

  7    118        5.979996      209.19.85.11     -> 13.144.2.9      ICMP
0000:  005056A9 0E6F0050 56A9D672 81000E08  .PV..o.PV..r....
0010:  08004500 00640000 0000FD01 87E1D113  ..E..d.....
0020:  550B0D90 02090800 CF120001 0000ABCD  U.....
0030:  ABCDABCD ABCDABCD ABCDABCD ABCDABCD  .....
```

Next, we will configure CSR6 and CSR7 for NAT44 operations. They will be connecting to CSR10 via PPPoE as well, but will use static addressing along with default static routes in lieu of IPCP. All of the PPPoE clients will share a VLAN as well. CSR6 will use static one-to-one NAT for its single LAN client; notice that it is using a new inside global address of 10.6.10.14. This address doesn't fit nicely into any subnet anywhere, and it doesn't have to. CSR7 will use the ACL NAT with overload, much like CSR3 and CSR4. The difference is that CSR7 will do this inside of a VRF just for variety, but the configuration is very similar.

```
! CSR6
interface GigabitEthernet2.564
 ip nat inside

interface Dialer6
 ip nat outside

ip nat inside source static 192.168.6.14 10.6.10.14
ip route 0.0.0.0 0.0.0.0 10.56.70.10 10

! CSR7
interface GigabitEthernet2.574
 vrf forwarding CPE
 ip nat inside

interface Dialer7
 vrf forwarding CPE
 ip nat outside

ip nat inside source list ACL_NAT interface Dialer7 vrf CPE overload
ip access-list standard ACL_NAT
 permit 192.168.7.0 0.0.0.255
```

```
ip route vrf CPE 0.0.0.0 0.0.0.0 10.56.70.10 10
```

In order for the PPPoE sessions to come up, we have to configure CSR10. Like CSR9, we will ignore the CGN configuration on CSR10 for now. CSR10 will also be performing VRF operations so we will configure that now as well. Notice that the per-VLAN PPPoE session limit is 3 because all three clients share a LAN, but the per-MAC limit is still 1. One key point is that CSR10 needs a route back to CSR6's inside global address representing XRv4, since it was a made-up address.

```
! CSR10
interface Virtual-Template567
  vrf forwarding LSN

bba-group pppoe PPPOE_LSN
  virtual-template 567
  sessions per-mac limit 1
  sessions per-vlan limit 3

interface GigabitEthernet2.556
  pppoe enable group PPPOE_LSN

ip route vrf LSN 10.6.10.0 255.255.255.0 10.56.70.6
```

We quickly verify the PPPoE sessions are up on CSR10, then perform a quick NAT test from XRv4 to the PPPoE server address via CSR6 and CSR7. Notice that both PPPoE clients are in the same VLAN as expected. CSR10 is not performing NAT currently as we are verifying CSR6 and CSR7 first. Because CSR6 is using static NAT with a non-extended entry, traffic would be allowed in from CSR10. We won't test this again since we tested it with CSR1 and CSR2 earlier, but the NAT table clearly shows the non-extended (permanent) entry. Both CSR6 and CSR7 show the same outside global address of CSR10's PPPoE server address. The inside local from each represents the XRv4 VRF that is used for testing, and the inside global is the post-NAT address.

```
R10#show pppoe session
  2 sessions in LOCALLY_TERMINATED (PTA) State
  2 sessions total
```

Uniq ID	PPPoE SID	RemMAC LocMAC	Port	VT	VA VA-st	State Type
8	8	0050.56a9.ea77	Gi2.556	567	Vi2.2	PTA
		0050.56a9.f961	VLAN:3556		UP	
9	9	0050.56a9.de0d	Gi2.556	567	Vi2.3	PTA
		0050.56a9.f961	VLAN:3556		UP	

```
R6#show ip nat translations
Pro  Inside global      Inside local      Outside local      Outside global
---  10.6.10.14           192.168.6.14      ---               ---
icmp 10.6.10.14:37041    192.168.6.14:37041 10.56.70.10:37041 10.56.70.10:37041
```

```
R7#show ip nat translations
Pro  Inside global      Inside local      Outside local      Outside global
icmp 10.56.70.7:1        192.168.7.14:41137 10.56.70.10:41137 10.56.70.10:1
```

Next, we will configure CGN on CSR10. Using CGN mode, we disable all outside local/global address tracking. We will use an overloaded route-map VRF-aware configuration on CSR10 for variety. The behavior is operationally equivalent to ACL-based overload. The NAT ACL must account for the bogus inside global addresses offered by CSR6 as well as the general PPPoE client subnet used by CSR7.

```
! CSR10
interface GigabitEthernet2.501
 vrf forwarding LSN
 ip nat outside

interface Virtual-Template567
 ip nat inside

ip access-list standard ACL_NAT
 permit 10.56.70.0 0.0.0.255
 permit 10.6.10.0 0.0.0.255

route-map RM_NAT permit 10
 match ip address ACL_NAT

ip nat settings mode cgn
no ip nat settings support mapping outside
ip nat inside source route-map RM_NAT interface Loopback209 vrf LSN overload
```

Testing connectivity from the XRv4 clients all the way to the Internet, we can see two NAT entries on CSR10. Since CSR10 is single-homed to XRv1, there is no need for interface matching in the route-map as demonstrated on CSR9. All post-NAT addresses (inside global) use 209.19.85.10, regardless of inside local or FIB decision.

```
R10#show ip nat translations
Pro  Inside global      Inside local      Outside local      Outside global
icmp 209.19.85.10:2    10.56.70.7:1      ---               ---
icmp 209.19.85.10:1    10.6.10.14:53425  ---               ---
Total number of translations: 2
```

Last, we will configure CSR5. It is multi-homed with two PPPoE connections, one to CSR9 and one to CSR10. It conforms to the address policies of each; that is, it receives an IPCP address from CSR9, while using static addressing towards CSR10. No default routes exist on CSR5 since we will use PBR in conjunction with NAT to influence traffic patterns on a per-source basis. This cannot be achieved without PBR since we are trying to achieve sourced-based routing. Specifically, because inside NAT happens before routing, the NAT process relies on the RIB process to tell it the outgoing interface. Using PBR, we can set this outgoing interface or next-hop, then match it in a NAT route-map to select the

inside global address. In this case, sources in 192.168.5.0/25 (bottom half) will use Dialer59's dynamic PPPoE pool address and be sent to CSR9. The addresses within 192.168.5.128/25 (top half) will use Dialer50's static address and be sent to CSR10. The NAT route-maps don't need to match source addresses because in this case, we know CSR5 has no routes, and can only route via PBR. Since PBR is only matching the sources we care about, we can omit an ACL with our NAT route-maps, despite it being a dangerous practice.

```
! CSR5
interface Dialer50
 ip nat outside

interface Dialer59
 ip nat outside

ip access-list standard ACL_EAST_TOP
 permit 192.168.5.128 0.0.0.127
ip access-list standard ACL_WEST_BOTTOM
 permit 192.168.5.0 0.0.0.127

route-map RM_PBR permit 10
 match ip address ACL_WEST_BOTTOM
 set interface Dialer59
route-map RM_PBR permit 20
 match ip address ACL_EAST_TOP
 set interface Dialer50

route-map RM_EAST permit 10
 match interface Dialer50
route-map RM_WEST permit 10
 match interface Dialer59

interface GigabitEthernet2.554
 ip nat inside
 ip policy route-map RM_PBR

ip nat inside source route-map RM_EAST interface Dialer50 overload
ip nat inside source route-map RM_WEST interface Dialer59 overload
```

When XRv4 sends traffic from 192.168.5.14 and 192.168.5.214, we see two separate inside global addresses for each. The east entry is shown in yellow and the west entry in green. PBR directed traffic from .14 to use CSR9 as the next-hop and .214 to use CSR10 as the next-hop. NAT honored those routing decisions by selecting the proper inside global (post-NAT) address. The PBR packet counters are useful to ensure the policies are configured properly.

```
R5#show ip nat translations
Pro  Inside global      Inside local      Outside local      Outside global
icmp 10.34.59.105:1     192.168.5.14:53425 13.144.2.1:53425 13.144.2.1:1
icmp 10.56.70.5:1     192.168.5.214:49329 13.144.2.1:49329 13.144.2.1:1
```

Total number of translations: 2

```
R5#show route-map RM_PBR
route-map RM_PBR, permit, sequence 10
  Match clauses:
    ip address (access-lists): ACL_WEST_BOTTOM
  Set clauses:
    interface Dialer59
  Policy routing matches: 5 packets, 590 bytes
route-map RM_PBR, permit, sequence 20
  Match clauses:
    ip address (access-lists): ACL_EAST_TOP
  Set clauses:
    interface Dialer50
  Policy routing matches: 5 packets, 590 bytes
```

We can confirm that the packets were routed of the proper interfaces by checking CSR9 and CSR10. Each one should only have one NAT entry; CSR10 has the east entry and CSR9 has the west entry. Notice that the inside local on the CGN router is the same as the inside global on the CPE router. As expected, CSR9 selects one of its global addresses based on its ECMP scheme, while CSR10 uses its only global address. Again, neither CGN router stores outside local/global address information.

```
R9#show ip nat translations
Pro  Inside global      Inside local      Outside local      Outside global
icmp 209.19.85.9:1      10.34.59.105:1    ---               ---
Total number of translations: 1
```

```
R10#show ip nat translations
Pro  Inside global      Inside local      Outside local      Outside global
icmp 209.19.85.10:1     10.56.70.5:1      ---               ---
Total number of translations: 1
```

NAT also supports port forwarding for specific outside to inside communications. This might be valuable for VPN/remote access from the outside. Although unrelated to CGN/LSN, it is a common NAT technique in general. We will quickly demonstrate this on CSR1. This will allow outside hosts, such as CSR8, to access internal resources on specific TCP/UDP ports. We can specify the outside interface or an IP address. If the inside global address is not already present on the router (such as 10.1.8.22), the router automatically creates a dynamic alias for it. In this example, we allow outside routers to access XRv4 via telnet on port 2300 when using CSR1's outside address. Alternatively, they can use SSH on port 2200 using a new address 10.1.8.22.

```
! CSR1
ip nat inside source static tcp 192.168.1.14 23 interface Gig2.518 2300
ip nat inside source static tcp 192.168.1.14 22 10.1.8.22 2200 extendable
```

```
R1#show ip alias
Address Type      IP Address      Port
Interface         10.1.8.1
```

```
Dynamic          10.1.8.22
Interface        192.168.1.1
Interface        192.168.1.2
```

Testing this from CSR8, it can access XRv4 via telnet and SSH. This requires that XRv4 be configured as a telnet and SSH server (not shown).

```
R8#telnet 10.1.8.1 2300
Trying 10.1.8.1, 2300 ... Open

R8#ssh -l cisco -p 2200 10.1.8.22
IMPORTANT: READ CAREFULLY
[snip]
```

CSR1 shows the non-extended entries, which we must have to allow the outside unsolicited traffic. The fully extended entries represents actual NAT sessions currently in place: telnet is highlighted green and SSH is highlighted yellow. Specific sessions spawn these fully-extended entries on a per-session basis. The outside local/global address will always be the initiator of the port-forwarded session, which is CSR8 in this case.

```
R1#show ip nat translations tcp
Pro  Inside global      Inside local      Outside local      Outside global
tcp  10.1.8.1:2300       192.168.1.14:23   ---               ---
tcp  10.1.8.22:2200      192.168.1.14:22   ---               ---
tcp  10.1.8.22:2200      192.168.1.14:22   10.1.8.8:39246    10.1.8.8:39246
tcp  10.1.8.1:2300      192.168.1.14:23   10.1.8.8:33914    10.1.8.8:33914
```

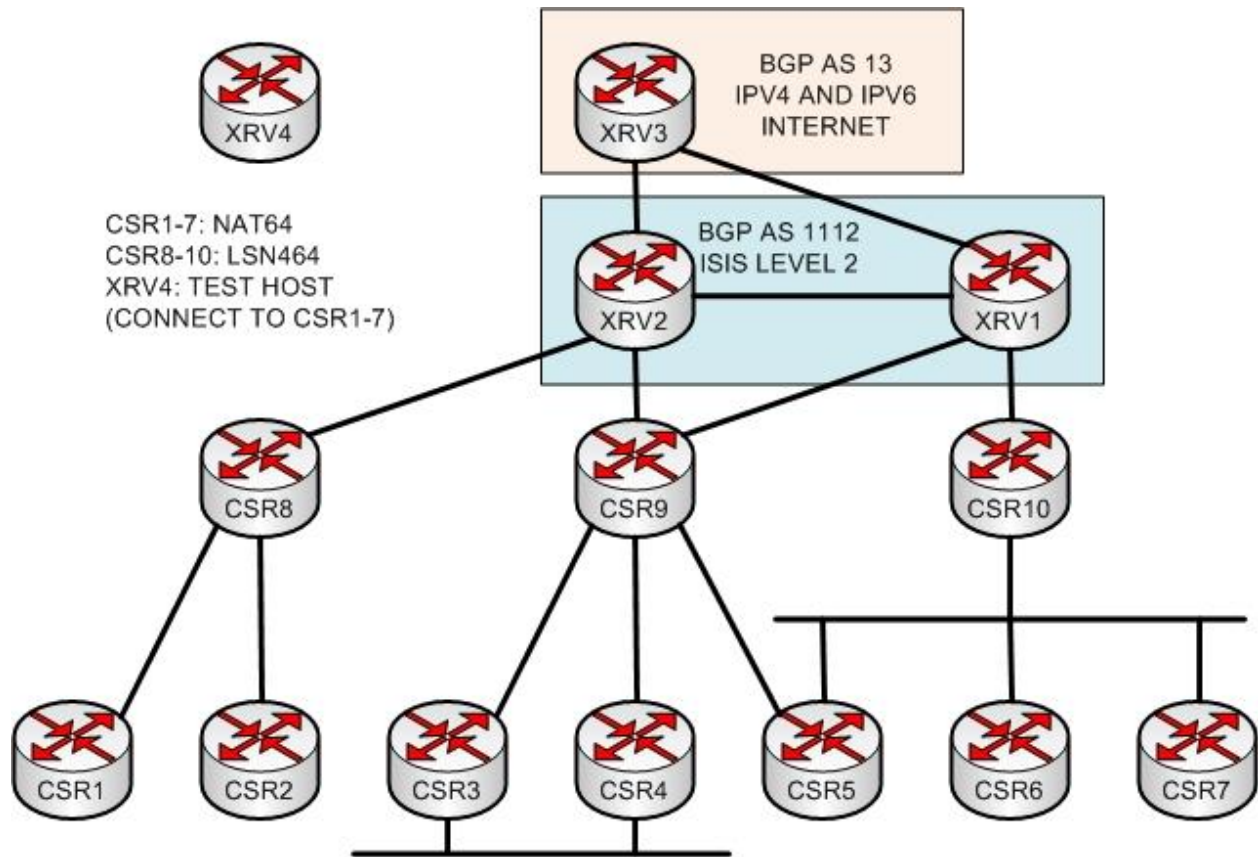
Although not significant to NAT44, IPv6 can run alongside this architecture assuming all devices are dual-stack. Only IPv4 is subject to NAT44, so IPv6 reachability from XRv4's VRFs to the Internet is not impacted. All transports, such as Ethernet, PPPoE, etc can carry IPv6 and NAT44 will simply ignore it. The IPv6 routing in this design is very sloppy to create additional complexity (more elegant ways exist to issue IPv6 prefixes over PPPoE and are shown in the PPPoE section). The routing between CSR8 and its CPEs (CSR1 and CSR2) is static, which is redistributed into IS-IS and later BGP. CSR9 uses OSPFv3 to learn the downstream IPv6 prefixes from CPE devices. PPPoE IPv6 client does not appear to work at all inside a VRF, so CSR7 has no IPv6 connectivity. The NAT64 section details the translation mechanisms for interworking between IPv4 and IPv6 to support LSN architectures. The point is that NAT44 is not in play when native IPv6 hosts are trying to reach the IPv6 Internet.

Additional Reading – Reference configurations "Isn-444"

36.2 NAT64 and NAT464

This test continues from the previous topology with some modifications. From an LSN perspective, a common use case of NAT64 is to deploy a NAT464 architecture. Like NAT444, NAT still occurs twice, but NAT464 introduces NAT64 at the CPE and the LSN. The original IPv4 packet is translated to IPv6 by the CPE, transiting the PE-CE link as IPv6. This can help bypass any firewall rules for east-west traffic that might be filtered by private IPv4 addressing. In the examples seen in this lab, notice that IPv4 has been

completed removed between the CPE and LSN nodes as it is no longer needed. The LSN translates the IPv6 traffic from the CPE back to IPv4 so it can be routed to the IPv4 Internet. Native IPv6 traffic, like before, it not subject to NAT64 and would be routed normally to the IPv6 Internet. The network diagram is below; the same architecture is used for NAT464 as was used for NAT444.



NAT64 comes in two variants: stateful and stateless. These are quickly described below and are demonstrated in great detail later.

1. **Stateful:** Like traditional NAT, this feature maintains a state table of all translations. This includes original and translated IPv4 and IPv6 addresses. Classic NAT features such as port-overloading, ACL-based matching, and static NAT are supported. Because overload is supported, address conservation is good use-case.
2. **Stateless:** This feature maps IPv4 addresses directly into IPv6 addresses and does so without maintaining any state in the router. IPv4 sources and destinations can be configured to share a common prefix or use different ones. This is much less computationally expensive than stateful NAT. There is no conversation of addressing since there is no concept of “overloading”.

On CSR1, where NAT64 will transform IPv4 into IPv6 for traffic heading towards the IPv4 Internet, we will configure stateless NAT64. This is a good option for CPE routers as it is less computationally expensive and less memory hungry. Aside from enabling NAT64 on the interfaces (IPv6 must be enabled

on both the v4 and v6 facing links), there are three key commands. First, we must define a NAT64 prefix. Both the IPv4 source and destination will be mapped to this prefix, and the prefix can vary in length. I find it most straightforward to use a /96 as the IPv4 address will always be encoded into the lowest-order 32 bits. The NAT64 “route” command adds a static route to the IPv4 RIB that directs traffic towards the NAT virtual interface (NVI). This forces traffic towards 13.0.0.0/8 to be NAT64’ed rather than routed normally, which is the same thing that happens for 3001::/96 when the NAT64 prefix is defined. A default route cannot be used for the NAT64 IPv4 route, so we must be selective; this is not totally realistic as the CPE should have a default route to cover all IPv4 prefixes on the Internet. The final IPv6 route is critical from a routing perspective. Since the destination and source both fit into 3001::/96, the router needs to have a more specific route to the destination prefixes. There is a 3001::/96 in the IPv6 RIB directing IPv6 traffic to the NVI, so the router needs a longer match to direct traffic to CSR8. Since the IPv4 destination always begins with 13, I create a route that matches this first octet, and is therefore 8 bits longer than the existing /96.

```
! CSR1
interface GigabitEthernet2.514
  nat64 enable
  ipv6 address FE80::1 link-local
  ipv6 nd ra suppress all

interface GigabitEthernet2.518
  nat64 enable
  ipv6 address FE80::1 link-local
  ipv6 nd ra suppress all

nat64 prefix stateless 3001::/96
nat64 route 13.0.0.0/8 GigabitEthernet2.518
ipv6 route 3001::D00:0/104 GigabitEthernet2.518 FE80::8
```

First, we will verify the NAT64 stateless prefix. If the prefix was invalid or undefined, this output would reveal that issue. Both the v4 and v6 interfaces are using this prefix since we did not define interface-specific prefixes for this test.

```
R1#show nat64 prefix stateless global
Global Stateless Prefix: is valid, 3001::/96
IFs Using Global Prefix
  Gi2.514
  Gi2.518
```

Next, we verify that the NAT64 route is correct. It is mapped to the global prefix, which means traffic towards 13.0.0.0/8 will have its IPv4 source/destination encoded into 3001::/96 when it is sent into the IPv6 network.

```
R1#show nat64 routes prefix 13.0.0.0/8
IPv4 Prefix      Adj. Address      Enabled Output IF
```

```

Global IPv6 Prefix
13.0.0.0/8      0.0.0.5      TRUE      Gi2.518
TRUE      3001::/96

```

```

R1#show ip route 13.0.0.0 255.0.0.0
Routing entry for 13.0.0.0/8
  Known via "static", distance 1, metric 0
  Routing Descriptor Blocks:
    * 0.0.0.5, via NVI0
      Route metric is 0, traffic share count is 1

```

Last, we verify the IPv6 route. Because all post-NAT64 traffic will have an IPv6 destination beginning with 3001::/96, followed by 8-bits representing the first IPv4 octet, we crafted a longer-match /104 route. When the router tries to send traffic towards this final destination, it sends it to CSR8's LL address. Without this manual route, traffic would loop (in software) back into the NVI by following the /96 and be dropped.

```

R1#show ipv6 route 3001::/96 longer-prefixes | begin Appl
a - Application
S   3001::/96 [1/0]
    via ::42, NVI0
S   3001::D00:0/104 [1/0]
    via FE80::8, GigabitEthernet2.518

```

From XRv4, I will send some large ping packets so they are easy to see. Using EPC on CSR1, we can confirm that NAT64 is working correctly. Since this is a stateless translation, there is no translation table associated, so real-time verification is preferred. CSR8 has not been configured for NAT64 yet, so we do not expect the ping to succeed, but we can verify that CSR1 is translating the packets properly. Capturing inbound IPv4 and outbound IPv6 on CSR1, we see the same packet twice. First, we see the original IPv4 source/destination as originated by XRv4. After NAT64 occurs, those same 32-bit addresses are encoded inside the IPv6 prefix 3001::/96 as shown in the outgoing IPv6 packet. Sources are in yellow and destinations are in green. Notice that, since a single global prefix was defined, both the source and destination addresses are contained within the same prefix. This can complicate routing significantly in large networks by making aggregation difficult.

```

R1#show monitor capture CAP buffer detail
  0  518      0.000000    192.168.1.14      -> 13.144.2.1      ICMP
0000:  005056A9 1AAA0050 56A9862A 81000DBA      .PV....PV..*....
0010:  08004500 01F40000 0000FF01 E8C1C0A8      ..E.....
0020:  010E0D90 02010800 25C9709C 0000ABCD      .....%.p....
0030:  ABCDABCD ABCDABCD ABCDABCD ABCDABCD      .....

  1  546      0.000000    3001::*:010E      -> 3001::*:0201      IPv6-Frag
0000:  005056A9 FB1C0050 56A91AAA 81000DBE      .PV....PV.....
0010:  86DD6000 000001E8 2CFE3001 00000000      ..`......,0....
0020:  00000000 0000C0A8 010E3001 00000000      .....0.....

```

```
0030: 00000000 00000D90 02013A00 00000000 .....:.....
```

NAT64 keeps track of packet counters as well, which is a quick way to verify NAT64 operation without EPC. The number says 13 only because I sent several other pings while testing. This output indicates a problem because packets are being translated v4 to v6, but not vice versa, which indicates a lack of bidirectional reachability. This is expected since the rest of the network has not been configured.

```
R1#show nat64 statistics prefix stateless 3001::/96
NAT64 Statistics
3001::/96:
  Packets translated (IPv4 -> IPv6)
    Stateless: 13
    Stateful: 0
    MAP-T: 0
  Packets translated (IPv6 -> IPv4)
    Stateless: 0
    Stateful: 0
    MAP-T: 0
  Packets dropped: 0
```

Before configuring CSR8, we will configure CSR2 with stateless NAT64 using a variation of the method above. The interfaces are almost identical except CSR2 uses an interface-level prefix of length /32. This would allow a multi-homed CPE (like CSR5) to use different NAT64 prefixes for different ISPs if necessary. When a NAT64 prefix length other than /96 is used, two things change. First, bits 64 to 71, inclusive (8 bits), are fixed to 0, which are called universal-bits (u-bits); these are not well documented and are likely reserved for future use. The IPv4 addresses are encoded immediately following the prefix, putting it at bits 32 to 63, inclusive. Unlike the /96, the IPv4 address is in the middle of the IPv6 address, but still technically following the network prefix as it did for the /96. These fields are discussed more when the packet capture is examined, but the immediate relevance is building the IPv6 static route correctly. Without it, NAT64 cannot route traffic upstream. By specifying a v6v4 prefix at the link level and a v4v6 prefix globally, we can effectively use different IPv6 source/destination addresses for my post-NAT traffic. This resolves the network scalability issues related to aggregation we observed on CSR1.

```
! CSR2
interface GigabitEthernet2.524
  nat64 enable
  ipv6 address FE80::2 link-local
  ipv6 nd ra suppress all

interface GigabitEthernet2.528
  nat64 enable
  nat64 prefix stateless v6v4 3002::/32
  ipv6 address FE80::2 link-local
  ipv6 nd ra suppress all
```

```

nat64 prefix stateless v4v6 2222::/32
nat64 route 13.0.0.0/8 GigabitEthernet2.528
ipv6 route 3002:0:D00::/40 GigabitEthernet2.528 FE80::8

```

We will quickly verify the configuration as we did for CSR1. First, we verify that the interface-level prefix is enabled and valid. This v6v4 prefix will determine the post-NAT64 IPv6 destination addresses. The v4v6 global prefix will determine the post-NAT64 IPv6 source addresses.

```

R2#show nat64 prefix stateless v6v4 interfaces
v6v4 Stateless Prefixes
Interface
  NAT64 Enabled Global Prefix
GigabitEthernet2.528
  TRUE FALSE 3002::/32

```

```

R2#show nat64 prefix stateless v4v6 global
Global v4v6 Stateless Prefix: is valid, 2222::/32
IFs Using Global Prefix
  Gi2.524
  Gi2.528

```

Next, we confirm that the NAT64 IPv4 route is directing traffic for 13.0.0.0/8 to the NVI. This is identical to CSR1 and required to direct incoming IPv4 traffic to the NAT64 process.

```

R2#show ip route 13.0.0.0 255.0.0.0
Routing entry for 13.0.0.0/8
  Known via "static", distance 1, metric 0
  Routing Descriptor Blocks:
    * 0.0.0.1, via NVI0
      Route metric is 0, traffic share count is 1

```

Last, we confirm that the IPv6 static route is directing traffic for post-NAT IPv6 flows towards CSR8, rather than looping it back into the NVI. This is the same type of configuration required on CSR1.

```

R2#show ipv6 route 3002::/32 longer-prefixes | begin Appl
  a - Application
S   3002::/32 [1/0]
    via ::44, NVI0
S   3002:0:D00::/40 [1/0]
    via FE80::8, GigabitEthernet2.528

```

I begin sending large pings from XRv4 through CSR2. The NAT64 statistics are increasing, which suggests that it is working. Again, packets are being translated v4 to v6, but not vice versa, which implies a unidirectional forwarding path. Notice that we specifically check the v4v6 prefix of 2222::/32 since the only functional NAT64 direction is v4 to v6 at this point.

```

R2#show nat64 statistics prefix stateless v4v6 2222::/32
NAT64 Statistics
2222::/32:
  Packets translated (IPv4 -> IPv6)
    Stateless: 5
    Stateful: 0
    MAP-T: 0
  Packets translated (IPv6 -> IPv4)
    Stateless: 0
    Stateful: 0
    MAP-T: 0
  Packets dropped: 0

```

Using EPC in the same manner as on CSR1, we can see the IPv4 to IPv6 translation. Source addresses are yellow and destination addresses are green. The IPv4 address is embedded in a different spot as a result of the NAT64 prefix length. The u-bits and suffix are all zeroes as well, but of greater significance, stateless NAT64 is working. The most significant piece of this output is the difference between the IPv6 source and destination prefixes.

```

R2#show monitor capture CAP buffer detailed
  0  518      0.000000   192.168.2.14    ->  13.144.2.1      ICMP
0000:  005056A9 BE8A0050 56A9862A 81000DC4   .PV....PV...*....
0010:  08004500 01F40000 0000FF01 E7C1C0A8   ..E.....
0020:  020E0D90 02010800 95C9009C 0000ABCD   .....
0030:  ABCDABCD ABCDABCD ABCDABCD ABCDABCD   .....

  1  546      0.000000   2222::*:0000    ->  3002::*:0000    IPv6-Frag
0000:  005056A9 FB1C0050 56A9BE8A 81000DC8   .PV....PV.....
0010:  86DD6000 000001E8 2CFE2222 0000C0A8   ..`......,."....
0020:  020E0000 00000000 00003002 00000D90   .....0.....
0030:  02010000 00000000 00003A00 00000000   .....:.....

```

Next, we will configure NAT64 on CSR8, which is an LSN node. This will translate IPv6 traffic from the customers CSR1 and CSR2 into public IPv4 traffic that can be used on the IPv4 Internet. As an LSN, CSR8 should be conservative with IPv4 addressing, so it will use stateful NAT64 to overload a small set of public IPv4 addresses using port numbers as differentiators. Defining an IPv4 pool is similar to regular NAT44, and the pool size is limited to a single IPv4 address. The IPv6 ACL matches the traffic candidate for NAT64, and because the CPEs are performing stateless NAT64, we can create very generic rules. The source/destination of the IPv6 packets from CSR1 and CSR2 are highly deterministic since we know the NAT64 stateless prefixes in use by CSR1 and CSR2. Last, we must define a NAT64 stateful prefix on CSR8. Like the stateless prefix, this creates a static route to the NVI so IPv6 traffic for these destinations can be NAT64'ed. I use per-interface prefixes since the CPEs are using different NAT64 prefixes; the formats should match so addressing can be derived properly. CSR8 needs to know where the IPv4 address is encoded within the IPv6 address to decode it properly, so a generic prefix only works when all CPEs are using the same prefix length. The two IPv6 static routes are used to direct traffic back to the CPEs.

Notice that CSR1's route is somewhat complex since it encompasses the entire NAT64 prefix, then uses a more specific match to keep traffic from looping into the NVI. CSR2 uses an entirely different source IPv6 range, which is cleaner from a routing perspective.

```
! CSR8
interface GigabitEthernet2.518
  nat64 enable
  nat64 prefix stateful 3001::/96
  ipv6 address FE80::8 link-local

interface GigabitEthernet2.528
  nat64 enable
  nat64 prefix stateful 3002::/32
  ipv6 address FE80::8 link-local

interface GigabitEthernet2.582
  nat64 enable
  ipv6 address FE80::8 link-local

ipv6 access-list ACL_LSN_64
  permit ipv6 3001::/96 3001::/96
  permit ipv6 2222::/32 3002::/32

nat64 v4 pool NAT64_POOL 209.19.85.8 209.19.85.8
nat64 v6v4 list ACL_LSN_64 pool NAT64_POOL overload

ipv6 route 2222::/16 GigabitEthernet2.528 FE80::2
ipv6 route 3001::C0A8:100/120 GigabitEthernet2.518 FE80::1
```

One interesting behavior of stateful NAT64 is that the addresses contained within the IPv4 pool cannot exist on any interfaces, even shutdown ones. This is because NAT64 installs an NVI route to the pool addresses which forces them into the NAT process; having a connected route already in the routing table would break this logic. I completely remove Loopback209 (not shown) so that the command below is accepted.

```
R8(config)#nat64 v4 pool NAT64_POOL 209.19.85.8 209.19.85.8
%NAT64: The range 209.19.85.8-209.19.85.8 cannot contain an interface address
(found overlap with address on interface Loopback209)
```

Unlike stateless NAT64, we did not need to add a static route to forward traffic further upstream. CSR8 learns an IS-IS summary route for 13.0.0.0/8 from XRv2, which is sufficient for traffic towards the Internet from the LSN.

```
R8#show ip route 13.0.0.0 255.0.0.0
Routing entry for 13.0.0.0/8
  Known via "isis", distance 115, metric 20, type level-2
```

```
Redistributing via isis 1112
Last update from 10.8.12.12 on GigabitEthernet2.582, 01:56:07 ago
Routing Descriptor Blocks:
* 10.8.12.12, from 0.0.0.0, 01:56:07 ago, via GigabitEthernet2.582
  Route metric is 20, traffic share count is 1
```

Return traffic is broken as a result of removing Loopback209. This was advertised into IS-IS earlier which ultimately allowed the Internet to know about it. Although summarization was used in previous labs to influence routing decisions, I am illustrating a situation where such alternate paths would not exist. There is technically a /30 covering this prefix, known to XRv3, but that is beyond the scope of the current problem.

```
RP/0/0/CPU0:XRv3#show cef 209.19.85.8/32
%Prefix not found or IP is not running. VRF default.
```

```
RP/0/0/CPU0:XRv3#show cef 209.19.85.8
209.19.85.8/30, version 480, internal 0x5000001 0x0 (ptr 0xa1413cf4) [1], 0x0
(0x0), 0x0 (0x0)
local adjacency 10.12.13.12
  Prefix Len 30, traffic index 0, precedence n/a, priority 4
  via 10.12.13.12, 2 dependencies, recursive, bgp-ext [flags 0x6020]
  path-idx 0 NHID 0x0 [0xa14144f4 0x0]
  next hop 10.12.13.12 via 10.12.13.12/32
```

Fortunately, NAT64 adds this IPv4 pool to the RIB as a static route, so we can simply redistribute it into IS-IS to achieve reachability. From NAT64 perspective, this is critical since returning traffic is directed into the NVI to be translated from v4 to v6. From a routing perspective, it gives us a mechanism to redistribute the NAT64 pool to IPv4 routing protocols. Once complete, the Internet router sees this exact match prefix from XRv1.

```
! CSR8
ip prefix-list PL_PUBLIC seq 5 permit 209.19.85.8/32
```

```
route-map RM_STATIC_TO_ISIS permit 10
  match ip address prefix-list PL_PUBLIC
```

```
router isis 1112
  redistribute static ip route-map RM_STATIC_TO_ISIS
```

```
RP/0/0/CPU0:XRv3#show cef 209.19.85.8/32
209.19.85.8/32, version 486, internal 0x5000001 0x0 (ptr 0xa14147f4) [1], 0x0
(0x0), 0x0 (0x0)
local adjacency 10.11.13.11
  Prefix Len 32, traffic index 0, precedence n/a, priority 4
  via 10.11.13.11, 2 dependencies, recursive, bgp-ext [flags 0x6020]
  path-idx 0 NHID 0x0 [0xa1414574 0x0]
  next hop 10.11.13.11 via 10.11.13.11/32
```

With basic IPv4 routing resolved, we will continue verifying stateful NAT64. Verifying the prefix is similar to stateless NAT64 as it shows the user-defined prefixes as valid and applied to the proper interfaces. Again, we used different interface-level prefixes on the LSN because the downstream clients have different NAT64 stateless prefixes in this example.

```
R8#show nat64 prefix stateful interfaces
Stateful Prefixes
Interface
    NAT64 Enabled Global Prefix
GigabitEthernet2.518
    TRUE                FALSE  3001::/96
GigabitEthernet2.528
    TRUE                FALSE  3002::/32
```

Next, we verify that the pool was configured properly. It consists of exactly 1 useable IPv4 address.

```
R8#show nat64 pools name NAT64_POOL
Protocol HSL ID      Name
    Is Single Range
    Ranges
IPv4      1          NAT64_POOL
    TRUE          (209.19.85.8 - 209.19.85.8)
    209.19.85.8 - 209.19.85.8
```

As a troubleshooting aid when dealing with routing issues, we can see the static routes created by NAT64 for IPv6 prefixes and IPv4 pools using the commands below. A single /32 sufficiently covers the IPv4 pool in this case. An IPv4 pool may require a collection of static routes to cover the entire range. Additionally, each IPv6 prefix causes NAT64 to add a static route to direct traffic into the NVI.

```
R8#show nat64 prefix stateful static-routes
Stateful Prefixes
NAT64 Prefix
    Static Route Ref-Count
3001::/96
    1
3002::/32
    1
```

```
R8#show nat64 pools routes
Pools configured: 1
Protocol HSL ID      Name
    Is Single Range
    Ranges
IPv4      1          NAT64_POOL
    TRUE          (209.19.85.8 - 209.19.85.8)
    209.19.85.8 - 209.19.85.8
```


Static Routes for Range: 1 209.19.85.8/32

Because we know how traffic is being encoded inside IPv6, we can quickly check the IPv6 FIB to ensure return traffic is routed correctly. For CSR2 this is easier since 2222::/16 is not used for anything else. For CSR1, we must ensure the longer-match route was configured correctly. I highlight the embedded IPv4 private source addresses for clarity (192.168.1.14 or 192.168.2.14 for CSR1 and CSR2, respectively).

```
R8#show ipv6 cef 3001::C0A8:10E
3001::C0A8:100/120
  nexthop FE80::1 GigabitEthernet2.518
```

```
R8#show ipv6 cef 2222:0:c0a8:020E::
2222::/16
  nexthop FE80::2 GigabitEthernet2.528
```

To test it, I begin sending large pings from XRv4 towards the Internet. I begin with using CSR2 as the gateway since it is the simpler case. This time, the pings are successful as shown below. With EPC enabled on CSR8, we can see the incoming IPv6 packet from CSR2 and the outgoing IPv4 packet from CSR8. The public IPv4 address of 209.19.85.8 is used as the IPv4 source with a destination of the original IPv4 Internet host. Because CSR8 had a matching IPv6 stateful prefix for CSR2's NAT64 stateless prefix, it knew exactly where to find the IPv4 addresses.

```
R8#show monitor capture CAP buffer detailed
 0 546      0.000000      2222*:0000      -> 3002*:0000      IPv6-Frag
0000: 005056A9 FB1C0050 56A9BE8A 81000DC8      .PV....PV.....
0010: 86DD6000 000001E8 2CFD2222 0000C0A8      ..`.....,..."....
0020: 020E0000 00000000 00003002 00000D90      .....0.....
0030: 02010000 00000000 00003A00 00000000      .....:.....

 1 518      0.000000      209.19.85.8      -> 13.144.2.1      ICMP
0000: 005056A9 0E6F0050 56A9FB1C 81000DFE      .PV..o.PV.....
0010: 08004500 01F40000 0000FD01 865CD113      ..E.....\..
0020: 55080D90 02010800 96620003 0000ABCD      U.....b.....
0030: ABCDABCD ABCDABCD ABCDABCD ABCDABCD      .....
```

Viewing the translation table, the output is somewhat similar to the deprecated NAT-PT feature. There are 4 key pieces of information. The original IPv4 address is the destination that XRv4 was trying to reach initially (Internet host). When this was translated into an IPv6 address, it was wrapped inside the 3002::/32 prefix, which is highlighted. The second row shows the original IPv6 address, which was the source of the traffic entering CSR8. After translations, the post-NAT64 address came from the IPv4 pool and represents CSR8's public address. A simpler way to read this table would be to read it by columns. The IPv4 column shows the destination/source of the post-NAT IPv4 packet (outgoing). The IPv6 column shows that destination/source of the pre-NAT IPv6 packet (incoming). This "shortcut" is confirmed by the EPC capture above.

```
R8#show nat64 translations
```

Proto	Original IPv4	Translated IPv4
	Translated IPv6	Original IPv6
icmp	13.144.2.1:1	[3002:0:d90:201::]:24732
	209.19.85.8:1	[2222:0:c0a8:20e::]:24732

We can check the statistics to confirm that all 5 packets in the ping test succeeded. Gig2.528 translated packets from v6 to v4, while Gig2.582 translated them back from v4 to v6.

```
R8#show nat64 statistics interface gig2.528
```

```
NAT64 Statistics
```

```
Interface Statistics
```

```
GigabitEthernet2.528 (IPv4 configured, IPv6 configured):
```

```
Packets translated (IPv4 -> IPv6)
```

```
Stateless: 0
```

```
Stateful: 0
```

```
MAP-T: 0
```

```
Packets translated (IPv6 -> IPv4)
```

```
Stateless: 0
```

```
Stateful: 5
```

```
MAP-T: 0
```

```
Packets dropped: 0
```

```
R8#show nat64 statistics interface gig2.582
```

```
NAT64 Statistics
```

```
Interface Statistics
```

```
GigabitEthernet2.582 (IPv4 configured, IPv6 configured):
```

```
Packets translated (IPv4 -> IPv6)
```

```
Stateless: 0
```

```
Stateful: 5
```

```
MAP-T: 0
```

```
Packets translated (IPv6 -> IPv4)
```

```
Stateless: 0
```

```
Stateful: 0
```

```
MAP-T: 0
```

```
Packets dropped: 0
```

Quickly backtracking to CSR2 (and clearing the NAT64 statistics for clarity), we can check the v4v6 prefix for translations. It shows that 5 packets were translated in both directions, which suggests connectivity is functional. When we viewed this output before CSR8 was configured, we only saw v4 to v6 translations. Seeing bidirectional translations on the CPE is a good sign that the LSN is functional.

```
R2#clear nat64 statistics
```

```
RP/0/0/CPU0:XRv4#ping vrf 2 13.144.2.1 size 500
```

Type escape sequence to abort.

Sending 5, 500-byte ICMP Echos to 13.144.2.1, timeout is 2 seconds:

!!!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 1/57/89 ms

R2#show nat64 statistics prefix stateless v4v6 2222::/32

NAT64 Statistics

2222::/32:

Packets translated (IPv4 -> IPv6)

Stateless: 5

Stateful: 0

MAP-T: 0

Packets translated (IPv6 -> IPv4)

Stateless: 5

Stateful: 0

MAP-T: 0

Packets dropped: 0

When we attempt the same test using CSR1, it fails. We did not trace the detailed routing process for CSR2 since it “just worked”, so for completeness, we will trace it using CSR1 in the forwarding path.

RP/0/0/CPU0:XRv4#ping vrf 1 13.144.2.1 size 500

Type escape sequence to abort.

Sending 5, 500-byte ICMP Echos to 13.144.2.1, timeout is 2 seconds:

.....

Success rate is 0 percent (0/5)

We used EPC earlier to identify that CSR1 is sending traffic to CSR8 with IPv6 source 3001::C0A8:010E (192.168.1.14) and IPv6 destination 3001::0D90:0201 (13.144.2.1). Based on this, CSR8 should be routing this traffic to the NVI. The NAT64-installed static route for the IPv6 stateful prefix 3001::/96 appears to be working.

R8#show ipv6 cef 3001::d90:201

3001::/96

nexthop ::100.0.0.1 NVI0

Once in the NVI, the NAT64 ACL is processed. This particular flow matches sequence 10. Assuming NAT64 worked, CSR8 also has a route to the final destination as well. Without this route, the earlier test using CSR2 would have also failed.

R8#show access-lists ACL_LSN_64

IPv6 access list ACL_LSN_64

permit ipv6 3001::/96 3001::/96 sequence 10

permit ipv6 2222::/32 3002::/32 sequence 20

R8#show ip cef 13.144.2.1

13.0.0.0/8

```
nexthop 10.8.12.12 GigabitEthernet2.582
```

At this point, NAT64 should be occurring, but it isn't. Instead, all 5 of the pings were dropped.

```
R8#show nat64 statistics prefix stateful 3001::/96
NAT64 Statistics
3001::/96:
  Packets translated (IPv4 -> IPv6)
    Stateless: 0
    Stateful: 0
    MAP-T: 0
  Packets translated (IPv6 -> IPv4)
    Stateless: 0
    Stateful: 0
    MAP-T: 0
  Packets dropped: 5
```

Even with all NAT64 debugging enabled, no information is given. I send more pings via XRv4 (no shown) yet we are presented with no debug output, no NAT64 translations, and another 5 dropped packets.

```
R8#debug nat64 all
NAT64 debugging is on
[no output]
```

```
R8#show nat64 translations
Proto  Original IPv4      Translated IPv4
      Translated IPv6  Original IPv6
-----
```

```
Total number of translations: 0
```

```
R8#show nat64 statistics prefix stateful 3001::/96 | include dropped
Packets dropped: 10
```

At this point, the problem is impossible to identify without an educated guess. The only significant difference between CSR1 and CSR2 is that CSR1's source IPv6 address also falls within the NAT64 stateful prefix range. That is to say, when CSR8 tries to perform NAT64 on a specific destination inside the range 3001::/96, it also sees the IPv6 packet's source falls within the same range. I assume that this is the reason NAT64 silently discards these packets as a loop control mechanism. After hours of testing and troubleshooting (absent any formal documentation), I believe this is the reason for the failure. I leave CSR1 broken for reference; if we were going to fix it, we would need to use a separate stateless NAT64 prefix for source and destination as CSR2 does.

We progress to configuring CSR3 and CSR4 with stateless NAT64. They both serve a single client on a LAN so their configurations will be similar. For brevity, I do not show the NAT44 removal and IPv4 cleanup. I also do not show enabling IPv6 or NAT64 on the interfaces as this is very basic. There is

nothing preventing CSR3 and CSR4 from sending traffic to the same IPv6 destination, so the NAT64 v6v4 prefix applied to the dialers can be identical. The v4v6 prefix applied globally, used for IPv6 source addresses, should be different to guarantee optimal routing. The prefix lengths between v6v4 and v4v6 prefixes do not have to match. I create the standard 13.0.0.0/8 route to the NVI for incoming IPv4 traffic, and enable a minor fragmentation option. IPv6 can insert fragmentation headers for fragmented packets, and NAT64 does this automatically. That is why the EPC captures shown earlier displayed “IPv6-Frag” versus “IPv6-ICMP”. This option prevents that fragmentation header from being imposed and I enable it on CSR3 only. The /104 static routes are similar to the one used on CSR1; this directs the IPv6 traffic towards the LSN rather than looping it through the NVI indefinitely.

```
! CSR3
interface Dialer3
  nat64 prefix stateless v6v4 3034:BEEF:2BAD::/96

nat64 prefix stateless v4v6 3034:3::/32
nat64 settings fragmentation header disable
nat64 route 13.0.0.0/8 Dialer3
ipv6 route 3034:BEEF:2BAD:0:0:0:0d00:0/104 Dialer3

! CSR4
interface Dialer4
  nat64 prefix stateless v6v4 3034:BEEF:2BAD::/96

nat64 prefix stateless v4v6 3034:4::/32
nat64 route 13.0.0.0/8 Dialer4
ipv6 route 3034:BEEF:2BAD:0:0:0:0d00:0/104 Dialer4
```

Since HSRP was originally tracking the presence of an IPv4 default route, I remove the tracking mechanism entirely since it is beyond the scope of NAT64 and complicates our test. CSR3 is active for both IPv4 and IPv6 due to having a higher priority.

```
R3#show standby brief
                P indicates configured to preempt.
                |
Interface      Grp   Pri P State   Active          Standby          Virtual IP
Gi2.534        344   105 P Active   local          192.168.34.4    192.168.34.254
Gi2.534        346   105 P Active   local          FE80::4         FE80::254
```

The CSR3 and CSR4 configurations are almost identical, so I only verify CSR3 for brevity. Below, I quickly verify the global v4v6 prefix (sources) and interface-specific v6v4 prefix (destination). CSR4 would show a different v4v6 prefix but the same v6v4 prefix.

```
R3#show nat64 prefix stateless v4v6 global
Global v4v6 Stateless Prefix: is valid, 3034:3::/32
IFs Using Global Prefix
  Gi2.534
```

Di3

```
R3#show nat64 prefix stateless v6v4 interfaces
v6v4 Stateless Prefixes
Interface
NAT64 Enabled Global Prefix
```

```
Dialer3
TRUE FALSE 3034:BEEF:2BAD::/96
```

Using a specific IPv4 Internet host, I confirm that this traffic is forwarded into the NVI so that NAT64 can occur. This static route was installed via the “nat64 route” command.

```
R3#show ip route 13.144.2.1
Routing entry for 13.0.0.0/8
Known via "static", distance 1, metric 0
Routing Descriptor Blocks:
* 0.0.0.1, via NVI0
Route metric is 0, traffic share count is 1
```

After NAT64, the router needs to send traffic upstream to CSR9. The longer-match /104 static routes accomplish this nicely. The same route can be used on both CSR3 and CSR4 since they use a common v6v4 (destination) prefix, as seen in the configuration snippets earlier.

```
R3#show ipv6 route 3034:beef:2bad::/96 longer-prefixes | begin Appl
a - Application
S 3034:BEEF:2BAD::/96 [1/0]
via ::44, NVI0
S 3034:BEEF:2BAD::D00:0/104 [1/0]
via Dialer3, directly connected
```

Using EPC while pinging from XRv4, we can confirm CSR3 is functioning correctly; an IPv4 packet enters and an IPv6 packet exits. The IPv6 source/destination addresses are from different major networks, which is a requirement for stateful NAT64 to work at the LSN. Notice that the IPv6 packet is 546 bytes, which was the same length as the IPv6 packet seen earlier with the fragmentation header. That IPv6 fragmentation header is 8 bytes, as is the PPPoE encapsulation (cyan), which is why the sizes appear the same now. This proves that the IPv6 fragmentation header was removed from the IPv6 packet after NAT64 occurred; if it was not removed, the size would have been 554. We will see this on CSR4 next.

```
R3#show monitor capture CAP buffer detail
4 518 2.020003 192.168.34.14 -> 13.144.2.1 ICMP
0000: 00000C9F F1580050 56A9862A 81000DCE .....X.PV...*....
0010: 08004500 01F40002 0000FF01 C7BFC0A8 ..E.....
0020: 220E0D90 02010800 D5C6C09C 0002ABCD ".....
0030: ABCDABCD ABCDABCD ABCDABCD ABCDABCD .....
```

```

5 546 2.020003 00:50:56:A9:8C:CF -> 00:50:56:A9:D6:72 PPPoE Session
Stage
0000: 005056A9 D6720050 56A98CCF 81000DD3 .PV..r.PV.....
0010: 88641100 0005020A 00576000 000001E0 .d.....W`.....
0020: 3AFE3034 0003C0A8 220E0000 00000000 :.04....".....
0030: 00003034 BEEF2BAD 00000000 00000D90 ..04...+.....
0040: 02018000 1E5DC09C 0001ABCD ABCDABCD .....].....

```

I shut down CSR3's client-side LAN interface so that CSR4 becomes the HSRP active router. Using EPC, I perform the same test. Everything is identical except for two details. The IPv6 source address is using the prefix 3034:4::/32 versus 3034:3::/32, and the packet is 8 bytes larger due to the IPv6 fragmentation header being added by NAT64. This header is shown in pink for completeness, but is not examined/discussed in detail. The IPv6 packet is 554 bytes as expected when accounting for this additional header.

```

R4#show monitor capture CAP buffer detail
3 518 2.413965 192.168.34.14 -> 13.144.2.1 ICMP
0000: 00000C9F F1580050 56A9862A 81000DCE .....X.PV..*....
0010: 08004500 01F40000 0000FF01 C7C1C0A8 ..E.....
0020: 220E0D90 02010800 A5C8F09C 0000ABCD ".....
0030: ABCDABCD ABCDABCD ABCDABCD ABCDABCD .....

```

```

4 554 2.413965 00:50:56:A9:2C:57 -> 00:50:56:A9:D6:72 PPPoE Session
Stage
0000: 005056A9 D6720050 56A92C57 81000DDD .PV..r.PV.,W....
0010: 88641100 00040212 00576000 000001E8 .d.....W`.....
0020: 2CFE3034 0004C0A8 220E0000 00000000 ,.04....".....
0030: 00003034 BEEF2BAD 00000000 00000D90 ..04...+.....
0040: 02013A00 00000000 00008000 EE5CF09C ...:.....\..

```

Both CSR3 and CSR4 reported 5 packets translated from IPv4 to IPv6, which is accurate given the quick testing we conducted. This is a good indication that CPE stateless NAT64 is functioning properly in one direction so far. I bring CSR3's LAN interface back up before continuing.

```

R3#show nat64 statistics global | begin Global
Global Stats:
Packets translated (IPv4 -> IPv6)
Stateless: 5
Stateful: 0
MAP-T: 0
Packets translated (IPv6 -> IPv4)
Stateless: 0
Stateful: 0
MAP-T: 0

```

```

R4#show nat64 statistics global | begin Global
Global Stats:

```

```

Packets translated (IPv4 -> IPv6)
  Stateless: 5
  Stateful: 0
  MAP-T: 0
Packets translated (IPv6 -> IPv4)
  Stateless: 0
  Stateful: 0
  MAP-T: 0

```

CSR9 will perform stateful NAT64 just as CSR8 does. CSR9 has two public addresses available that are not contiguous, so I will create two separate NAT64 rules with different ACLs. This can be used for “port sharing” so that a single public IPv4 “less overloaded” as there are multiple public IPv4 addresses. CSR9’s public loopbacks are completely removed in order to create the NAT pools, and the resulting NVI-generated IPv4 static routes are redistributed into IS-IS. Since two pools are created, two NAT64 rules are required. One rule maps traffic from CSR3 to 209.19.85.9 and the other maps traffic from CSR4 to 209.19.85.11. With this design (assuming CSR3 and CSR4 had some kind of FHRP load sharing scheme, which they don’t currently), each public address on CSR9 is less saturated with ports. Because CSR3 and CSR4 use the same v6v4 (destination) prefix, CSR9 can define a global NAT64 stateful prefix. CSR8 had to use these per-interface since CSR1 and CSR2 used different destination IPv6 addresses. With PPPoE, this would be impossible since there is only a single “interface” towards the CPE routers, so having the PPPoE clients use the same v6v4 (destination) prefix is required. Enabling NAT64 on the virtual-template and upstream transit links is not shown for brevity.

```

! CSR9
ipv6 access-list ACL_NAT_FROM_3
  permit ipv6 3034:3::/32 3034:BEEF:2BAD::/96

ipv6 access-list ACL_NAT_FROM_4
  permit ipv6 3034:4::/32 3034:BEEF:2BAD::/96

nat64 prefix stateful 3034:BEEF:2BAD::/96
nat64 v4 pool NAT64_POOL_11 209.19.85.11 209.19.85.11
nat64 v4 pool NAT64_POOL_9 209.19.85.9 209.19.85.9
nat64 v6v4 list ACL_NAT_FROM_3 pool NAT64_POOL_9 overload
nat64 v6v4 list ACL_NAT_FROM_4 pool NAT64_POOL_11 overload

```

Rather than use static routes on CSR9 for return traffic to 3034::3/96 and 3034:4::/96, we can use an IPv6 IGP; in this case, we use OSPFv3. It is already in place for IPv6 routing to the IPv6 Internet, and the static routes were auto-generated on the CPEs for stateless NAT64. I redistribute these static NVI routes into OSPFv3 on the CPEs, which provides dynamic return paths for CSR9 (the LSN). I use the most specific prefix-list possible to capture both source prefixes to keep the configuration the same.

```

! CSR3 and CSR4
ipv6 prefix-list PL_V4V6 seq 5 permit 3034::/29 ge 32 le 32

```



```
route-map RM_STATIC_TO_OSPF permit 10
  match ipv6 address prefix-list PL_V4V6
```

```
router ospfv3 9
  address-family ipv6 unicast
    redistribute static route-map RM_STATIC_TO_OSPF
```

We confirm that CSR3 and CSR4 properly redistribute these from static into OSPF. CSR9 learns these routes as OSPF external type-2 routes, as expected.

```
R3#show ospfv3 ipv6 rib redistribution
      OSPFv3 9 address-family ipv6 (router-id 192.168.34.3)
3034:3::/32, type 2, metric 20, tag 0, from static
  via ::43, NVI0
```

```
R4#show ospfv3 ipv6 rib redistribution
      OSPFv3 9 address-family ipv6 (router-id 192.168.34.4)
3034:4::/32, type 2, metric 20, tag 0, from static
  via ::43, NVI0
```

```
R9#show ipv6 route 3034::/29 longer-prefixes | begin Appl
  a - Application
OE2 3034:3::/32 [110/20]
    via FE80::3, Virtual-Access2.2
OE2 3034:4::/32 [110/20]
    via FE80::4, Virtual-Access2.1
```

Before testing NAT64, we quickly configure the validate flow inside CSR9. When packets arrive, whether from CSR3 or CSR4, they are directly into the NVI as expected.

```
R9#show ipv6 route 3034:BEEF:2BAD::13.144.2.1
Routing entry for 3034:BEEF:2BAD::/96
  Known via "static", distance 1, metric 0
  Route count is 1/1, share count 0
  Routing paths:
    ::100.0.0.1, NVI0
    Last updated 00:02:45 ago
```

Assuming there is a match against one of the NAT64 ACLs, the traffic is routed to the final IPv4 destination. CSR9 has two ECMP paths for this via XRv1 and XRv2, which is fine.

```
R9#show ip cef 13.144.2.1
13.0.0.0/8
  nexthop 10.9.11.11 GigabitEthernet2.591
  nexthop 10.9.12.12 GigabitEthernet2.592
```

Returning traffic will be destined to either 209.19.85.9 or 209.19.85.11 depending on whether it came from CSR3 or CSR4. In either case, traffic is directed into the NVI as a result of the NAT64 IPv4 pools. The other public addresses are shown below from the other LSNs (learned via IS-IS) but are not relevant for this test. Assuming there was an existing NAT64 translation, traffic would be routed back towards CSR3 or CSR4. This return-flow routing occurs via the OSPFv3 routes we verified earlier.

```
R9#show ip route 209.19.85.8 255.255.255.252 longer-prefixes | begin Gate
Gateway of last resort is not set
    209.19.85.0/32 is subnetted, 4 subnets
i L2      209.19.85.8 [115/20] via 10.9.12.12, 03:28:12, GigabitEthernet2.592
S         209.19.85.9 [1/0] via 100.0.0.2, NVI0
i L2      209.19.85.10 [115/20] via 10.9.11.11, 05:30:58, GigabitEthernet2.591
S         209.19.85.11 [1/0] via 100.0.0.2, NVI0
```

To test it, we use XRv4 again inside VRF 34 which is currently using CSR3 as its default gateway. The pings immediately work, which implies bidirectional connectivity has been achieved.

```
RP/0/0/CPU0:XRv4#ping vrf 34 13.144.2.1 size 500
Type escape sequence to abort.
Sending 5, 500-byte ICMP Echos to 13.144.2.1, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/59/89 ms
```

Checking the NAT64 translation table on CSR9, we can see that the original IPv4 packet was destined to 13.144.2.1, an Internet host, and its source was adjusted to be 209.19.85.9. This makes sense since the traffic came from CSR3 (3034:3::/32) and the NAT64 rule specified 209.19.85.9 as the address to use.

```
R9#show nat64 translations
Proto  Original IPv4      Translated IPv4
      Translated IPv6  Original IPv6
-----
icmp    13.144.2.1:1      [3034:beef:2bad::d90:201]:156
        209.19.85.9:1  [3034:3:c0a8:220e::]:156
```

The NAT64 statistics also indicate that 5 packets were translated bidirectionally across CSR9.

```
R9#show nat64 statistics global | begin Global
Global Stats:
    Packets translated (IPv4 -> IPv6)
        Stateless: 0
        Stateful: 5
        MAP-T: 0
    Packets translated (IPv6 -> IPv4)
        Stateless: 0
        Stateful: 5
        MAP-T: 0
```

We can use EPC for additional verification. It does not appear capable of capturing the PPPoE traffic entering the router on the server side (incoming IPv6 packet), so we will limit the verification to the outgoing IPv4 packet. Since ECMP was in play, CSR9 had to decide whether to forward the packet to XRv1 or XRv2. Using show commands, we confirm that XRv2 was selected, and EPC proves it with the destination MAC address in cyan. As usual, I show the source address in yellow (209.19.85.9) and the destination address in green (13.144.2.1).

```
R9#show ip cef exact-route 209.19.85.9 13.144.2.1
209.19.85.9 -> 13.144.2.1 =>IP adj out of GigabitEthernet2.592, addr
10.9.12.12
```

```
R9#show ip arp 10.9.12.12
Protocol Address Age (min) Hardware Addr Type Interface
Internet 10.9.12.12 99 0050.56a9.0e6f ARPA Gig2.592
```

```
R9#show monitor capture CAP buffer detailed
0 518 0.000000 209.19.85.9 -> 13.144.2.1 ICMP
0000: 005056A9 0E6F0050 56A9D672 81000E08 .PV..o.PV..r....
0010: 08004500 01F40000 4000FD01 465BD113 ..E.....@...F[..
0020: 55090D90 02010800 96640001 0000ABCD U.....d.....
0030: ABCDABCD ABCDABCD ABCDABCD ABCDABCD .....
```

We expect CSR4 to work identically, except that post-NAT64 address of 209.19.85.11 should be used. Shutting down CSR3's LAN interface, we can force traffic through CSR4. XRv4 can still ping the Internet host (not shown) which means the HSRP failover is functioning as designed. Checking the NAT translations, we see that the other public IPv4 address is used. The statistics also show an additional 5 packets successfully translated, for a total of 10 packets. All of this output is indicative of success so far.

```
R9#show nat64 translations
Proto Original IPv4 Translated IPv4
Translated IPv6 Original IPv6
-----
icmp 13.144.2.1:1 [3034:beef:2bad::d90:201]:20636
209.19.85.11:1 [3034:4:c0a8:220e::]:20636
```

```
Total number of translations: 1
```

```
R9#show nat64 statistics global | begin Global
Global Stats:
Packets translated (IPv4 -> IPv6)
Stateless: 0
Stateful: 10
MAP-T: 0
Packets translated (IPv6 -> IPv4)
Stateless: 0
Stateful: 10
```

MAP-T: 0

Using EPC again, we ensure the packet headers are correct. This time, CSR9 selects XRv1 as the next-hop towards the Internet host since the CEF ECMP function is a combination of both source and destination addresses. Only the source was modified and CEF made a different routing decision. We confirm this decision by checking the destination MAC address in the Ethernet frame. The IPv4 source and destination are in yellow and green, respectively, and represent the correct addresses. Before continuing, CSR3 is brought back online.

```
R9#show ip cef exact-route 209.19.85.11 13.144.2.1
209.19.85.11 -> 13.144.2.1 =>IP adj out of GigabitEthernet2.591, addr
10.9.11.11
```

```
R9#show ip arp 10.9.11.11
Protocol Address Age (min) Hardware Addr Type Interface
Internet 10.9.11.11 124 0050.56a9.2dc6 ARPA Gig2.591
```

```
R9#show monitor capture CAP buffer detailed
 0 518 0.000000 209.19.85.11 -> 13.144.2.1 ICMP
0000: 005056A9 2DC60050 56A9D672 81000E07 .PV.-..PV..r....
0010: 08004500 01F40000 0000FD01 8659D113 ..E.....Y..
0020: 550B0D90 02010800 96640001 0000ABCD U.....d.....
0030: ABCDABCD ABCDABCD ABCDABCD ABCDABCD .....
```

Next, we will configure CSR5 for stateless NAT64 in a dual-homed scenario. NAT64 does not currently support route-maps, so the source-based PBR approach cannot be used as it was for NAT444. Instead, we will use destination-based load sharing using the longest-match routing principle with respect to directing traffic into the NVI. Traffic to 13.0.0.0/8 will be sent to CSR9 via Dialer59, but traffic to 13.144.2.24/29 will be sent to CSR10 via Dialer50. You cannot configure multiple NAT64 routes for the same destination in ECMP fashion, but using longest-match to approximate some load sharing is accepted. This could be valuable when a large portion of the upstream traffic flow is directed to subnet 13.144.2.24/29 and it warrants using a separate path for better load distribution. The v4v6 prefix 5000:AAAA::/64 is used for the IPv6 source addressing and there are separate v6v4 (destination) prefixes facing CSR9 and CSR10. Dialer59 uses v6v4 prefix 3034:BEEF:2BAD::/96 just like CSR3 and CSR4, which simplifies the configuration on CSR9. Dialer50 uses v4v6 prefix 3005:DEED:567A::/64 towards CSR10.

```
! CSR5
interface Dialer50
 nat64 prefix stateless v6v4 3005:DEED:567A::/64

interface Dialer59
 nat64 prefix stateless v6v4 3034:BEEF:2BAD::/96

nat64 prefix stateless v4v6 5000:AAAA::/64
```

```

nat64 route 13.0.0.0/8 Dialer59
nat64 route 13.144.2.24/29 Dialer50

```

Before continuing, we verify this configuration. CSR5 has two NAT64-generated static routes to direct traffic towards 13.0.0.0/8 and 13.144.2.24/29 into the NVI. Although not terribly significant, the next-hops are different for each NAT64 route. This differentiation allows NAT64 to know which v6v4 (destination) prefix to use since the “NAT64” route command identified an outgoing interface. The NVIO is a “multi-access network” in this way since a next-hop is specified along with the outgoing interface.

```

R5#show ip route 13.0.0.0 255.0.0.0 longer-prefixes | begin Gate
Gateway of last resort is not set
    13.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
S       13.0.0.0/8 [1/0] via 0.0.0.1, NVIO
S       13.144.2.24/29 [1/0] via 0.0.0.2, NVIO

```

We can see the mapping of IPV4 NAT64 routes to NVIO next-hops using the show command below. This information serves as input to the IPv6 process that generates the static routes.

```

R5#show nat64 routes
IPv4 Prefix      Adj. Address      Enabled Output IF
Global IPv6 Prefix
13.0.0.0/8       0.0.0.1          TRUE      Di59
13.144.2.24/29   0.0.0.2          TRUE      Di50

```

The IPv6 routing is somewhat difficult on CSR5. We need two static IPv6 routes since CSR5 has two uplinks; the masks on these routes “matches” what the NAT64 IPv4 routes were. Dialer59 services traffic inside 13.0.0.0/8, while Dialer50 services traffic inside 13.144.2.24/29. Because each of these uplinks has a different v6v4 (destination) prefix length, the IPv6 routes will differ significantly. Dialer59 is easy since the first 96 bits are the prefix with the last 32 as the IPv4 address. Dialer50’s IPv4 address begins at bit 73 because there are 8 u-bits immediately following the /64 prefix. Because this is complicated, I highlight those bits in pink below. The overall route prefix becomes 64 (prefix) + 8 (u-bits) + 29 (IPv4 mask) = 101. Following the route addition, the v4v6 (source) prefix must be redistributed into OSPFv3; this was required on CSR3 and CSR4 as well. This will allow CSR9 to learn the prefix in order to route returning traffic to CSR5. CSR10 has not yet been configured so we do not expect to have connectivity to 13.144.2.24/29 at this time.

```

! CSR5
ipv6 route 3034:BEEF:2BAD::D00:0/104 Dialer59
ipv6 route 3005:DEED:567A:0:000D:9002:1800:0/101 Dialer50

ipv6 prefix-list PL_V4V6 seq 5 permit 5000:AAAA::/64

route-map RM_STATIC_TO_OSPF permit 10
 match ipv6 address prefix-list PL_V4V6

```

```

router ospfv3 9
 address-family ipv6 unicast
  redistribute static route-map RM_STATIC_TO_OSPF

```

We quickly verify that the redistribution and IPv6 route additions were successful. CSR5 redistributes the v4v6 (source) prefix into OSPFv3 and CSR9 learns it as an OSPF E2 route. The manual static routes remain in CSR5's local IPv6 RIB to forward traffic towards the ISPs.

```
R5#show ospfv3 ipv6 rib redistribution
```

```

      OSPFv3 9 address-family ipv6 (router-id 192.168.5.5)

```

```

5000:AAAA::/64, type 2, metric 20, tag 0, from static
  via ::43, NVI0

```

```

R5#show ipv6 route 3034:BEEF:2BAD::D00:0/104 longer-prefixes | begin Appl
  a - Application

```

```

S   3034:BEEF:2BAD::D00:0/104 [1/0]
    via Dialer59, directly connected
S   3005:DEED:567A:0:D:9002:1800:0/101 [1/0]
    via Dialer50, directly connected

```

```
R9#show ipv6 route 5000:AAAA::/64
```

```

Routing entry for 5000:AAAA::/64
  Known via "ospf 9", distance 110, metric 20, type extern 2
  Redistributing via isis 1112
  Route count is 1/1, share count 0
  Routing paths:
    FE80::5, Virtual-Access2.3
    Last updated 00:01:21 ago

```

In order for CSR9 to perform NAT64 against traffic from CSR5, we must update CSR9 to account for this new v4v6 (source) prefix 5000:AAAA::/64 that is used by CSR5. CSR9 has two public, post-NAT IPv4 addresses. CSR5 must be configured to use one of them; I simply extend one of the NAT64 ACLs to include CSR5's source prefix. This will cause traffic from CSR5 to use 209.19.85.9 just as CSR3 does.

```

! CSR9
ipv6 access-list ACL_NAT_FROM_3
 sequence 20 permit ipv6 5000:AAAA::/64 3034:BEEF:2BAD::/96

```

There is one minor clean-up we should perform on CSR9 before continuing. Currently, CSR9 is redistributing OSPFv3 into ISIS as this was required for raw IPv6 Internet reachability, beyond the scope of NAT entirely. Because CSR9 is learning NAT64 stateless v4v6 (source) prefixes such as 3034:3::/32, 3034:4::/32, and 5000:AAAA::/64, we should apply a filter to prevent these from leaking to the Internet. Normally these prefixes would be unique local addresses from FC00::/7, but I used public IPv6 prefixes for demonstration. I apply a simple route-map that does not include external routes; only OSPF internal

routes are candidates for redistribution. Checking the local ISIS LSP, we can see that the NAT64 v4v6 (source) prefixes are not included. These PE-CE transit links should never leak to the Internet as their only purpose is to support the NAT64 architecture.

```
! CSR9
route-map RM_OSPF_TO_ISIS_V6 permit 10
  match route-type internal

router isis 1112
  address-family ipv6
    redistribute ospf 9 route-map RM_OSPF_TO_ISIS_V6

R9#show isis database detail R9.00-00 | begin IPv6 Add
IPv6 Address: 2001:10:9:12::9
Metric: 0          IPv6 (MT-IPv6) 2001:192:168:34::/64
Metric: 0          IPv6 (MT-IPv6) 2001:192:168:5::/64
```

To test this new NAT64 configuration, I begin sending pings from XRv4 to 13.144.2.1 which succeed (not shown). This will match the 13.0.0.0/8 NAT64 route on CSR5 and should be sent towards CSR9. Since we have never used a /64 for the source prefix before, I use EPC on CSR5 to see exactly where the IPv4 address is encoded. The source addresses are yellow and the destination addresses are green. The destination IPv4 address is encoded into the v6v4 prefix specified to Dialer59 yet it appears 1 byte (8 bits) after the /64 prefix ends, colored grey. These are the u-bits described earlier, and in this case, they are set to zero. This separates the IPv4 address from the NAT64 v6v4 prefix which makes it a little harder to read, but otherwise has no significance at present. Notice that CSR5 includes the IPv6 fragmentation header since we did not explicitly disable it (pink) as we did on CSR3.

```
R5#show monitor capture CAP buffer detail
  2  518    0.011993  192.168.5.14    ->  13.144.2.1      ICMP
0000:  005056A9 DC630050 56A9862A 81000DE2  .PV..c.PV..*....
0010:  08004500 01F40000 0000FF01 E4C1C0A8  ..E.....
0020:  050E0D90 02010800 D5C8C09C 0000ABCD  .....
0030:  ABCDABCD ABCDABCD ABCDABCD ABCDABCD  .....

  3  554    0.011993  00:50:56:A9:DC:63 -> 00:50:56:A9:D6:72 PPPoE Session
Stage
0000:  005056A9 D6720050 56A9DC63 81000DE7  .PV..r.PV..c....
0010:  88641100 00070212 00576000 000001E8  .d.....W`.....
0020:  2CFE5000 AAAA0000 000000C0 A8050E00  ,.P.....
0030:  00003034 BEEF2BAD 00000000 00000D90  ..04..+.....
0040:  02013A00 00000000 00008000 7FDBC09C  ...:.....
```

Checking CSR9, we can see a stateful translation entry for this ICMP flow. The “Original IPv6” address is the most interesting here since this clearly shows the u-bit offset that EPC reveal above. The IPv4 address is still encoded correctly but it doesn’t “look nice”. Just before the “c0” in this address are two zeroes to represent those u-bits; the router doesn’t show these leading zeroes for brevity. Also notice

that the “Translated IPv6” address is the same post-NAT address used when traffic comes from CSR3 (green); this is the result of modifying the “FROM_3” ACL.

```
R9#show nat64 translations
```

Proto	Original IPv4	Translated IPv4
	Translated IPv6	Original IPv6
icmp	13.144.2.1:1	[3034:beef:2bad::d90:201]:45212
	209.19.85.9:1	[5000:aaaa::c0:a805:e00:0]:45212

CSR5 will not be able to send traffic to 13.144.2.24/29 until CSR10 is configured. We quickly configure CSR10 with a straightforward stateful NAT64 configuration similar to CSR9 and CSR10. In the NAT444 lab, CSR10 was running VRF-lite since NAT44 is VRF-aware. NAT64 is not currently VRF-aware, so all VRFs are removed from CSR10 and only the global table is used (VRF removal and subsequent cleanup not shown). CSR10 will match traffic using the prefixes defined by CSR5’s stateless NAT64 process and map those to its single public IPv4 address of 209.19.85.10. This pool spawns a static route to the NVI and must be redistributed into IS-IS so upstream routers can forward returning traffic back towards the LSN.

```
! CSR10
ipv6 access-list ACL_NAT
 permit ipv6 5000:AAAA::/64 3005:DEED:567A::/64

nat64 v4 pool NAT64_POOL 209.19.85.10 209.19.85.10
nat64 v6v4 list ACL_NAT pool NAT64_POOL overload
nat64 prefix stateful 3005:DEED:567A::/64

ip prefix-list PL_PUBLIC seq 5 permit 209.19.85.10/32

route-map RM_STATIC_TO_ISIS permit 10
 match ip address prefix-list PL_PUBLIC

router isis 1112
 redistribute static ip route-map RM_STATIC_TO_ISIS
```

First, we check to ensure CSR10 creates the IPv4 static route for the NAT64 pool (pointing to the NVI) and redistributes it into IS-IS. This ensures that the Internet routers will have reachability to this post-NAT public address on the LSN.

```
R10#show ip route 209.19.85.10
Routing entry for 209.19.85.10/32
  Known via "static", distance 1, metric 0
  Redistributing via isis 1112
  Advertised by isis 1112 metric-type internal level-2 route-map
RM_STATIC_TO_ISIS
  Routing Descriptor Blocks:
    * 100.0.0.2, via NVI0
```


Route metric is 0, traffic share count is 1

```
R10#show isis database detail R10.00-00 | include 209
```

```
Metric: 0          IP 209.19.85.10/32
```

```
RP/0/0/CPU0:XRv3#show cef 209.19.85.10/32
```

```
209.19.85.10/32, version 494, internal 0x5000001 0x0 (ptr 0xa1413ef4) [1],  
0x0 (0x0), 0x0 (0x0)
```

```
local adjacency 10.11.13.11
```

```
Prefix Len 32, traffic index 0, precedence n/a, priority 4
```

```
via 10.11.13.11, 2 dependencies, recursive, bgp-ext [flags 0x6020]
```

```
path-idx 0 NHID 0x0 [0xa1414574 0x0]
```

```
next hop 10.11.13.11 via 10.11.13.11/32
```

Next, we ensure that the NAT64 stateful prefix has a corresponding static route directly IPv6 into the NVI. This captures traffic from CSR5 by matching its v6v4 (destination) stateless NAT64 prefix.

```
R10#show ipv6 route 3005:DEED:567A::/64
```

```
Routing entry for 3005:DEED:567A::/64
```

```
Known via "static", distance 1, metric 0
```

```
Route count is 1/1, share count 0
```

```
Routing paths:
```

```
::100.0.0.1, NVI0
```

```
Last updated 00:13:41 ago
```

Because there is no good way for CSR10 to configure a static route towards CSR5's v4v6 (source) prefix, I use EIGRPv6 on CSR5 to advertise it dynamically. CSR6, CSR7, and CSR10 were already using EIGRPv6 for this purpose so I extend it to CSR5 as well. It is not directly related to NAT64, but CSR10 must have a route back to 5000:AAAA::/64. Since CSR5 had to redistribute this static route into OSPFv3 for CSR9, the prefix-list/route-map constructs are already in place. EIGRPv6 can simply reuse them even if the route-map name is unintuitive. After the static route is redistributed into EIGRP, we check the topology locally on CSR5 to verify. I also verify that CSR10 learns this is an EIGRP external route, which allows it to send return traffic from the IPv4 Internet back to CSR5. We confirm that the virtual-access interface actually belongs to CSR5 by checking the PPPoE and interface MAC address details.

```
! CSR5
```

```
router eigrp LSN
```

```
address-family ipv6 unicast autonomous-system 10
```

```
topology base
```

```
redistribute static route-map RM_STATIC_TO_OSPF
```

```
R5#show eigrp address-family ipv6 topology 5000:AAAA::/64
```

```
EIGRP-IPv6 VR(LSN) Topology Entry for AS(10)/ID(192.168.5.5) for
```

```
5000:AAAA::/64
```

```
State is Passive, Query origin flag is 1, 1 Successor(s), FD is 12030537142
```

```
Descriptor Blocks:
```

```
::43, from Rstatic, Send flag is 0x0
```

```

Composite metric is (12030537142/0), route is External
Vector metric:
  Minimum bandwidth is 56 Kbit
  Total delay is 5000000000 picoseconds
  Reliability is 255/255
  Load is 1/255
  Minimum MTU is 9216
  Hop count is 0
External data:
  Originating router is 192.168.5.5 (this system)
  AS number of route is 0
  External protocol is Static, external metric is 0
  Administrator tag is 0 (0x00000000)

```

```

R10#show ipv6 route 5000:AAAA::/64
Routing entry for 5000:AAAA::/64
  Known via "eigrp 10", distance 170, metric 145188571, type external
Route count is 1/1, share count 0
Routing paths:
  FE80::21E:14FF:FEFF:9B00, Virtual-Access2.2
  Last updated 00:07:16 ago

```

```

R10#show pppoe session
  3 sessions in LOCALLY_TERMINATED (PTA) State
  3 sessions total

```

Uniq ID	PPPoE	RemMAC	Port	VT	VA	State
	SID	LocMAC			VA-st	Type
6	6	0050.56a9.ea77	Gi2.556	567	Vi2.1	PTA
		0050.56a9.f961	VLAN:3556		UP	
10	10	0050.56a9.dc63	Gi2.556	567	Vi2.2	PTA
		0050.56a9.f961	VLAN:3556		UP	
11	11	0050.56a9.de0d	Gi2.556	567	Vi2.3	PTA
		0050.56a9.f961	VLAN:3556		UP	

```

R5#show interfaces gigabitEthernet2 | include bia
Hardware is CSR vNIC, address is 0050.56a9.dc63 (bia 0050.56a9.dc63)

```

To protect the IPv6 Internet from these “bogus” v4v6 (source) NAT64 prefixes, I use the same filtering technique on CSR10 as I did on CSR9 except using a prefix-list. The “route-type” matching inside the route-map does not appear to work for EIGRPv6 at this time. This configuration ensures that the NAT64 prefixes are not redistributed from EIGRP into IS-IS.

```

! CSR10
ipv6 prefix-list PL_V4V6 seq 5 permit 5000:AAAA::/64

route-map RM_EIGRP_TO_ISIS_V6 deny 10
match ipv6 address prefix-list PL_V4V6

```

```

route-map RM_EIGRP_TO_ISIS_V6 per 1000

router isis 1112
 address-family ipv6
  redistribute eigrp 10 route-map RM_EIGRP_TO_ISIS_V6

R10#show isis database detail R10.00-00 | begin IPv6 Add
IPv6 Address: 2001:10:10:11::10
Metric: 0          IPv6 (MT-IPv6) 2001:192:168:5::/64
Metric: 0          IPv6 (MT-IPv6) 2001:192:168:6::/64
Metric: 0          IPv6 (MT-IPv6) 2001:192:168:7::/64

```

At this point, XRv4 should have reachability to 13.144.2.25 which is a host address on the Internet. This traffic should be routed via CSR10. The source will be changed mapped into v4v6 prefix 5000:AAAA::/64 and the destination will be mapped into v6v4 prefix 3005:DEED:567A::/64 associated with Dialer50. CEF directs traffic for 13.144.2.24/20 to the NVI using address 0.0.0.2, which is tied to Dialer50.

```

R5#show ip cef 13.144.2.25
13.144.2.24/29
  nexthop 0.0.0.2 NVI0

R5#show nat64 route
IPv4 Prefix      Adj. Address      Enabled Output IF
  Global IPv6 Prefix
13.0.0.0/8        0.0.0.1          TRUE      Di59
13.144.2.24/29    0.0.0.2          TRUE      Di50

R5#show nat64 prefix stateless v4v6 global
Global v4v6 Stateless Prefix: is valid, 5000:AAAA::/64
IFs Using Global Prefix
  Gi2.554
  Di50
  Di59

R5#show nat64 prefix stateless v6v4 interfaces
v6v4 Stateless Prefixes
Interface
  NAT64 Enabled Global Prefix
Dialer50
  TRUE          FALSE  3005:DEED:567A::/64
Dialer59
  TRUE          FALSE  3034:BEEF:2BAD::/96

```

After NAT64 occurs, the destination IPv6 address will fit within a relatively specific route which routes traffic towards the IPv4 prefix 13.144.2.24/29 towards Dialer50. Without this route, traffic would loop back into the NVI and be dropped. These are the static routes we verified earlier but are shown again for completeness.

```

R5#show ipv6 route 3005:DEED:567A::/64 longer-prefixes | begin Appl
    a - Application
S   3005:DEED:567A::/64 [1/0]
    via ::44, NVIO
S   3005:DEED:567A:0:D:9002:1800:0/101 [1/0]
    via Dialer50, directly connected

```

Using EPC on CSR5, we can see the incoming LAN IPv4 and outgoing WAN IPv6 packet. Both the IPv6 source and destination include the u-bits, shown in grey, since their prefix lengths are shorter than /96. Even the /32 prefixes included the u-bits, but it was less significant since the IPv4 addresses were encoded first. If the prefix length is less than or equal to 64, the u-bits will always exist. If they exist, u-bits are always in the same place as described earlier. Source addresses are shown in yellow with destination addresses in green, as usual. For consistency, the PPPoE encapsulation is shown in cyan and the IPv6 fragmentation header is shown in pink, but neither one is relevant to this discussion.

```

R5#show monitor capture CAP buffer detailed
  2  518      2.615966  192.168.5.14    ->  13.144.2.25      ICMP
0000:  005056A9 DC630050 56A9862A 81000DE2    .PV..c.PV..*....
0010:  08004500 01F40000 0000FF01 E4A9C0A8    ..E.....
0020:  050E0D90 02190800 D5C8C09C 0000ABCD    .....
0030:  ABCDABCD ABCDABCD ABCDABCD ABCDABCD    .....

  3  554      2.615966  00:50:56:A9:DC:63 -> 00:50:56:A9:F9:61 PPPoE Session
Stage
0000:  005056A9 F9610050 56A9DC63 81000DE4    .PV..a.PV..c....
0010:  88641100 000A0212 00576000 000001E8    .d.....W`.....
0020:  2CFE5000 AAAA0000 000000C0 A8050E00    ,.P.....
0030:  00003005 DEED567A 0000000D 90021900    ..0...Vz.....
0040:  00003A00 00000000 00008000 9BC0C09C    .....

```

Quickly checking the statistics, we can see 5 packets being translated v4-to-v6 inbound on the LAN and v6-to-v4 inbound on the WAN (return flow). This is an indication that NAT64 is working.

```

R5#show nat64 statistics interface gig2.554
NAT64 Statistics
Interface Statistics
  GigabitEthernet2.554 (IPv4 configured, IPv6 configured):
    Packets translated (IPv4 -> IPv6)
      Stateless: 5
      Stateful: 0
      MAP-T: 0
    Packets translated (IPv6 -> IPv4)
      Stateless: 0
      Stateful: 0
      MAP-T: 0
    Packets dropped: 0

```

```

R5#show nat64 statistics interface dialer50
NAT64 Statistics
Interface Statistics
  Dialer50 (IPv4 not configured, IPv6 configured):
    Packets translated (IPv4 -> IPv6)
      Stateless: 0
      Stateful: 0
      MAP-T: 0
    Packets translated (IPv6 -> IPv4)
      Stateless: 5
      Stateful: 0
      MAP-T: 0
    Packets dropped: 0

```

CSR10 will receive packets towards 3005:DEED:567A:0:D:9002:1900:0 which was the destination seen in the EPC above. This is the “Translated IPv4” address from CSR10’s perspective seen below. The public IPv4 address used comes from the NAT v4 pool and is 209.19.85.10, which is redistributed into ISIS. XRv4 can now reach all Internet hosts using both CSR9 and CSR10.

```

R10#show nat64 translations
Proto  Original IPv4      Translated IPv4
      Translated IPv6  Original IPv6
-----
icmp    13.144.2.25:1      [3005:deed:567a:0:d:9002:1900:0]:57500
        209.19.85.10:1  [5000:aaaa::c0:a805:e00:0]:57500

```

Assuming that CSR6 is not capable of stateless NAT64, we can use stateful NAT64 with static mappings. Although this does not support address overloading, it does allow us to specify direct mappings between IPv4 and IPv6 addresses. When traffic enters CSR6 destined to 13.144.2.1, CSR6 will encode the destination inside of the already-supported destination prefix 3005:DEED:567A::/64. We manually have to determine the correct destination IPv6 address because CSR10 will care. CSR10 is already expecting traffic for this stateful prefix as it supports CSR5. We have to expand CSR10’s NAT ACL to account for a new source, CSR6, as well as advertise this new source into EIGRP.

```

! CSR6
nat64 prefix stateful 6006:FEED:BEAD::/96
nat64 v6v4 static 3005:DEED:567A:0:D:9002:100:0 13.144.2.1

ipv6 prefix-list PL_V4V6 seq 5 permit 6006:FEED:BEAD::/96

route-map RM_STATIC_TO_EIGRP permit 10
  match ipv6 address prefix-list PL_V4V6

router eigrp LSN
  address-family ipv6 unicast autonomous-system 10
  topology base

```

```
redistribute static route-map RM_STATIC_TO_EIGRP
```

CSR10 should learn this as an EIGRP external route from CSR6. We confirm this using some PPPoE/ARP show commands before continuing. Without this route, CSR10 would not be able to deliver returning traffic to CSR6.

```
R10#show ipv6 route 6006:FEED:BEAD::/96
Routing entry for 6006:FEED:BEAD::/96
  Known via "eigrp 10", distance 170, metric 145188571, type external
Route count is 1/1, share count 0
Routing paths:
  FE80::21E:BDFF:FE69:6200, Virtual-Access2.3
  Last updated 00:09:51 ago
```

```
R10#show pppoe session
  3 sessions in LOCALLY_TERMINATED (PTA) State
  3 sessions total
```

Uniq	ID	PPPoE	RemMAC	Port	VT	VA	State
		SID	LocMAC			VA-st	Type
6	6		0050.56a9.ea77	Gi2.556	567	Vi2.1	PTA
			0050.56a9.f961	VLAN:3556		UP	
10	10		0050.56a9.dc63	Gi2.556	567	Vi2.2	PTA
			0050.56a9.f961	VLAN:3556		UP	
11	11		0050.56a9.de0d	Gi2.556	567	Vi2.3	PTA
			0050.56a9.f961	VLAN:3556		UP	

```
R6#show interfaces gigabitEthernet2 | include bia
Hardware is CSR vNIC, address is 0050.56a9.de0d (bia 0050.56a9.de0d)
```

We will expand the NAT ACL to include the new source, and adjust the EIGRP-to-ISIS redistribution to prevent this “bogus” route from polluting the IPv6 Internet. This is the same procedure we configured earlier so that CSR10 could support CSR5. I show the full PL and ACL configuration for completeness; there is one line for each supported CPE router at this point. CSR10’s logic hardly changes. After updating the NAT ACL, traffic from 6006:FEED:BEAD::/96 will be candidate for NAT using 209.19.85.10 as the post-NAT public address. In this way, both the CPE and LSN are technically using stateful NAT64.

```
! CSR10
ipv6 prefix-list PL_V4V6 seq 5 permit 5000:AAAA::/64
ipv6 prefix-list PL_V4V6 seq 10 permit 6006:FEED:BEAD::/96

ipv6 access-list ACL_NAT
  permit ipv6 5000:AAAA::/64 3005:DEED:567A::/64
  permit ipv6 6006:FEED:BEAD::/96 3005:DEED:567A::/64
```

When traffic enters CSR6 towards 13.144.2.1, it is directed to the NVI via a static route. This was installed by NAT64 when the static mapping was added. The route is a /32 host route so no other Internet destinations are included.

```
R6#show ip cef 13.144.2.1
13.144.2.1/32
  nexthop 100.0.0.1 NVI0
```

After NAT64 occurs, the resulting IPv6 packet is forwarded to CSR10 by following the default route. This route was originated by CSR10 inside EIGRPv6 from the last lab, and CSR6 will rely on it here. Unlike stateless NAT64, there is no static NVI route created for this destination IPv6 address; there is no concept of a v4v6 (source) prefix with static mappings. The source address used comes from the stateful NAT prefix 6006:FEED:BEAD::/96 defined above. CSR6 does install a static route for this prefix since the returning traffic must be directed into the NVI for translation back to IPv4.

```
R6#show ipv6 cef 3005:DEED:567A:0:D:9002:100:0
::/0
  nexthop FE80::21E:14FF:FE6F:8300 Dialer6
```

```
R6#show ipv6 route 6006:FEED:BEAD::/96
Routing entry for 6006:FEED:BEAD::/96
  Known via "static", distance 1, metric 0
  Redistributing via eigrp 10
  Route count is 1/1, share count 0
  Routing paths:
    ::100.0.0.1, NVI0
    Last updated 00:19:27 ago
```

Since CSR6 has a static, stateful mapping. This will appear in the NAT64 translation table when no traffic is flowing. This behavior is consistent with NAT44 as this static entry works in either direction. These are both effectively “destination” addresses when examined from an upstream traffic flow perspective. Traffic to 13.144.2.1 has its destination changed to the “Original IPv6” address and its source changed to the NAT64 stateful prefix encoded with the original IPv4 source address (which varies).

```
R6#show nat64 translations
Proto Original IPv4      Translated IPv4
      Translated IPv6    Original IPv6
-----
---      ---      ---
      13.144.2.1      3005:deed:567a:0:d:9002:100:0
```

When XRv4 sends traffic to 13.144.2.1, CSR6 translates it as shown below. The “Original IPv4” address is the original source, which is XRv4. This is encoded into the last 32 bits of the NAT64 /96 stateful prefix as examined many times thus far. The “destination” IPv4/IPv6 addresses are fixed and do not change, with the exception of a port number used for demultiplexing.

```
R6#show nat64 translations
```

Proto	Original IPv4	Translated IPv4
	Translated IPv6	Original IPv6
---	---	---
	13.144.2.1	3005:deed:567a:0:d:9002:100:0
icmp	192.168.6.14:12444	[6006:feed:bead::c0a8:60e]:12444
	13.144.2.1:12444	[3005:deed:567a:0:d:9002:100:0]:12444

Checking CSR10, NAT64 is performed based on traffic entering the router with a destination inside the stateful NAT64 prefix of 3005:DEED:567A::/64. Because CSR6 manually formatted this correctly in the static NAT64 mapping, CSR10 is able to perform dynamic NAT64 by extracting the original IPv4 destination address of 13.144.2.1. The “Original IPv6” address, which carries to the encoded IPv4 original source, doesn’t get decoded again. It is only used for mapping returning flows destined to 209.19.85.10 back to the proper downstream customer.

```
R10#show nat64 translations
```

Proto	Original IPv4	Translated IPv4
	Translated IPv6	Original IPv6
---	---	---
icmp	13.144.2.1:1	[3005:deed:567a:0:d:9002:100:0]:16540
	209.19.85.10:1	[6006:feed:bead::c0a8:60e]:16540

Quickly checking CSR6’s NAT64 statistics, we can see packets being translated bidirectionally on the proper interfaces.

```
R6#show nat64 statistics interface gig2.564
```

```
NAT64 Statistics
```

```
Interface Statistics
```

```
GigabitEthernet2.564 (IPv4 configured, IPv6 configured):
```

```
Packets translated (IPv4 -> IPv6)
```

```
Stateless: 0
```

```
Stateful: 5
```

```
MAP-T: 0
```

```
Packets translated (IPv6 -> IPv4)
```

```
Stateless: 0
```

```
Stateful: 0
```

```
MAP-T: 0
```

```
Packets dropped: 0
```

```
R6#show nat64 statistics interface dialer6
```

```
NAT64 Statistics
```

```
Interface Statistics
```

```
Dialer6 (IPv4 not configured, IPv6 configured):
```

```
Packets translated (IPv4 -> IPv6)
```

```
Stateless: 0
```



```

    Stateful: 0
    MAP-T: 0
    Packets translated (IPv6 -> IPv4)
    Stateless: 0
    Stateful: 5
    MAP-T: 0
    Packets dropped: 0

```

With this static NAT64 architecture in place, accessing new Internet hosts is simple. To access XRv3's remaining 3 interfaces, we simply need to add a static mappings for each. We manually determine the post-NAT IPv6 destination address so that CSR10 can correctly decode the embedded IPv4 address. From XRv4, I ping all 4 destinations and examine the translation table. All of the pings succeed.

```

! CSR6
nat64 v6v4 static 3005:DEED:567A:0:D:9002:900:0 13.144.2.9
nat64 v6v4 static 3005:DEED:567A:0:D:9002:1100:0 13.144.2.17
nat64 v6v4 static 3005:DEED:567A:0:D:9002:1900:0 13.144.2.25

```

```
R6#show nat64 translations protocol icmp
```

Proto	Original IPv4 Translated IPv6	Translated IPv4 Original IPv6
icmp	192.168.6.14:61596 13.144.2.9:61596	[6006:feed:bead::c0a8:60e]:61596 [3005:deed:567a:0:d:9002:900:0]:61596
icmp	192.168.6.14:8348 13.144.2.25:8348	[6006:feed:bead::c0a8:60e]:8348 [3005:deed:567a:0:d:9002:1900:0]:8348
icmp	192.168.6.14:53404 13.144.2.1:53404	[6006:feed:bead::c0a8:60e]:53404 [3005:deed:567a:0:d:9002:100:0]:53404
icmp	192.168.6.14:156 13.144.2.17:156	[6006:feed:bead::c0a8:60e]:156 [3005:deed:567a:0:d:9002:1100:0]:156

On CSR7, I demonstrate using the well-known prefix (WKP) for stateful NAT64. The WKP prefix is 64:FF9B::/96 and cannot be disabled. Any router with NAT64 enabled, even stateless NAT64, will have a static route directing traffic to this destination into the NVI. CSR7 will use this as the destination IPv6 prefix as CSR10 will automatically direct this into its NVI without us having to configure it. The NAT64 manually-specified prefix configured on CSR7 will be used as the source address when NAT64 generates IPv6 packets. The basic NVI static route redistribution into EIGRP is shown again which is similar to CSR6.

```

! CSR7
nat64 prefix stateful 7007:FEED:BEAD::/96
nat64 v6v4 static 64:FF9B::D90:201 13.144.2.1

ipv6 prefix-list PL_V4V6 seq 5 permit 7007:FEED:BEAD::/96

route-map RM_STATIC_TO_EIGRP permit 10
 match ipv6 address prefix-list PL_V4V6

```

```

router eigrp LSN
 address-family ipv6 unicast autonomous-system 10
 topology base
 redistribute static route-map RM_STATIC_TO_EIGRP

```

CSR7 must define a longer-match static route to ensure traffic towards this destination isn't looped into its local NVI. Since CSR7 is running NAT64 also, we need to totally ignore the 64:FF9B::/96 prefix from a routing perspective. A /97 is not very specific, but it's good enough for this test case.

```

! CSR7
ipv6 route 64:FF9B::/97 Dialer7

R7#show ipv6 route 64:FF9b::/96 longer-prefixes | begin Appl
  a - Application
S   64:FF9B::/96 [1/0]
    via ::100.0.0.2, NVI0
S   64:FF9B::/97 [1/0]
    via Dialer7, directly connected

```

CSR10 must expand its list of "bogus" IPv6 prefixes to filter from ISIS, along with its NAT ACL. The NAT ACL now matches traffic from CSR7 towards the WKP. Notice that CSR10 does not have to define a new NAT64 prefix, since 64:FF9B::/96 is automatically created. The IPv6 static route for the WKP towards the NVI is automatically generated as soon as NAT64 is enabled as we saw on CSR7 also.

```

! CSR10
ipv6 prefix-list PL_V4V6 seq 5 permit 5000:AAAA::/64
ipv6 prefix-list PL_V4V6 seq 10 permit 6006:FEED:BEAD::/96
ipv6 prefix-list PL_V4V6 seq 15 permit 7007:FEED:BEAD::/96

ipv6 access-list ACL_NAT
 permit ipv6 5000:AAAA::/64 3005:DEED:567A::/64
 permit ipv6 6006:FEED:BEAD::/96 3005:DEED:567A::/64
 permit ipv6 7007:FEED:BEAD::/96 64:FF9B::/96

R10#show ipv6 route 64:FF9B::/96
Routing entry for 64:FF9B::/96
  Known via "static", distance 1, metric 0
  Route count is 1/1, share count 0
  Routing paths:
    ::100.0.0.2, NVI0
    Last updated 00:44:27 ago

```

After CSR7's redistribution of its local NAT64 prefix into EIGRP, CSR10 now has a path back to CSR7's NAT64 source prefix. We confirm this using the PPPoE and interface-level show commands.

```

R10#show ipv6 route 7007:FEED:BEAD::/96

```

```

Routing entry for 7007:FEED:BEAD::/96
  Known via "eigrp 10", distance 170, metric 145188571, type external
  Route count is 1/1, share count 0
  Routing paths:
    FE80::21E:49FF:FECA:A400, Virtual-Access2.2
    Last updated 00:00:14 ago

```

```

R10#show pppoe session
  3 sessions in LOCALLY_TERMINATED (PTA) State
  3 sessions total

```

Uniq ID	PPPoE SID	RemMAC LocMAC	Port	VT	VA VA-st	State Type
8	8	0050.56a9.ea77	Gi2.556	567	Vi2.2	PTA
		0050.56a9.f961	VLAN:3556		UP	
7	7	0050.56a9.dc63	Gi2.556	567	Vi2.1	PTA
		0050.56a9.f961	VLAN:3556		UP	
9	9	0050.56a9.de0d	Gi2.556	567	Vi2.3	PTA
		0050.56a9.f961	VLAN:3556		UP	

```

R7#show interfaces gigabitEthernet 2 | include bia
  Hardware is CSR vNIC, address is 0050.56a9.ea77 (bia 0050.56a9.ea77)

```

On CSR7, we can see the new static translation. Tracing the routing logic, incoming IPv4 packets to 13.144.2.1 are sent to the NVI. Outgoing IPv6 packets match the /97 route via Dialer7 towards CSR10. We verified CSR10's route to the Internet earlier, as well as its local WKP route to the NVI for incoming IPv6 packets.

```

R7#show nat64 translations
Proto Original IPv4      Translated IPv4
      Translated IPv6    Original IPv6
-----
---      ---          ---
      13.144.2.1        64:ff9b::d90:201

```

```
Total number of translations: 1
```

```

R7#show ip cef 13.144.2.1
13.144.2.1/32
  nexthop 100.0.0.1 NVI0

```

```

R7#show ipv6 cef 64:ff9b::d90:201
64:FF9B::/97
  attached to Dialer7

```

Just like CSR6, this works without issue. CSR7 translates private IPv4 into transitory IPv6, then CSR10 translates transitory IPv6 into public IPv4. I highlight the private and public IPv4 source addresses, which

is the primary use-case of NAT464 for LSN in general. I don't verify all the statistics again since this logic is identical to CSR6 except uses the WKP.

```
R7#show nat64 translations
```

Proto	Original IPv4	Translated IPv4
	Translated IPv6	Original IPv6
---	---	---
	13.144.2.1	64:ff9b::d90:201
icmp	192.168.7.14:45212	[7007:feed:bead::c0a8:70e]:45212
	13.144.2.1:45212	[64:ff9b::d90:201]:45212

```
R10#show nat64 translations
```

Proto	Original IPv4	Translated IPv4
	Translated IPv6	Original IPv6
---	---	---
icmp	13.144.2.1:1	[64:ff9b::d90:201]:45212
	209.19.85.10:1	[7007:feed:bead::c0a8:70e]:45212

Rather than enumerate the remaining Internet mappings on CSR7 as we did on CSR6, we will use TCP/UDP specific mappings for variety. Starting with TCP, we create a mapping for telnet. This will give XRv4 telnet access to XRv3, but nothing else. Because I use port 23 for both the IPv4 and IPv6 components, the number is effectively passed through. This allows CSR10 to maintain the destination port 23 when it performs NAT64-overload. I use IPv4 address 13.144.2.9 on XRv3 for variety.

```
! CSR7
```

```
nat64 v6v4 static tcp 64:FF9B::D90:209 23 13.144.2.9 23
```

```
R7#show nat64 translations protocol tcp
```

Proto	Original IPv4	Translated IPv4
	Translated IPv6	Original IPv6
---	---	---
tcp	13.144.2.9:23	[64:ff9b::d90:209]:23

When the telnet flow begins, CSR7 creates a new state entry for the specific flow, which includes the source IPv4 address and source port. The source port is carried over exactly (cannot be modified), but the destination port is treated the same due to our configuration. The flow appears to be coming from CSR7 towards the WKP destination shown below.

```
R7#show nat64 translations protocol tcp
```

Proto	Original IPv4	Translated IPv4
	Translated IPv6	Original IPv6
---	---	---
tcp	13.144.2.9:23	[64:ff9b::d90:209]:23

```

tcp      192.168.7.14:29960    [7007:feed:bead::c0a8:70e]:29960
        13.144.2.9:23    [64:ff9b::d90:209]:23

```

This WKP destination is routed to CS10's NVI. The IPv4 address is decoded from the packet and the TCP port number is maintained. Due to overloading, the original source port is rewritten to some demultiplexing value that CSR10 selects; it is 1015 in this case. The final IPv4 packet is highlighted. XRv3 correctly acknowledges this as a telnet session from 209.19.85.10 using port 1025 as shown in its TCP table.

```

R10#show nat64 translations protocol tcp
Proto  Original IPv4          Translated IPv4
       Translated IPv6    Original IPv6
-----
tcp    13.144.2.9:23          [64:ff9b::d90:209]:23
       209.19.85.10:1025 [7007:feed:bead::c0a8:70e]:29960

```

```

RP/0/0/CPU0:XRv3#show tcp brief | include :23
0x1014721c 0x60000000 0 0 13.144.2.9:23 209.19.85.10:1025 ESTAB

```

After clearing the NAT64 statistics on CSR7, I sent exactly 8 packets from XRv4 to XRv3. It fails since CSR7 has no matching translation entry for this protocol. This is the expected behavior of using port/protocol specific static mappings.

```

RP/0/0/CPU0:XRv4#ping vrf 7 13.144.2.9 count 8 timeout 1
Type escape sequence to abort.
Sending 8, 100-byte ICMP Echos to 13.144.2.9, timeout is 1 seconds:
.....
Success rate is 0 percent (0/8)

```

```

R7#show nat64 statistics interface gig2.574
NAT64 Statistics
Interface Statistics
  GigabitEthernet2.574 (IPv4 configured, IPv6 configured):
    Packets translated (IPv4 -> IPv6)
      Stateless: 0
      Stateful: 0
      MAP-T: 0
    Packets translated (IPv6 -> IPv4)
      Stateless: 0
      Stateful: 0
      MAP-T: 0
    Packets dropped: 8

```

Next, we will perform the same test with UDP. Using IP SLA, we will create a probe using port 42518 towards 13.144.2.17. IP SLA might be useful to test a custom application across NAT64 devices to an Internet destination. The IP SLA configuration on XRv3 and XRv4 is very basic; the details are described

in the IP SLA section. XRv3 is configured to respond to 13.144.2.17:42518 and XRv4 is configured to send traffic to that address/port pair.

```
! XRv3
ipsla
 responder
  type udp ipv4 address 13.144.2.17 port 42518

! XRv4
ipsla
 operation 7
  type udp echo
  vrf 7
  destination address 13.144.2.17
  control disable
  timeout 3000
  destination port 42518
  frequency 5

schedule operation 7
 start-time now
 life forever
```

On CSR7, we use the same logic as we did for TCP. The port number 42518 is maintained across the NAT64 process; otherwise, the configuration is identical.

```
! CSR7
nat64 v6v4 static udp 64:FF9B::D90:211 42518 13.144.2.17 42518
```

Checking the translations on CSR7 and CSR10, we can see the IP SLA probe. XRv4 is using source port 35760 which is transparent across the static NAT64 mapping. The destination port is also transparent due to the configuration. CSR10 adjusts the source port since it is overloading a single IP address, but otherwise the logic is similar. Port 42518 is ultimately revealed to XRv3 who can respond correctly.

```
R7#show nat64 translations protocol udp
Proto  Original IPv4      Translated IPv4
      Translated IPv6  Original IPv6
-----
udp    ---
      13.144.2.17:42518  [64:ff9b::d90:211]:42518
udp    192.168.7.14:35760  [7007:feed:bead::c0a8:70e]:35760
      13.144.2.17:42518  [64:ff9b::d90:211]:42518

R10#show nat64 translations protocol udp
Proto  Original IPv4      Translated IPv4
      Translated IPv6  Original IPv6
-----
```

```
udp      13.144.2.17:42518      [64:ff9b::d90:211]:42518
        209.19.85.10:1024      [7007:feed:bead::c0a8:70e]:35760
```

To ensure it is working, we check the IP SLA statistics on XRv4. The probe is succeeding and there is bidirectional reachability for this new “custom application”.

```
RP/0/0/CPU0:XRv4#show ipsla statistics 7 | utility egrep 'Number|return'
Number of operations attempted: 74
Number of operations skipped   : 0
Latest operation return code   : OK
```

Last, I demonstrate changes the destination port. This can mask the user from the actual ports used by the target. For example, to access 13.144.2.25 via telnet, the user (XRv4) must use port 2300 rather than 23.

```
! CSR7
nat64 v6v4 static tcp 64:FF9B::D90:219 23 13.144.2.25 2300
```

```
R7#show nat64 translations v6 translated 13.144.2.25
```

Proto	Original IPv4	Translated IPv4
	Translated IPv6	Original IPv6

tcp	---	---
	13.144.2.25:2300	[64:ff9b::d90:219]:23

The logic is identical to the original telnet test, except XRv4 specifies port 2300 now. CSR7 translates this to destination port 23; CSR10 remains unaware of any changes.

```
R7#show nat64 translations v6 translated 13.144.2.25
```

Proto	Original IPv4	Translated IPv4
	Translated IPv6	Original IPv6

tcp	---	---
	13.144.2.25:2300	[64:ff9b::d90:219]:23
tcp	192.168.7.14:23444	[7007:feed:bead::c0a8:70e]:23444
	13.144.2.25:2300	[64:ff9b::d90:219]:23

Additional Reading – Reference configurations "Isn-464"

36.3 Dual stack lite (DS-lite)

DS-Lite takes the advantages of NAT464 and simplifies the deployment concept. Because NAT64 may not be supported on low-end CPE routers in residential deployments, DS-Lite is an alternative that allows IPv4 to be encapsulated inside IPv6. Just like the NAT464 model, the CPE-to-LSN links are still IPv6, but the redundancy, performance, and scaling concerns are eliminated since translation is no

longer occurring at the CPE. IPv4 traffic tunneled to the LSN inside IPv6 is decapsulated, and then regular NAT44 is performed.

When coupled with NAT44, however, DS-Lite adds a key piece of information. Because NAT44 was not performed on the CPEs, it is possible that the same private addressing could be used by multiple CPEs beneath a single LSN. With NAT444, this is not a concern since the SP-allocated private addressing, which is the outside address for the CPE, would be unique at each CPE. With DS-Lite, no such concept exists. The NAT44 process, when interacting with DS-Lite, adds the source IPv6 address of the DS-Lite tunnel to the NAT state. This mapping provides a destination address for return traffic entering the LSN from the outside, as well as differentiates CPE private addressing. Like with NAT444 and NAT464, native IPv6 traffic is routed normally with no additional encapsulation or NAT applied.

This feature is not currently supported on IOS, IOS-XE, or XRv. It is supported on IOS-XR platforms but only with specific hardware enhancements. Specifically, The ASR9000 series Integrated Service Module (ISM) for CGN and the CRS Carrier Grade Services Engine (CGSE). A sample configuration from Cisco is shown below, though it cannot be demonstrated on XRv.

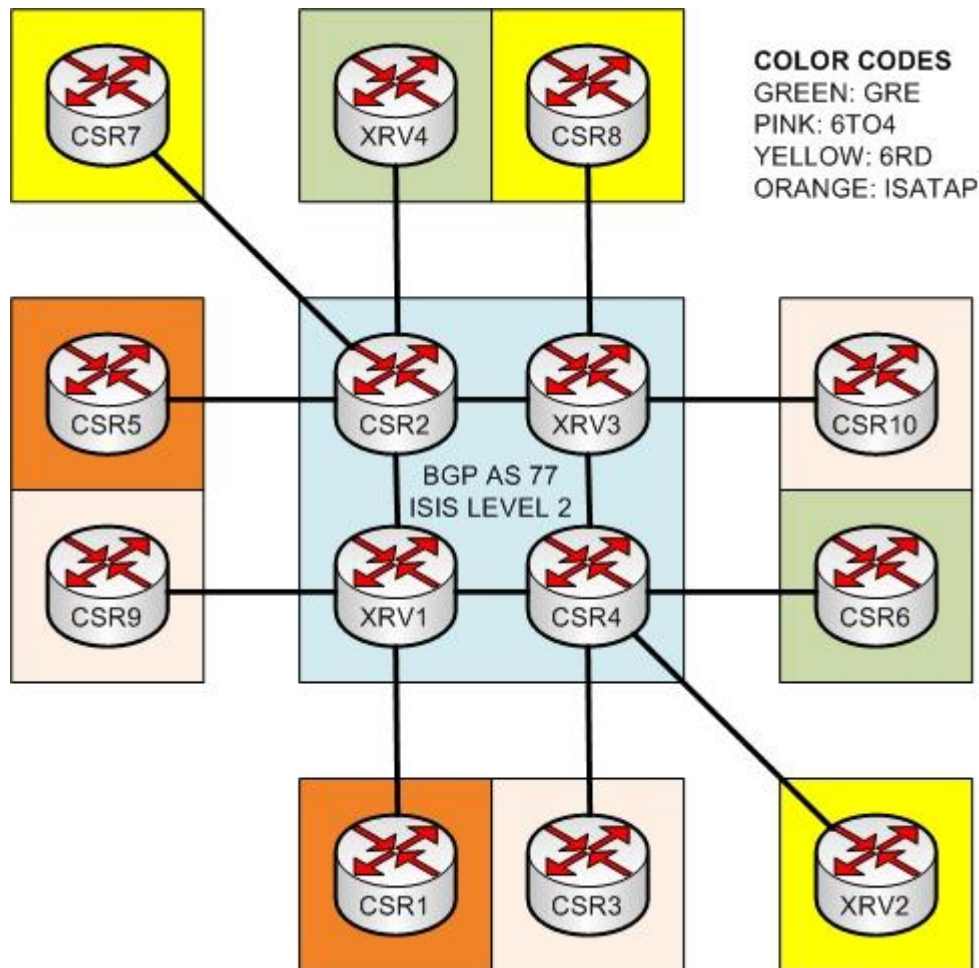
```
interface te0/0/0/0
  ipv6 address 2001:db8:ff00::1/64
!
interface te0/1/0/0
  ipv4 address 192.168.100.1/24
!
interface ServiceApp61
  ipv6 address 2001:db8:1::1/64
  service cgn demo service-type ds-lite
!
interface ServiceApp41
  ipv4 address 192.168.1.1 255.255.255.252
  service cgn demo service-type ds-lite
!
service cgn demo
  service-type ds-lite dslite-1
  map address-pool x.y.z.0/24
  after-tunnel-endpoint-address 2001:db8:ffff::1
!
address-family ipv4
  interface ServiceApp42
!
address-family ipv6
  interface ServiceApp41
!
!
router static
  address-family ipv4 unicast
    x.y.z.0/24 ServiceApp42
```


!

```
address-family ipv6 unicast
  2001:db8:ffff::1/128 ServiceApp41
```

36.4 IPv6 tunneling over IPv4 networks

There are many techniques for connecting isolated IPv6 networks over IPv4 transport. Several methods are tested below, and they all share a common network diagram. Each subsection is independent from the others but all rely on the same MPLS network providing L3VPN service for IPv4 only. The MPLS configurations are very basic with XRv1 as a VPNv4 RR and all LSRs running LDP. The PE-CE routing protocol is nonexistent as, like the LISP lab, the PEs just redistribute the connected transit link into BGP and the CEs have a static default IPv4 route pointing towards the PEs. Routers are color-coded in the diagram below to show which tunneling techniques they will use. The focus is on 6RD since the other mechanisms are less applicable to service providers, but are valid IPv6 transition options nonetheless.



Below is a comparison of different IPv6 tunneling techniques. Some techniques are clearly better than others based on the natural progression of time (e.g., 6RD is a newer variant of 6to4), but each of them has valid use cases. Legacy auto-tunnels are not examined here at all. One thing that all tunnels have in common is they require an IPv4 source address, which is omitted from the chart for brevity.

Protocol/Method	Uses	Notes
Manual IPv6-in-IPv4	P2P tunnel, intra or inter site, manually configured	Carries IPv6 only, can run IGP and BGP
GRE	P2P tunnel, intra or inter site, manually configured	Carries IPv4, IPv6, CLNS, etc. but adds extra 4-byte encapsulation, can run IGP and BGP
6to4	P2MP tunnel, inter site, automatic based on IPv4 address	Limited to 2002::/16 prefix, inflexible prefix format, anycast-based relay for Internet access using specific addresses
6RD	P2MP tunnel, inter site, automatic based on IPv4 address	Flexible prefix format, simplified relay mechanism, can use SPs own address space
ISATAP	P2MP tunnel, intra site, automatic based on IPv4 address	Can run IGPs (unicast neighbors) and BGP

Additional Reading - Reference configurations "ipv6-tunnels"

36.4.1 GRE / Manual IPv6 tunnels

The simplest method to bridge disparate IPv6 islands is to create a GRE tunnel between sites and run IPv6 inside of it. This is also supported with DMVPN and any other GRE-based technology. XR also supports this method, but none of the others (at least not without special hardware modules). CSR6 and XRv4 form this basic GRE tunnel between one another. The tunnel endpoints have reachability through the L3VPN via IPv4 using default routes. There are no tunnel IPv4 addresses since the customer sites are IPv6 only.

```
! CSR6
interface GigabitEthernet2.546
 encapsulation dot1Q 3546
 ip address 10.4.6.6 255.255.255.0

ip route 0.0.0.0 0.0.0.0 10.4.6.4

interface Tunnel614
 description BASIC GRE
 ipv6 address FE80::6 link-local
 ipv6 address FD00:10:6:14::6/64
 tunnel source 10.4.6.6
 tunnel destination 10.2.14.14

! XRv4
interface GigabitEthernet0/0/0/0.524
 ipv4 address 10.2.14.14 255.255.255.0
 encapsulation dot1q 3524

router static
 address-family ipv4 unicast
 0.0.0.0/0 10.2.14.2
```

```

interface tunnel-ip614
  description BASIC GRE
  ipv6 address fe80::14 link-local
  ipv6 address fd00:10:6:14::14/64
  tunnel source 10.2.14.14
  tunnel destination 10.4.6.6

```

A quick verification on each router shows that the traffic is carried inside IPv4 between the tunnel source and destination. This is nothing new as we tested this in the GRE section as well.

```

R6#show interfaces tunnel614 | include Tunnel
Tunnel614 is up, line protocol is up
  Hardware is Tunnel
  Tunnel linestate evaluation up
  Tunnel source 10.4.6.6, destination 10.2.14.14
  Tunnel protocol/transport GRE/IP
  Tunnel TTL 255, Fast tunneling enabled
  Tunnel transport MTU 1476 bytes
  Tunnel transmit bandwidth 8000 (kbps)
  Tunnel receive bandwidth 8000 (kbps)

```

```

RP/0/0/CPU0:XRv4#show interfaces tunnel-ip614 | include Tunnel
  Hardware is Tunnel
  Tunnel TOS 0
  Tunnel mode GRE IPv4,
  Tunnel source 10.2.14.14, destination 10.4.6.6
  Tunnel TTL 255

```

There is no complexity to the BGP configuration either. The snippets are shown below. Dynamic routing is one of the benefits of GRE tunneling that is not supported in 6to4 or 6RD, although one could argue that it isn't necessary for those models in the first place.

```

! CSR6
router bgp 614
  no bgp default ipv4-unicast
  neighbor FD00:10:6:14::14 remote-as 614
  address-family ipv6
    network ::10:6:6:6/128
  neighbor FD00:10:6:14::14 activate

! XRv4
router bgp 614
  address-family ipv6 unicast
    network ::10:14:14:14/128

  neighbor fd00:10:6:14::6
    remote-as 614
  address-family ipv6 unicast

```

From CSR6, we can verify the BGP session is up then check to ensure CSR6 learns XRv4's loopback. We also quickly test reachability.

```
R6#show bgp ipv6 unicast summary | begin ^Neighbor
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
FD00:10:6:14::14
              4      14      22      23      3    0    0 00:18:10      1
```

```
R6#show bgp ipv6 unicast | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
*>  ::10:6:6:6/128  ::              0          32768 i
*>i  ::10:14:14:14/128
                                FD00:10:6:14::14
                                0      100      0 i
```

```
R6#ping ::10:14:14:14 source ::10:6:6:6
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to ::10:14:14:14, timeout is 2 seconds:
Packet sent with a source address of ::10:6:6:6
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 5/11/33 ms
```

As a final data-plane check, we capture packets inbound on CSR2 (PE) to see IPv6 packets encapsulated inside GRE. Notice that the GRE header indicates that the tunneled protocol is IPv6 using 0x86DD. The IPv4 and IPv6 source addresses are highlighted in yellow, where the IPv4 comes first and is the outer encapsulation. IPv4 and IPv6 destination addresses are in green and represent CSR6's address. The GRE header is in pink which clearly shows the IPv6 protocol type. Notice that the IPv4 IP protocol is 47 (0x2F, colored cyan) which is regular GRE; there is nothing special for IPv6 in the outer IP header since the GRE header shows the tunneled protocol.

```
R2#show monitor capture CAP buffer dump
0
0000: 005056A9 BE8A0050 56A9862A 81000DC4 .PV....PV..*....
0010: 08004500 007C0000 4000FF2F 53390A02 ..E...|..@../S9..
0020: 0E0E0A04 06060000 86DD6000 0000003C .....`....<
0030: 3A400000 00000000 00000010 00140014 :@.....
0040: 00140000 00000000 00000010 00060006 .....
0050: 00068100 CCAC24C8 00000001 02030405 .....$......
```

Manual tunneling is not examined here because it is very simple and almost identical to GRE. The difference is that the tunnel mode is changed to "ipv6ip" with no additional options like 6to4 or 6RD. This is similar to "ip-in-ip" tunneling, IP protocol 4, which is slightly less overhead than GRE but can only carry IPv4 inside IPv4. IPv6-in-IPv4 is the same concept, so non-IPv6 traffic cannot be tunneled inside. IPv6-in-IPv4 uses IP protocol 41 and is shared by 6to4 and 6RD we will see soon. The downside of these approaches (both GRE and manual) is that they are labor-intensive and difficult to deploy large scale.

36.4.2 6to4 automatic tunnels

6to4 tunnels are a fast way to deploy IPv6 tunnels automatically (and without storing state) throughout the network. Somewhat similar to DMVPN, no tunnel destination is specified. Since the IPv6 addresses are so large and can easily encompass IPv4 addresses, the IPv4 tunnel endpoints are embedded inside of the IPv6 addresses. The 6to4 prefix header always begins with 2002::/16 as this prefix was allocated by IANA specifically for 6to4 tunneling. The next 32 bits represent the local IPv4 address for a given site converted to hexadecimal. If the tunnel source was 10.4.6.6 as tested earlier, the 6to4 prefix for that site would be 2002:0A04:0606::/48, which leaves a 16-bit subnet ID as is normally expected for IPv6. This implies that all IPv6 networks local to that site must be encompassed within 2002:0A04:0606::/48, since any remote nodes sending traffic to this prefix automatically know to encapsulate the IPv6 packet into an IPv4 packet destined for 10.4.6.6. CSR3, CSR9, and CSR10 are used for this test. First, we will configure it the “hard way” on CSR9 and CSR10 which does not use the general-prefix construct. The tunnel mode is changed to IPv6-in-IPv4 (protocol 41) with the special “6to4” option. This means that the tunnel does not need an explicit destination as it is embedded inside the IPv6 prefix. Also notice that we don’t need to configure global addressing on the tunnel itself. CSR9 and CSR10 have LAN simulations on loopback0 which are subnets within the /48 prefix specific to each site; you cannot deviate from this, but you do have 65536 subnets of size /64 to allocate, which is a good solution in my opinion.

```
! CSR9
interface Tunnel3910
  description 6TO4
  ipv6 address FE80::9 link-local
  tunnel source 10.9.11.9
  tunnel mode ipv6ip 6to4

interface Loopback0
  ipv6 address 2002:A09:B09:9998::9/128
  ipv6 address 2002:A09:B09:9999::9/128

ipv6 route 2002::/16 Tunnel3910

! CSR10
interface Tunnel3910
  description 6TO4
  ipv6 address FE80::10 link-local
  tunnel source 10.10.13.10
  tunnel mode ipv6ip 6to4

interface Loopback0
  ipv6 address 2002:A0A:D0A:AAAA::A/128
  ipv6 address 2002:A0A:D0A:AAAB::A/128

ipv6 route 2002::/16 Tunnel3910
```

Since CSR9 and CSR10 have almost identical configurations, we limit our verification to CSR9 for brevity. Despite not having a destination, the tunnel comes up. This is a result of 6to4 being enabled which is a multipoint tunnel, similar to DMVPN. The source address is the PE-CE link address which has connectivity with MPLS L3VPN. Notice that MTU is 1480 bytes as opposed to the more commonly seen 1476 with GRE tunnels. Since the IPv6 packet is wrapped inside of a standard 20-byte IPv4 header with no 4-byte GRE encapsulation, the MTU is automatically optimized to the maximum possible value (1500 – 20).

```
R9#show interfaces tunnel3910 | include Tunnel
Tunnel3910 is up, line protocol is up
  Hardware is Tunnel
  Tunnel linestate evaluation up
  Tunnel source 10.9.11.9
  Tunnel protocol/transport IPv6 6to4
  Tunnel TTL 255
  Tunnel transport MTU 1480 bytes
  Tunnel transmit bandwidth 8000 (kbps)
  Tunnel receive bandwidth 8000 (kbps)
```

If we want to send traffic to CSR10, we follow the highly generic 2002::/16 route which directs traffic out of the tunnel. Looking at the IPv6 CEF table, we can see an interesting action in the output chain. After identifying Tunnel3910 as the outgoing interface, another lookup is performed in the IPv4 table. This lookup is based on the embedded IPv4 address for CSR10's tunnel so that the router knows how to ultimately encapsulate the packet inside IPv4, then add the appropriate layer 2 encapsulation.

```
R9#show ipv6 route 2002::/16
Routing entry for 2002::/16
  Known via "static", distance 1, metric 0
  Route count is 1/1, share count 0
  Routing paths:
    directly connected via Tunnel3910
    Last updated 02:05:41 ago

R9#show ipv6 cef 2002::/16 internal | begin output
output chain:
  IPV6 midchain out of Tunnel3910, addr :: 7F3DE5349238
  Lookup in table IPv4:Default
```

Additional details in the adjacency table reveal this as well. We can also confirm the IP protocol number of 41 this way. Notice that the destination IPv4 address in the encapsulation string is all zeroes and the comment mentions that it must “copy from payload”, which is accurate (yellow) The tunnel source IPv4 address is shown just before it and is colored green.

```
R9#show adjacency tunnel3910 :: encapsulation
Protocol Interface Address
```

```

IPV6      Tunnel3910      ::(7)
  Encap length 20
  4500000000000000FF29A6C30A090B09
  00000000
  Provider: TUNNEL
  Protocol header count in macstring: 1
  HDR 0: ipv4
    dst: per packet, copy from payload: ipv6 dst (2-5)
    src: static, 10.9.11.9
    prot: static, 41
    ttl: static, 255
    df: static, cleared
  per packet fields: dst tos ident tl chksum

```

As a safety mechanism, it is good practice (but technically not required for connectivity) to ensure that traffic assigned to a site is not allowed to be sent from that site into the 6to4 network. For example, CSR9 owns the aggregate prefix 2002:A09:B09::/48. If a host behind CSR9 sends traffic to a subnet of this prefix for which CSR9 does not have a longer match, it will try to send it of the 6to4 tunnel. Although the packet won't actually go anywhere since the destination is local, it is sloppy and could introduce a control-plane DoS attack.

```

R9#show ipv6 cef 2002:A09:B09:1:2:3:4:5
2002::/16
  attached to Tunnel3910

```

We can correct this easily with null routes on CSR9 and CSR10. It is still sufficiently generic enough such that longer matches won't get blackholed, but specific enough to override the 2002::/16 for local traffic.

```

! CSR9
ipv6 route 2002:A09:B09::/48 null0

```

```

! CSR10
ipv6 route 2002:A0A:D0A::/48 null0

```

```

R9#show ipv6 cef 2002:A09:B09:1:2:3:4:5
2002:A09:B09::/48
  attached to Null0

```

```

R10#show ipv6 cef 2002:A0A:D0A:1:2:3:4:5
2002:A0A:D0A::/48
  attached to Null0

```

We can test connectivity across the 6to4 tunnel while using EPC outbound on CSR9 at the same time. We will initiate the ping from CSR10 so that we capture CSR9's echo-replies. I also highlight the IPv4 addresses embedded in the IPv6 prefix for clarity; it shows packets going from 10.10.13.10 to 10.9.11.9, which is correct.

```
R10#ping 2002:A09:B09:9998::9 source 2002:A0A:D0A:AAAA::A
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2002:A09:B09:9998::9, timeout is 2 seconds:
Packet sent with a source address of 2002:A0A:D0A:AAAA::A
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/8/22 ms
```

EPC dumps on CSR9 show the entire packet. Unlike the GRE tunnel we saw earlier, the IP protocol is now 41 (0x29) which indicates IPv6-in-IPv4 encapsulation, shown in cyan. As a result, there is no GRE header, which saves 4 bytes of encapsulation (but makes IPv6 the only possible payload protocol). The IPv4 and IPv6 source addresses are shown in yellow again, which is nothing different than before. Likewise, the IPv4 and IPv6 destination addresses are shown in green, which is also similar to the GRE example. In summary, IPv6 traffic flows from CSR9 to CSR10 inside IPv4.

```
R9#show monitor capture CAP buffer dump
0
0000: 005056A9 9C600050 56A9D672 81000E07 .PV..`.PV..r....
0010: 08004500 00780014 0000FF29 8F230A09 ..E..x.....).#..
0020: 0B090A0A 0D0A6000 0000003C 3A402002 .....`....<:@ .
0030: 0A090B09 99980000 00000000 00092002 .....
0040: 0A0A0D0A AAAA0000 00000000 000A8100 .....
0050: 26C01AA2 00000001 02030405 06070809 &.....
```

Next, we will bring CSR3 into the 6to4 network. We will use the general-prefix construct to simplify the configuration of LAN networks at the CSR3 site. The special “6to4” option is used with the general-prefix and the tunnel source is specified. This auto-computes the 6to4 addressing so you don’t have to do it manually. Notice the odd-looking addresses under loopback0; when referencing a general-prefix, the “::” identifier means that the starting bits of the prefix are copied from the general-prefix. In this case, we specify the last 80 bits of the address since the first 48 come from the general-prefix.

```
! CSR3
ipv6 general-prefix GP_6TO4 6to4 GigabitEthernet2.534

interface Tunnel3910
description 6TO4
ipv6 address FE80::3 link-local
tunnel source GigabitEthernet2.534
tunnel mode ipv6ip 6to4

interface Loopback0
ipv6 address GP_6TO4 ::3333:0:0:0:3/128
ipv6 address GP_6TO4 ::3334:0:0:0:3/128
```

We can use the general-prefix information to help us configure a null route so that traffic for CSR3 is not sent over the tunnel, as we did on CSR9 and CSR10 earlier. It makes sense to copy/paste the general-prefix when building the static route to avoid making a mistake.


```

R3#show ipv6 general-prefix
IPv6 Prefix GP_6TO4, acquired via 6to4
  2002:A03:403::/48 Valid lifetime infinite, preferred lifetime infinite
    Loopback0 (Address command)
    Loopback0 (Address command)

! CSR3
ipv6 route 2002:A03:403::/48 null0

```

6to4 scales very well because there is nothing special we have to do on the existing 6to4 sites to communicate to CSR3 now. As long as they can reach CSR3's tunnel source (IPv4 address), the sites will communicate. A quick test from CSR3 shows that there is reachability to both CSR9 and CSR10.

```

R3#ping 2002:A09:B09:9999::9 source 2002:A03:403:3333::3
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2002:A09:B09:9999::9, timeout is 2 seconds:
Packet sent with a source address of 2002:A03:403:3333::3
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/7/22 ms

```

```

R3#ping 2002:A0A:D0A:AAAA::A source 2002:A03:403:3334::3
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2002:A0A:D0A:AAAA::A, timeout is 2 seconds:
Packet sent with a source address of 2002:A03:403:3334::3
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/7/22 ms

```

6to4 suffers from some inflexibilities, though. The prefix always begins with 2002::/16, which may not fit nicely into the addressing scheme developed by an organization or service provider. Furthermore, all 32 bits of the IPv4 address must be included in the IPv6 prefix, which can be wasteful if all IPv4 tunnel IPs share some common octets. The subnet ID is always 16 bits as well, so this is a hard upper-bound on the number of usable subnets in an organization. There is also an unpredictability associated with deploying it since the IPv4 relay address of 192.88.99.0/24 is commonly used for Internet access. This is an anycast address which could be anywhere at any time, and poor routing decisions made in the SP network could affect customer traffic negatively. 6RD improves upon these shortfalls as seen next. Neither 6to4 nor 6RD was really designed to support dynamic IGPs, which are supported on manual, GRE, and ISATAP tunnels (semi-dynamic on ISATAP).

36.4.3 6 Rapid Deployment (6RD)

6RD operates very similarly to 6to4 but overcomes the inflexibilities and relay issues inherent to 6to4 designs. The 6RD prefix format is an important concept since it is very flexible and warrants some explanation. The entire IPv6 prefix still must be 64 bits long, but is the concatenation of the IPv6 prefix, IPv4 address, and subnet ID. These are all variable length fields, and as long as the bit-wise sum of their prefix lengths is 64, 6RD can support it. The individual components are explained below:

1. **IPv6 prefix:** This was fixed at 2002::/16 with 6to4. A provider may allocate a different block of addresses for a specific customer, say 200B:CC00::/24. This allows providers to intelligently carve their IPv6 prefixes for customers.
2. **IPv4 address:** 6to4 used all 32 bits of the IPv4 tunnel IP inside the IPv6 address. This makes sense in the case that the IPv4 addresses are totally different. Inside of a private WAN controlled by a provider, these addresses can be prescribed, and possibly assigned sequentially. This often means that the left-most bits in the IP address won't change. For example, if three customer sites have IP addresses 10.20.30.1, 10.20.30.2, and 10.20.30.3, at least the first 24 bits are all identical. There is no reason to waste 24 bits of the IPv6 address to carry this information if 6RD is aware of the similarities. As such, only differing bits (that is, $32 - 24 = 8$) can be carried in the 6RD prefix.
3. **Subnet ID:** When using 6to4, there were 16-bits remaining for the subnet ID. This can be larger if more subnets are needed for a site, but at the cost of having a less specific prefix and/or less IPv4 address bits represented. This is the third contributor to the 64-bit limit for the 6RD prefix and is not configured explicitly since it is a function of the difference: $64 - \text{IPv6 prefix} - \text{IPv4 address} = \text{subnet ID range}$. Evaluating the example above, a 24 bit prefix plus an 8 bit IPv4 address equals 32, which leaves 32 bits available for subnet IDs. That significantly increases the number of subnets available for a given site.

The actual prefixes for the example above are shown below. The IPv6 prefix of 200B:CC00::/24 is combined with the last 8 bits of the IPv4 address (tunnel source) to create a unique prefix per site.

1. Site 1 (10.20.30.1): 200B:CC01::/32
2. Site 2 (10.20.30.2): 200B:CC02::/32
3. Site 3 (10.20.30.3): 200B:CC03::/32

As an adjustment, let's assume the IPv4 addresses are now required to change in the third octet. This means only the first 16 bits of the IPv4 address are consistent between all sites in the 6RD mesh, so we must now include the 8-bit third octet inside the 6RD prefix. This means there are 8 bits less for subnet IDs, which results in 24 bits available for subnet IDs, which is still better than 16 with 6to4. I am keeping the math simple by adjusting at convenient bit-boundaries, but you don't have to.

1. Site 1 (10.20.185.1): 200B:CCB9:0100::/40
2. Site 2 (10.20.30.2): 200B:CC1E:0200::/40
3. Site 3 (10.20.99.3): 200B:CC63:0300::/40

Next, we will configure 6RD on CSR7 and CSR8 to start. We will introduce CSR4 (PE) and XRv2 (Internet) later. CSR7 uses the general-prefix method while CSR8 configures the IPv6 addressing manually. Like 6to4, we can build a general-prefix based on the 6RD prefix which is auto-calculated. Since all PE-CE links begin with 10.x.x.x, we assume only the first octet is common between all sites and specify this as the "ipv4 prefix-len". 6RD is smart enough to subtract this number from 32 and encoded the remaining 24

bits of the IPv4 address inside the final 6RD prefix. We manually specify the IPv6 prefix at whatever length we want; since this is a small 6RD deployment, a /32 has been allocated instead of something typically larger. This means that there are 8 bits left for the subnet ID ($32 + 24 = 56$), which means there can only be 256 subnets per site. Both CSR7 and CSR8 configure some loopback addresses inside their 6RD prefix, much like 6to4. Notice that the subnet IDs are only 8 bits in this setup. Last, we configure a static route which directs traffic for this general /32 prefix out of the tunnel, much like we did for 2002::/16 with 6to4. We also ensure that traffic local to the site (we know this based on the constructed prefix) does not get sent over the tunnel using a null route.

```
! CSR7
ipv6 general-prefix GP_6RD 6rd Tunnel7812

interface Tunnel7812
  description 6RD
  ipv6 address GP_6RD ::7/128
  tunnel source 10.2.7.7
  tunnel mode ipv6ip 6rd
  tunnel 6rd ipv4 prefix-len 8
  tunnel 6rd prefix 2003:1E84::/32

interface Loopback0
  ipv6 address GP_6RD ::77:0:0:0:7/128
  ipv6 address GP_6RD ::77:0:0:0:77/128

ipv6 route 2003:1E84:207:700::/56 Null0
ipv6 route 2003:1E84::/32 Tunnel7812

! CSR8
interface Tunnel7812
  description 6RD
  ipv6 address 2003:1E84:80D:800::8/128
  tunnel source 10.8.13.8
  tunnel mode ipv6ip 6rd
  tunnel 6rd ipv4 prefix-len 8
  tunnel 6rd prefix 2003:1E84::/32

interface Loopback0
  ipv6 address 2003:1E84:80D:888::8/128
  ipv6 address 2003:1E84:80D:888::88/128

ipv6 route 2003:1E84:80D:800::/56 Null0
ipv6 route 2003:1E84::/32 Tunnel7812
```

We can verify the tunnels using a custom show command. This shows the important tunnel characteristics including IPv4 source and prefix length (only 10.0.0.0/8 is common), along with the configure IPv6 prefix. The “general prefix” at the bottom shows the post-calculation 6RD prefix for the site. The first 32 bits are fixed as the specified IPv6 prefix, the next 24 bits represents the last 24 bits of

the IPv4 address per site, and the remaining 8 bits are available for the subnet ID. Even though we didn't configure a general-prefix on CSR8, the calculation still gets done, and this show command can be useful for verifying the static null routes.

```
R7#show tunnel 6rd
Interface Tunnel7812:
  Tunnel Source: 10.2.7.7
  6RD: Operational, V6 Prefix: 2003:1E84::/32
    V4 Prefix, Length: 8, Value: 10.0.0.0
    V4 Suffix, Length: 0, Value: 0.0.0.0
  General Prefix: 2003:1E84:207:700::/56
```

```
R8#show tunnel 6rd
Interface Tunnel7812:
  Tunnel Source: 10.8.13.8
  6RD: Operational, V6 Prefix: 2003:1E84::/32
    V4 Prefix, Length: 8, Value: 10.0.0.0
    V4 Suffix, Length: 0, Value: 0.0.0.0
  General Prefix: 2003:1E84:80D:800::/56
```

Like 6to4, we will verify the routing next. The route is static via the 6RD tunnel and the CEF table says an IPv4 lookup must be performed to build the encapsulation for these packets.

```
R8#show ipv6 route 2003:1E84:207:700::/56
Routing entry for 2003:1E84::/32
  Known via "static", distance 1, metric 0
  Route count is 1/1, share count 0
  Routing paths:
    directly connected via Tunnel7812
    Last updated 01:50:13 ago

R8#show ipv6 cef 2003:1E84:207:700:: internal | begin output
output chain:
  IPV6 midchain out of Tunnel7812, addr :: 7F069FA84EC8
  Lookup in table IPv4:Default
```

The encapsulation details are similar to 6to4 with some extra information. The destination IPv4 address is still gleaned from the tunneled IPv6 address, but we can see some extra fields to describe how to handle this process. It copies 24 bits from right to left (start at the 32nd bit and copy 24 of them) which essentially grabs the 24 least significant bits. The IP protocol is still 41, like a manual or 6to4 tunnel.

```
R8#show adjacency tunnel7812 :: encapsulation
Protocol Interface Address
IPV6 Tunnel7812 ::(7)
  Encap length 20
  4500000000000000FF29A4C50A080D08
  00000000
```

```

Provider: TUNNEL
Protocol header count in macstring: 1
HDR 0: ipv4
    dst: static and per packet, copy from payload:
        addr msb 10.0.0.0, addr lsb 0.0.0.0
        num msb 8, num lsb 0, start 32, len 24
    src: static, 10.8.13.8
    prot: static, 41
    ttl: static, 255
    df: static, cleared
    per packet fields: dst tos ident tl chksum

```

Next, we will capture packets on CSR7 outbound and initiates pings on CSR8. This will capture CSR7's echo-replies so we can see the encapsulation. It looks very similar to 6to4; protocol 41 (cyan) with IPv4 source/destination on the outside, and the appropriate IPv6 addressing on the inside. Yellow and green represent source and destination addresses, respectively, as usual.

```

R7#show monitor capture CAP buffer dump
0
0000: 005056A9 BE8A0050 56A9EA77 8100DC7 .PV....PV..w....
0010: 08004500 00780028 0000FF29 931C0A02 ..E..x.(...)....
0020: 07070A08 0D086000 0000003C 3A402003 .....`....<:@ .
0030: 1E840207 07770000 00000000 00072003 .....w..... .
0040: 1E84080D 08880000 00000000 00888100 .....
0050: 4814121E 00000001 02030405 06070809 H.....

```

Last, we will bring CSR4 into the 6RD network. This is loosely analogous to a LISP PxTR since it will provide interworking between 6RD and non-6RD sites. It is known as a border router (BR, sometimes called border relay) and requires no special configuration. 6RD does work inside of a VRF, so we must configure the tunnel VRF (for endpoint reachability) and the VRF on the tunnel itself (for passing traffic to the Internet). The VRF configuration is not specific to its role as a BR, but I wanted to test the VRF capabilities of 6RD. CSR4 doesn't have any loopbacks since it's just a transit router for the 6RD clients. CSR4 uses the general-prefix for simplicity and configures the static routes as required. The difference is that the static default route points towards XRv2 for Internet access.

```

! CSR4
ipv6 general-prefix GP_6RD 6rd Tunnel17812

interface Tunnel17812
    description 6RD
    vrf forwarding 6RD
    ipv6 address GP_6RD ::4/128
    tunnel source 10.4.12.4
    tunnel mode ipv6ip 6rd
    tunnel vrf 6RD
    tunnel 6rd ipv4 prefix-len 8

```

```
tunnel 6rd prefix 2003:1E84::/32

ipv6 route vrf 6RD 2003:1E84:40C:400::/56 Null0
ipv6 route vrf 6RD 2003:1E84::/32 Tunnel7812
ipv6 route vrf 6RD ::/0 GigabitEthernet2.542 FE80::12
```

The 6RD clients need to identify the BR by IPv4 address and configure static default routes towards the IPv6 address of the BR, which gives them Internet connectivity. The IPv4 address is revealed clearly with the 6RD show command in green. We can glean the IPv6 address of the BR by looking at its 6RD general-prefix and adding the host address to it. We could also check CSR4's IPv6 interface assignments to be sure. These are shown in yellow.

```
R4#show tunnel 6rd
Interface Tunnel7812:
  Tunnel Source: 10.4.12.4
  6RD: Operational, V6 Prefix: 2003:1E84::/32
      V4 Prefix, Length: 8, Value: 10.0.0.0
      V4 Suffix, Length: 0, Value: 0.0.0.0
  General Prefix: 2003:1E84:40C:400::/56
```

```
R4#show ipv6 interface brief tunnel7812
Tunnel7812                [up/up]
  FE80::A04:C04
  2003:1E84:40C:400::4
```

```
! CSR7 and CSR8 (all 6RD clients)
interface Tunnel7812
  tunnel 6rd br 10.4.12.4

ipv6 route ::/0 2003:1E84:40C:400::4
```

For brevity we will only verify CSR8 since CSR7 has an identical configuration. After adding the BR configuration, we can verify it with the 6RD show command. The BR is a fancy way of building a static P2P tunnel from a 6RD client to the BR; specifying the remote IPv4 address is effectively specifying a tunnel destination. This is loosely analogous to a static NHRP mapping on a spoke towards an NHS. We can confirm this by checking the encapsulation details for CSR4's IPv6 address in the 6RD tunnel on CSR8. The destination is static which does not require an IPv4 FIB lookup and the encapsulation string is complete already. All of the other 6to4 and 6RD encapsulation strings needed to fill this in dynamically based on the destination IPv6 packet.

```
R8#show tunnel 6rd
Interface Tunnel7812:
  Tunnel Source: 10.8.13.8
  6RD: Operational, V6 Prefix: 2003:1E84::/32
      V4 Prefix, Length: 8, Value: 10.0.0.0
      V4 Suffix, Length: 0, Value: 0.0.0.0
```

Border Relay address: 10.4.12.4

General Prefix: 2003:1E84:80D:800::/56

```
R8#show adjacency tunnel7812 2003:1E84:40C:400::4 encapsulation
Protocol Interface Address
IPV6 Tunnel7812 2003:1E84:40C:400::4 (11)
Encap length 20
4500000000000000FF298EBD0A080D08
0A040C04
Provider: TUNNEL
Protocol header count in macstring: 1
HDR 0: ipv4
dst: static, 10.4.12.4
src: static, 10.8.13.8
prot: static, 41
ttl: static, 255
df: static, cleared
per packet fields: tos ident tl chksum
```

We quickly verify Internet reachability (test loopbacks added to XRv2 are not shown here) from both CSR7 and CSR8. Notice that the prefix starting with 2323 is not part of the 6RD network at all, but the 6RD clients have reachability by following their IPv6 default routes towards the BR.

```
R7#ping 2323::1212 source 2003:1E84:207:777::7
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2323::1212, timeout is 2 seconds:
Packet sent with a source address of 2003:1E84:207:777::7
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/8/22 ms
```

```
R8#ping 2323::1212 source 2003:1E84:80d:888:0:0:0:88
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2323::1212, timeout is 2 seconds:
Packet sent with a source address of 2003:1E84:80D:888::88
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 3/7/20 ms
```

Another interesting concept of 6D is specifying a common suffix. This can be configured in lieu of or in addition to the common prefix for IPv4. For example, in our case, every site's IPv4 address begins with 10.x.x.x. If the sites all began with 10.x.x.x and ended with .1, then all of them would fit the format of 10.x.x.1. In this case, we can specify that the first 8 bits and last 8 bits are common, which means 6RD only needs to carry the middle 16 bits. This is an additional size-optimization for when sites have very common configurations; for example, all 6RD tunnel endpoints could have host address .1 and begin with the same first two octets, and the third octet changes per customer. In this case, only the third octet (8 bits) would need to be carried in the 6RD prefix. The concept is identical to the prefix concept

except starts counting from the least-significant bit rather than the most-significant bit, so it is not tested here.

36.4.4 Intra-Site Automatic tunnel Addressing Protocol (ISATAP)

ISATAP tunnels are, by the virtue of their name, designed for connectivity between nodes within a given site. In this example, we use for it WAN tunneling alongside GRE, 6to4, and 6RD tunnels. This is possible but generally not the use-case; ideally, a router would hand out IPv6 prefixes and clients could auto-configure addresses on their ISATAP tunnels to reach one another in a site. The concepts are similar to 6to4 in that the IPv4 address is embedded within the IPv6 address, except this time it is not part of the prefix. An alternative EUI-64 mechanism is used on ISATAP tunnels to develop the IPv6 address and it relies on the tunnel source rather than the Ethernet MAC. CSR1 and CSR5 run ISATAP with one another and glean their IPv6 LL addresses via EUI-64 (by default). There is nothing fancy about the configuration so far.

```
! CSR1
interface Tunnel15
  description ISATAP
  ipv6 enable
  tunnel source 10.1.11.1
  tunnel mode ipv6ip isatap

! CSR5
interface Tunnel15
  description ISATAP
  ipv6 enable
  tunnel source 10.2.5.5
  tunnel mode ipv6ip isatap
```

When we check the IPv6 LL addresses on the tunnel, we can see that only the least significant 32-bits were modified. ISATAP uses the same decimal-to-hexadecimal conversion method as 6to4 by taking the full IPv4 address and embedded it in the IPv6 address without trying to optimize bit-boundaries like 6RD. The 32 bits preceding this IPv4 address are always 0000:5EFE, which is the buffer similar to the FFFE inserted in the middle of the MAC address when using ordinary EUI-64. Again, normally ISATAP clients would derive addresses from an ISATAP router rather than (improperly) use it for inter-site communications as I demonstrate here.

```
R1#show ipv6 interface brief tunnel15
Tunnel15          [up/up]
                  FE80::5EFE:A01:B01

R5#show ipv6 interface brief tunnel15
Tunnel15          [up/up]
                  FE80::5EFE:A02:505
```


Before continuing, we will check LL address reachability over the tunnel. As always, we verify the encapsulation with EPC. CSR1 will capture outgoing packets and CSR5 will initiate pings. We can see the IP protocol remains 41 since this is an IPv6-in-IPv4 tunnel (not GRE). IPv4/IPv6 source addresses are shown in yellow with destinations in green. Notice that the converted IPv4 addresses are included at the very end (least significant bits) of the IPv6 addresses, which is different from 6RD and 6to4.

```
R5#ping FE80::5EFE:A01:B01
Output Interface: tunnel15
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to FE80::5EFE:A01:B01, timeout is 2 seconds:
Packet sent with a source address of FE80::5EFE:A02:505%Tunnel15
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 3/8/22 ms
```

```
R1#show monitor capture CAP buffer dump
0
0000: 005056A9 9C600050 56A91AAA 81000DB7 .PV..`.PV.....
0010: 08004500 00780625 0000FF29 912F0A01 ..E..x.%...)../.
0020: 0B010A02 05056000 0000003C 3A40FE80 .....`....<:@..
0030: 00000000 00000000 5EFE0A01 0B01FE80 .....^.....
0040: 00000000 00000000 5EFE0A02 05058100 .....^.....
0050: 015E117D 00000001 02030405 06070809 .^.}.....
```

One interesting benefit of ISATAP is being able to run IGP over it. While there is no mechanism to transport IPv6 multicast traffic (which means no IGP or PIM neighbor discovery), we can use unicast neighbors over the tunnel to exchange routes. Using OSPFv3, for example, we can use network P2MP non-broadcast and specify the remote LL address as the neighbor. Since the tunnels have direct LL reachability, there is nothing stopping OSPFv3 from converging. Obviously, configuring the LL address neighbors of every node in the ISATAP mesh scales poorly, but this is a demonstration of the capability.

```
! CSR1 and CSR5
interface Tunnel15
  ospfv3 15 ipv6 area 0
  ospfv3 15 ipv6 network point-to-multipoint non-broadcast
  ospfv3 15 ipv6 hello-interval 10

! CSR1 only
interface Tunnel15
  ospfv3 15 ipv6 neighbor FE80::5EFE:A02:505

! CSR5 only
interface Tunnel15
  ospfv3 15 ipv6 neighbor FE80::5EFE:A01:B01
```

On CSR5, we verify the OSPFv3 neighbor is up. We only verify one side for brevity.

```
R5#show ospfv3 15 ipv6 neighbor
```

```
OSPFv3 15 address-family ipv6 (router-id 10.5.5.5)
```

Neighbor ID	Pri	State	Dead Time	Interface ID	Interface
10.1.1.1	0	FULL/ -	00:00:32	20	Tunnel15

We also add some loopbacks into OSPFv3 to ensure we can exchange public IPv6 prefixes. This means we will have OSPFv3 intra-area routes on either side via the ISATAP tunnel.

```
! CSR1
interface Loopback0
  ipv6 address 2001:1111:BABE::1/128
  ipv6 address 2001:1111:BEEF::1/128
  ospfv3 15 ipv6 area 0
```

```
! CSR5
interface Loopback0
  ipv6 address 2001:5555:CAFE::5/128
  ipv6 address 2001:5555:DEED::5/128
  ospfv3 15 ipv6 area 0
```

```
R1#show ipv6 route ospf | begin ^O
O   2001:5555:CAFE::5/128 [110/1000]
    via FE80::5EFE:A02:505, Tunnel15
O   2001:5555:DEED::5/128 [110/1000]
    via FE80::5EFE:A02:505, Tunnel15
```

```
R5#show ipv6 route ospf | begin ^O
O   2001:1111:BABE::1/128 [110/1000]
    via FE80::5EFE:A01:B01, Tunnel15
O   2001:1111:BEEF::1/128 [110/1000]
    via FE80::5EFE:A01:B01, Tunnel15
```

It is interesting to note that the ISATAP CEF lookups are fully built already, unlike 6RD and 6to4. This is because there is actually an IPv6 neighbor relationship (via ND) over the tunnel interface so that LL addressing is exchanged. The IPv4 CEF lookup knows that XRv1 is the next hop towards the destination because the IPv6 CEF table indicates CSR5's LL address the next-hop.

```
R1#show ipv6 route 2001:5555:DEED::5/128
Routing entry for 2001:5555:DEED::5/128
  Known via "ospf 15", distance 110, metric 1000, type intra area
  Route count is 1/1, share count 0
  Routing paths:
    FE80::5EFE:A02:505, Tunnel15
    Last updated 04:14:03 ago
```

```
R1#show ipv6 cef 2001:5555:DEED::5/128 internal | begin output
output chain:
  IPV6 midchain out of Tunnel15, addr FE80::5EFE:A02:505 7F27796C5DB8
```

```
IP adj out of GigabitEthernet2.511, addr 10.1.11.11 7F27796C6370
```

We can verify the encapsulation string for the ISATAP tunnel as well. The final destination of CSR5 is included as CEF is smart enough to see this information from the next-hop address from the RIB. This is somewhat similar to the 6RD BR routes on the 6RD clients; the remote encapsulation was known because the information was computed ahead of time and written to the adjacency table.

```
R1#show adjacency Tunnel15 FE80::5EFE:A02:505 encapsulation
Protocol Interface Address
IPV6 Tunnel15 FE80::5EFE:A02:505(8)
Encap length 20
4500000000000000FF2997CC0A010B01
0A020505
Provider: TUNNEL
Protocol header count in macstring: 1
HDR 0: ipv4
dst: static, 10.2.5.5
src: static, 10.1.11.1
prot: static, 41
ttl: static, 255
df: static, cleared
per packet fields: tos ident tl chksum
```

A quick ping check verifies that all 4 addresses are reachable across the ISATAP mesh.

```
R1#ping 2001:5555:CAFE::5 source 2001:1111:BABE::1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2001:5555:CAFE::5, timeout is 2 seconds:
Packet sent with a source address of 2001:1111:BABE::1
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 3/72/326 ms
```

```
R1#ping 2001:5555:DEED::5 source 2001:1111:BEEF::1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2001:5555:DEED::5, timeout is 2 seconds:
Packet sent with a source address of 2001:1111:BEEF::1
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 3/8/22 ms
```

36.5 IPv4/IPv6 Internet Access over MPLS using NAT44

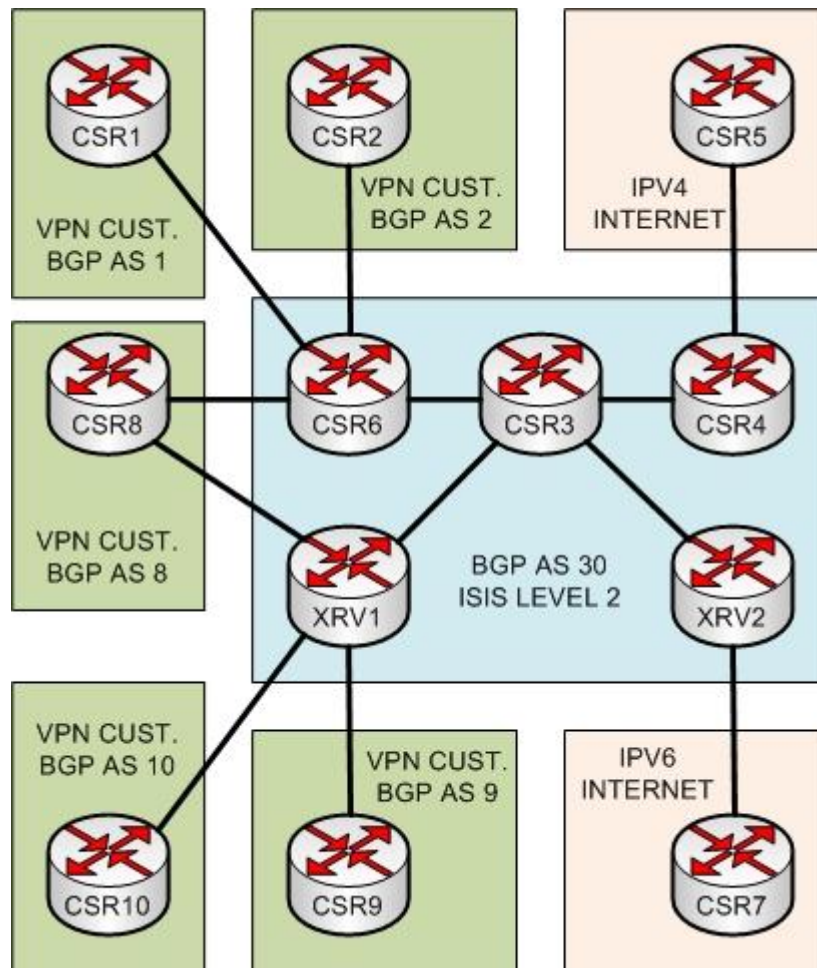
This lab shows how NAT44 and MPLS cooperate in a network to provide IPv4 Internet access to L3VPN customers. IPv6 Internet access is also demonstrated but without the NAT component. There are 3 general NAT44 solutions:

1. **Perform NAT44 on the CE.** Since the number of CEs is very large, this is a highly distributed solution. The SP never deals with any private source addresses from the customer for Internet-destined traffic and never deals with NAT at all. This requires the most NAT configuration overall

and increases complexity for customers. The complexity is further increased since customers need to perform NAT only for Internet-bound traffic, not for inter-site traffic.

2. **Perform NAT44 on the ingress PE.** The number of ingress PEs is less than the number of CEs, but greater than the number of Internet peering points (probably). This solution has medium complexity and medium scalability as the PE is now responsible for learning private LAN source addresses for Internet-bound traffic. The NAT matching could become complex as the ingress PE would need to differentiate between Internet-bound and private-WAN bound traffic to perform NAT intelligently.
3. **Perform NAT44 on the egress PE.** This solution scales the best as the number of Internet peering points is likely less than the number of user-facing PEs in the network. It also simplifies the NAT rules since inter-customer traffic would never cross a NAT device (the UPE would not perform NAT). The drawback is that the Internet peering router would be performing potentially billions of NAT operations with millions of NAT state entries, so the hardware would need to be very powerful.

The network diagram is shown below. CSR6 and XRv1 are PEs supporting IPv4, IPv6 labeled-unicast (6PE), VPNv4, and VPNv6. CSR4 is the IPv4 Internet gateway and XRv2 is the IPv6 Internet gateway. All other routers are customers that require access to both the IPv4 and IPv6 Internet. The basic IGP, MPLS, and BGP configurations are not shown. To focus on NAT and MPLS interaction, all of the sites are only drawing Internet service and are not connected in any kind of inter-site topology. That is to say, each customer router is its own customer site with no lateral VPN route exchanges.



CSR4 has IPv4 and VPNv4 peers to each PE. XRV2 has IPv6 labeled-unicast and VPNv6 peers to each PE. The purpose of the IPv4/v6 unicast AFs is to support route leaking. This allows the PEs to import the Internet routes from global to VRF (the Internet gateways exported them from VRF to global) as another mechanism for providing central services. Although more complex and sloppier, it assumes that the routers already have Internet routes in their global tables in the first place, so replicating that information in a VPN could be inefficient with respect to memory consumption. We quickly verify these BGP sessions are functional before continuing.

```
R4#show bgp ipv4 unicast summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent   TblVer  InQ  OutQ Up/Down  State/PfxRcd
30.0.0.6      4       30   8535   8581     53    0    0 22:25:16      0
30.0.0.11     4       30   8270   8805     53    0    0 22:59:44      0
```

```
R4#show bgp vpnv4 unicast all summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent   TblVer  InQ  OutQ Up/Down  State/PfxRcd
10.4.5.5      4       5    1568   1591    171    0    0 23:13:16      4
30.0.0.6      4       30   8514   8560    171    0    0 22:21:53      0
30.0.0.11     4       30   8249   8784    171    0    0 22:56:21      0
```

```
RP/0/0/CPU0:XRV2#show bgp vpnv6 unicast summary | begin ^Neigh
```

Neighbor	Spk	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	St/PfxRcd
30.0.0.6	0	30	8924	8419	39	0	0	21:29:54	0
30.0.0.11	0	30	8405	8414	39	0	0	21:29:36	0

```
RP/0/0/CPU0:XRv2#show bgp ipv6 labeled-unicast summary | begin ^Neigh
```

Neighbor	Spk	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	St/PfxRcd
30.0.0.6	0	30	8925	8420	7	0	0	21:30:05	0
30.0.0.11	0	30	8406	8415	7	0	0	21:29:47	0

The Internet routers are injecting prefixes into AS 30. Both CSR4 and XRv2 store these in a VRF table, which is a central services design. The VRF configuration is shown below for CSR4 and XRv2. The Internet routes are verified to be present inside VPNv4 BGP but with bogus RTs (no one is importing RT:4:999). Within the L3VPN, only the default route needs to be advertised to the remote PEs as this will guarantee the customers do not get the “full view” of all Internet routes. The PEs and other ISP routers, however, may require this full view so that they could be Internet transit points. These are arbitrary requirements fabricated for the purpose of this lab. The export map for VPN attributes on CSR4 appends RT:4:5 to the default route only, which all customer VRFs will import. The export map for route-leaking permits the default route as well as the Internet routing table. CSR4 imports the RTs of all customers to provide central services. The customer VRFs have not been configured yet but we can stage the RT import policies to save time later.

```
! CSR4
ip prefix-list PL_DEFAULT seq 5 permit 0.0.0.0/0

route-map RM_EXPORT_VRF_5 permit 10
 match ip address prefix-list PL_DEFAULT
 set extcommunity rt 4:5 additive

ip prefix-list PL_INTERNET seq 5 permit 55.0.0.0/8 le 32
ip prefix-list PL_INTERNET seq 10 permit 0.0.0.0/0

route-map RM_VRF_5_TO_GLOBAL permit 10
 match ip address prefix-list PL_INTERNET

vrf definition 5
 rd 4:5
 address-family ipv4
  export map RM_EXPORT_VRF_5
  export ipv4 unicast map RM_VRF_5_TO_GLOBAL
  route-target export 4:999
  route-target import 6:2
  route-target import 6:1
  route-target import 6:8
  route-target import 11:8
  route-target import 11:9
  route-target import 11:10
```

The logic is similar on XRv2. XR route leaking logic is simpler than XE since no bogus export-RTs are required. The export policy for RT assigns RT:12:7 to the default route as the parameterized RPL makes clear. The export route leaking policy leaks the default route only. This example might be valuable to provide IPv6 Internet service to a customer not in a VPN, so leaking it at the ingress PE might be easier than leaking it at the egress PE.

```
! XRv2
vrf 7
  address-family ipv6 unicast
    import route-target
      6:1
      6:2
      6:8
      11:8
      11:9
      11:10

    export route-policy RPL_IF_DEST_SET_RT(PS_DEFAULT, RT_12_7)
    export to default-vrf route-policy RPL_IF_DEST_PASS(PS_DEFAULT)

  extcommunity-set rt RT_12_7
    12:7
  end-set

  prefix-set PS_DEFAULT
    ::/0
  end-set

  route-policy RPL_IF_DEST_PASS($PS)
    if destination in $PS then
      pass
    endif
  end-policy

  route-policy RPL_IF_DEST_SET_RT($PS, $RT)
    if destination in $PS then
      set extcommunity rt $RT additive
    endif
  end-policy
```

Now, the Internet VRF is configured to apply the correct behavior to the default route. This will allow it to leak into the global table as well as receive a valid RT so that customers may import it. CSR4 and XRv2 inject static default routes into their Internet VPNs as shown below. This can be advertised towards the central services customers in lieu of the Internet routing table. The minor difference between CSR4 and XRv2 in static route next-hops has a significant impact on MPLS forwarding. This is analyzed later.

```
! CSR4
```

```

ip route vrf 5 0.0.0.0 0.0.0.0 Null0

router bgp 30
  address-family ipv4 vrf 5
    redistribute static
    default-information originate

! XRv2
route-policy RPL_PASS
  pass
end-policy

route-policy RPL_IF_TAG_PASS($TAG)
  if tag is $TAG then
    pass
  endif
end-policy

router static
  vrf 7
  address-family ipv6 unicast
    ::/0 GigabitEthernet0/0/0/0.572 fe80::7 tag 7

router bgp 30
  vrf 7
  rd 12:7
  default-information originate
  address-family ipv6 unicast
    redistribute static route-policy RPL_IF_TAG_PASS(7)

```

Most of the basic configuration is complete on CSR4 and XRv2 at this point. To verify this configuration on CSR4, we can check the default route and any other arbitrary Internet route. The Internet route only has RT:4:999, and the only reason it needs any RT is because route leaking requires it (this is detailed in the route leaking section). The default route carrying the bogus RT in addition to the real one has no significance. This verifies that the RT export-map worked. To verify the route-leaking, we check the global IPv4 BGP RIB. We see both the Internet and default routes, as expected.

```

R4#show bgp vpnv4 unicast vrf 5 55.46.253.0/26 | include RT:
  Extended Community: RT:4:999

```

```

R4#show bgp vpnv4 unicast vrf 5 0.0.0.0/0 | include RT:
  Extended Community: RT:4:5 RT:4:999

```

```

R4#show bgp ipv4 unicast 0.0.0.0/0 | include import
  Local, imported path from 4:5:0.0.0.0/0 (5)

```

```

R4#show bgp ipv4 unicast 55.46.253.128 | include import
  5, imported path from 4:5:55.46.253.128/26 (5)

```


To verify the configuration on XRv2, we follow a similar procedure. The default route gets RT:12:7 which is what all central services customers must import in order to get Internet connectivity. The original IPv6 Internet route has no RTs applied, which is expected. Unlike IPv4, the IPv6 Internet routes are not allowed to leak into the global table; only the default route is. Again, this is just for variety and is not necessary a good design choice. We confirm this by checking the IPv6 RIB. Notice that the default route has a next-hop to be resolved inside VRF 7, as expected.

```
RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast vrf 7 ::/0 | include RT:
    Extended community: RT:12:7
```

```
RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast vrf 7 ::77:babe:3/128 | include RT:
[no output]
```

```
RP/0/0/CPU0:XRv2#show route ipv6 ::/0
Routing entry for ::/0
    Known via "bgp 30", distance 200, metric 0, candidate default path, type
    locally generated
    Routing Descriptor Blocks
        fe80::7, from ::, via GigabitEthernet0/0/0/0.572
        Nexthop in Vrf: "7", Table: "default", IPv6 Unicast, Table Id:
        0xe0800012
        Route metric is 0
        No advertising protos.
```

```
RP/0/0/CPU0:XRv2#show route ipv6 ::77:babe:3/128
% Network not in table
```

Before continuing, we must understand the forwarding differences between the IPv4 and IPv6 egress PEs given their default route configurations. CSR4 has a VPNv4 route for 0.0.0.0/0 which is a local aggregate. When packets arrive with label 4009, CSR4 removes all labels from the stack, and is directed by the LFIB to consult the IPv4 FIB inside VRF 5. CSR4 would need a longer-match route for the final destination as a result, which it learns via BGP for all Internet routes. This two-step lookup process can be inefficient since the incoming VPN label cannot map to an LFIB next-hop as the underlying RIB does not specify one for the aggregate default route.

```
R4#show bgp vpnv4 unicast vrf 5 0.0.0.0/0
BGP routing table entry for 4:5:0.0.0.0/0, version 169
Paths: (1 available, best #1, table 5)
    Advertised to update-groups:
        1
    Refresh Epoch 1
    Local
        0.0.0.0 (via default) from 0.0.0.0 (30.0.0.4)
        Origin incomplete, metric 0, localpref 100, weight 32768, valid,
        sourced, best
        Extended Community: RT:4:5 RT:4:999
```

```
mpls labels in/out 4009/aggregate(5)
rx pathid: 0, tx pathid: 0x0
```

```
R4#show mpls forwarding-table labels 4009
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Label	Outgoing interface	Next Hop
4009	No Label	0.0.0.0/0[V]	4720		aggregate/5	

```
R4#show ip route vrf 5 55.0.0.0 255.0.0.0 longer-prefixes | begin Gate
Gateway of last resort is 0.0.0.0 to network 0.0.0.0
```

```
55.0.0.0/26 is subnetted, 4 subnets
B       55.46.253.0 [20/0] via 10.4.5.5, 22:24:44
B       55.46.253.64 [20/0] via 10.4.5.5, 22:24:44
B       55.46.253.128 [20/0] via 10.4.5.5, 22:24:44
B       55.46.253.192 [20/0] via 10.4.5.5, 22:24:44
```

On XRv2, this problem does not exist. The static route is specified as being accessible via CSR7, so the LFIB can use this when binding the VPN label to the next-hop inside the VRF. This is efficient since only a single lookup needs to occur. In general, this is a better design. Alternative designs could include simply learning a default from the Internet in the first place, if possible.

```
RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast vrf 7 ::/0
BGP routing table entry for ::/0, Route Distinguisher: 12:7
```

```
Versions:
```

Process	bRIB/RIB	SendTblVer
Speaker	33	33

```
Local Label: 92008
```

```
Paths: (1 available, best #1)
```

```
Advertised to update-groups (with more than one peer):
0.2
```

```
Path #1: Received by speaker 0
```

```
Advertised to update-groups (with more than one peer):
0.2
```

```
Local
```

```
fe80::7 from 0.0.0.0 (30.0.0.12)
```

```
Origin incomplete, metric 0, localpref 100, weight 32768, valid,
redistributed, best, group-best, import-candidate
```

```
Received Path ID 0, Local Path ID 1, version 33
```

```
Extended community: RT:12:7
```

```
RP/0/0/CPU0:XRv2#show mpls forwarding vrf 7 | include 92008
```

92008	Unlabelled	::/0[V]	Gi0/0/0/0.572 fe80::7	2500

Next, we confirm that the remote PEs (CSR6 and XRv1) learn the IPv4/v6 Internet and default routes. Beginning with the global table on CSR6, we see the full Internet routing table. CSR6 could technically leak the entire IPv4 Internet routing table to a customer, if necessary. For IPv6, only the default route is learned since XRv2 did not leak the IPv6 Internet routing table.

```
R6#show ip route bgp | begin Gate
Gateway of last resort is 30.0.0.4 to network 0.0.0.0
```

```
B*    0.0.0.0/0 [200/0] via 30.0.0.4, 1d13h
      55.0.0.0/26 is subnetted, 4 subnets
B      55.46.253.0 [200/0] via 10.4.5.5, 1d13h
B      55.46.253.64 [200/0] via 10.4.5.5, 1d13h
B      55.46.253.128 [200/0] via 10.4.5.5, 1d13h
B      55.46.253.192 [200/0] via 10.4.5.5, 1d13h
```

```
R6#show ipv6 route bgp | begin Appl
      a - Application
B    ::/0 [200/0]
      via 30.0.0.12%default, indirectly connected
```

Of note, the IPv6 default route also has an associated BGP label from XRv2. This is the result of using 6PE, which is required to tunnel IPv6 traffic across an IPv4-only MPLS core inside the global table.

```
R6#show ipv6 route ::/0
Routing entry for ::/0
  Known via "bgp 30", distance 200, metric 0, type internal
  Route count is 1/1, share count 0
  Routing paths:
    30.0.0.12%default indirectly connected
    MPLS label: 92009
    Last updated 3d11h ago
```

Now that the IPv4/v6 Internet routing advertisements have been verified, we will configure the customer VPNs. CSR6 and XRv1 won't learn any VPNv4/v6 routes until some VRFs are configured to import the RTs the Internet peering routers are exporting (RT retention is always disabled by default). Each CE router will perform a different combination of IPv4 and IPv6 Internet access for variety. CSR9 will use BGP for IPv4 to learn the default route from XRv1 inside of the VPN; this is a traditional central services model. At the same time, the IPv6 default route will be leaked from global to VRF on XRv1. First, we must configure XRv1, the PE, to provide these routing services. For IPv4, the central services RT:4:5 is imported and a unique RT:11:9 is exported. Earlier, we saw RT:11:9 imported by CSR4, which is required for bidirectional connectivity. Under IPv6, no RTs are imported since only global routes are leaked. XRv2 imports RT:11:9 which is exported by XRv1.

```
! XRv1
prefix-set PS_DEFAULT
  ::/0
end-set

route-policy RPL_IF_DEST_PASS($PS)
  if destination in $PS then
```

```

        pass
    endif
end-policy

vrf 9
    address-family ipv4 unicast
        import route-target
        4:5
    export route-target
    11:9

address-family ipv6 unicast
    import from default-vrf route-policy RPL_IF_DEST_PASS(PS_DEFAULT)
    export route-target
    11:9

```

XRv1's corresponding BGP configuration is very straightforward. Since routing information is exchanged with CSR9 in both AFIs, both IPv4 and IPv6 unicast are configured.

```

! XRv1
route-policy RPL_PASS
    pass
end-policy

router bgp 30
    vrf 9
        rd 11:9
        address-family ipv4 unicast
        address-family ipv6 unicast

    neighbor 10.9.11.9
        remote-as 9
        address-family ipv4 unicast
            route-policy RPL_PASS in
            route-policy RPL_PASS out

    neighbor fd00:10:9:11::9
        remote-as 9
        address-family ipv6 unicast
            route-policy RPL_PASS in
            route-policy RPL_PASS out

```

Despite using different methods to import the default route into the IPv4 and IPv6 AFIs within the VPN, the routes generally look the same once imported. Both of them have next-hops in the default table.

```

RP/0/0/CPU0:XRv1#show route vrf 9 ipv4 0.0.0.0/0
Routing entry for 0.0.0.0/0

```

```

Known via "bgp 30", distance 200, metric 0, candidate default path, type
internal
Routing Descriptor Blocks
  30.0.0.4, from 30.0.0.4
    Nexthop in Vrf: "default", Table: "default", IPv4 Unicast, Table Id:
0xe0000000
      Route metric is 0
      No advertising protos.

```

```

RP/0/0/CPU0:XRv1#show route vrf 9 ipv6 ::/0
Routing entry for ::/0
  Known via "bgp 30", distance 200, metric 0, candidate default path, type
internal
Routing Descriptor Blocks
  ::ffff:30.0.0.12, from ::ffff:30.0.0.12
    Nexthop in Vrf: "default", Table: "default", IPv4 Unicast, Table Id:
0xe0000000
      Route metric is 0
      No advertising protos.

```

CSR9's BGP configuration is very basic and is shown below. After configuring BGP, CSR9 should learn exactly 1 route for each AFI from XRv1, which is the default.

```

! CSR9
router bgp 9
  no bgp default ipv4-unicast
  neighbor 10.9.11.11 remote-as 30
  neighbor FD00:10:9:11::11 remote-as 30

  address-family ipv4
    neighbor 10.9.11.11 activate
    neighbor 10.9.11.11 maximum-prefix 5

  address-family ipv6
    neighbor FD00:10:9:11::11 activate

```

```

R9#show bgp ipv4 unicast summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
10.9.11.11    4        30   5105   5677     106    0    0 3d13h      1

```

```

R9#show bgp ipv6 unicast summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
FD00:10:9:11::11
                4        30   5015   5572     207    0    0 3d11h      1

```

```

R9#show ip cef 0.0.0.0 0.0.0.0
0.0.0.0/0
  nexthop 10.9.11.11 GigabitEthernet2.591

```

```
R9#show ipv6 cef ::/0
::/0
  nexthop FE80::11 GigabitEthernet2.591
```

CSR9 has only IPv4 private addressing on its LAN simulated by loopback0. One NAT solution for this “IPv4 Internet access” architecture is to perform NAT on the CE. This is the simplest solution but adds complexity for the customer. This problem does not exist for IPv6, so we configure a public IPv6 subnet on the loopback. The NAT configuration is very basic as CSR9 has a pool of 16 usable IP addresses. A static null route matches the pool and is advertised into BGP so that return traffic routes back to CSR9 properly.

```
! CSR9
interface GigabitEthernet2.591
 ip nat outside

interface Loopback0
 ip address 192.168.9.1 255.255.255.255
 ip nat inside
 ipv6 address 9000::9/16

ip nat pool NAT_POOL 9.0.0.0 9.0.0.15 prefix-length 28
ip nat inside source list ACL_NAT pool NAT_POOL
ip access-list standard ACL_NAT
 permit 192.168.9.0 0.0.0.255

ip route 9.0.0.0 255.255.255.240 Null0

router bgp 9
 address-family ipv4
  network 9.0.0.0 mask 255.255.255.240

 address-family ipv6
  network 9000::/16
```

We verify that the PE learns the IPv4 NAT pool and IPv6 public routes from CSR9. These will ultimately be advertised towards the IPv4/v6 Internet.

```
RP/0/0/CPU0:XRv1#show bgp vrf 9 ipv4 unicast neighbors 10.9.11.9 routes |
begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 11:9 (default for vrf 9)					
*> 9.0.0.0/28	10.9.11.9	0		0	9 i

```
RP/0/0/CPU0:XRv1#show bgp vrf 9 ipv6 unicast neighbors fd00:10:9:11::9 routes
| begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 11:9 (default for vrf 9)					

```
*> 9000::/16          fd00:10:9:11::9          0          0 9 i
```

We quickly confirm that the Internet routers CSR5 and CSR7 learn these routes originated by CSR9. This completes the control plane verification.

```
R5#show bgp ipv4 unicast 9.0.0.0/28
BGP routing table entry for 9.0.0.0/28, version 21
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  30 9
    10.4.5.4 from 10.4.5.4 (30.0.0.4)
      Origin IGP, localpref 100, valid, external, best
      rx pathid: 0, tx pathid: 0x0
```

```
R7#show bgp ipv6 unicast 9000::/16
BGP routing table entry for 9000::/16, version 1014
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  30 9
    FD00:10:7:12::12 (FE80::12) from FD00:10:7:12::12 (30.0.0.12)
      Origin IGP, localpref 100, valid, external, best
      rx pathid: 0, tx pathid: 0x0
```

Testing outbound reachability from CSR9 for IPv4, we can see that connectivity is achieved. A NAT entry was created with the source IP addresses 192.168.9.1 translated to 9.0.0.0. The destination addresses are not modified with basic “inside” NAT.

```
R9#ping 55.46.253.1 source loopback0
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 55.46.253.1, timeout is 2 seconds:
Packet sent with a source address of 192.168.9.1
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 7/8/10 ms
```

```
R9#show ip nat translations
Pro  Inside global      Inside local      Outside local
Outside global
---  9.0.0.0               192.168.9.1      ---
icmp  9.0.0.0:3            192.168.9.1:3    55.46.253.1:3
55.46.253.1:3
Total number of translations: 2
```

XRv1 imposes two MPLS labels to transport the IPv4 packets towards CSR4. The VPN label was allocated by CSR4 to describe the path towards the default route; since XRv1 has no longer-matching VPN route

for 55.46.253.1, the label 4009 for the default route is used. Label 3001 is CSR3's LDP label for 34.0.0.4/32.

```
! XRv1
RP/0/0/CPU0:XRv1#show cef vrf 9 55.46.253.1
0.0.0.0/0, version 4, proxy default, internal 0x5000011 0x0 (ptr 0xa1448074)
[1], 0x0 (0x0), 0x208 (0xa156daf0)
Prefix Len 0, traffic index 0, precedence n/a, priority 3
via 30.0.0.4, 7 dependencies, recursive [flags 0x6000]
path-idx 0 NHID 0x0 [0xa15d68f4 0x0]
recursion-via-/32
next hop VRF - 'default', table - 0xe0000000
next hop 30.0.0.4 via 91001/0/21
next hop 30.3.11.3/32 Gi0/0/0/0.531 labels imposed {3001 4009}
```

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast vrf 9 labels | begin Network
Network           Next Hop           Rcvd Label         Local Label
Route Distinguisher: 11:9 (default for vrf 9)
*>i0.0.0.0/0        30.0.0.4           4009               nolabel
*> 9.0.0.0/28       10.9.11.9          nolabel            91003
```

```
RP/0/0/CPU0:XRv1#show mpls ldp bindings 30.0.0.4/32 neighbor 30.0.0.3
30.0.0.4/32, rev 12
Local binding: label: 91001
Remote bindings: (1 peers)
Peer           Label
-----
30.0.0.3:0     3001
```

CSR3 performs PHP as a P router to expose label 4009 to CSR4. When CSR4 receives label 4009, the LFIB directs the router to perform another lookup inside the VRF 5 FIB. This is because 0.0.0.0/0 was a local aggregate and CSR4 cannot forward the packet based on the MPLS label alone.

```
R3#show mpls forwarding-table labels 3001
Local      Outgoing  Prefix           Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id     Switched     interface
3001       Pop Label  30.0.0.4/32      8706497      Gi2.534   30.3.4.4
```

```
R4#show mpls forwarding-table labels 4009
Local      Outgoing  Prefix           Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id     Switched     interface
4009       No Label  0.0.0.0/0[V]     5310         aggregate/5
```

```
R4#show ip cef vrf 5 55.46.253.1
55.46.253.0/26
nexthop 10.4.5.5 GigabitEthernet2.545
```


When CSR5 responds, it sends its replies to 9.0.0.0. CSR4 imposes two labels just as XRv1 did in the opposite direction. Label 91003 is the VPN label for 9.0.0.0/28 from XRv1 and label 3002 is CSR3's LDP binding for 30.0.0.11/32.

```
R4#show ip cef vrf 5 9.0.0.0
9.0.0.0/28
  nexthop 30.3.4.3 GigabitEthernet2.534 label 3002 91003

R4#show bgp vpnv4 unicast vrf 5 9.0.0.0/28
BGP routing table entry for 4:5:9.0.0.0/28, version 53
Paths: (1 available, best #1, table 5)
  Advertised to update-groups:
    3
  Refresh Epoch 1
  9, imported path from 11:9:9.0.0.0/28 (global)
    30.0.0.11 (metric 20) (via default) from 30.0.0.11 (30.0.0.11)
      Origin IGP, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:11:9
      mpls labels in/out nolaabel/91003
      rx pathid: 0, tx pathid: 0x0

R4#show mpls ldp bindings 30.0.0.11 32 neighbor 30.0.0.3
  lib entry: 30.0.0.11/32, rev 8
    remote binding: lsr: 30.0.0.3:0, label: 3002
```

CSR3 is a P router performing PHP and XRv11 forwards packets towards CSR9. Since 9.0.0.0/28 was not a local aggregate defined on XRv1, we do not see the inefficiency of multiple LFIB/FIB lookups on XRv1 as we did on CSR4. When CSR9 receives the packets on the outside interface, NAT happens before routing, so the destination of 9.0.0.0 is changed back to 192.168.9.1 and the packet is routed to the loopback.

```
R3#show mpls forwarding-table labels 3002
Local      Outgoing   Prefix      Bytes Label  Outgoing   Next Hop
Label      Label        or Tunnel Id Switched     interface
3002      Pop Label    30.0.0.11/32 8534837      Gi2.531    30.3.11.11

RP/0/0/CPU0:XRv1#show mpls forwarding vrf 9 prefix 9.0.0.0/28
Local      Outgoing   Prefix      Outgoing     Next Hop      Bytes
Label      Label        or ID       Interface    Interface      Switched
-----
91003      Unlabelled   9.0.0.0/28[V] Gi0/0/0/0.591 10.9.11.9      2000
```

CSR9's connectivity to the IPv6 Internet is less interesting as there is no NAT in play. Rather than trace the LSPs for a very basic L3VPN connection (the route leaking as already verified), I use traceroute from CSR9 (CE) and CSR7 (IPv6 Internet) for brevity. 2 labels are imposed by the ingress LSR in either direction as expected, and the packets are label-switched across the core. The only interesting part of the output below is label 92009; this is technically an IPv6 labeled-unicast label, not a VPNv6 label, since the route

was imported from the global table. When XRv2 leaked the route from VRF to global, a new local label was allocated and advertised via BGP IPv6 labeled-unicast.

```
R9#traceroute ipv6
Target IPv6 address: ::77:BABE:0
Source address: 9000::9
[snip]
Tracing the route to ::77:BABE:0
 1 FD00:10:9:11::11 [AS 30] 3 msec 3 msec 2 msec
 2 ::FFFF:30.3.11.3 [AS 30] [MPLS: Labels 3003/92009 Exp 0] 9 msec 7 msec 8
msec
 3 FD00:10:7:12::12 [AS 30] [MPLS: Label 92009 Exp 0] 25 msec 20 msec 27
msec
 4 FD00:10:7:12::7 [AS 30] 27 msec 15 msec 15 msec
```

```
R7#traceroute ipv6
Target IPv6 address: 9000::9
Source address: ::77:BABE:0
[snip]
Tracing the route to 9000::9
 1 FD00:10:7:12::12 [AS 30] 3 msec 3 msec 3 msec
 2 ::FFFF:30.3.12.3 [AS 30] [MPLS: Labels 3002/91007 Exp 0] 7 msec 7 msec 9
msec
 3 FD00:10:9:11::11 [AS 30] [MPLS: Label 91007 Exp 0] 23 msec 23 msec 22
msec
 4 FD00:10:9:11::9 [AS 30] 27 msec 15 msec 15 msec
```

A quick verification on XRv2 proves that a new label was issued; the VPNv6 label of 92008 was not carried over during the VRF-to-global export process. This is the label learned by XRv1 in the global table.

```
RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast vrf 7 ::/0 | include Local Label
Local Label: 92008
```

```
RP/0/0/CPU0:XRv2#show route ipv6 ::/0 detail | include Local Label
Local Label: 0x16769 (92009)
```

```
RP/0/0/CPU0:XRv1#show route ipv6 unicast ::/0 detail | include Label
Label: 0x16769 (92009)
```

Next, we will use a set of alternative techniques on CSR2 and CSR8. Like CSR9, these are CE routers with private IPv4 addressing, and thus require NAT. CSR6, the ingress PE, will perform NAT for these clients. This means that CSR6 will need to learn the private addresses from the CE and not distribute them indiscriminately to the IPv4 Internet. For IPv6, both clients will be in a basic L3VPN central services VPN. This is usually the best approach for IPv6 Internet connectivity when only a default route is required. For brevity, CSR2 and CSR8 interface configurations are not shown, but their private networks are

192.168.2.1/24 and 192.168.8.1/24, respectively. The IPv6 networks 2000::/16 and 8000::/16 are used by CSR2 and CSR8, respectively, as well. First, I show the VPN-specific configurations on CSR6. There is no route leaking with this example; the VRFs just perform standard RT import/export actions. The only imported RT for both VRFs is the central services RT for each AFI.

```
! CSR6
vrf definition 2
  rd 6:2
  address-family ipv4
    route-target export 6:2
    route-target import 4:5

  address-family ipv6
    route-target export 6:2
    route-target import 12:7

vrf definition 8
  rd 6:8
  address-family ipv4
    route-target export 6:8
    route-target import 4:5

  address-family ipv6
    route-target export 6:8
    route-target import 12:7
```

Next, we configure BGP on the PE. CSR6 has the responsibility to ensure that private routes learned from the customer are not advertised further. Rather than actually match the private range, CSR6 marks all routes from the customers with the “no-advertise” community, which prevents it from being advertised at all. This solution works well when the customer is ONLY sending private routes to the CE and there is no possibility of public IPv4 routes being advertised.

```
! CSR6
route-map RM_SET_NO_ADV permit 10
  set community no-advertise

router bgp 30
  address-family ipv4 vrf 2
    neighbor 10.2.6.2 remote-as 2
    neighbor 10.2.6.2 activate
    neighbor 10.2.6.2 route-map RM_SET_NO_ADV in

  address-family ipv6 vrf 2
    neighbor FD00:10:2:6::2 remote-as 2
    neighbor FD00:10:2:6::2 activate

  address-family ipv4 vrf 8
```

```
neighbor 10.6.8.8 remote-as 8
neighbor 10.6.8.8 activate
neighbor 10.6.8.8 route-map RM_SET_NO_ADV in
```

```
address-family ipv6 vrf 8
neighbor FD00:10:6:8::8 remote-as 8
neighbor FD00:10:6:8::8 activate
```

Below are the basic BGP configurations on CSR2 and CSR8. They advertise their private networks to CSR6 and do not perform local NAT44.

```
! CSR2
router bgp 2
no bgp default ipv4-unicast
neighbor 10.2.6.6 remote-as 30
neighbor FD00:10:2:6::6 remote-as 30
```

```
address-family ipv4
network 192.168.2.0
neighbor 10.2.6.6 activate
```

```
address-family ipv6
network 2000::/16
neighbor FD00:10:2:6::6 activate
```

```
! CSR8
router bgp 8
no bgp default ipv4-unicast
neighbor 10.6.8.6 remote-as 30
neighbor FD00:10:6:8::6 remote-as 30
```

```
address-family ipv4
network 192.168.8.0
neighbor 10.6.8.6 activate
```

```
address-family ipv6
network 8000::/16
neighbor FD00:10:6:8::6 activate
```

Before configuring NAT on CSR6, we verify that the PE learns the private IPv4 routes from both customers and the “no-advertise” community is set. The IPv6 routes are also present but lack this community as they are public and can be advertised to the IPv6 Internet.

```
R6#show bgp vpnv4 unicast vrf 2 192.168.2.0/24
BGP routing table entry for 6:2:192.168.2.0/24, version 5
Paths: (1 available, best #1, table 2, not advertised to any peer)
Not advertised to any peer
Refresh Epoch 1
```

2

```
10.2.6.2 (via vrf 2) from 10.2.6.2 (10.2.6.2)
  Origin IGP, metric 0, localpref 100, valid, external, best
  Community: no-advertise
  Extended Community: RT:6:2
  mpls labels in/out 6019/nolabel
  rx pathid: 0, tx pathid: 0x0
```

```
R6#show bgp vpnv4 unicast vrf 8 192.168.8.0/24
BGP routing table entry for 6:8:192.168.8.0/24, version 6
Paths: (1 available, best #1, table 8, not advertised to any peer)
  Not advertised to any peer
  Refresh Epoch 1
```

8

```
10.6.8.8 (via vrf 8) from 10.6.8.8 (192.168.8.1)
  Origin IGP, metric 0, localpref 100, valid, external, best
  Community: no-advertise
  Extended Community: RT:6:8
  mpls labels in/out 6023/nolabel
  rx pathid: 0, tx pathid: 0x0
```

```
R6#show bgp vpnv6 unicast vrf 2 2000::/16
BGP routing table entry for [6:2]2000::/16, version 397
Paths: (1 available, best #1, table 2)
  Advertised to update-groups:
```

3

Refresh Epoch 1

2

```
FD00:10:2:6::2 (FE80::2) (via vrf 2) from FD00:10:2:6::2 (10.2.6.2)
  Origin IGP, metric 0, localpref 100, valid, external, best
  Extended Community: RT:6:2
  mpls labels in/out 6043/nolabel
  rx pathid: 0, tx pathid: 0x0
```

```
R6#show bgp vpnv6 unicast vrf 8 8000::/16
BGP routing table entry for [6:8]8000::/16, version 398
Paths: (1 available, best #1, table 8)
  Advertised to update-groups:
```

3

Refresh Epoch 1

8

```
FD00:10:6:8::8 (FE80::8) (via vrf 8) from FD00:10:6:8::8 (192.168.8.1)
  Origin IGP, metric 0, localpref 100, valid, external, best
  Extended Community: RT:6:8
  mpls labels in/out 6021/nolabel
  rx pathid: 0, tx pathid: 0x0
```

On the other side of the BGP connection, we verify that both CSR2 and CSR8 learn the default route from CSR6 in both AFIs.

```

R2#show bgp ipv4 unicast 0.0.0.0/0
BGP routing table entry for 0.0.0.0/0, version 17
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  30
    10.2.6.6 from 10.2.6.6 (30.0.0.6)
      Origin incomplete, localpref 100, valid, external, best
      rx pathid: 0, tx pathid: 0x0

```

```

R8#show bgp ipv4 unicast 0.0.0.0/0
BGP routing table entry for 0.0.0.0/0, version 113
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  30
    10.6.8.6 from 10.6.8.6 (30.0.0.6)
      Origin incomplete, localpref 100, valid, external, best
      rx pathid: 0, tx pathid: 0x0

```

```

R2#show bgp ipv6 unicast ::/0
BGP routing table entry for ::/0, version 212
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  30
    FD00:10:2:6::6 (FE80::6) from FD00:10:2:6::6 (30.0.0.6)
      Origin incomplete, localpref 100, valid, external, best
      rx pathid: 0, tx pathid: 0x0

```

```

R8#show bgp ipv6 unicast ::/0
BGP routing table entry for ::/0, version 242
Paths: (1 available, best #1, table default)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  30
    FD00:10:6:8::6 (FE80::6) from FD00:10:6:8::6 (30.0.0.6)
      Origin incomplete, localpref 100, weight 66, valid, external, best
      rx pathid: 0, tx pathid: 0x0

```

NAT configuration, even when crossing VRFs, is very simple. Inside interfaces are configured towards the customer (inside VRF) and outside interfaces are configured towards the MPLS core (global interfaces). Two NAT pools are defined, along with corresponding VRF-aware static null routes. NAT rules are applied within a VRF to perform NAT source translation for the private networks within AS 2 and AS 8. There is nothing different about the NAT configuration other than ensuring the static routes and NAT rules are VRF-aware.

```

! CSR6
interface GigabitEthernet2.536
 ip nat outside

interface GigabitEthernet2.526
 ip nat inside

interface GigabitEthernet2.568
 ip nat inside

ip nat pool NAT_POOL_2 2.0.0.0 2.0.0.15 prefix-length 28
ip nat pool NAT_POOL_8 8.0.0.0 8.0.0.15 prefix-length 28

ip route vrf 2 2.0.0.0 255.255.255.240 Null0
ip route vrf 8 8.0.0.0 255.255.255.240 Null0

ip access-list standard ACL_NAT_2
 permit 192.168.2.0 0.0.0.255
ip access-list standard ACL_NAT_8
 permit 192.168.8.0 0.0.0.255

ip nat inside source list ACL_NAT_2 pool NAT_POOL_2 vrf 2
ip nat inside source list ACL_NAT_8 pool NAT_POOL_8 vrf 8

```

To complete the configuration, the static null routes defined above are advertised into BGP via network statements. To ensure these routes are not advertised down to the customers, a prefix-list that only permits the default route is applied. Although learning these NAT pool routes are harmless for CSR2 and CSR8, it is sloppy from a design perspective.

```

! CSR6
ip prefix-list PL_DEFAULT seq 5 permit 0.0.0.0/0

router bgp 30
 address-family ipv4 vrf 2
  network 2.0.0.0 mask 255.255.255.240
  neighbor 10.2.6.2 prefix-list PL_DEFAULT out

 address-family ipv4 vrf 8
  network 8.0.0.0 mask 255.255.255.240
  neighbor 10.6.8.8 prefix-list PL_DEFAULT out

```

As a quick verification, we confirm that CSR5 (IPv4 Internet) learns the routes to 2.0.0.0/28 and 8.0.0.0/28, but do not see routes for any private networks. We also confirm that CSR2 and CSR8 (CE routers) do not learn the NAT pools that CSR6 defined; they only learn the default routes for IPv4 as seen earlier.

```
R5#show bgp ipv4 unicast 2.0.0.0/28
BGP routing table entry for 2.0.0.0/28, version 37
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  30
    10.4.5.4 from 10.4.5.4 (30.0.0.4)
      Origin IGP, localpref 100, valid, external, best
      rx pathid: 0, tx pathid: 0x0
```

```
R5#show bgp ipv4 unicast 8.0.0.0/28
BGP routing table entry for 8.0.0.0/28, version 38
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  30
    10.4.5.4 from 10.4.5.4 (30.0.0.4)
      Origin IGP, localpref 100, valid, external, best
      rx pathid: 0, tx pathid: 0x0
```

```
R5#show bgp ipv4 unicast 192.168.2.0/24
% Network not in table
```

```
R5#show bgp ipv4 unicast 192.168.8.0/24
% Network not in table
```

```
R2#show bgp ipv4 unicast 2.0.0.0/28
% Network not in table
```

```
R8#show bgp ipv4 unicast 8.0.0.0/28
% Network not in table
```

The IPv6 configuration is much less complicated since the routes on CSR2 and CSR8 were public already; these were advertised directly into BGP and are learned by CSR7 as expected.

```
R7#show bgp ipv6 unicast 2000::/16
BGP routing table entry for 2000::/16, version 1038
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  30 2
    FD00:10:7:12::12 (FE80::12) from FD00:10:7:12::12 (30.0.0.12)
      Origin IGP, localpref 100, valid, external, best
      rx pathid: 0, tx pathid: 0x0
```

```
R7#show bgp ipv6 unicast 8000::/16
BGP routing table entry for 8000::/16, version 1039
Paths: (1 available, best #1, table default)
  Not advertised to any peer
```


Refresh Epoch 1

30 8

FD00:10:7:12::12 (FE80::12) from FD00:10:7:12::12 (30.0.0.12)

Origin IGP, localpref 100, valid, external, best

rx pathid: 0, tx pathid: 0x0

To test connectivity, CSR2 and CSR8 both attempt to ping the IPv4 Internet from their loopbacks. The pings succeeds (not shown) and CSR6 creates NAT state for each. Since overloading is not used, outside-to-inside traffic is also permitted when using an ACL with NAT pool; this is just an observation and is not relevant for this test. The post-NAT IPv4 source addresses for CSR2 and CSR8 are 2.0.0.0 and 8.0.0.0, respectively.

R6#show ip nat translations

Pro	Inside global	Inside local	Outside local	Outside global
---	2.0.0.0	192.168.2.1	---	---
---	8.0.0.0	192.168.8.1	---	---
icmp	2.0.0.0:6	192.168.2.1:6	55.46.253.193:6	55.46.253.193:6
icmp	8.0.0.0:5	192.168.8.1:5	55.46.253.193:5	55.46.253.193:5

Because this IPv4 Internet access is provided via a central services architecture, the MPLS label stack is identical for VRFs 2 and 8 as the exact same VPN route is used. Rather than trace this basic LSP, we will end the forward-LSP verification here.

R6#show ip cef vrf 2 55.46.253.193

0.0.0.0/0

nexthop 30.3.6.3 GigabitEthernet2.536 label 3001 4009

R6#show ip cef vrf 8 55.46.253.193

0.0.0.0/0

nexthop 30.3.6.3 GigabitEthernet2.536 label 3001 4009

Returning traffic is very similar except there are 2 separate VPN routes on CSR4 which originated in different VRFs (one per NAT pool). The transport label is the same but the VPN label differs in this case. Nonetheless, this is basic L3VPN and does not warrant additional verification.

R4#show ip cef vrf 5 2.0.0.0/28

2.0.0.0/28

nexthop 30.3.4.3 GigabitEthernet2.534 label 3000 6013

R4#show ip cef vrf 5 8.0.0.0/28

8.0.0.0/28

nexthop 30.3.4.3 GigabitEthernet2.534 label 3000 6020

What is interesting is how a command like “ip nat outside” can be honored when traffic entering CSR6 from CSR3 is MPLS encapsulated. The reason this configuration works is because CSR6 locally generated the static null routes for 2.0.0.0/28 and 8.0.0.0/28. This means that the label action on CSR6 will be to

strip all labels and perform an IPv4 VRF-aware FIB lookup on the destination. As seen above, labels 6013 and 6020 were allocated by CSR6 for the NAT pools.

```
R6#show mpls forwarding-table labels 6013
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
6013	No Label	2.0.0.0/28 [V]	1770	aggregate/2	

```
R6#show mpls forwarding-table labels 6020
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
6020	No Label	8.0.0.0/28 [V]	1180	aggregate/8	

However, performing a VRF-aware routing lookup on 2.0.0.0/28 or 8.0.0.0/28 will drop traffic as these are connected to null0. Because outside NAT happens before routing, the traffic is un-NAT'ed first and then the routing lookup occurs. The key is that the order of operations still applies with respect to MPLS label removal, outside NAT, and routing. After the label is removed, the order of operations "restarts" for IP traffic. After un-NAT, traffic is forwarded back towards the proper customer.

```
R6#show ip cef vrf 2 2.0.0.0/28
```

```
2.0.0.0/28
```

```
attached to Null0
```

```
R6#show ip cef vrf 8 8.0.0.0/28
```

```
8.0.0.0/28
```

```
attached to Null0
```

IPv6 Internet connectivity in this design is uninteresting as it is a basic L3VPN central services design with no complexity. A quick traceroute within the customer VRFs confirms this works properly.

```
R2#traceroute ipv6
```

```
Target IPv6 address: ::77:BABE:2
```

```
Source address: 2000::2
```

```
[snip]
```

```
Tracing the route to ::77:BABE:2
```

```
 1 FD00:10:2:6::6 [AS 30] 4 msec 4 msec 4 msec
 2 ::FFFF:30.3.6.3 [AS 30] [MPLS: Labels 3003/92008 Exp 0] 6 msec 8 msec 8 msec
 3 FD00:10:7:12::12 [AS 30] [MPLS: Label 92008 Exp 0] 23 msec 22 msec 22 msec
 4 FD00:10:7:12::7 [AS 30] 23 msec 15 msec 15 msec
```

```
R8#traceroute ipv6
```

```
Target IPv6 address: ::77:BABE:2
```

```
Source address: 8000::8
```

```
[snip]
```

```
Tracing the route to ::77:BABE:2
```

```

1 FD00:10:6:8::6 [AS 30] 4 msec 4 msec 3 msec
2 ::FFFF:30.3.6.3 [AS 30] [MPLS: Labels 3003/92008 Exp 0] 7 msec 7 msec 9
msec
3 FD00:10:7:12::12 [AS 30] [MPLS: Label 92008 Exp 0] 23 msec 22 msec 22
msec
4 FD00:10:7:12::7 [AS 30] 23 msec 15 msec 15 msec

```

We will examine another set of Internet connectivity methods on CSR1. For IPv4, CSR6 will leak the default IPv4 route from global into the VRF. CSR1 has no requirement for NAT44 as it only supports a few users, so using direct public addressing is acceptable. We will assume that CSR1 is an older router that cannot support IPv6 BGP, so instead uses a static default route. The prefix-list filter is applied as a best practice to ensure CSR1 does not become overloaded with IPv4 Internet routes accidentally. No RTs are imported into VRF 1 for IPv4, so this accidental advertisement is currently impossible. Since CSR6 cannot learn CSR1's public IPv6 routes, we configure a VRF-aware static route and redistribute it into BGP.

```

! CSR6
route-map RM_VRF_1_IMPORT_BGP permit 10
  match ip address prefix-list PL_DEFAULT

vrf definition 1
  rd 6:1
  address-family ipv4
    import ipv4 unicast map RM_VRF_1_IMPORT_BGP
    route-target export 6:1

  address-family ipv6
    route-target export 6:1
    route-target import 12:7

ipv6 route vrf 1 1000::/16 GigabitEthernet2.516 FE80::1 tag 1

route-map RM_STATIC_TO_BGP_VRF_1 permit 10
  match tag 1

router bgp 30
  address-family ipv4 vrf 1
    neighbor 10.1.6.1 remote-as 1
    neighbor 10.1.6.1 activate
    neighbor 10.1.6.1 prefix-list PL_DEFAULT out

  address-family ipv6 vrf 1
    redistribute static route-map RM_STATIC_TO_BGP_VRF_1

```

The most interesting part of this configuration is the global-to-VRF route leaking. CSR6 learns the route inside the default table natively. Once it is exported from global (imported to VRF), the route is marked "af-export" with the VRF name of the target VRF. The newly-created VPN route has no RTs or MPLS

labels, but does specify a global next-hop. The reason labels are not required is because the MPLS core is IPv4-capable, so if the penultimate hop removes the transport label and exposes the IPv4 header to the egress PE, the egress PE can route it correctly.

```
R6#show bgp ipv4 unicast 0.0.0.0
BGP routing table entry for 0.0.0.0/0, version 25
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 2
  Local
    30.0.0.4 (metric 20) from 30.0.0.4 (30.0.0.4)
      Origin incomplete, metric 0, localpref 100, valid, internal, af-
export(1), best
      rx pathid: 0, tx pathid: 0x0

R6#show bgp vpnv4 unicast vrf 1 0.0.0.0/0
BGP routing table entry for 6:1:0.0.0.0/0, version 3
Paths: (1 available, best #1, table 1)
  Advertised to update-groups:
    12
  Refresh Epoch 2
  Local, imported path from 0.0.0.0/0 (global)
    30.0.0.4 (metric 20) (via default) from 30.0.0.4 (30.0.0.4)
      Origin incomplete, metric 0, localpref 100, valid, internal, no-import,
no-import, best
      rx pathid: 0, tx pathid: 0x0
```

The configuration on CSR1 is very basic. The loopback prefix of 1.0.0.0/28 is advertised into BGP so CSR6 can learn it. A default IPv6 route is also configured which complements the specific 1000::/16 static route added onto CSR6.

```
! CSR1
router bgp 1
  no bgp default ipv4-unicast
  neighbor 10.1.6.6 remote-as 30

address-family ipv4
  network 1.0.0.0 mask 255.255.255.240
  neighbor 10.1.6.6 activate

ipv6 route ::/0 GigabitEthernet2.516 FE80::6
```

We quickly confirm that CSR6 learns 1.0.0.0/28 from CSR1 and CSR1 learns the default route from CSR6.

```
R6#show bgp vpnv4 unicast vrf 1 1.0.0.0/28
BGP routing table entry for 6:1:1.0.0.0/28, version 4
Paths: (1 available, best #1, table 1)
```

```

Advertised to update-groups:
  15
Refresh Epoch 1
1
10.1.6.1 (via vrf 1) from 10.1.6.1 (1.0.0.1)
  Origin IGP, metric 0, localpref 100, valid, external, best
  Extended Community: RT:6:1
  mpls labels in/out 6011/nolabel
  rx pathid: 0, tx pathid: 0x0

```

```

R1#show bgp ipv4 unicast 0.0.0.0/0
BGP routing table entry for 0.0.0.0/0, version 8
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
30
10.1.6.6 from 10.1.6.6 (30.0.0.6)
  Origin incomplete, localpref 100, valid, external, best
  rx pathid: 0, tx pathid: 0x0

```

Because no VPN label was allocated for the default route when it was leaked into the VRF, only one label is imposed. This is the LDP label from CSR3 to describe the path to 30.0.0.4/32.

```

R6#show ip cef vrf 1 55.46.253.1 detail
0.0.0.0/0, epoch 0, flags [rib only nolabel, rib defined all labels, default
route]
recursive via 30.0.0.4
  nexthop 30.3.6.3 GigabitEthernet2.536 label 3001

```

```

R6#show mpls ldp bindings 30.0.0.4 32 neighbor 30.0.0.3
lib entry: 30.0.0.4/32, rev 9
  remote binding: lsr: 30.0.0.3:0, label: 3001

```

When CSR3 pops this label, the IPv4 packet is exposed to CSR4. Since CSR4 leaked the Internet route from VRF to global, it has a route for the final destination in the global table. The next-hop is in VRF 5, but connectivity still works. Only a single MPLS label is required to tunnel the traffic through the P routers as there is no need to bind VPN labels for IPv4 prefixes in the global table. The exception was 6PE, which is required when the core is not IPv6-capable (verified on CSR9 earlier).

```

R3#show mpls forwarding-table labels 3001
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
3001      Pop Label  30.0.0.4/32     8870220     Gi2.534   30.3.4.4

```

```

R4#show ip route 55.46.253.1
Routing entry for 55.46.253.0/26

```

```

Known via "bgp 30", distance 20, metric 0
Tag 5, type external
Last update from 10.4.5.5 3d14h ago
Routing Descriptor Blocks:
* 10.4.5.5 (5), from 10.4.5.5, 3d14h ago
  Route metric is 0, traffic share count is 1
  AS Hops 1
  Route tag 5
  MPLS label: none

```

Traffic in the reverse direction uses basic L3VPN for transport. Since CSR6 learns 1.0.0.0/28 via BGP, label 6011 actually forwards traffic directly to CSR1 inside the VPN. This is different than the static null routes for 2.0.0.0/8 and 8.0.0.0/28 observed earlier for NAT support.

```

R4#show ip cef vrf 5 1.0.0.0/28
1.0.0.0/28
  nexthop 30.3.4.3 GigabitEthernet2.534 label 3000 6011

```

```

R6#show mpls forwarding-table labels 6011 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
6011      No Label  1.0.0.0/28[V]   1180         Gi2.516   10.1.6.1
          MAC/Encaps=18/18, MRU=1504, Label Stack{}
          005056A91AAA005056A9DE0D81000DBC0800
          VPN route: 1
          No output feature configured

```

For IPv6, the default route was imported into VRF 1 using the normal RT-import process. This is why label 92008, the original VPNv6 label allocated by XRv2, is used. This is in contrast to XRv1 where label 92009 was used (IPv6 labeled-unicast) since route leaking was used at the IPv6 Internet peering point. Returning traffic in the reverse direction is also using a basic L3VPN forwarding scheme.

```

R6#show ipv6 cef vrf 1 ::/0
::/0
  nexthop 30.3.6.3 GigabitEthernet2.536 label 3003 92008

```

```

RP/0/0/CPU0:Xshow cef vrf 7 ipv6 1000::/16
1000::/16, version 60, internal 0x5000001 0x0 (ptr 0xa13faa74) [1], 0x0
(0x0), 0x208 (0xa156d3c0)
Prefix Len 16, traffic index 0, precedence n/a, priority 3
via ::ffff:30.0.0.6, 3 dependencies, recursive [flags 0x6000]
path-idx 0 NHID 0x0 [0xa18350bc 0x0]
recursion-via-/128
next hop VRF - 'default', table - 0xe0000000
next hop ::ffff:30.0.0.6 via ::ffff:30.0.0.6:0
next hop 30.3.12.3/32 Gi0/0/0/0.532 labels imposed {3000 6022}

```

We verify connectivity using IPv4 ping and IPv6 traceroute for brevity.

```
R1#ping 55.46.253.1 source 1.0.0.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 55.46.253.1, timeout is 2 seconds:
Packet sent with a source address of 1.0.0.1
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 7/7/9 ms
```

```
R1#traceroute ipv6
Target IPv6 address: ::77:BABE:0
Source address: 1000::1
[snip]
Tracing the route to ::77:BABE:0
 0  FE80::6 11 msec 3 msec 4 msec
 1  ::FFFF:30.3.6.3 [MPLS: Labels 3003/92008 Exp 0] 7 msec 5 msec 11 msec
 2  FD00:10:7:12::12 [MPLS: Label 92008 Exp 0] 23 msec 22 msec 22 msec
 3  FD00:10:7:12::7 23 msec 15 msec 15 msec
```

CSR8 can be multi-homed to XRv1 as well. Normally this would call for providing the full routing table to CSR8, but an alternative approach is to use simply BGP manipulations to prefer one default gateway over the other. We will examine IPv6 first, saving the final NT44 option for later. XRv1 will simply provide a default route, just like CSR6. By importing RT:12:7, this will match the default route and nothing else. We confirm that only the IPv6 default route is imported before continuing.

```
! XRv1
vrf 8
 address-family ipv6 unicast
   import route-target
     12:7
   export route-target
     11:8

router bgp 30
vrf 8
 rd 11:8
 address-family ipv6 unicast

 neighbor fd00:10:8:11::8
   remote-as 8
   address-family ipv6 unicast
     route-policy RPL_PASS in
     route-policy RPL_PASS out
```

```
RP/0/0/CPU0:XRv1#show bgp vpnv6 unicast vrf 8 | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 11:8 (default for vrf 8)
*>i::/0            30.0.0.12            0      100      0 ?
```

Next, we configure CSR8 for BGP IPv6 unicast. The configuration is very basic, and to ensure XRv1 is used, we adjust the weight for that peer. Otherwise, CSR8 will prefer the older default route when comparing CSR6 and XRv1. The other BGP parameters, such as the network statements, are already configured on CSR8 from earlier. I add a lower weight to CSR6 just for clarity, although it is not required. This will make configuration adjustments easier later. We confirm that CSR8 learns both IPv6 default routes and prefers XRv1 over CSR6 due to the BGP weight attribute.

```
! CSR8
router bgp 8
 neighbor FD00:10:8:11::11 remote-as 30
 address-family ipv6
  neighbor FD00:10:6:8::6 weight 66
  neighbor FD00:10:8:11::11 activate
  neighbor FD00:10:8:11::11 weight 1111

R8#show bgp ipv6 unicast ::/0
BGP routing table entry for ::/0, version 249
Paths: (2 available, best #1, table default)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  30
    FD00:10:8:11::11 (FE80::11) from FD00:10:8:11::11 (30.0.0.11)
      Origin incomplete, localpref 100, weight 1111, valid, external, best
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 2
  30
    FD00:10:6:8::6 (FE80::6) from FD00:10:6:8::6 (30.0.0.6)
      Origin incomplete, localpref 100, weight 66, valid, external
      rx pathid: 0, tx pathid: 0
```

We confirm that XRv1 learns CSR8's public IPv6 prefix 8000::/16. Because the exported RT is something that XRv2 is already importing, we assume that this route is being advertised to the IPv6 Internet. Again, this is basic L3VPN so traceroute is used to speed up the verification.

```
RP/0/0/CPU0:XRv1#show bgp vrf 8 ipv6 unicast 8000::/16 | begin " 8$"
 8
  fd00:10:8:11::8 from fd00:10:8:11::8 (192.168.8.1)
    Origin IGP, metric 0, localpref 100, valid, external, best, group-best,
import-candidate
    Received Path ID 0, Local Path ID 1, version 64
    Extended community: RT:11:8

R8#traceroute ipv6
Target IPv6 address: ::77:BABE:2
Source address: 8000::8
```



```
[snip]
Tracing the route to ::77:BABE:2
 1 FD00:10:8:11::11 [AS 30] 3 msec 2 msec 2 msec
 2 ::FFFF:30.3.11.3 [AS 30] [MPLS: Labels 3003/92008 Exp 0] 7 msec 6 msec 10
msec
 3 FD00:10:7:12::12 [AS 30] [MPLS: Label 92008 Exp 0] 30 msec 26 msec 22
msec
 4 FD00:10:7:12::7 [AS 30] 23 msec 15 msec 15 msec
```

We progress to CSR10's IPv6 connectivity. Similar to CSR9, XRv1 will leak the default IPv6 route from the global table. The difference is that CSR10 is an old router, like CSR1, and is incapable of running BGP IPv6. It uses a static default route to reach the IPv6 Internet. XRv1 configures a VRF-aware static route to CSR1's public IPv6 prefix A000::/16. Some RPL infrastructure is re-used to selectively redistribute this into BGP.

```
! XRv1
prefix-set PS_10
  a000::/16
end-set

vrf 10
  address-family ipv6 unicast
    import from default-vrf route-policy RPL_IF_DEST_PASS(PS_DEFAULT)
    export route-target
      11:10

router bgp 30
  vrf 10
    rd 11:10
    address-family ipv6 unicast
      redistribute static route-policy RPL_IF_DEST_PASS(PS_10)

! CSR10
ipv6 route ::/0 GigabitEthernet2.510 FE80::1
```

Because the prefix A000::/16 was locally originated into BGP but was not a null aggregate (that is, it has a valid next-hop), a prefix-specific VPNv6 label is allocated. The export RT of 11:10 is imported by XRv2 so it is assumed this prefix can be reached from the IPv6 Internet.

```
RP/0/0/CPU0:XRv1#show bgp vpnv6 unicast vrf 10 a000::/16
BGP routing table entry for a000::/16, Route Distinguisher: 11:10
Versions:
  Process          bRIB/RIB   SendTblVer
  Speaker          20         20
  Local Label: 91008
Paths: (1 available, best #1)
  Advertised to peers (in unique update groups):
```

```

30.0.0.12
Path #1: Received by speaker 0
Advertised to peers (in unique update groups):
30.0.0.12
Local
fe80::10 from 0.0.0.0 (30.0.0.11)
Origin incomplete, metric 0, localpref 100, weight 32768, valid,
redistributed, best, group-best, import-candidate
Received Path ID 0, Local Path ID 1, version 20
Extended community: RT:11:10

```

As a quick review, the IPv6 labeled-unicast label of 92009 is used since the core is not IPv6-aware. This was allocated by XRv2 since traffic received from the core will use this label and the LFIB lookup occurs in the global table. When XRv2 receives traffic along this LSP, label stack is removed and traffic is forwarded to CSR7.

```

RP/0/0/CPU0:XRv1#show cef vrf 10 ipv6 ::/0
::/0, version 6, proxy default, internal 0x5000011 0x0 (ptr 0xa13f9d74) [1],
0x0 (0x0), 0x208 (0xa156d460)
Prefix Len 0, traffic index 0, precedence n/a, priority 3
via ::ffff:30.0.0.12, 9 dependencies, recursive [flags 0x6000]
path-idx 0 NHID 0x0 [0xa1801050 0x0]
recursion-via-/128
next hop VRF - 'default', table - 0xe0000000
next hop ::ffff:30.0.0.12 via ::ffff:30.0.0.12:0
next hop 30.3.11.3/32 Gi0/0/0/0.531 labels imposed {3003 92009}

RP/0/0/CPU0:XRv2#show mpls forwarding | include 92009
92009 Unlabelled ::/0 Gi0/0/0/0.572 fe80::7 321200

```

Using traceroute, we can verify IPv6 Internet connectivity. Label 92009 is technically an IPv6 labeled-unicast label from XRv2, but on XRv1, it is used as a VPNv6 label at imposition. The label carries over when the prefix is imported from default to VRF. IPv6 Internet connectivity is significantly more straightforward, even with route leaking, than IPv4 as there is seldom a need for NAT.

```

R10#traceroute ipv6
Target IPv6 address: ::77:BABE:3
Source address: A000::A
[snip]
Tracing the route to ::77:BABE:3
 1 ::FFFF:30.3.11.11 4 msec 3 msec 2 msec
 2 ::FFFF:30.3.11.3 [MPLS: Labels 3003/92009 Exp 0] 8 msec 8 msec 8 msec
 3 FD00:10:7:12::12 [MPLS: Label 92009 Exp 0] 23 msec 22 msec 22 msec
 4 FD00:10:7:12::7 23 msec 15 msec 15 msec

```

This completes IPv6 Internet connectivity for all CE routers. CSR8's link via XRv1 and CSR10 have not yet been enabled for IPv4 Internet connectivity. There is a third option that this document has not yet

explored. The first option was local NAT on the CE device as seen on CSR9. The second option was NAT on the ingress PE as seen on CSR6 supporting CSR2 and CSR8. The third option is NAT on the egress PE towards the IPv4 Internet. This might be used when there are many ingress PEs and a small set of very powerful IPv4 Internet peering routers. The NAT44 can be centralized on these routers rather than managed on the ingress PEs. The drawback is that increase in processing power as a result of the consolidating NAT44 processing on a smaller set of devices. First, the VRF RT import/export policies are defined. There is no route-leaking with this solution to keep the focus on NAT44.

```
! XRv1
vrf 8
  address-family ipv4 unicast
    import route-target
      4:5
    export route-target
      11:8

vrf 10
  address-family ipv4 unicast
    import route-target
      4:5
    export route-target
      11:10
```

Next, BGP is configured between the CEs (CSR10 and CSR8) and XRv1. Weight is used on CSR8 to prefer XRv1 for testing, just as it was in IPv6. The BGP configuration is very simple as there is no route leaking or advanced filtering. The most interesting part of this configuration is that CSR8 and CSR10 advertise their private LAN routes to XRv1, and XRv1 makes no effort to filter them. This is because the routes must be advertised all the way to XRv2, the NAT44 point.

```
! CSR8
router bgp 8
  neighbor 10.8.11.11 remote-as 30
  address-family ipv4
    network 192.168.8.0
    neighbor 10.6.8.6 weight 66
    neighbor 10.8.11.11 activate
    neighbor 10.8.11.11 weight 1111

! CSR10
router bgp 10
  no bgp default ipv4-unicast
  neighbor 10.10.11.11 remote-as 30
  address-family ipv4
    network 192.168.10.0
    neighbor 10.10.11.11 activate

! XRv1
```

```

router bgp 30
vrf 8
address-family ipv4 unicast
neighbor 10.8.11.8
remote-as 8
address-family ipv4 unicast
route-policy RPL_PASS in
route-policy RPL_PASS out

```

```

router bgp 30
vrf 10
address-family ipv4 unicast
neighbor 10.10.11.10
remote-as 10
address-family ipv4 unicast
route-policy RPL_PASS in
route-policy RPL_PASS out

```

Before continuing, we quickly verify that XRv1 learns the routes from both CEs. We also verify that both CEs learn a default from XRv1. CSR8 learns the same route from CSR6 and XRv1, but prefers XRv1 due to the BGP weight configuration.

```

RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast 192.168.8.0/22 longer-prefixes |
begin Network

```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 11:8 (default for vrf 8)					
*> 192.168.8.0/24	10.8.11.8	0		0 8	i
Route Distinguisher: 11:10 (default for vrf 10)					
*> 192.168.10.0/24	10.10.11.10	0		0 10	i

```

R8#show bgp ipv4 unicast 0.0.0.0/0
BGP routing table entry for 0.0.0.0/0, version 116
Paths: (2 available, best #1, table default)
  Advertised to update-groups:
    2
  Refresh Epoch 1
  30
    10.8.11.11 from 10.8.11.11 (30.0.0.11)
      Origin incomplete, localpref 100, weight 1111, valid, external, best
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 2
  30
    10.6.8.6 from 10.6.8.6 (30.0.0.6)
      Origin incomplete, localpref 100, weight 66, valid, external
      rx pathid: 0, tx pathid: 0

```

```

R10#show bgp ipv4 unicast 0.0.0.0/0
BGP routing table entry for 0.0.0.0/0, version 2

```

```

Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  30
    10.10.11.11 from 10.10.11.11 (30.0.0.11)
      Origin incomplete, localpref 100, valid, external, best
      rx pathid: 0, tx pathid: 0x0

```

To protect the IPv4 Internet from learning these private LAN routes, a basic prefix filter is applied outbound on CSR4. This denies all private and IPv4 link-local prefixes, as well as the default route. The default route prefix-list is reused from before and contained in the route-map match clause using Boolean “or” logic. A quick check on CSR5 ensures that these private routes are not learned.

```

! CSR4
ip prefix-list PL_PRIVATE seq 5 permit 10.0.0.0/8 le 32
ip prefix-list PL_PRIVATE seq 10 permit 172.16.0.0/12 le 32
ip prefix-list PL_PRIVATE seq 15 permit 192.168.0.0/16 le 32
ip prefix-list PL_PRIVATE seq 20 permit 169.254.0.0/16 le 32

route-map RM_INTERNET_OUT deny 10
  match ip address prefix-list PL_DEFAULT PL_PRIVATE
route-map RM_INTERNET_OUT permit 100

router bgp 30
  address-family ipv4 vrf 5
    neighbor 10.4.5.5 route-map RM_INTERNET_OUT out

```

```

R5#show bgp ipv4 unicast 192.168.8.0/22 longer-prefixes
[no output]

```

Next, we configure NAT44 on CSR4. Unlike the other NAT44 configurations, we use port overloading on CSR4. This is not specific to running NAT on the egress PE; I introduce it here for variety only. The NAT pools contain a single route and there are corresponding VRF-aware static routes per pool. Although the NAT interfaces are in different VRFs, the router seamlessly moves packets across as necessary. CSR4 creates ACLs to match the private LANs on both CSR8 and CSR10 which are used for the inside NAT rules. These rules are also VRF-aware.

```

! CSR4
interface GigabitEthernet2.534
  ip nat inside

interface GigabitEthernet2.545
  ip nat outside

ip nat pool NAT_POOL_8 4.8.0.0 4.8.0.0 prefix-length 30
ip nat pool NAT_POOL_10 4.10.0.0 4.10.0.0 prefix-length 30

```

```

ip route vrf 5 4.8.0.0 255.255.255.255 Null0
ip route vrf 5 4.10.0.0 255.255.255.255 Null0

ip access-list standard ACL_NAT_10
 permit 192.168.10.0 0.0.0.255
ip access-list standard ACL_NAT_8
 permit 192.168.8.0 0.0.0.255

ip nat inside source list ACL_NAT_10 pool NAT_POOL_10 vrf 5 overload
ip nat inside source list ACL_NAT_8 pool NAT_POOL_8 vrf 5 overload

```

The BGP configuration involves redistributing these static routes. Since there is a VRF-aware default route already being redistributed into BGP, we technically don't need to configure anything new. The configuration is shown again for reference, but this command was already configured to advertise the VRF-aware default into BGP. We quickly confirm that these NAT pool prefixes were advertised into BGP properly.

```

! CSR4, reference only
address-family ipv4 vrf 5
 redistribute static

```

```

R4#show bgp vpnv4 unicast vrf 5 4.8.0.0/14 longer-prefixes | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 4:5 (default for vrf 5)
Export Map: RM_VRF_5_TO_GLOBAL, Address-Family: IPv4 Unicast, Pfx
Count/Limit: 5/1000
*> 4.8.0.0/32          0.0.0.0          0          32768 ?
*> 4.10.0.0/32         0.0.0.0          0          32768 ?

```

Because CSR4's export RT policy for VRF 5 adds a bogus RT by default, these NAT pool routes carry that RT. No router imports this RT anywhere in the network and it was only required to force route leaking to work. As such, no fancy filtering is required to prevent the CEs from learning these NAT pool routes. XRv1 is unable to import either route as shown below.

```

R4#show bgp vpnv4 unicast vrf 5 4.8.0.0/32 | include RT:
      Extended Community: RT:4:999

```

```

R4#show bgp vpnv4 unicast vrf 5 4.10.0.0/32 | include RT:
      Extended Community: RT:4:999

```

```

RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast vrf 8 4.8.0.0/32
% Network not in table

```

```

RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast vrf 10 4.10.0.0/32
% Network not in table

```

Of greater importance is ensuring the IPv4 Internet (CSR5) learns these routes. We quickly confirm that below. This completes the control-plane verification as all of the routing information is correct.

```
R5#show bgp ipv4 unicast 4.8.0.0/14 longer-prefixes | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
*>  4.8.0.0/32      10.4.5.4          0           0 30 ?
*>  4.10.0.0/32     10.4.5.4          0           0 30 ?
```

When traffic arrives at XRv1 from the customers, it follows the exact same LSP since both paths recursive to the default route. The transport LSP using label 3001 has been traced multiple times already, so that part is skipped.

```
RP/0/0/CPU0:XRv1#show cef vrf 8 55.46.253.193 | include labels
      next hop 30.3.11.3/32 Gi0/0/0/0.531 labels imposed {3001 4009}
```

```
RP/0/0/CPU0:XRv1#show cef vrf 10 55.46.253.193 | include labels
      next hop 30.3.11.3/32 Gi0/0/0/0.531 labels imposed {3001 4009}
```

When CSR4 receives label 4009, it is directed to perform an ordinary IPv4 FIB lookup in VRF 5. Because the IPv4 packet is revealed to CSR4, it can perform inside NAT on this traffic. The MPLS label identified the proper VRF which will specify which NAT rule to use.

```
R4#show mpls forwarding-table labels 4009
Local      Outgoing   Prefix      Bytes Label  Outgoing   Next Hop
Label      Label      or Tunnel Id Switched     interface
4009       No Label   0.0.0.0/0[V] 6490         aggregate/5
```

```
R4#show ip cef vrf 5 55.46.253.193
55.46.253.192/26
      nexthop 10.4.5.5 GigabitEthernet2.545
```

Because NAT overload is used, there are no outside-to-inside mappings created. The post-NAT source addresses are 4.8.0.0 and 4.10.0.0, which are the only IPv4 addresses available in each pool. Since CSR4 learns the original private addresses (inside locals), those are identified so that the return flow can be un-NAT'ed.

```
R4#show ip nat translations
Pro Inside global   Inside local   Outside local   Outside global
icmp 4.8.0.0:1      192.168.8.1:6  55.46.253.193:6 55.46.253.193:1
icmp 4.10.0.0:1     192.168.10.1:5 55.46.253.193:5 55.46.253.193:1
Total number of translations: 2
```

Because outside NAT happens before routing, the reply traffic destination to 4.8.0.0 or 4.10.0.0 is translated back into the correct inside local address as identified in the NAT translation table. Since the

return path is regular L3VPN, two labels are imposed, and the VPN label differs for each as these are unique, non-aggregate prefixes. This completes the return path.

```
R4#show ip cef vrf 5 192.168.8.0
192.168.8.0/24
  nexthop 30.3.4.3 GigabitEthernet2.534 label 3002 91009

R4#show ip cef vrf 5 192.168.10.0
192.168.10.0/24
  nexthop 30.3.4.3 GigabitEthernet2.534 label 3002 91004
```

Additional Reading – Reference configurations "inet-access"

37. Describe, implement, and troubleshoot end-to-end fast convergence

37.1 Loop Free Alternate (LFA) for IPv4

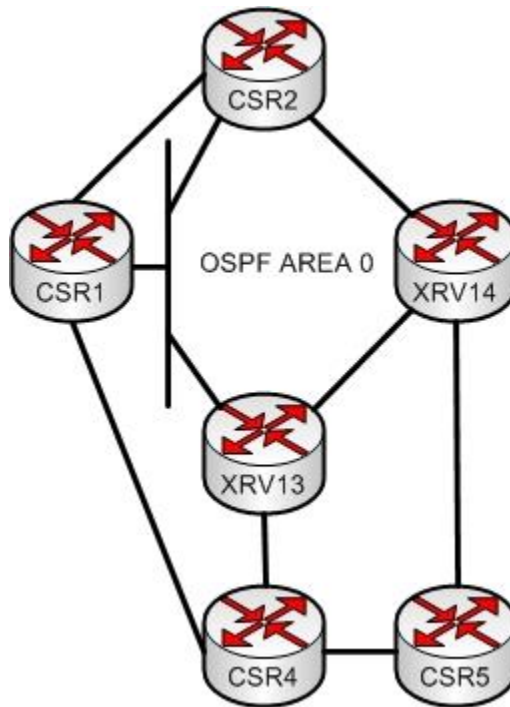
IPv4 LFA is supported for OSPFv2, IS-IS, and EIGRP. The purpose of this feature is to create a loop-free path from a given node by running SPF using other routers as the root of the tree (not including EIGRP, which is discussed in a dedicated section). Specifically, a router with LFA configuration only runs SPF from the perspective of its neighbors, not every router in the area/level, as the local router is just trying to find a backup next-hop towards a specific destination. Because all of the LFA testing was all done using the same topology, the configurations are provided once. To move through the demonstrations, all you need to do is shutdown or enable certain interfaces, change administrative distances, enable/disable LFA where desired, tune metrics, etc. All of the IGP's for both IPv4 and IPv6 have been pre-built.

Additional Reading (includes all direct and IPv4 rLFA examples) – Reference configurations "lfa"

37.1.1 OSPFv2

37.1.1.1 Direct LFA

The network diagram is shown below. Many diverse IGP paths are used so that testing the different LFA variables is made easy.



The basic LFA configuration is shown below, beginning with the XE routers. LFA is enabled on a per-prefix basis, which is the only option supported on XE (XR also supports per-link, which is discussed later). The concept of prefix-priority is related to the order in which prefixes are updated in the RIB/FIB after SPF is finished. Higher priorities are processed first, and generally these are loopbacks (LSP endpoints) or IPTV SSM sources. A route-map can be applied to select prefixes for high-priority, but the default is /32 prefixes. Prefix-priority is not discussed in detail as it is very straightforward.

```
! All XE routers
router ospf 1
  fast-reroute per-prefix enable area 0 prefix-priority high
```

A quick summary on CSR1 shows the default tiebreaker policy for backup paths. The tiebreakers are used to select the best backup when multiple backups exist. These are examined in great detail.

```
CSR1#show ip ospf fast-reroute
      OSPF Router with ID (1.1.1.1) (Process ID 1)
Loop-free Fast Reroute protected prefixes:
      Area      Topology name  Priority  Remote LFA Enabled
      0          Base         High      No
Repair path selection policy tiebreaks (built-in default policy):
 10  srlg
 20  primary-path
 30  interface-disjoint
 40  lowest-metric
 50  linecard-disjoint
 60  node-protecting
```

```
70 broadcast-interface-disjoint
256 load-sharing
```

To simplify the output initially, I have increased the cost of CSR1's LAN segment and P2P CSR4 link to 5. This gives us a single best-path through P2P CSR2 connection, which is a good starting point. I have configured all of the CSRs to keep all backup paths, even ignored ones, as this will make troubleshooting easier. The command consumes more memory with little operational gain other than more information seen to the administrator. It is very useful for troubleshooting as we will see soon.

```
! CSR1
interface GigabitEthernet2.513
 ip ospf cost 5

interface GigabitEthernet2.514
 ip ospf cost 5

! All XE routers
router ospf 1
 fast-reroute keep-all-paths
```

Let's examine how CSR1 sees XRv14's loopback with IPv4 address 14.14.14.14/32. For readability, I am breaking the output of this command into sections and describing each section individually.

```
CSR1#show ip ospf rib 14.14.14.14 255.255.255.255
      OSPF Router with ID (1.1.1.1) (Process ID 1)
      Base Topology (MTID 0)
OSPF local RIB
Codes: * - Best, > - Installed in global RIB
LSA: type/LSID/originator
```

The first entry is the best path via the P2P link. The “RIB” flag without the “Repair” flag is an indication that this is the best path.

```
*> 14.14.14.14/32, Intra, cost 3, area 0
    SPF Instance 20, age 00:00:19
    Flags: RIB, HiPrio
    via 12.0.0.2, GigabitEthernet2.512
    Flags: RIB
    LSA: 1/14.14.14.14/14.14.14.14
```

This is the repair path via XRv13 over the LAN. When compared against the best path, it is a different interface (and by extension a different broadcast interface), is downstream to the destination (closer), and provides node protection since CSR2 is not in the repair path. These are all “good” things that generally make a repair path more desirable. Having repair paths rely on the same interfaces, same linecards, and same nodes may result in lower availability.

```
repair path via 123.0.0.13, GigabitEthernet2.513, cost 7
Flags: RIB, Repair, IntfDj, BcastDj, CostWon, NodeProt, Downstr
LSA: 1/14.14.14.14/14.14.14.14
```

This candidate path was ignored because of the cost tiebreaker. The path is also interface disjoint but has a higher cost than the path shown above.

```
repair path via 14.0.0.4, GigabitEthernet2.514, cost 8
Flags: Ignore, Repair, IntfDj, BcastDj, NodeProt
LSA: 1/14.14.14.14/14.14.14.14
```

This path is equal in many ways to the selected repair path, but was ignored due to not providing node protection. This is another link through CSR2, so LFA would rather use a different node in the transit path, all other more important things being equal. Node protection guarantees that if CSR2 fails completely, the LFA does not rely on CSR2 as the next-hop.

```
repair path via 123.0.0.2, GigabitEthernet2.513, cost 7
Flags: Ignore, Repair, IntfDj, BcastDj, Downstr
LSA: 1/14.14.14.14/14.14.14.14
```

Let's say we want to use CSR4 as the backup path. We can use the shared risk link group (SRLG) to identify the PSP CSR2 and LAN interfaces as sharing risk; that is, we don't want to re-route traffic between two links in the same SRLG because perhaps they use the same fiber pair or physical conduit. This is an administrative setting similar to the MPLS-TE SRLG and serves an identical purpose, except that it is local only (not carried in topology information).

```
! CSR1
interface GigabitEthernet2.512
  srlg gid 222

interface GigabitEthernet2.513
  srlg gid 222
```

The paths sharing the SRLG are marked in the OSPF RIB with the "SRLG" flag and are now ignored. Notice that CSR4 is technically not "downstream" from CSR1 towards the destination. This is because both nodes have a cost of 3 to reach 14.14.14.14/32, and like the EIGRP feasibility condition, the next-hop node must be strictly closer (lower metric, cannot be equal) to the destination. CSR4 is also higher cost than those LAN interfaces, but SRLG is the first and most important tiebreaker by default. This is the "trump card" that can administratively exclude certain links from LFA consideration. CSR4 is the highest cost backup path, but is also the most desirable.

```
CSR1#show ip ospf rib 14.14.14.14 255.255.255.255
[snip]
*> 14.14.14.14/32, Intra, cost 3, area 0
    SPF Instance 21, age 00:07:24
```

```

Flags: RIB, HiPrio
  via 12.0.0.2, GigabitEthernet2.512
    Flags: RIB
    LSA: 1/14.14.14.14/14.14.14.14
  repair path via 14.0.0.4, GigabitEthernet2.514, cost 8
    Flags: RIB, Repair, IntfDj, BcastDj, NodeProt
    LSA: 1/14.14.14.14/14.14.14.14
  repair path via 123.0.0.13, GigabitEthernet2.513, cost 7
    Flags: Ignore, Repair, IntfDj, BcastDj, SRLG, NodeProt, Downstr
    LSA: 1/14.14.14.14/14.14.14.14
  repair path via 123.0.0.2, GigabitEthernet2.513, cost 7
    Flags: Ignore, Repair, IntfDj, BcastDj, SRLG, Downstr
    LSA: 1/14.14.14.14/14.14.14.14

```

With the SRLGs in place, next we will change some OSPF costs. The cost on the CSR2 P2P link (on CSR1 side) has been set to 5 with other CSR1 link costs set to 1. The cost on CSR2 to XRv14 has been increased to 2, making CSR1 select the LAN interface via XRv13 to reach 14.14.14.14/32. I did not remove the SRLG, so the best-path is through an interface with SRLG set. This means that we don't really want to use the CSR2 P2P link as a backup. It also has a worse cost, so even without the SRLG, CSR2 P2P link would not be preferred.

```

! CSR1
interface GigabitEthernet2.512
  ip ospf cost 5

```

```

interface GigabitEthernet2.513
  ip ospf cost 1

```

```

interface GigabitEthernet2.514
  ip ospf cost 1

```

```

! CSR2
interface GigabitEthernet2.524
  ip ospf cost 2

```

```

CSR1#show ip ospf rib 14.14.14.14 255.255.255.255
[snip]
*> 14.14.14.14/32, Intra, cost 3, area 0
  SPF Instance 25, age 00:00:11
  Flags: RIB, HiPrio
    via 123.0.0.13, GigabitEthernet2.513
      Flags: RIB
      LSA: 1/14.14.14.14/14.14.14.14
    repair path via 14.0.0.4, GigabitEthernet2.514, cost 4
      Flags: RIB, Repair, IntfDj, BcastDj
      LSA: 1/14.14.14.14/14.14.14.14
    repair path via 12.0.0.2, GigabitEthernet2.512, cost 8
      Flags: Ignore, Repair, IntfDj, SRLG

```

```
LSA: 1/14.14.14.14/14.14.14.14
```

The prove the claim above, we remove the SRLGs; the end result in the same, but cost is the differentiator. Notice that the path through CSR4 is not considered node-protecting because CSR4 has equal cost paths through XRv13 and CSR5 to reach 14.14.14.14/32. That is, we cannot guarantee that if we fast-reroute traffic to CSR4 that CSR4 won't forward it through XRv13. CSR4's ECMP load-sharing mechanism could potentially forward traffic through XRv13 which means node-protection cannot be assumed. Node-protection was not analyzed for this LFA decision, but confirming the observation is good practice.

```
! CSR1
interface GigabitEthernet2.512
  no srlg gid 222

interface GigabitEthernet2.513
  no srlg gid 222

CSR1#show ip ospf rib 14.14.14.14 255.255.255.255
[snip]
*> 14.14.14.14/32, Intra, cost 3, area 0
    SPF Instance 26, age 00:00:49
    Flags: RIB, HiPrio
    via 123.0.0.13, GigabitEthernet2.513
    Flags: RIB
    LSA: 1/14.14.14.14/14.14.14.14
    repair path via 14.0.0.4, GigabitEthernet2.514, cost 4
    Flags: RIB, Repair, IntfDj, BcastDj, CostWon
    LSA: 1/14.14.14.14/14.14.14.14
    repair path via 12.0.0.2, GigabitEthernet2.512, cost 8
    Flags: Ignore, Repair, IntfDj
```

```
CSR4#show ip route 14.14.14.14
Routing entry for 14.14.14.14/32
  Known via "ospf 1", distance 110, metric 3, type intra area
  Last update from 45.0.0.5 on GigabitEthernet2.545, 00:29:25 ago
  Routing Descriptor Blocks:
    45.0.0.5, from 14.14.14.14, 00:29:25 ago, via GigabitEthernet2.545
      Route metric is 3, traffic share count is 1
      Repair Path: 34.0.0.13, via GigabitEthernet2.534
    * 34.0.0.13, from 14.14.14.14, 00:29:25 ago, via GigabitEthernet2.534
      Route metric is 3, traffic share count is 1
      Repair Path: 45.0.0.5, via GigabitEthernet2.545
```

We will attempt to make CSR1's LFA shown above "node-protecting". A quick cost adjustment on CSR4 (increase cost on link towards XRv13 to 2) shows the path above as being node-protected since CSR4 is only routing through CSR5. This doesn't change the LFA selection but is just an interesting note. Personally, even if node-protection is low on the tiebreaker list, I would try to design the network such

that node-protection was made available as often as possible. Now XRv13 could fail completely and it would have no affect on the backup path, which was not the case before the CSR4 cost adjustment.

```
! CSR4
interface GigabitEthernet2.534
 ip ospf cost 2

CSR4#show ip route 14.14.14.14
Routing entry for 14.14.14.14/32
  Known via "ospf 1", distance 110, metric 3, type intra area
  Last update from 45.0.0.5 on GigabitEthernet2.545, 00:30:10 ago
  Routing Descriptor Blocks:
  * 45.0.0.5, from 14.14.14.14, 00:30:10 ago, via GigabitEthernet2.545
    Route metric is 3, traffic share count is 1

CSR1#show ip ospf rib 14.14.14.14 255.255.255.255
[snip]
*> 14.14.14.14/32, Intra, cost 3, area 0
  SPF Instance 27, age 00:06:17
  Flags: RIB, HiPrio
  via 123.0.0.13, GigabitEthernet2.513
  Flags: RIB
  LSA: 1/14.14.14.14/14.14.14.14
  repair path via 14.0.0.4, GigabitEthernet2.514, cost 4
  Flags: RIB, Repair, IntfDj, BcastDj, CostWon, NodeProt
  LSA: 1/14.14.14.14/14.14.14.14
  repair path via 12.0.0.2, GigabitEthernet2.512, cost 8
  Flags: Ignore, Repair, IntfDj
  LSA: 1/14.14.14.14/14.14.14.14
```

One final example shows the best-path through XRv13 with the CSR4 P2P link disabled. CSR1 can select either link on CSR2, which is both downstream and node-protecting. LFA should select the P2P link versus the LAN interface as it interface-disjoint (disjoint meaning the same LAN interface was not re-used). To facilitate this, CSR2's cost on the LAN interface is increased to 2 because if CSR2 routes to XRv13 over that link, it is not loop free, and cannot be a backup path. The below output shows that the path through the CSR2 P2P link was selected based on "IntfDj". The path is also broadcast interface disjoint but that was not considered (this particular option is discussed later). Of note, notice that the cost through this preferred backup path is higher than the ignored one; this is because interface disjoint is processed before lowest metric by default.

```
! CSR1
interface GigabitEthernet2.514
 shutdown

! CSR2
interface GigabitEthernet2.513
```

```
ip ospf cost 2
```

```
CSR1#show ip ospf rib 14.14.14.14 255.255.255.255
[snip]
*> 14.14.14.14/32, Intra, cost 3, area 0
    SPF Instance 30, age 00:17:41
    Flags: RIB, HiPrio
    via 123.0.0.13, GigabitEthernet2.513
    Flags: RIB
    LSA: 1/14.14.14.14/14.14.14.14
    repair path via 12.0.0.2, GigabitEthernet2.512, cost 8
    Flags: RIB, Repair, IntfDj, BcastDj, NodeProt
    LSA: 1/14.14.14.14/14.14.14.14
    repair path via 123.0.0.2, GigabitEthernet2.513, cost 4
    Flags: Ignore, Repair, NodeProt
    LSA: 1/14.14.14.14/14.14.14.14
```

To modify the tiebreaker options, you can manually configure a new sequence. In this example, we re-apply an SRLG to CSR1 CSR P2P and LAN interfaces. We then tell OSPF that SRLG disjoint is required (not just preferred), as are downstream nodes. Notice that changing the selection process wipes out the default settings, so you cannot make a few changes and maintain the default policy. You'd need to redefine the entire selection process. CSR1 only cares about the SRLG-disjoint and downstream characteristics now; cost, for example, is not considered by LFA at all.

```
! CSR1
interface GigabitEthernet2.512
  srlg gid 45

interface GigabitEthernet2.513
  srlg gid 45

! CSR1
router ospf 1
  fast-reroute per-prefix tie-break srlg required index 10
  fast-reroute per-prefix tie-break downstream required index 20

CSR1#show ip ospf fast-reroute
[snip]
Repair path selection policy tiebreaks:
  10  srlg (required)
  20  downstream (required)
  256 load-sharing
```

All OSPF costs have been reset to 1 on CSR2. The link between CSR1 and CSR4 has been enabled again. CSR1's cost on the LAN interface has been set to 5, making the P2P CSR2 link the best path. However, notice how there is no backup path. The path through CSR2 and XRv13 on the LAN is not available because it shares an SRLG with the primary path (but one is downstream). The link through CSR4 is not

available because it is not downstream to the destination (both have a cost of 3), but passes the SRLG test. This demonstrates the danger of using the "required" keyword for LFA tiebreakers as the logic between required options is Boolean AND, not Boolean OR. All conditions must be true, but in our case we have 3 candidate repair paths that are ignored because no single path meets both criteria.

```
! CSR2
interface GigabitEthernet2.513
 ip ospf cost 1

! CSR1
interface GigabitEthernet2.514
 no shutdown

interface GigabitEthernet2.513
 ip ospf cost 5

interface GigabitEthernet2.512
 ip ospf cost 1

CSR1#show ip ospf rib 14.14.14.14 255.255.255.255
[snip]
*> 14.14.14.14/32, Intra, cost 3, area 0
   SPF Instance 37, age 00:00:12
   Flags: RIB, HiPrio
   via 12.0.0.2, GigabitEthernet2.512
   Flags: RIB
   LSA: 1/14.14.14.14/14.14.14.14
   repair path via 14.0.0.4, GigabitEthernet2.514, cost 4
   Flags: Ignore, Repair, IntfDj, BcastDj, NodeProt
   LSA: 1/14.14.14.14/14.14.14.14
   repair path via 123.0.0.2, GigabitEthernet2.513, cost 7
   Flags: Ignore, Repair, IntfDj, BcastDj, SRLG, Downstr
   LSA: 1/14.14.14.14/14.14.14.14
   repair path via 123.0.0.13, GigabitEthernet2.513, cost 7
   Flags: Ignore, Repair, IntfDj, BcastDj, SRLG, NodeProt, Downstr
   LSA: 1/14.14.14.14/14.14.14.14
```

A quick fix would be to somehow make CSR4 downstream (closer) to 14.14.14.14/32. If we increase the cost on CSR1's P2P interface to CSR2 to a number less than 5, CSR1 could potentially pick CSR4 as a best-path. Let's assume we want to keep the current best-path (via the CSR2 P2P link) but use CSR4 as a repair-path without modifying the tie-breakers. Increasing the cost on both the CSR2 and XRv13 P2P links to XRv14, followed by increasing the cost on CSR1's P2P link to CSR4, should meet the goal. The first modification would make CSR1 think that CSR2, XRv13, and CSR4 are all equidistant from XRv14. Increasing the cost on CSR1's P2P link to CSR4 ensures that the best-path is maintained and CSR4 is now considered downstream.

```
! CSR2
```



```

interface GigabitEthernet2.524
  ip ospf cost 2

! Xrv13
router ospf 1
  area 0
    interface GigabitEthernet0/0/0/0.534
      cost 2

! CSR1
interface GigabitEthernet2.514
  ip ospf cost 2

```

Now, the path through CSR4 is both downstream (CSR4's cost is 3, CSR1's cost is 4, and $3 < 4$) and does not share an SRLG with the primary path. It is a valid backup-path per our tie-breaker configuration. Before continuing, those custom tie-breaker commands are removed.

```

CSR1#show ip ospf rib 14.14.14.14 255.255.255.255
[snip]
*> 14.14.14.14/32, Intra, cost 4, area 0
  SPF Instance 40, age 00:00:52
  Flags: RIB, HiPrio
  via 12.0.0.2, GigabitEthernet2.512
    Flags: RIB
    LSA: 1/14.14.14.14/14.14.14.14
  repair path via 14.0.0.4, GigabitEthernet2.514, cost 5
    Flags: RIB, Repair, IntfDj, BcastDj, NodeProt, Downstr
    LSA: 1/14.14.14.14/14.14.14.14
  repair path via 123.0.0.2, GigabitEthernet2.513, cost 8
    Flags: Ignore, Repair, IntfDj, BcastDj, SRLG, Downstr
    LSA: 1/14.14.14.14/14.14.14.14
  repair path via 123.0.0.13, GigabitEthernet2.513, cost 7
    Flags: Ignore, Repair, IntfDj, BcastDj, SRLG, NodeProt, Downstr
    LSA: 1/14.14.14.14/14.14.14.14

```

An interesting corner-case is using the broadcast-interface-disjoint tiebreaker. This ensures that an LFA does not transit through the same broadcast interface used by the primary path, but from the perspective another router. For example, if CSR1 selects the best path via Xrv13 (LAN interface) to reach 14.14.14.14/32, if CSR2 was an LFA, it is preferable for CSR2 not to follow the path CSR2 > Xrv13 > Xrv14. This would cause the primary and backup paths to both rely on the same LAN, which is what the BcastDj option attempts to avoid. Instead, the LFA via CSR4 > Xrv13 > Xrv14 is a better option. Adjusting costs as shown below ensures that Xrv13 > Xrv14 is the primary path, but the LFA via CSR2 is ignored because it uses the same LAN as the primary path.

```

! CSR1
interface GigabitEthernet2.512
  ip ospf cost 2

```

```

interface GigabitEthernet2.514
 ip ospf cost 2

! CSR2
interface GigabitEthernet2.524
 ip ospf cost 3

! CSR4
interface GigabitEthernet2.545
 ip ospf cost 2

CSR1#show ip ospf rib 14.14.14.14 255.255.255.255
[snip]
*> 14.14.14.14/32, Intra, cost 3, area 0
   SPF Instance 15, age 00:00:20
   Flags: RIB, HiPrio
   via 123.0.0.13, GigabitEthernet2.513
     Flags: RIB
     LSA: 1/14.14.14.14/14.14.14.14
     repair path via 14.0.0.4, GigabitEthernet2.514, cost 5
       Flags: RIB, Repair, IntfDj, BcastDj
       LSA: 1/14.14.14.14/14.14.14.14
       repair path via 12.0.0.2, GigabitEthernet2.512, cost 5
         Flags: Ignore, Repair, IntfDj
         LSA: 1/14.14.14.14/14.14.14.14

```

```

CSR2#show ip route 14.14.14.14
Routing entry for 14.14.14.14/32
  Known via "ospf 1", distance 110, metric 3, type intra area
  Last update from 123.0.0.13 on GigabitEthernet2.513, 00:07:26 ago
  Routing Descriptor Blocks:
  * 123.0.0.13, from 14.14.14.14, 00:07:26 ago, via GigabitEthernet2.513
    Route metric is 3, traffic share count is 1
    Repair Path: 24.0.0.14, via GigabitEthernet2.524

```

Next, we will do some basic tests with XR direct LFA using OSPFv2. The network has been reset with all OSPF costs set to 1 and all interfaces enabled (cleanup not shown). We enable per-prefix LFA under area 0 which applies to all area 0 interfaces also (by inheritance). This is applied on XRv14 only since XRv13 will be used for per-link LFA later.

```

! XRv14
router ospf 1
 area 0
  fast-reroute per-prefix

```

Examining the route to CSR1's loopback, we see two equal cost paths, both of which back each other up. Notice they are primary paths (both are used for ECMP), downstream (both CSR2 and XRv13 are closer

to CSR1 than XRv14 is), node protecting (the paths do not rely on the same transit routers), and SRLG disjoint.

```
RP/0/0/CPU0:XRv14#show ospf routes 1.1.1.1/32 backup-path detail
OSPF Route entry for 1.1.1.1/32
  Route type:  Intra-area
  Metric: 3
  SPF priority: 4,  SPF version: 69
  RIB version: 0,  Source: Unknown
    43.0.0.13, from 1.1.1.1, via GigabitEthernet0/0/0/0.543, path-id 2
      Backup path:
        24.0.0.2, from 1.1.1.1, via GigabitEthernet0/0/0/0.524,
protected bitmap 0000000000000002
      Attributes: Metric: 3, Primary, Downstream, Node Protect, SRLG
Disjoint
    24.0.0.2, from 1.1.1.1, via GigabitEthernet0/0/0/0.524, path-id 1
      Backup path:
        43.0.0.13, from 1.1.1.1, via GigabitEthernet0/0/0/0.543,
protected bitmap 0000000000000001
      Attributes: Metric: 3, Primary, Downstream, Node Protect, SRLG
Disjoint
```

We can see another path through CSR5 by looking at the diagram but OSPF does not display it here. We will apply SRLGs to XRv14's interfaces facing CSR2 and XRv13 so that we prefer not to use backup paths that way. We make SRLG the #1 tiebreaker also, following the same convention tested with XE earlier. Additionally, we will increase the cost of XRv14's interface facing CSR2 to 2 so that ECMP is not in play. Notice that applying SRLGs in XR is identical across LFA and MPLS-TE. In XE, there are two different mechanisms, but XR consolidates this nicely.

```
! XRv14
srlg
interface GigabitEthernet0/0/0/0.524
  8 value 654

interface GigabitEthernet0/0/0/0.543
  8 value 654

router ospf 1
fast-reroute per-prefix tiebreaker srlg-disjoint index 1
area 0
interface GigabitEthernet0/0/0/0.524
cost 2
```

```
RP/0/0/CPU0:XRv14#show ospf routes 1.1.1.1/32 backup-path detail

OSPF Route entry for 1.1.1.1/32
  Route type:  Intra-area
  Metric: 3
```

```

SPF priority: 4,  SPF version: 74
RIB version: 0,  Source: Unknown
  43.0.0.13, from 1.1.1.1, via GigabitEthernet0/0/0/0.543, path-id 1
    Backup path:
      54.0.0.5, from 1.1.1.1, via GigabitEthernet0/0/0/0.554,
protected bitmap 00000000000000001
  Attributes: Metric: 4, Node Protect, SRLG Disjoint

```

If we remove the SRLGs (delete the entire feature for brevity), the router goes back to picking CSR2. At this point, both paths are SRLG disjoint but CSR2 is downstream (closer to CSR1 than XRv14 is) where CSR5 is not.

```

! XRv14
no srlg

```

```

RP/0/0/CPU0:XRv14#show ospf routes 1.1.1.1/32 backup-path detail

```

```

OSPF Route entry for 1.1.1.1/32
  Route type: Intra-area
  Last updated: MON  8 12:54:59.512
  Metric: 3
  SPF priority: 4,  SPF version: 74
  RIB version: 0,  Source: Unknown
    43.0.0.13, from 1.1.1.1, via GigabitEthernet0/0/0/0.543, path-id 1
      Backup path:
        24.0.0.2, from 1.1.1.1, via GigabitEthernet0/0/0/0.524,
protected bitmap 00000000000000001
  Attributes: Metric: 4, Downstream, Node Protect, SRLG Disjoint

```

Let's make CSR5 downstream somehow. We accomplish this by significantly increasing all of XRv14's metrics so that all neighbors appear downstream. We will also increase CSR2's metric by even more so that it loses the LFA tie-break based on worse cost. To prove that metric is the tie-breaker, we will put the downstream attribute as the #2 item in the tie-breaker process.

```

! XRv14
router ospf 1
  fast-reroute per-prefix tiebreaker downstream index 2
  area 0
    interface GigabitEthernet0/0/0/0.524
      cost 110

    interface GigabitEthernet0/0/0/0.543
      cost 101

    interface GigabitEthernet0/0/0/0.554
      cost 101

```

```

RP/0/0/CPU0:XRv14#show ospf routes 1.1.1.1/32 backup-path detail
OSPF Route entry for 1.1.1.1/32
  Route type: Intra-area
  Metric: 103
  SPF priority: 4,  SPF version: 91
  RIB version: 0,  Source: Unknown
    43.0.0.13, from 1.1.1.1, via GigabitEthernet0/0/0/0.543, path-id 1
      Backup path:
        54.0.0.5, from 1.1.1.1, via GigabitEthernet0/0/0/0.554,
protected bitmap 0000000000000001
  Attributes: Metric: 104, Downstream, Node Protect, SRLG Disjoint

```

We now reset all the costs on XRv14 back to 1 (not shown) and shift focus to XRv13. We will briefly test per-link LFA, which is an alternative LFA technique. Per-link runs SPF for all prefixes learned through a primary next-hop. This is obviously less ideal as certain links will not yield LFA paths for all prefixes, but is significantly less CPU intensive. This is only supported on XR and appears to be a less popular variant of LFA. I use a medium prefix-priority just for variety, although it doesn't mean much in a network where we are only evaluating loopback addresses.

```

! XRv13
router ospf 1
  fast-reroute per-link priority-limit medium
  area 0
  fast-reroute per-link

```

Based on the output below, we can conclude the following. The “(!)” symbol indicates an FRR path.

Prefixes with next-hop 123.0.0.1 (1.1.1.1/32, 4.4.4.4/32) use CSR4 34.0.0.4

Prefixes with next-hop 123.0.0.2 (2.2.2.2/32) use XRv14 43.0.0.14

Prefixes with next-hop 43.0.0.14 (5.5.5.5/32, 14.14.14.14/32) use CSR 123.0.0.2

Prefixes with next-hop 34.0.0.4 (5.5.5.5/32, 4.4.4.4/32) use CSR1 123.0.0.1

```

RP/0/0/CPU0:XRv13#show route ospf
O    1.1.1.1/32 [110/2] via 123.0.0.1, 00:02:34, GigabitEthernet0/0/0/0.513
      [110/0] via 34.0.0.4, 00:02:34, GigabitEthernet0/0/0/0.534 (!)
O    2.2.2.2/32 [110/0] via 43.0.0.14, 00:02:34, GigabitEthernet0/0/0/0.543 (!)
      [110/2] via 123.0.0.2, 00:02:34, GigabitEthernet0/0/0/0.513
O    4.4.4.4/32 [110/0] via 123.0.0.1, 00:02:03, GigabitEthernet0/0/0/0.513 (!)
      [110/2] via 34.0.0.4, 00:02:03, GigabitEthernet0/0/0/0.534
O    5.5.5.5/32 [110/3] via 43.0.0.14, 00:02:03, GigabitEthernet0/0/0/0.543
      [110/0] via 123.0.0.2, 00:02:03, GigabitEthernet0/0/0/0.513 (!)
      [110/0] via 123.0.0.1, 00:02:03, GigabitEthernet0/0/0/0.513 (!)
      [110/3] via 34.0.0.4, 00:02:03, GigabitEthernet0/0/0/0.534
O    14.14.14.14/32 [110/2] via 43.0.0.14, 00:02:03, GigabitEthernet0/0/0/0.543
      [110/0] via 123.0.0.2, 00:02:03, GigabitEthernet0/0/0/0.513 (!)
O    24.0.0.0/24 [110/2] via 43.0.0.14, 00:02:34, GigabitEthernet0/0/0/0.543
O    54.0.0.0/24 [110/2] via 43.0.0.14, 00:02:34, GigabitEthernet0/0/0/0.543

```

If we add several prefixes to (or behind) XRv14, their FRR paths will be identical to 14.14.14.14/32 regardless of the number of prefixes we add and without additional computation. I've added 4 to demonstrate the point. Of course, the result would be the same with per-prefix, but with additional computation for each prefix. When the network has an enormous number of prefixes, it is likely that the number of links on a router is significantly less than the number of prefixes, so this can be a way of scaling LFA.

```
RP/0/0/CPU0:XRv13#show route longer-prefixes 14.14.14.0/24 | begin ^O
O    14.14.14.1/32 [110/0] via 123.0.0.2, 00:00:32, GigabitEthernet0/0/0/0.513 (!)
      [110/2] via 43.0.0.14, 00:00:32, GigabitEthernet0/0/0/0.543
O    14.14.14.2/32 [110/0] via 123.0.0.2, 00:00:32, GigabitEthernet0/0/0/0.513 (!)
      [110/2] via 43.0.0.14, 00:00:32, GigabitEthernet0/0/0/0.543
O    14.14.14.3/32 [110/0] via 123.0.0.2, 00:00:32, GigabitEthernet0/0/0/0.513 (!)
      [110/2] via 43.0.0.14, 00:00:32, GigabitEthernet0/0/0/0.543
O    14.14.14.4/32 [110/0] via 123.0.0.2, 00:00:32, GigabitEthernet0/0/0/0.513 (!)
      [110/2] via 43.0.0.14, 00:00:32, GigabitEthernet0/0/0/0.543
O    14.14.14.14/32 [110/0] via 123.0.0.2, 00:12:21, GigabitEthernet0/0/0/0.513 (!)
      [110/2] via 43.0.0.14, 00:12:21, GigabitEthernet0/0/0/0.543
```

37.1.1.2 Remote LFA

In ring designs with 4 or more nodes, you cannot use regular LFA to reach some nodes (assuming all costs are equal) since your neighbors sometimes route through you. The network has been modified by setting all OSPF costs to 1, shutting down the shared LAN interface, removing SRLGs, and removing CSR5 completely. CSR1's tiebreaker policies have been set to the default values. We now have a pentagon shape where all costs are equal (CSR1-CSR2-XRv14-XRv13-CSR4-CSR1). If CSR1 tries to compute a backup-path to CSR2's loopback, CSR4 says that it routes through CSR1 to get there. This is not going to work for ordinary LFA as it is not a loop-free path. Remote LFA (rLFA) logic solves this problem by applying the following logic: "I can tunnel the traffic to a node that CAN reach CSR2 through a backup path (like XRv13) and let that node deliver the traffic". In this scenario, each node will have a backup path for nodes that are exactly two hops away; for example, CSR1 has backup paths for XRv13 and XRv14, but not CSR2 and CSR4. This is using regular LFA with no modifications, but ideally we want to protect paths to CSR2 and CSR4 also around the ring topology. **Note: Only MPLS encapsulation with LDP signaling is currently supported.** LDP is enabled on all OSPF-enabled links in the network (not shown). I suppose this is a limitation of Segment Routing (SR) in that remote-LFA cannot use it for tunnel label bindings.

```
CSR1#show ip ospf rib
[snip]
*    1.1.1.1/32, Intra, cost 1, area 0, Connected
      via 1.1.1.1, Loopback0
*>   2.2.2.2/32, Intra, cost 2, area 0
      via 12.0.0.2, GigabitEthernet2.512
*>   4.4.4.4/32, Intra, cost 2, area 0
      via 14.0.0.4, GigabitEthernet2.514
*>   13.13.13.13/32, Intra, cost 3, area 0
      via 14.0.0.4, GigabitEthernet2.514
```

```

repair path via 12.0.0.2, GigabitEthernet2.512, cost 4
*> 14.14.14.14/32, Intra, cost 3, area 0
    via 12.0.0.2, GigabitEthernet2.512
    repair path via 14.0.0.4, GigabitEthernet2.514, cost 4

```

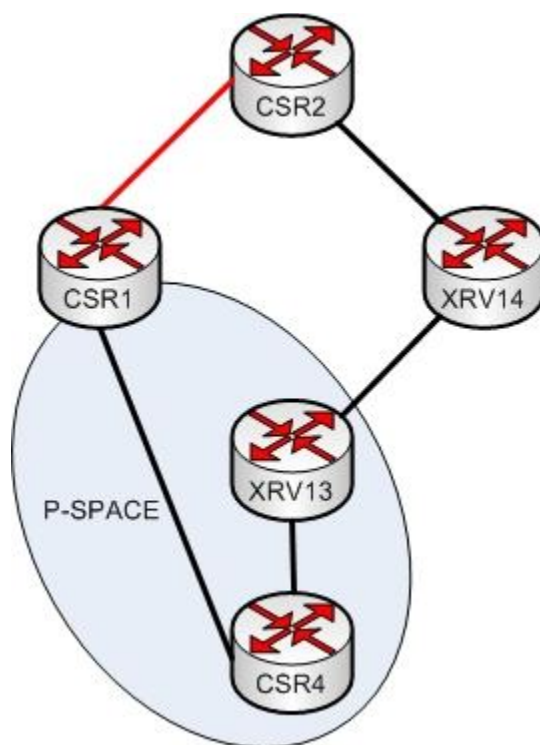
Remote LFA introduces several new terms which are discussed first. There are two "spaces" that have been defined to show how remote-LFA works. I derive these definitions from "draft-shand-remote-lfa-00" as they are succinct and clear.

P-space: Set of routers reachable without any path (including ECMP options) transiting the protected link. For example, from CSR1's perspective we are trying to protect a path to CSR2. Our shortest path is the direct link, which is what we are trying to protect, so the P-space routers would not be reachable over that link. A quick look at CSR1's routing table can tell us which routers we CAN reach through the link towards CSR4. Clearly, we can reach CSR4 and XRV13 this way. XRV14 and CSR2 are not considered P-space routers because CSR1's shortest path to them is via the protected link. Attached is a picture to help visualize the P-space; the red colored link is the link we are trying to protect.

```

CSR1#show ip route | include 14.0.0.4
[snip]
O      4.4.4.4 [110/2] via 14.0.0.4, 00:06:40, GigabitEthernet2.514
O      13.13.13.13 [110/3] via 14.0.0.4, 00:09:23, GigabitEthernet2.514

```



Q-space: Set of routers from which the destination can be reached without any path (including ECMP options) transiting the protected link. For example, we are trying to reach CSR2; that is the destination.

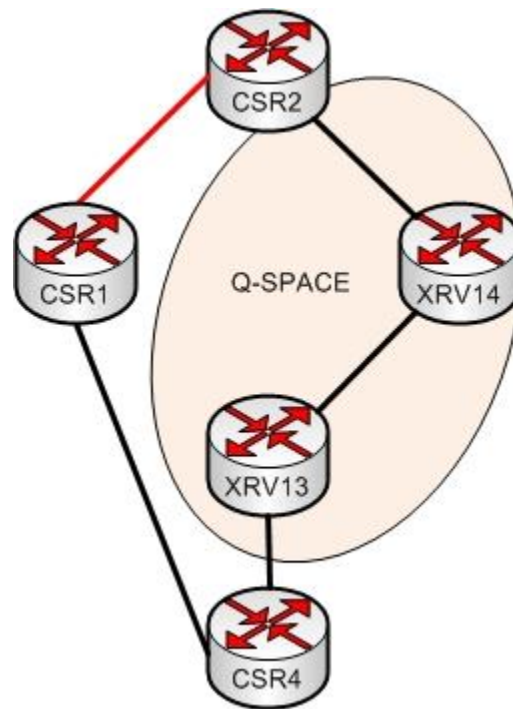
Q-space determines which routers can reach CSR2 on their best-paths without transiting the protected link (CSR1-CSR2). We would need to check each router's route to 2.2.2.2/32 to see which ones fit within the Q-space and which don't. I will use traceroute to speed things up. The highlights represent the protected link, which implies the router is NOT part of the Q-space if its path traverses that link. This output reveals that XRv13 and XRv14 are in the Q-space, but CSR1 and CSR4 are not. XRv13 and XRv14 can both reach CSR2 by not transiting the link we are trying to protect. Attached is a picture to help visualize the Q-space.

```
CSR1#traceroute 2.2.2.2 source loopback0
Type escape sequence to abort.
Tracing the route to 2.2.2.2
VRF info: (vrf in name/id, vrf out name/id)
 1 12.0.0.2 1 msec * 1 msec
```

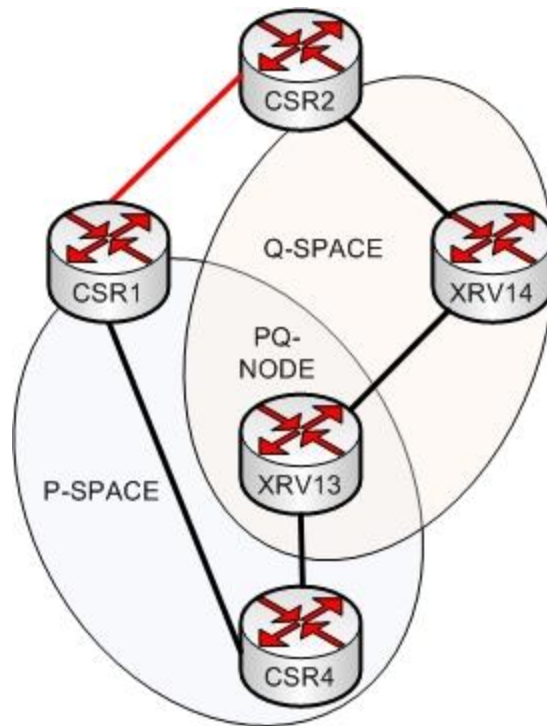
```
CSR4#traceroute 2.2.2.2 source loopback 0
Type escape sequence to abort.
Tracing the route to 2.2.2.2
VRF info: (vrf in name/id, vrf out name/id)
 1 14.0.0.1 [MPLS: Label 1002 Exp 0] 1 msec 1 msec 1 msec
 2 12.0.0.2 1 msec * 1 msec
```

```
RP/0/0/CPU0:XRv13#traceroute 2.2.2.2 source 13.13.13.13
Type escape sequence to abort.
Tracing the route to 2.2.2.2
 1 43.0.0.14 [MPLS: Label 94001 Exp 0] 29 msec 19 msec 29 msec
 2 24.0.0.2 29 msec * 29 msec
```

```
RP/0/0/CPU0:XRv14#traceroute 2.2.2.2 source 14.14.14.14
Type escape sequence to abort.
Tracing the route to 2.2.2.2
 1 24.0.0.2 0 msec * 0 msec
```

PQ-node: A router that exists in both the P and Q spaces concurrently. The logic is as follows: the source router (CSR1) can reach any P-space router without transiting the protected link. Any Q-space router can reach the destination (CSR2) without transiting the protected link. Therefore, CSR1 can reach XRV13 and XRV13 can reach CSR2 without transiting the protected link. CSR1 establishes a "repair tunnel" to XRV13 (the PQ-node) inside of MPLS. XRV13 is the tail end of this repair tunnel, sending tunneled packets along their natural best-path to CSR2. Attached is a picture to help visualize the PQ-node.



Like regular LFA, the configuration is very straightforward and is demonstrated on CSR1. The first command tells it what kind of encapsulation to use, and right now “mpls-ldp” is the only option. Future encapsulations could potentially include MPLS with Segment Routing, GRE/IP, or any other tunneling techniques. The second command puts an upper-bound on how far the tunnel can go. Realistically, you do not want to tunnel backup traffic too long in your network. In our case, I set an aggressive value of 3 to ensure we do not tunnel traffic more than 2 hops (plus the cost of the loopback). After all, if CSR1 can tunnel traffic to XRV13 (the PQ router), that is all we need in a small 5-node ring. Likewise, in the opposite direction, if we can tunnel to XRV14 as a PQ-node, we can backup CSR4's loopback. I am only enabling rLFA in area 0 although you can omit the area and enable it for all areas (plus external routes). In fact, **you must omit the area if you want to protect external routes**, which implicitly means rLFA is enabled for all areas as well.

```
! CSR1
router ospf 1
  fast-reroute per-prefix remote-lfa area 0 tunnel mpls-ldp
  fast-reroute per-prefix remote-lfa area 0 maximum-cost 3
```

The head-end (CSR1) sends targeted LDP hellos towards 13.13.13.13/32 (and also XRV14 to protect CSR4). We need to enable the PQ-nodes to accept these targeted hellos; we will enable it on XRV13 only first. Notice what happens if you skip this step as we did on XRV14. The LDP neighbor does not form and rLFA will not work. CSR1 picked XRV13 because it was a PQ-router; the algorithm that determined this happens behind the scenes automatically based on the aforementioned P-space / Q-space discussion. You do not need to specify the rLFA tunnel destination, which greatly simplifies the configuration.

```

! XRv13
mpls ldp
  address-family ipv4
    discovery targeted-hello accept

CSR1#show mpls ldp discovery
Local LDP Identifier:
  1.1.1.1:0
Discovery Sources:
Interfaces:
  GigabitEthernet2.512 (ldp): xmit/recv
    LDP Id: 2.2.2.2:0
  GigabitEthernet2.514 (ldp): xmit/recv
    LDP Id: 4.4.4.4:0
Targeted Hellos:
  1.1.1.1 -> 13.13.13.13 (ldp): active, xmit/recv
    LDP Id: 13.13.13.13:0
  1.1.1.1 -> 14.14.14.14 (ldp): active, xmit

CSR1#show mpls ldp neighbor | include Peer_LDP
Peer LDP Ident: 4.4.4.4:0; Local LDP Ident 1.1.1.1:0
Peer LDP Ident: 2.2.2.2:0; Local LDP Ident 1.1.1.1:0
Peer LDP Ident: 13.13.13.13:0; Local LDP Ident 1.1.1.1:0

```

CSR1 learns XRv13's LDP local-label for 2.2.2.2/32 via the rLFA tunnel, which is 93008. This will be the inner label when rLFA traffic is tunneled to XRv13. Also take note of CSR4's label for 2.2.2.2/32, which is 4002, as this will be used after CSR1's SPF process completes and traffic stops forwarding over the rLFA tunnel. Just like TE-FRR in ordinary MPLS, the backup path is only used for a very short time until the tunnel head-end repairs failure. In this case, once IGP is aware of the failure, it uses the backup path immediately (LFA or rLFA) and computes SPF in the meantime. Assuming SPF completes quickly, traffic will follow a new, more stable LSP, which in this case would use label 4002 via CSR4 directly.

```

CSR1#show mpls ldp bindings 2.2.2.2 32
lib entry: 2.2.2.2/32, rev 4
  local binding:  label: 1000
  remote binding: lsr: 4.4.4.4:0, label: 4002
  remote binding: lsr: 2.2.2.2:0, label: imp-null
  remote binding: lsr: 13.13.13.13:0, label: 93008

```

When sending traffic destined to XRv13 (which is the tail end of the rLFA tunnel), we must use CSR4's label for 13.13.13.13/32, which is 4004. This is the outer label for the rLFA tunnel, as well as the targeted LDP session with XRv13.

```

CSR1#show mpls ldp bindings 13.13.13.13 32
lib entry: 13.13.13.13/32, rev 10
  local binding:  label: 1002
  remote binding: lsr: 4.4.4.4:0, label: 4004

```

```
remote binding: lsr: 2.2.2.2:0, label: 2003
remote binding: lsr: 13.13.13.13:0, label: imp-null
```

As seen below, a repair tunnel called "MPLS-Remote-Lfa1" was created to backup this destination. Assuming XRv14 was correctly configured to accept tLDP sessions (which it is not), we would see a second rLFA tunnel towards XRv14. The number of this tunnel changes every time the protected link flaps, because after CSR1 converges and uses the real LSP, the rLFA tunnel is torn down as it is not needed.

```
CSR1#show ip ospf rib 2.2.2.2 255.255.255.255
[snip]
*> 2.2.2.2/32, Intra, cost 2, area 0
    SPF Instance 52, age 00:28:45
    Flags: RIB, HiPrio
    via 12.0.0.2, GigabitEthernet2.512
    Flags: RIB
    LSA: 1/2.2.2.2/2.2.2.2
    repair path via 13.13.13.13, MPLS-Remote-Lfa1, cost 4
    Flags: RIB, Repair, IntfDj, BcastDj
    LSA: 1/2.2.2.2/2.2.2.2
```

You can see some extra tunnel details which helps define the path. This shows XRv13 as the tail-end (PQ-node) with CSR4 being the first hop in the tunnel's path.

```
CSR1#show ip ospf fast-reroute remote-lfa tunnels
Interface MPLS-Remote-Lfa1
Tunnel type: MPLS-LDP
Tailend router ID: 13.13.13.13
Termination IP address: 13.13.13.13
Outgoing interface: GigabitEthernet2.514
First hop gateway: 14.0.0.4
Tunnel metric: 2
Protects:
    12.0.0.2 GigabitEthernet2.512, total metric 4
```

BFD must be configured for rLFA to work. The control-plane is fully operational without it as verified above, but you must use BFD. IGP fast hellos will not work, regardless of how fast they are, and BFD will work, no matter how slow it is. Thus, BFD has been enabled and bound to OSPF on the link between CSR1 and CSR2 (basic BFD configuration not shown). The "FRR" registered protocol represents LFA. Cisco suggests BFD is requirement for direct LFA as well, which makes sense.

```
CSR1#show bfd neighbors details | include 12.0.0.2|Register
12.0.0.2                4097/4097        Up        Up        Gi2.512
Registered protocols: OSPF CEF FRR
```

I added a quick Flexible Netflow (FNF) configuration to CSR4 to track incoming MPLS packets. This way, we can visualize CSR1 sending MPLS-encapsulated backup traffic through the rLFA tunnel.

```
! CSR4
flow record FNF_MPLS_RECORD
 match mpls label 1 details
 match mpls label 2 details
 collect counter packets

flow monitor FNF_MPLS_MONITOR
 record FNF_MPLS_RECORD

interface GigabitEthernet2.514
 mpls flow monitor FNF_MPLS_MONITOR input
```

The output below is showing the targeted LDP messages being sent from CSR1 to XRv13 (notice the label value of 4004 with EXP 6). This is just a basic TCP session between CSR1 and XRv13. This isn't relevant for rLFA, but we use it to verify that FNF was configured correctly.

```
CSR4#show flow monitor FNF_MPLS_MONITOR cache format table | begin MPLS
MPLS LABEL 1  MPLS LABEL 2      pkts
=====
4004 /6      0 /0      118
```

Next, I send several pings from CSR1's loopback to CSR2's loopback with DSCP CS1 (for identification, just in case something else generates traffic behind the scenes), then break the protected link on CSR2. We expect to see some traffic (not many packets) with a label stack of {4004 93008} which is used to tunnel traffic inside the rLFA tunnel to XRv13 (4004), then use XRv13's label for 2.2.2.2/32 to reach the final destination (93008). After OSPF converges, CSR1 uses the ordinary LDP-derived LSP via CSR4's label for 2.2.2.2/32 (4002). Tunneled rLFA traffic counters are shown in green and native forwarding through CSR4, after IGP convergence, is shown in yellow.

```
CSR1#ping 2.2.2.2 source loopback0 tos 32 repeat 100000000
```

```
! CSR2
interface GigabitEthernet2.512
 shutdown
```

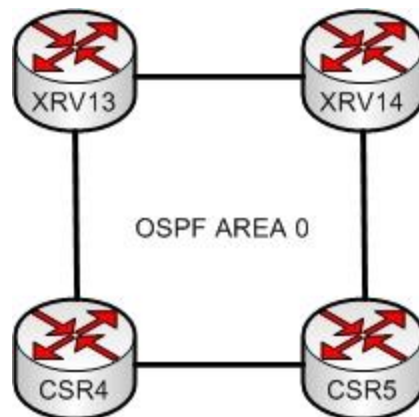
```
CSR4#show flow monitor FNF_MPLS_MONITOR cache format table | begin MPLS
MPLS LABEL 1  MPLS LABEL 2      pkts
=====
4002 /1      0 /0      242
4004 /1      93008 /1      128
4004 /6      0 /0      18
```

```
CSR4#show flow monitor FNF_MPLS_MONITOR cache format table | begin MPLS
```

MPLS LABEL 1		MPLS LABEL 2		pkts
=====		=====		=====
4002	/1	0	/0	267
4004	/1	93008	/1	128
4004	/6	0	/0	18

In summary, the traffic with label stack {4004 93008} was inside the rLFA tunnel and was used for only the amount of time it took CSR1 to reconverge (I did not tune OSPF timers for ultra-fast convergence). Notice the difference in packet counters between the two FNF show commands; the rLFA stopped forwarding after 128 packets and the label stack {4002} was used thereafter as this was the post-SPF LSP. I highlighted the packet counters in different colors above so you can see that rLFA counters stop while the normal LSP counters continue to increase. We don't need to trace all the LSPs as the FNF output reveals the key behavioral components of this feature.

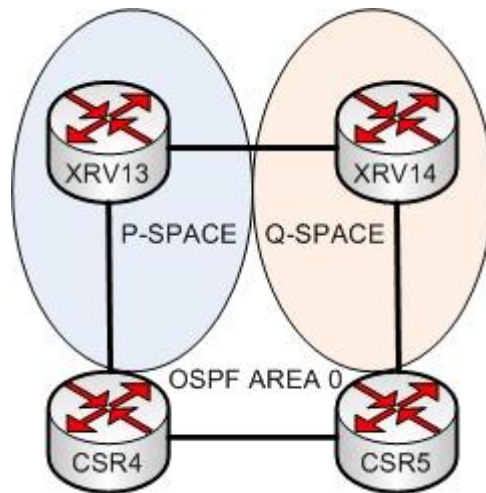
When we have a ring topology with an even number of nodes greater than 3, remote LFA becomes a little more complicated. CSR1 is removed from the network and CSR4's link to XRv13 is restored, making a square.



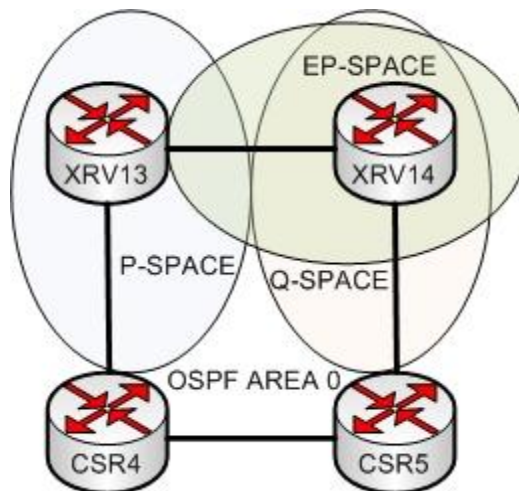
Given what we know about P and Q spaces, assuming we want to protect the link between CSR4 and CSR5, there is no traditional PQ-node. Without a PQ-node, it seems that remote LFA cannot work. The reason this seems to be the case is because CSR4's route to XRv14 is via an ECMP path-split between XRv13 and CSR5, which means XRv14 is not in CSR4's P-space. Additionally, because XRv13's route to CSR5 is via ECMP also, it is not part of CSR5's Q-space.

```
CSR4#show ip cef 14.14.14.14 | include _nexthop
  nexthop 34.0.0.13 GigabitEthernet2.534 label [93015|5004]
  nexthop 45.0.0.5 GigabitEthernet2.545 label [5004|93015]
```

```
RP/0/0/CPU0:XRv13#show cef ipv4 5.5.5.5/32 | include next hop
  next hop 34.0.0.4
  next hop 43.0.0.14
```



In order for rLFA to work, XRV14 needs to be in CSR4's P-space or XRV13 needs to be in CSR5's Q-space. This problem is solved by leveraging the "extended P-space" which makes the first condition somewhat true. The logic is as follows: CSR4 will only use the repair path when the direct link to CSR5 is down. This initial hop does not need to be subject to CSR4's normal routing process; this effectively allows us to relax the "you can't use ECMP" rule. The extended P-space is the union (combination) of the P-spaces of the source router (CSR4) and all of its P-space neighbors (XRV13). In terms of cost, a router is in the extended P-space if the cost XRV13->XRV14 is less than the cost XRV13->CSR4->CSR5->XRV14 ($1 < 3$). Basically, your neighbor (XRV13) has to be closer to the target PQ-node. In this case, XRV14 is within XRV13's P-space since it is reachable without using the protected link and is closer than CSR4 (downstream); XRV14 is in CSR4's extended P-space and is thus the PQ-node.



Fortunately, there is nothing special you have to configure for this advanced feature to work. Just enable the rLFA feature and the router "figures it out". We should see an rLFA tunnel from CSR4 to XRV14. In earlier examples we did not have LDP targeted session acceptance configured on XRV14 for demonstration purposes, so we must add it.

```

! XRv14
mpls ldp
  address-family ipv4
    discovery targeted-hello accept

```

I've also set up BFD and bound OSPF to it on the link between CSR4 and CSR5; remember that this is a requirement for rLFA to work. CSR4 actually builds two tunnels, one via CSR5 and one via XR13. Since our current objective is protecting the link to CSR5, we will ignore the second tunnel. Both of them terminate on XRv14 since it would be the PQ-node in either direction. Since XRv doesn't support BFD, we test link failures between CSR4 and CSR5 only.

```

CSR4#show ip ospf fast-reroute remote-lfa tunnels
[snip]
Interface MPLS-Remote-Lfa5
  Tunnel type: MPLS-LDP
  Tailend router ID: 14.14.14.14
  Termination IP address: 14.14.14.14
  Outgoing interface: GigabitEthernet2.534
  First hop gateway: 34.0.0.13
  Tunnel metric: 2
  Protects:
    45.0.0.5 GigabitEthernet2.545, total metric 3
[snip]

```

Performing a similar test as before, we send pings from CSR4 to CSR5, break the direct link, and ensure XRv13 sees the proper rLFA label stack. CSR4 learns label 94003 for the prefix 5.5.5.5/32 from XRv14 via tLDP. This will be the bottom label in the rLFA tunnel stack. CSR4 also needs a label for XRv14's loopback, the tunnel tail end. Because the route to 14.14.14.14/32 is IGP learned, an LDP label must be used, and XRv13 advertises label 93015. We expect the rLFA tunnel to use label stack {93015 94003} to tunnel traffic first to XRv14, then use XRv14's label for CSR5's loopback to deliver the traffic. When OSPF converges, we should use XRv13's LDP label for CSR5's loopback, which is 93002.

```

CSR4#show mpls ldp bindings 5.5.5.5 32
  lib entry: 5.5.5.5/32, rev 41
    local binding:  label: 4001
    remote binding: lsr: 5.5.5.5:0, label: imp-null
    remote binding: lsr: 13.13.13.13:0, label: 93002
    remote binding: lsr: 14.14.14.14:0, label: 94003

CSR4#show mpls ldp bindings 14.14.14.14 32
  lib entry: 14.14.14.14/32, rev 14
    local binding:  label: 4003
    remote binding: lsr: 13.13.13.13:0, label: 93015
    remote binding: lsr: 5.5.5.5:0, label: 5004
    remote binding: lsr: 14.14.14.14:0, label: imp-null

```


FNF does not appear to be supported on XRv, so we will use MPLS packet debugging, which does work. The XR debugs for this are extremely detailed; we will send these logs to the buffer only. We expect to see a short burst of traffic with label stack {93015 94003} with all follow-on traffic using label 93002. Like last time, we use MPLS EXP1 to differentiate the traffic against other potential flows, enhancing readability.

```
XRv13#debug mpls packet
```

```
CSR4#ping 5.5.5.5 source loopback0 tos 32 repeat 100000000
```

```
! CSR5
interface GigabitEthernet2.545
 shutdown
```

Here is debug output from XRv13 that shows traffic coming in with label 93015, which is XRv13's label for XRv14's loopback. We assume this is the rLFA tunnel because of the EXP bits being set to 1 and the label values appear correct. The LFIB states PHP should occur, so only label 94003 should go out towards XRv14. Notice that the packet length is 130 bytes on ingress.

```
! XRv13
mpls_switch: FINT0_0_CPU0, mpls eos 0, ttl 254, len 130, inlabel 93015,
tbl_id=0xe0000000, vrf_id=0x60000000 in=0xb00
mpls_rewrite
mpls_rewrite: tos 32 eos 0 ttl 254, out_label 1048577, #labels 0, RA - 0,
out_intf GigabitEthernet0_0_0.543
mpls_rewrite: POP to mpls
mpls_rewrite: tos 1 (0x1) eos 1 ttl 254, out_label 94003, top_of_stack
0x16f333fe
```

```
RP/0/0/CPU0:XRv13#show mpls forwarding labels 93015
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
93015	Pop	14.14.14.14/32	Gi0/0/0/0.543	43.0.0.14	1665321

A few seconds later, OSPF converges and XRv13 only sees traffic using label 93002 to reach CSR5's loopback as the rLFA tunnel is torn down. Notice the packet length is also 4 bytes less (126 vs. 130) than it was when using rLFA as the label stack contains one label, not two. XRv13 performs a label swap, not a pop, to send this traffic towards CSR5 via XRv14. The debug earlier clearly showed "POP to mpls" where this debug does not.

```
! XRv13
mpls_switch-2: Setting default table-id
Flag set to 0x00000001
mpls_switch: FINT0_0_CPU0, mpls eos 1, ttl 254, len 126, inlabel 93002,
tbl_id=0xe0000000, vrf_id=0x60000000 in=0xb00
```

```

mpls_rewrite
mpls_rewrite: tos 32 eos 1 ttl 254, out_label 94003, #labels 1, RA - 0,
out_intf GigabitEthernet0_0_0_0.543
mpls_rewrite: tos 1 (0x1) eos 1 ttl 254, out_label 94003, top_of_stack
0x16f333fe

```

Because BFD is not supported on XRv, we will not test the data plane of rLFA, but instead will verify control plane signaling when the rLFA tunnel headend is on XRv. Using the same square topology, we enable rLFA on XRv13 to protect the link to XRv14. On CSR5, we will configure LDP to accept targeted hellos as XRv13 must learn CSR5's label for XRv14.

```

! XRv13
router ospf 1
  fast-reroute per-prefix remote-lfa tunnel mpls-ldp
  fast-reroute per-prefix remote-lfa maximum-cost 20

! CSR5
mpls ldp discovery targeted-hello accept

```

To verify that we learn the label, we can check the LDP bindings to discover the value 5004. This is the bottom label in the rLFA tunnel. The top label is CSR4's label for CSR5, which is 4001. Thus, the rLFA tunnel encapsulation is {4001 5004}.

```

RP/0/0/CPU0:XRv13#show mpls ldp bindings 14.14.14.14/32 neighbor 5.5.5.5
14.14.14.14/32, rev 129

```

```

    Local binding: label: 93015
    Remote bindings: (3 peers)
      Peer                Label
      -----
      5.5.5.5:0           5004

```

```

RP/0/0/CPU0:XRv13#show mpls ldp bindings 5.5.5.5/32 neighbor 4.4.4.4
5.5.5.5/32, rev 164

```

```

    Local binding: label: 93002
    Remote bindings: (3 peers)
      Peer                Label
      -----
      4.4.4.4:0           4001

```

The OSPF RIB, like XE, also tracks the backup paths with all of the tiebreaker attributes (SRLG, node protecting, interface disjoint, etc).

```

RP/0/0/CPU0:XRv13#show ospf routes 14.14.14.14/32 backup-path detail
OSPF Route entry for 14.14.14.14/32
  Route type: Intra-area
  Metric: 2
  SPF priority: 4,  SPF version: 134

```

```

RIB version: 0, Source: Unknown
  43.0.0.14, from 14.14.14.14, via GigabitEthernet0/0/0/0.543, path-id 1
    Backup path: Remote, LFA: 5.5.5.5
      34.0.0.4, from 14.14.14.14, via GigabitEthernet0/0/0/0.534,
protected bitmap 0000000000000001
  Attributes: Metric: 2, SRLG Disjoint

```

We can check the FIB to ensure this is the correct label stack and that the backup path is installed also.

```

RP/0/0/CPU0:XRv13#show cef ipv4 14.14.14.14/32
14.14.14.14/32, version 479, internal 0x1000001 0x0 (ptr 0xa140f174) [1], 0x0
(0xa13f46a4), 0xa28 (0xa177c158)
  local adjacency 43.0.0.14
  Prefix Len 32, traffic index 0, precedence n/a, priority 3
    via 34.0.0.4, GigabitEthernet0/0/0/0.534, 5 dependencies, weight 0, class
0, backup [flags 0x300]
  path-idx 0 NHID 0x0 [0xa0e8f34c 0x0]
  next hop 34.0.0.4, PQ-node 5.5.5.5
  local adjacency
    local label 93015      labels imposed {4001 5004}
  via 43.0.0.14, GigabitEthernet0/0/0/0.543, 5 dependencies, weight 0, class
0, protected [flags 0x400]
  path-idx 1 bkup-idx 0 NHID 0x0 [0xa184c384 0xa184c1b4], parent-ifh 0x100
  next hop 43.0.0.14
    local label 93015      labels imposed {ImplNull}

```

Just for extra XR rLFA verification/practice, we will repeat the exercise to trace the LSP protecting the link between XRv13 and CSR4. Remember that XRv13 will build two tunnels to its PQ-node: one protects the link to XRv14 and one protects the link to CSR4. First we check the label bindings from LDP, then the OSPF RIB, then the FIB. We see that the label stack {94003 5002} contains two labels: the bottom label is 5002 which represents CSR5's label for 4.4.4.4/32, while the top label is 94003 which represents XRv14's label for 5.5.5.5/32. The RIB shows us the protected routing information and the FIB shows us the full label stack.

```

RP/0/0/CPU0:XRv13#show mpls ldp bindings 4.4.4.4/32 neighbor 5.5.5.5
4.4.4.4/32, rev 121
  Local binding: label: 93009
  Remote bindings: (3 peers)
    Peer          Label
    -----
    5.5.5.5:0      5002

```

```

RP/0/0/CPU0:XRv13#show mpls ldp bindings 5.5.5.5/32 neighbor 14.14.14.14
5.5.5.5/32, rev 164
  Local binding: label: 93002
  Remote bindings: (3 peers)
    Peer          Label

```

14.14.14.14:0

94003

```
RP/0/0/CPU0:XRv13#show ospf routes 4.4.4.4/32 backup-path detail
OSPF Route entry for 4.4.4.4/32
  Route type: Intra-area
  Metric: 2
  SPF priority: 4,   SPF version: 134
  RIB version: 0,   Source: Unknown
    34.0.0.4, from 4.4.4.4, via GigabitEthernet0/0/0/0.534, path-id 1
      Backup path: Remote, LFA: 5.5.5.5
        43.0.0.14, from 4.4.4.4, via GigabitEthernet0/0/0/0.543,
protected bitmap 0000000000000001
      Attributes: Metric: 2, SRLG Disjoint

RP/0/0/CPU0:XRv13#show cef ipv4 4.4.4.4/32
4.4.4.4/32, version 481, internal 0x1000001 0x0 (ptr 0xa140ee74) [1], 0x0
(0xa13f4368), 0xa28 (0xa177c1b0)
  local adjacency 34.0.0.4
  Prefix Len 32, traffic index 0, precedence n/a, priority 3
    via 34.0.0.4, GigabitEthernet0/0/0/0.534, 5 dependencies, weight 0, class
0, protected [flags 0x400]
      path-idx 0 bkup-idx 1 NHID 0x0 [0xa184c29c 0xa184c0cc], parent-ifh 0x100
      next hop 34.0.0.4
      local label 93009      labels imposed {ImplNull}
    via 43.0.0.14, GigabitEthernet0/0/0/0.543, 5 dependencies, weight 0, class
0, backup [flags 0x300]
      path-idx 1 NHID 0x0 [0xa0e8f49c 0x0]
      next hop 43.0.0.14, PQ-node 5.5.5.5
      local adjacency
      local label 93009      labels imposed {94003 5002}
```

We will do one more test with XRV rLFA with OSPF for practice. We will revert to the pentagon topology for this, which means the router does not need to invoke an extended P-space calculation. **Note: Unlike IS-IS, we do not have the capability to filter rLFA targets by RID, but we can enable the process globally versus having to do it on every interface.** This should confirm that we have an LDP neighbor with CSR1, along with some new labels. The rLFA tunnel to CSR1 allows XRv13 to protect the path to XRv14's loopback, which is normally reachable via a direct link.

```
RP/0/0/CPU0:XRv13#show ospf route 14.14.14.14/32 backup-path
Topology Table for ospf 1 with ID 13.13.13.13
Codes: O - Intra area, O IA - Inter area
      O E1 - External type 1, O E2 - External type 2
      O N1 - NSSA external type 1, O N2 - NSSA external type 2
O      14.14.14.14/32, metric 2
      43.0.0.14, from 14.14.14.14, via GigabitEthernet0/0/0/0.543, path-id 1
      Backup path: Remote, LFA: 1.1.1.1
```

34.0.0.4, from 14.14.14.14, via GigabitEthernet0/0/0/0.534,
protected bitmap 0000000000000001

RP/0/0/CPU0:XRv13#show mpls ldp neighbor 1.1.1.1

Peer LDP Identifier: 1.1.1.1:0

TCP connection: 1.1.1.1:646 - 13.13.13.13:24247

Graceful Restart: No

Session Holdtime: 180 sec

State: Oper; Msgs sent/rcvd: 14/17; Downstream-Unsolicited

Up time: 00:03:33

LDP Discovery Sources:

IPv4: (1)

Targeted Hello (13.13.13.13 -> 1.1.1.1, active/passive)

IPv6: (0)

Addresses bound to this peer:

IPv4: (3)

1.1.1.1 12.0.0.1 14.0.0.1

IPv6: (0)

We now learn CSR1's label for 14.14.14.14/32 which is 1007. This is the bottom label in the rLFA tunnel stack. The top label is whatever CSR4's label is for 1.1.1.1/32 (the rLFA endpoint), which is 4000. Thus, the rLFA label stack is {4000 1007}. After OSPF converges, XRv13 will use label 4004 to reach XRv14's loopback as this is the ordinary LDP LSP, using labels learned from the IGP nexthop (CSR4).

RP/0/0/CPU0:XRv13#show mpls ldp bindings 14.14.14.14/32
14.14.14.14/32, rev 186

Local binding: label: 93000

Remote bindings: (3 peers)

Peer	Label
1.1.1.1:0	1007
4.4.4.4:0	4004
14.14.14.14:0	ImpNull

RP/0/0/CPU0:XRv13#show mpls ldp bindings 1.1.1.1/32
1.1.1.1/32, rev 214

Local binding: label: 93004

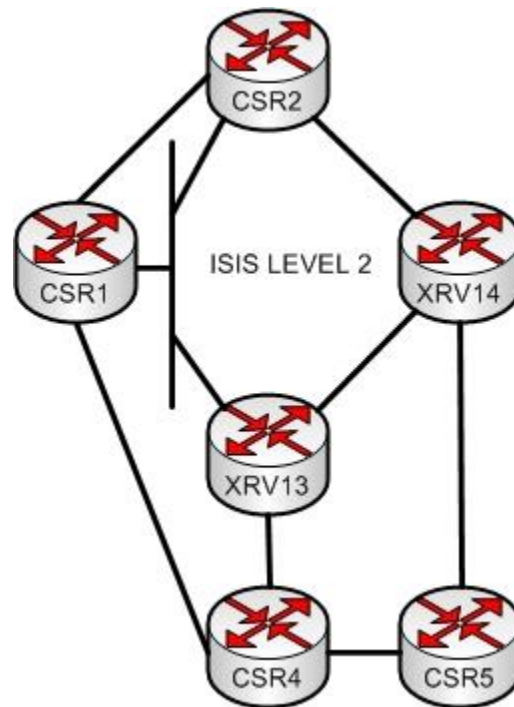
Remote bindings: (3 peers)

Peer	Label
1.1.1.1:0	ImpNull
4.4.4.4:0	4000
14.14.14.14:0	94002

37.1.2 IS-IS

37.1.2.1 Direct LFA

The LFA topology for IS-IS is identical to the one used for OSPF, except we set the OSPF AD to 255 to remove its routes from the RIB. The LFA process is very similar between OSPF and IS-IS so this test will focus on the differences. The diagram is shown again below.



General notes:

- On XE, IPv6 LFA for OSPFv3 and IS-IS both appear unsupported. Although the configuration is generic to the IS-IS process and not to an AF, the IS-IS IPv6 RIB doesn't show LFA paths.
- IS-IS does not have a concept of "broadcast interface disjoint", so that isn't a tiebreaker.
- When you enable LFA in XE, you must select an IS-IS level, unlike OSPF where the area is optional. In XR, the level is optional.
- IS-IS does not appear to have a show command to list the tie-breaker sequence.
- IS-IS considers loopbacks (/32) to be "medium" priority by default.

Like the OSPFv2 direct LFA test, I've increased CSR1's metric on the LAN to CSR2/XRv13 and the P2P link to CSR4 to 15. CSR1 prefers the P2P path via CSR2 to reach XRv14's loopback, 14.14.14.14/32. The repair path via XRv13 is downstream (XRv13 is closer to XRv14 than CSR1 is), node-protecting (we don't go through CSR2 on the repair-path), and SRLG-disjoint (no SRLGs configured yet).

```

! CSR1
interface GigabitEthernet2.513
 isis metric 15

interface GigabitEthernet2.514
 isis metric 15

```

```

CSR1#show isis rib 14.14.14.14
IPv4 local RIB for IS-IS process 1
IPv4 unicast topology base (TID 0, TOPOID 0x0) =====
Repair path attributes:
    DS - Downstream, LC - Linecard-Disjoint, NP - Node-Protecting
    PP - Primary-Path, SR - SRLG-Disjoint

Routes under majornet 14.0.0.0/8:

14.14.14.14/32
  [115/L2/20] via 12.0.0.2(GigabitEthernet2.512), from 14.14.14.14, tag 0, LSP[6/25]
              (installed)
    repair path: 123.0.0.13(GigabitEthernet2.513) metric:25 (DS,NP,SR) LSP[3]
  [115/L2/25] via 123.0.0.13(GigabitEthernet2.513), from 13.13.13.13, tag 0, LSP[3/28]

```

If we remove XRv13 by shutting down its LAN interface, CSR1 will install the repair path via CSR2 through the LAN, but it is not longer node-protecting since we rely on CSR2 for primary and repair paths. This is less ideal and is the reason IS-IS picked the XRv13 repair path over the CSR2 repair path in the first place. Notice the extra path via CSR4 at the bottom; this is like an "ignored" but valid backup path.

```

! XRv13
interface GigabitEthernet0/0/0/0.513
 shutdown

CSR1#show isis rib 14.14.14.14 | begin 14\.14
14.14.14.14/32
  [115/L2/20] via 12.0.0.2(GigabitEthernet2.512), from 14.14.14.14, tag 0, LSP[6/28]
              (installed)
    repair path: 123.0.0.2(GigabitEthernet2.513) metric:25 (DS,SR) LSP[6]
  [115/L2/35] via 14.0.0.4(GigabitEthernet2.514), from 13.13.13.13, tag 0, LSP[3/31]

```

Let's force the backup path to use CSR4 versus CSR2 or XRv13. We bring XRv13 back into the network and set SRLGs on CSR1's interface to CSR2 (P2P) and LAN. Now, the backup path through CSR4 is preferred due to the SRLG being disjoint. Node-protection is achieved since CSR2 is not in CSR4's path to XRv14. Notice that CSR4 is not downstream when compared to CSR1 since both of their metrics to XRv14 are 20.

```

! CSR1
interface GigabitEthernet2.512
 srlg gid 222

interface GigabitEthernet2.513
 srlg gid 222

! XRv13
interface GigabitEthernet0/0/0/0.513
 no shutdown

```

```

CSR1#show isis rib 14.14.14.14 | begin 14\.14
14.14.14.14/32
  [115/L2/20] via 12.0.0.2(GigabitEthernet2.512), from 14.14.14.14, tag 0, LSP[6/29]
              (installed)
              repair path: 14.0.0.4(GigabitEthernet2.514) metric:35 (NP,SR) LSP[3]
  [115/L2/25] via 123.0.0.13(GigabitEthernet2.513), from 13.13.13.13, tag 0, LSP[3/32]

```

The next example changes CSR1's interface costs again and removes the SRLGs. The P2P link to CSR2 is changed to 15 and the LAN remains 15, while the cost to CSR4 is set to 10. We have three-way ECMP but one of the paths (via XRv13 LAN) is listed as a backup for the CSR P2P link path. We see a new option, primary-path, which indicates that the repair-path is also an ECMP path already used for forwarding.

```

! CSR1
interface GigabitEthernet2.512
  isis metric 15
  no srlg gid 222

interface GigabitEthernet2.513
  no srlg gid 222

interface GigabitEthernet2.514
  isis metric 10

```

```

CSR1#show isis rib 14.14.14.14 | begin 14\.14
14.14.14.14/32
  [115/L2/25] via 123.0.0.13(GigabitEthernet2.513), from 13.13.13.13, tag 0, LSP[4/42]
              (installed)
  [115/L2/25] via 123.0.0.13(GigabitEthernet2.513), from 14.14.14.14, tag 0, LSP[3/42]
  [115/L2/25] via 12.0.0.2(GigabitEthernet2.512), from 14.14.14.14, tag 0, LSP[3/42]
              (installed)
              repair path: 123.0.0.13(GigabitEthernet2.513) metric:25 (PP,DS,NP,SR) LSP[3]
  [115/L2/25] via 123.0.0.2(GigabitEthernet2.513), from 14.14.14.14, tag 0, LSP[3/42]
              (installed)

```

We can also check the repair path to CSR5 from here. CSR1 should prefer CSR4 based on lowest metric but uses the P2P path via CSR2 as a repair-path. CSR2 is not downstream since both routers have a metric of 20 to reach 5.5.5.5/32, but it is outside of the primary path (node protecting) and does not share an SRLG.

```

CSR1#show isis rib 5.5.5.5 | begin 5\.5
5.5.5.5/32
  [115/L2/20] via 14.0.0.4(GigabitEthernet2.514), from 5.5.5.5, tag 0, LSP[5/39]
              (installed)
              repair path: 12.0.0.2(GigabitEthernet2.512) metric:35 (NP,SR) LSP[5]

```

Shutting down CSR2's P2P link to CSR1 forces the repair path through the CSR2 LAN hop. Considering the attributes of this path are identical to those of the P2P link, I am assuming IS-IS prefers P2P over multi-

access links. Cisco mentions LFA doesn't work on IS-IS broadcast networks, but this might be a hardware-based limitation since the control-plane has shown multi-access-facing LFAs in earlier tests.

```
CSR1#show isis rib 5.5.5.5 | begin 5\.5
5.5.5.5/32
  [115/L2/20] via 14.0.0.4(GigabitEthernet2.514), from 5.5.5.5, tag 0, LSP[5/41]
              (installed)
              repair path: 123.0.0.2(GigabitEthernet2.513) metric:35 (NP,SR) LSP[5]
```

Completely removing CSR2 from the network, CSR1 now prefers XRv3 via the LAN. This is less preferable because it does not offer node protection as XRv13 has an ECMP path to CSR5 that routes through CSR4.

```
! CSR2
interface GigabitEthernet2
 shutdown

CSR1#show isis rib 5.5.5.5 | begin 5\.5
5.5.5.5/32
  [115/L2/20] via 14.0.0.4(GigabitEthernet2.514), from 5.5.5.5, tag 0, LSP[5/40]
              (installed)
              repair path: 123.0.0.13(GigabitEthernet2.513) metric:35 (SR) LSP[5]
```

Of course, we can change XRv13's costs so that it prefers to route through XRv14 to reach CSR5. Reducing the metric on the XRv13 link facing XRv14 to 5 should force traffic through XRv14 only. We should achieve node protection (and downstream status) for this backup path now. The metric reduction from 35 to 30 also shows as the backup path's cost follows XRv13's best path through XRv14. In summary, IS-IS LFA behaves almost identically to OSPF LFA.

```
! XRv13
router isis 1
 interface GigabitEthernet0/0/0/0.534
   address-family ipv4 unicast
   metric 5

CSR1#show isis rib 5.5.5.5 | begin 5\.5
5.5.5.5/32
  [115/L2/20] via 14.0.0.4(GigabitEthernet2.514), from 5.5.5.5, tag 0, LSP[13/17]
              (installed)
              repair path: 123.0.0.13(GigabitEthernet2.513) metric:30 (DS,NP,SR) LSP[13]
```

Next, we quickly examine direct LFA on IOS-XR. Both IPv4 and IPv6 are supported, which is an improvement over XE. I have enabled LFA for both AFs on XRv13 but IPv6 will be analyzed later. The configuration is highly redundant since the feature must be enabled under each interface per-AFI.

```
! XRv13
router isis 1
 interface GigabitEthernet0/0/0/0.513
```

```

address-family ipv4 unicast
  fast-reroute per-prefix
address-family ipv6 unicast
  fast-reroute per-prefix

interface GigabitEthernet0/0/0/0.534
  address-family ipv4 unicast
    fast-reroute per-prefix
  address-family ipv6 unicast
    fast-reroute per-prefix

interface GigabitEthernet0/0/0/0.543
  address-family ipv4 unicast
    fast-reroute per-prefix
  address-family ipv6 unicast
    fast-reroute per-prefix

```

Earlier we changed the IPv4 metrics so that XRv13 routes via XRv14 to reach 5.5.5.5/32. It now has an LFA via CSR4; the "detail" keyword lets us see which path attributes apply. This path is not a primary-path (P), total metric is 20 (TM), is not linecard disjoint (LC), is node protecting (NP), is downstream (D), and is SRLG disjoint. The "SRLG: Yes" phrase is confusing because XR is really saying "SRLG is absent", not "SRLG is present". The output also shows us that the default prefix-priority of this loopback is medium, not high like XE. Default tie-breakers for XR IS-IS LFA are as follows. Notice that SRLG is not considered by default.

10 Primary path

20 Lowest metric

30 Line card disjoint

40 Node protection

```

RP/0/0/CPU0:XRv13#show isis ipv4 fast-reroute 5.5.5.5/32 detail
L2 5.5.5.5/32 [15/115] medium priority
   via 43.0.0.14, GigabitEthernet0/0/0/0.543, XRv14, Weight: 0
   FRR backup via 34.0.0.4, GigabitEthernet0/0/0/0.534, CSR4, Weight: 0
   P: No, TM: 20, LC: No, NP: Yes, D: Yes, SRLG: Yes
src CSR5.00-00, 5.5.5.5

```

Next, we will use SRLG to see if we can select the path through CSR1 instead of CSR4. Below is the SRLG configuration. XR SRLG configuration is nice because you can use descriptive names or values. We also have to tell IS-IS to use the SRLG in its computations. I also adjust SRLG to be the most important tie-breaker for IS-IS LFA, which is specific to level-2.

```

! XRv13
srlg
  name SAME_FIBER_PAIR value 134
interface GigabitEthernet0/0/0/0.534

```

```

name SAME_FIBER_PAIR

interface GigabitEthernet0/0/0/0.543
name SAME_FIBER_PAIR

router isis 1
address-family ipv4 unicast
fast-reroute per-prefix tiebreaker srlg-disjoint index 1 level 2

```

Now, the LFA is via CSR1, which is not downstream (CSR1 cost is 20, XRv13 is 15), but is node protecting and SRLG disjoint.

```

RP/0/0/CPU0:XRv13#show isis fast-reroute 5.5.5.5/32 detail

L2 5.5.5.5/32 [15/115] medium priority
  via 43.0.0.14, GigabitEthernet0/0/0/0.543, XRv14, Weight: 0
  FRR backup via 123.0.0.1, GigabitEthernet0/0/0/0.513, CSR1, Weight: 0
    P: No, TM: 30, LC: No, NP: Yes, D: No, SRLG: Yes
src CSR5.00-00, 5.5.5.5

```

Last, we modify the IS-IS path selection to make lowest-metric the most important tie-breaker with SRLG as the second most important. The original backup path via CSR4 should be preferred since its total metric is lower (20 versus 30), but notice that this path is not SRLG disjoint. As a reminder, the text “SRLG: No” indicates that the SRLG is present and the LFA is not SRLG-disjoint.

```

! XRv13
router isis 1
address-family ipv4 unicast
fast-reroute per-prefix tiebreaker lowest-backup-metric index 1 level 2
fast-reroute per-prefix tiebreaker srlg-disjoint index 2 level 2

RP/0/0/CPU0:XRv13#show isis fast-reroute 5.5.5.5/32 detail
L2 5.5.5.5/32 [15/115] medium priority
  via 43.0.0.14, GigabitEthernet0/0/0/0.543, XRv14, Weight: 0
  FRR backup via 34.0.0.4, GigabitEthernet0/0/0/0.534, CSR4, Weight: 0
    P: No, TM: 20, LC: No, NP: Yes, D: Yes, SRLG: No
src CSR5.00-00, 5.5.5.5

```

In summary, direct (or local) LFA is very similar between IS-IS and OSPF, also between XE and XR. The exact tiebreakers and their sequential arrangements vary slightly between platforms and protocols.

37.1.2.2 Remote LFA

Remote LFA (rLFA) for IS-IS is very similar to OSPF. We still require MPLS encapsulation with LDP signaling regardless of the IGP in play. Because the mechanics of rLFA (P-space, extended P-space, Q-space, and PQ-nodes) were discussed already, it is not discussed again because the concepts are the same. The testing below just demonstrates the feature using IS-IS. We will use the same pentagon

topology as we used in OSPF, ensuring that we bind IS-IS to BFD on that P2P link between CSR1 and CSR2. All interfaces have IS-IS metric of 10. For brevity we will not demonstrate the square topology as the configuration and verification is identical.

A quick verification of BFD is in order because **without BFD, LFA data-plane operations do not work**. This test is targeting the CSR1-CSR2 link as the protected link, so ensuring BFD is enabled on that link is a good first step.

```
CSR1#show bfd neighbors client isis
IPv4 Sessions
NeighAddr                LD/RD                RH/RS                State                Int
12.0.0.2                4097/4097                Up                Up                Gi2.512
```

We enable rLFA on CSR1 under the IS-IS process. We must specify an IS-IS level, which is different than OSPF. In OSPF, we could optionally specify an area, and failing to specify an area would protect all areas as well as external routes. IS-IS is less generic and requires the rLFA feature specified for each level. The output below is very terse and there is no "detail" option, but gives quick insight. We don't really care about Lfa24 because we are trying to protect the link to CSR2, which implies an rLFA tunnel to XRv13. If we were trying to protect the link to CSR4, the rLFA tunnel to XRv14 would matter.

```
! CSR1
router isis 1
 fast-reroute remote-lfa level-2 mpls-ldp maximum-metric 20
```

```
CSR1#show isis fast-reroute remote-lfa tunnels
Tag 1 - Fast-Reroute Remote-LFA Tunnels:
 MPLS-Remote-Lfa23: use Gi2.514, nexthop 14.0.0.4, end point 13.13.13.13
 MPLS-Remote-Lfa24: use Gi2.512, nexthop 12.0.0.2, end point 14.14.14.14
```

We should verify that we have an LDP neighbor with the PQ-node, XRv13, and that we learn its LDP local-label for 2.2.2.2/32. This prefix is the final destination that we are trying to protect. Forming an LDP session with XRv13 assumes that XRv13 is accepting tLDP sessions from CSR1 (configured earlier). The bottom label in the rLFA tunnel will be 93003. Also note that label 4007 will be used for normal forwarding once the rLFA tunnel is torn down after IS-IS reconverges. The top label is the rLFA tunnel endpoint, which is towards 13.13.13.13/32; the label value is learned by CSR4 with a label of 4001.

```
CSR1#show mpls ldp neighbor 13.13.13.13
Peer LDP Ident: 13.13.13.13:0; Local LDP Ident 1.1.1.1:0
TCP connection: 13.13.13.13.48869 - 1.1.1.1.646
State: Oper; Msgs sent/rcvd: 23/24; Downstream
Up time: 00:06:33
LDP discovery sources:
 Targeted Hello 1.1.1.1 -> 13.13.13.13, active
Addresses bound to peer LDP Ident:
13.13.13.13      34.0.0.13      43.0.0.13
```

```

CSR1#show mpls ldp bindings 2.2.2.2 32
lib entry: 2.2.2.2/32, rev 4
    local binding:  label: 1004
    remote binding: lsr: 2.2.2.2:0, label: imp-null
    remote binding: lsr: 4.4.4.4:0, label: 4007
    remote binding: lsr: 13.13.13.13:0, label: 93003

CSR1#show mpls ldp bindings 13.13.13.13 32 neighbor 4.4.4.4
lib entry: 13.13.13.13/32, rev 10
    remote binding: lsr: 4.4.4.4:0, label: 4001

```

We still have MPLS FNF configured on CSR4 facing CSR1 from the OSPF tests. When we send traffic from CSR1 to CSR2's loopback, then break the P2P link between them (breakage not shown), we see a short burst of traffic with label stack {4001 93003} which is rLFA-encapsulated traffic shown in green. Traffic with label 4007 is used thereafter for normal forwarding when IS-IS converges, shown in yellow. Notice that after 119 packets, the rLFA is no longer used, and the traffic uses the regular LDP signaled LSP for transport.

```

CSR4#show flow monitor FNF_MPLS_MONITOR cache format table | begin MPLS
MPLS LABEL 1  MPLS LABEL 2      pkts
=====

```

MPLS LABEL 1		MPLS LABEL 2		pkts
4007	/1	0	/0	5
4001	/1	93003	/1	119

```

CSR4#show flow monitor FNF_MPLS_MONITOR cache format table | begin MPLS
MPLS LABEL 1  MPLS LABEL 2      pkts
=====

```

MPLS LABEL 1		MPLS LABEL 2		pkts
4007	/1	0	/0	33
4001	/1	93003	/1	119

Because BFD is not supported on XRv, we will limit our rLFA tests to control-plane verifications. We will enable rLFA on XRv13 to protect the path to 14.14.14.14/32 (protect the link between XRv13 and XRv14). rLFA for IS-IS in XR is enabled at the link-level; the AF-level command is used to filter PQ-node candidacy based on RID with a prefix-list. I added this filter to the configuration just to test the feature; I don't care about backing up the path to 4.4.4.4/32 so I do not need an rLFA tunnel to CSR2 for this test, only CSR1. This can help scale rLFA since building too many tunnels to protect links unnecessarily may resource-intensive.

```

! XRv13
ipv4 prefix-list PL_CSR1
10 permit 1.1.1.1/32

router isis 1
address-family ipv4 unicast
fast-reroute per-prefix remote-lfa prefix-list PL_CSR1

```

```

interface GigabitEthernet0/0/0/0.513
  address-family ipv4 unicast
    fast-reroute per-prefix remote-lfa tunnel mpls-ldp level 2
    fast-reroute per-prefix remote-lfa maximum-metric 20

interface GigabitEthernet0/0/0/0.534
  address-family ipv4 unicast
    fast-reroute per-prefix remote-lfa tunnel mpls-ldp level 2
    fast-reroute per-prefix remote-lfa maximum-metric 20

interface GigabitEthernet0/0/0/0.543
  address-family ipv4 unicast
    fast-reroute per-prefix remote-lfa tunnel mpls-ldp level 2
    fast-reroute per-prefix remote-lfa maximum-metric 20

```

The output below shows that XRv14's loopback is backed up by an rLFA tunnel terminating on CSR1. Notice that the total metric (TM) is the cost to the tunnel endpoint, not the entire path.

```

RP/0/0/CPU0:XRv13#show isis fast-reroute 14.14.14.14/32 detail
L2 14.14.14.14/32 [10/115] medium priority
    via 43.0.0.14, GigabitEthernet0/0/0/0.543, XRv14, Weight: 0
    Remote FRR backup via CSR1 [1.1.1.1], via 34.0.0.4,
GigabitEthernet0/0/0/0.534 CSR4, Weight: 0
    P: No, TM: 20, LC: No, NP: No, D: No, SRLG: No
    src XRv14.00-00, 14.14.14.14

```

```

RP/0/0/CPU0:XRv13#show mpls ldp neighbor 1.1.1.1
Peer LDP Identifier: 1.1.1.1:0
  TCP connection: 1.1.1.1:646 - 13.13.13.13:51129
  Graceful Restart: No
  Session Holdtime: 180 sec
  State: Oper; Msgs sent/rcvd: 13/13; Downstream-Unsolicited
  Up time: 00:02:32
  LDP Discovery Sources:
    IPv4: (1)
      Targeted Hello (13.13.13.13 -> 1.1.1.1, active/passive)
    IPv6: (0)
  Addresses bound to this peer:
    IPv4: (3)
      1.1.1.1          12.0.0.1          14.0.0.1
    IPv6: (0)

```

Next, we determine the label stacks required if this rLFA tunnel were to be used. We learn CSR1's label for XRv14's loopback is 1007 which would be the bottom label in the rLFA stack. Label 4004 would be the post-IGP convergence label we would use after the rLFA tunnel is no longer needed. Last, we need to know the transport label to reach the rLFA tunnel endpoint, which is 1.1.1.1/32. We have label 4000

from CSR4 that directs us there. The rLFA stack would be {4000 1007} and the label used after IS-IS converges is 4004.

```
RP/0/0/CPU0:XRv13#show mpls ldp bindings 14.14.14.14/32
```

```
14.14.14.14/32, rev 186
```

```
Local binding: label: 93000
```

```
Remote bindings: (3 peers)
```

Peer	Label
1.1.1.1:0	1007
4.4.4.4:0	4004
14.14.14.14:0	ImpNull

```
RP/0/0/CPU0:XRv13#show mpls ldp bindings 1.1.1.1/32 neighbor 4.4.4.4
```

```
1.1.1.1/32, rev 214
```

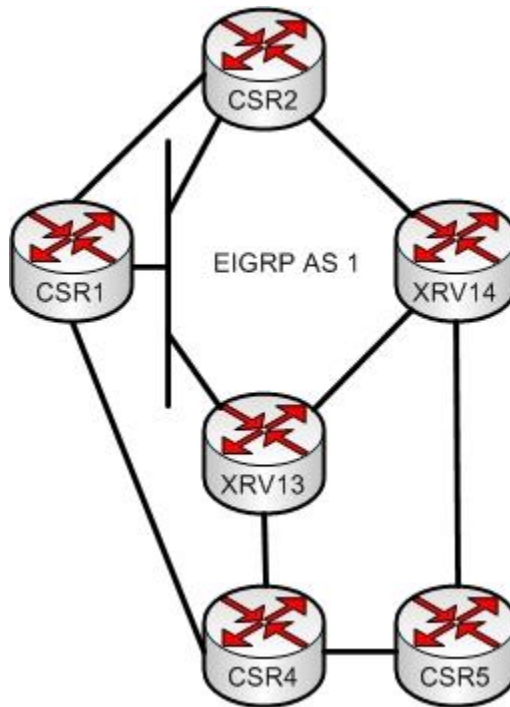
```
Local binding: label: 93004
```

```
Remote bindings: (3 peers)
```

Peer	Label
4.4.4.4:0	4000

37.1.3 EIGRP

EIGRP LFA is the simplest topic in this section. All EIGRP LFA does is take feasible successors and pre-install them into the FIB for slightly faster convergence. EIGRP is not actually doing more work than it used to, and only feasible successor (FS) paths are candidates for FRR. After all, they are guaranteed to be LFAs already, unlike OSPF and IS-IS which required additional logic. This implies that all EIGRP LFA paths must be downstream; in OSPF and IS-IS we saw that backup paths did not need to be downstream because the link-state nature of those protocols could run SPF as their neighbors. EIGRP does not have that intelligence, so meeting the feasibility condition (which is effectively mandating “downstream”) is required. In a virtual platform, EIGRP LFA probably does not have a significant performance impact since there are not hardware linecards or TCAM components to program. We will use the same topology we used earlier for EIGRP FRR with IPv4. IPv6 does not appear supported at this time, nor is there a concept of remote LFA in EIGRP. EIGRP LFA is also not supported in XR. As a result, the EIGRP FRR options are much less comprehensive than OSPF and IS-IS. For this test, only interface delay is considered, which effectively allows us to mimic IS-IS and OSPF path selection mechanisms; using bandwidth makes the EIGRP calculation more complicated since it's evaluated per-path, not per-hop.



We will begin by evaluating CSR5's route to XRV13 with metric modifications. With FRR enabled in OSPF or IS-IS, we expect two ECMP paths via CSR4 and XRV14, each of which backs the other one up. In EIGRP, this concept does not apply, and we only see the two ECMP paths as we normally would. This is acceptable since the paths are both installed in the FIB anyway (primary paths used for ECMP), so EIGRP need not identify them as mutually-supporting LFAs.

```

! CSR5
router eigrp LFA_TEST
 address-family ipv4 unicast autonomous-system 1
   topology base
     fast-reroute per-prefix all

```

```

CSR5#show eigrp address-family ipv4 topology frr | section 13\.13
P 13.13.13.13/32, 2 successors, FD is 3997696
  via 45.0.0.4 (3997696/720896), GigabitEthernet2.545
  via 54.0.0.14 (3997696/720896), GigabitEthernet2.554

```

Increasing the delay on CSR5's link to XRV14 will allow CSR5 to prefer the path via CSR4 as primary. The path through XRV14 is an LFA because it is an EIGRP feasible successor (XRV14 is closer to XRV13 than CSR5 is, since $720896 < 3997696$). OSPF and IS-IS call this "downstream" which is a more generic term of expressing the feasibility condition in EIGRP (same concept).

```

CSR5#show eigrp address-family ipv4 topology frr | section 13\.13
P 13.13.13.13/32, 1 successors, FD is 3997696
  via 45.0.0.4 (3997696/720896), GigabitEthernet2.545

```



```
via 54.0.0.14 (7274496/720896), GigabitEthernet2.554, [LFA]
```

Next, we will set the delay of all EIGRP interfaces to 5 to make traffic engineering a little easier (not shown). Using EIGRP named-mode on XE displays the delay in picoseconds versus microseconds (ms was used in classic EIGRP). Note: A quick conversion from ps to ms is to trim the last 6 zeroes of the ps field. If we increase the delay value on XRv14's link to XRv13 to 10, then CSR5 and XRv14 are equidistant from XRv13. The EIGRP feasibility condition (or downstream condition) is not satisfied, and the alternate path via XRv14 path is not an LFA. OSPF and IS-IS are more intelligent, and given their link-state information they can still use non-downstream paths in some cases (unlike when FD = RD, which is 6619136 in this case). Here, we prove that EIGRP's FS condition remains inflexible despite the LFA feature.

```
CSR5#show eigrp address-family ipv4 topology frr | section 13\.13
P 13.13.13.13/32, 1 successors, FD is 3997696
    via 45.0.0.4 (6619136/3342336), GigabitEthernet2.545, serno 152
    via 54.0.0.14 (9895936/6619136), GigabitEthernet2.554
```

EIGRP also has a limited number of tie-breakers, such as interface-disjoint, linecard-disjoint, lowest metric, and SRLG-disjoint. We will test interface-disjoint as the tie-breaker since we have a LAN interface. We will use CSR1 for this demonstration as it has additional path diversity. All interfaces have been reset to delay 5 as a baseline. Initially, CSR1 does 3-way load sharing to XRv14's loopback via CSR2 LAN, CSR2 P2P, and XRv13 LAN. Changing XRv13's cost to XRv14 to 4 makes it the best path. However, rather than prefer CSR2 over the LAN, CSR1 prefers CSR2 via the P2P link to be interface-disjoint. The CSR2 LAN link is definitely a feasible successor (LFA) but was not preferred. Notice that CSR4 is not a candidate LFA because its RD equals CSR1's FD (5963776). Also note that we did not need to explicitly configure this tie-breaker since its enabled within EIGRP and cannot be disabled.

```
! XRv3
router eigrp LFA_TEST
 address-family ipv4
   interface GigabitEthernet0/0/0/0.543
     metric delay 4
```

```
CSR1#show eigrp address-family ipv4 topology frr | sec 14\.14
P 14.14.14.14/32, 1 successors, FD is 5963776
    via 123.0.0.13 (5963776/2686976), GigabitEthernet2.513
    via 12.0.0.2 (6619136/3342336), GigabitEthernet2.512, [LFA]
    via 123.0.0.2 (6619136/3342336), GigabitEthernet2.513
    via 14.0.0.4 (9240576/5963776), GigabitEthernet2.514
```

Increasing CSR1's cost on the CSR2 P2P link increases the metric to 14.14.14.14/32 via that link, but it's still the preferred LFA. In the previous example, both of these FS paths had the same FD, but the interface-disjoint characteristic served as the tie-breaker. This proves that interface-disjoint is a higher tie-breaker than lowest metric.

```

! CSR1
interface GigabitEthernet2.512
  delay 6

CSR1#show eigrp address-family ipv4 topology frr | sec 14\.14
P 14.14.14.14/32, 1 successors, FD is 5963776
  via 123.0.0.13 (5963776/2686976), GigabitEthernet2.513
  via 12.0.0.2 (7274496/3342336), GigabitEthernet2.512, [LFA]
  via 123.0.0.2 (6619136/3342336), GigabitEthernet2.513
  via 14.0.0.4 (9240576/5963776), GigabitEthernet2.514

```

When we configure lowest metric as the #1 tie-breaker, the LFA via CSR2 LAN is used. None of the metrics have changed, but the order in which tie-breakers are evaluated allowed us to select a new LFA. This tie-breaker prioritization process works identically to OSPF and IS-IS.

```

! CSR1
router eigrp LFA_TEST
  address-family ipv4 unicast autonomous-system 1
  topology base
  fast-reroute tie-break lowest-backup-path-metric 1

CSR1#show eigrp address-family ipv4 topology frr | sec 14\.14
P 14.14.14.14/32, 1 successors, FD is 5963776
  via 123.0.0.13 (5963776/2686976), GigabitEthernet2.513
  via 12.0.0.2 (7274496/3342336), GigabitEthernet2.512
  via 123.0.0.2 (6619136/3342336), GigabitEthernet2.513, [LFA]
  via 14.0.0.4 (9240576/5963776), GigabitEthernet2.514

```

Let's reset the network to baseline delays of 5 (not shown) and test the SRLG feature. The only noteworthy metric changes are CSR1's link to CSR4 has delay 2 and CSR4's link to CSR5 has delay 3. This allows CSR1 to do 4-way load sharing to XRv14 as all paths have a cumulative delay value of 11 (XRv4's loopback counts as delay 1).

```

! CSR1
interface GigabitEthernet2.514
  delay 2

! CSR4
interface GigabitEthernet2.545
  delay 3

CSR1#show ip eigrp topology frr | section 14\.14
P 14.14.14.14/32, 4 successors, FD is 5963776
  via 12.0.0.2 (6619136/3342336), GigabitEthernet2.512
  via 123.0.0.2 (6619136/3342336), GigabitEthernet2.513
  via 123.0.0.13 (6619136/3342336), GigabitEthernet2.513
  via 14.0.0.4 (6619136/5308416), GigabitEthernet2.514

```

We now reduce CSR1's delay to CSR2 on the P2P link to 4 so it becomes the best path, but there are three candidate LFAs all with equal cost. CSR1 selects XRv13 but the evaluation criteria for this decision is unclear. My initial guess is because XRv13 has the higher IP address.

```
! CSR1
interface GigabitEthernet2.512
  delay 4
```

```
CSR1#show ip eigrp topology frr | section 14\.14
P 14.14.14.14/32, 1 successors, FD is 5963776
    via 12.0.0.2 (5963776/3342336), GigabitEthernet2.512
    via 123.0.0.2 (6619136/3342336), GigabitEthernet2.513
    via 123.0.0.13 (6619136/3342336), GigabitEthernet2.513, [LFA]
    via 14.0.0.4 (6619136/5308416), GigabitEthernet2.514
```

Shutting down the LAN interface on XRv13 causes the router to pick CSR4. Despite having a lower IP address than 123.0.0.2, 14.0.0.4 was likely chosen due to offering node-protection, where 123.0.0.2 is the same router as 12.0.0.2 (successor path). Clearly the RD is not being evaluated since CSR4's is the highest, but this was never a criteria for LFA paths provided the RD < FD (meets feasibility condition). Node-protection is not a term used in the EIGRP LFA documentation anywhere, so I cannot prove that node-protection is even evaluated by EIGRP. Considering it does not have a concept of "nodes" (no corresponding graph structure), node-protection in EIGRP is unlikely nonexistent.

```
! XRv13
interface GigabitEthernet0/0/0/0.513
  shutdown
```

```
CSR1#show ip eigrp topology frr | section 14\.14
P 14.14.14.14/32, 1 successors, FD is 5963776
    via 12.0.0.2 (5963776/3342336), GigabitEthernet2.512
    via 123.0.0.2 (6619136/3342336), GigabitEthernet2.513
    via 14.0.0.4 (6619136/5308416), GigabitEthernet2.514, [LFA]
```

Using SRLG, let's assume we want the router to always pick CSR4 over the LAN interfaces if possible. We apply the same SRLG to the CSR2 P2P link and the LAN interface. We must manually assign SRLG to the EIGRP tie-break list because it is not assigned by default. This is similar to IS-IS, but different than OSPF where SRLG is evaluated by default. Below, we add it as the second tiebreaker (lowest metric was the first). We also bring XRv13 back into the LAN. Without this SRLG configuration, XRv13 would immediately become the preferred LFA again. Using SRLGs in this way allows us to semi-administratively select CSR4 as the preferred LFA.

```
! CSR1
interface GigabitEthernet2.512
  srlg gid 45
```

```

interface GigabitEthernet2.513
  srlg gid 45

router eigrp LFA_TEST
  address-family ipv4 unicast autonomous-system 1
  topology base
  fast-reroute tie-break srlg-disjoint 2

! XRV13
interface GigabitEthernet0/0/0/0.513
  no shutdown

CSR1#show ip eigrp topology frr | section 14\.14
P 14.14.14.14/32, 1 successors, FD is 5963776
  via 12.0.0.2 (5963776/3342336), GigabitEthernet2.512
  via 123.0.0.13 (6619136/3342336), GigabitEthernet2.513
  via 123.0.0.2 (6619136/3342336), GigabitEthernet2.513
  via 14.0.0.4 (6619136/5308416), GigabitEthernet2.514, [LFA]

```

We can override the SRLG by making the cost to CSR4 less desirable. We increase the delay from 2 to 3 on CSR1's P2P link to CSR4 and now XRV13 is preferred again. CSR4 is still an EIGRP FS, and therefore a candidate LFA, but is rejected since lowest backup metric is the #1 tie-breaker. This path violates the SRLG configuration; SRLG is the #2 tie-breaker so EIGRP doesn't even evaluate SRLG since the tie was already broken in the previous step.

```

! CSR1
interface GigabitEthernet2.514
  delay 3

CSR1#show ip eigrp topology frr | section 14\.14
P 14.14.14.14/32, 1 successors, FD is 5963776
  via 12.0.0.2 (5963776/3342336), GigabitEthernet2.512
  via 123.0.0.13 (6619136/3342336), GigabitEthernet2.513, [LFA]
  via 123.0.0.2 (6619136/3342336), GigabitEthernet2.513
  via 14.0.0.4 (7274496/5308416), GigabitEthernet2.514

```

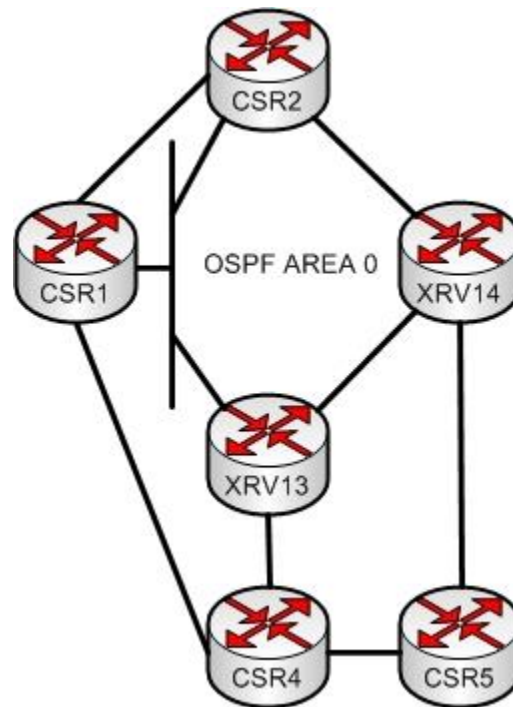
37.2 Loop Free Alternate (LFA) for IPv6 (XR Only)

This feature only seems to be supported on XR currently. Remote LFA for IPv6 is going to require LDPv6 which supposedly came out in XR 5.3.0, which is newer than what the CCIE lab will use. We will examine it briefly for completeness since IPv6 is a major focus area of the CCIE SP exam.

37.2.1 OSPFv3

37.2.1.1 Direct LFA

We will evaluate using direct LFA with OSPFv3 on XR. Be sure to adjust the administrative distances to make OSPF preferred over IS-IS if you were previously testing IS-IS. The network is shown again below for reference.



We will examine the network from XRV14's perspective for this test. We enable LFA under area 0 in per-prefix mode. Initially, there is a familiar situation with two ECMP paths that are backing each other up. These are ECMP paths from XRV14 to ::1:1:1:1/128, which is CSR1's IPv6 loopback prefix.

```

! XRV14
router ospfv3 1
 area 0
  fast-reroute per-prefix

```

```

RP/0/0/CPU0:XRV14#show ospfv3 routes ::1:1:1:1/128 backup-path
Topology Table for OSPFv3 1 with ID 14.14.14.14
* ::1:1:1:1/128, Intra, cost 2/0, area 0
  GigabitEthernet0/0/0/0.524, fe80::2, path-id 1
    Backup path:
      fe80::13, from 1.1.1.1, via GigabitEthernet0/0/0/0.543, protected
  bitmap 0x1
    Attributes: Metric: 2, Primary, Downstream, Node Protect, SRLG
  Disjoint
    GigabitEthernet0/0/0/0.543, fe80::13, path-id 2
      Backup path:
        fe80::2, from 1.1.1.1, via GigabitEthernet0/0/0/0.524, protected
  bitmap 0x2

```

Attributes: Metric: 2, Primary, Downstream, Node Protect, SRLG Disjoint

For consistency, we recycle an old SRLG configuration on XRv14 to verify SRLGs work with OSPFv3. We then increase the OSPFv3 cost to XRv13 so that CSR2 is the primary with CSR5 as the backup. We also achieve node protection as CSR2 is not in the backup path. XRv13 is node-protecting also, but would violate the SRLG policy.

```
! XRv14
srlg
 interface GigabitEthernet0/0/0/0.524
   8 value 654

 interface GigabitEthernet0/0/0/0.543
   8 value 654

router ospfv3 1
 fast-reroute per-prefix tiebreaker srlg-disjoint index 1
 area 0
 interface GigabitEthernet0/0/0/0.543
   cost 2

RP/0/0/CPU0:XRv14#show ospfv3 routes ::1:1:1:1/128 backup-path
Topology Table for OSPFv3 1 with ID 14.14.14.14
* ::1:1:1:1/128, Intra, cost 2/0, area 0
  GigabitEthernet0/0/0/0.524, fe80::2, path-id 1
    Backup path:
      fe80::5, from 1.1.1.1, via GigabitEthernet0/0/0/0.554, protected
bitmap 0x1
  Attributes: Metric: 3, Node Protect, SRLG Disjoint
```

Rather than repeat the exact set of tests we did for IPv4, we will test new things. Next, we will examine XRv14's path to CSR4's loopback. Due to the cost modification, XRv14 routes through CSR5 with a backup via XRv13. This is SRLG disjoint and node-protecting as expected.

```
RP/0/0/CPU0:XRv14#show ospfv3 routes ::4:4:4:4/128 backup-path
Topology Table for OSPFv3 1 with ID 14.14.14.14
* ::4:4:4:4/128, Intra, cost 2/0, area 0
  GigabitEthernet0/0/0/0.554, fe80::5, path-id 1
    Backup path:
      fe80::13, from 4.4.4.4, via GigabitEthernet0/0/0/0.543, protected
bitmap 0x1
  Attributes: Metric: 3, Downstream, Node Protect, SRLG Disjoint
```

Before continuing, we remove our OSPFv3 cost modification so XRv14 performs ECMP back to CSR4 with the primary paths repairing one another (output not shown). We will test a new technology that also applies to IPv4 but was not evaluated earlier. This is the usage of "lfa-candidate" and "use-candidate-

only". The former is used to identify interfaces that are LFA candidate interfaces (that is, allowed to host LFA paths exiting them) and the second command forces LFAs to use only those interfaces explicitly identified as LFA candidates. It's like a stricter version of SRLG since we can hard-code candidate interfaces and for LFA to follow those constraints. The configuration below shows our LFA parameters. We want to use the link via CSR2 for LFAs if possible.

```
! XRv14
router ospfv3 1
  fast-reroute per-prefix lfa-candidate interface GigabitEthernet0/0/0/0.524
  fast-reroute per-prefix use-candidate-only enable
```

After applying these changes, we still have two ECMP paths, but both are backed up via CSR2. This makes sense since only that interface is able to host LFA paths. When using CSR5, the backup path is SRLG disjoint (the SRLG is on the CSR2 and XRv13 links) and node-protecting. When routing through XRv13, the backup path through CSR2 is neither SRLG disjoint nor node protecting, which would normally make it a relatively undesirable LFA.

```
RP/0/0/CPU0:XRv14#show ospfv3 routes ::4:4:4:4/128 backup-path
Topology Table for OSPFv3 1 with ID 14.14.14.14
* ::4:4:4:4/128, Intra, cost 2/0, area 0
  GigabitEthernet0/0/0/0.554, fe80::5, path-id 1
    Backup path:
      fe80::2, from 4.4.4.4, via GigabitEthernet0/0/0/0.524, protected
  bitmap 0x3
    Attributes: Metric: 3, Node Protect, SRLG Disjoint
  GigabitEthernet0/0/0/0.543, fe80::13, path-id 2
    Backup path:
      fe80::2, from 4.4.4.4, via GigabitEthernet0/0/0/0.524, protected
  bitmap 0x3
    Attributes: Metric: 3,
```

The reason the path is not node protecting is because CSR2 does 3-way ECMP between its two links to CSR1 and its link to XRv13. Thus, if the primary path from XRv14 to CSR4 routes through XRv13, we cannot guarantee node protection if we install an LFA via CSR2. Though LFA is configured locally, the resulting backup paths are highly dependent on the network views from the adjacency nodes.

```
CSR2#show ipv6 route ::4:4:4:4
Routing entry for ::4:4:4:4/128
  Known via "ospf 1", distance 110, metric 2, type intra area
  Backup from "isis 1 [115]"
  Route count is 3/3, share count 0
  Routing paths:
    FE80::1, GigabitEthernet2.512
      Last updated 00:24:47 ago
    FE80::13, GigabitEthernet2.513
      Last updated 00:00:00 ago
```

```
FE80::1, GigabitEthernet2.513
Last updated 00:00:00 ago
```

We can engineer node protection by forcing CSR2 not to use its LAN interface to reach CSR4 by increasing the OSPFv3 cost to 2. As soon as we do that, we achieve node protection, since the LFA path (via CSR2) does not rely on XRv13 for reachability to CSR4. Instead, CSR2 routes via CSR1 to reach CSR4, bypassing XRv13 entirely. This short series of tests proves that LFA is very similar between IPv4 and IPv6.

```
! CSR2
interface GigabitEthernet2.513
  ospfv3 1 ipv6 cost 2

RP/0/0/CPU0:XRv14#show ospfv3 routes ::4:4:4:4/128 backup-path
Topology Table for OSPFv3 1 with ID 14.14.14.14
* ::4:4:4:4/128, Intra, cost 2/0, area 0
  GigabitEthernet0/0/0/0.554, fe80::5, path-id 1
    Backup path:
      fe80::2, from 4.4.4.4, via GigabitEthernet0/0/0/0.524, protected
  bitmap 0x3
    Attributes: Metric: 3, Node Protect, SRLG Disjoint
  GigabitEthernet0/0/0/0.543, fe80::13, path-id 2
    Backup path:
      fe80::2, from 4.4.4.4, via GigabitEthernet0/0/0/0.524, protected
  bitmap 0x3
    Attributes: Metric: 3, Node Protect,
```

```
CSR2#show ipv6 route ::4:4:4:4
Routing entry for ::4:4:4:4/128
  Known via "ospf 1", distance 110, metric 2, type intra area
  Backup from "isis 1 [115]"
  Route count is 1/1, share count 0
  Routing paths:
    FE80::1, GigabitEthernet2.512
    Last updated 00:24:30 ago
```

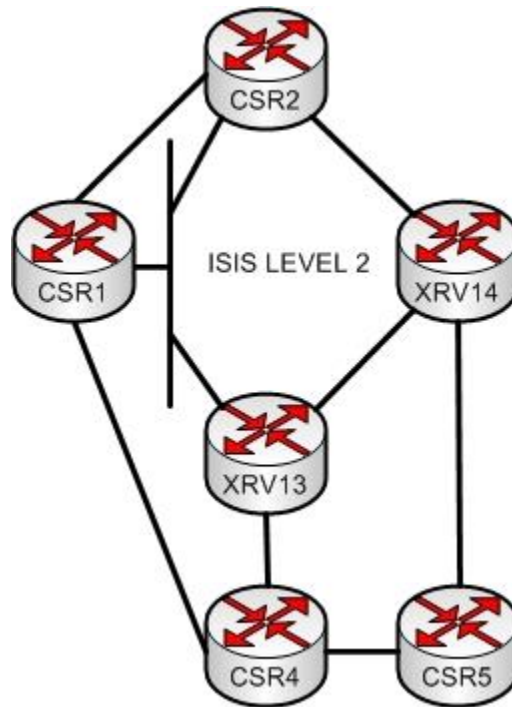
37.2.1.2 Remote LFA

Remote LFA does not appear to be supported for OSPFv3 (XR 5.3.0) at this time. This chapter is left here as a placeholder for completeness since I expect the feature to be supported soon.

37.2.2 IS-IS

37.2.2.1 Direct LFA

We will use XRv13 to reach CSR5's loopback for this demonstration. The diagram is shown again below for reference.



Enabling IS-IS LFA for IPv6 is identical to IPv4 and is done on a per-interface, per-AFI basis as shown below. Initially we see the ECMP paths backing each other up. Both CSR4 and XRV14 are downstream (closer to CSR5), node protecting, primary (ECMP) paths, equal metrics, and SRLG disjoint.

```

! XRV13
router isis 1
interface GigabitEthernet0/0/0/0.513
  address-family ipv6 unicast
  fast-reroute per-prefix

interface GigabitEthernet0/0/0/0.534
  address-family ipv6 unicast
  fast-reroute per-prefix

interface GigabitEthernet0/0/0/0.543
  address-family ipv6 unicast
  fast-reroute per-prefix
  
```

```

RP/0/0/CPU0:XRv13#show isis ipv6 fast-reroute ::5:5:5:5/128 detail
L2 ::5:5:5:5/128 [20/115] medium priority
  via fe80::14, GigabitEthernet0/0/0/0.543, XRv14, Weight: 0
    FRR backup via fe80::4, GigabitEthernet0/0/0/0.534, CSR4, Weight: 0
    P: Yes, TM: 20, LC: No, NP: Yes, D: Yes, SRLG: Yes
  via fe80::4, GigabitEthernet0/0/0/0.534, CSR4, Weight: 0
    FRR backup via fe80::14, GigabitEthernet0/0/0/0.543, XRv14, Weight: 0
    P: Yes, TM: 20, LC: No, NP: Yes, D: Yes, SRLG: Yes
src CSR5.00-00, ::5:5:5:5
  
```

We will increase XRv13's metric to XRv14 to 15 so that XRv13 prefers the path through CSR4 to ::5:5:5:5/128. We see the increase in total-metric (TM) to 25, and the path is no longer ECMP (primary, indicated by the "P" flag). We have introduced a strict primary/backup relationship between the CSR4/XRv14 paths after this metric adjustment.

```
! XRv13
router isis 1
 interface GigabitEthernet0/0/0/0.543
   address-family ipv6 unicast
   metric 15

RP/0/0/CPU0:XRv13#show isis ipv6 fast-reroute ::5:5:5:5/128 detail
L2 ::5:5:5:5/128 [20/115] medium priority
   via fe80::4, GigabitEthernet0/0/0/0.534, CSR4, Weight: 0
     FRR backup via fe80::14, GigabitEthernet0/0/0/0.543, XRv14, Weight: 0
     P: No, TM: 25, LC: No, NP: Yes, D: Yes, SRLG: Yes
   src CSR5.00-00, ::5:5:5:5
```

Next, we remove that metric adjustment and instead apply an SRLG to XRv13's links to CSR4 and XRv14. Notice that the SRLG disjoint attribute is removed but we still use it for the backup paths. This is because XR does not consider SRLGs by default; notice how the two paths back each other up as they did in the very first test.

```
! XRv13
srlg
 interface GigabitEthernet0/0/0/0.534
   8 value 333

 interface GigabitEthernet0/0/0/0.543
   8 value 333

router isis 1
 interface GigabitEthernet0/0/0/0.543
   address-family ipv6 unicast
   no metric

RP/0/0/CPU0:XRv13#show isis ipv6 fast-reroute ::5:5:5:5/128 detail
L2 ::5:5:5:5/128 [20/115] medium priority
   via fe80::14, GigabitEthernet0/0/0/0.543, XRv14, Weight: 0
     FRR backup via fe80::4, GigabitEthernet0/0/0/0.534, CSR4, Weight: 0
     P: Yes, TM: 20, LC: No, NP: Yes, D: Yes, SRLG: No
   via fe80::4, GigabitEthernet0/0/0/0.534, CSR4, Weight: 0
     FRR backup via fe80::14, GigabitEthernet0/0/0/0.543, XRv14, Weight: 0
     P: Yes, TM: 20, LC: No, NP: Yes, D: Yes, SRLG: No
   src CSR5.00-00, ::5:5:5:5
```

Next, we make SRLG the #1 tiebreaker. Now, XRv13 will prefer to not backup those ECMP paths with one another.

```
! XRv13
router isis 1
 address-family ipv6 unicast
  fast-reroute per-prefix tiebreaker srlg-disjoint index 1
```

XRv13 uses the paths through XRv14 and CSR4 for ECMP but backs them both up via CSR1. CSR1 is not downstream to CSR5 as XRv13 is closer, but when used to backup the XRv14 path only, node protection is achieved. Overall, it is a less desirable LFA, but since CSR4 and XRv4 “fate-share”, it makes sense to introduce a third link for better resiliency.

```
RP/0/0/CPU0:XRv13#show isis ipv6 fast-reroute ::5:5:5:5/128 detail
L2 ::5:5:5:5/128 [20/115] medium priority
  via fe80::14, GigabitEthernet0/0/0/0.543, XRv14, Weight: 0
    FRR backup via fe80::1, GigabitEthernet0/0/0/0.513, CSR1, Weight: 0
    P: No, TM: 30, LC: No, NP: Yes, D: No, SRLG: Yes
  via fe80::4, GigabitEthernet0/0/0/0.534, CSR4, Weight: 0
    FRR backup via fe80::1, GigabitEthernet0/0/0/0.513, CSR1, Weight: 0
    P: No, TM: 30, LC: No, NP: No, D: No, SRLG: Yes
src CSR5.00-00, ::5:5:5:5
```

The path from XRv13 to CSR2 is another candidate path but provides node protection for CSR4, not XRv14. It is functionally equivalent to the CSR1 path. By shutting down CSR1's LAN interface, the router selects CSR2. When CSR1 comes back online, XRv13 still prefers CSR2, and it appears to be similar to the eBGP tie-breaker of selecting the oldest route when all other tiebreakers are equal. Notice that node protection is achieved when backing up CSR4 but not XRv14. This is the opposite problem we saw above; there is one LFA protecting both primary paths when the second LFA is just as good.

```
! CSR1
interface GigabitEthernet2.513
 shutdown
```

```
RP/0/0/CPU0:XRv13#show isis ipv6 fast-reroute ::5:5:5:5/128 detail
L2 ::5:5:5:5/128 [20/115] medium priority
  via fe80::14, GigabitEthernet0/0/0/0.543, XRv14, Weight: 0
    FRR backup via fe80::2, GigabitEthernet0/0/0/0.513, CSR2, Weight: 0
    P: No, TM: 30, LC: No, NP: No, D: No, SRLG: Yes
  via fe80::4, GigabitEthernet0/0/0/0.534, CSR4, Weight: 0
    FRR backup via fe80::2, GigabitEthernet0/0/0/0.513, CSR2, Weight: 0
    P: No, TM: 30, LC: No, NP: Yes, D: No, SRLG: Yes
src CSR5.00-00, ::5:5:5:5
```

It would be ideal if we could use CSR1 to backup the path through XRv14 and CSR2 to backup the path through CSR4. This way, we achieve node protection for the primary paths using two node-protecting LFAs for maximal redundancy. We can do this by telling XR to evaluate the node-protect (NP) attribute, which it was not doing by default.

```
! XRv13
router isis 1
 address-family ipv6 unicast
  fast-reroute per-prefix tiebreaker node-protecting index 5
```

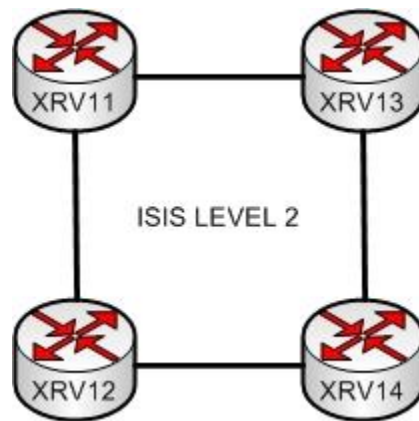
We can clearly see that that CSR1 backs up XRv14 and CSR2 backs up CSR4 as desired. After the SRLG attribute is evaluated, node-protection comes next, which allows XRv13 to more intelligent select LFAs. Tuning these tie-breakers must be carefully considered based on the network architecture since adjusting the selection process could have undesirable affects. In this case, we tuned the process to achieve a highly desirable result.

```
RP/0/0/CPU0:XRv13#show isis ipv6 fast-reroute ::5:5:5:5/128 detail
L2 ::5:5:5:5/128 [20/115] medium priority
  via fe80::14, GigabitEthernet0/0/0/0.543, XRv14, Weight: 0
    FRR backup via fe80::1, GigabitEthernet0/0/0/0.513, CSR1, Weight: 0
      P: No, TM: 30, LC: No, NP: Yes, D: No, SRLG: Yes
  via fe80::4, GigabitEthernet0/0/0/0.534, CSR4, Weight: 0
    FRR backup via fe80::2, GigabitEthernet0/0/0/0.513, CSR2, Weight: 0
      P: No, TM: 30, LC: No, NP: Yes, D: No, SRLG: Yes
src CSR5.00-00, ::5:5:5:5
```

37.2.2.2 Remote LFA

Remote LFA on XR using LDPv6 is probably beyond the scope of the blueprint given it is supported on XR 5.3.0 whereas the test uses version XR 5.2.0. For completeness, we will examine it briefly; LDPv6 is also introduced in XR 5.3.0 and is not discussed in detail. It works identically to LDP for IPv4 by allocating labels to IPv6 prefixes instead of IPv4 prefixes. OSPFv3 does not support rLFA at this time, so only IS-IS is tested. Basic rLFA concepts like P-space, extended P-space, Q-space, and PQ-node are not discussed again as they are identical between all rLFA variations (OSPF, IS-IS, IPv4, IPv6).

We will use a square topology of XRv routers for this test. The network diagram is below. IS-IS level 2 is enabled everywhere, all transit links have only link-local addressing. The OSPFv3 and LDP RIDs are hardcoded to X.X.X.X where X is the router number. The IPv6 loopbacks are ::X:X:X:X/128 and there is no IPv4 configuration anywhere.



We enable rLFA under each interface within IS-IS on XRv11 so that it can tunnel traffic to XRv14 to protect its connected links to XRv12 and XRv13. The rLFA configuration is staged under all of the other routers, but it needs to be activated with the base command "fast-reroute per-prefix" in XR.

```
! XRv11
router isis 1
interface GigabitEthernet0/0/0/0.512
  address-family ipv6 unicast
    fast-reroute per-prefix
    fast-reroute per-prefix remote-lfa tunnel mpls-ldp
    fast-reroute per-prefix remote-lfa maximum-metric 20

interface GigabitEthernet0/0/0/0.513
  address-family ipv6 unicast
    fast-reroute per-prefix
    fast-reroute per-prefix remote-lfa tunnel mpls-ldp
    fast-reroute per-prefix remote-lfa maximum-metric 20
```

The basic LDPv6 configuration is shown below but is not discussed in detail. The RID is set manually as is the discovery transport-address; IPv4 is not running so the TCP session is IPv6-based as well. I also disable LDPv4 globally since it doesn't need to be enabled. Last, I enabled LDPv6 on the proper interfaces. Similar configurations exist for all other XR routers in the network.

```
! XRv11
mpls ldp
default-vrf implicit-ipv4 disable
router-id 11.11.11.11
address-family ipv6
  discovery targeted-hello accept
  discovery transport-address ::11:11:11:11

interface GigabitEthernet0/0/0/0.512
  address-family ipv6

interface GigabitEthernet0/0/0/0.513
```

```
address-family ipv6
```

The output below shows us that the route to XRv12's loopback is protected via an rLFA tunnel to XRv14. This will show even if LDPv6 is broken (that is, the targeted session fails).

```
RP/0/0/CPU0:XRv11#show isis ipv6 fast-reroute detail ::12:12:12:12/128
L2 ::12:12:12:12/128 [10/115] medium priority
    via fe80::12, GigabitEthernet0/0/0/0.512, XRv12, Weight: 0
        Remote FRR backup via XRv14 [::14:14:14:14], via fe80::13,
GigabitEthernet0/0/0/0.513 XRv13, Weight: 0
    P: No, TM: 20, LC: No, NP: No, D: No, SRLG: Yes
    src XRv12.00-00, ::12:12:12:12
```

Next, we check XRv11 to make sure that it learns the LDP label for ::12:12:12:12/128 from XRv14 (using its RID that looks like an IPv4 address). Checking the neighbor (using the IPv6 address), we see the proper prefix association.

```
RP/0/0/CPU0:XRv11#show mpls ldp ipv6 discovery 14.14.14.14
Local LDP Identifier: 11.11.11.11:0
Discovery Sources:
  Targeted Hellos:
    ::11:11:11:11 -> ::14:14:14:14 (active), xmit/rcv
    LDP Id: 14.14.14.14:0
    Hold time: 90 sec (local:90 sec, peer:90 sec)
```

```
RP/0/0/CPU0:XRv11#show mpls ldp neighbor ::14:14:14:14
Peer LDP Identifier: 14.14.14.14:0
  TCP connection: ::14:14:14:14:60551 - ::11:11:11:11:646
  Graceful Restart: No
  Session Holdtime: 180 sec
  State: Oper; Msgs sent/rcvd: 10/11; Downstream-Unsolicited
  Up time: 00:03:14
  LDP Discovery Sources:
    IPv4: (0)
    IPv6: (1)
    Targeted Hello (::11:11:11:11 -> ::14:14:14:14, active)
  Addresses bound to this peer:
    IPv4: (0)
    IPv6: (1)
    ::14:14:14:14
```

Lastly, we check the LDPv6 LIB to ensure we learn XRv14's label for ::12:12:12:12/128 which is 94011. This is the inner label of the rLFA tunnel; the outer label will be XRv13's label for ::14:14:14:14/128, which is 93002. After IGP convergence, XRv1 will use XRv13's label for ::12:12:12:12/128 directly, which is 93016, and the rLFA tunnel is torn down since there is no more PQ-node. The rLFA label stack would

be {93002 94011} assuming rLFA was in use during the IGP convergence period. This is the same logic we evaluated with rLFA earlier for both OSPFv2 and IPv4 IS-IS.

```
RP/0/0/CPU0:XRv11#show mpls ldp ipv6 bindings ::12:12:12:12/128 neighbor
14.14.14.14:0
::12:12:12:12/128, rev 8
    Local binding: label: 91004
    Remote bindings: (2 peers)
        Peer                Label
        -----
        14.14.14.14:0       94011
```

```
RP/0/0/CPU0:XRv11#show mpls ldp ipv6 bindings ::14:14:14:14/128 neighbor
13.13.13.13:0
::14:14:14:14/128, rev 6
    Local binding: label: 91001
    Remote bindings: (3 peers)
        Peer                Label
        -----
        13.13.13.13:0       93002
```

```
RP/0/0/CPU0:XRv11#show mpls ldp ipv6 bindings ::12:12:12:12/128 neighbor
13.13.13.13:0
::12:12:12:12/128, rev 8
    Local binding: label: 91004
    Remote bindings: (3 peers)
        Peer                Label
        -----
        13.13.13.13:0       93016
```

As a final check, we validate the CEF entry and ensure the label binding is correct. I don't know why the second label is not shown in the label stack for the backup path, but without BFD we can't actually test it. CEF imposes the topmost label and I am assuming the rLFA process, behind the scenes, is imposing label 94011 before CEF sees it.

```
RP/0/0/CPU0:XRv11#show cef ipv6 ::12:12:12:12/128
::12:12:12:12/128, version 520, internal 0x1000001 0x0 (ptr 0xa13b8874) [1],
0x0 (0xa13842d8), 0xa28 (0xa173512c)
    local adjacency fe80::12
    Prefix Len 128, traffic index 0, precedence n/a, priority 3
    via fe80::12, GigabitEthernet0/0/0/0.512, 5 dependencies, weight 0, class
0, protected [flags 0x400]
    path-idx 0 bkup-idx 1 NHID 0x0 [0xa180d384 0xa180d29c], parent-ifh 0x200
    next hop fe80::12
        local label 91004          labels imposed {ImplNull}
        via fe80::13, GigabitEthernet0/0/0/0.513, 5 dependencies, weight 0, class
0, backup [flags 0x300]
        path-idx 1 NHID 0x0 [0xa0c7f4a4 0x0]
```

```

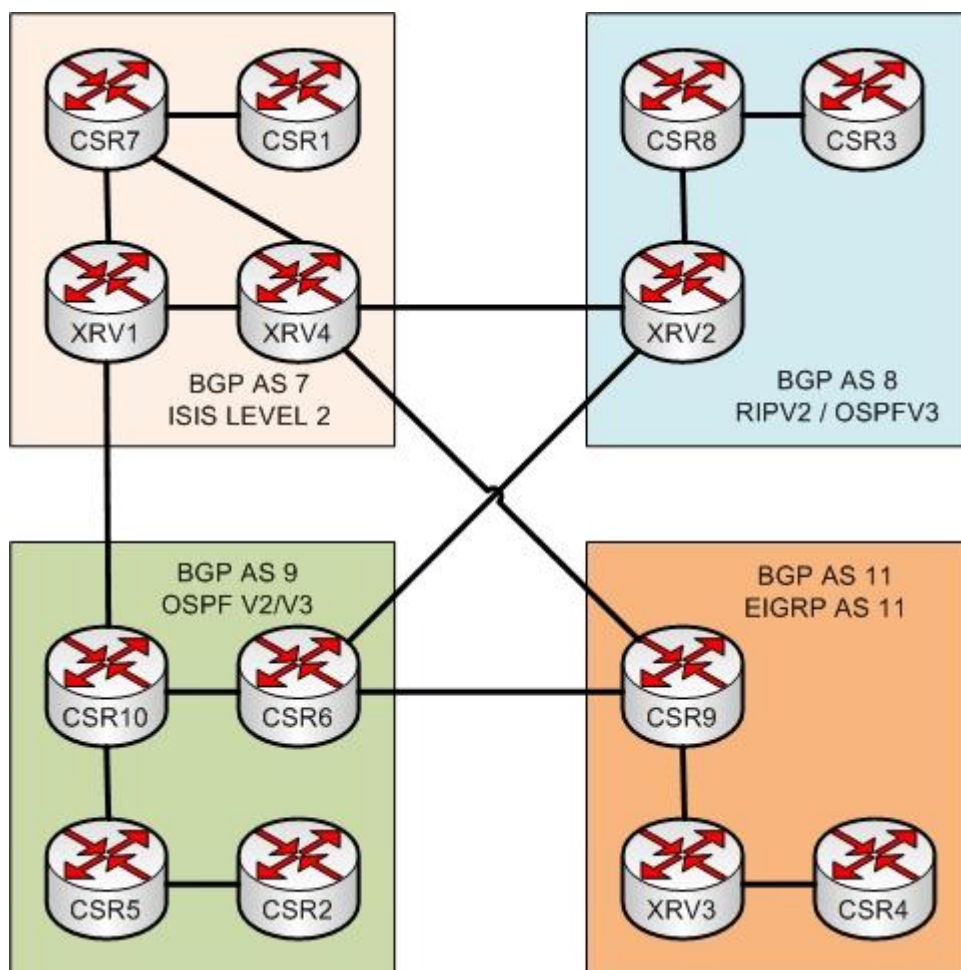
next hop fe80::13
local adjacency
local label 91004 labels imposed {93002}

```

Additional Reading – Reference configurations "rlfa-ipv6"

37.3 Convergence optimizations for BGP

BGP has many special timers that can tune its behavior. This lab explores many of them within the context of IPv4 and IPv6 AFIs. The ordinary “neighbor timers” are evaluated in a specific section dedicated to individual protocol hellos and are not discussed here. The lab topology is similar to that seen in the inter-AS multicast section, except that the BGP IPv4/v6 multicast AFIs have been disabled. Other multicast features remain enabled but are not the focus of this test, and are thus ignored. The diagram is shown again for reference. There are 4 different ASes which have many external and internal BGP peerings for additional testing. Each AS has its own IPv4/v6 IGP for intra-AS reachability. Unlike the inter-AS multicast lab, I enabled BGP IPv4/v6 unicast on CSR1 through CSR4; these were test hosts in the multicast lab but are now used as BGP routers for additional testing. XRV1, CSR10, and CSR8 are route-reflectors for both AFIs in their respective ASes, while AS11 has a full mesh of iBGP peers.



The first timer is the advertisement interval. This timer determines the rate at which advertisements are sent to a peer; the default for eBGP is 30 seconds while iBGP is 0 seconds (no delay). This allows for updates to be grouped together and advertised in chunks rather than independently. This grouping mechanism is loosely analogous to the OSPF LSA-group pacing mechanism for LSA age refreshing. When there are multiple changes in an AS a few seconds apart, they are propagated immediately within the AS but delayed externally so that a more efficient update can be assembled. Since the primary purpose of BGP was very high scalability, this could reduce convergence effectiveness. Before making configuration adjustments, we can confirm the default timers on XE and XR for eBGP and iBGP peers. A timer of 0 seconds, the iBGP default, implies no delay in propagating updates.

```
R6#show bgp ipv4 unicast neighbors | include ^BGP|advertisement
BGP neighbor is 9.0.0.10, remote AS 9, internal link
  Default minimum time between advertisement runs is 0 seconds
BGP neighbor is 10.6.9.9, remote AS 11, external link
  Default minimum time between advertisement runs is 30 seconds
BGP neighbor is 10.6.12.12, remote AS 8, external link
  Default minimum time between advertisement runs is 30 seconds
```

```
RP/0/0/CPU0:XRv4#show bgp ipv4 unicast neighbors | utility egrep
'^BGP|advertisement'
BGP neighbor is 7.0.0.11
  Minimum time between advertisement runs is 0 secs
BGP neighbor is 10.9.14.9
  Minimum time between advertisement runs is 30 secs
  Policy for incoming advertisements is PASS
  Policy for outgoing advertisements is PASS
BGP neighbor is 10.12.14.12
  Minimum time between advertisement runs is 30 secs
  Policy for incoming advertisements is PASS
  Policy for outgoing advertisements is PASS
```

We can confirm this behavior by creating two changes in quick succession within AS 7 and observing the behavior. Adding a new network on CSR2 into IPv4 BGP, this is advertised immediately to iBGP peers, and then to eBGP peers. The iBGP advertisement interval expires immediately so when a second network is added, it is also advertised to iBGP peers immediately. The eBGP advertisement interval begins counting down from 30, and until it expires, updates will not be sent towards a given peer.

To test it, on CSR2 and CSR6 I enable BGP update debugging for IPv4. Adding the first network to CSR2, we see that CSR2 advertises it to its iBGP peers immediately, and when CSR6 receives it from the iBGP RR (CSR10), it advertises it to its eBGP peers immediately. Outbound updates are shown in green with inbound updates shown in yellow. The clocks on CSR2 and CSR6 are not synchronized, but it does not matter since we are only comparing time differences in each local router. The difference for both is 1 ms, which is effectively zero.

! CSR2

```

interface Loopback1
 ip address 9.2.2.2 255.255.255.255

router bgp 9
 address-family ipv4
  network 9.2.2.2 mask 255.255.255.255

! CSR2
01:57:19.681: BGP: topo global:IPv4 Unicast:base Remove_fwdroute for
9.2.2.2/32
01:57:19.681: BGP(0): redistributedlocal route 9.2.2.2/32 modified
01:57:19.682: BGP(0): 9.0.0.10 NEXT_HOP is set to self for net 9.2.2.2/32,
01:57:19.682: BGP(0): (base) 9.0.0.10 send UPDATE (format) 9.2.2.2/32, next
9.0.0.2, metric 0, path Local

! CSR6
07:41:19.505: BGP(0): 9.0.0.10 rcvd UPDATE w/ attr: nexthop 9.0.0.2, origin
i, localpref 100, metric 0, originator 9.0.0.2, clusterlist 9.0.0.10
07:41:19.505: BGP(0): 9.0.0.10 rcvd 9.2.2.2/32
07:41:19.505: BGP(0): Revise route installing 1 of 1 routes for 9.2.2.2/32 ->
9.0.0.2(global) to main IP table
07:41:19.506: BGP(0): (base) 10.6.12.12 send UPDATE (format) 9.2.2.2/32, next
10.6.12.6, metric 0, path Local

```

Quickly adding a second network into IPv4 BGP on CSR2 (within a few seconds), we notice that CSR2 advertises it to its iBGP peers immediately, but CSR6 waits. The first eBGP update was sent at 07:41:19 which means the next update should not be sent before 07:41:49; it might be a few seconds after, but never before. CSR6 receives the update immediately but cannot advertise it until 07:41:49 which is when the advertisement-interval for eBGP peers will expire. The timestamp of the eBGP update for the second network proves this.

```

! CSR2
interface Loopback1
 ip address 9.2.2.22 255.255.255.255 secondary

router bgp 9
 address-family ipv4
  network 9.2.2.22 mask 255.255.255.255

! CSR2
01:57:31.579: BGP: topo global:IPv4 Unicast:base Remove_fwdroute for
9.2.2.22/32
01:57:31.579: BGP(0): redistributedlocal route 9.2.2.22/32 modified
01:57:31.579: BGP(0): 9.0.0.10 NEXT_HOP is set to self for net 9.2.2.22/32,
01:57:31.579: BGP(0): (base) 9.0.0.10 send UPDATE (format) 9.2.2.22/32, next
9.0.0.2, metric 0, path Local

! CSR6

```

```

07:41:31.402: BGP(0): 9.0.0.10 rcvd UPDATE w/ attr: nexthop 9.0.0.2, origin
i, localpref 100, metric 0, originator 9.0.0.2, clusterlist 9.0.0.10
07:41:31.402: BGP(0): 9.0.0.10 rcvd 9.2.2.22/32
07:41:31.402: BGP(0): Revise route installing 1 of 1 routes for 9.2.2.22/32 -
> 9.0.0.2(global) to main IP table

! CSR6, about 18 seconds later
07:41:49.827: BGP(0): (base) 10.6.12.12 send UPDATE (format) 9.2.2.22/32,
next 10.6.12.6, metric 0, path Local

```

The construction of these more efficient updates is handled by the BGP update-group feature. When updates are assembled, they are assembled once and replicated to peers that have the same outbound policies. These outbound policies include route filters, advertisement intervals, etc. Internal and external peers are always in different update groups regardless of the outbound policies. We can see the update groups by specifying a neighbor and then seeing all of the members. In this case, both XRv2 and CSR9 are in the same update group, which is why only one update was created above.

```

R6#show bgp ipv4 unicast update-group 10.6.12.12
BGP version 4 update-group 6, external, Address Family: IPv4 Unicast
  BGP Update version : 275/0, messages 0, active RGs: 1
  Topology: global, highest version: 275, tail marker: 275
  Format state: Current working (OK, last minimum advertisement interval)
                 Refresh blocked (not in list, last not in list)
  Update messages formatted 98, replicated 171, current 0, refresh 0, limit
1000
  Number of NLRI's in the update sent: max 3, min 0
  Minimum time between advertisement runs is 30 seconds
  Has 2 members:
    10.6.12.12 10.6.9.9

```

Adjusting the advertisement-interval for one of the peers will break the update group efficiency. To make things interesting, I will increase the advertisement-interval (slow the convergence) towards CSR9. We now see two update groups, one with each eBGP peer, due to the advertisement interval changing. The output also, assuming the interval is counting down, will show you the time remaining as demonstrated below.

```

! CSR6
router bgp 9
  address-family ipv4
    neighbor 10.6.9.9 advertisement-interval 45

R6#show bgp ipv4 unicast update-group
BGP version 4 update-group 6, external, Address Family: IPv4 Unicast
  BGP Update version : 277/0, messages 0, active RGs: 1
  Topology: global, highest version: 277, tail marker: 277
  Format state: Current blocked (minimum advertisement interval, last minimum
advertisement interval)

```

```

Refresh blocked (not in list, last not in list)
Update messages formatted 100, replicated 175, current 0, refresh 0, limit
1000
Number of NLRI's in the update sent: max 3, min 0
Minimum time between advertisement runs is 30 seconds (expires in 1
seconds)
Has 1 member:
10.6.12.12

BGP version 4 update-group 7, external, Address Family: IPv4 Unicast
BGP Update version : 277/0, messages 0, active RGs: 1
Topology: global, highest version: 277, tail marker: 277
Format state: Current blocked (minimum advertisement interval, last minimum
advertisement interval)
Refresh blocked (not in list, last not in list)
Update messages formatted 11, replicated 11, current 0, refresh 0, limit
1000
Number of NLRI's in the update sent: max 3, min 0
Minimum time between advertisement runs is 45 seconds (expires in 43
seconds)
Has 1 member:
10.6.9.9

```

Quickly retesting the advertisement using the test networks on CSR2, we enable debugging on CSR6 only. Since we didn't change the iBGP settings, we won't confirm how advertisement works over iBGP again. We see that CSR6 receives the update 9.2.2.2/32 at time 08:00:22 and the update for 9.2.2.22/32 at time 08:00:28, about 6 seconds later. 9.2.2.2/32 was immediately advertised to both eBGP peers in two separate updates (thanks to the two separate update-groups) and their advertisement-intervals behind counting down.

```

! CSR6
08:00:22.850: BGP(0): 9.0.0.10 rcvd UPDATE w/ attr: nexthop 9.0.0.2, origin
i, localpref 100, metric 0, originator 9.0.0.2, clusterlist 9.0.0.10
08:00:22.850: BGP(0): 9.0.0.10 rcvd 9.2.2.2/32
08:00:22.850: BGP(0): Revise route installing 1 of 1 routes for 9.2.2.2/32 ->
9.0.0.2(global) to main IP table
08:00:22.852: BGP(0): (base) 10.6.12.12 send UPDATE (format) 9.2.2.2/32, next
10.6.12.6, metric 0, path Local
08:00:22.852: BGP(0): (base) 10.6.9.9 send UPDATE (format) 9.2.2.2/32, next
10.6.9.6, metric 0, path Local

08:00:28.719: BGP(0): 9.0.0.10 rcvd UPDATE w/ attr: nexthop 9.0.0.2, origin
i, localpref 100, metric 0, originator 9.0.0.2, clusterlist 9.0.0.10
08:00:28.719: BGP(0): 9.0.0.10 rcvd 9.2.2.22/32
08:00:28.719: BGP(0): Revise route installing 1 of 1 routes for 9.2.2.22/32 -
> 9.0.0.2(global) to main IP table

```

30 seconds after the time the first update is sent to XRv2, the update for 9.2.2.22/32 is also sent; this is time 08:00:52. 45 seconds after that initial time (or 15 seconds after the update was sent to XRv2), the update for 9.2.2.22/32 is sent to CSR9 since we slowed down the advertisement-interval. This later time is 08:01:07. This proves that the advertisement-interval works as expected.

```
! CSR6
08:00:52.733: BGP(0): (base) 10.6.12.12 send UPDATE (format) 9.2.2.22/32,
next 10.6.12.6, metric 0, path Local
08:01:07.071: BGP(0): (base) 10.6.9.9 send UPDATE (format) 9.2.2.22/32, next
10.6.9.6, metric 0, path Local
```

The feature works identically in XR, and for variety we will perform our configurations for IPv6 BGP. Currently, XRv2 and CSR9 are in the same update-group for this AFI due to their identical outbound filtering RPLs and matching advertisement-intervals.

```
RP/0/0/CPU0:XRv4#show bgp ipv6 unicast update-group neighbor fd00:10:9:14::9
Update group for IPv6 Unicast, index 0.3:
  Attributes:
    Outbound policy: PASS
    First neighbor AS: 11
    Directly connected IPv6 EBGP
    4-byte AS capable
    Non-labeled address-family capable
    Minimum advertisement interval: 30 secs
  Update group desynchronized: 0
  Sub-groups merged: 8
  Number of refresh subgroups: 0
  Messages formatted: 201, replicated: 292
  All neighbors are assigned to sub-group(s)
  Neighbors in sub-group: 0.1, Filter-Groups num:1
    Neighbors in filter-group: 0.4(RT num: 0)
      fd00:10:9:14::9
      fd00:10:12:14::12
```

Modifying an advertisement interval will break this. In this case, I reduce the interval to 15.5 seconds on the peer to XRv2 (15 seconds plus 500 ms). XR adds greater capability by permitting millisecond granularity as well. However, the configuration is per-neighbor, not per-neighbor per-AFI. Even though we only test IPv6, this adjustment affects all AFIs negotiated to a given neighbor. Checking the update groups, we can see the update group has been split as the advertisement intervals differ (30 vs. 15.5 seconds). For brevity (and because XR's BGP update debugging is not very clear) we will not test this in real-time, but the effect is identical to XE.

```
! XRv4
router bgp 7
  neighbor fd00:10:12:14::12
    advertisement-interval 15 500
```

```
RP/0/0/CPU0:XRv4#show bgp ipv6 unicast update-group neighbor fd00:10:9:14::9
Update group for IPv6 Unicast, index 0.3:
```

```
Attributes:
```

```
Outbound policy: PASS
First neighbor AS: 11
Directly connected IPv6 EBGp
4-byte AS capable
Non-labeled address-family capable
Minimum advertisement interval: 30 secs
```

```
Update group desynchronized: 0
Sub-groups merged: 8
Number of refresh subgroups: 0
Messages formatted: 201, replicated: 292
All neighbors are assigned to sub-group(s)
Neighbors in sub-group: 0.1, Filter-Groups num:1
Neighbors in filter-group: 0.4(RT num: 0)
fd00:10:9:14::9
```

```
RP/0/0/CPU0:XRv4#show bgp ipv6 unicast update-group neighbor
fd00:10:12:14::12
Update group for IPv6 Unicast, index 0.1:
```

```
Attributes:
```

```
Outbound policy: PASS
First neighbor AS: 8
Directly connected IPv6 EBGp
4-byte AS capable
Non-labeled address-family capable
Minimum advertisement interval: 15.500 secs
```

```
Update group desynchronized: 0
Sub-groups merged: 0
Number of refresh subgroups: 0
Messages formatted: 11, replicated: 11
All neighbors are assigned to sub-group(s)
Neighbors in sub-group: 0.3, Filter-Groups num:1
Neighbors in filter-group: 0.1(RT num: 0)
fd00:10:12:14::12
```

The next timer is the BGP scan-time. This is the timer that is used for BGP to “walk the RIB” to verify IGP reachability for BGP next-hops. This occurs every 60 seconds by default, and in order to see it in action, we have to disable a smarter feature known as next-hop tracking (NHT), which is discussed later. Since the scan-time is timer-based and can be tuned very coarsely, it is not a good choice for event-driven or high-availability architectures. This is another per-AFI feature; below I show the defaults for IPv4/IPv6 on XE and XR.

```
R10#show bgp ipv4 unicast summary | include scan
BGP activity 93/55 prefixes, 406/352 paths, scan interval 60 secs
```

```
R10#show bgp ipv6 unicast summary | include scan
BGP activity 93/55 prefixes, 406/352 paths, scan interval 60 secs
```

```
RP/0/0/CPU0:XRv1#show bgp ipv4 unicast | include scan
BGP generic scan interval 60 secs
BGP scan interval 60 secs
```

```
RP/0/0/CPU0:XRv1#show bgp ipv6 unicast | include scan
BGP generic scan interval 60 secs
BGP scan interval 60 secs
```

To disable NHT and adjust the scan timer concurrently for brevity, I issue the commands below. The scanner has been set to 3 times its normal speed so that we can detect next-hop changes in 20 seconds. NHT cannot be disabled on XR, but we will increase the next-hop trigger delays to larger numbers than the scan time which effectively disables it. NHT is discussed in greater detail later.

```
! CSR10
router bgp 9
  address-family ipv4
    no bgp nexthop trigger enable
    bgp scan-time 20

! XRv1
router bgp 7
  address-family ipv6 unicast
    bgp scan-time 20
    nexthop trigger-delay critical 30000
    nexthop trigger-delay non-critical 30000
```

Before testing this with any prefixes, we can watch the scanner in action by debugging BGP events. On XE, I enable the debugging for IPv4 unicast yet the debug shows all of the scanners. Notice that all of the scanners start at once, but then the IPv4 run runs twice more 20 seconds apart, while the others are still waiting for their next cycle of 60 seconds.

```
R10#debug bgp ipv4 unicast events
02:54:32.322: BGP: tbl IPv4 Unicast:base Performing BGP Nexthop scanning for general scan
02:54:32.322: BGP(0): Future scanner version: 5972, current scanner version: 5971
02:54:32.322: BGP: tbl IPv6 Unicast:base Performing BGP Nexthop scanning for general scan
02:54:32.322: BGP(1): Future scanner version: 5958, current scanner version: 5957
02:54:32.322: BGP: tbl L2VPN E-VPN:base Performing BGP Nexthop scanning for general scan
02:54:32.322: BGP(10): Future scanner version: 5958, current scanner version: 5957
02:54:32.322: BGP: tbl MVPNV4 Unicast:base Performing BGP Nexthop scanning for general scan
02:54:32.322: BGP(15): Future scanner version: 5958, current scanner version: 5957
02:54:32.322: BGP: tbl MVPNV6 Unicast:base Performing BGP Nexthop scanning for general scan
```

```

02:54:32.322: BGP(16): Future scanner version: 5958, current scanner version: 5957
02:54:52.325: BGP: Regular scanner timer event
02:54:52.325: BGP: tbl IPv4 Unicast:base Performing BGP Nexthop scanning for nhop scan
02:54:52.325: BGP(0): Future scanner version: 5973, current scanner version: 5972
02:55:12.324: BGP: Regular scanner timer event
02:55:12.324: BGP: tbl IPv4 Unicast:base Performing BGP Nexthop scanning for nhop scan
02:55:12.324: BGP(0): Future scanner version: 5974, current scanner version: 5973

```

The output is similar on XR. We can see the scans running 20 seconds apart.

```

RP/0/0/CPU0:XRv1#debug bgp event afi ipv6 unicast
02:58:02.990 : bgp[1052]: [default-event] (ip6u): Scanning IPv6 Unicast
routing table
02:58:02.990 : bgp[1052]: [default-event] (ip6u): Garbage collection for
table TBL:default (2/1) prefix tree - Target version 260
02:58:02.990 : bgp[1052]: [default-event] (ip6u): Prefix count 18, Segment
count 1, Segment size 100000
02:58:02.990 : bgp[1052]: [default-event] (ip6u): Finished scanning IPv6
Unicast routing table
02:58:23.009 : bgp[1052]: [default-event] (ip6u): Scanning IPv6 Unicast
routing table
[snip]
02:58:43.027 : bgp[1052]: [default-event] (ip6u): Scanning IPv6 Unicast
routing table
[snip]

```

To test it, I will reduce the scan timer even further on CSR2 to 10 seconds. This creates a log message warning the user about potential CPU issues as the scanner will scan all prefixes in the BGP table, even non-best-paths. NHT is also disabled on CSR2 which allows the scanner to operate properly. Then, I shut down CSR6's loopback. After about 40 seconds, this will also kill the BGP session to CSR10, but we see the work of the scanner with respect to the next-hop reachability of all eBGP routes learned from next-hop 9.0.0.6. When the loopback gets shut down, IGP converges quickly and for a few seconds, CSR2 still thinks the BGP routes have valid next-hops. After the scanner runs, those next-hops become inaccessible and the BGP routes are no longer candidates for the best-path.

```

! CSR2
R2(config-router-af)#bgp scan-time 10
%BGP-5-AGGRESSIVE_SCAN_TIME: bgp scan-time configuration less than 15 seconds
can cause high cpu usage by BGP Scanner.

```

Before CSR6's loopback is shutdown, the "baseline" network show commands are below. 9.0.0.6/32 is reachable and the BGP prefixes that rely on it are candidates for the best-path selection algorithm.

```

R2#show ip route 9.0.0.6
Routing entry for 9.0.0.6/32
  Known via "ospfv3 9", distance 110, metric 3, type intra area
  Last update from 9.2.5.5 on GigabitEthernet2.525, 00:00:44 ago

```


Routing Descriptor Blocks:

```
* 9.2.5.5, from 9.0.0.6, 00:00:44 ago, via GigabitEthernet2.525
  Route metric is 3, traffic share count is 1
```

```
R2#show bgp ipv4 unicast 8.0.0.3/32
BGP routing table entry for 8.0.0.3/32, version 248
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 2
  8
    9.0.0.6 (metric 3) from 9.0.0.10 (9.0.0.10)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Originator: 9.0.0.6, Cluster list: 9.0.0.10
      rx pathid: 0, tx pathid: 0x0
```

IGP has already withdrawn reachability to 9.0.0.6/32 but BGP is not yet aware. This is the purpose of the scanner; it detects these IGP changes and update the information about the BGP next-hop, such as the accessibility, metric, etc.

```
R2#show ip route 9.0.0.6
% Subnet not in table
```

```
R2#show bgp ipv4 unicast 8.0.0.3/32
BGP routing table entry for 8.0.0.3/32, version 248
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 2
  8
    9.0.0.6 (metric 3) from 9.0.0.10 (9.0.0.10)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Originator: 9.0.0.6, Cluster list: 9.0.0.10
      rx pathid: 0, tx pathid: 0x0
```

When the scanner runs, it notices that 9.0.0.6 is inaccessible, and marks the BGP prefixes that use this next-hop as such. The debug doesn't show us the exact scanner activity; it only reveals that the scanner ran according to schedule.

```
! CSR2
03:16:13.780: BGP: Regular scanner timer event
03:16:13.780: BGP: tbl IPv4 Unicast:base Performing BGP Nexthop scanning for
nhop scan
03:16:13.780: BGP(0): Future scanner version: 1609, current scanner version:
1608
03:16:15.278: BGP: aggregate timer expired
```

```
R2#show ip route 9.0.0.6
% Subnet not in table
```

```

R2#show bgp ipv4 unicast 8.0.0.3/32
BGP routing table entry for 8.0.0.3/32, version 261
Paths: (1 available, no best path)
  Not advertised to any peer
  Refresh Epoch 2
  8
    9.0.0.6 (inaccessible) from 9.0.0.10 (9.0.0.10)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Originator: 9.0.0.6, Cluster list: 9.0.0.10
      rx pathid: 0, tx pathid: 0

```

The BGP scanner is similar to RIP for unicast routing. Being timer-based, it does not work in large deployments where routing changes occur frequently, nor does it facilitate an end-to-end HA design. The NHT feature was introduced as an event-driven mechanism; the BGP “RIB watcher” process maintains a list of all BGP next-hops. This process will report back to BGP when one of two events happen: the next-hop becomes unreachable or the IGP metric to the next-hop changes. The first event is more important and, in most cases, should be reported more quickly; the default is 5 seconds in XE. This value should be tuned to slightly longer than it takes IGP to converge because notifying BGP of a next-hop failure in a timely manner is not valuable unless there is an alternative path. Alerting BGP before IGP has a chance to calculate the backup path means that routes will be marked inaccessible and may be withdrawn entirely. Waiting a second or so is a less disruptive option than immediate withdrawal. The NHT delay value can be set to as low as 0 seconds, which implies no delay, and should only be used in cases where some sort of IP fast-reroute is in play. This may include ECMP, OSPF LFA, ISIS LFA, or even basic EIGRP feasible successor designs. For the demonstration, I configure the feature on CSR9 and XRv2. Notice that only XR allows us to differentiate between “critical” events, such as prefix reachability, and “non-critical” events, such as IGP metric changes. The defaults for these two event types is 0 and 3 seconds, respectively, so XR assumes you have an immediately-converging IGP by default. XR measures these timers in milliseconds while XE measures it in seconds.

```

! CSR9
router bgp 11
  address-family ipv6
    bgp nexthop trigger delay 2

! XRv2
router bgp 8
  address-family ipv4 unicast
    nexthop trigger-delay critical 2000
    nexthop trigger-delay non-critical 4000

```

Before testing this, we can also specify a route-map/RPL to determine which routes can qualify as next-hops. Although not specifically related to the delay adjustments, this is useful for avoiding aggregate routes. For example, we may want to adjust certain iBGP nodes to always prefer prefixes of length 30 or longer for next-hops. This would cover transit links redistributed into IGP or next-hop-self on ASBRs using their loopbacks. It ensures that the route recursion doesn’t select a default route, for example,

then claim the next-hop is reachable. All next-hops would be reachable in this case, which is undesirable. For IPv6, next-hops should be length /64 or longer, which means that the route is likely not an aggregate.

```
! CSR9
ipv6 prefix-list PL_64_AND_LONGER seq 5 permit ::/0 ge 64
route-map RM_BGP_NHOP_V6 permit 10
  match ipv6 address prefix-list PL_64_AND_LONGER

router bgp 11
  address-family ipv6
    bgp nexthop route-map PL_64_AND_LONGER

! XRv2
route-policy RPL_BGP_NHOP
  if destination in (0.0.0.0/0 ge 30) then
    pass
  endif
end-policy

router bgp 8
  address-family ipv4 unicast
    nexthop route-policy RPL_BGP_NHOP
```

Applying this to XRv2, we immediately see a problem. In most real deployments, private transit links would be /30 or longer (or possibly unnumbered, but that is unrelated to this). In this test lab, they are /24. This means that all of the eBGP routes learned on XRv2 will have /24 routes for their next-hops. Below is the BGP table where we can clearly see none of the eBGP routes have best-paths due to next-hop inaccessibility. The iBGP routes, however, are reachable since the next-hop is the advertising iBGP peers loopback address.

```
RP/0/0/CPU0:XRv2#show bgp ipv4 unicast | begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
* 7.0.0.1/32	10.6.12.6			0	9 7 ?
* 7.0.0.2/32	10.12.14.14			0	7 ?
* 7.0.0.7/32	10.6.12.6			0	9 7 ?
* 7.0.0.8/32	10.12.14.14			0	7 ?
* 7.0.0.11/32	10.6.12.6			0	9 7 i
* 7.0.0.12/32	10.12.14.14			0	7 i
* 7.0.0.14/32	10.6.12.6			0	9 7 i
* 7.0.0.15/32	10.12.14.14	0		0	7 i
* 7.1.7.0/24	10.6.12.6			0	9 7 i
* 7.1.7.1/24	10.12.14.14			0	7 i
*>i 8.0.0.3/32	8.0.0.3	0	100	0	? ?
*>i 8.0.0.8/32	8.0.0.8	0	100	0	? ?
*> 8.0.0.12/32	0.0.0.0	0		32768	i
*>i 8.3.8.0/24	8.0.0.8	0	100	0	i

```
RP/0/0/CPU0:XRv2#show bgp ipv4 unicast 7.0.0.1/32
BGP routing table entry for 7.0.0.1/32
Versions:
  Process          bRIB/RIB   SendTblVer
  Speaker          188       188
Paths: (2 available, no best path)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  9 7
    10.6.12.6 (inaccessible) from 10.6.12.6 (9.0.0.6)
      Origin incomplete, localpref 100, valid, external
      Received Path ID 0, Local Path ID 0, version 0
      Origin-AS validity: not-found
  Path #2: Received by speaker 0
  Not advertised to any peer
  7
    10.12.14.14 (inaccessible) from 10.12.14.14 (7.0.0.14)
      Origin incomplete, localpref 100, valid, external
      Received Path ID 0, Local Path ID 0, version 0
      Origin-AS validity: not-found
```

```
RP/0/0/CPU0:XRv2#show route 10.6.12.6
Routing entry for 10.6.12.0/24
  Known via "connected", distance 0, metric 0 (connected)
  Routing Descriptor Blocks
    directly connected, via GigabitEthernet0/0/0/0.562
    Route metric is 0
  No advertising protos.
```

```
RP/0/0/CPU0:XRv2#show route 10.12.14.14
Routing entry for 10.12.14.0/24
  Known via "connected", distance 0, metric 0 (connected)
  Routing Descriptor Blocks
    directly connected, via GigabitEthernet0/0/0/0.524
    Route metric is 0
  No advertising protos.
```

As an adjustment for this misconfiguration, I add an exception for 10-networks that are exactly /24 in length. This still follows the spirit of the “no aggregates for next-hops” policy since we know the 10.x.x.x/24 routes are not aggregates. The XR RPL structure has an “eq” option so that we don’t have to use “ge 24 le 24” as we would in an XE prefix-list. The BGP routes are immediately valid again since their BGP next-hops are reachable.

```
! XRv2
route-policy RPL_BGP_NHOP
  if destination in (0.0.0.0/0 ge 30) or destination in (10.0.0.0/8 eq 24)
  then
```

```

    pass
endif
end-policy

```

```

RP/0/0/CPU0:XRv2#show bgp ipv4 unicast | begin Network
  Network          Next Hop          Metric LocPrf Weight Path
* 7.0.0.1/32       10.6.12.6                          0 9 7 ?
*>                10.12.14.14                       0 7 ?
* 7.0.0.7/32       10.6.12.6                          0 9 7 ?
*>                10.12.14.14                       0 7 ?
[snip]

```

We can also view all of the BGP next-hops. XE prints a simple table showing the next-hop addresses and number of BGP prefixes bound to that next-hop which also includes non best-paths. This is specific to the BGP table, not the routing table.

```

R9#show bgp ipv6 unicast nexthops
# Paths Nexthop Address
  14 FD00:10:9:14::14 (FE80::14)
  14 FD00:10:6:9::6 (FE80::6)
   1 200B::4
   2 200B::D

```

The show command in XR is much more involved. It includes many pieces of statistical information which respect to next-hop processing and path changes. It also shows us the configured NHT values for both critical and non-critical events, which serves as a good verification of the configuration. The status codes clearly show which next-hops are reachable, connected versus IGP learned, and other detailed status information. Like CSR9, we see a few connected next-hops for eBGP routes and a few IGP-learned next-hops for iBGP routes.

```

RP/0/0/CPU0:XRv2#show bgp ipv4 unicast nexthops
Total Nexthop Processing
  Time Spent: 0.000 secs

```

```

Maximum Nexthop Processing
  Received: 3d04h
  Bestpaths Deleted: 0
  Bestpaths Changed: 16
  Time Spent: 0.000 secs

```

```

Last Notification Processing
  Received: 3d04h
  Time Spent: 0.000 secs

```

```

Gateway Address Family: IPv4 Unicast
Table ID: 0xe0000000
Nexthop Count: 5
Critical Trigger Delay: 2000msec

```

Non-critical Trigger Delay: 4000msec

Nexthop Version: 1, RIB version: 1

Status codes: R/UR Reachable/Unreachable
C/NC Connected/Not-connected
L/NL Local/Non-local
PR Pending Registration
I Invalid (Policy drop)

Next Hop	Status	Metric	Notf	LastRIBEvent	RefCount
8.0.0.3	[R] [NC] [NL]	2	0/0	2d01h (Reg)	1/3
8.0.0.8	[R] [NC] [NL]	1	1/0	4d08h (Cri)	2/5
10.6.12.6	[R] [C] [NL]	0	1/0	4d08h (Cri)	16/19
10.12.14.14	[R] [C] [NL]	0	1/0	4d08h (Cri)	16/19

We can drill into a single next-hop as well by specifying the next-hop explicitly. This reveals additional information about the next-hop such as the routing source (RIP), BGP prefix reference counts, and other low-level debugging information.

RP/0/0/CPU0:XRv2#show bgp ipv4 unicast nexthops 8.0.0.3

Nexthop: 8.0.0.3

VRF: default
Nexthop ID: 0x600007b, Version: 0x0
Nexthop Flags: 0x00000000, Gateway Flags: 0x00000080
Nexthop Handle: 0x10c6f890, Gateway Handle: 0x10799720

RIB Related Information

Gateway: reachable, non-Connected route, prefix length 32

Resolving Route: 8.0.0.3/32 (rip)

Paths: 0

RIB Nexthop ID: 0x0

Status: [Reachable][Not Connected][Not Local]

Metric: 2

Registration: Synchronous, Completed: 2d01h

Events: Critical (0)/Non-critical (0)

Last Received: 2d01h (Registration)

Last gw update: (Crit-sync) 2d01h(rib)

Reference Count: 1

Prefix Related Information

Active Tables: [IPv4 Unicast]

Metrics: [0x2]

Reference Counts: [1]

Interface Handle: 0x0

To test NHT, I will shutdown CSR8's loopback. XRv2 learns this via IGP and it is the next-hop for one prefix, 8.0.0.3/32. This is a RIB failure, but it doesn't matter for the purpose of this test. The slow

convergence of RIP will take up to 180 seconds for this route to be invalidated, and once that happens, the NHT delay is 2 seconds longer. Once the route is withdrawn (debug reveals this), a critical next-hop change occurs. BGP marks the next-hop as inaccessible and removes the BGP prefixes associated with this next-hop. We can see a very large metric, but most importantly, a critical event occurred a few seconds before. This “critical event” was the removal of the route. If we issue the command a few seconds later, the next-hop information for 8.0.0.3/32 is removed entirely.

```
RP/0/0/CPU0:XRv2#debug routing ipv4
ipv4_rib[1144]: RIB Routing: Vrf: "default", Tbl: "default" IPv4 Unicast,
Delete active route to 8.0.0.3 via 8.8.12.8 interface
GigabitEthernet0/0/0/0.582, metric [120/2], label None, by client-id 14
```

```
RP/0/0/CPU0:XRv2#show bgp ipv4 unicast nexthops 8.0.0.3
Nexthop: 8.0.0.3
  VRF: default
  Nexthop ID: 0x600007e, Version: 0x0
  Nexthop Flags: 0x00000000, Gateway Flags: 0x00000080
  Nexthop Handle: 0x10c6f890, Gateway Handle: 0x10799698
```

RIB Related Information

```
Gateway: unreachable, non-Connected route, prefix length 512
Resolving Route: 8.0.0.3/32 (rip)
Paths: 0
RIB Nexthop ID: 0x0
Status: [Unreachable]
Metric: 4294967295

Registration: Synchronous, Completed: 00:00:22
Events: Critical (1)/Non-critical (0)
Last Received: 00:00:02 (Critical)
Last gw update: (Crit-notif) 00:00:02(rib)
Reference Count: 0
```

Prefix Related Information

```
Active Tables: [IPv4 Unicast]
Metrics: [0xffffffff]
Reference Counts: [0]
Interface Handle: 0x0
```

```
RP/0/0/CPU0:XRv2#show bgp ipv4 unicast nexthops 8.0.0.3
% Nexthop information not found
```

On CSR9, we will actually test the timers in greater detail. I will trigger a non-critical event by changing the bandwidth on CSR4’s loopback interface. EIGRP will recompute a new metric and BGP NHT will notice this. I enable BGP event debugging (only available under the IPv4 parser tree but affects all AFIs) and IPv6 routing table debugs. When EIGRP adds the new (worse) metric, the BGP “RIB watcher” immediately notifies BGP of the change. The prefix 200B::4/128 is still reachable, so this would be a

“non-critical” next-hop change event, and the penalty information applies only to dampening (not configured). Exactly 2 seconds later (the NHT delay, shown in green), BGP begins walking its prefixes for updates as it would have normally done via the scanner in older versions. In between those 2 seconds, in a real network, IGP should have converged. In summary, this means the BGP RIB watcher process intentionally delays notifying BGP in order to give IGP time to converge after a topology change event. Routes that are not directly affected by the next-hop change are still reassessed, as seen below.

```
! CSR9
debug bgp ipv4 unicast events
debug ipv6 routing

03:20:42.212: [EIGRP-IPv6]IPv6RT[default]: eigrp 11, Route add 200B::4/128
[worse metric 7178700, 16000]
03:20:42.212: [EIGRP-IPv6]IPv6RT[default]: eigrp 11, Update path
FE80::13/GigabitEthernet2.593 Flags : 0 : 0, tag 0, metric 7178700
03:20:42.213: BGP: bgp_rwatch_notify: BGP_RWATCH_APPL_NHOP
03:20:42.213: EvD: charge penalty 500, new accum. penalty 500, flap count 11
03:20:42.213: BGP: bgp_rwatch_notify: BGP_RWATCH_APPL_NBR
03:20:42.213: BGP: nbr global 200B::4 bgp_process_bnbr_notification reachable

03:20:44.213: [BGP Router]IPv6RT[default]: bgp 11, Update 2007::1/128 [20/0],
1 paths tag 0
03:20:44.213: [BGP Router]IPv6RT[default]: bgp 11, Updating route 2007::1/128
[20/0], [20/0] 20
03:20:44.213: [BGP Router]IPv6RT[default]: bgp 11, Update path
FE80::14/GigabitEthernet2.594 Flags : 4 : 4, tag 0, metric 0
[snip]
```

Another global BGP timer is the update-delay and generally applies only to BGP during initial startup. This timer starts counting when the first BGP peer is formed. When the timer expires, the router will begin the BGP best-path selection process for all routes and be allowed to advertise best-paths to peers. The idea is to operate in “read-only” mode for a short time when BGP starts to collect all prefixes from all peers before running best-path initially, which ultimately determines which routes a BGP speaker advertises to others. The default timer is 120 seconds and this is a global BGP setting, not per-AFI. We will test a faster timer on CSR3. This timer is different from the rest because it represents a maximum time; we will configure timer of 30 seconds so that CSR3 could begin advertising updates in some amount of time less than 30 if all prefixes are received. Note that this feature is only valuable when a BGP speaker has multiple peers; with only one peer, there is never a reason to wait once the routes have been received.

```
! CSR9
router bgp 11
  bgp update-delay 30

R9#debug bgp ipv4 unicast events
BGP events debugging is on
```



```
R9#debug bgp ipv4 unicast update
BGP updates debugging is on for address family: IPv4 Unicast
```

We enable debugging and then “clear bgp ipv4 unicast” to restart the entire process. The peer with CSR4 comes up first at 23:21:54. Because this is an iBGP peer, CSR4 immediately advertises its BGP routes to CSR9. For brevity, I do not show all of the other eBGP peer formations and received updates, but this is the “read-only” phase. CSR9 will receive all of these updates from all peers, and having a reasonable (not zero) update-delay allows BGP time to collect this information.

```
! CSR9
23:21:54.579: BGP(0): 11.0.0.4 was the first peer to be established for IPv4
Unicast
23:21:54.579: BGP: nopeerup-delay post-boot, set to default, 60s
23:21:54.579: %BGP-5-ADJCHANGE: neighbor 11.0.0.4 Up
23:21:54.580: BGP: nbr_topo global 11.0.0.4 IPv4 Unicast:base
(0x7F6994DBB708:1) rcvd Refresh Start-of-RIB
23:21:54.580: BGP: nbr_topo global 11.0.0.4 IPv4 Unicast:base
(0x7F6994DBB708:1) refresh_epoch is 2
23:21:54.580: BGP(0): 11.0.0.4 rcvd UPDATE w/ attr: nexthop 11.0.0.4, origin
?, localpref 100, metric 0
23:21:54.580: BGP(0): 11.0.0.4 rcvd 11.0.0.4/32
23:21:54.580: BGP: nbr_topo global 11.0.0.4 IPv4 Unicast:base
(0x7F6994DBB708:1) rcvd Refresh End-of-RIB
```

As soon as the first peer is established, the local router “needs to delay read-only mode” so the BGP UPDATE process must exit. This message is printed continuously, once per second and once per AFI, until RO mode ends. Mixed in with this reminder are the syslog messages that the other BGP peers have come up, but this is less significant since the update-delay begins counting when the first peer comes up, which explains the debug output above. Those peers were discussed above (neighbor forms, updates received, etc) but not shown.

```
! CSR9
23:21:54.969: BGP: tbl IPv4 Unicast:base Generate update check
23:21:54.969: BGP: tbl IPv4 Unicast:base AF needs to delay RO exit
```

After the update-delay expires, about 30 seconds in this case, the IPv4 unicast AFI becomes read-write and the BGP best-path selection process begins. I omit some of the prefixes for brevity but all BGP prefixes are evaluated for their best-paths at this time. The debug makes it easy for troubleshooting this timer as well.

```
! CSR9
23:22:24.670: BGP(base): waited 30s[update delay] for all peers
23:22:24.670: BGP: tbl IPv4 Unicast:base Computed bestpaths, table version
went from 1 to 20
23:22:24.670: BGP(0): Revise route installing 1 of 1 routes for 7.0.0.1/32 ->
10.9.14.14(global) to main IP table
```

```

23:22:24.670: BGP(0): Revise route installing 1 of 1 routes for 7.0.0.7/32 ->
10.9.14.14(global) to main IP table
[snip]
23:22:24.670: BGP: tbl IPv4 Unicast:base AF is now RW
23:22:24.671: BGP: tbl IPv4 Unicast:base IMP Initial import complete.
23:22:24.671: BGP: notified IGP's about convergence

```

Immediately after best-path runs, neighbor updates can be sent provided the advertisement-interval allows for it.

```

! CSR9
23:22:24.674: BGP(0): 11.0.0.13 NEXT_HOP is set to self for net 7.0.0.1/32,
23:22:24.674: BGP(0): (base) 11.0.0.13 send UPDATE (format) 7.0.0.1/32, next
11.0.0.9, metric 0, path 7

```

Although we do not test it due to lack of revealing XR debugs, we configure the BGP update-delay timer on XRv3 for completeness. Because the update-delay is a maximum timer by default in XR, we can optionally enforce this delay as a minimum timer by using the “always” keyword. This forces the specified delay regardless of whether or not BGP receives all prefixes from all peers in a timely manner.

```

! XRv3
router bgp 11
  bgp update-delay 30 always

```

Another minor BGP timer is the nopeerup-delay, which prevents peer formation after some “hard” event. We can delay BGP peers from forming in during four different disruptive events:

1. **Cold-boot:** When the router first boots up. I assume this occurs when the router was powered-off or was forcefully reset (not “reload” command).
2. **NSF-switchover:** After an NSF switchover only.
3. **Post-boot:** When the system is already booted and all peers go down. This includes the “reload” command.
4. **User-initiated:** When a BGP peer is manually cleared by the administrator.

The default value for all of these fields is zero according to the documentation and 60 seconds according to the debug (I trust the debug). The feature only appears supported on XE platforms. This network is not NSF-capable since that feature doesn’t apply to virtual routers, so we can test the other three timers. I configure all timers on CSR6 since it has a combination of internal and external peers.

```

! CSR6
router bgp 7
  bgp nopeerup-delay cold-boot 240
  bgp nopeerup-delay post-boot 300
  bgp nopeerup-delay nsf-switchover 120
  bgp nopeerup-delay user-initiated 30

```

First, I reload CSR6 and immediately enable BGP debugging as soon as the console is available. This will allow us to see the nopeerup-delay post-boot timer of 300 seconds configured above. Based on this test, we see that “post-boot” relates to administrator-initiated reloads as this is the timer revealed in the debugs. However, this feature does not appear to work. All BGP peers form immediately despite the debug saying that the backoff should be 5 minutes. The first log message was printed at 02:32:30 and the first BGP came up ~38 seconds later, which indicates unexpected behavior. This may be a CSR1000v artificiality operating in a virtual environment.

```
! CSR6
debug bgp ipv4 unicast
debug bgp ipv4 unicast events
```

```
02:32:30.646: %VUDI-6-EVENT: [serial number: 92V72A6SE1F], [vUDI: ], vUDI is
successfully retrieved from license file
```

```
02:33:08.388: BGP: ses global 10.6.12.12 (0x7F2109416840:1) Up
02:33:08.388: BGP: nopeerup-delay post-boot [config:300s/default:60s] set to
300s
02:33:08.388: %BGP-5-ADJCHANGE: neighbor 10.6.12.12 Up
```

The user-initiated timer also does not appear to work. Clearing a peer manually, it comes back up almost immediately despite the 30-second delay configured. Although the configuration guide makes reference to this feature typically being used with NSF, it is not listed as a prerequisite. While this test was not able to demonstrate the feature functioning properly, we can at least verify it was configured properly by checking the debugs that show the newly configured values (as observed above).

```
R6#clear bgp ipv4 unicast 10.6.9.9
02:37:46.231: %BGP-3-NOTIFICATION: sent to neighbor 10.6.9.9 6/4
(Administrative Reset) 0 bytes
02:37:46.231: %BGP-5-ADJCHANGE: neighbor 10.6.9.9 Down User reset
02:37:46.231: %BGP_SESSION-5-ADJCHANGE: neighbor 10.6.9.9 IPv4 Unicast
topology base removed from session User reset
02:37:46.387: %BGP-5-ADJCHANGE: neighbor 10.6.9.9 Up
```

The last BGP feature we will test is the slow-peer feature. A slow-peer is a BGP peer that cannot keep up with the routing updates sent to it. For example, peering a large carrier router with a small branch router, then trying to send it many updates very quickly may not work. Since this slow-peer might be in the same update-group as other peers operating correctly, the sending router would be slowing all updates down for all peers in the update-group. The number of updates to be serviced would queue, and convergence is slowed significantly. There are 3 ways to deal with slow-peers:

1. **Slow-peer detection:** The router simply issues a syslog message to notify the administrator of a dynamically-detected slow-peer but takes no action. The update-groups remain intact as they were before the discovery was made; this is just a warning mechanism.
2. **Slow-peer dynamic protection:** The router detects the slow peer and creates a “slow-peer update-group” for this peer, and any other slow peers with identical outbound attributes. Peers can remain in this slow group indefinitely or temporarily until they stop being slow. This feature relies on the slow-peer detection mechanism as well.
3. **Slow-peer static configuration:** This is slow-peer protection but is statically configured. This option may be the result of using slow-peer detection and manually taking action after an internal network policy review, for example. This is also a mechanism to simply force a BGP peer into a different update-group without having to adjust the advertisement-interval.

We will configure all 3 options on XE only as the feature does not appear to be supported on XR. On CSR6, we will enable slow-peer detection for IPv4 unicast (all peers) except XRv2. For IPv6 unicast, we enable it for the eBGP peer CSR9 only. The detection is enabled per-AFI and has a threshold value associated with it. This measurements how “slow” a “slow-peer” is and is 5 minutes by default; this is the difference between the oldest update in the peer’s update queue versus the current time. If an update was added to a peer’s update queue at 12:00 and it is still in the queue at 12:05, the peer is considered slow. Smaller values imply a more strict definition of “slow-peer”, so I reduce it to the minimum value of 2 minutes. I explicitly disabled slow-peer detection to XRv2 for IPv4 and enable it to CSR9 only for IPv6.

```
! CSR6
router bgp 9
  address-family ipv4
    bgp slow-peer detection threshold 120
    neighbor 10.6.12.12 slow-peer detection disable

  address-family ipv6
    neighbor FD00:10:6:9::9 slow-peer detection threshold 120
```

To verify the configuration, we can check the BGP neighbor details per-peer, per-AFI. For IPv4, we see that slow-peer detection is enabled for both CSR10 and CSR9, but not XRv2. For IPv6, only the peer to CSR9 is running the feature.

```
R6#show bgp ipv4 unicast neighbors | include ^BGP|detection
BGP neighbor is 9.0.0.10, remote AS 9, internal link
  Slow-peer detection is enabled, threshold value is 120
BGP neighbor is 10.6.9.9, remote AS 11, external link
  Slow-peer detection is enabled, threshold value is 120
BGP neighbor is 10.6.12.12, remote AS 8, external link
  Slow-peer detection is disabled
```

```
R6#show bgp ipv6 unicast neighbors | include ^BGP|detection
```

BGP neighbor is 2009::A, remote AS 9, internal link
 Slow-peer detection is disabled
 BGP neighbor is FD00:10:6:9::9, remote AS 11, external link
 Slow-peer detection is enabled, threshold value is 120
 BGP neighbor is FD00:10:6:12::12, remote AS 8, external link
 Slow-peer detection is disabled

As seen earlier, adjusting the advertisement-interval for the eBGP peers caused the router to create a different update group for XRv2 and CSR9. As such, CSR6 has three update groups. We can tell this is the case as each update-group has only one member, so in this case, slow-peer detection isn't very effective. Each update-group has a cache (queue) where BGP updates are stored before being sent to peers. The size of the cache is sized dynamically based on available router memory, number of peers, etc. Although an inefficient use of update-groups, this AFI on CSR6 doesn't need to worry about slow-peers as each peer is already in a separate update-group. As a side note, the group "leader" is the peer with the highest IPv4/v6 address, and when BGP prepares an update for the group, it does so targeting the "leader". In a way, BGP is performing business-as-usual by preparing an update for a single peer, although it is replicated to all others in the update-group.

R6#show bgp ipv4 unicast replication

Index	Members	Leader	MsgFmt	MsgRepl	Csize	Current Version	Next Version
6	1	10.6.12.12	221	296	0/1000	554/0	
14	1	9.0.0.10	75	75	0/1000	554/0	
15	1	10.6.9.9	12	12	0/1000	554/0	

For IPv6, we did not make any adjustments to advertisement-intervals, so the eBGP peers are in the same update-group.

R6#show bgp ipv6 unicast replication

Index	Members	Leader	MsgFmt	MsgRepl	Csize	Current Version	Next Version
9	2	FD00:10:6:12::12	734	1449	0/1000	1782/0	
16	1	2009::A	200	200	0/1000	1782/0	

We will attempt to treat CSR9 as a slow-peer. First, we check CSR6 to see how many prefixes it is advertising to CSR9 and find a total of 15.

R6#show bgp ipv6 unicast neighbors FD00:10:6:9::9 advertised-routes | include Total
 Total number of prefixes 15

Next, we define an MQC policy to drop BGP messages shorter than 120 bytes. This is an approximate value used to differentiate small keepalives (60-80 bytes) from larger BGP updates. This will mean that

the BGP updates will be unacknowledged and CSR6 will keep trying to send them. This filter is temporary as it will break basic BGP operation entirely between CSR6 and CSR9.

```
! CSR9
class-map match-all CMAP_DROP_BGP_UPDATES
  match protocol bgp
  match packet length min 120

policy-map PMAP_FILTER
  class CMAP_DROP_BGP_UPDATES
    police 8000 conform-action drop
  class class-default

interface GigabitEthernet2.569
  service-policy input PMAP_FILTER
```

I will configure CSR6 to perform an outbound route-refresh to CSR9 which will trigger the update process.

```
R6#debug bgp ipv6 unicast FD00:10:6:9::9 updates out
BGP updates debugging is on for neighbor FD00:10:6:9::9 (outbound) for
address family: IPv6 Unicast

! CSR6
BGP(1): (base) FD00:10:6:12::12 send UPDATE (format) 2007::1/128, next
FD00:10:6:12::6, metric 0, path 7
BGP(1): (base) FD00:10:6:12::12 send UPDATE (format) 2007::B/128, next
FD00:10:6:12::6, metric 0, path 7
BGP(1): FD00:10:6:12::12 NEXT_HOP part 1 net 2008:8:3:8::/64, next
FD00:10:6:12::12
BGP(1): (base) FD00:10:6:12::12 send UPDATE (format) 2008:8:3:8::/64, next
FD00:10:6:12::12, metric 0, path 8
BGP(1): (base) FD00:10:6:12::12 send UPDATE (format) 200B::9/128, next
FD00:10:6:12::6, metric 0, path 11
```

We check to ensure CSR9 is dropping these large update packets. Although we cannot be 100% sure these are the BGP updates, we are fairly confident this is the case based on the MQC policy.

```
R9#show policy-map interface gig2.569
GigabitEthernet2.569

Service-policy input: PMAP_FILTER

Class-map: CMAP_DROP_BGP_UPDATES (match-all)
  7 packets, 6854 bytes
  5 minute offered rate 0000 bps, drop rate 0000 bps
  Match: protocol bgp
  Match: packet length min 120
```

```

police:
  cir 8000 bps, bc 1500 bytes
  conformed 7 packets, 6854 bytes; actions:
    drop
  exceeded 0 packets, 0 bytes; actions:
    drop
  conformed 0000 bps, exceeded 0000 bps

Class-map: class-default (match-any)
  85 packets, 7422 bytes
  5 minute offered rate 0000 bps, drop rate 0000 bps
Match: any

```

We can see the updates piling up in the cache for that update-group. Unfortunately, I am not able to find a way to actually make the slow-peer detection discover a slow-peer despite these efforts. This illustrates an example of how a slow-peer could form, though, and in the interest of time I continue on. CSR9's inability to acknowledge dropped BGP packets is probably less of a "slow-peer" issue and more of a fundamentally broken BGP relationship. With professional-grade test equipment to inject thousands of BGP routes, this would be easier to test.

```
R6#show bgp ipv6 unicast replication
```

Index	Members	Leader	MsgFmt	MsgRepl	Csize	Current Version	Next Version
9	2	FD00:10:6:12::12					
			765	1484	5/1000	1789/0	
16	1	2009::A					
			206	206	0/1000	1789/0	

If a slow-peer were detected, the time at which the last detection takes place is recorded in the neighbor details. This is valuable for historical tracking.

```

R6#show bgp ipv6 unicast neighbors FD00:10:6:9::9 | include detect
Slow-peer detection is enabled, threshold value is 120
Last detected as dynamic slow peer: never

```

Beyond detection, we can configure a router to automatically add a slow-peer to its own update group. Since a single slow-peer effectively slows down updates for the entire group, this is often the best method for dealing with slow-peers (as opposed to shutting them down, etc). On CSR9, we will assume XRv4 is a potential slow-peer for both AFIs. For IPv4, the slow-peer will be dynamically added to its own update-group (along with other slow-peers potentially) only when the slow-peer is detected. This implies that slow-peer detection is also enabled when we configure slow-peer protection. After the peer is no longer considered slow (say, the CPU utilization drops sufficiently so that BGP updates can be processed at normal speeds), it can be moved into the normal update group, which is the essence of dynamic recovery. The "permanent" option disables this recovery and is discussed later.

```

! CSR9
router bgp 11
  address-family ipv4
    neighbor 10.9.14.14 slow-peer split-update-group dynamic

  address-family ipv6
    neighbor FD00:10:9:14::14 slow-peer split-update-group dynamic permanent

```

We can verify that slow-peer protection is configured for the IPv4 peer of XRv4. Detection is implicitly enabled, dynamic protection is explicitly enabled, and the peer is allowed to recover. Like the detection timer, the dynamic recovery timestamp will update when the slow-peer is no longer deemed as such.

```

R9#show bgp ipv4 unicast neighbors 10.9.14.14 | include low.peer
Slow-peer detection is enabled, threshold value is 300
Slow-peer split-update-group dynamic is enabled, and active
Last detected as dynamic slow peer: never
Dynamic slow peer recovered: never

```

The output for the IPv6 peer of XRv4 is almost identical. The difference is that the “permanent” keyword implies this neighbor can never recover from being labeled a slow-peer (short of manually clearing it, reloading the router, etc). Once deemed a slow-peer, the router makes no assumption and has no expectation that the peer will ever speed up.

```

R9#show bgp ipv6 unicast neighbors FD00:10:9:14::14 | include low.peer
Slow-peer detection is enabled, threshold value is 300
Slow-peer split-update-group dynamic permanent is enabled, and active
Last detected as dynamic slow peer: never
Dynamic slow peer recovered: never

```

If a peer has been identified as slow, we can clear this penalty manually using the command below. This is the only way for a slow-peer to gracefully recovery from dynamic protection mode with the “permanent” option also applied.

```

R9#clear bgp ipv6 unicast FD00:10:9:14::14 slow

```

Both the slow-peer detection and slow-peer dynamic protection can be configured per neighbor or per-AFI, and I have demonstrated a number of these options. The final option for dealing with slow-peers is to statically configure them on a per-neighbor basis only. This is fail-safe mechanism for breaking the update-group by excluding specific peers known to be slow. Before the slow-peer feature was introduced, adjusting the advertisement-interval was the only way to guarantee this, which is a valid workaround but less elegant.

```

! CSR9
router bgp 11
  address-family ipv4

```



```
neighbor 10.6.9.6 slow-peer split-update-group static
```

When we verify this, the output changes slightly. Slow-peer detection remains disabled because there is nothing to detect; we statically configure this peer as slow. The static mode of the command is enabled, and we know that the timestamps for detection and recovery time do not apply here.

```
R9#show bgp ipv4 unicast neighbors 10.6.9.6 | include low.peer
```

```
Slow-peer split-update-group static is enabled
```

```
Slow-peer detection is disabled
```

```
Slow-peer split-update-group dynamic is disabled
```

```
Last detected as dynamic slow peer: never
```

```
Dynamic slow peer recovered: never
```

Looking at all of CSR9's update-groups, we now expect to see 3. There is one that is common for all iBGP peers since their advertisement-intervals and outbound policies are identical. There is a second for the IPv4 eBGP peer to XRv4, which has slow-peer detection enabled dynamically, and it is currently in a regular update-group (that is, not currently identified as a slow-peer). Other non-slow eBGP peers, if they existed, would also be in this update-group. The final group is the update-group for slow peers and is identified as a "slow update group". Other slow peers would end up in this group if identified as such; XRv4, if dynamically determined to be slow, is an example. **Note: Since the slow-peer syntax is not supported in XR, you can use the "advertisement-interval" to target a slow peer and achieve the same effect.**

```
R9#show bgp ipv4 unicast update-group
```

```
BGP version 4 update-group 7, internal, Address Family: IPv4 Unicast
```

```
BGP Update version : 26/0, messages 0, active RGs: 1
```

```
NEXT_HOP is always this router for eBGP paths
```

```
Community attribute sent to this neighbor
```

```
Topology: global, highest version: 26, tail marker: 26
```

```
Format state: Current working (OK, last not in list)
```

```
Refresh blocked (not in list, last not in list)
```

```
Update messages formatted 14, replicated 28, current 0, refresh 0, limit 1000
```

```
Number of NLRI's in the update sent: max 3, min 0
```

```
Minimum time between advertisement runs is 0 seconds
```

```
Has 2 members:
```

```
11.0.0.13 11.0.0.4
```

```
BGP version 4 update-group 8, external, Address Family: IPv4 Unicast
```

```
BGP Update version : 26/0, messages 0, active RGs: 1
```

```
Topology: global, highest version: 26, tail marker: 26
```

```
Format state: Current working (OK, last minimum advertisement interval)
```

```
Refresh blocked (not in list, last not in list)
```

```
Update messages formatted 27, replicated 43, current 0, refresh 0, limit 1000
```

```
Number of NLRI's in the update sent: max 3, min 0
```

```
Minimum time between advertisement runs is 30 seconds
```

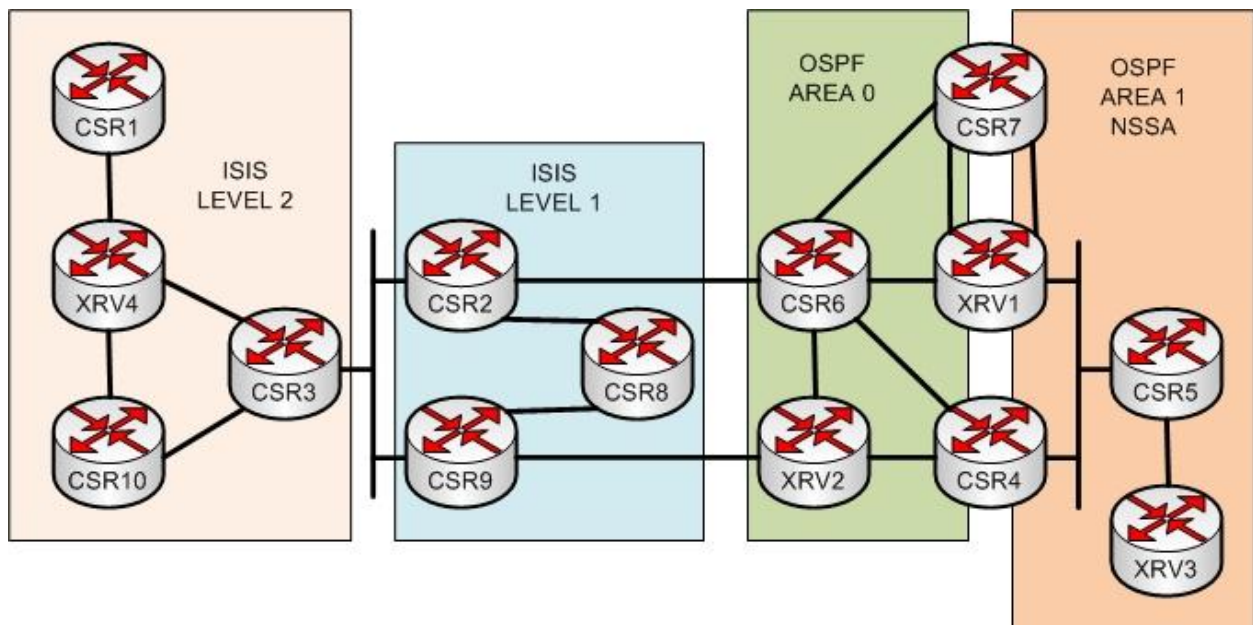
Has 1 member:
10.9.14.14

BGP version 4 update-group 9, external, Address Family: IPv4 Unicast
BGP Update version : 26/0, messages 0, active RGs: 1
Slow update group
Topology: global, highest version: 26, tail marker: 26
Format state: Current working (OK, last minimum advertisement interval)
Refresh blocked (not in list, last not in list)
Update messages formatted 11, replicated 11, current 0, refresh 0, limit 1000
Number of NLRI's in the update sent: max 2, min 0
Minimum time between advertisement runs is 30 seconds
Has 1 member:
10.6.9.6

Additional Reading – Reference configurations "bgp-conv-timers"

37.4 Convergence optimizations for IGP's

This lab explores the various OSPF and IS-IS fast convergence techniques, specifically related to SPF calculation and topology advertisement. Most of these are detailed timers; dead neighbor detection timers (hellos) are not covered in this section as there is a dedicated section for that. This lab uses the same topology and basic configurations as the LDP lab, except has all advanced LDP features removed. There are multiple IS-IS levels and multiple OSPF areas, which makes it an ideal topology for this test. Both the IS-IS and OSPFv2/v3 testing uses the same set of configurations which are shown beneath the diagram.



Additional Reading – Reference configurations "fast-conv"

37.4.1 IS-IS

IS-IS has a handful of convergence timers which are used to fine-tune how the internal IS-IS components operate. These components include intra-level SPF, partial recalculations for non-intra-level topology or stub network information (PRC), LSP generation adjustments, and more. We will define each one in detail and test some examples.

spf-interval: This is the time between consecutive SPF runs. This is a throttling mechanism, not a dampening mechanism (nothing is disabled or moved into a hold-down state) which is designed to protect the router from IS-IS hogging the CPU during times of continuous network instability within a level. There are three values that make up this throttling mechanism.

The first timer is the max-hold time, which is the maximum time a router waits between SPF. If 2 or more SPF runs are needed, the SPF backoff timer waits twice as long as the previous run up to the max-hold. The default is 10 seconds and this will make more sense when we examine the other 2 timers.

The second value is the initial wait time, which is the time a node waits before starting the first SPF run. Having this be smaller means the first SPF run happens faster, but waiting a short time allows the router to collect more changed LSPs, which might mean less SPF runs overall. The default is 5500 ms.

The third value is how long the router waits from after the completion of the first SPF run to beginning the second. Multiple SPF runs might be necessary if a router receives more LSP updates after the first changes. The default is 5500 ms.

It is challenging to find the perfect values to optimize convergence as this is entirely network dependent. This is best illustrated with examples below. Inside level-1, we will start the first SPF run after 100 ms, with the second run being 200 ms after the first. The max backoff time will be 2 second. Thus, if multiple SPF runs are needed within level-1, they would run at after 100, 200, 400, 800, 1600, and 2000 ms periods; the wait-timer doubles until the max-hold is reached. For level-1, these aggressive timers might be OK if the topology is stable. If more than 6 SPF runs are required, they would all be spaced at 2000 ms apart.

```
! CSR3
router isis LDP
  spf-interval level-1 2 100 200
```

For level-2, more conservative timers might be more appropriate. Here, we will configure the routers to wait 3 seconds before starting SPF, with the next SPF run being 7 seconds later. The max-hold time is 28 seconds, so the SPF runs will occur at 3, 7, 14, and 28 second intervals.

```
! CSR3
router isis LDP
  spf-interval level-2 28 3000 7000
```

We can quickly test these timers in some capacity. Rather than dig into unnecessary detail on the exact operation of SPF, we will perform cursory checks. If we clear IS-IS on CSR10, we can see that ~3 seconds after the adjacency comes back up, SPF begins. We can reveal this with debugs.

```
R3#debug isis spf-events
IS-IS SPF events debugging is on for IPv4 unicast topology base

! CSR3
00:53:18.260: %CLNS-5-ADJCHANGE: ISIS: Adjacency to R10
(GigabitEthernet2.530) Down, neighbor forgot us
00:53:18.332: %CLNS-5-ADJCHANGE: ISIS: Adjacency to R10
(GigabitEthernet2.530) Up, new adjacency

! About 3 seconds later
00:53:21.311: ISIS-SPF: Compute L2 SPT
00:53:21.311: ISIS-SPF: 4 nodes for level-2
[snip]
```

We will also test the more aggressive level-1 timers. If we clear ISIS on a node that isn't connected to CSR3, we won't see an adjacency flap, so we can also enable a debug to show us LSP changes. More generally, the command shows us any event that triggers SPF, and we see that the first LSP change seen by CSR3 starts the 100 ms timer. After 100 ms (perfect timing in this case), SPF runs for level-1.

```
R3#debug isis spf-triggers
IS-IS SPF triggering events debugging is on for IPv4 unicast topology base

! CSR3
01:04:46.046: ISIS-SPF-TRIG: L1, 0000.0000.0009.00-00 TLV contents changed,
code 22
01:04:46.146: ISIS-SPF: Compute L1 SPT
01:04:46.146: ISIS-SPF: 7 nodes for level-1
[snip]
```

prc-interval: This is the time between consecutive partial recalculation (PRC) runs. The command has identical parameters as the spf-interval and is configured on a per-process only (not per-level). The concepts of max-hold, initial-wait, and backoff-wait are measured the same with the exception of the default timers. The max-hold is 5 seconds, the initial-wait is 2 seconds, and the backoff-wait is 5 seconds. This effectively means that the first PRC waits 2 seconds and every PRC afterwards will be run 5 seconds apart. The significant difference is that this feature relates to how often partial recalculations are run, not full SPF invocations. A partial recalculation is invoked when a leaf (stub) network changes, such as an IP route. Inside level-2, we will make the timer a little more aggressive on CSR10. The initial-wait is reduced to 1500 ms so PRC begins faster. The second PRC, if necessary, runs 2500 ms after the first, with a max-hold of 5000 ms (the default). This means that the first PRC would be at the 1500 ms mark, followed by runs after 2500 and 5000 ms periods as needed.

```
! CSR10
router isis LDP
 prc-interval 5 1500 2500
```

To trigger a partial recalculation, we can enable SPF event and trigger debugging on CSR10 (level-2) and clear CSR8 (level-1). This will change the IP routes inside CSR3, and CSR10 will invoke the PRC process to essentially withdraw reachability to these prefixes. The timing was almost perfect with PRC occurring 1499 ms after receiving the LSP update.

```
! CSR10
01:21:53.712: ISIS-SPF: L2 LSP 1 (0000.0000.0003.00-00) flagged for
recalculation from 7FBCAB81848D
01:21:55.211: ISIS-SPF: LSP 1 (0000.0000.0003.00-00) Type STD
01:21:55.211: ISIS-SPF: spf_result: next_hop_parents:0x7FBC415CF020
root_distance:10, parent_count:1, parent_index:4 db_on_paths:1
01:21:55.211: ISIS-SPF: Calculating routes for L2 LSP 1 (0000.0000.0003.00-
00)
[snip]
```

max-lsp-lifetime: Measured in seconds, this is how long the router will retain LSPs learned from other ISIS routers. The default is 1200 seconds (20 minutes) and this is configured per-process, not per-level. A quick check of the database shows many LSPs each with an LSP holdtime of less than 1200 seconds. When an LSP is refreshed, its holdtime is set to the max-lsp-lifetime, and the number counts down to zero. This hold-time is actually carried in the LSP that a router originates.

```
R3#show isis database
Tag LDP:
IS-IS Level-1 Link State Database:
LSPID          LSP Seq Num  LSP Checksum  LSP Holdtime  ATT/P/OL
R2.00-00        0x00000240   0x3FC0        1110          0/0/0
R3.00-00        * 0x00000248   0x3C8E        559           1/0/0
R3.01-00        * 0x00000228   0x3EC2        500           0/0/0
R6.00-00        0x00000231   0x6CAF        843           0/0/0
R8.00-00        0x0000023C   0xD046        1161          0/0/0
R9.00-00        0x00000247   0xE1CA        1102          0/0/0
XRv2.00-00      0x00000241   0x92D8        854           0/0/0
IS-IS Level-2 Link State Database:
LSPID          LSP Seq Num  LSP Checksum  LSP Holdtime  ATT/P/OL
R1.00-00        0x000001F0   0x8FF4        484           0/0/0
R3.00-00        * 0x00000237   0xC4FC        427           0/0/0
R10.00-00       0x0000020F   0x26AF        354           0/0/0
XRv4.00-00      0x00000208   0xA7D1        963           0/0/0
```

On CSR3, we will configure a very aggressive holdtime, so aggressive that other routers don't generate their LSPs fast enough to keep the network stable. If we check CSR8, we can see that the hold-time for CSR3's LSP is now counting down from 120 seconds.

```
! CSR3
router isis LDP
 max-lsp-lifetime 120
```

```
R8#show isis database R3.00-00
```

```
Tag LDP:
```

```
IS-IS Level-1 LSP R3.00-00
```

LSPID	LSP Seq Num	LSP Checksum	LSP Holdtime	ATT/P/OL
R3.00-00	0x0000024F	0x5B68	70	1/0/0

Notice that after the LSP expires ~70 seconds later, it is refreshed immediately by CSR3. The LSP sequence number increases from 0x24F to 0x250, the next number in sequence, and the LSP has a new checksum.

```
R8#show isis database R3.00-00
```

```
Tag LDP:
```

```
IS-IS Level-1 LSP R3.00-00
```

LSPID	LSP Seq Num	LSP Checksum	LSP Holdtime	ATT/P/OL
R3.00-00	0x00000250	0x5969	102	1/0/0

We can confirm this using debug on CSR8. Letting it run a few minutes, we can see the LSP updates coming every 60-90 seconds for all of CSR3's LSPs, to include the DIS. I assume there is some LSP refresh timer approximation happening on CSR3 implicitly as a result of changing the max lifetime. I use yellow and green to show the router and DIS LSPs, respectively.

```
R8#debug isis update-packets
```

```
IS-IS Update related packet debugging is on for router process LDP
```

```
! CSR8
```

```
01:43:17.739: ISIS-Upd: Rec L1 LSP 0000.0000.0003.00-00, seq 252, ht 117,
```

```
01:43:18.080: ISIS-Upd: Rec L1 LSP 0000.0000.0003.01-00, seq 230, ht 117,
```

```
01:44:39.576: ISIS-Upd: Rec L1 LSP 0000.0000.0003.01-00, seq 231, ht 117,
```

```
01:44:43.360: ISIS-Upd: Rec L1 LSP 0000.0000.0003.00-00, seq 253, ht 117,
```

```
01:45:47.601: ISIS-Upd: Rec L1 LSP 0000.0000.0003.01-00, seq 232, ht 117,
```

```
01:45:54.352: ISIS-Upd: Rec L1 LSP 0000.0000.0003.00-00, seq 254, ht 117,
```

lsp-refresh-interval: Measured in seconds, this is how long the router will wait before re-flooding its own generated LSPs (local networks, DIS, etc). The default is 15 minutes and this is set per-process, not per-level. This number has to be less than the max-lsp-lifetime since an LSP must be refreshed before

being purged. On CSR3, we will configure this to be 15 seconds, which will generate even more frequent debugging output on CSR8. Now we can see the timers are about 15 seconds apart (slightly faster).

```
! CSR3
router isis LDP
  lsp-refresh-interval 15

! CSR8
01:49:30.508: ISIS-Upd: Rec L1 LSP 0000.0000.0003.00-00, seq 259, ht 117,
01:49:32.113: ISIS-Upd: Rec L1 LSP 0000.0000.0003.01-00, seq 237, ht 117,

01:49:42.269: ISIS-Upd: Rec L1 LSP 0000.0000.0003.00-00, seq 25A, ht 117,
01:49:46.172: ISIS-Upd: Rec L1 LSP 0000.0000.0003.01-00, seq 238, ht 117,

01:49:54.561: ISIS-Upd: Rec L1 LSP 0000.0000.0003.00-00, seq 25B, ht 117,
01:49:59.666: ISIS-Upd: Rec L1 LSP 0000.0000.0003.01-00, seq 239, ht 117,
```

lsp-gen-interval: This specifies how much time is in between creating new versions of a single LSP. This can be set on a per-process (both levels) or per-level basis. Just like the SPF and PRC calculations, this command has the max-hold, initial-wait, and backoff-wait timers. The max-wait is 5 seconds, the initial-wait is 50 ms (fast), and the backoff-wait is 5 seconds. Essentially this means that the first LSP change is propagated quickly but any subsequent changes occur 5 seconds later. Testing this is a little tricky, but we can easily verify the max-wait and backoff-wait timers. On CSR3, we will slow things down by making the routing wait a full second before propagating any updates, then backing off subsequent updates at 10 seconds each time.

```
! CSR3
router isis LDP
  lsp-gen-interval level-2 10 1000 10000
```

To simulate this, we will add a new loopback to IS-IS on CSR3, then remove it about 2 seconds later. By making the first change, the IS-IS process waits 1 second then floods the update. We can't effectively measure this other than a good guess by watching debug output on other routers. If we wait more than 1 seconds but less than 10 seconds and remove the loopback, this change will not be regenerated until 10 seconds after the first one was.

```
! CSR3
interface Loopback33
  ip address 33.33.33.33 255.255.255.255

router isis LDP
  passive-interface Loopback33
  ! wait 2 seconds
  no passive-interface Loopback33
```

By debugging on CSR1, we can see the first update where there was a “leaf route changed” in topology ID (TID) 0 which is IPv4. This carries the new IPv4 prefix 33.33.33.33/32 which we added to IS-IS. Just under 10 seconds later (9999 ms), another update occurs to say the leaf routes changed again, which indicates the removal of the loopback. Whether we removed the loopback at the 1.1 or 9.9 second mark, the LSP generation throttle would ensure that the time delta here is always 10 seconds given our configuration. We can see the sequence number increments from 0x26A to 0x26B, indicating that the LSP has been updated.

```
! CSR1
01:56:03.989: ISIS-Upd: Rec L2 LSP 0000.0000.0003.00-00, seq 26A, ht 118,
01:56:03.989: ISIS-Upd: from SNPA 0050.56a9.862a (GigabitEthernet2.514)
01:56:03.989: ISIS-Upd: LSP newer than database copy
01:56:03.989: ISIS-Upd: TLV contents different, code 135
01:56:03.989: ISIS-Upd: TID 2: TLV contents different, code 0x87
01:56:03.989: ISIS-Upd: TID 2 no change
01:56:03.989: ISIS-Upd: TID 0 leaf routes changed

01:56:13.988: ISIS-Upd: Rec L2 LSP 0000.0000.0003.00-00, seq 26B, ht 118,
01:56:13.988: ISIS-Upd: from SNPA 0050.56a9.862a (GigabitEthernet2.514)
01:56:13.988: ISIS-Upd: LSP newer than database copy
01:56:13.988: ISIS-Upd: TLV contents different, code 135
01:56:13.988: ISIS-Upd: TID 2: TLV contents different, code 0x87
01:56:13.988: ISIS-Upd: TID 2 no change
01:56:13.988: ISIS-Upd: TID 0 leaf routes changed
```

Another interesting IS-IS feature is incremental SPF (iSPF). This command is used for intra-level SPF calculations to only rebuild the parts of the tree requiring it. In a sparsely-connected graph, this can be very valuable, since a full SPF run is CPU intensive and iSPF can reduce the impact. This is different than PRC which is for prefix information (or any NLRI carried by IS-IS) and is for stub networks. iSPF helps converge a network more efficiently when intra-level links and nodes change state. The feature is interoperable with routers that don’t support it also, so we will enable it on CSR10 only. It can be enabled for level-1, level-2, or both. The numerical argument is somewhat benign; it simply details how long the router should wait before activating IS-IS from the time you configure it. The default is 120 seconds, but I set it to 1 second. If I enter the command, I don’t want the router to wait 2 minutes to obey the instruction.

```
! CSR10
router isis LDP
 ispf level-2 1
```

To test it, I will shutdown the link to XRv4 on CSR1. Since CSR1 is a stubby node, CSR10 should only need to recompute that part of the network without needing to perform much work on CSR3. We don’t examine the whole process in detail, but we can see the iSPF is enabled and functional.

```
R10#debug isis spf-events terse
```


IS-IS SPF events debugging terse is on for IPv4 unicast topology base

```
! CSR10
02:15:41.202: ISIS-SPF: Compute L2 SPT
02:15:41.202: ISIS-SPF: 4 nodes for level-2
02:15:41.202: ISIS-SPF: I-SPF: Adding LSP: (1) to NewLSP, metric: 10
02:15:41.202: ISIS-SPF: I-SPF: Adding LSP: (1) to 7FBC413E5BE0, metric: 10.
From 7FBCAB7EBE28
02:15:41.202: ISIS-SPF: I-SPF: Adding LSP: (2) to NewLSP, metric: 10
02:15:41.202: ISIS-SPF: I-SPF: Adding LSP: (2) to 7FBC413E5BE0, metric: 10.
From 7FBCAB7EBE28
[snip]
```

Another feature is IS-IS fast flood. This is used to ensure that an LSP that causes a reconvergence event (say, a failed node) is flooded to other neighbors before the local router runs SPF. The idea is to quickly “spread the alarm” within the level so other routers can run SPF in parallel, rather than running SPF immediately upon receipt and flooding the LSP later. However, flooding too many LSPs can be counterproductive, so the number can be set between 1 and 15 LSPs, where the default is 5. This often times improves IS-IS convergence in a network, and is configured on CSR3. CSR3 will fast-flood 2 LSPs before running SPF during a reconvergence event.

```
! CSR3
router isis LDP
fast-flood 2
```

These IS-IS timers work identically on IOS-XR. There does not seem to be a PRC interval timer, but the others exist. XR does a better job of labeling the values with their names rather than arraying them in a command with vague context-sensitive help information. LSP-related command are not AFI-specific and are configured under the process. AFI-specific commands are configured under the AFI stanzas.

```
! XRv4
router isis LDP
lsp-gen-interval maximum-wait 4800 initial-wait 600 secondary-wait 1200
level 2
max-lsp-lifetime 1800 level 2
address-family ipv4 unicast
ispf level 2
spf-interval maximum-wait 3000 initial-wait 50 secondary-wait 3000 level 2
```

37.4.2 OSPFv2 and OSPFv3

The mechanism by which OSPF builds a graph, at least within an OSPF area, is very similar to IS-IS within a level. The SPF timers are similar in behavior but vary slightly in syntax. It also introduces many more complicated timers compared to IS-IS as well.

timers throttle spf: This timer is identical to the IS-IS spf-interval timer. The difference is that the timers are arrayed differently on the CLI. First is the initial-wait, then the backoff-wait, then the max-hold. The defaults for each one are 5 seconds, 10 seconds, and 10 seconds, respectively. This means that OSPF waits 5 seconds for the first SPF run (very slow), and 10 seconds for every other run. We will speed things up a bit. In this case, we configure CSR7 to wait 500 ms from the moment it receives an LSA1 or LSA2 update to the time it starts SPF. The second SPF run happens 1500 ms after the first, with subsequent runs happening at the 3000, 6000, and 12000 ms marks. OSPF also shows you this information in a show command, which IS-IS does not.

```
! CSR7
router ospf 92
  timers throttle spf 500 1500 12000
```

```
R7#show ip ospf | include SPF
```

```
Initial SPF schedule delay 500 msec
Minimum hold time between two consecutive SPF 1500 msec
Maximum wait time between two consecutive SPF 12000 msec
[snip]
```

We can verify this with debugging. On CSR4, we will flap a link to create an SPF event (not shown). CSR7 receives the update from XRv2 and CSR4, and the SPF initial timer starts upon receipt of the first SPF-triggering LSA update. Exactly 500 ms later, SPF begins, which is 10 times faster than the default. The 500 ms was a good buffer because it gave CSR7 time to collect another LSA update from CSR4 before running SPF for the first time; a value of 10 ms, for example, would have necessitated at least two SPF runs, since the second LSU from CSR4 would have come in too late to be considered in the first SPF run.

```
R7#debug ip ospf spf intra
OSPF SPF intra debugging is on
```

```
! CSR7
08:26:29.656: OSPF-92 SPF : Detect change in LSA type 1, LSID 92.0.0.12 from
92.0.0.12 area 0
08:26:30.052: OSPF-92 SPF : Detect change in LSA type 1, LSID 92.0.0.4 from
92.0.0.4 area 0
08:26:30.156: OSPF-92 MON : Begin SPF at 477043.804ms, process time 8623ms
[snip]
```

timers throttle lsa: This is equivalent to the IS-IS lsp-gen-interval and uses the same trio of timers we have seen many times. The initial-timer is 0 ms (happens right away), with the backoff and max-hold timers are set to 5 seconds. This means if two interfaces flap on a router in quick succession, one of them is reflected immediately, but the other must wait 5 seconds. By increasing the initial timer to 50 ms or so, you can actually speed up convergence in the network since both link-state changes would be captured in the first update.

```
! CSR7
```

```
router ospf 92
  timers throttle lsa 50 100 500
```

Once we configure these timers, we can confirm them using show commands.

```
R7#show ip ospf | include LSA throttle
  Initial LSA throttle delay 50 msec
  Minimum hold time for LSA throttle 100 msec
  Maximum wait time for LSA throttle 500 msec
```

As a quick test, we will shut down one of CSR7's transit links while showing the clock just before we do (copy/paste works well). First, we enable LSA-generation debugs, then quickly capture a timestamp and shutdown the interface. Since CSR7 is an ABR and its only link to area 1 was lost, we see CSR7 generate LSAs for both areas. The only reason it does this in area 0 is to clear the ABR flag, not because an area 0 link failed. In any case, we can see that these LSAs are rate-limited from being created right away. Exactly 50 ms later, they are generated and flooded soon thereafter.

```
R7#debug ip ospf lsa-generation
OSPF LSA generation debugging is on
```

```
R7(config)#interface gig2.571
R7(config-subif)#do show clock
08:36:49.476 UTC
R7(config-subif)#shutdown
```

```
! CSR7
08:36:49.481: %OSPF-5-ADJCHG: Process 92, Nbr 92.0.0.11 on
GigabitEthernet2.571 from FULL to DOWN, Neighbor Down: Interface down or
detached
08:36:49.481: OSPF-92 LSGEN: Scheduling rtr LSA for area 1, build flag 0x41
(from 0x7FAFE34F1C42)
08:36:49.481: OSPF-92 LSGEN: Scheduling rtr LSA for area 1, build flag 0x41
(from 0x7FAFE34D0247)
08:36:49.483: OSPF-92 LSGEN: Scheduling rtr LSA for area 0, build flag 0x41
(from 0x7FAFE34EC3C2)
08:36:49.483: OSPF-92 LSGEN: Scheduling rtr LSA for area 1, build flag 0x41
(from 0x7FAFE34EC3C2)
08:36:49.485: OSPF-92 LSGEN: Rate limit LSA generation for
1/92.0.0.7/92.0.0.7
08:36:49.485: OSPF-92 LSGEN: Rate limit LSA generation for
1/92.0.0.7/92.0.0.7
08:36:49.535: OSPF-92 LSGEN: Suppressing prefix 92.6.7.0/24 from router LSA
08:36:49.535: OSPF-92 LSGEN: Suppressing prefix 92.11.7.0/24 from router LSA
08:36:49.535: OSPF-92 LSGEN: Build router LSA for area 0, router ID 92.0.0.7,
seq 0x800000E3
08:36:49.536: OSPF-92 LSGEN: Build router LSA for area 1, router ID 92.0.0.7,
seq 0x800000CB
```

timers lsa arrival: This timer is measured in ms and serves to ensure a router does not keep accepting the same LSA too quickly. Used in conjunction with the LSA throttle timers earlier for LSA generation, this timer enforces LSA reception instead. The default is 1000 ms and if the same LSA arrives faster than this period, the LSA is dropped. This might mean the LSAs would need to be re-flooded, but it would allow the local router to have a more stable LSDB if one router is generating LSAs too quickly.

```
! CSR7
router ospf 92
  timers lsa arrival 500
```

We can quickly verify this by checking the OSPF summary.

```
R7#show ip ospf | include arrival
  Minimum LSA arrival 500 msec
```

timers pacing flood: This timer determines how long to wait before flooding an LSA on an interface after the last one was sent. When multiple updates need to be sent, they are queued in an interface flood-list. The inter-update interval is controlled with this command. Every interface has a flood-list, and it's hard to actually see the queue grow since the timer is very fast.

```
R7#show ip ospf flood-list GigabitEthernet2.567

      OSPF Router with ID (92.0.0.7) (Process ID 92)

Interface GigabitEthernet2.567, Queue length 0
```

The timer is measured in ms with a default value of 33 ms. We will increase the value to 75 to create more delay between updates to potentially reduce CPU utilization at the cost of slower convergence. We can confirm it by checking the OSPF process summary.

```
! CSR7
router ospf 92
  timers pacing flood 75

R7#show ip ospf | include flood_pacing
  Interface flood pacing timer 75 msec
```

timers pacing retransmission: This timer determines how frequently to retransmit unacknowledged LSAs on a link using the logic described above. The “flood” timer controls the inter-packet flood delay when packets are first flooded, while the “retransmission” timer controls the subsequent inter-packet flood delay which, in a perfect world, would not occur. It makes sense to make the retransmission timer slower than the flood timer to conserve router resources, and by default it is 66 ms (twice the flood timer). The LSAs candidate for retransmission are placed in an interface-level retransmission-list, much

like the flood-list above. The flood/retransmission timers accomplish the same thing but apply to different queues. The configuration and verification is below.

```
! CSR7
router ospf 92
  timers pacing retransmission 150

R7#show ip ospf | include Retrans
  Retransmission pacing timer 150 msec
```

Like the flood-list, each interface keeps track of the LSAs requiring retransmission as well. While the queue is empty, the command to see those LSAs is below.

```
R7#show ip ospf retransmission-list GigabitEthernet2.567

      OSPF Router with ID (92.0.0.7) (Process ID 92)

Neighbor 92.0.0.6, interface GigabitEthernet2.567 address 92.6.7.6
```

timers pacing lsa-group: This timer is meant to optimize the refresh process for aging LSAs. Rather than refresh all LSAs at the 30 minute mark regardless of age, this timer groups LSAs together with similar ages. The default is 240 seconds (4 minutes), which means that LSAs within 4 minutes of one another are refreshed together. This is sometimes referred to as “controlled bursting” and is a better approach than refreshing all LSAs at the 30 minute mark (half of MAXAGE) or refreshing them individually. The smaller the number, the more frequent (but less inclusive) the bursts. We can effectively disable the feature by setting the timer to its maximum value of 1800 (30 minutes). We could also approximate the “individual LSA” refresh approach by using the minimum of 10 seconds. I use a value of 2 minutes, which means there will be twice as many refreshes per 30 minute interval, but each refresh will contain approximately half as many LSAs. Testing these pacing timers is difficult to do in a timely manner, so we will limit verification to some show commands.

```
! CSR7
router ospf 92
  timers pacing lsa-group 120

R7#show ip ospf | include LSA_group
  LSA group pacing timer 120 secs
```

We can do a little extra verification for the LSA-group timer since there is a special command for it. We can see that the grouped updates are approximately 120 seconds apart, and the interval is displayed for sanity. This helps us verify that the configuration is correct. We can also see that the next update is due in about 65 seconds; this number is always less than the LSA-group interval.

```
R7#show ip ospf timers lsa-group
```

OSPF Router with ID (92.0.0.7) (Process ID 92)

```
Group size 8, Head 2, Search Index 4, Interval 120 sec
Next update due in 00:01:05
Current time 550287
Index 0 Timestamp 550353
Index 1 Timestamp 550475
Index 2 Timestamp 550595
Index 3 Timestamp 550723
Index 4 Timestamp 550845
Index 5 Timestamp 550966
Index 6 Timestamp 551089
Index 7 Timestamp 551212
```

Failure Head 0, Last 0 LSA group failure logged

queue-depth: These commands are not well documented as the IOS XE 3S documentation makes no mention of them. They are self-explanatory though; you can control how many OSPF hellos/updates may exist in the software queue for processing. This will ensure the router doesn't get stuck processing OSPF packets for forever, but also allows to router to have some backlog mechanism. Below, I allow up to 64 hello packets to be queued, but an unlimited number of LSUs. There does not appear to be a show command to verify this.

```
! CSR7
router ospf 92
  queue-depth update unlimited
  queue-depth hello 64
```

ispf: When enabled, SPF will only recalculate the part of the tree that requires it. This works identically in logic to the IS-IS feature seen earlier. We can verify it by looking at the OSPF summary.

```
! CSR7
router ospf 92
  ispf
```

```
R7#show ip ospf | include Increment
Incremental-SPF enabled
```

We can quickly test this by changing the IGP cost on CSR4's loopback (not shown). A minor change like this would cause an IS-IS PRC without needing iSPF enabled, but for OSPFv2, this is an intra-area change to the LSA1, which normally is a full SPF update. With iSPF, this alleviates the need for CSR7 to recompute paths to CSR6, XRv2 or XRv1. OSPF invokes an "Increment" SPF mechanism which, based on the minor change from CSR4's LSA1, can be significantly more CPU-efficient than a full SPF run.

```
! CSR7
OSPF-92 SPF: Detect change in LSA type 1, LSID 92.0.0.4 from 92.0.0.4 area 0
```

```

OSPF-92 INTRA: Insert LSA to New_LSA list type 1, LSID 92.0.0.4, from
92.0.0.4 area 0
OSPF-92 SPF: Detect change in LSA type 3, LSID 92.0.0.4 from 92.0.0.11 area 1
OSPF-92 SPF : Do not schedule partial SPF type 3, LSID 92.0.0.4, adv_rtr
92.0.0.11, area 1: INTRA/INTER spf scheduled
OSPF-92 MON : Begin SPF at 554752.098ms, process time 10137ms
OSPF-92 INTRA: Running SPF for area 0, SPF-type Incremental
OSPF-92 INTRA: Initializing to run spf
OSPF-92 INTRA: Running incremental SPF for area 0
[snip]

```

The last OSPFv2 convergence topic is knowing how to read the SPF statistics page, which can be cryptic since the command provides no legend. On CSR7, we can see several SPF runs over the past 24 hours. Below are the key field definitions.

Delta T: Time that has passed from when a given SPF run started to the current time. In the example below, this was about 20 hours ago. Essentially, it says “I started an SPF run this long ago”.

Intra: Time in ms for SPF to process intra-area LSAs, such as LSA1 and LSA2, and to build the topology graph within an area. The timer also accounts for installing intra-area routes in the RIB, which can be a lengthy process. If a topology is densely connected and/or has many IP prefixes, this number can be large.

D-Intra: Same concept as “Intra” except measures the time required to delete intra-area routes (O) from the RIB, measured in ms.

Summ: Same concept as “Intra” except measures the time required to run partial SPF on inter-area LSAs (LSA3) and install them in the RIB. This is also measured in ms.

D-Summ: Same concept as “Summ” except measures the time required to delete inter-area routes (O IA) from the RIB, measured in ms.

Ext: Same concept as “Intra” and “Summ” except measures the time required to run partial SPF on external LSAs (LSA5 and LSA7) and install them in the RIB. This is also measured in ms.

D-Ext: Same concept as “Ext” except measures the time required to delete external (E and N) routes from the RIB, measured in ms.

Total: The sum of all aforementioned counters, measured in ms. This is how long it took the entire SPF process to do everything required, to include additions and deletes for all OSPF route types.

Reason: This is a code that justifies the need to run SPF.

R: A change in a router LSA (type 1) has occurred. This means full SPF (or possibly iSPF) must run within the area since the topology changed or an intra-area prefix changed.

N: A change in a network LSA (type 2) has occurred. This might indicate that a DR failed (which would also be an “R” condition) when the BDR succeeded it, or that a network type was changed to include or exclude the DR election. When the reason is “N”, the reason is often “R” as well.

SA: A change in a summary ASBR LSA (type 4) has occurred. This would change if a new ASBR was detected in an area and the ABRs needed to notify the other areas about it. The opposite is also true; if an ASBR stops performing redistribution, that would trigger a removal of this LSA. Changes in the cost to reach the ASBR would also cause SPF to run using this code.

SN: A change in a summary LSA (type 3) has occurred. If an ABR changes what prefixes it advertises between areas, that would cause a partial SPF run for those prefixes. Changes in the inter-area prefix costs would also trigger SPF using this code.

X: A change in an external LSA (type 5 or 7) has occurred. If an ASBR changes what prefixes it advertises into OSPF via redistribution (or potentially a default route that is not inter-area), that would cause a partial SPF run. Changes in the external prefix costs would also trigger SPF using this code.

```
R7#show ip ospf statistics
```

```
OSPF Router with ID (92.0.0.7) (Process ID 92)
```

```
Area 0: SPF algorithm executed 3 times
```

```
Area 1: SPF algorithm executed 3 times
```

```
Summary OSPF SPF statistic
```

```
SPF calculation time
```

Delta T	Intra	D-Intra	Summ	D-Summ	Ext	D-Ext	Total	Reason
20:22:16	0	0	0	0	0	0	0	R
20:22:11	0	0	0	0	0	0	0	R, SN
00:36:11	0	0	0	0	0	0	0	R, N, SN, SA, X
00:36:10	0	0	0	0	0	0	0	R
00:36:07	0	1	0	0	0	0	1	R, N, SN
00:36:01	0	0	0	0	0	1	1	R, SN, X
00:35:53	0	0	0	0	0	0	0	R, N, SN, SA, X
00:35:52	0	0	0	0	0	1	1	R, SN, X
00:35:48	0	0	0	0	0	0	0	R, N, SN
00:35:45	0	0	0	0	0	0	0	R, SN

Looking at a specific entry, we can see the number of link-state IDs processes of each type, along with all of the routers that were involved in the change. This was mostly likely the result of clearing OSPF on a handful of area 0 routers at once, since all of them had a change.

```
R7#show ip ospf statistics detail
```

```
OSPF Router with ID (92.0.0.7) (Process ID 92)
```

```
Area 0: SPF algorithm executed 8 times
```

```
SPF 1 executed 00:55:34 ago, SPF type Full
```

```
SPF calculation time (in msec):
```

SPT	Intra	D-Intr	Summ	D-Summ	Ext7	D-Ext7	Total
0	0	0	0	0	0	0	0

```
LSIDs processed R:1 N:0 Stub:1 SN:9 SA:0 X7:0
```

```
Change record 0x0
```

```
LSIDs changed 8
```

```
Changed LSAs. Recorded is LS ID and LS type:
```

```
92.0.0.7(R) 92.0.0.4(R) 92.0.0.6(R) 92.0.0.11(R)
```

```
92.0.0.12(R) 92.0.0.11(R) 92.0.0.7(R) 92.0.0.6(R)
```

Before continuing to OSPFv3, we will configure all of these OSPFv2 timers on XRv1 for completeness. Some timers, such as the retransmission timer, do not appear to exist. The retransmission pacing interval appears to be set to twice the pacing flood interval, which is logical behavior. Incremental SPF does not appear to be supported, either.

```
! XRv1
```

```
router ospf 92
```

```
timers throttle lsa all 50 2000 5000
```

```
timers throttle spf 100 2000 10000
```

```
timers lsa group-pacing 360
```

```
timers lsa min-arrival 150
```

```
timers lsa refresh 2400
```

```
timers pacing flood 50
```

```
RP/0/0/CPU0:XRv1#show ospf | include sec
```

```
Initial SPF schedule delay 100 msec
```

```
Minimum hold time between two consecutive SPFs 2000 msec
```

```
Maximum wait time between two consecutive SPFs 10000 msec
```

```
Initial LSA throttle delay 50 msec
```

```
Minimum hold time for LSA throttle 2000 msec
```

```
Maximum wait time for LSA throttle 5000 msec
```

```
Minimum LSA interval 2000 msec. Minimum LSA arrival 150 msec
```

```
LSA refresh interval 2400 seconds
```

```
Flood pacing interval 50 msec. Retransmission pacing interval 100 msec
```

XR also shows the OSPF statistics for SPF, as well as many other features (TE, interface, and other details). XR also gives you a summary of the reason codes which is very handy. If you forget the XE codes, you can use this command on XR since the codes are identical across platforms.

```
RP/0/0/CPU0:XRv1#show ospf statistics spf
SPF statistics for OSPF 92
```

```
Reason Codes: R - Router-LSA, N - Network-LSA,
              SN - Summary-LSA (IP network),
              SA - Summary-LSA (ASBR), X - AS-external-LSA
```

```
Last 40 Dijkstra Calculations
Delta T   Area           Runtime   Reason
1d00h  0                0         R,
1d00h  0                0         R,
1d00h  0                0         R,
23:57:16 0                0         R,
23:57:14 0                0         R,
21:04:49 0                0         R,
21:04:48 0                0         R,
21:04:46 0                0         R,
21:03:56 0                0         R,
[snip]
```

OSPFv3 is very similar to OSPFv2 in terms of how SPF is performed and the timers that are available. Because the original LDP lab did not have OSPFv3 configured, we will configure it quickly on all OSPFv2 routers using the same area boundaries. The configuration is not shown, but we can quickly verify that the loopbacks are all advertised into OSPFv3. Checking the intra and inter-area prefixes, we can see CSR5 and XRv3 are inter-area, while all other loopbacks are in area 0 from XRv2's perspective. CSR6 also originates a default route if the IS-IS IPv6 default route is present, which is an external route.

```
RP/0/0/CPU0:XRv2#show ospfv3 database prefix | include ::92
```

```
Prefix Address: ::92:0:0:4
Prefix Address: ::92:0:0:6
Prefix Address: ::92:0:0:7
Prefix Address: ::92:0:0:11
Prefix Address: ::92:0:0:12
```

```
RP/0/0/CPU0:XRv2#show ospfv3 database inter-area prefix | include ::92
```

```
Prefix Address: ::92:0:0:5
Prefix Address: ::92:0:0:13
Prefix Address: ::92:0:0:13
Prefix Address: ::92:0:0:5
Prefix Address: ::92:0:0:5
Prefix Address: ::92:0:0:13
```

```
R7#show ospfv3 database external ::/0
```

```
OSPFv3 92 address-family ipv6 (router-id 92.0.0.7)
```

Type-5 AS External Link States

```
LS age: 367
LS Type: AS External Link
Link State ID: 0
Advertising Router: 92.0.0.6
LS Seq Number: 80000001
Checksum: 0x70E3
Length: 32
Prefix Address: ::
Prefix Length: 0, Options: None
Metric Type: 2 (Larger than any link state path)
Metric: 1
External Route Tag: 92
```

The convergence timers and queue-depth adjustments in OSPFv3 work identically to those in OSPFv2, so they are not evaluated again. We can literally copy/paste the OSPFv2 configuration on CSR7 into OSPFv3 and it works the same. We verify it with a similar show command. The only convergence feature not also supported in OSPFv3 is incremental SPF.

```
! CSR7
router ospfv3 92
 address-family ipv6 unicast
   timers throttle spf 500 1500 12000
   timers throttle lsa 50 100 500
   timers lsa arrival 500
   timers pacing lsa-group 120
   timers pacing flood 75
   timers pacing retransmission 150
 queue-depth hello 64
 queue-depth update unlimited

R7#show ospfv3 92 ipv6 | include sec
Initial SPF schedule delay 500 msecs
Minimum hold time between two consecutive SPFs 1500 msecs
Maximum wait time between two consecutive SPFs 12000 msecs
Initial LSA throttle delay 50 msecs
Minimum hold time for LSA throttle 100 msecs
Maximum wait time for LSA throttle 500 msecs
Minimum LSA arrival 500 msecs
LSA group pacing timer 120 secs
Interface flood pacing timer 75 msecs
Retransmission pacing timer 150 msecs
```

However, one significant improvement introduced by OSPFv3 is the introduction of the intra-area prefix LSA (type 9). OSPFv3 intelligently decouples the topology information from the IP prefix information into two separate LSAs. Legitimate topology changes, such as failed intra-area links or nodes, will trigger an SPF run (LSA1). Minor changes to prefixes within that area, like IS-IS, should not (LSA9). The OSPFv3 SPF statistics table is very similar to the OSPFv2 table with some new entries. Only the new entries are defined below for brevity.

SPT: Since IP prefixes are decoupled from the tree construction process, this is a new field that represents the time it takes to process all LSA1s and to find shortest paths to each node. This does not include interaction with stub networks (IPv4/v6 prefixes), measured in ms.

Prefix: This is the time it takes to process all stub networks (IPv4/v6 prefixes) and installs them in the RIB, measured in ms. In other words, this processes the LSA9s.

We also see new reason codes of “P” and “L”. The “P” code is for “prefix” and represents a partial SPF recalculation when only LSA9 (prefix) information is changed. The “L” code is for “link” and represents changes to the LSA8 Link LSA, which is not discussed here.

```
R7#show ospfv3 92 statistic
```

```

      OSPFv3 92 address-family ipv6 (router-id 92.0.0.7)

Area 0: SPF algorithm executed 8 times

Area 1: SPF algorithm executed 5 times

SPF calculation time
Delta T   SPT    Prefix D-Int  Sum    D-Sum  Ext    D-Ext  Total Reason
00:23:35  0      0      0      0      0      0      0      0      R N SN SA X L P
00:23:26  0      0      0      0      0      0      0      0      R N SN SA X L P
00:19:40  0      0      0      0      0      0      0      0      R X L P
00:19:30  0      0      0      0      0      0      0      0      R SN X
00:18:53  0      0      0      0      0      0      0      0      R P
00:18:21  0      0      0      0      0      0      0      0      P
00:18:11  0      0      0      0      0      0      0      0      R N SN SA L P
00:16:48  0      0      0      0      0      0      0      0      R N SN X L P
00:16:38  0      0      0      0      0      0      0      0      N SN X P
00:16:19  0      0      0      1      0      0      0      1      P

```

If we clear the OSPFv3 process on CSR7, we can see many subsequent SPF runs where there were many changes. These definitely qualify as full SPF runs, since there are many changes to LSA1 and LSA2.

```
R7#show ospfv3 92 statistic
```

```

[snip]
00:20:35  0      0      0      0      0      0      0      0      R N SN X L P
00:20:25  0      0      0      0      0      0      0      0      N SN X P
00:20:06  0      0      0      1      0      0      0      1      P
00:00:16  0      0      0      0      0      0      0      0      R N SN SA X L P

```

00:00:14	0	0	0	0	0	0	0	0	R	N	SN	X	L	P
00:00:11	0	0	0	0	0	0	0	0	R	SN	X	L		
00:00:05	0	0	0	0	0	0	0	0	R	SN	X	P		

Changing the IGP cost on a stub network, or perhaps adding/deleting the network entirely, registers only as a “P” to indicate a prefix-based recalculation, which is lightweight. In this example, I add a new loopback to CSR6 while debugging on CSR7. This is one of the most significant improvements of OSPFv3 over OSPFv2.

```
R7#debug ospfv3 spf intra
OSPFv3 SPF intra debugging is on for process 92, IPv6, Default vrf
```

```
! CSR6
interface Loopback66
  ipv6 address ::92:0:0:66/128
  ospfv3 92 ipv6 area 0
```

```
! CSR7
OSPFv3-92-IPv6 MON   : Begin SPF at 555747.351, process time 38ms
OSPFv3-92-IPv6 INTRA: Running SPF for area 0, cause P
OSPFv3-92-IPv6 INTRA: Starting Intra-Area SPF (Prefix)
OSPFv3-92-IPv6 INTRA: Process Prefix LSAs
```

```
R7#show ospfv3 92 statistic
[snip]
00:05:14  0      0      0      0      0      0      0      0      0      R  N  SN  SA  X  L  P
00:05:13  0      0      0      0      0      0      0      0      0      R  N  SN  X  L  P
00:05:10  0      0      0      0      0      0      0      0      0      R  SN  X  L
00:05:04  0      0      0      0      0      0      0      0      0      R  SN  X  P
00:00:42  0      0      0      0      0      0      0      0      0      P
```

This is also true for external routes. Since CSR6 is originating an IPv6 default only when it learns one from IS-IS, we can shut down CSR3’s link to the level-1 domain (not shown). This ultimately causes CSR6 to withdraw the external LSA from area 0; with debugging on CSR7, we can see this SPF get scheduled as well.

```
! CSR7
OSPFv3-92-IPv6 SPF   : Schedule partial SPF for LSA 4005/0/92.0.0.6
```

```
R7#show ospfv3 92 statistic
[snip]
00:15:53  0      0      0      0      0      0      0      0      0      R  SN  X  P
00:11:32  0      0      0      0      0      0      0      0      0      P
```

Unfortunately, the SPF statistic table doesn’t show this as reason “X”. In fact, it doesn’t show any new activity at all, although we know it happened. Even when we bring the link on CSR3 back up (not shown), SPF runs again, but the statistics don’t show it. Personally, I find this SPF statistics command to be of

limited value because there are other cases where it records SPF runs only selectively. I assume this is a cosmetic decision because the fundamental IPv6 OSPFv3 routing is fully functional.

```
! CSR7
```

```
OSPFv3-92-IPv6 SPF : Schedule partial SPF for LSA 4005/1/92.0.0.6
```

```
R7#show ospfv3 92 statistic
```

```
[snip]
```

```
00:16:27 0 0 0 0 0 0 0 0 R SN X P
00:12:05 0 0 0 0 0 0 0 0 P
```

XR has similar output, and since this intra-area prefix calculation is an OSPFv3 feature (not a Cisco-specific enhancement), XR also shows “PFX” as a reason to run partial SPF inside an area. In this example, the SPF run was actually a full run since we see the “R” and “N” codes also, but I illustrate that XR uses “PFX” and not “P” to indicate a prefix-based SPF run. “ASE” stands for AS external, which is equivalent to “X” on XE.

```
RP/0/0/CPU0:XRv1#show ospfv3 92 statistics spf
OSPFv3 router 92.0.0.11 (Process 92)
```

```
OSPFv3 SPF statistics summary
SPF algorithm executed 14 times
```

```
Area 0 executed 14 times
```

```
Area 1 executed 14 times
```

```
SPF calculation times (in msec)
```

Delta T	Dijkstra	Intra	D-Intra	Inter	D-Inter	Ext	D-Ext	Reason
00:28:10	0	0	0	0	0	0	0	
00:28:10	0	0	0	0	0	0	0	R
00:28:10	0	0	0	0	0	0	0	R N
00:28:09	0	0	0	0	0	0	0	R N PFX ASE

```
[snip]
```

Regarding OSPFv3 timers on XR, they are very similar to the OSPFv2 timers on XR. Some of them aren’t supported (LSA refresh, etc) but the syntax is otherwise similar, if not identical. They have the same effect as the OSPFv2 timers and are not tested in detail.

```
! XRv1
```

```
router ospfv3 92
```

```
timers lsa arrival 150
```

```
timers pacing flood 50
```

```
timers pacing lsa-group 360
```

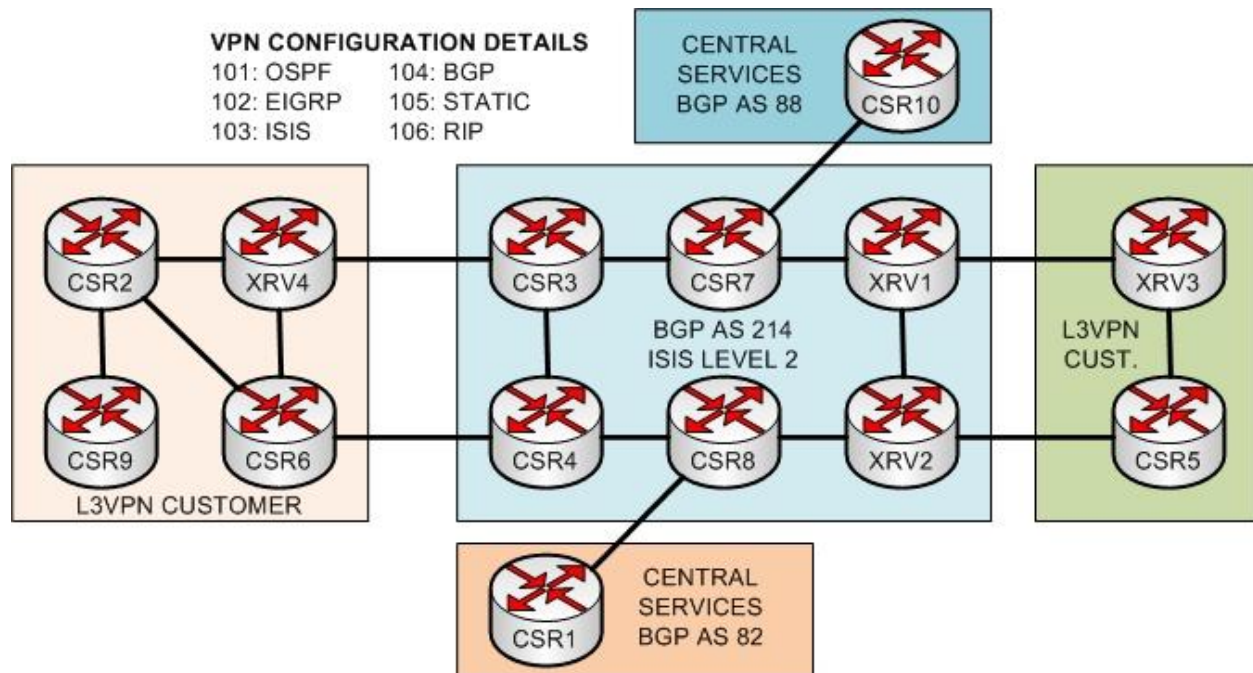
```
timers throttle lsa all 50 2000 5000
```

```
timers throttle spf 100 2000 10000
```

38. Describe, implement, and troubleshoot multi-VRF CE and advanced VRF techniques

This entire section uses a large topology with an MPLS core and several customer VRFs. Every combination of XE and XR serving in PE and CE roles is represented. VRF-Lite for all protocols is tested on

both XE and XR (where supported). Backdoor links for OSPF, EIGRP, and BGP are also examined, along with many other VRF features. The network diagram is below. CSR10 and CSR1 are meant to be generalized “central services” sites so their RTs can be imported/exported as needed for testing. CSR2 and CSR9 are used for traffic leaking and are not part of the MPLS-specific testing. Note that this section is very configuration intensive with the vast majority of it being highly repetitive. Not all configurations will be shown “in line” with the documentation as a result.



Additional Reading - Reference configurations "vrf"

38.1 Multi-VRF CE (VRF-Lite)

This section merges all of the PE-CE routing protocols together with VRF-lite on the CEs. The PE-CE routing techniques are the generally same whether the CE uses VRF-lite or not, but for brevity they are combined here with a single analysis.

38.1.1 Basic VRF-Lite

VRF-Lite is not a difficult feature. Just like a VLAN represents a virtual switch, a VRF represents a virtual router. Unless manually configured, traffic/routes do not leak between VRFs, which provides good software isolation between tables. This is especially useful for modeling larger networks from a single router. VRFs were initially designed for MPLS L3VPN PE routers so that customer routes could be kept isolated from other customers; for L3VPN, the main motivation was overlapping address space within different customer networks. VRF-Lite removes the MPLS overtone and simply segments a router into SVRs, as discussed earlier. The concept of RD is only relevant when the VRF needs to participate in any kind of BGP routing. XR correctly places the RD configuration under BGP because it only has relevance to BGP. In XE, we don't need to define an RD under the VRF unless BGP is used. RTs serve no purpose for

VRF-Lite (assuming no route leaking) and can be skipped for now. Notice that the non-BGP related VRFs don't have RDs configured. We still have to initialize each AFI we want the VRF to support; this causes the router to allocate the appropriate RIB/FIB constructs for the specified AFIs.

```
! Basic XE VRF-Lite
vrf definition BGP
  rd 214:104
  address-family ipv4
  address-family ipv6

vrf definition EIGRP
  address-family ipv4
  address-family ipv6

! Basic XR VRF-Lite
vrf BGP
  address-family ipv4 unicast
  address-family ipv6 unicast

vrf RIP
  address-family ipv4 unicast
  address-family ipv6 unicast

router bgp 104
  vrf BGP
    rd 214:104
```

Verifying the configuration is very straightforward. Since there are no advanced VRF features activated (yet), we can use the XE summary commands to verify the VRFs. The only real difference between any pair of VRFs might be the presence or absence of an RD. We can also see the AFIs and interfaces.

```
R5#show vrf ISIS
  Name          Default RD    Protocols    Interfaces
  ISIS          <not set>    ipv4,ipv6    Gi2.103
                                     Lo103

R5#show vrf BGP
  Name          Default RD    Protocols    Interfaces
  BGP           214:104     ipv4,ipv6    Gi2.104
                                     Lo104
                                     Gi2.204
```

In XR, the interfaces are not shown in summary form, but are revealed in the VRF details. We can verify the RD, interface, and AFIs. Not seeing IPv6 enabled under the first VRF, for example, would be a good troubleshooting reason why something like OSPFv3 was not working.

```
RP/0/0/CPU0:XRv3#show vrf OSPF detail | exclude No
VRF OSPF; RD not set; VPN ID not set
```



```
VRF mode: Regular
Description not set
Interfaces:
  GigabitEthernet0/0/0/0.101
  Loopback101
  GigabitEthernet0/0/0/0.201
Address family IPV4 Unicast
Address family IPV6 Unicast
```

```
RP/0/0/CPU0:XRv3#show vrf BGP detail | exclude No
VRF BGP; RD 214:104; VPN ID not set
VRF mode: Regular
Description not set
Interfaces:
  GigabitEthernet0/0/0/0.104
  Loopback104
  GigabitEthernet0/0/0/0.204
Address family IPV4 Unicast
Address family IPV6 Unicast
```

The show command syntax in both platforms changes when introducing VRFs. In XR, the syntax is very deterministic, generally in the format of “show (process) [vrf (vrf-name)] [afi] (feature)” where the VRF and AFI fields are optional. Below are some examples.

```
RP/0/0/CPU0:XRv3#show ipv4 vrf BGP interface brief loopback104
```

Interface	IP-Address	Status	Protocol
Loopback104	192.104.13.13	Up	Up

```
RP/0/0/CPU0:XRv3#show eigrp vrf EIGRP ipv6 neighbors gig0/0/0/0.102
```

IPv6-EIGRP VR(VPN) Neighbors for AS(102) VRF EIGRP							
H	Address	Interface	Hold Uptime (sec)	SRTT (ms)	RTO	Q Cnt	Seq Num
0	Link Local Address: fe80::11	Gi0/0/0/0.102	14 12:43:50	16	200	0	27

```
RP/0/0/CPU0:XRv3#show ospf vrf OSPF our-address
```

Local Address Database for ospf 101, VRF OSPF with ID 0.0.101.13	
192.5.101.13	GigabitEthernet0/0/0/0.201
192.11.101.13	GigabitEthernet0/0/0/0.101
192.101.13.13	Loopback101

The XE syntax also includes the “vrf” keyword for many show commands, but is less deterministic/consistent. For example, the OSPFv2 show commands never reference a VRF, just the process ID, which might be mapped to a VRF. The EIGRP show commands reference the VRF after the AFI, which is backwards compared to XR. The RIP command contains the process and feature before specifying the VRF, which is also inconsistent with the XE EIGRP and all XR commands. In short, the XE commands are inconsistent, which is one of the reasons XR was designed to be simple.

```
R5#show ip ospf 101 interface brief
```

Interface	PID	Area	IP Address/Mask	Cost	State	Nbrs	F/C
Lo101	101	0	192.101.5.5/32	1	LOOP	0/0	
Gi2.201	101	0	192.5.101.5/24	1	P2P	1/1	
Gi2.101	101	0	192.12.101.5/24	1	P2P	1/1	

```
R5#show eigrp address-family ipv4 vrf EIGRP neighbors gig2.102
```

```
EIGRP-IPv4 VR(VPN) Address-Family Neighbors for AS(102)
```

```
VRF(EIGRP)
```

H	Address	Interface	Hold (sec)	Uptime	SRTT (ms)	RTO	Q Cnt	Seq Num
0	192.12.102.12	Gi2.102	10	13:31:27	15	100	0	19

```
R5#show ip rip database vrf RIP 192.106.14.14 255.255.255.255
```

```
192.106.14.14/32
```

```
[2] via 192.12.106.12, 00:00:12, GigabitEthernet2.106
```

Interfaces are assigned to VRFs by putting the VRF under that interface. The XR VRF interface syntax is identical to the VRF definition and BGP VRF configuration; it is very consistent. XE uses different commands for each one. Changing the VRF of an interface on XE will remove all IPv4/v6 addresses from that interface. To change it on XR, you must remove all IPv4/v6 addresses manually, remove/change the VRF, then add the IPv4/v6 addresses back along with the new VRF (if desired).

```
! XRv3
interface Loopback101
 vrf OSPF
 ipv4 address 192.101.13.13 255.255.255.255
 ipv6 address ::192:101:13:13/128
```

```
! CSR5
interface Loopback101
 vrf forwarding OSPF
 ip address 192.101.5.5 255.255.255.255
 ipv6 address ::192:101:5:5/128
```

38.1.2 OSPF and sham-links

Both OSPFv2 and OSPFv3 can be used as on both XE and XR as a PE-CE routing protocol. In a traditional environment where the CE routing is not running a VRF, the PE routing would run the OSPF process inside of the customer VRF while the CE would run OSPF in the global routing table. Using VRF-Lite on the CE, both sites will be running it in a VRF. Because the configurations for PE and CE are very similar up to this point, only one of each is shown. CSR5 is a CE and XRv2 is a PE. OSPFv3 instance numbers are shown here just for variety and have nothing to do with VRF-Lite; the value is link-local and must match on a segment for a neighbor to form. This is true whether OSPFv3 is inside a VRF or the global table.

```
! CSR5
interface GigabitEthernet2.101
```

```

vrf forwarding OSPF
ip ospf network point-to-point
ip ospf 101 area 0
ospfv3 network point-to-point
ospfv3 101 ipv6 area 0 instance 12

router ospfv3 101
  address-family ipv6 unicast vrf OSPF

router ospf 101 vrf OSPF

! XRv2
interface GigabitEthernet0/0/0/0.101
  vrf OSPF

router ospf 101
  router-id 0.0.101.12
  vrf OSPF
    area 0
    interface GigabitEthernet0/0/0/0.101
      network point-to-point

router ospfv3 101
  router-id 0.0.101.12
  vrf OSPF
    area 0
    interface GigabitEthernet0/0/0/0.101
      network point-to-point
      instance 12

```

Assuming we replicate these configurations appropriately on all 8 devices (all PEs and CEs), we should have OSPFv2/v3 neighbors everywhere. For brevity, we will verify this from the PE side only on the 4 PE routers. Notice that we do not need to specify an AFI for XR's OSPFv3; the IPv4 AFI is not supported for OSPFv3 on XR, so IPv6 is assumed. We see 4 OSPFv2 and 4 OSPFv3 neighbors, so we can assume that OSPFv2/v3 is working properly.

```

R3#show ip ospf 101 neighbor | include FULL
0.0.101.14      0    FULL/  -          00:00:38    192.3.101.14    Gig2.101

R3#show ospfv3 vrf OSPF ipv6 neighbor | include FULL
0.0.101.14      0    FULL/  -          00:00:30      8              Gig2.101

R4#show ip ospf 101 neighbor | include FULL
192.4.101.6     0    FULL/  -          00:00:31    192.4.101.6    Gig2.101

R4#show ospfv3 vrf OSPF ipv6 neighbor | include FULL
192.4.101.6     0    FULL/  -          00:00:37    17              Gig2.101

```

```

RP/0/0/CPU0:XRv1#show ospf vrf OSPF neighbor | include FULL
0.0.101.13      1      FULL/ -      00:00:30      192.11.101.13      Gig0/0/0/0.101

RP/0/0/CPU0:XRv1#show ospfv3 vrf OSPF neighbor | include FULL
0.0.101.13      1      FULL/ -      00:00:36      13      Gig0/0/0/0.101

RP/0/0/CPU0:XRv2#show ospf vrf OSPF neighbor | include FULL
192.12.101.6    1      FULL/ -      00:00:36      192.12.101.5      Gig0/0/0/0.101

RP/0/0/CPU0:XRv2#show ospfv3 vrf OSPF neighbor | include FULL
192.12.101.6    1      FULL/ -      00:00:37      12      Gig0/0/0/0.101

```

Next, we will quickly check to see if each PE learned OSPFv2/v3 routes from the CE. Each CE has a loopback with a single route. We will check CSR3 and XRv1 only. CSR3 learns the IPv4 and IPv6 routes from XRv4 and XRv1 does the same from XRv3.

```

R3#show ip route vrf OSPF ospf | begin Gate
Gateway of last resort is not set
O      192.101.14.14
      [110/2] via 192.3.101.14, 14:09:56, GigabitEthernet2.101

R3#show ipv6 route vrf OSPF ospf | begin Application
a - Application
O      ::192:101:14:14/128 [110/1]
      via FE80::14, GigabitEthernet2.101

RP/0/0/CPU0:XRv1#show route vrf OSPF ospf
O      192.101.13.13/32 [110/2] via 192.11.101.13, 14:14:37,
GigabitEthernet0/0/0/0.101

RP/0/0/CPU0:XRv1#show route vrf OSPF ipv6 ospf
O      ::192:101:13:13/128
      [110/1] via fe80::13, 14:14:19, GigabitEthernet0/0/0/0.101

```

Next, we must configure bidirectional redistribution between OSPF and BGP. OSPF to BGP allows customers routes to be advertised via VPNv4/v6 (and have labels allocated) to remote BGP peers. Redistributing from BGP into OSPF allows the received BGP routes to be learned by the customer. We will show the configuration on CSR3 and XRv1 only. The connected redistribution on OSPFv3 isn't a requirement, but IOS platforms automatically exclude connected routes when redistributing IPv6 protocols.

```

! CSR3
router bgp 214
 address-family ipv4 vrf OSPF
   redistribute ospf 101

 address-family ipv6 vrf OSPF

```

```

        redistribute ospf 101 include-connected

router ospfv3 101
  address-family ipv6 unicast vrf OSPF
    redistribute bgp 214

router ospf 101 vrf OSPF
  redistribute bgp 214 subnets

! XRv1
router bgp 214
  vrf OSPF
    rd 214:101
    address-family ipv4 unicast
      redistribute ospf 101

    address-family ipv6 unicast
      redistribute ospfv3 101

router ospf 101
  vrf OSPF
    redistribute bgp 214

router ospfv3 101
  vrf OSPF
    redistribute bgp 214

```

At this point, the PEs should have the locally redistributed routes inside their BGP tables, as well as the remote iBGP routes from the remote peer. Both PEs can see the loopbacks of XRv3 and XRv4, the CEs, within the BGP table. This verifies that the OSPF to BGP redistribution was successful at both ends.

```

R3#show bgp vpnv4 unicast vrf OSPF | include 1[34]\.1[34]
  *>i 192.101.13.13/32 214.0.0.11          2      100      0 ?
  *> 192.101.14.14/32 192.3.101.14       2          32768 ?

RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast vrf OSPF | include 1[34]\.1[34]
  *> 192.101.13.13/32 192.11.101.13       2          32768 ?
  *>i192.101.14.14/32 214.0.0.3          2      100      0 ?

R3#show bgp vpnv6 unicast vrf OSPF | include 1[34]\:1[34]
  *>i ::192:101:13:13/128
  *>  ::192:101:14:14/128

RP/0/0/CPU0:XRv1#show bgp vpnv6 unicast vrf OSPF | include 1[34]\:1[34]
  *> ::192:101:13:13/128
  *>i::192:101:14:14/128

```

To verify that the BGP to OSPF redistribution worked, we can check the local OSPF LSDB on each PE to look for the Type 3 (for IPv6) or Type 5 LSA (for IPv4). The reason for the discrepancy is discussed later. For now, we can see that the redistribution worked.

```
R3#show ip ospf 101 database | include Link_States|13\.13
Router Link States (Area 0)
Summary Net Link States (Area 0)
Type-5 AS External Link States
192.101.13.13 192.3.101.3 896 0x8000001A 0x007D87 3489661142
```

```
RP/0/0/CPU0:XRv1#show ospf vrf OSPF database | utility egrep 'Link
States|14\.14
Router Link States (Area 0)
Summary Net Link States (Area 0)
Type-5 AS External Link States
192.101.14.14 0.0.101.11 941 0x8000001a 0x0017a7 3489661142
```

```
R3#show ospfv3 vrf OSPF ipv6 database | include Link_States|13:13
Router Link States (Area 0)
Inter Area Prefix Link States (Area 0)
192.3.101.3 1092 0x8000001A ::192:101:13:13/128
Link (Type-8) Link States (Area 0)
Intra Area Prefix Link States (Area 0)
```

```
RP/0/0/CPU0:XRv1#show ospfv3 vrf OSPF database | utility egrep 'Link
States|14\.14
Router Link States (Area 0)
Inter Area Prefix Link States (Area 0)
0.0.101.11 1002 0x8000001a ::192:101:14:14/128
Link (Type-8) Link States (Area 0)
Intra Area Prefix Link States (Area 0)
```

At this point, one would assume there is full reachability between sites. The tricky part about using OSPF with VRF-Lite on the CEs is the notion of the down bit (DN bit). The PEs will set this bit on LSA3 and LSA5 routes redistributed from BGP which help control the “flow” of the LSA. The idea is that when backdoor links exist, the LSA originated at one PE is flooded to a second PE. That second PE should never install this route as it could create suboptimal routing at best, or a loop at worst. Examining a remote prefix (redistributed from BGP to OSPF locally) on CSR3, we can see the “downward” bit in the LSA options field. IOS routers only recently started setting the DN-bit in their LSA5 as it used to only happen for LSA3. The older mechanism for LSA5 loop-prevention was to use route tags, which still happens for backwards compatibility. Converting the route tag below to hex yields 0xD00000D6, and converting the last 2 bytes back into decimal yields the BGP AS number, which is 214. The output is almost identical for XRv1, as the DN-bit is set and the route-tag represents BGP AS 214.

```
R3#show ip ospf 101 database external 192.101.13.13
OSPF Router with ID (192.3.101.3) (Process ID 101)
```

```

Type-5 AS External Link States
LS age: 1857
Options: (No TOS-capability, DC, Downward)
LS Type: AS External Link
Link State ID: 192.101.13.13 (External Network Number )
Advertising Router: 192.3.101.3
LS Seq Number: 8000001A
Checksum: 0x7D87
Length: 36
Network Mask: /32
    Metric Type: 2 (Larger than any link state path)
    MTID: 0
    Metric: 2
    Forward Address: 0.0.0.0
    External Route Tag: 3489661142

RP/0/0/CPU0:XRv1#show ospf vrf OSPF database external 192.101.14.14
    OSPF Router with ID (0.0.101.11) (Process ID 101, VRF OSPF)
        Type-5 AS External Link States
        LS age: 76
        Options: (No TOS-capability, DC, DN)
        LS Type: AS External Link
        Link State ID: 192.101.14.14 (External Network Number)
        Advertising Router: 0.0.101.11
        LS Seq Number: 8000001b
        Checksum: 0x15a8
        Length: 36
        Network Mask: /32
            Metric Type: 2 (Larger than any link state path)
            TOS: 0
            Metric: 2
            Forward Address: 0.0.0.0
            External Route Tag: 3489661142

```

We quickly check the OSPFv3 LSA3 details to see the DN-bit set on them as well. The same logic applies for any OSPF AFI.

```

R3#show ospfv3 vrf OSPF ipv6 database inter-area prefix ::192:101:13:13/128
    OSPFv3 101 address-family ipv6 vrf OSPF (router-id 192.3.101.3)
    LS age: 1133
    LS Type: Inter Area Prefix Links
    Link State ID: 4
    Advertising Router: 192.3.101.3
    LS Seq Number: 8000001B
    Checksum: 0x2A6D
    Length: 44
    Metric: 1
    Prefix Address: ::192:101:13:13

```

Prefix Length: 128, Options: DN

```
RP/0/0/CPU0:XRv1#show ospfv3 vrf OSPF database inter-area prefix
::192:101:14:14
    OSPFv3 Router with ID (0.0.101.11) (Process ID 101 VRF OSPF)
    Inter Area Prefix Link States (Area 0)
    LS age: 850
    LS Type: Inter Area Prefix Links
    Link State ID: 6
    Advertising Router: 0.0.101.11
    LS Seq Number: 8000001b
    Checksum: 0xf658
    Length: 44
    Metric: 1
    Prefix Address: ::192:101:14:14
    Prefix Length: 128, Options: DN , Priority: Critical
```

If a router is running OSPF inside of a VRF, it automatically assumes that it is a PE for an MPLS network. On CSR6, we can see this by looking at the OSPF summary information for OSPFv2 and v3. This output isn't available on XR, but the assumption is the same, as XRv3 and XRv4 also thinks they are MPLS PEs. These VRF-Lite CE routers clearly are not MPLS PEs and are not connected to the MPLS superbackbone.

```
R6#show ip ospf 101 | include VPN
    Connected to MPLS VPN Superbackbone, VRF OSPF

R6#show ospfv3 vrf OSPF ipv6 | include VPN
    Connected to MPLS VPN Superbackbone
```

The result of this is that all of the CE routers will learn these LSA3/LSA5 inside their LSDBs, but cannot install them in the routing table. Neither CSR6 nor XRv4 make any mention that the LSA has the routing bit set, which means it is not candidate for RIB entry.

```
R6#show ip ospf 101 database external 192.101.13.13
    OSPF Router with ID (192.4.101.6) (Process ID 101)
    Type-5 AS External Link States
    LS age: 1511
    Options: (No TOS-capability, DC, Downward)
    LS Type: AS External Link
    Link State ID: 192.101.13.13 (External Network Number )
    Advertising Router: 192.4.101.4
    LS Seq Number: 8000001B
    Checksum: 0x6D94
    Length: 36
    Network Mask: /32
    Metric Type: 2 (Larger than any link state path)
    MTID: 0
    Metric: 2
```


Forward Address: 0.0.0.0
External Route Tag: 3489661142

```
RP/0/0/CPU0:XRv4#show ospf vrf OSPF database external 192.101.13.13
      OSPF Router with ID (0.0.101.14) (Process ID 101, VRF OSPF)
        Type-5 AS External Link States

LS age: 673
Options: (No TOS-capability, DC, DN)
LS Type: AS External Link
Link State ID: 192.101.13.13 (External Network Number)
Advertising Router: 192.3.101.3
LS Seq Number: 8000001b
Checksum: 0x7b88
Length: 36
Network Mask: /32
    Metric Type: 2 (Larger than any link state path)
    TOS: 0
    Metric: 2
    Forward Address: 0.0.0.0
    External Route Tag: 3489661142
```

The same is true for the IPv6. CSR6 and XRv4 both learn the LSA3 but cannot install it in their RIB. They assume they are MPLS L3VPN PE routers since OSPF is inside a VRF, so installing this route could cause a loop.

```
RP/0/0/CPU0:XRv4#show ospfv3 vrf OSPF database inter-area prefix
::192:101:13:13/128
      OSPFv3 Router with ID (0.0.101.14) (Process ID 101 VRF OSPF)
        Inter Area Prefix Link States (Area 0)

LS age: 1952
LS Type: Inter Area Prefix Links
Link State ID: 4
Advertising Router: 192.3.101.3
LS Seq Number: 8000001b
Checksum: 0x2a6d
Length: 44
Metric: 1
Prefix Address: ::192:101:13:13
Prefix Length: 128, Options: DN , Priority: Critical
```

```
R6#show ospfv3 vrf OSPF ipv6 database inter-area prefix ::192:101:13:13/128
      OSPFv3 101 address-family ipv6 vrf OSPF (router-id 192.101.6.6)

LS age: 1027
LS Type: Inter Area Prefix Links
Link State ID: 3
Advertising Router: 192.4.101.4
LS Seq Number: 8000001C
Checksum: 0x2471
```

```
Length: 44
Metric: 1
Prefix Address: ::192:101:13:13
Prefix Length: 128, Options: DN
```

To solve this problem, we need to notify OSPF that it is running inside of a VRF-Lite environment and to ignore the DN bit. The two essentially mean the same thing, and I phrase it this way because XE and XR have different syntaxes. XE is consistent using the “capability vrf-lite” syntax for both OSPFv2 and v3. XR is less consistent since it uses “capability vrf-lite” for OSPFv3 and “disable-dn-bit-check” for OSPFv2. On XE only, applying this command will reset the OSPFv2/v3 neighbors.

```
! CSR6
router ospfv3 101
address-family ipv6 unicast vrf OSPF
  capability vrf-lite

router ospf 101 vrf OSPF
  capability vrf-lite

! XRv4
router ospf 101
  vrf OSPF
  disable-dn-bit-check

router ospfv3 101
  vrf OSPF
  capability vrf-lite
```

In most of the outputs seen below, the routing bit will now be set on the LSA3/LSA5 on the CE routers. The DN-bit is still being set by the PEs, but the CEs are ignoring it entirely. Oddly, the routing bit output does not appear on the XE LSAs at all, which used to happen in older IOS versions. XR clearly displays it, as shown below.

```
R6#show ip ospf 101 database external 192.101.13.13
      OSPF Router with ID (192.4.101.6) (Process ID 101)
      Type-5 AS External Link States
LS age: 1392
Options: (No TOS-capability, DC, Downward)
[snip]

R6#show ospfv3 vrf OSPF ipv6 database inter-area prefix ::192:101:13:13/128
      OSPFv3 101 address-family ipv6 vrf OSPF (router-id 192.101.6.6)
LS age: 1644
LS Type: Inter Area Prefix Links
[snip]
```

```
RP/0/0/CPU0:XRv4#show ospf vrf OSPF database external 192.101.13.13
```

```

        OSPF Router with ID (0.0.101.14) (Process ID 101, VRF OSPF)
        Type-5 AS External Link States
        Routing Bit Set on this LSA
        LS age: 246
        Options: (No TOS-capability, DC, DN)
        LS Type: AS External Link
        Link State ID: 192.101.13.13 (External Network Number)
        [snip]
        Network Mask: /32
        [snip]

RP/0/0/CPU0:XRv4#show ospfv3 vrf OSPF database inter-area prefix
::192:101:13:13/128
        OSPFv3 Router with ID (0.0.101.14) (Process ID 101 VRF OSPF)
        Inter Area Prefix Link States (Area 0)
        Routing Bit Set on this LSA
        [snip]
        Prefix Address: ::192:101:13:13
        Prefix Length: 128, Options: DN , Priority: Critical

```

Quickly checking the routes on CSR6, we can see all the routes are installed to the remote loopbacks. Notice that some are external and some are inter-area. Specifically, the routes reachable via the XR PEs are external, and the route via the CSR PE is inter-area. These discrepancies are resolved later.

```

R6#show ip route vrf OSPF | include 192.101
    192.101.5.0/32 is subnetted, 1 subnets
O E2    192.101.5.5 [110/2] via 192.4.101.4, 00:05:04, GigabitEthernet2.101
    192.101.6.0/32 is subnetted, 1 subnets
C        192.101.6.6 is directly connected, Loopback101
    192.101.13.0/32 is subnetted, 1 subnets
O E2    192.101.13.13 [110/2] via 192.4.101.4, 00:05:04, Gigabit2.101
    192.101.14.0/32 is subnetted, 1 subnets
O IA    192.101.14.14 [110/3] via 192.4.101.4, 00:05:04, Gigabit2.101

```

Looking into the BGP routing information for VPNv4 on a PE, we see there are several new extended communities used by BGP when transporting OSPF routes inside of an L3VPN. Specifically, the routes from the XE PE have a domain-ID encoded in them. The domain-ID type is 0x0005, which is the default in IOS platforms, and the value 0x0065 converted to decimal is 101. This is where the OSPF PID has global significance; it is used by default to seed the domain-ID for an MPLS L3VPN. XR makes no assumptions and does not encode this extended community by default. We can see this by checking the iBGP next-hops of each route and seeing the difference. The routes from XRv1 and XRv2 don't include this domain ID by default.

```

R4#show bgp vpnv4 unicast vrf OSPF 192.101.14.14
BGP routing table entry for 214:101:192.101.14.14/32, version 255
Paths: (2 available, best #2, table OSPF)
    Not advertised to any peer

```

```

[snip]
Refresh Epoch 2
Local
  214.0.0.3 (metric 10) (via default) from 214.0.0.3 (214.0.0.3)
    Origin incomplete, metric 2, localpref 100, valid, internal, best
    Extended Community: RT:214:101 OSPF DOMAIN ID:0x0005:0x000000650200
      OSPF RT:0.0.0.0:2:0 OSPF ROUTER ID:192.3.101.3:0
    mpls labels in/out nolabel/3015
    rx pathid: 0, tx pathid: 0x0

R4#show bgp vpnv4 unicast vrf OSPF 192.101.13.13
BGP routing table entry for 214:101:192.101.13.13/32, version 127
Paths: (2 available, best #1, table OSPF)
  Not advertised to any peer
  Refresh Epoch 2
  Local
    214.0.0.11 (metric 30) (via default) from 214.0.0.3 (214.0.0.3)
      Origin incomplete, metric 2, localpref 100, valid, internal, best
      Extended Community: RT:214:101 OSPF RT:0.0.0.0:1:0
        OSPF ROUTER ID:0.0.101.11:0
      Originator: 214.0.0.11, Cluster list: 0.0.3.12
      mpls labels in/out nolabel/91014
      rx pathid: 0, tx pathid: 0x0
[snip]

```

When the domain-IDs match at remote ends of a VPN, they are considered to be in the same “AS” so to speak. The MPLS network acts like a “super backbone” as a third level of hierarchy greater than area 0. This allows area 0 networks to be bridged together and see one another as inter-area. When the domain-ID does not match, BGP makes no assumptions about this continuity and treats the routes as external when redistributing them. If one router has a domain-ID and the other does not, this is considered a non-match. We could set the domain-ID to be the same as we saw above on the routes from CSR3, but for variety, we will change it to a new value of 244 (0xF4) so that we can demonstrate the change on XE platforms as well. We quickly verify that the configuration is correct by checking the OSPF process details. For now, we will limit this change only to OSPFv2 processes on the PEs.

```

! CSR4
router ospf 101 vrf OSPF
  domain-id type 0005 value 000000F40200

! XRv1
router ospf 101
  vrf OSPF
    domain-id type 0005 value 000000F40200

R4#show ip ospf 101 | include Domain
  Domain ID type 0x0005, value 0x000000F40200

```

```
RP/0/0/CPU0:XRv1#show ospf vrf OSPF | begin Domain
Primary Domain ID:
0x5:0x000000f40200
[snip]
```

Now, all routes learned by CSR4 will have the same domain-ID. Provided these extended communities match the local domain-ID (verified above), then the routes are considered inter-area.

```
R4#show bgp vpnv4 unicast vrf OSPF 192.101.13.13
BGP routing table entry for 214:101:192.101.13.13/32, version 324
Paths: (2 available, best #1, table OSPF)
  Not advertised to any peer
  Refresh Epoch 2
  Local
    214.0.0.11 (metric 30) (via default) from 214.0.0.3 (214.0.0.3)
      Origin incomplete, metric 2, localpref 100, valid, internal, best
      Extended Community: RT:214:101 OSPF DOMAIN ID:0x0005:0x000000F40200
        OSPF RT:0.0.0.0:1:0 OSPF ROUTER ID:0.0.101.11:0
      Originator: 214.0.0.11, Cluster list: 0.0.3.12
      mpls labels in/out nolabel/91014
      rx pathid: 0, tx pathid: 0x0
  [snip]
```

```
R4#show bgp vpnv4 unicast vrf OSPF 192.101.14.14 bestpath
BGP routing table entry for 214:101:192.101.14.14/32, version 330
Paths: (2 available, best #2, table OSPF)
  Not advertised to any peer
  Refresh Epoch 4
  Local
    214.0.0.3 (metric 10) (via default) from 214.0.0.3 (214.0.0.3)
      Origin incomplete, metric 2, localpref 100, valid, internal, best
      Extended Community: RT:214:101 OSPF DOMAIN ID:0x0005:0x000000F40200
        OSPF RT:0.0.0.0:2:0 OSPF ROUTER ID:192.3.101.3:0
      mpls labels in/out nolabel/3015
      rx pathid: 0, tx pathid: 0x0
```

We can verify this worked by checking the CE routers to ensure all of their OSPF routes are inter-area. Notice that CSR6 shows 8 inter-area routes and 0 external type-2 routes, which is what we had before. XRv4 doesn't show 0 external routes, it just displays nothing except the inter-area routes, which is an implicit verification method.

```
R6#show ip route vrf OSPF summary | section ospf 101
ospf 101          5          3          0          768          2336
  Intra-area: 0  Inter-area: 8  External-1: 0  External-2: 0
  NSSA External-1: 0  NSSA External-2: 0
```

```
RP/0/0/CPU0:XRv4#show route vrf OSPF summary detail | begin ospf 101
ospf 101          7          7          0          0
```

Inter-area:	7	7	0	0
Total	12	12	1	1

The concept is identical for IPv6 routes. The only difference is that suddenly XE no longer makes assumptions about the domain-ID. At this point, neither XE nor XR PEs will encode a domain-ID since neither of them consider the OSPF PID as a seed value.

```
R4#show bgp vpnv6 unicast vrf OSPF ::192:101:13:13/128
BGP routing table entry for [214:101]::192:101:13:13/128, version 363
Paths: (2 available, best #1, table OSPF)
  Not advertised to any peer
  Refresh Epoch 6
  Local
    ::FFFF:214.0.0.11 (metric 30) (via default) from 214.0.0.3 (214.0.0.3)
      Origin incomplete, metric 1, localpref 100, valid, internal, best
      Extended Community: RT:214:101 OSPF RT:0.0.0.0:2:0
      OSPF ROUTER ID:0.0.101.11:0
      Originator: 214.0.0.11, Cluster list: 0.0.3.12
      mpls labels in/out nlabel/91017
      rx pathid: 0, tx pathid: 0x0
    [snip]
```

```
R4#show bgp vpnv6 unicast vrf OSPF ::192:101:14:14/128
BGP routing table entry for [214:101]::192:101:14:14/128, version 368
Paths: (2 available, best #2, table OSPF)
  Not advertised to any peer
  [snip]
  Refresh Epoch 6
  Local
    ::FFFF:214.0.0.3 (metric 10) (via default) from 214.0.0.3 (214.0.0.3)
      Origin incomplete, metric 1, localpref 100, valid, internal, best
      Extended Community: RT:214:101 OSPF ROUTER ID:192.3.101.3:0
      OSPF RT:0.0.0.0:2:0
      mpls labels in/out nlabel/3016
      rx pathid: 0, tx pathid: 0x0
```

Checking the local domain-IDs on CSR4 and XRv1, we see that there are no defaults. Personally, I prefer the XR behavior because it never makes a domain-ID assumption and is consistent between OSPFv2 and OSPFv3. XE makes assumptions about the OSPFv2 domain-ID only by using the OSPF PID, which is inconsistent.

```
R4#show ospfv3 vrf OSPF | include Domain
Domain ID (none)
```

```
RP/0/0/CPU0:XRv1#show ospfv3 vrf OSPF | begin Domain
[no output]
```

Since there are no domain-IDs carried in any VPNv6 routes and also no local domain-IDs, OSPF makes the bold assumption that the routes are indeed in the same domain. This behavior is in accordance with RFC 4577 and is not a Cisco-specific behavior. We can confirm this on CSR6 and XRv4. At this point, we could consider our configuration “complete” since all of the OSPFv2/v3 CE routers see one another’s routes as inter-area (not external).

```
R6#show ipv6 route vrf OSPF summary | section ospf 101
```

```
ospf 101      8      1152      1664
  Intra-area: 0  Inter-area: 8  External-1: 0  External-2: 0
  NSSA External 1: 0  NSSA External 2: 0
```

```
RP/0/0/CPU0:XRv4#show route vrf OSPF ipv6 summary detail | begin ospf 101
```

```
ospf 101      7      7      0      0
  Inter-area: 7      7      0      0
Total      12      12      1      1
```

We will first configure the domain-ID on all of the XE PEs only, this time using value 246 (0xF6). This should introduce some external routes into the VPN routing tables since routes from CSR3 and CSR4 will have domain-IDs. XRv1 and XRv2 will not; this behavior would mimic the original issue observed with OSPFv2 where XE PEs added domain-IDs but XR PEs did not. After configuring the feature, we confirm it on both routers.

```
! CSR3 and CSR4
router ospfv3 101
  address-family ipv6 unicast vrf OSPF
    domain-id type 0005 value 000000F60200
```

```
R3#show ospfv3 vrf OSPF | include Domain
Domain ID 0x000000F60200, type 0x0005
```

```
R4#show ospfv3 vrf OSPF | include Domain
Domain ID 0x000000F60200, type 0x0005
```

Checking CSR4, we can see that VPNv6 prefixes from CSR3 have this new domain-ID, which matches the local domain-ID, and therefore means routes from CSR3 will be inter-area. Routes from the remote XR PEs have no domain-ID, and are therefore external. Both CSR6 and XRv14 now see a mix of inter-area and external routes.

```
R4#show bgp vpnv6 unicast vrf OSPF ::192:101:13:13/128
BGP routing table entry for [214:101]::192:101:13:13/128, version 363
Paths: (2 available, best #1, table OSPF)
  Not advertised to any peer
  Refresh Epoch 8
  Local
    ::FFFF:214.0.0.11 (metric 30) (via default) from 214.0.0.3 (214.0.0.3)
    Origin incomplete, metric 1, localpref 100, valid, internal, best
```

```

Extended Community: RT:214:101 OSPF RT:0.0.0.0:2:0
  OSPF ROUTER ID:0.0.101.11:0
  Originator: 214.0.0.11, Cluster list: 0.0.3.12
  mpls labels in/out nolabel/91017
  rx pathid: 0, tx pathid: 0x0
[snip]

R4#show bgp vpnv6 unicast vrf OSPF ::192:101:14:14/128
BGP routing table entry for [214:101]::192:101:14:14/128, version 377
Paths: (2 available, best #2, table OSPF)
  Not advertised to any peer
  [snip]
  Refresh Epoch 8
  Local
    ::FFFF:214.0.0.3 (metric 10) (via default) from 214.0.0.3 (214.0.0.3)
    Origin incomplete, metric 1, localpref 100, valid, internal, best
    Extended Community: RT:214:101 OSPF DOMAIN ID:0x0005:0x000000F60200
    OSPF ROUTER ID:192.3.101.3:0 OSPF RT:0.0.0.0:2:0
    mpls labels in/out nolabel/3016
    rx pathid: 0, tx pathid: 0x0

R6#show ipv6 route vrf OSPF summary | section ospf 101
ospf 101          8          1152          1664
  Intra-area: 0   Inter-area: 3   External-1: 0   External-2: 5
  NSSA External 1: 0   NSSA External 2: 0

RP/0/0/CPU0:XRv4#show route vrf OSPF ipv6 summary detail | begin ospf 101
ospf 101          7          7          0          0
  Inter-area: 2          2          0          0
  External-2: 5          5          0          0
Total            12          12          1          1

```

For completeness, we will configure the IPv6 domain-ID on the XR PEs as well, and quickly verify that the configuration was successful. As mentioned earlier, XR is more consistent with domain-ID configuration and default behavior than XE.

```

! XRv1 and XRv2
router ospfv3 101
  vrf OSPF
    domain-id type 0005 value 000000F60200

RP/0/0/CPU0:XRv1#show ospfv3 vrf OSPF | begin Domain
Primary Domain ID:
  0x0005:0x000000f60200

RP/0/0/CPU0:XRv2#show ospfv3 vrf OSPF | begin Domain
Primary Domain ID:
  0x0005:0x000000f60200

```


Finally, we will verify that the XR-originated VPNv6 prefixes are carrying the proper domain-ID extended community, and that the XE routers only see inter-area routes again.

```
R4#show bgp vpnv6 unicast vrf OSPF ::192:101:13:13/128
BGP routing table entry for [214:101]::192:101:13:13/128, version 380
Paths: (2 available, best #1, table OSPF)
  Not advertised to any peer
  Refresh Epoch 8
  Local
    ::FFFF:214.0.0.11 (metric 30) (via default) from 214.0.0.3 (214.0.0.3)
      Origin incomplete, metric 1, localpref 100, valid, internal, best
      Extended Community: RT:214:101 OSPF DOMAIN ID:0x0005:0x000000F60200
        OSPF RT:0.0.0.0:2:0 OSPF ROUTER ID:0.0.101.11:0
      Originator: 214.0.0.11, Cluster list: 0.0.3.12
      mpls labels in/out nolabel/91017
      rx pathid: 0, tx pathid: 0x0
      [snip]
```

```
R4#show bgp vpnv6 unicast vrf OSPF ::192:101:14:14/128
BGP routing table entry for [214:101]::192:101:14:14/128, version 377
Paths: (2 available, best #2, table OSPF)
  Not advertised to any peer
  [snip]
  Refresh Epoch 8
  Local
    ::FFFF:214.0.0.3 (metric 10) (via default) from 214.0.0.3 (214.0.0.3)
      Origin incomplete, metric 1, localpref 100, valid, internal, best
      Extended Community: RT:214:101 OSPF DOMAIN ID:0x0005:0x000000F60200
        OSPF ROUTER ID:192.3.101.3:0 OSPF RT:0.0.0.0:2:0
      mpls labels in/out nolabel/3016
      rx pathid: 0, tx pathid: 0x0
```

When we try to verify this claim, we find some unanticipated results. CSR6 and XRv4 still see external routes, which doesn't make sense.

```
R6#show ipv6 route vrf OSPF summary | section ospf 101
ospf 101          8          1152          1664
  Intra-area: 0   Inter-area: 6   External-1: 0   External-2: 2
  NSSA External 1: 0   NSSA External 2: 0
```

```
RP/0/0/CPU0:XRv4#show route vrf OSPF ipv6 summary detail | begin ospf 101
ospf 101          7          7          0          0
  Inter-area: 5          5          0          0
  External-2: 2          2          0          0
Total            12          12          1          1
```

If we look at the specific routes that are still external, we see that the PE-CE transit links connected to XRv1 and XRv2 are the only ones. For whatever reason, XRv1 and XRv2 did not set the domain-ID on those routes during redistribution, possibly because they were not OSPF-learned. XE is smarter than XR in this regard and considers those are part of the OSPFv3 domain, even as connected routes. I am not able to find a workaround for this since the RPL does not contain a mechanism to manually set this extended community. At least we know that the OSPF-learned routes, which are much more important and relevant, have the extended community applied and the feature is working properly. Inter-area reachability to the transit links is not important, so XR's inability to add the domain-ID to these VPNv6 prefixes is insignificant.

```
R6#show ipv6 route vrf OSPF | include E2
      OE2 - OSPF ext 2, ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2
OE2 FD00:192:11:101::/64 [110/1]
OE2 FD00:192:12:101::/64 [110/1]

RP/0/0/CPU0:XRv4#show route vrf OSPF ipv6 | include E2
      E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
O E2 fd00:192:11:101::/64
O E2 fd00:192:12:101::/64

RP/0/0/CPU0:XRv1#show bgp vpnv6 unicast vrf OSPF fd00:192:11:101::/64 |
include Extended
      Extended community: OSPF route-type:0:2:0x0 OSPF router-id:214.0.0.11
RT:214:101

RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast vrf OSPF fd00:192:12:101::/64 |
include Extended
      Extended community: OSPF route-type:0:2:0x0 OSPF router-id:214.0.0.12
RT:214:101
```

It is worth discussing the other new extended communities and OSPF domain-ID types. First, the OSPF route-type is something not present in normal OSPF/BGP architectures. The attribute carries the same information for both IPv4 and IPv6 routes. XE and XR display them a little differently but the premise is the same. The first 32 bits represent the area-ID, which is 0 in this case (XR shows 0 and XE shows 0.0.0.0). The next 8 bits specify the route type: 1 or 2 for intra-area routes (whether it came from LSA1 or LSA2), 3 for inter-area, 5 for external, and 7 for NSSA-external. The creators of this extended community kept it simple by using the same numbers as the OSPF LSA types. Below, we can see two area 0 routes with route-type 2. Although we don't have any DRs in the network, I am not sure why the route-type is 2 and not 1, but we at least know it is intra-area. The last field of the route-type is the options field, which is 8 bits, and is used to signal information about external routes. Since neither route was external, these flags are 0 (XR shows 0x0 and XE shows 0).

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast vrf OSPF 192.101.6.6/32 | include
Extended
```

```
Extended community: OSPF domain-id:0x5:0x000000f40200 OSPF route-  
type:0:2:0x0 OSPF router-id:192.4.101.4 RT:214:101
```

```
R4#show bgp vpnv6 unicast vrf OSPF ::192:101:14:14/128 | include OSPF_RT  
OSPF ROUTER ID:192.3.101.3:0 OSPF RT:0.0.0.0:2:0
```

The area-ID value is 0 for external routes as well. To see this, we will create a new loopback on CSR6 and redistribute it into OSPFv2 and v3.

```
! CSR6  
interface Loopback600  
description EXTERNAL TEST  
vrf forwarding OSPF  
ip address 192.60.6.6 255.255.255.255  
ipv6 address ::192:60:6:6/128  
  
route-map RM_CONN_TO_OSPF permit 10  
match interface Loopback600  
  
router ospfv3 101  
address-family ipv6 unicast vrf OSPF  
redistribute connected route-map RM_CONN_TO_OSPF  
  
router ospf 101 vrf OSPF  
redistribute connected subnets route-map RM_CONN_TO_OSPF
```

We can verify the redistribution was successful by checking the local LSDB. Note: The PEs must be configured to match external routes when redistributing from OSPF into BGP. This is not allowed by default and must be explicitly specified. Because there are no external routes on CSR5 or XRv3, we only need to configure external redistribution on CSR3 and CSR4.

```
R6#show ip ospf 101 database external self-originate | include Link|Mask  
Type-5 AS External Link States  
LS Type: AS External Link  
Link State ID: 192.60.6.6 (External Network Number )  
Network Mask: /32
```

```
R6#show ospfv3 vrf OSPF database external self-originate | include Prefix  
Prefix Address: ::192:60:6:6  
Prefix Length: 128, Options: None
```

```
! CSR3 and CSR4  
router bgp 214  
address-family ipv6 vrf OSPF  
redistribute ospf 101 include-connected match internal external  
  
address-family ipv4 vrf OSPF
```

```
redistribute ospf 101 match internal external
```

When we check the route on XRv1, we can see the route-type is 5 to signal an external route. The lower-order bit of the options byte is set to 1 which indicates external type-2.

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast vrf OSPF 192.60.6.6/32 | include
Extend
      Extended community: OSPF domain-id:0x5:0x000000f40200 OSPF route-
type:0:5:0x1 OSPF router-id:192.4.101.4 RT:214:101
```

When the route is external type-1, the options field is clear, and the route-type remains 5. We update the route-map to make this change, then tell OSPFv2/v3 to perform the redistribution again to capture the changes. XRv1 shows the new route-type. Also note there is no need to carry the OSPF metric in a new extended community as it is already carried in the MED value. This is standard BGP behavior when redistributing from an IGP, and OSPF can use that to keep metrics consistent across the L3VPN.

```
! CSR6
route-map RM_CONN_TO_OSPF permit 10
  set metric-type type-1

R6#clear ip ospf 101 redistribution
R6#clear ospfv3 vrf OSPF redistribution
```

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast vrf OSPF 192.60.6.6/32 | include
Extend
      Extended community: OSPF domain-id:0x5:0x000000f40200 OSPF route-
type:0:5:0x0 OSPF router-id:192.4.101.4 RT:214:101
```

The OSPF RID community needs no explaining. It contains the OSPF RID of the originating PE for the route in question. Looking at CSR6 and XRv4 loopbacks side by side from XRv2's perspective, we can see different OSPF RID's since the egress PEs differ for those two prefixes.

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf OSPF 192.101.14.14 | include
Extend
      Extended community: OSPF domain-id:0x5:0x000000f40200 OSPF route-
type:0:2:0x0 OSPF router-id:192.3.101.3 RT:214:101

RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf OSPF 192.101.6.6 | include Extend
      Extended community: OSPF domain-id:0x5:0x000000f40200 OSPF route-
type:0:2:0x0 OSPF router-id:192.4.101.4 RT:214:101
```

The domain-ID types are just meant to be differentiators between various domain-ID formats. The Cisco default is 0x005, as we saw earlier. All four domain-ID formats (0x0005, 0x0105, 0x0205, 0x8005) are 8 bytes long where the first 2 bytes always represent the type. Type 0x0005 uses the next 2 bytes for the "global administrator" field and the last 4 for the "local administrator" field, which is where the OSPF

PID is encoded. Types 0x0105 and 0x0205 use 4 bytes for global and 2 bytes for local, but the difference is really just cosmetic. You can set the actual value to whatever is necessary to make the OSPF routes appear the way they need to (sometimes you may want inter-area, sometimes external). The 0x8005 type is exceptional; using 0x8005 is a way to achieve backward compatibility between all domain-ID types. When this domain-ID type is used, the first 2 bytes are ignored, so the domain-ID types no longer need to match. It is a raw bit-for-bit comparison of the remaining 6 bits, which works perfectly for backwards compatibility. We will configure XRv1 with a domain-ID type of 0x8005 but use the same ID value of 0xF4 for OSPFv2.

```
! XRv1
router ospf 101
 vrf OSPF
  domain-id type 8005 value 000000F40200
```

XRv4 still sees the route to XRv3 as inter-area, which proves that CSR3 ignored the difference between 0x0005 and 0x8005.

```
R3#show bgp vpnv4 unicast vrf OSPF 192.101.13.13
BGP routing table entry for 214:101:192.101.13.13/32, version 596
Paths: (1 available, best #1, table OSPF)
  Advertised to update-groups:
    3
  Refresh Epoch 1
  Local, (Received from a RR-client)
    214.0.0.11 (metric 20) (via default) from 214.0.0.11 (214.0.0.11)
      Origin incomplete, metric 2, localpref 100, valid, internal, best
      Extended Community: RT:214:101 OSPF RT:0.0.0.0:1:0
        OSPF ROUTER ID:0.0.101.11:0 OSPF DOMAIN ID:0x8005:0x000000F40200
      mpls labels in/out nlabel/91014
      rx pathid: 0, tx pathid: 0x0
```

```
RP/0/0/CPU0:XRv4#show route vrf OSPF 192.101.13.13
Routing entry for 192.101.13.13/32
  Known via "ospf 101", distance 110, metric 3, type inter area
  Routing Descriptor Blocks
    192.3.101.3, from 192.3.101.3, via GigabitEthernet0/0/0/0.101
      Route metric is 3
  No advertising protos.
```

We will quickly test type 0x8005 on XE using OSPFv3. Changing CSR3's domain-ID to 0x8005, XRv2 still considers it within the same domain since the low-order 6-byte domain value still matches. CSR5 still sees the route as inter-area, as we expect.

```
! CSR3
router ospfv3 101
 address-family ipv6 unicast vrf OSPF
```

```
domain-id type 8005 value 000000F60200
```

```
RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast vrf OSPF ::192:101:14:14
```

```
[snip]
```

```
Local, (Received from a RR-client)
```

```
214.0.0.3 (metric 30) from 214.0.0.3 (214.0.0.3)
```

```
Received Label 3016
```

```
Origin incomplete, metric 1, localpref 100, valid, internal, best,  
group-best, import-candidate, imported
```

```
Received Path ID 0, Local Path ID 1, version 535
```

```
Extended community: OSPF router-id:192.3.101.3 OSPF route-type:0:2:0x0
```

```
OSPF domain-id:0x8005:0x000000f60200 RT:214:101
```

```
Source VRF: OSPF, Source Route Distinguisher: 214:101
```

```
R5#show ipv6 route vrf OSPF ::192:101:14:14
```

```
Routing entry for ::192:101:14:14/128
```

```
Known via "ospf 101", distance 110, metric 2, type inter area
```

```
Route count is 1/1, share count 0
```

```
Routing paths:
```

```
FE80::12, GigabitEthernet2.101
```

```
Last updated 03:20:39 ago
```

Let's assume that between XRv4 and CSR6, and XRv3 and CSR5, that we have installed a pair of low-speed backdoor links. These could be non-MPLS private WAN leased lines that we want to use only for backup, while preferring the L3VPN MPLS path via the PEs primarily. The backdoor link configuration is not shown, but we can quickly check the neighbors on all four CEs for both IPv4/v6 to verify the neighbor was formed properly. We also increase the cost of this link to a very high value in an attempt to prefer MPLS. The backdoor link is Gig2.201 on XE or Gig0/0/0/0.201 on XR and has a cost of 500. We can quickly verify this on CSR6 and XRv4 as an example. Notice that there is 1 neighbor on the link also, as expected. This serves as a quick check to verify the cost configuration and OSPFv2/v3 neighbors.

```
R6#show ip ospf 101 interface brief
```

Interface	PID	Area	IP Address/Mask	Cost	State	Nbrs	F/C
Lo101	101	0	192.101.6.6/32	1	LOOP	0/0	
Gi2.201	101	0	192.6.101.6/24	500	P2P	1/1	
Gi2.101	101	0	192.4.101.6/24	1	P2P	1/1	

```
R6#show ospfv3 101 vrf OSPF ipv6 interface brief
```

Interface	PID	Area	AF	Cost	State	Nbrs	F/C
Lo101	101	0	ipv6	1	LOOP	0/0	
Gi2.201	101	0	ipv6	500	P2P	1/1	
Gi2.101	101	0	ipv6	1	P2P	1/1	

```
RP/0/0/CPU0:XRv4#show ospf vrf OSPF interface brief
```

```
[snip]
```

Interface	PID	Area	IP Address/Mask	Cost	State	Nbrs	F/C
Lo101	101	0	192.101.14.14/32	1	LOOP	0/0	
Gi0/0/0/0.101	101	0	192.3.101.14/24	1	P2P	1/1	

```
Gi0/0/0/0.201      101    0          192.6.101.14/24    500    P2P    1/1
```

```
RP/0/0/CPU0:XRv4#show ospfv3 vrf OSPF interface brief
```

Interface	PID	Area	IPv6 Address	Cost	State	Nbrs	F/C
Gi0/0/0/0.101	101	0	fe80::14	1	P2P	1/1	
Gi0/0/0/0.201	101	0	fe80::14	500	P2P	1/1	
Lo101	101	0	fe80::e92e:ddff:fef3:2b48	0	LOOP	0/0	

When we check the routing table on CSR6, we can see that it is preferring the slow backdoor link over the fast MPLS connection. This is because the backdoor route is intra-area, and is always preferred over inter-area. This OSPF “route preference” behavior cannot be modified and the problem persists for both IPv4 and IPv6. Assigning a different area to the transit link would not work as this would imply bridging area 0 over a non-backbone area. Technically this is possible with virtual links and other awkward workarounds, but that is beyond the scope of this problem.

```
R6#show ip route vrf OSPF 192.101.14.14
```

```
Routing Table: OSPF
```

```
Routing entry for 192.101.14.14/32
```

```
Known via "ospf 101", distance 110, metric 501, type intra area
```

```
Last update from 192.6.101.14 on GigabitEthernet2.201, 00:04:04 ago
```

```
Routing Descriptor Blocks:
```

```
* 192.6.101.14, from 0.0.101.14, 00:04:04 ago, via GigabitEthernet2.201
```

```
Route metric is 501, traffic share count is 1
```

```
R6#show ipv6 route vrf OSPF ::192:101:14:14/128
```

```
Routing entry for ::192:101:14:14/128
```

```
Known via "ospf 101", distance 110, metric 500, type intra area
```

```
Route count is 1/1, share count 0
```

```
Routing paths:
```

```
FE80::14, GigabitEthernet2.201
```

```
Last updated 00:05:52 ago
```

A quick check on XRv4 shows the same problem in the opposite direction. It is preferring the slow backdoor link to CSR6 when we prefer that it use the MPLS link primarily.

```
RP/0/0/CPU0:XRv4#show route vrf OSPF ipv4 192.101.6.6
```

```
Routing entry for 192.101.6.6/32
```

```
Known via "ospf 101", distance 110, metric 501, type intra area
```

```
Routing Descriptor Blocks
```

```
192.6.101.6, from 192.4.101.6, via GigabitEthernet0/0/0/0.201
```

```
Route metric is 501
```

```
No advertising protos.
```

```
RP/0/0/CPU0:XRv4#show route vrf OSPF ipv6 ::192:101:6:6
```

```
Routing entry for ::192:101:6:6/128
```

```
Known via "ospf 101", distance 110, metric 500, type intra area
```

```
Routing Descriptor Blocks
```

```
fe80::6, from ::, via GigabitEthernet0/0/0/0.201
```

```
Route metric is 500
```

```
No advertising protos.
```

The aptly-named “sham-link” can be used to solve this problem. Like a virtual-link, it will connect two non-adjacent OSPF speakers using a demand-circuit that allows LSAs to be exchanged across the network. Sham-link routers must be in the same domain (same or compatible domain-ID type with identical domain-ID value). The sham-link can be used to connect two L3VPN PEs together over an MPLS core to create an intra-area link between them. This would form a square between the PEs and CEs, which would allow the OSPF cost to be the deciding factor in terms of routing. To build a sham link, we must create host-route loopbacks (/32 for IPv4 and /128 for IPv6) which exist inside of the same VPN as the process we intend to bridge. The loopbacks must be advertised into BGP and must not be advertised into the VRF-aware OSPF process; the purpose is to create reachability between these endpoints over the MPLS network, not the customer network. We will configure OSPFv2 and OSPFv3 sham-links between CSR3 and CSR4 first. Note that even if we were using OSPFv3 for the IPv4 AFI, the sham-link endpoint must be a /128 IPv6 address. We can advertise the sham-links into BGP via network statements or redistribution; it does not matter. We will use both methods for variety.

```
! CSR3
interface Loopback101
  description SHAM LINK ENDPOINT
  vrf forwarding OSPF
  ip address 101.0.0.3 255.255.255.255
  ipv6 address ::101:0:0:3/128

router bgp 214
  address-family ipv4 vrf OSPF
    network 101.0.0.3 mask 255.255.255.255

  address-family ipv6 vrf OSPF
    network ::101:0:0:3/128

! CSR4
interface Loopback101
  description SHAM LINK ENDPOINT
  vrf forwarding OSPF
  ip address 101.0.0.4 255.255.255.255
  ipv6 address ::101:0:0:4/128

route-map RM_SHAM_TO_BGP permit 10
  match interface Loopback101

router bgp 214
  address-family ipv4 vrf OSPF
    redistribute connected route-map RM_SHAM_TO_BGP

  address-family ipv6 vrf OSPF
```



```
redistribute connected route-map RM_SHAM_TO_BGP
```

Before configuring the sham-link itself, we should verify that we have connectivity between these sham-link endpoints. The traffic will be MPLS-encapsulated since it is inside a VPN as these look like ordinary VPNv4/v6 routes with BGP allocated labels. We only need to perform the ping test from one PE since the return traffic (echo-reply) confirms bidirectional connectivity.

```
R3#show bgp vpnv4 unicast vrf OSPF | include 101.0.0.[34]
*> 101.0.0.3/32      0.0.0.0          0          32768 i
*>i 101.0.0.4/32      214.0.0.4         0          100      0 ?

R4#show bgp vpnv4 unicast vrf OSPF | include 101.0.0.[34]
* i 101.0.0.3/32      214.0.0.3         0          100      0 i
*> 101.0.0.4/32      0.0.0.0          0          32768 ?

R3#ping vrf OSPF 101.0.0.4 source 101.0.0.3
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 101.0.0.4, timeout is 2 seconds:
Packet sent with a source address of 101.0.0.3
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 7/8/10 ms
```

We quickly perform the same verification for the IPv6 sham-link endpoints. Connectivity is achieved via 6VPE, the technique we have been using for ordinary IPv6 L3VPN connectivity.

```
R3#show bgp vpnv6 unicast vrf OSPF | include ::101:0:0:[34]
*> ::101:0:0:3/128   ::              0          32768 i
*>i ::101:0:0:4/128   ::FFFF:214.0.0.4

R4#show bgp vpnv6 unicast vrf OSPF | include ::101:0:0:[34]
* i ::101:0:0:3/128   ::FFFF:214.0.0.3
*> ::101:0:0:4/128   ::              0          32768 ?

R3#ping vrf OSPF ::101:0:0:4 source ::101:0:0:3
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to ::101:0:0:4, timeout is 2 seconds:
Packet sent with a source address of ::101:0:0:3%OSPF
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/2/3 ms
```

Although technically not required, we should ensure that these new loopbacks do not get advertised into the customer OSPF process. Because OSPF is redistributing from BGP without a filter at present, the OSPF process will contain these as external routes. We can manually filter them out on both CSR3 and CSR4. There is no reason to expose these addresses to the customer.

```
! CSR3 and CSR4
ip prefix-list PL_SHAM_ENDPOINTS seq 10 permit 101.0.0.0/8 ge 32
```

```

route-map RM_BGP_TO_OSPF_V4 deny 10
  match ip address prefix-list PL_SHAM_ENDPOINTS
route-map RM_BGP_TO_OSPF_V4 permit 1000

ipv6 prefix-list PL_SHAM_ENDPOINTS seq 5 permit ::101:0:0:0/80 ge 128
route-map RM_BGP_TO_OSPF_V6 deny 10
  match ipv6 address prefix-list PL_SHAM_ENDPOINTS
route-map RM_BGP_TO_OSPF_V6 permit 1000

router ospf 101
  redistribute bgp 214 subnets route-map RM_BGP_TO_OSPF_V4

router ospfv3 101
  address-family ipv6 unicast vrf OSPF
  redistribute bgp 214 route-map RM_BGP_TO_OSPF_V6

```

We can verify this was successful by checking the local LSDBs for any external routes beginning with 101.0.0.X or ::101:0:0:X. We don't see any. The reason we see IPv6 externals is due to the XR behavior earlier about failing to add the domain-ID to connected redistributed routes. The point is that the sham-link endpoints do not exist as external OSPFv2/v3 routes, which means the customer cannot learn them.

```
R3#show ip ospf 101 database | begin Type-5
```

```
[no output]
```

```
R3#show ospfv3 vrf OSPF database | begin External
```

```

Type-5 AS External Link States
ADV Router    Age      Seq#      Prefix
192.3.101.3   1457    0x80000005 FD00:192:11:101::/64
192.3.101.3   1457    0x80000003 FD00:192:12:101::/64
192.4.101.4   1511    0x80000005 FD00:192:11:101::/64
192.4.101.4   1511    0x80000005 FD00:192:12:101::/64

```

```
R4#show ip ospf 101 database | begin Type-5
```

```
[no output]
```

```
R4#show ospfv3 vrf OSPF database | begin External
```

```

Type-5 AS External Link States
ADV Router    Age      Seq#      Prefix
192.3.101.3   1682    0x80000005 FD00:192:11:101::/64
192.3.101.3   1682    0x80000003 FD00:192:12:101::/64
192.4.101.4   1730    0x80000005 FD00:192:11:101::/64
192.4.101.4   1730    0x80000005 FD00:192:12:101::/64

```

Next, we will configure the sham-link. We can optionally specify the OSPF cost as well, and provided this cost (plus the cost of the PE-CE transit links) is less than the backdoor cost, we can prefer MPLS over the backdoor. We will use a cost of 214 on the OSPFv2 sham-link without authentication. On the OSPFv3 sham-link, we use the default cost of 1, but enable authentication for variety. Older version of IOS do not allow for sham-link authentication, so we test that in XE for OSPFv3 only. For brevity, only CSR3 is

shown, but the CSR4 configuration is identical with reversed IPv4/v6 addresses for the sham-link endpoints.

```
! CSR3
key chain KC_SHAM_AUTH
  key 1
    key-string SHAM_AUTH
    cryptographic-algorithm hmac-sha-512

router ospf 101 vrf OSPF
  area 0 sham-link 101.0.0.3 101.0.0.4 cost 214

router ospfv3 101
  address-family ipv6 unicast vrf OSPF
    area 0 sham-link ::101:0:0:4 ::101:0:0:3 authentication key-chain
    KC_SHAM_AUTH
```

You cannot use MD5 authentication with OSPFv3 sham-links and must use one of the SHA options.
Failure to specify a valid option inside the key chain will results with some error messages.

```
! CSR3
%OSPFv3-5-NOCRYPTOALG: Key ID 1 in key chain KC_SHAM_AUTH does not have a
valid cryptographic algorithm
%OSPFv3-4-NOVALIDKEY: No valid authentication key under key-chain
KC_SHAM_AUTH
```

Assuming the configuration is correct, the routers will generate syslog messages to indicate that the sham-links are up. In case we miss these messages, we can check the status of the sham-links with a special show command. The pertinent information includes the fact that it is a demand-circuit with suppressed hellos. The DNA bit is allowed which implies the link is DC-capable, which is an LSA option in the LSA1. The link is considered point-to-point, which is a link type within the LSA1 as well. This is different than the virtual-link, which has its own link type in the LSA1. The sham-link is graphically equivalent to an ordinary point-to-point link. That is why we can explicitly specify a sham-list cost, but we cannot explicitly specify a virtual-link cost.

```
! CSR3
%OSPF-5-ADJCHG: Process 101, Nbr 192.4.101.4 on OSPF_SL0 from LOADING to
FULL, Loading Done
%OSPFv3-5-ADJCHG: Process 101, IPv6, VRF OSPF, Nbr 192.4.101.4 on OSPFv3_SL0
from LOADING to FULL, Loading Done

R3#show ip ospf 101 sham-links
Sham Link OSPF_SL0 to address 101.0.0.4 is up
Area 0 source address 101.0.0.3
  Run as demand circuit
  DoNotAge LSA allowed. Cost of using 214 State POINT_TO_POINT,
```

```

Timer intervals configured, Hello 10, Dead 40, Wait 40,
Hello due in 00:00:07
Adjacency State FULL (Hello suppressed)
Index 2/2, retransmission queue length 0, number of retransmission 0
First 0x0(0)/0x0(0) Next 0x0(0)/0x0(0)
Last retransmission scan length is 0, maximum is 0
Last retransmission scan time is 0 msec, maximum is 0 msec

```

```

R3#show ospfv3 vrf OSPF sham-links
      OSPFv3 101 address-family ipv6 vrf OSPF (router-id 192.3.101.3)
Sham Link OSPFv3_SL0 to address ::101:0:0:4 is up
Interface ID 25
Area 0 source address ::101:0:0:3
Run as demand circuit
DoNotAge LSA allowed.
Cost of using 1
Transmit Delay is 1 sec, State POINT_TO_POINT,
Timer intervals configured, Hello 10, Dead 40, Wait 40, Retransmit 5
Adjacency State FULL (Hello suppressed)
Index 1/2/2, retransmission queue length 0, number of retransmission 0
First 0x0(0)/0x0(0)/0x0(0) Next 0x0(0)/0x0(0)/0x0(0)
Last retransmission scan length is 0, maximum is 0
Last retransmission scan time is 0 msec, maximum is 0 msec

```

To verify the topological significance of the sham-link, we will inspect the LSA1 from the perspective of CSR3. Most of the output is removed, but we can see a new link inside the LSA1. It uses a link address of 0.0.0.24, which is a new software IDB generated by the router specifically for the sham-link. XE allows us to see this new interface number for confirmation as well. The link is point-to-point and has a cost of 214 as expected. If we didn't know a sham-link was being used, this LSA1 output is indistinguishable from an ordinary OSPF P2P link using IP unnumbered.

```

R3#show ip ospf 101 database router self-originate
      OSPF Router with ID (192.3.101.3) (Process ID 101)
      Router Link States (Area 0)
LS age: 732
Options: (No TOS-capability, DC)
[snip]

Link connected to: another Router (point-to-point)
(Link ID) Neighboring Router ID: 192.4.101.4
(Link Data) Router Interface address: 0.0.0.24
Number of MTID metrics: 0
TOS 0 Metrics: 214
[snip]

```

```

R3#show ifnum list | include 24
SWIDB      24      ACTIVE      SL0      08E00000 00000000 00000000
00000000 00000000 00000000 00000000 00000000

```

For IPv6, the output is similar. The router is DC-capable, and has a link to CSR4 using interface ID 25. The IDB process confirms this is used for a sham-link as well. The link is point-to-point with the default cost of 1.

```
R3#show ospfv3 vrf OSPF database router self-originate
      OSPFv3 101 address-family ipv6 vrf OSPF (router-id 192.3.101.3)
        Router Link States (Area 0)
LS age: 674
Options: (V6-Bit, E-Bit, R-Bit, DC-Bit)
[snip]
```

```
Link connected to: another Router (point-to-point)
Link Metric: 1
Local Interface ID: 25
Neighbor Interface ID: 25
Neighbor Router ID: 192.4.101.4
```

```
R3#show ifnum list | include 25
SWIDB      25      ACTIVE  SL0      08EA0000 00000000 00000000
00000000 00000000 00000000 00000000 00000000
```

Because the CEs share area 0 with the PEs, they can see this new link in the OSPF graph. Like the MPLS-TE forwarding-adjacency, they use this in their SPF calculations. CSR6 now routes over MPLS to reach XRv4's loopback. The total OSPF path cost is 217. A cost of 1 represents the local PE-CE hop, 214 for the sham-link, 1 more for the remote PE-CE hop, and 1 more for the loopback itself. Of greatest significance is the route-type, which is now intra-area. The routes that were once inter-area, as defined by being in the same domain, are now bridged into the same area by the sham-link. A virtual-link connects a disjoint area 0 over a non-backbone OSPF area while a sham-link connects a disjoint area 0 over MPLS. The IPv6 route is similar, but has a metric of 3 to account for the two PE-CE links and the sham-link default cost of 1. The cost of the IPv6 loopback is 0 compared to the IPv4 loopback cost of 1.

```
R6#show ip route vrf OSPF 192.101.14.14
Routing Table: OSPF
Routing entry for 192.101.14.14/32
  Known via "ospf 101", distance 110, metric 217, type intra area
  Last update from 192.4.101.4 on GigabitEthernet2.101, 00:17:57 ago
Routing Descriptor Blocks:
  * 192.4.101.4, from 0.0.101.14, 00:17:57 ago, via GigabitEthernet2.101
    Route metric is 217, traffic share count is 1
```

```
R6#show ipv6 route vrf OSPF ::192:101:14:14
Routing entry for ::192:101:14:14/128
  Known via "ospf 101", distance 110, metric 3, type intra area
  Route count is 1/1, share count 0
Routing paths:
```

FE80::4, GigabitEthernet2.101

Last updated 00:15:46 ago

A quick traceroute from CSR6 proves that the sham-link is working correctly. Traffic is being routed towards the L3VPN PEs and encapsulated inside MPLS.

```
R6#traceroute vrf OSPF 192.101.14.14 source 192.101.6.6
Type escape sequence to abort.
Tracing the route to 192.101.14.14
VRF info: (vrf in name/id, vrf out name/id)
 1 192.4.101.4 4 msec 4 msec 4 msec
 2 192.3.101.3 [MPLS: Label 3015 Exp 0] 4 msec 4 msec 4 msec
 3 192.3.101.14 5 msec 13 msec 13 msec
```

```
R6#traceroute vrf OSPF ipv6
Target IPv6 address: ::192:101:14:14
Source address: ::192:101:6:6
[snip]
Type escape sequence to abort.
Tracing the route to ::192:101:14:14

 1 FD00:192:4:101::4 4 msec 4 msec 3 msec
 2 FD00:192:3:101::3 [MPLS: Label 3016 Exp 0] 4 msec 4 msec 4 msec
 3 ::192:101:14:14 212 msec 5 msec 5 msec
```

To verify that there is no disruption to the customer network, we will verify that the sham-link endpoints did not leak into the customer RIB. This should be impossible given the filtering we configured earlier, but we will ensure the routes do not exist. This makes the sham-link entirely transparent to the customer CE devices.

```
R6#show ip route vrf OSPF 101.0.0.0 255.0.0.0 longer-prefixes | begin Gateway
Gateway of last resort is not set
[no output]
```

```
RP/0/0/CPU0:XRv4#show route vrf OSPF ipv4 | include 101.0.0
[no output]
```

```
R6#show ipv6 route vrf OSPF ::101:0:0:0/80 longer-prefixes | begin Applica
a - Application
[no output]
```

```
RP/0/0/CPU0:XRv4#show route vrf OSPF ipv6 | include ::101
[no output]
```

The same exact problem presents itself on the other side of the network with XRv1 and XRv2 as PEs. XRv3 prefers the high-cost backdoor link because the route to CSR5's loopback is intra-area via that path.

```

RP/0/0/CPU0:XRv3#show route vrf OSPF 192.101.5.5
Routing entry for 192.101.5.5/32
  Known via "ospf 101", distance 110, metric 501, type intra area
  Routing Descriptor Blocks
    192.5.101.5, from 192.12.101.6, via GigabitEthernet0/0/0/0.201
      Route metric is 501
  No advertising protos.

```

The concepts for sham-links in XR are exactly the same and the configuration is similar. First, we will create the loopbacks on XRv1 and XRv2 inside of the customer VPN. Then, we advertise them into BGP using a network statement or redistribution. XR does not support the “match interface” or anything equivalent within RPL, so we can minimize RPL configuration using parameters.

```

! XRv1
interface Loopback101
  description SHAM LINK ENDPOINT
  vrf OSPF
  ipv4 address 101.0.0.11 255.255.255.255
  ipv6 address ::101:0:0:11/128

router bgp 214
  vrf OSPF
    address-family ipv4 unicast
      network 101.0.0.11/32
    address-family ipv6 unicast
      network ::101:0:0:11/128

! XRv2
interface Loopback101
  description SHAM LINK ENDPOINT
  vrf OSPF
  ipv4 address 101.0.0.12 255.255.255.255
  ipv6 address ::101:0:0:12/128

prefix-set PS_SHAM_V4
  101.0.0.0/8 ge 32
end-set

prefix-set PS_SHAM_V6
  ::101:0:0:0/80 ge 128
end-set

route-policy RPL_SHAM_TO_BGP($PS)
  if destination in $PS then
    pass
  endif
end-policy

```

```

router bgp 214
vrf OSPF
address-family ipv4 unicast
redistribute connected route-policy RPL_SHAM_TO_BGP(PS_SHAM_V4)

address-family ipv6 unicast
redistribute connected route-policy RPL_SHAM_TO_BGP(PS_SHAM_V6)

```

We will verify that these routes were successfully advertised into BGP on both PEs. Notice that the remote sham-link endpoints from CSR3 and CSR4 are also learned; they are still inside the iBGP advertisement space. This allows us to build sham-links to CSR3 and CSR4 if we desire, but since there are no east-to-west backdoor links, there is little reason to do so. We can see all 4 routes from all 4 PEs within IPv4 and IPv6 scopes.

```

RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast vrf OSPF | include 101.0.0
*>i101.0.0.3/32      214.0.0.3      0      100      0 i
*>i101.0.0.4/32      214.0.0.4      0      100      0 ?
*> 101.0.0.11/32     0.0.0.0        0          32768 i
* i101.0.0.12/32     214.0.0.12     0      100      0 ?

RP/0/0/CPU0:XRv1#show bgp vpnv6 unicast vrf OSPF | include ::101:0:0
*>i::101:0:0:3/128   214.0.0.3      0      100      0 i
*>i::101:0:0:4/128   214.0.0.4      0      100      0 ?
*> ::101:0:0:11/128  ::             0          32768 i
* i::101:0:0:12/128  214.0.0.12     0      100      0 ?

```

We verify reachability between these XRv1/XRv2 sham-link endpoints, again relying on basic MPLS L3VPN (6VPE) connectivity. This is exactly the same procedure we followed for the XE sham-link.

```

RP/0/0/CPU0:XRv1#ping vrf OSPF 101.0.0.12 source 101.0.0.11
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 101.0.0.12, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/20/89 ms

RP/0/0/CPU0:XRv1#ping vrf OSPF ::101:0:0:12 source ::101:0:0:11
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to ::101:0:0:12, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

```

Like last time, we should remove these sham-link endpoints from OSPF. Notice that XRv2 is originating XRv1's sham-link endpoint and vice versa; they are learning one another's BGP routes then redistributing them back into OSPF. Fortunately, the DN-bit protects the PEs from installing these

routes, but it is still bad design and may be confusing for customers, who will install these routes. XRv1, for example, has these new loopbacks inside of OSPFv3.

```
RP/0/0/CPU0:XRv1#show ospfv3 vrf OSPF database | include ::101:0:0:1[12]
0.0.101.11      141      0x80000001  ::101:0:0:12/128
214.0.0.12      81       0x80000001  ::101:0:0:11/128
```

To correct it, we can use RPL to filter any sham-link endpoints when redistributing from BGP to OSPF. The prefix-lists we used above on XRv2 can be re-used for this task again and are not shown again. They must be copied to XRv1 since only XRv2 has them configured at present (XRv1 used network statements instead). Using a new RPL policy along with parameterization, we instruct OSPF to filter any sham-link endpoints from entering from BGP. The filter also accounts for CSR3/CSR4 sham-link endpoints as well. For this reason, when there are many PEs in an OSPF-serving L3VPN, the sham-link endpoints should aggregate nicely to simplify this filtering task.

```
! XRv1 and XRv2
route-policy RPL_BGP_TO_OSPF($SHAMS)
  if destination in $SHAMS then
    drop
  else
    pass
  endif
end-policy

router ospf 101
  vrf OSPF
  redistribute bgp 214 route-policy RPL_BGP_TO_OSPF(PS_SHAM_V4)

router ospfv3 101
  vrf OSPF
  redistribute bgp 214 route-policy RPL_BGP_TO_OSPF(PS_SHAM_V6)
```

To verify it, we can check XRv1 for any 101.0.0.X or ::101:0:0:X routes. Now, we don't see any in the PE LSDBs or in the customer routing tables.

```
RP/0/0/CPU0:XRv1#show ospfv3 vrf OSPF database | include ::101:0:0:
[no output]
```

```
RP/0/0/CPU0:XRv1#show ospf vrf OSPF database | include 101.0.0
[no output]
```

```
RP/0/0/CPU0:XRv3#show route vrf OSPF ipv4 | include 101.0.0
[no output]
```

```
RP/0/0/CPU0:XRv3#show route vrf OSPF ipv6 | include ::101
[no output]
```

Next, we will configure the sham-link. The sham-link has its own stanza in XR, which is convenient because it has many sub-options like authentication and cost. We will set a custom cost and enable basic MD5 authentication on the OSPFv2 sham-link. The OSPFv3 sham-link will be more secure with IPSec encryption using DES and SHA-1 hash. For brevity, only XRv1's configuration is shown since the only difference on XRv2 is reversing the IPv4/v6 addresses.

```
! XRv1
router ospf 101
  vrf OSPF
  area 0
    sham-link 101.0.0.11 101.0.0.12
      cost 12
      authentication message-digest
      message-digest-key 1 md5 SHAM_AUTH

router ospfv3 101
  vrf OSPF
  area 0
    sham-link ::101:0:0:11 ::101:0:0:12
      encryption ipsec spi 1212 esp des clear 1234567890abcdef authentication
      sha1 clear 12345678901234567890123456789012345678901234567890
```

Like any other OSPFv2/v3 neighbor, we will see a log message for the sham-links, along with the numbers allocated to them. Unlike XE, XR allocates different numbers for different sham-links between processes. XE assigned both SLs a value of 0 on CSR3 and CSR4 because the OSPF processes are entirely different.

```
! XRv1
ospfv3[1028]: %ROUTING-OSPFv3-5-ADJCHG : Process 101, Nbr 214.0.0.12 on
OSPF_SL1 from LOADING to FULL, Loading Done
ospf[1018]: %ROUTING-OSPF-5-ADJCHG : Process 101, Nbr 214.0.0.12 on OSPF_SL0
in area 0 from LOADING to FULL, Loading Done, vrf OSPF vrfid 0x60000007
```

In case you miss the log messages, the XR show commands give the same details as the XE show commands. We verify that the link is a demand-circuit, point-to-point, suppressing hellos, has the proper cost/authentication, and is operational.

```
RP/0/0/CPU0:XRv1#show ospf vrf OSPF sham-links
Sham Links for OSPF 101, VRF OSPF
Sham Link OSPF_SL0 to address 101.0.0.12 is up
Area 0, source address 101.0.0.11
IfIndex = 2
  Run as demand circuit
  DoNotAge LSA allowed., Cost of using 12
  Transmit Delay is 1 sec, State POINT_TO_POINT,
  Timer intervals configured, Hello 10, Dead 40, Wait 40, Retransmit 5
```

```

Hello due in 00:00:07:199
Adjacency State FULL (Hello suppressed)
Number of DBD retrans during last exchange 0
Index 2/2, retransmission queue length 0, number of retransmission 0
First 0(0)/0(0) Next 0(0)/0(0)
Last retransmission scan length is 0, maximum is 0
Last retransmission scan time is 0 msec, maximum is 0 msec
Message digest authentication enabled
Youngest key id is 1

RP/0/0/CPU0:XRv1#show ospfv3 vrf OSPF sham-links
Sham Links for OSPFv3 101, VRF OSPF
Sham Link OSPF_SL1 to address ::101:0:0:12 is up, ipsec is up
Area 0, source address ::101:0:0:11
IfIndex = 2
ESP Encryption DES, Authentication SHA1, SPI 1212
Run as demand circuit
DoNotAge LSA allowed., Cost of using 1
Transmit Delay is 1 sec, State POINT_TO_POINT,
Timer intervals configured, Hello 10, Dead 40, Wait 40, Retransmit 5
Hello due in 00:00:02
Adjacency State FULL (Hello suppressed)
Number of DBD retrans during last exchange 0
Index 2/2, retransmission queue length 0, number of retransmission 0
First 0(0)/0(0) Next 0(0)/0(0)
Last retransmission scan length is 0, maximum is 0
Last retransmission scan time is 0 msec, maximum is 0 msec

```

Next, we will confirm that the link exists within the router LSA on XRv2 for both OSPFv2 and OSPFv3. We can see that the OSPFv2 link has a cost of 12, is point-to-point, is DC-capable, and connects directly to XRv1. The OSPFv3 link has all the same attributes except uses a cost of 1, the default. Just like ordinary OSPF P2P interfaces, the costs need not be symmetric, but for simplicity, I configure them so.

```

RP/0/0/CPU0:XRv2#show ospf vrf OSPF database router self-originate
    OSPF Router with ID (214.0.0.12) (Process ID 101, VRF OSPF)
        Router Link States (Area 0)
LS age: 490
Options: (No TOS-capability, DC)
[snip]
Link connected to: another Router (point-to-point)
(Link ID) Neighboring Router ID: 0.0.101.11
(Link Data) Router Interface address: 0.0.0.2
Number of TOS metrics: 0
TOS 0 Metrics: 12

RP/0/0/CPU0:XRv2#show ospfv3 vrf OSPF database router self-originate
    OSPFv3 Router with ID (214.0.0.12) (Process ID 101 VRF OSPF)
        Router Link States (Area 0)

```

```

LS age: 554
Options: (V6-Bit E-Bit R-Bit DC-Bit)
[snip]
Link connected to: another Router (point-to-point)
Link Metric: 1
Local Interface ID: 2
Neighbor Interface ID: 2
Neighbor Router ID: 0.0.101.11

```

Because XRv3 and CSR5 can use this sham-link in their SPF calculations, the MPLS network is considered intra-area and its cost is can be directly compared to the slow backdoor. XRv3 now routes to CSR5's loopback via MPLS. The total cost is 15: 1 for the local PE-CE link, 12 for the sham-link, 1 for the remote PE-CE link, and 1 for the loopback. The type is intra-area as expected. The output is similar for IPv6 with a different metric based on the IPv6 topology and sham-link cost.

```

RP/0/0/CPU0:XRv3#show route vrf OSPF 192.101.5.5
Routing entry for 192.101.5.5/32
  Known via "ospf 101", distance 110, metric 15, type intra area
Routing Descriptor Blocks
  192.11.101.11, from 192.12.101.6, via GigabitEthernet0/0/0/0.101
    Route metric is 15
  No advertising protos.

```

```

RP/0/0/CPU0:XRv3#show route vrf OSPF ipv6 ::192:101:5:5
Routing entry for ::192:101:5:5/128
  Known via "ospf 101", distance 110, metric 3, type intra area
Routing Descriptor Blocks
  fe80::11, from ::, via GigabitEthernet0/0/0/0.101
    Route metric is 3
  No advertising protos.

```

Traceroute confirms the proper traffic pattern. XR does not show the labels for IPv6, possibly due to a cosmetic adjustment, but we can see the 214.X.X.X address which represents the MPLS core. There is no possibility of this IPv6 traceroute working without MPLS encapsulation, so we can safely assume the traffic was labeled.

```

RP/0/0/CPU0:XRv3#traceroute vrf OSPF 192.101.5.5 source 192.101.13.13
Type escape sequence to abort.
Tracing the route to 192.101.5.5
 1  192.11.101.11 0 msec  0 msec  0 msec
 2  214.11.12.12 [MPLS: Label 92011 Exp 0] 0 msec  0 msec  0 msec
 3  192.12.101.5 0 msec  0 msec  0 msec

```

```

RP/0/0/CPU0:XRv3#traceroute vrf OSPF ::192:101:5:5 source ::192:101:13:13
Type escape sequence to abort.
Tracing the route to ::192:101:5:5
 1  fd00:192:11:101::11 9 msec 19 msec 9 msec

```

```
2 2001:214:11:12::12 0 msec 0 msec 0 msec
3 fd00:192:12:101::5 0 msec 0 msec 0 msec
```

For good measure, we will use TCL to verify reachability from CSR5 and CSR6 to all endpoints for both AFIs. XR technically supports TCL, but is very complicated and not mentioned in this chapter. We will therefore limit the TCL verifications to XE CE routers. The outputs from CSR5 and CSR6 are not shown, but the pings were 100% successful to all destinations. Notice the OSPF “external” test prefixes are included here also.

```
! CSR5 and CSR6
tclsh
foreach x {
192.101.5.5
192.101.6.6
192.101.13.13
192.101.14.14
192.60.6.6
::192:101:5:5
::192:101:6:6
::192:101:13:13
::192:101:14:14
::192:60:6:6
} { ping vrf OSPF $x source loopback101 repeat 3 timeout 1 }
```

38.1.3 EIGRP and Site-of-Origin (SoO)

EIGRP is another option for PE-CE routing with L3VPN. The VRF-aware EIGRP configuration is almost identical for IPv4 and IPv6, assuming we used named-mode on the XE platforms. There is no “DN-bit” concept in EIGRP, so the VRF-lite configuration is not much different from the PE, with the exception of BGP redistribution. For EIGRP IPv6, there is no “network” statement, so all IPv6-enabled interfaces are automatically configured for EIGRP. They can selectively disabled by shutting them down under the EIGRP process as needed, but in our case, we don’t need to disable EIGRP IPv6 from any interfaces. The configurations for CSR5 and XRv3 are shown for brevity. The backdoor link is Gig2.202; it exists between XRv4/CSR6 and XRv3/CSR5 and is currently shut down. Note that XR does not log EIGRP neighbor changes by default, so I personally enable this feature at all times to assist with troubleshooting.

```
! CSR5
router eigrp VPN
 address-family ipv4 unicast vrf EIGRP autonomous-system 102
   network 192.0.0.0 0.255.255.255

 address-family ipv6 unicast vrf EIGRP autonomous-system 102

! XRv3
router eigrp VPN
 vrf EIGRP
 address-family ipv4
```

```

router-id 13.13.13.13
log-neighbor-changes
autonomous-system 102
interface Loopback102
  passive-interface
interface GigabitEthernet0/0/0/0.102
interface GigabitEthernet0/0/0/0.202

address-family ipv6
router-id 13.13.13.13
log-neighbor-changes
autonomous-system 102
interface Loopback102
  passive-interface
interface GigabitEthernet0/0/0/0.102
interface GigabitEthernet0/0/0/0.202

```

We quickly verify the neighbors are up on all EIGRP PEs for both AFIs. We only expect to see one neighbor on each PE, which is the corresponding CE.

```

R3#show eigrp address-family ipv4 vrf EIGRP neighbors | begin ^0
0  192.3.102.14          Gi2.102          11 1d20h          29   174   0   67

```

```

R3#show eigrp address-family ipv6 vrf EIGRP neighbors | begin ^0
0  Link-local address:   Gi2.102          12 1d09h          27   162   0   64
   FE80::14

```

```

R4#show eigrp address-family ipv4 vrf EIGRP neighbors | begin ^0
0  192.4.102.6          Gi2.102          13 2d01h           1    100   0   57

```

```

R4#show eigrp address-family ipv6 vrf EIGRP neighbors | begin ^0
0  Link-local address:   Gi2.102          11 2d01h           1    100   0   62
   FE80::6

```

```

RP/0/0/CPU0:XRv1#show eigrp vrf EIGRP ipv4 neighbors | begin ^0
0  192.11.102.13        Gi0/0/0/0.102    11   1d20h    15    200   0   60

```

```

RP/0/0/CPU0:XRv1#show eigrp vrf EIGRP ipv6 neighbors | begin ^0
0  Link Local Address:   Gi0/0/0/0.102    13 11:20:03    15    200   0   82
   fe80::13

```

```

RP/0/0/CPU0:XRv2#show eigrp vrf EIGRP ipv4 neighbors | begin ^0
0  192.12.102.5         Gi0/0/0/0.102    11 06:16:52     1    200   0   78

```

```

RP/0/0/CPU0:XRv2#show eigrp vrf EIGRP ipv6 neighbors | begin ^0
0  Link Local Address:   Gi0/0/0/0.102    11 05:40:54     1    200   0   90
   fe80::5

```

We verify that CSR4 and XRv2 learn some EIGRP routes from CSR6 and CSR5 in both IPv4 and IPv6 AFIs. In this case, there is a single loopback from each CE, just like there was for OSPF. This confirms the CE-to-PE route exchange.

```
R4#show ip route vrf EIGRP eigrp | begin Gateway
Gateway of last resort is not set
    192.102.6.0/32 is subnetted, 1 subnets
D    192.102.6.6 [90/10880] via 192.4.102.6, 1d21h, GigabitEthernet2.102
```

```
R4#show ipv6 route vrf EIGRP eigrp | begin Applic
a - Application
D    ::192:102:6:6/128 [90/10880]
    via FE80::6, GigabitEthernet2.102
```

```
RP/0/0/CPU0:XRv2#show route vrf EIGRP ipv4 eigrp
D    192.102.5.5/32 [90/10880] via 192.12.102.5, 06:22:27,
GigabitEthernet0/0/0/0.102
```

```
RP/0/0/CPU0:XRv2#show route vrf EIGRP ipv6 eigrp
D    ::192:102:5:5/128
    [90/10880] via fe80::5, 05:46:31, GigabitEthernet0/0/0/0.102
```

These EIGRP routes are redistributed from EIGRP into BGP, and vice versa, on all four PEs. The configurations on CSR4 and XRv2 are shown; they are identical to the VRF-aware CE configurations with the addition of redistribution discussed earlier.

```
! CSR4
router eigrp VPN
 address-family ipv4 unicast vrf EIGRP autonomous-system 102
   topology base
   redistribute bgp 214

 address-family ipv6 unicast vrf EIGRP autonomous-system 102
   topology base
   redistribute bgp 214

! XRv2
router eigrp VPN
 vrf EIGRP
   address-family ipv4
     redistribute bgp 214

   address-family ipv6
     redistribute bgp 214
```

When we verify the routes inside of VPNv4/v6, we see that there are several new extended communities attached to them. The cost-community has a dedicated section and is not discussed in

detail, but notice that the point of insertion (POI) is pre-bestpath, which is value 128. XE decodes this for us but XR does not. The second 128 represents the community-ID which allows cost-communities to be compared directly, not to be confused with the POI value of 128 meant to indicate pre-bestpath evaluation. When used in this fashion, the cost-community always uses POI and community ID values of 128; this cannot be modified. The metric value is easy to read and is directly derived from the RIB metric of 10880.

```
R4#show ip route vrf EIGRP 192.102.6.6 | include metric
  Known via "eigrp 102", distance 90, metric 10880, type internal
    Route metric is 10880, traffic share count is 1
```

```
RP/0/0/CPU0:XRv2#show route vrf EIGRP ipv4 192.102.5.5/32 | include metric
  Known via "eigrp 102", distance 90, metric 10880, type internal
    Route metric is 10880
```

The other extended-community values are EIGRP-specific and are encoded according to the table below.

Attribute	Type	Usage	Value
General	0x8800	Route information	Route flag and tag
Metric	0x8801	Route metric and AS	ASN and delay
	0x8802	Route metric	Reliability, next-hop, bandwidth
	0x8803	Route metric	Reserve, Load, and MTU
External	0x8804	External route metric	Remote AS and Remote ID
	0x8805	External route metric	Remote Protocol and Remote Metric
Originating RID	0x8806	Originating RID	Originating RID

```
R4#show bgp vpnv4 unicast vrf EIGRP 192.102.6.6/32
[snip]
  Local
    192.4.102.6 (via vrf EIGRP) from 0.0.0.0 (214.0.0.4)
      Origin incomplete, metric 10880, localpref 100, weight 32768, valid,
sourced, best
      Extended Community: RT:214:102 Cost:pre-bestpath:128:10880
        0x8800:32768:0 0x8801:102:288 0x8802:65281:2560 0x8803:65281:1500
        0x8806:0:3221513734
      mpls labels in/out 4012/nolabel
      rx pathid: 0, tx pathid: 0x0
```

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf EIGRP 192.102.5.5/32
[snip]
  Local
    192.12.102.5 from 0.0.0.0 (214.0.0.12)
      Origin incomplete, metric 10880, localpref 100, weight 32768, valid,
redistributed, best, group-best, import-candidate
      Received Path ID 0, Local Path ID 1, version 639
      Extended community: COST:128:128:10880 EIGRP route-info:0x8000:0 EIGRP
AD:102:288 EIGRP RHB:255:1:2560 EIGRP LM:0x0:1:1500 EIGRP VRR:0x0:5.5.12.192
RT:214:102
```


There isn't much reason to deep-dive the extended communities, but we will verify the values for sanity once. XR makes it a little easier by naming the communities by their purpose. 0x8800 carries the "route-flag" which is 0x8000 (32768) and the route-tag (not set) on both routes. 0x8801 carries the AS number (102) and the route delay (288). We can check the EIGRP topology on both routers to confirm a value of 288, which is the delay in tens of microseconds scaled by 256 (11,250,000 picoseconds * 256 = 2,880,000,000 picoseconds = 2880 microseconds).

```
R4#show eigrp address-family ipv4 vrf EIGRP topology 192.102.6.6/32
EIGRP-IPv4 VR(VPN) Topology Entry for AS(102)/ID(192.4.102.4)
      Topology(base) TID(0) VRF(EIGRP)
EIGRP-IPv4(102): Topology base(0) entry for 192.102.6.6/32
      State is Passive, Query origin flag is 1, 1 Successor(s), FD is 1392640,
RIB is 10880
      Descriptor Blocks:
      192.4.102.6 (GigabitEthernet2.102), from 192.4.102.6, Send flag is 0x0
      Composite metric is (1392640/163840), route is Internal
      Vector metric:
      Minimum bandwidth is 1000000 Kbit
      Total delay is 11250000 picoseconds
      [snip]
```

Verifying 0x8802 is also somewhat tricky. The first 2 byte value contains both the reliability and next-hop flag together. 65281 translates to 0xFF01, which represents a reliability of 255 and a hop count of 1.

```
R4#show eigrp address-family ipv4 vrf EIGRP topology 192.102.6.6/32 | include
Reliab|Hop
      Reliability is 255/255
      Hop count is 1
```

Extended community 0x8802 carries the link bandwidth of 2560. This appears to be measured in hundreds of megabits scaled by 256 (1,000,000,000 bps * 256 = 256,000,000,000 bps, = 256 Gbps).

```
R4#show eigrp address-family ipv4 vrf EIGRP topology 192.102.6.6/32 | include
bandw
      Minimum bandwidth is 1000000 Kbit
```

The extended community 0x8803 has a decimal value of 65281 which we know evaluates to 0xFF01. The load represents the 1, and I assume the "Reserved" field is set to 0xFF. XR seems to set this to zero, though, so it is probably ignored. The route MTU is clearly shown as a decimal value of 1500, which is convenient.

```
R4#show eigrp address-family ipv4 vrf EIGRP topology 192.102.6.6/32 | include
MTU|Load
      Load is 1/255
```

Minimum MTU is 1500

The decimal number carried in 0x8806 by CSR4 is 0xC0046606 in hex. This translates to 192.4.102.6, while XRv2 shows CSR5's loopback in reverse order. This is the originating router's EIGRP RID. We can quickly confirm this by checking the topology information. Ideally both RIDs would be the loopback, but since they are not manually set, EIGRP selecting the first available interface at the time I configured it. For external routes, this is an important field for loop prevention.

```
R4#show eigrp address-family ipv4 vrf EIGRP topology 192.102.6.6/32 | include Origin
```

```
Originating router is 192.4.102.6
```

```
RP/0/0/CPU0:XRv2#show eigrp vrf EIGRP ipv4 topology 192.102.5.5/32 | include Origin
```

```
Originating router is 192.102.5.5
```

These extended communities are automatically and by default added to BGP routes when EIGRP is redistributed from within a VPN. The reason all this information is necessary is so that routes across the VPN, assuming the ASN matches, can be considered internal. The MPLS network essentially collapses to a single node in the EIGRP topology and can be used to seamlessly bridge EIGRP islands. This is somewhat like a sham-link except zero cost and in-band, whereas the sham-link was an explicit, out-of-band link with an associated OSPF cost (it was part of the graph). As is normal for any IGP to BGP redistribution, the IGP metric is also copied into the BGP MED. Checking XRv2, we can see the IGP metric inside the BGP MED, which is installed in the local routing table as an iBGP route.

```
RP/0/0/CPU0:XRv2#show route vrf EIGRP ipv4 192.102.6.6/32 | include metric
```

```
Known via "bgp 214", distance 200, metric 10880, type internal
```

```
Route metric is 10880
```

Checking the EIGRP topology on CSR5, we can see all of the original routing information from CSR6's loopback is preserved across the L3VPN. The hop count is only two, since the MPLS network (including the PEs) only counts as 1 hop. The originating router is still the same, and the reported distance (RD) of 1392640 is the FD of the remote PE (CSR4). This proves that the MPLS network is viewed as a single hop; because it's one hop, it looks like one router, and thus has no cost. The PE-CE link costs are accounted for, however. For OSPF, the sham-link itself incurred a cost of at least 1.

```
R5#show eigrp address-family ipv4 vrf EIGRP topology 192.102.6.6/32
```

```
EIGRP-IPv4 VR(VPN) Topology Entry for AS(102)/ID(192.102.5.5)
```

```
Topology(base) TID(0) VRF(EIGRP)
```

```
EIGRP-IPv4(102): Topology base(0) entry for 192.102.6.6/32
```

```
State is Passive, Query origin flag is 1, 1 Successor(s), FD is 2048000, RIB is 16000
```

```
Descriptor Blocks:
```

```
192.12.102.12 (GigabitEthernet2.102), from 192.12.102.12, Send flag is 0x0
```

```
Composite metric is (2048000/1392640), route is Internal
```

```

Vector metric:
  Minimum bandwidth is 1000000 Kbit
  Total delay is 21250000 picoseconds
  Reliability is 255/255
  Load is 1/255
  Minimum MTU is 1500
  Hop count is 2
  Originating router is 192.4.102.6

```

```

R4#show eigrp address-family ipv4 vrf EIGRP topology 192.102.6.6/32 | include
Composite_metric
  Composite metric is (1392640/163840), route is Internal

```

We can disable the extended community population on CSR4 by telling EIGRP not to pass this information to BGP on a per AFI basis. Notice that the normal BGP redistribution actions still occur: the IGP metric is copied to the BGP MED, the export route-target is imposed, and an MPLS label is allocated. The VPN can still work, but the remote PEs will not be able to assume this route is part of the same EIGRP AS as its local customer.

```

! CSR4
router eigrp VPN
  address-family ipv4 unicast vrf EIGRP autonomous-system 102
  no populate bgp-ext-comm

R4#show bgp vpnv4 unicast vrf EIGRP 192.102.6.6/32 | begin Local
Local
  192.4.102.6 (via vrf EIGRP) from 0.0.0.0 (214.0.0.4)
  Origin incomplete, metric 10880, localpref 100, weight 32768, valid,
sourced, best
  Extended Community: RT:214:102
  mpls labels in/out 4035/nolabel
  rx pathid: 0, tx pathid: 0x0

```

As soon as this happens, XRv2 prints out an angry syslog message. The message warns us that during the BGP to EIGRP redistribution, we did not set a metric. Normally this is required when redistributing into EIGRP (except when using connected, static, or other EIGRP routes). We didn't set a metric earlier because the extended-communities handled that for us. Now, without any extended-community information, EIGRP cannot accept this route since it has no idea what the seed metric should be.

```

! XRv2
eigrp[1002]: %ROUTING-EIGRP-3-NO_REDISTMETRIC : EIGRP-VPN: EIGRP-v4 102: No
metric set while redistributing route 192.102.6.6/32

```

```

RP/0/0/CPU0:XRv2#show eigrp vrf EIGRP ipv4 topology 192.102.6.6/32
% IPv4-EIGRP (VRF EIGRP): Route not in topology table of EIGRP VPN.

```

To solve it, we will set a bogus set of default metric inputs on XRv1 and XRv2 for EIGRP IPv4. The route is very much external now as the extended-community enhancements were removed, but at least EIGRP can see it. Last, we can check the routing on CSR5 to ensure it installs this route. The bogus metrics make the external routes stand out very clearly.

```
! XRv1 and XRv2
router eigrp VPN
 vrf EIGRP
  address-family ipv4
  default-metric 12345 543 234 123 1500
```

```
RP/0/0/CPU0:XRv2#show eigrp vrf EIGRP ipv4 topology 192.102.6.6/32
IPv4-EIGRP VR(VPN) AS(102) VRF EIGRP: Topology entry for 192.102.6.6/32
  State is Passive, Query origin flag is 1, 1 Successor(s), FD is 408947559,
  RIB is 3194902
```

Routing Descriptor Blocks:

```
214.0.0.4, from Redistributed, Send flag is 0x0
  Composite metric is (408947559/0), Route is External
  Vector metric:
    Minimum bandwidth is 12345 Kbit
    Total delay is 5430000000 picoseconds
    Reliability is 234/255
    Load is 123/255
    Minimum MTU is 1500
    Hop count is 0
  External data:
    Originating router is 214.0.0.12 (this system)
    AS number of route is 102
    External protocol is BGP, external metric is 10880
    Administrator tag is 0 (0x00000000)
```

```
R5#show ip route vrf EIGRP eigrp | section include EX
  D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
D EX    192.102.6.6
        [170/3200022] via 192.12.102.12, 00:03:13, GigabitEthernet2.102
```

One of the pitfalls of disabling extended-community support on CSR4 is that it has bidirectional implications. That is to say, not only will EIGRP not instruct BGP to carry the information, but it also will not honor information from BGP, either. On CSR4, it learns CSR5's loopback via iBGP and it has all of the extended-community values. However, the route does not exist in the EIGRP topology since, like XRv2, there is no default redistribution metric and EIGRP is now blind to these extended-communities.

```
R4#show bgp vpnv4 unicast vrf EIGRP 192.102.5.5/32 bestpath | begin Local
Local
214.0.0.12 (metric 20) (via default) from 214.0.0.12 (214.0.0.12)
  Origin incomplete, metric 10880, localpref 100, valid, internal, best
  Extended Community: RT:214:102 Cost:pre-bestpath:128:10880
```

```

0x8800:32768:0 0x8801:102:288 0x8802:65281:2560 0x8803:1:1500
0x8806:0:3227911429
mpls labels in/out nolabel/92012
rx pathid: 0, tx pathid: 0x0

```

```

R4#show eigrp address-family ipv4 vrf EIGRP topology 192.102.5.5/32
EIGRP-IPv4 VR(VPN) Topology Entry for AS(102)/ID(192.4.102.4)
    Topology(base) TID(0) VRF(EIGRP)
%Entry 192.102.5.5/32 not in topology table

```

To fix this, we will set an easily-visible default-metric on CSR4 as well. Like on XRv2, the route is now external and locally originated by CSR4. Any EIGRP speakers behind CSR4 in the customer VPN will not know there is another EIGRP process across MPLS in this configuration.

```

! CSR4
router eigrp VPN
  address-family ipv4 unicast vrf EIGRP autonomous-system 102
  topology base
  default-metric 112347 10203 100 200 1499

R4#show eigrp address-family ipv4 vrf EIGRP topology 192.102.5.5/32
EIGRP-IPv4 VR(VPN) Topology Entry for AS(102)/ID(192.4.102.4)
    Topology(base) TID(0) VRF(EIGRP)
EIGRP-IPv4(102): Topology base(0) entry for 192.102.5.5/32
  State is Passive, Query origin flag is 1, 1 Successor(s), FD is 6692471435
  Descriptor Blocks:
  214.0.0.12, from Redistributed, Send flag is 0x0
    Composite metric is (6692471435/0), route is External
  Vector metric:
    Minimum bandwidth is 112347 Kbit
    Total delay is 102030000000 picoseconds
    Reliability is 100/255
    Load is 200/255
    Minimum MTU is 1499
    Hop count is 0
    Originating router is 192.4.102.4
  External data:
    AS number of route is 214
    External protocol is BGP, external metric is 10880
    Administrator tag is 0 (0x00000000)

```

Unlike CSR5, who only saw one external route, CSR6 will see all routes as external since the router not supporting extended-communities is its local PE (CSR4). We will ensure CSR6 has the route for CSR5's loopback and test reachability.

```

R6#show ip route vrf EIGRP eigrp | section 5\.5
D EX    192.102.5.5
        [170/52290053] via 192.4.102.4, 00:03:25, GigabitEthernet2.102

```

192.102.13.0/32 is subnetted, 1 subnets

```
R6#ping vrf EIGRP 192.102.5.5 source 192.102.6.6
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.102.5.5, timeout is 2 seconds:
Packet sent with a source address of 192.102.6.6
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 6/7/11 ms
```

The capability to disable this is identical for IPv6. Before we disable it, we will quickly verify that both CSR5 and CSR6 IPv6 loopbacks have the proper extended communities on XRv2. There is a new community, 0x8807, and it is so confusing that XR doesn't even know what to call it. The hex values decode into 80:169:171:0:0:0 which is meaningless at first sight. The important thing is the communities are being passed.

```
RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast vrf EIGRP ::192:102:5:5/128 | begin
Local
[snip]
Local
  fe80::5 from 0.0.0.0 (214.0.0.12)
    Origin incomplete, metric 10880, localpref 100, weight 32768, valid,
redistributed, best, group-best, import-candidate
    Received Path ID 0, Local Path ID 1, version 570
    Extended community: COST:128:128:10880 EIGRP route-info:0x8000:0 EIGRP
AD:102:288 EIGRP RHB:255:1:2560 EIGRP LM:0x0:1:1500 EIGRP VRR:0x0:5.5.102.192
RT:214:102
```

```
RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast vrf EIGRP ::192:102:6:6/128 | begin
Local
[snip]
Local, (Received from a RR-client)
  214.0.0.4 (metric 20) from 214.0.0.4 (214.0.0.4)
    Received Label 4043
    Origin incomplete, metric 10880, localpref 100, valid, internal, best,
group-best, import-candidate, imported
    Received Path ID 0, Local Path ID 1, version 593
    Extended community: COST:128:128:10880 EIGRP route-info:0x8000:0 EIGRP
AD:102:288 EIGRP RHB:255:1:2560 EIGRP LM:0xff:1:1500 EIGRP
VRR:0x0:6.102.4.192 0x8807:0x50:0xa9:0xab:0x00:0x00:0x00:0x00 RT:214:102
    Source VRF: EIGRP, Source Route Distinguisher: 214:102
```

For brevity, we will do a quick EIGRP topology check on both CE devices, then verify reachability. The logic of the MPLS network being viewed as a single EIGRP router is still the same as verified by the hop count.

```
R5#show eigrp address-family ipv6 vrf EIGRP topology ::192:102:6:6/128 |
include Hop
```

Hop count is 2

```
R5#show eigrp address-family ipv6 vrf EIGRP topology | section 6:6
P ::192:102:6:6/128, 1 successors, FD is 2048000
   via FE80::12 (2048000/1392640), GigabitEthernet2.102
```

```
R6#show eigrp address-family ipv6 vrf EIGRP topology | section 5:5
P ::192:102:5:5/128, 1 successors, FD is 2048000
   via FE80::4 (2048000/1392640), GigabitEthernet2.102
```

```
R6#show eigrp address-family ipv6 vrf EIGRP topology ::192:102:5:5/128 |
include Hop
```

Hop count is 2

```
R5#ping vrf EIGRP ::192:102:6:6 source ::192:102:5:5
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to ::192:102:6:6, timeout is 2 seconds:
Packet sent with a source address of ::192:102:5:5%EIGRP
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 6/10/26 ms
```

On CSR4, we will disable populating these communities into VPNv6 BGP. We verify that the route learned from CSR6 does not have these communities applied, but the one received from XRv2 does.

```
! CSR4
router eigrp VPN
 address-family ipv6 unicast vrf EIGRP autonomous-system 102
   no populate bgp-ext-comm
```

```
R4#show bgp vpnv6 unicast vrf EIGRP ::192:102:6:6/128 | begin Local
Local
  FE80::6 (FE80::6) (via vrf EIGRP) from 0.0.0.0 (214.0.0.4)
    Origin incomplete, metric 10880, localpref 100, weight 32768, valid,
sourced, best
    Extended Community: RT:214:102
    mpls labels in/out 4043/nolabel
    rx pathid: 0, tx pathid: 0x0
```

```
R4#show bgp vpnv6 unicast vrf EIGRP ::192:102:5:5/128 | begin Local
Local
  ::FFFF:214.0.0.12 (metric 20) (via default) from 214.0.0.12 (214.0.0.12)
    Origin incomplete, metric 10880, localpref 100, valid, internal, best
    Extended Community: RT:214:102 Cost:pre-bestpath:128:10880
    0x8800:32768:0 0x8801:102:288 0x8802:65281:2560 0x8803:1:1500
    0x8806:0:3227911429
    mpls labels in/out nolabel/92010
    rx pathid: 0, tx pathid: 0x0
```

XRv2 learns this route, but is unable to redistribute it into EIGRP as there is no default-metric specified. The EIGRP-v6 process on XRv2 didn't give us a log message this time, so we must be aware of EIGRP redistribution behavior with respect to unspecified seed metrics. We can confirm that other VPNv6 routes, like XRv4's loopback, did get redistributed from BGP into EIGRP successfully. For that route, all of the extended-communities were properly interpreted as the hop count is 1 (the MPLS network is one EIGRP router) and the route is internal.

```
RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast vrf EIGRP ::192:102:6:6/128 | begin
Local
  Local, (Received from a RR-client)
    214.0.0.4 (metric 20) from 214.0.0.4 (214.0.0.4)
      Received Label 4043
      Origin incomplete, metric 10880, localpref 100, valid, internal, best,
group-best, import-candidate, imported
      Received Path ID 0, Local Path ID 1, version 594
      Extended community: RT:214:102
      Source VRF: EIGRP, Source Route Distinguisher: 214:102
```

```
RP/0/0/CPU0:XRv2#show eigrp vrf EIGRP ipv6 topology ::192:102:6:6/128
% IPv6-EIGRP (VRF EIGRP): Route not in topology table of EIGRP VPN.
```

```
RP/0/0/CPU0:XRv2#show eigrp vrf EIGRP ipv6 topology ::192:102:14:14/128
IPv6-EIGRP VR(VPN) AS(102) VRF EIGRP: Topology entry for ::192:102:14:14/128
  State is Passive, Query origin flag is 1, 1 Successor(s), FD is 1377280,
RIB is 10760
  Routing Descriptor Blocks:
    ::ffff:214.0.0.3, from VPNv6 Sourced, Send flag is 0x0
      Composite metric is (1377280/0), Route is Internal (VPNv6 Sourced)
      Vector metric:
        Minimum bandwidth is 1000000 Kbit
        Total delay is 11015625 picoseconds
        Reliability is 255/255
        Load is 1/255
        Minimum MTU is 1500
        Hop count is 1
      Originating router is 14.14.14.14
```

To solve it, we will hardcode a metric to the BGP redistribution, rather than set a default metric. This is accomplished with an RPL and cannot be placed in-line as XE allows.

```
! XRv1 and XRv2
route-policy RPL_BGP_TO_EIGRP
  set eigrp-metric 111111 222 33 44 1455
end-policy

router eigrp VPN
  vrf EIGRP
```



```
address-family ipv6
  redistribute bgp 214 route-policy RPL_BGP_TO_EIGRP
```

The significant downside of this hardcoded-metric approach is that, unless specific prefixes are matched in the RPL, all of the EIGRP metrics will be modified in this way, overriding the legitimate extended-community values. This does allow CSR6's loopback to enter the EIGRP topology, but also overrides the values for XRv4's loopback. Fortunately, the EIGRP ASN is still honored so the route remains internal.

```
RP/0/0/CPU0:XRv2#show eigrp vrf EIGRP ipv6 topology ::192:102:6:6/128
IPv6-EIGRP VR(VPN) AS(102) VRF EIGRP: Topology entry for ::192:102:6:6/128
  State is Passive, Query origin flag is 1, 1 Successor(s), FD is 151388165,
  RIB is 1182720
```

Routing Descriptor Blocks:

```
::ffff:214.0.0.4, from Redistributed, Send flag is 0x0
  Composite metric is (151388165/0), Route is External
  Vector metric:
    Minimum bandwidth is 111111 Kbit
    Total delay is 2220000000 picoseconds
    Reliability is 33/255
    Load is 44/255
    Minimum MTU is 1455
    Hop count is 0
  External data:
    Originating router is 214.0.0.12 (this system)
    AS number of route is 102
    External protocol is BGP, external metric is 10880
    Administrator tag is 0 (0x00000000)
```

```
RP/0/0/CPU0:XRv2#show eigrp vrf EIGRP ipv6 topology ::192:102:14:14/128
IPv6-EIGRP VR(VPN) AS(102) VRF EIGRP: Topology entry for ::192:102:14:14/128
  State is Passive, Query origin flag is 1, 1 Successor(s), FD is 1377280,
  RIB is 1182720
```

Routing Descriptor Blocks:

```
::ffff:214.0.0.3, from VPNv6 Sourced, Send flag is 0x0
  Composite metric is (151388165/0), Route is Internal (VPNv6 Sourced)
  Vector metric:
    Minimum bandwidth is 111111 Kbit
    Total delay is 2220000000 picoseconds
    Reliability is 33/255
    Load is 44/255
    Minimum MTU is 1455
    Hop count is 1
```

A quick routing table check on CSR5 confirms that both routes are installed with the appropriate route types as verified in the EIGRP topology. They have the same RIB metric, which is probably not desirable.

```
R5#show ipv6 route vrf EIGRP eigrp | section include :6:6|:14:14
EX ::192:102:6:6/128 [170/1187840]
```

```

        via FE80::12, GigabitEthernet2.102
D   ::192:102:14:14/128 [90/1187840]
        via FE80::12, GigabitEthernet2.102

```

To fix it, we can modify the RPL on XRv1/XRv2 to target these routes. Since we cannot match on the presence or absence of the cost-community or any advanced EIGRP communities, we can match the next-hop of CSR4 to capture all prefixes easily. Be sure to identify this address as an IPv4-compatible IPv6 address; even though XR displays VPNv6 next-hops within the preceding “::FFFF:”, they are IPv6 next-hops. Notice that now only CSR6’s loopback, the only true external route, is marked as “Redistributed” with a bogus metric while the other routes are “VPNv6 Sourced” with the correct metric. Both the summary and detailed outputs are shown below.

```

! XRv1 and XRv2
route-policy RPL_BGP_TO_EIGRP
  if next-hop in (::FFFF:214.0.0.4) then
    set eigrp-metric 111111 222 33 44 1455
  else
    pass
  endif
end-policy

```

```

RP/0/0/CPU0:XRv2#show eigrp vrf EIGRP ipv6 topology | begin ::192
P ::192:102:5:5/128, 1 successors, FD is 1392640, RIB is 10880
    via fe80::5 (1392640/163840), GigabitEthernet0/0/0/0.102
P ::192:102:6:6/128, 1 successors, FD is 151388165, RIB is 1182720
    via Redistributed (151388165/0)
P ::192:102:13:13/128, 1 successors, FD is 1377280, RIB is 10760
    via VPNv6 Sourced (1377280/0)
P ::192:102:14:14/128, 1 successors, FD is 1377280, RIB is 10760
    via VPNv6 Sourced (1377280/0)

```

```

RP/0/0/CPU0:XRv2#show eigrp vrf EIGRP ipv6 topology ::192:102:6:6/128
IPv6-EIGRP VR(VPN) AS(102) VRF EIGRP: Topology entry for ::192:102:6:6/128
  State is Passive, Query origin flag is 1, 1 Successor(s), FD is 151388165,
RIB is 1182720

```

```

Routing Descriptor Blocks:
::ffff:214.0.0.4, from Redistributed, Send flag is 0x0
  Composite metric is (151388165/0), Route is External
  Vector metric:
    Minimum bandwidth is 111111 Kbit
    Total delay is 2220000000 picoseconds
    Reliability is 33/255
    Load is 44/255
    Minimum MTU is 1455
    Hop count is 0
  External data:
    Originating router is 214.0.0.12 (this system)
    AS number of route is 102

```

```
External protocol is BGP, external metric is 10880
Administrator tag is 0 (0x00000000)
```

```
RP/0/0/CPU0:XRv2#show eigrp vrf EIGRP ipv6 topology ::192:102:13:13/128
IPv6-EIGRP VR(VPN) AS(102) VRF EIGRP: Topology entry for ::192:102:13:13/128
State is Passive, Query origin flag is 1, 1 Successor(s), FD is 1377280,
RIB is 10760
Routing Descriptor Blocks:
::ffff:214.0.0.11, from VPNv6 Sourced, Send flag is 0x0
Composite metric is (1377280/0), Route is Internal (VPNv6 Sourced)
Vector metric:
Minimum bandwidth is 1000000 Kbit
Total delay is 11015625 picoseconds
Reliability is 255/255
Load is 1/255
Minimum MTU is 1500
Hop count is 1
Originating router is 13.13.13.13
```

As we know, CSR4 is now ignoring the extended-community values that XRv2 advertised inside of CSR5's loopback, so connectivity will be broken as it was for IPv4. CSR4 is totally ignoring all BGP-carried EIGRP extended-communities.

```
R4#show eigrp address-family ipv6 vrf EIGRP topology ::192:102:5:5/128
EIGRP-IPv6 VR(VPN) Topology Entry for AS(102)/ID(192.4.102.4)
Topology(base) TID(0) VRF(EIGRP)
%Entry ::192:102:5:5/128 not in topology table
```

We will use the same destructive approach on CSR4 by setting the metric within the redistribute statement. Using “default-metric” was a more appropriate solution as it respects the extended-communities, if they exist.

```
! CSR4
route-map RM_BGP_TO_EIGRP permit 10
set metric 55555 444 33 22 1411

router eigrp VPN
address-family ipv6 unicast vrf EIGRP autonomous-system 102
topology base
redistribute bgp 214 route-map RM_BGP_TO_EIGRP
```

```
R4#show eigrp address-family ipv6 vrf EIGRP topology ::192:102:5:5/128
EIGRP-IPv6 VR(VPN) Topology Entry for AS(102)/ID(192.4.102.4)
Topology(base) TID(0) VRF(EIGRP)
EIGRP-IPv6(102): Topology base(0) entry for ::192:102:5:5/128
State is Passive, Query origin flag is 1, 1 Successor(s), FD is 302776437
Descriptor Blocks:
::FFFF:214.0.0.12, from Redistributed, Send flag is 0x0
```

```

Composite metric is (302776437/0), route is External
Vector metric:
  Minimum bandwidth is 55555 Kbit
  Total delay is 4440000000 picoseconds
  Reliability is 33/255
  Load is 22/255
  Minimum MTU is 1411
  Hop count is 0
External data:
  Originating router is 192.4.102.4 (this system)
  AS number of route is 214
  External protocol is BGP, external metric is 10880
  Administrator tag is 0 (0x00000000)

```

As expected, since CSR4 is the one not supporting the extended-communities, all remote routes are viewed as external, so there is little sense in matching specific entities in the route-map as all routes will have the same suboptimal appearance. The topology lists these as “redistributed” as opposed to “VPNv6 Sourced”.

```

R4#show eigrp address-family ipv6 vrf EIGRP topology | section include ::192
P ::192:102:14:14/128, 1 successors, FD is 302776437
    via Redistributed (302776437/0)
P ::192:102:13:13/128, 1 successors, FD is 302776437
    via Redistributed (302776437/0)
P ::192:102:6:6/128, 1 successors, FD is 1392640
    via FE80::6 (1392640/163840), GigabitEthernet2.102
P ::192:102:5:5/128, 1 successors, FD is 302776437
    via Redistributed (302776437/0)

```

To achieve full reachability, I must instruct XRv1 to perform the same metric manipulations as XRv2 in order for its EIGRP VPN process to redistribute CSR6’s loopback from VPNv4/v6 (configuration not shown since it is identical). We quickly verify that XRv3 has the routes and that they are of the proper type.

```

RP/0/0/CPU0:XRv3#show route vrf EIGRP ipv4 eigrp | include 192.102.
D 192.102.5.5/32 [90/16000] via 192.11.102.11, 02:08:13,
GigabitEthernet0/0/0/0.102
D EX 192.102.6.6/32 [170/3200022] via 192.11.102.11, 00:04:50,
GigabitEthernet0/0/0/0.102
D 192.102.14.14/32 [90/15880] via 192.11.102.11, 1d07h,
GigabitEthernet0/0/0/0.102

```

```

RP/0/0/CPU0:XRv3#show route vrf EIGRP ipv6 eigrp | include ::192:102:
D ::192:102:5:5/128
D EX ::192:102:6:6/128
D ::192:102:14:14/128

```

Likewise on CSR3, it needs to set metrics for routes coming from CSR4 only, since it can honor the extended-communities from XRv1 and XRv2. The configuration for EIGRP-v4 is identical using the default-metric under the EIGRP AFI. The IPv6 configuration varies slightly to account for only adjusting metrics from CSR4. Since XE doesn't support in-line matches like XR does, we define an IPv6 prefix-list with an IPv4-compatible IPv6 address representing CSR4's VPNv6 endpoint. We quickly check the routes on XRv4 to ensure it learned all routes and that their route-types are correct. If "default-metric" cannot be used, using route-maps/RPL in this way can retain the EIGRP metrics for internal routes and seed new ones for external routes.

```
! CSR3
ipv6 prefix-list PL_IPV6_NHOP_CSR4 seq 5 permit ::FFFF:214.0.0.4/128

route-map RM_BGP_TO_EIGRP permit 10
  match ipv6 next-hop prefix-list PL_IPV6_NHOP_CSR4
  set metric 55555 444 33 22 1411
route-map RM_BGP_TO_EIGRP permit 1000

RP/0/0/CPU0:XRv4#show route vrf EIGRP ipv4 eigrp | include 192.102.
D    192.102.5.5/32 [90/16000] via 192.3.102.3, 02:12:01,
GigabitEthernet0/0/0/0.102
D EX 192.102.6.6/32 [170/52290053] via 192.3.102.3, 00:00:18,
GigabitEthernet0/0/0/0.102
D    192.102.13.13/32 [90/15880] via 192.3.102.3, 1d07h,
GigabitEthernet0/0/0/0.102

RP/0/0/CPU0:XRv4#show route vrf EIGRP ipv6 eigrp | include ::192:102:
D    ::192:102:5:5/128
D EX ::192:102:6:6/128
D    ::192:102:13:13/128
```

For completeness, we will advertisement an external route to the MPLS network by redistributing a new loopback on CSR5, much like we did for OSPF. We cannot use CSR6 again because it's PE is not capable of encoding the extended-communities. We also cannot use anything inside 192.0.0.0/8 due to CSR5's network statement.

```
! CSR5
interface Loopback500
  description EXTERNAL TEST EIGRP
  vrf forwarding EIGRP
  ip address 50.50.5.5 255.255.255.255
  ipv6 address ::50:50:5:5/128

route-map RM_CONN_TO_EIGRP permit 10
  match interface Loopback500

router eigrp VPN
  address-family ipv4 unicast vrf EIGRP autonomous-system 102
```

```
topology base
  redistribute connected route-map RM_CONN_TO_EIGRP
```

```
address-family ipv6 unicast vrf EIGRP autonomous-system 102
topology base
  redistribute connected route-map RM_CONN_TO_EIGRP
```

XRv12 learns this route but has some new extended-communities. We can check the EIGRP topology to try and decode the values.

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf EIGRP 50.50.5.5/32
[snip]
  Local
    192.12.102.5 from 0.0.0.0 (214.0.0.12)
      Origin incomplete, metric 10880, localpref 100, weight 32768, valid,
      redistributed, best, group-best, import-candidate
      Received Path ID 0, Local Path ID 1, version 695
      Extended community: COST:128:129:10880 EIGRP route-info:0x0:0 EIGRP
      AD:102:288 EIGRP RHB:255:1:2560 EIGRP LM:0x0:1:1500 EIGRP AR:0:192.102.5.5
      EIGRP PM:11:0 RT:214:102
```

The first 0 represents the remote AS, which is 0 per the EIGRP topology (route was redistributed from “connected”). The next 4 bytes represents the originator ID, which is CSR5’s loopback. The second community carries the remote protocol/metric; I assume 11 means “connected” and the remote metric of 0.

```
RP/0/0/CPU0:XRv2#show eigrp vrf EIGRP ipv4 topology 50.50.5.5/32 | begin
External data
  External data:
    Originating router is 192.102.5.5
    AS number of route is 0
    External protocol is Connected, external metric is 0
    Administrator tag is 0 (0x00000000)
```

A quick look at CSR3’s BGP and EIGRP tables shows similar information. XE is less friendly in decoding information, but converting 3227911429 to hex yields 0xC0660505, which is 192.102.5.5, which is the originator ID. This is important for loop prevention with EIGRP external routes, so transporting it is important.

```
R3#show bgp vpnv4 unicast vrf EIGRP 50.50.5.5/32
[snip]
  Local, (Received from a RR-client)
    214.0.0.12 (metric 30) (via default) from 214.0.0.12 (214.0.0.12)
      Origin incomplete, metric 10880, localpref 100, valid, internal, best
      Extended Community: RT:214:102 Cost:pre-bestpath:129:10880 0x8800:0:0
      0x8801:102:288 0x8802:65281:2560 0x8803:1:1500 0x8804:0:3227911429
      0x8805:11:0
```

```
mpls labels in/out nolabel/92024
rx pathid: 0, tx pathid: 0x0
```

Even though the route is external, carrying the extended-communities still allows the MPLS domain to carry the EIGRP metrics for the external route. XRv4 sees the router as two hops away with the proper vector metric information.

```
RP/0/0/CPU0:XRv3#show eigrp vrf EIGRP ipv4 topology 50.50.5.5/32
IPv4-EIGRP VR(VPN) AS(102) VRF EIGRP: Topology entry for 50.50.5.5/32
  State is Passive, Query origin flag is 1, 1 Successor(s), FD is 2048000,
  RIB is 16000
  Routing Descriptor Blocks:
    192.11.102.11 (GigabitEthernet0/0/0/0.102), from 192.11.102.11, Send flag
  is 0x0
    Composite metric is (2048000/1392640), Route is External
    Vector metric:
      Minimum bandwidth is 1000000 Kbit
      Total delay is 21250000 picoseconds
      Reliability is 255/255
      Load is 1/255
      Minimum MTU is 1500
      Hop count is 2
    External data:
      Originating router is 192.102.5.5
      AS number of route is 0
      External protocol is Connected, external metric is 0
      Administrator tag is 0 (0x00000000)
```

The last significant feature of EIGRP is the site-of-origin (SoO). This is an extended-community carried in a new EIGRP TLV that allows an administrator to identify an EIGRP “site”. A multi-homed site, such as one with backdoor links, might be susceptible to transient routing loops during EIGRP convergence. For example, if XRv4’s loopback fails, it would issue QUERY messages to CSR6 and CSR3 (assume there was a backdoor to CSR3). If CSR4, the remote PE, receives the QUERY before CSR3 has a chance to respond, it will REPLY affirmatively that there is an alternate path (the BGP route). Although count-to-infinity may eventually break the loop, this is not an ideal circumstance. SoO works like route-tags except is more dynamic and solves a very specific problem. Setting it on a PE-CE link on the PE side tells EIGRP to copy the SoO value into a BGP extended-community during redistribution. Below, we can see the new SoO community of 102:14 on CSR3.

```
! CSR3
route-map RM_EIGRP_SOO permit 10
  set extcommunity soo 102:14

interface GigabitEthernet2.102
  ip vrf sitemap RM_EIGRP_SOO
```

```

R3#show bgp vpnv4 unicast vrf EIGRP 192.102.14.14/32
[snip]
  Local
    192.3.102.14 (via vrf EIGRP) from 0.0.0.0 (214.0.0.3)
      Origin incomplete, metric 10752, localpref 100, weight 32768, valid,
sourced, best
      Extended Community: SoO:102:14 RT:214:102 Cost:pre-bestpath:128:10752
        0x8800:32768:0 0x8801:102:282 0x8802:65281:2560 0x8803:65281:1500
        0x8806:0:235802126
      mpls labels in/out 3025/nolabel
      rx pathid: 0, tx pathid: 0x0

```

When remote PEs receive this and perform redistribution back into EIGRP, this community is carried over again. Because CSR4 cannot understand these communities, it does not carry this information from BGP into EIGRP. This is consistent with its treatment of the EIGRP vector metrics as well.

```

R5#show eigrp address-family ipv4 vrf EIGRP topology 192.102.14.14/32 |
include SoO
  Extended Community: SoO:102:14

```

```

R6#show eigrp address-family ipv4 vrf EIGRP topology 192.102.14.14/32 |
include SoO
[no output]

```

```

RP/0/0/CPU0:XRv3#show eigrp vrf EIGRP topology 192.102.14.14/32 | begin Ext
  Extended Community:
    SoO:102:14

```

If another site (including CEs) have an SoO set on an interface, routes received that match the SoO are discarded. We cannot use the backdoor (currently shut down) between CSR6 and XRv4 since CSR4 does not support the extended-communities. Instead, we can configure XRv3 to be in the same “site” as XRv4. This means that XRv3 will not be able to learn routes marked with SoO:104:14 from the PE.

```

! XRv1
router eigrp VPN
  vrf EIGRP
    interface GigabitEthernet0/0/0/0.102
      site-of-origin 102:14

```

```

RP/0/0/CPU0:XRv3#show eigrp vrf EIGRP topology 192.102.14.14/32
% IPv4-EIGRP (VRF EIGRP): Route not in topology table of EIGRP VPN.

```

Notice that this is unlike a route-tag in that routes sent from XRv3 do not have the SoO set. This is set only on the redistribution from EIGRP into BGP, and used as an ingress filter for routes carrying the SoO TLV in EIGRP messages. XRv1 is not aware of any SoO configuration on XRv3 at all.


```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast vrf EIGRP 192.102.13.13 | include
Extended
Extended community: COST:128:128:10752 EIGRP route-info:0x8000:0 EIGRP
AD:102:282 EIGRP RHB:255:1:2560 EIGRP LM:0x0:1:1500 EIGRP VRR:0x0:13.13.13.13
RT:214:102
```

If we bring up the backdoor link between CSR5 and XRv3 using Gig2.202 (not shown), XRv3 can still learn these routes via that link. Notice that the route still has the SoO set; CSR5 did not filter it since it does not have any SoO configurations that match that particular route.

```
RP/0/0/CPU0:XRv3#show eigrp vrf EIGRP topology 192.102.14.14/32
IPv4-EIGRP VR(VPN) AS(102) VRF EIGRP: Topology entry for 192.102.14.14/32
State is Passive, Query origin flag is 1, 1 Successor(s), FD is 2688000,
RIB is 21000
Routing Descriptor Blocks:
192.5.102.5 (GigabitEthernet0/0/0/0.202), from 192.5.102.5, Send flag is
0x0
Composite metric is (2688000/2032640), Route is Internal
Vector metric:
Minimum bandwidth is 1000000 Kbit
Total delay is 31015625 picoseconds
Reliability is 255/255
Load is 1/255
Minimum MTU is 1500
Hop count is 3
Originating router is 14.14.14.14
Extended Community:
SoO:102:14
```

A more realistic use of SoO is to prevent loops within the same site when a backdoor link is present. To do this, we can configure a number of different designs. We will examine the first using IPv4 and the third using IPv6; the second test case is difficult to verify and test.

1. Same SoO on the PEs, configured on the PE-CE link only. This is used so that the two sites have no reachability over MPLS whatsoever. If the two CE routers are actually in the same site, this design is very logical (the backdoor link is a high-speed LAN interface, for example).
2. Different SoO on the PE, configured on the PE-CE link only. This allows failover to occur but once a route comes “full circle” around the backdoor link, it is dropped at ingress on the PE that originally injected it, the route will be discarded. This prevents transient routing loops.
3. SoO X on the local CE’s backdoor link and the local PE’s PE-CE link. SoO Y on the remote CE’s backdoor link and the remote PE’s PE-CE link. This provides better protection than #2, still allows sites to communicate unlike #1, but provides less redundancy if the CE and backdoor routers are not the same device (that is to say, there is a larger EIGRP network in between the CE routers).

The first setup is very straightforward. Both PEs will use an SoO value for the PE-CE link. When the two sites exchange routes, the extended-communities are carried over BGP. We will configure this for the IPv4 AFI only, leaving the advanced example for IPv6. In XR, only one SoO can be applied per interface, per AFI.

```
! XRv1 and XRv2
router eigrp VPN
 vrf EIGRP
  address-family ipv4
    interface GigabitEthernet0/0/0/0.102
      site-of-origin 102:1212
```

In the ideal state, the entire MPLS network looks like a single EIGRP router, and so XRv2 prefers the BGP route to XRv1 versus the EIGRP route via CSR5. XRv1 adds the SoO and XRv2 sees it.

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast vrf EIGRP 192.102.13.13/32 | begin
from
192.11.102.13 from 0.0.0.0 (214.0.0.11)
  Origin incomplete, metric 10752, localpref 100, weight 32768, valid,
redistributed, best, group-best, import-candidate, import suspect
  Received Path ID 0, Local Path ID 1, version 788
  Extended community: SoO:102:1212 COST:128:128:10752 EIGRP route-
info:0x8000:0 EIGRP AD:102:282 EIGRP RHB:255:1:2560 EIGRP LM:0x0:1:1500 EIGRP
VRR:0x0:13.13.13.13 RT:214:102
```

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf EIGRP 192.102.13.13/32 | begin
from
Local, (Received from a RR-client)
214.0.0.11 (metric 10) from 214.0.0.11 (214.0.0.11)
  Received Label 91016
  Origin incomplete, metric 10752, localpref 100, valid, internal, best,
group-best, import-candidate, imported, import suspect
  Received Path ID 0, Local Path ID 1, version 745
  Extended community: SoO:102:1212 COST:128:128:10752 EIGRP route-
info:0x8000:0 EIGRP AD:102:282 EIGRP RHB:255:1:2560 EIGRP LM:0x0:1:1500 EIGRP
VRR:0x0:13.13.13.13 RT:214:102
  Source VRF: EIGRP, Source Route Distinguisher: 214:102
```

XRv2 is not allowed to advertise this route to CSR5 due to SoO. Checking CSR5's EIGRP topology, it only has one route via the backdoor link.

```
R5#show eigrp address-family ipv4 vrf EIGRP topology 192.102.13.13/32
EIGRP-IPv4 VR(VPN) Topology Entry for AS(102)/ID(192.102.5.5)
  Topology(base) TID(0) VRF(EIGRP)
EIGRP-IPv4(102): Topology base(0) entry for 192.102.13.13/32
  State is Passive, Query origin flag is 1, 1 Successor(s), FD is 1376256,
RIB is 10752
```

Descriptor Blocks:

```
192.5.102.13 (GigabitEthernet2.202), from 192.5.102.13, Send flag is 0x0  
[snip]
```

Shutting down this backdoor link completely breaks connectivity between XRv3 and CSR5. This is expected since we told the PEs that these routers were in the same “site”, and thus should not be using the WAN for transport between one another.

```
RP/0/0/CPU0:XRv3#show eigrp vrf EIGRP topology 192.102.5.5/32
```

```
% IPv4-EIGRP (VRF EIGRP): Route not in topology table of EIGRP VPN.
```

```
R5#show eigrp address-family ipv4 vrf EIGRP topology 192.102.13.13/32
```

```
EIGRP-IPv4 VR(VPN) Topology Entry for AS(102)/ID(192.102.5.5)
```

```
Topology(base) TID(0) VRF(EIGRP)
```

```
%Entry 192.102.13.13/32 not in topology table
```

We will bring the backdoor link up for our next test using IPv6. Here, we configure XRv3’s backdoor and XRv1 to be in one site, while CSR5’s backdoor and XRv2 are in another. This ensures that routes from a given site cannot re-enter the source site under any circumstances. It is loosely analogous to the “transit AS” prevention techniques used in BGP.

```
! XRv1  
router eigrp VPN  
vrf EIGRP  
address-family ipv6  
interface GigabitEthernet0/0/0/0.102  
site-of-origin 102:11
```

```
! XRv2  
router eigrp VPN  
vrf EIGRP  
address-family ipv6  
interface GigabitEthernet0/0/0/0.102  
site-of-origin 102:12
```

```
! XRv3  
router eigrp VPN  
vrf EIGRP  
address-family ipv6  
interface GigabitEthernet0/0/0/0.202  
site-of-origin 102:11
```

```
! CSR5  
route-map RM_EIGRP_V6_SOO per 10  
set extcommunity soo 102:12
```

```
interface GigabitEthernet2.202
```

```
ip vrf sitemap RM_EIGRP_V6_SOO
```

We check XRv2 for XRv3's loopback which should have SoO:102:11. Then, we check XRv1 for CSR5's loopback which should have SoO:102:12. The reason XRv1 has two lines of output is because it is an RR-client where XRv2 is an RR.

```
RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast vrf EIGRP ::192:102:13:13/128 |  
include SoO
```

```
Extended community: SoO:102:11 COST:128:128:10752 EIGRP route-  
info:0x8000:0 EIGRP AD:102:282 EIGRP RHB:255:1:2560 EIGRP LM:0x0:1:1500 EIGRP  
VRR:0x0:13.13.13.13 RT:214:102
```

```
RP/0/0/CPU0:XRv1#show bgp vpnv6 unicast vrf EIGRP ::192:102:5:5/128 | include  
SoO
```

```
Extended community: SoO:102:12 COST:128:128:10880 EIGRP route-  
info:0x8000:0 EIGRP AD:102:288 EIGRP RHB:255:1:2560 EIGRP LM:0x0:1:1500 EIGRP  
VRR:0x0:5.5.102.192 RT:214:102
```

```
Extended community: SoO:102:12 COST:128:128:10880 EIGRP route-  
info:0x8000:0 EIGRP AD:102:288 EIGRP RHB:255:1:2560 EIGRP LM:0x0:1:1500 EIGRP  
VRR:0x0:5.5.102.192 RT:214:102
```

These routes are allowed to be learned by remote CEs as there is no SoO on the PE-CE link configured on the CE side. This allows MPLS to interconnect the two sites if the backdoor link fails. The EIGRP metrics have not been adjusted for any kind of route-preference, and the MPLS network does not qualify as a feasible successor, so we must use "all-links" to see the alternate paths when using the summary command.

```
R5#show eigrp address-family ipv6 vrf EIGRP topology all-links | section  
13:13
```

```
P ::192:102:13:13/128, 1 successors, FD is 1376256, serno 198
```

```
via FE80::13 (1376256/131072), GigabitEthernet2.202
```

```
via FE80::12 (2032640/1377280), GigabitEthernet2.102
```

```
RP/0/0/CPU0:XRv3#show eigrp vrf EIGRP ipv6 topology all-links | begin 102:5:5
```

```
P ::192:102:5:5/128, 1 successors, FD is 1392640, RIB is 10880, serno 322
```

```
via fe80::5 (1392640/163840), GigabitEthernet0/0/0/0.202
```

```
via fe80::11 (2048000/1392640), GigabitEthernet0/0/0/0.102
```

We quickly shut down the backdoor link to verify MPLS WAN connectivity with traceroute. This allows to sites to have some form of redundancy with loop protection concurrently.

```
R5#traceroute vrf EIGRP ipv6
```

```
Target IPv6 address: ::192:102:13:13
```

```
Source address: ::192:102:5:5
```

```
[snip]
```

```
Tracing the route to ::192:102:13:13
```

```

1 FD00:192:12:102::12 3 msec 2 msec 2 msec
2 2001:214:11:12::11 [MPLS: Label 91018 Exp 0] 93 msec 6 msec 5 msec
3 ::192:102:13:13 94 msec 6 msec 6 msec

```

To demonstrate the loop protection, we will bring the backdoor link online and shutdown CSR5's PE-CE link. Because this PE-CE link was considered a connected route from CSR5's perspective, shutting down the link will remove this connected route. The link is still up on the PE, XRv2, so it will be redistributed into BGP and assigned SoO:102:12. XRv1 will advertise the PE-CE link from XRv2 to XRv3 since the SoO is SoO:102:11 and does not match XRv1's local SoO. XRv3 will do the same and advertise it to CSR5.

```

RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast vrf EIGRP fd00:192:12:102::/64 |
begin Local$
Local
:: from 0.0.0.0 (214.0.0.12)
Origin incomplete, metric 0, localpref 100, weight 32768, valid,
redistributed, best, group-best, import-candidate
Received Path ID 0, Local Path ID 1, version 677
Extended community: SoO:102:12 COST:128:128:10240 EIGRP route-
info:0x8000:0 EIGRP AD:102:256 EIGRP RHB:255:0:2560 EIGRP LM:0x0:1:1500 EIGRP
VRR:0x0:12.0.0.214 RT:214:102

```

```

RP/0/0/CPU0:XRv1#show bgp vpnv6 unicast vrf EIGRP fd00:192:12:102::/64 |
begin Local$
[snip, alternate RR route not shown]
Not advertised to any peer
Local
214.0.0.12 (metric 10) from 214.0.0.12 (214.0.0.12)
Received Label 92008
Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, imported
Received Path ID 0, Local Path ID 1, version 660
Extended community: SoO:102:12 COST:128:128:10240 EIGRP route-
info:0x8000:0 EIGRP AD:102:256 EIGRP RHB:255:0:2560 EIGRP LM:0x0:1:1500 EIGRP
VRR:0x0:12.0.0.214 RT:214:102
Source VRF: EIGRP, Source Route Distinguisher: 214:102

```

```

RP/0/0/CPU0:XRv3#show eigrp vrf EIGRP ipv6 topology fd00:192:12:102::/64
IPv6-EIGRP VR(VPN) AS(102) VRF EIGRP: Topology entry for fd00:192:12:102::/64
State is Passive, Query origin flag is 1, 1 Successor(s), FD is 1966080,
RIB is 15360
Routing Descriptor Blocks:
fe80::11 (GigabitEthernet0/0/0/0.102), from fe80::11, Send flag is 0x0
Composite metric is (1966080/1310720), Route is Internal
Vector metric:
Minimum bandwidth is 1000000 Kbit
Total delay is 20000000 picoseconds
Reliability is 255/255
Load is 1/255

```

```
Minimum MTU is 1500
Hop count is 1
Originating router is 214.0.0.12
Extended Community:
  SoO:102:12
```

CSR5 should reject the route on ingress because its backdoor link is configured with SoO:102:12. This might reduce some redundancy if there were hosts on that LAN, or other networks directly connected to the PE; CSR5 would no longer have access. Unfortunately, the feature does not appear to work, and CSR5 is able to learn the route through the backdoor. I suspect that the “ip vrf sitemap” command doesn’t work on CEs for IPv6.

```
R5#show eigrp address-family ipv6 vrf EIGRP topology FD00:192:12:102::/64
EIGRP-IPv6 VR(VPN) Topology Entry for AS(102)/ID(192.102.5.5)
  Topology(base) TID(0) VRF(EIGRP)
EIGRP-IPv6(102): Topology base(0) entry for FD00:192:12:102::/64
  State is Passive, Query origin flag is 1, 1 Successor(s), FD is 2621440,
RIB is 20480
Descriptor Blocks:
FE80::13 (GigabitEthernet2.202), from FE80::13, Send flag is 0x0
  Composite metric is (2621440/1966080), route is Internal
  Vector metric:
    Minimum bandwidth is 1000000 Kbit
    Total delay is 30000000 picoseconds
    Reliability is 255/255
    Load is 1/255
    Minimum MTU is 1500
    Hop count is 2
    Originating router is 214.0.0.12
  Extended Community: SoO:102:12
```

Because this test did not work, we will reverse the problem. We will bring CSR5’s link with XRv2 (PE) back up and shut down XRv3’s link to XRv1 (PE) instead. We will now we tracing the PE-CE link between XRv1 and XRv3. We first ensure that XRv1 creates the route, assigns SoO:102:11, and advertises it to XRv2. XRv2 should advertise it to CSR5, who should also see the SoO in the EIGRP topology.

```
RP/0/0/CPU0:XRv1#show bgp vpnv6 unicast vrf EIGRP fd00:192:11:102::/64 |
begin Local$
Local
  :: from 0.0.0.0 (214.0.0.11)
    Origin incomplete, metric 0, localpref 100, weight 32768, valid,
redistributed, best, group-best, import-candidate
    Received Path ID 0, Local Path ID 1, version 688
    Extended community: SoO:102:11 COST:128:128:10240 EIGRP route-
info:0x8000:0 EIGRP AD:102:256 EIGRP RHB:255:0:2560 EIGRP LM:0x0:1:1500 EIGRP
VRR:0x0:11.0.0.214 RT:214:102
```

```

RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast vrf EIGRP fd00:192:11:102::/64 |
begin Local
  Local, (Received from a RR-client)
    214.0.0.11 (metric 10) from 214.0.0.11 (214.0.0.11)
      Received Label 91006
      Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, imported
      Received Path ID 0, Local Path ID 1, version 683
      Extended community: SoO:102:11 COST:128:128:10240 EIGRP route-
info:0x8000:0 EIGRP AD:102:256 EIGRP RHB:255:0:2560 EIGRP LM:0x0:1:1500 EIGRP
VRR:0x0:11.0.0.214 RT:214:102
      Source VRF: EIGRP, Source Route Distinguisher: 214:102

R5#show eigrp address-family ipv6 vrf EIGRP topology FD00:192:11:102::/64
EIGRP-IPv6 VR(VPN) Topology Entry for AS(102)/ID(192.102.5.5)
  Topology(base) TID(0) VRF(EIGRP)
EIGRP-IPv6(102): Topology base(0) entry for FD00:192:11:102::/64
  State is Passive, Query origin flag is 1, 1 Successor(s), FD is 1966080,
RIB is 15360
  Descriptor Blocks:
    FE80::12 (GigabitEthernet2.102), from FE80::12, Send flag is 0x0
      Composite metric is (1966080/1310720), route is Internal
      Vector metric:
        Minimum bandwidth is 1000000 Kbit
        Total delay is 20000000 picoseconds
        Reliability is 255/255
        Load is 1/255
        Minimum MTU is 1500
        Hop count is 1
        Originating router is 214.0.0.11
      Extended Community: SoO:102:11

```

Now, when CSR5 advertises the route to XRv3, the route is rejected. This proves that XR's implementation of SoO works correctly for IPv4 and IPv6, whereas XE only works reliably for IPv4.

```

RP/0/0/CPU0:XRv3#show eigrp vrf EIGRP ipv6 topology fd00:192:11:102::/64
% IPv6-EIGRP (VRF EIGRP): Route not in topology table of EIGRP VPN.

```

Before continuing, we will restore the PE-CE link on XRv3. Just like in the OSPF section, we will use TCL to verify full reachability from CSR5 and CSR6. The validation is not shown, but the script is below.

```

! CSR5 and CSR6
tclsh
foreach x {
192.102.5.5
192.102.6.6
192.102.13.13
192.102.14.14

```

```

50.50.5.5
::192:102:5:5
::192:102:6:6
::192:102:13:13
::192:102:14:14
::50:50:5:5
} { ping vrf EIGRP $x source loopback102 repeat 3 timeout 1 }

```

We will immediately notice the possibility of problems. A single PE (CSR4) routing not supporting the critical EIGRP extended communities can break many things. CSR4 thinks that the best path to CSR6's loopback is via XRv1, which is not true. Because this path has a relatively low pre-bestpath cost-community, it is compared against the local route's default value of 2147483648 (2^{31}) and is much better. If CSR4 supported the community, the cost-community on the local route would certainly be better.

```

R4#show bgp vpnv4 unicast vrf EIGRP 192.102.6.6/32
BGP routing table entry for 214:102:192.102.6.6/32, version 884
Paths: (3 available, best #1, table EIGRP, RIB-failure(17) - next-hop
mismatch)
  Not advertised to any peer
  Refresh Epoch 4
  Local
    214.0.0.11 (metric 30) (via default) from 214.0.0.3 (214.0.0.3)
      Origin incomplete, metric 3210262, localpref 100, valid, internal, best
      Extended Community: SoO:102:1212 RT:214:102
      Cost:pre-bestpath:129:3210262 (default-2144273385) 0x8800:0:0
      0x8801:102:139776 0x8802:59907:207360 0x8803:123:1500
      0x8804:102:3590324236 0x8805:9:10880
      Originator: 214.0.0.11, Cluster list: 0.0.3.12
      mpls labels in/out nolabel/91030
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  Local
    214.0.0.11 (metric 30) (via default) from 214.0.0.12 (214.0.0.12)
      Origin incomplete, metric 3210262, localpref 100, valid, internal
      Extended Community: SoO:102:1212 RT:214:102
      Cost:pre-bestpath:129:3210262 (default-2144273385) 0x8800:0:0
      0x8801:102:139776 0x8802:59907:207360 0x8803:123:1500
      0x8804:102:3590324236 0x8805:9:10880
      Originator: 214.0.0.11, Cluster list: 0.0.3.12
      mpls labels in/out nolabel/91030
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  Local
    192.4.102.6 (via vrf EIGRP) from 0.0.0.0 (214.0.0.4)
      Origin incomplete, metric 10880, localpref 100, weight 32768, valid,
sourced
      Extended Community: RT:214:102

```



```
rx pathid: 0, tx pathid: 0
```

To fix this problem, we must either re-enable the EIGRP extended-community processing on CSR4 or instruct BGP to ignore these communities entirely. We will choose the latter solution as it is more complicated. This feature cannot be modified on a per AF-basis and is global to the BGP process in its entirety. CSR4 will totally ignore the cost-community now, relying on the regular BGP best-path selection algorithm. Now, the local route is the best-path due to the weight attribute.

```
! CSR4
router bgp 214
  bgp bestpath cost-community ignore

R4#show bgp vpnv4 unicast vrf EIGRP 192.102.6.6/32 bestpath
[snip]
  Local
    192.4.102.6 (via vrf EIGRP) from 0.0.0.0 (214.0.0.4)
      Origin incomplete, metric 10880, localpref 100, weight 32768, valid,
sourced, best
      Extended Community: RT:214:102
      mpls labels in/out 4053/nolabel
      rx pathid: 0, tx pathid: 0x0
```

This still may not fix all of the problems in the network. In order for this to be effective, we would need to enable it everywhere, which would break many other things we tested. We will leave the EIGRP topology “broken” as a reference of how not to configure EIGRP over L3VPN. As a recommendation, either use cost-community everywhere or don’t use it anywhere. XRv1 is preferring XRv2 to reach CSR6 now, and advertises this as its bestpath to CSR4. Even though CSR4 is picking the proper route, the other routers are not. Manually manipulating the entire network to fix these issues is not worth it, so again, we leave the network broken.

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast vrf EIGRP 192.102.6.6/32 brief |
begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 214:102 (default for vrf EIGRP)
* i192.102.6.6/32      214.0.0.12          3210262    100      0 ?
*>i                    214.0.0.12          3210262    100      0 ?
```

As one final comment, EIGRP is capable of carrying SoO inside of external routes. Looking at 50.50.5.5/32, which was redistributed into EIGRP on CSR5, it carries the SoO from XRv2. When CSR3 redistributes it back into EIGRP, the SoO is carried with it as well. We won’t retest SoO for this specific route, but be aware that the SoO rules apply to all EIGRP routes that carry the SoO attribute.

```
R3#show bgp vpnv4 unicast vrf EIGRP 50.50.5.5/32
BGP routing table entry for 214:102:50.50.5.5/32, version 234
Paths: (1 available, best #1, table EIGRP)
  Advertised to update-groups:
```

14

```
Refresh Epoch 1
Local, (Received from a RR-client)
214.0.0.12 (metric 30) (via default) from 214.0.0.12 (214.0.0.12)
Origin incomplete, metric 10880, localpref 100, valid, internal, best
Extended Community: SoO:102:1212 RT:214:102 Cost:pre-bestpath:129:10880
0x8800:0:0 0x8801:102:288 0x8802:65281:2560 0x8803:1:1500
0x8804:0:3227911429 0x8805:11:0
mpls labels in/out nolabel/92024
rx pathid: 0, tx pathid: 0x0
```

```
R3#show eigrp address-family ipv4 vrf EIGRP topology 50.50.5.5/32
EIGRP-IPv4 VR(VPN) Topology Entry for AS(102)/ID(192.3.102.3)
Topology(base) TID(0) VRF(EIGRP)
EIGRP-IPv4(102): Topology base(0) entry for 50.50.5.5/32
State is Passive, Query origin flag is 1, 1 Successor(s), FD is 1392640
Descriptor Blocks:
214.0.0.12, from VPNv4 Sourced, Send flag is 0x0
Composite metric is (1392640/0), route is External (VPNv4 Sourced)
Vector metric:
Minimum bandwidth is 1000000 Kbit
Total delay is 11250000 picoseconds
Reliability is 255/255
Load is 1/255
Minimum MTU is 1500
Hop count is 1
Extended Community: SoO:102:1212
External data:
Originating router is 192.102.5.5
AS number of route is 0
External protocol is Connected, external metric is 0
Administrator tag is 0 (0x00000000)
```

38.1.4 IS-IS

IS-IS is not commonly used as PE-CE routing protocol as it is not common in enterprise (customer) networks. VRF-aware IS-IS is also not currently supported on XR, either, so demonstrating it in this topology is difficult. Additionally, IPv6 IS-IS is only supported in the XE global process. These limitations will drastically reduce the length of this section.

```
R3(config-subif)#ipv6 router isis 103
%ISIS: Cannot enable ISIS-IPv6 for non-default VRF process
```

```
R3(config-subif)#ipv6 router isis
%ISIS: Cannot enable ISIS-IPv6 for non-default VRF interface
```

The protocol itself does not support carrying SoO attributes nor is there is any concept of a sham-link for back-door protection. VRF-aware IS-IS is only enabled between CSR4 (PE) and CSR6 (CE) for a brief

demonstration. First, we verify the neighbor is up and IS-IS is operational. The configuration is identical to global-table IS-IS except the “vrf” keyword is included under the process. The PE configuration is shown below, which also redistributes from BGP.

```
! CSR4
router isis 103
 vrf ISIS
 net 49.0103.0000.0000.0004.00
 is-type level-2-only
 redistribute bgp 214
```

```
R4#show isis 103 neighbors
```

```
Tag 103:
```

System Id	Type	Interface	IP Address	State	Holdtime	Circuit Id
R6	L2	Gi2.103	192.4.103.6	UP	23	00

Because there is no remote IS-IS VPN, we will exchange VPN routes with CSR1 (central services) by importing RT:214:8. CSR8 will import RT:214:103 as well (not shown). Now, CSR6 will learn new routes via CSR4. We can see that all 15 routes are IS-IS Level 2 routes. The LSPDB shows that these are IP-External entries as part of CSR4’s LSP. Their metrics are 0 at the point of redistribution, much like loopbacks.

```
R6#show ip route vrf ISIS summary | section isis 103
```

```
isis 103          0          15          0          1440          4320
  Level 1: 0 Level 2: 15 Inter-area: 0
```

```
R6#show isis 103 database detail level-2 R4.00-00
```

```
Tag 103:
```

```
IS-IS Level-2 LSP R4.00-00
```

LSPID	LSP Seq Num	LSP Checksum	LSP Holdtime	ATT/P/OL
R4.00-00	0x000001C4	0xCAFB	1116	0/0/0
Area Address: 49.0103				
NLPID: 0xCC				
Hostname: R4				
Metric: 10 IS R6.00				
IP Address: 192.4.103.4				
Metric: 10 IP 192.4.103.0 255.255.255.0				
Metric: 0 IP-External 82.125.160.0 255.255.255.240				
Metric: 0 IP-External 82.125.160.16 255.255.255.240				
[snip]				
Metric: 0 IP-External 82.125.160.0 255.255.255.192				
Metric: 0 IP-External 82.125.160.64 255.255.255.192				
Metric: 0 IP-External 10.1.8.0 255.255.255.0				

It is worth noting that when redistributing from IS-IS, the connected routes are not included. We confirm this on CSR4 by checking the VPNv4 table for the CSR4-CSR6 PE-CE link. We verify that CSR6’s loopback made it into the table, which is an IS-IS route on CSR4.

```

R4#show bgp vpnv4 unicast vrf ISIS 192.4.103.0/24
% Network not in table

R4#show bgp vpnv4 unicast vrf ISIS 192.103.6.6/32
[snip]
Local
  192.4.103.6 (via vrf ISIS) from 0.0.0.0 (214.0.0.4)
    Origin incomplete, metric 10, localpref 100, weight 32768, valid,
sourced, best
    Extended Community: RT:214:103
    mpls labels in/out 4055/nolabel
    rx pathid: 0, tx pathid: 0x0

```

Something interesting about SoO is that, when configured on a PE router, is entirely protocol-agnostic. IS-IS cannot carry nor interpret SoO, but routes learned on the PE-CE link can still have this community encoded when redistributing into BGP. We quickly test this on CSR4 by applying the SoO on the link towards CSR6 and checking the BGP VPNv4 table.

```

! CSR4
route-map RM_SOO permit 10
  set extcommunity soo 103:4

interface GigabitEthernet2.103
  ip vrf sitemap RM_SOO

R4#show bgp vpnv4 unicast vrf ISIS 192.103.6.6/32
BGP routing table entry for 214:103:192.103.6.6/32, version 1177
BGP Bestpath: cost-community-ignore
Paths: (1 available, best #1, table ISIS)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local
    192.4.103.6 (via vrf ISIS) from 0.0.0.0 (214.0.0.4)
      Origin incomplete, metric 10, localpref 100, weight 32768, valid,
sourced, best
      Extended Community: SoO:103:4 RT:214:103
      mpls labels in/out 4056/nolabel
      rx pathid: 0, tx pathid: 0x0

```

This could be used to filter routes advertised to CSR1 on CSR8 using BGP SoO, which is covered in detail later. Identifying this peer as being in the same “site” as CSR6 means that routes can no longer be exchanged within the IPv4 AFI. CSR8 cannot advertise CSR6’s loopback to CSR1 now, but it can import them locally.

```
! CSR8
```

```
router bgp 214
 address-family ipv4 vrf BOTTOM
  neighbor 10.1.8.1 soo 103:4
```

```
R8#show bgp vpnv4 unicast vrf BOTTOM 192.103.6.6/32
```

```
[snip]
```

```
Local, imported path from 214:103:192.103.6.6/32 (global)
 214.0.0.4 (metric 10) (via default) from 214.0.0.3 (214.0.0.3)
  Origin incomplete, metric 10, localpref 100, valid, internal, best
  Extended Community: SoO:103:4 RT:214:103
  Originator: 214.0.0.4, Cluster list: 0.0.3.12
  mpls labels in/out nolabel/4056
  rx pathid: 0, tx pathid: 0x0
```

```
R8#show bgp vpnv4 unicast vrf BOTTOM neighbors 10.1.8.1 advertised-routes |
include 192.103
```

```
[no output]
```

```
R1#show bgp ipv4 unicast 192.103.6.6/32
```

```
% Network not in table
```

Interestingly, because IS-IS does not understand SoO, CSR4 still freely redistributes CSR1's routes from BGP into IS-IS despite the SoO being the same. Selecting any route from CSR1, we can see the SoO is set and IS-IS appears to ignore it. CSR6 installs this route in its RIB as expected.

```
R4#show bgp vpnv4 unicast vrf ISIS 82.125.160.48/28
```

```
[snip]
```

```
82, imported path from 214:8:82.125.160.48/28 (global)
 214.0.0.8 (metric 10) (via default) from 214.0.0.3 (214.0.0.3)
  Origin IGP, metric 0, localpref 100, valid, internal, best
  Extended Community: SoO:103:4 RT:214:8
  Originator: 214.0.0.8, Cluster list: 0.0.3.12
  mpls labels in/out nolabel/8028
  rx pathid: 0, tx pathid: 0x0
```

```
R4#show isis 103 database level-2 detail R4.00-00 | include 82.125.160.48
```

```
Metric: 0          IP-External 82.125.160.48 255.255.255.240
```

```
R6#show ip route vrf ISIS 82.125.160.48 255.255.255.240
```

```
Routing Table: ISIS
```

```
Routing entry for 82.125.160.48/28
```

```
Known via "isis", distance 115, metric 10, type level-2
Redistributing via isis 103
Last update from 192.4.103.4 on GigabitEthernet2.103, 00:19:25 ago
Routing Descriptor Blocks:
* 192.4.103.4, from 192.4.103.4, 00:19:25 ago, via GigabitEthernet2.103
  Route metric is 10, traffic share count is 1
```

In the interest of keeping the network interesting, we will leave the IS-IS to BGP AS 82 connectivity broken intentionally. Removing the SoO from either CSR4's PE-CE link or CSR8's neighbor statement to CSR1 will restore connectivity. IS-IS was never designed to use SoO in this context but this lab illustrates the protocol-agnostic characteristics of SoO.

38.1.5 BGP and Site-of-Origin (SoO)

BGP is the simplest, most popular, most appropriate PE-CE routing protocol option. Traditionally, it is used between ASNs for inter-network routing, and MPLS L3VPN PE-CE is a perfect use-case. BGP's flexible filtering support, as well as SoO, make it the best option. It is also widely supported by service providers in real life since it is supported on all vendor platforms and is straightforward to implement in many cases. The configuration on CSR5 (CE) and XRv2 (PE) are shown below as a summary reference. Note that the backdoor connections are configured but in a shutdown state initially.

```
! CSR5
router bgp 104
 no bgp default ipv4-unicast

address-family ipv4 vrf BGP
 redistribute connected
 neighbor 192.5.104.13 remote-as 104
 neighbor 192.5.104.13 shutdown
 neighbor 192.5.104.13 activate
 neighbor 192.12.104.12 remote-as 214
 neighbor 192.12.104.12 activate

address-family ipv6 vrf BGP
 redistribute connected
 neighbor FD00:192:5:104::13 remote-as 104
 neighbor FD00:192:5:104::13 shutdown
 neighbor FD00:192:5:104::13 activate
 neighbor FD00:192:12:104::12 remote-as 214
 neighbor FD00:192:12:104::12 activate

! XRv12
router bgp 214
 vrf BGP
  rd 214:104
  address-family ipv4 unicast
  address-family ipv6 unicast

  neighbor 192.12.104.5
   remote-as 104
  address-family ipv4 unicast
   route-policy PASS in
   route-policy PASS out

  neighbor fd00:192:12:104::5
```

```

remote-as 104
address-family ipv6 unicast
route-policy PASS in
route-policy PASS out

```

We quickly verify that the neighbors are up between PE-CE on these routers. The XR routers work well without needing to define multiple sessions for different AFs as the IPv6 next-hops are automatically adjusted. The verifications on the left-half of the network are not shown for brevity. The backdoor peer is marked as “Active” because XRv3 doesn’t have the peer shutdown, only CSR5 does, but the session is unable to form despite XRv3 attempting to establish it.

```

RP/0/0/CPU0:XRv3#show bgp vrf BGP summary | begin Neighbor
Neighbor      Spk    AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
192.5.104.5    0    104    111      86         0    0    0    3d07h  Active
192.11.104.11  0    214   5873   5849        74    0    0    4d01h      0

RP/0/0/CPU0:XRv3#show bgp vrf BGP ipv6 unicast summary | begin Neighbor
Neighbor      Spk    AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
192.11.104.11  0    214   5874   5850        67    0    0    4d01h      0
fd00:192:5:104::5
                0    104    108      86         0    0    0    3d07h  Active

R5#show bgp vpnv4 unicast vrf BGP summary | begin ^Neighbor
Neighbor      V      AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  State/PfxRcd
192.5.104.13   4     104      0        0         1    0    0    3d07h  Idle (Admin)
192.12.104.12  4     214   5858   6509        350    0    0    4d01h      0

R5#show bgp vpnv6 unicast vrf BGP summary | begin ^Neighbor
Neighbor      V      AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  State/PfxRcd
FD00:192:5:104::13
                4     104      0        0         1    0    0    3d07h  Idle (Admin)
FD00:192:12:104::12
                4     214   5856   6493       1288    0    0    4d01h      0

```

Because we are using the same basic MPLS L3VPN construct as we did for the other tests, we should see all of the customer VPN routes in each of the CE sites. The output above shows 0, which means the peer is up but no prefixes were received. We quickly check XRv2 to see if it is advertising routes to CSR5; it isn’t. XR is trying to be “smart” and not advertise routes to CSR5 that it knows will be dropped on ingress by the eBGP loop prevention mechanism. If a peer has AS X and AS X appears in the AS_PATH of a BGP route, it is highly likely that the peer will reject the routes on ingress.

```

RP/0/0/CPU0:XRv2#show bgp vrf BGP ipv4 unicast neighbors 192.12.104.5
advertised-routes
[no output]

```

Looking across the network to CSR3 and CSR4 as next-hops, we see that XRv2 has learned VPNv4 routes from those peers and imported them into the BGP VPN. Because they have AS 104 in the path, XR knows that CSR5 cannot accept this, and doesn’t even attempt to advertise it, as seen above.

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf BGP | include 214.0.0.[34]
*>i192.3.104.0/24      214.0.0.3      0      100      0 104 ?
*>i192.4.104.0/24      214.0.0.4      0      100      0 104 ?
*>i192.6.104.0/24      214.0.0.3      0      100      0 104 ?
*>i192.104.6.6/32      214.0.0.4      0      100      0 104 ?
*>i192.104.14.14/32    214.0.0.3      0      100      0 104 ?
```

We can disable this intelligent behavior on XRv2 for IPv4/v6 as shown below. Now, XRv2 is advertising these routes to CSR5.

```
! XRv12
router bgp 214
 vrf BGP
  address-family ipv4 unicast
    as-path-loopcheck out disable

  address-family ipv6 unicast
    as-path-loopcheck out disable
```

```
RP/0/0/CPU0:XRv2#show bgp vrf BGP ipv4 unicast neighbors 192.12.104.5
advertised-routes
```

Network	Next Hop	From	AS Path
Route Distinguisher: 214:104 (default for vrf BGP)			
192.3.104.0/24	192.12.104.12	214.0.0.3	214 104?
192.4.104.0/24	192.12.104.12	214.0.0.4	214 104?
192.6.104.0/24	192.12.104.12	214.0.0.3	214 104?
192.11.104.0/24	192.12.104.12	214.0.0.11	214 104?
192.104.6.6/32	192.12.104.12	214.0.0.4	214 104?
192.104.13.13/32	192.12.104.12	214.0.0.11	214 104?
192.104.14.14/32	192.12.104.12	214.0.0.3	214 104?

CSR5 still only has a set of local routes due to the loop prevention mechanism. Debugging BGP updates inbound on CSR5 reveals this. We expected this to happen once the PE started advertising the routes.

```
R5#show bgp vpnv4 unicast vrf BGP | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 214:104 (default for vrf BGP)
*> 192.5.104.0      0.0.0.0      0      32768 ?
*> 192.12.104.0     0.0.0.0      0      32768 ?
*> 192.104.5.5/32   0.0.0.0      0      32768 ?
```

```
! CSR5
BGP(0): 192.12.104.12 rcv UPDATE about 192.5.104.0/24 -- DENIED due to: AS-
PATH contains our own AS;
BGP: 192.12.104.12 Modifying prefix 192.5.104.0/24 from 0 -> 4 address
```


There are two options to fix it. The first solution is CE-based. This will relax the loop-prevention rule by telling CSR5 to allow BGP routes with a certain number of the local-AS in the AS path. This is a number between 1 and 10 on both XE and XR platforms; the local-AS path can occur anywhere in the AS path up to the number of times we specify. The larger the number, the larger the possibility of a loop. The configurations for CSR5 are shown below which allows exactly 1 instance of AS 104 in the AS path of any received route from XRv2. After configuring it, we will ensure the configuration is applied properly to the neighbor. Because we have fixed the problem for IPv4 and IPv6, we will verify both concurrently.

```
! CSR5
router bgp 104
 address-family ipv4 vrf BGP
   neighbor 192.12.104.12 allowas-in 1

 address-family ipv6 vrf BGP
   neighbor FD00:192:12:104::12 allowas-in 1
```

```
R5#show bgp vpnv4 unicast vrf BGP neighbors 192.12.104.12 | include For
address|allow
```

```
For address family: VPNv4 Unicast
My AS number is allowed for 1 number of times
```

```
R5#show bgp vpnv6 unicast vrf BGP neighbors FD00:192:12:104::12 | include For
address|allow
```

```
For address family: VPNv6 Unicast
My AS number is allowed for 1 number of times
```

Now, we are able to learn all of the remote prefixes from the other BGP speakers. AS 104 is still in the path, but this is permitted one time. If a remote CE router applied AS path prepending to influence the SP's routing decision, this solution would not work without increasing the value up from 1.

```
R5#show bgp vpnv4 unicast vrf BGP | include 192.12.104.12
```

```
*> 192.3.104.0      192.12.104.12      0 214 104 ?
*> 192.4.104.0      192.12.104.12      0 214 104 ?
*> 192.6.104.0      192.12.104.12      0 214 104 ?
*> 192.11.104.0     192.12.104.12      0 214 104 ?
*> 192.104.6.6/32   192.12.104.12      0 214 104 ?
*> 192.104.13.13/32 192.12.104.12      0 214 104 ?
*> 192.104.14.14/32 192.12.104.12      0 214 104 ?
```

```
R5#show bgp vpnv6 unicast vrf BGP | begin Network
```

```
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 214:104 (default for vrf BGP)
*>  ::192:104:5:5/128
      ::           0           32768 ?
*>  ::192:104:6:6/128
      FD00:192:12:104::12
      0 214 104 ?
```

```
*> ::192:104:13:13/128
      FD00:192:12:104::12
                                0 214 104 ?
[snip]
```

As a CE within the same VPN, XRv13 will have the same problem, but we can use an alternative solution. The second and most straightforward solution is to have the PE routers remark all instances of the customer AS number with the SP's AS number. That is, every instance of 104 in the AS path will be rewritten to 214 when advertised to the eBGP peer inside the VPN. The AS path length remains the same, but the values change. XRv1 will iterate over the AS path and make this adjustment before advertising the routes, "overriding" the VPN AS with the SP AS, hence the feature's name. We also don't have to disable the outbound loop-check on XRv1 since the AS override operation happens before the outbound route loop-check/advertisements. After configuring the feature, we can verify the configuration worked by checking the neighbor details.

```
! XRv1
router bgp 214
 vrf BGP
  neighbor 192.11.104.13
    address-family ipv4 unicast
      as-override
    address-family ipv6 unicast
      as-override
```

```
RP/0/0/CPU0:XRv1# show bgp vrf BGP ipv4 unicast neighbors 192.11.104.13 |
utility egrep 'For Address|override'
For Address Family: IPv4 Unicast
  AS override is set
For Address Family: IPv6 Unicast
  AS override is set
```

Next, we can verify that XRv1 is advertising these BGP VPN routes to the CE, which it is. This output does not account for the AS override operation and shows the original VPN AS.

```
RP/0/0/CPU0:XRv1#show bgp vrf BGP ipv4 unicast neighbors 192.11.104.13
advertised-routes
```

Network	Next Hop	From	AS Path
Route Distinguisher: 214:104 (default for vrf BGP)			
192.3.104.0/24	192.11.104.11	214.0.0.3	214 104?
192.4.104.0/24	192.11.104.11	214.0.0.3	214 104?
192.6.104.0/24	192.11.104.11	214.0.0.3	214 104?
192.12.104.0/24	192.11.104.11	214.0.0.12	214 104?
192.104.5.5/32	192.11.104.11	214.0.0.12	214 104?
192.104.6.6/32	192.11.104.11	214.0.0.3	214 104?
192.104.14.14/32	192.11.104.11	214.0.0.3	214 104?

```
RP/0/0/CPU0:XRv1#show bgp vrf BGP ipv6 unicast neighbors 192.11.104.13
advertised-routes
Network                Next Hop          From          AS Path
Route Distinguisher: 214:104 (default for vrf BGP)
::192:104:5:5/128    fd00:192:11:104::11
                        214.0.0.12      214 104?
::192:104:6:6/128    fd00:192:11:104::11
                        214.0.0.3       214 104?
::192:104:14:14/128
                        fd00:192:11:104::11
                        214.0.0.3     214 104?

[snip]
```

Last, we verify that the CE receives the routes without seeing its own AS. The originating AS appears to be 214 according to the new AS path. This scales much better than “allowas-in” because remote CEs can use AS path prepending without necessitating any changes on the local CE.

```
RP/0/0/CPU0:XRv3#show bgp vrf BGP ipv4 unicast neighbors 192.11.104.11 routes
| begin Network
      Network                Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 214:104 (default for vrf BGP)
*> 192.3.104.0/24          192.11.104.11
                                0 214 214 ?
*> 192.4.104.0/24          192.11.104.11
                                0 214 214 ?
*> 192.6.104.0/24          192.11.104.11
                                0 214 214 ?
*> 192.12.104.0/24         192.11.104.11
                                0 214 214 ?
*> 192.104.5.5/32          192.11.104.11
                                0 214 214 ?
*> 192.104.6.6/32          192.11.104.11
                                0 214 214 ?
*> 192.104.14.14/32        192.11.104.11
                                0 214 214 ?
```

```
RP/0/0/CPU0:XRv3#show bgp vrf BGP ipv6 unicast neighbors 192.11.104.11 routes
| begin Network
      Network                Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 214:104 (default for vrf BGP)
*> ::192:104:5:5/128    fd00:192:11:104::11
                                0 214 214 ?
*> ::192:104:6:6/128    fd00:192:11:104::11
                                0 214 214 ?
*> ::192:104:14:14/128
                        fd00:192:11:104::11
                                0 214 214 ?

[snip]
```

Now, CSR5 and XRv13 have reachability over the SP network to one another, but CSR6 and XRv14 will have the same problems.

```
R5#ping vrf BGP 192.104.13.13 source 192.104.5.5
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.104.13.13, timeout is 2 seconds:
```

Packet sent with a source address of 192.104.5.5

!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 5/5/6 ms

For variety, we will troubleshoot XRv4 as if we don't understand the problem. XRv4 only sees its local routes. Debugging BGP updates inbound, we see similar messages as CSR5 whereby the routes advertised by the PE are rejected due to the AS-path. It is interesting to note that XE makes no attempt to filter the routes outbound despite being able to see the loop; we had to disable this intelligent behavior on XRv2.

```
RP/0/0/CPU0:XRv4#show bgp vrf BGP ipv4 unicast | begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 214:104 (default for vrf BGP)					
*> 192.3.104.0/24	0.0.0.0	0		32768	?
*> 192.6.104.0/24	0.0.0.0	0		32768	?
*> 192.104.14.14/32	0.0.0.0	0		32768	?

```
! XRv4
```

```
bgp[1052]: [default-rtr] VRF BGP (ip4u): UPDATE from 192.3.104.3, prefix  
192.11.104.0/24 (path ID: none) DENIED due to:
```

```
bgp[1052]: [default-rtr] VRF BGP (ip4u): as-path contains our own AS, or 0;
```

We already know there are two solutions: allow AS 104 in on the CE (allowas-in), or change 104 to 214 outbound on the PE (as-override). We will use "allowas-in" to see how it works on XR. As always, after configuring the feature we will verify it was applied to the neighbor properly.

```
! XRv14
```

```
router bgp 104  
vrf BGP  
neighbor 192.3.104.3  
address-family ipv4 unicast  
allowas-in 1  
neighbor fd00:192:3:104::3  
address-family ipv6 unicast  
allowas-in 1
```

```
RP/0/0/CPU0:XRv4#show bgp vrf BGP ipv4 unicast neighbors 192.3.104.3 |  
utility egrep 'For Address|allow'
```

```
For Address Family: IPv4 Unicast
```

```
Maximum prefixes allowed 1048576
```

```
My AS number is allowed 1 times in received updates
```

```
RP/0/0/CPU0:XRv4#show bgp vrf BGP ipv6 unicast neighbors fd00:192:3:104::3 |  
utility egrep 'For Address|allow'
```

```
For Address Family: IPv6 Unicast
```

```
Maximum prefixes allowed 524288
```

```
My AS number is allowed 1 times in received updates
```

Next, we verify that XRv14 is actually learning routes with AS 104 in the path for both IPv4 and IPv6. This would imply that the configuration worked. AS 104 is still in the AS path which means the provider did not rewrite it.

```
RP/0/0/CPU0:XRv4#show bgp vrf BGP ipv4 unicast regexp 104$ | begin Network
      Network                Next Hop                Metric LocPrf Weight Path
Route Distinguisher: 214:104 (default for vrf BGP)
*> 192.4.104.0/24             192.3.104.3                                0 214 104 ?
*> 192.5.104.0/24             192.3.104.3                                0 214 104 ?
*> 192.11.104.0/24            192.3.104.3                                0 214 104 ?
*> 192.12.104.0/24            192.3.104.3                                0 214 104 ?
*> 192.104.5.5/32             192.3.104.3                                0 214 104 ?
*> 192.104.6.6/32             192.3.104.3                                0 214 104 ?
*> 192.104.13.13/32           192.3.104.3                                0 214 104 ?

RP/0/0/CPU0:XRv4#show bgp vrf BGP ipv6 unicast regexp 104$ | begin Network
      Network                Next Hop                Metric LocPrf Weight Path
Route Distinguisher: 214:104 (default for vrf BGP)
*> ::192:104:5:5/128          fd00:192:3:104::3                                0 214 104 ?
*> ::192:104:6:6/128          fd00:192:3:104::3                                0 214 104 ?
*> ::192:104:13:13/128        fd00:192:3:104::3                                0 214 104 ?

[snip]
```

A quick ping check to CSR5 and XRv13, using a mix of IPv4 and IPv6, shows that XRv14 now has connectivity over the VPN.

```
RP/0/0/CPU0:XRv4#ping vrf BGP 192.104.5.5 source 192.104.14.14
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.104.5.5, timeout is 2 seconds:
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/7/9 ms

RP/0/0/CPU0:XRv4#ping vrf BGP ::192:104:13:13 source ::192:104:14:14
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to ::192:104:13:13, timeout is 2 seconds:
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/6/9 ms
```

Last, we need to fix CSR6. We have already seen the problem several times so we will fix it quickly by enabling AS override on CSR4. After configuring, we verify that it was applied to the VPN peer.

! CSR6

```
router bgp 214
 address-family ipv4 vrf BGP
   neighbor 192.4.104.6 as-override
 address-family ipv6 vrf BGP
   neighbor FD00:192:4:104::6 as-override
```

```
R4#show bgp vpnv4 unicast vrf BGP neighbors 192.4.104.6 | include For
address|Override
```

```
For address family: VPNv4 Unicast
Overrides the neighbor AS with my AS before sending updates
```

```
R4#show bgp vpnv6 unicast vrf BGP neighbors FD00:192:4:104::6 | include For
address|Override
```

```
For address family: VPNv6 Unicast
Overrides the neighbor AS with my AS before sending updates
```

CSR6 receives the IPv4 and IPv6 routes. For variety, we will check the specific IPv4 route to XRv4 and IPv6 route to CSR5. Notice that the AS path is “214 214” as expected per the AS override feature.

```
R6#show bgp vpnv4 unicast vrf BGP 192.104.13.13/32
BGP routing table entry for 214:104:192.104.13.13/32, version 29
Paths: (1 available, best #1, table BGP)
```

```
Not advertised to any peer
```

```
Refresh Epoch 2
```

```
214 214
```

```
192.4.104.4 (via vrf BGP) from 192.4.104.4 (214.0.0.4)
```

```
Origin incomplete, localpref 100, valid, external, best
```

```
rx pathid: 0, tx pathid: 0x0
```

```
R6#show bgp vpnv6 unicast vrf BGP ::192:104:5:5/128
BGP routing table entry for [214:104]::192:104:5:5/128, version 1324
Paths: (1 available, best #1, table BGP)
```

```
Not advertised to any peer
```

```
Refresh Epoch 1
```

```
214 214
```

```
FD00:192:4:104::4 (FE80::4) (via vrf BGP) from FD00:192:4:104::4
(214.0.0.4)
```

```
Origin incomplete, localpref 100, valid, external, best
```

```
rx pathid: 0, tx pathid: 0x0
```

A quick ping check verifies VPN connectivity to all three remote sites. We can reasonably assume that all of the BGP customers can communicate now.

```
R6#ping vrf BGP 192.104.13.13 source 192.104.6.6
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.104.13.13, timeout is 2 seconds:
Packet sent with a source address of 192.104.6.6
```

```
!!!!!!
```

Success rate is 100 percent (5/5), round-trip min/avg/max = 6/7/9 ms

```
R6#ping vrf BGP ::192:104:5:5 source ::192:104:6:6
```

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to ::192:104:5:5, timeout is 2 seconds:

Packet sent with a source address of ::192:104:6:6%BGP

!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 6/7/12 ms

```
R6#ping vrf BGP ::192:104:14:14 source ::192:104:6:6
```

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to ::192:104:14:14, timeout is 2 seconds:

Packet sent with a source address of ::192:104:6:6%BGP

!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 5/8/11 ms

Next, we will bring up the backdoor links between CSR6/XRv4 and CSR5/XRv3. We quickly check XRv3 and XRv4 to ensure they see CSR6 and CSR5 as BGP peers for IPv4 and IPv6, respectively. These sessions were disabled in previous verifications.

```
RP/0/0/CPU0:XRv3#show bgp vrf BGP ipv4 unicast summary | begin Neighbor
Neighbor      Spk    AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
192.5.104.5    0    104    120     91      104    0    0 00:01:03      8
192.11.104.11  0    214   7306   7275      104    0    0 23:15:42      5
```

```
RP/0/0/CPU0:XRv3#show bgp vrf BGP ipv6 unicast summary | begin Neighbor
Neighbor      Spk    AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
192.11.104.11  0    214   7306   7275      103    0    0 23:15:46      5
fd00:192:5:104::5
                0    104    122     92      103    0    0 00:01:22      8
```

```
RP/0/0/CPU0:XRv4#show bgp vrf BGP ipv4 unicast summary | begin Neighbor
Neighbor      Spk    AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
192.3.104.3    0    214   7820   7091      112    0    0 20:37:18      5
192.6.104.6    0    104    118     90      112    0    0 00:00:21      8
```

```
RP/0/0/CPU0:XRv4#show bgp vrf BGP ipv6 unicast summary | begin Neighbor
Neighbor      Spk    AS  MsgRcvd  MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
fd00:192:3:104::3
                0    214   7759   7041      71    0    0 20:37:18      5
fd00:192:6:104::6
                0    104    116     88      71    0    0 00:00:17      8
```

SoO works similarly in BGP as it does in EIGRP. Since we are bending the eBGP loop-prevention rules, there are cases where AS 104 routes don't have AS 104 in the AS path, and may bounce around inside AS 104 freely. They may even become best-paths, which can cause suboptimal traffic patterns or loops. The simplest SoO deployment is having the PEs identify the PEs are being in the same site. For example, assuming CSR5 and XRv4 were located together, there may never be a case where traffic between these routers should transit the WAN. To implement this, we can configure XRv2 and XRv1 to identify a

common SoO for this site and apply it to the CE routers. We can apply it directly to a neighbor under a specific AFI or use an RPL for additional granularity (per-prefix matches, etc). I demonstrate both methods below.

```
! XRv1
router bgp 214
  vrf BGP
    neighbor 192.11.104.13
      remote-as 104
      address-family ipv4 unicast
        site-of-origin 214:513

! XRv2
route-policy RPL_FROM_CSR5
  set extcommunity soo (214:513)
end-policy

router bgp 214
  vrf BGP
    neighbor 192.12.104.5
      remote-as 104
      address-family ipv4 unicast
        route-policy RPL_FROM_CSR5 in
```

After applying these configurations and refreshing the BGP routes, we notice that the MPLS network cannot be used for transit between CSR5 and XRv3 anymore. This would be ideal for multi-homed scenarios where east-west traffic should be LAN-based (backdoor link only), even in failure cases. A quick check shows that the two routers are exchanging loopbacks via iBGP, which are both best-paths.

```
R5#show bgp vpnv4 unicast vrf BGP 192.104.13.13/32
BGP routing table entry for 214:104:192.104.13.13/32, version 10
Paths: (1 available, best #1, table BGP)
  Advertised to update-groups:
    18
  Refresh Epoch 1
  Local
    192.5.104.13 (via vrf BGP) from 192.5.104.13 (13.13.13.13)
      Origin incomplete, metric 0, localpref 100, valid, internal, best,
      rx pathid: 0, tx pathid: 0x0

RP/0/0/CPU0:XRv3#show bgp vpnv4 unicast vrf BGP 192.104.5.5/32 | begin Local$
  Local
    192.5.104.5 from 192.5.104.5 (5.5.5.5)
      Origin incomplete, metric 0, localpref 100, valid, internal, best,
      group-best, import-candidate
      Received Path ID 0, Local Path ID 1, version 118
```


As a PE, XRv2 sees both paths, each of which originates from AS 104. Looking at XRv3's loopback, it is received from XRv1 (iBGP over MPLS) and from CSR5 (eBGP from customer). Hot-potato routing shows that the eBGP path is preferred as best, so XRv2 will not advertise the route from XRv3 any further.

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf BGP 192.104.13.13/32 | begin
Paths
Paths: (2 available, best #1)
  Advertised to update-groups (with more than one peer):
    0.2
  Path #1: Received by speaker 0
  Advertised to update-groups (with more than one peer):
    0.2
  104
    192.12.104.5 from 192.12.104.5 (5.5.5.5)
      Origin incomplete, localpref 100, valid, external, best, group-best,
import-candidate, import suspect
      Received Path ID 0, Local Path ID 1, version 1143
      Extended community: SoO:214:513 RT:214:104
  Path #2: Received by speaker 0
  Not advertised to any peer
  104, (Received from a RR-client)
    214.0.0.11 (metric 10) from 214.0.0.11 (214.0.0.11)
      Received Label 91013
      Origin incomplete, metric 0, localpref 100, valid, internal, import-
candidate, imported, import suspect
      Received Path ID 0, Local Path ID 0, version 0
      Extended community: SoO:214:513 RT:214:104
      Source VRF: BGP, Source Route Distinguisher: 214:104
```

As a side note, it isn't a good idea to make a PE the RR. Because XRv2 has selected the path through CSR5 as best, other nodes will route through CSR5 to reach XRv3 when transiting XRv1 would have been better. This may cause suboptimal routing depending on the bandwidth of the links, administrative policies, etc. SoO does not aim to fix this specifically.

```
R6#traceroute vrf BGP 192.104.13.13 source 192.104.6.6
Type escape sequence to abort.
Tracing the route to 192.104.13.13
VRF info: (vrf in name/id, vrf out name/id)
 1 192.4.104.4 14 msec 4 msec 2 msec
 2 214.4.8.8 [MPLS: Labels 8003/92031 Exp 0] 6 msec 7 msec 9 msec
 3 214.8.12.12 [MPLS: Label 92031 Exp 0] 20 msec 20 msec 20 msec
 4 192.12.104.5 20 msec 12 msec 11 msec
 5 192.5.104.13 91 msec 6 msec 6 msec
```

We will continue testing BGP with SoO. If we break the BGP connection between CSR5 and XRv3, the two routers will no longer be able to communicate. Now, XRv2 selects XRv1 as the best-path to XRv3's

loopback, but should not advertise it to CSR5 due to SoO restrictions. However, XRv2 still is advertising it to CSR5 despite SoO. This appears incorrect.

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf BGP 192.104.13.13/32 | begin
Paths
Paths: (1 available, best #1)
  Advertised to update-groups (with more than one peer):
    0.2
  Path #1: Received by speaker 0
  Advertised to update-groups (with more than one peer):
    0.2
  104, (Received from a RR-client)
    214.0.0.11 (metric 10) from 214.0.0.11 (214.0.0.11)
      Received Label 91013
      Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, imported
      Received Path ID 0, Local Path ID 1, version 1177
      Extended community: SoO:214:513 RT:214:104
      Source VRF: BGP, Source Route Distinguisher: 214:104
```

```
RP/0/0/CPU0:XRv2#show bgp vrf BGP ipv4 unicast neighbors 192.12.104.5
advertised-routes
Network                Next Hop          From              AS Path
Route Distinguisher: 214:104 (default for vrf BGP)
192.3.104.0/24         192.12.104.12    214.0.0.4         214 104?
192.4.104.0/24         192.12.104.12    214.0.0.4         214 104?
192.6.104.0/24         192.12.104.12    214.0.0.4         214 104?
192.11.104.0/24        192.12.104.12    214.0.0.11        214 104?
192.104.6.6/32         192.12.104.12    214.0.0.4         214 104?
192.104.13.13/32       192.12.104.12    214.0.0.11        214 104?
192.104.14.14/32       192.12.104.12    214.0.0.4         214 104?
```

A similar check on XRv1 yields better results. XRv1 does not advertise CSR5's loopback to XRv3, as expected. Two-way connectivity is therefore broken, but CSR5 should not have received the routes from XRv2 in the first place.

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast vrf BGP 192.104.5.5/32
[snip]
104
  214.0.0.12 (metric 10) from 214.0.0.12 (214.0.0.12)
    Received Label 92014
    Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, imported
    Received Path ID 0, Local Path ID 1, version 1248
    Extended community: SoO:214:513 RT:214:104
    Source VRF: BGP, Source Route Distinguisher: 214:104
```

```
RP/0/0/CPU0:XRv1#show bgp vrf BGP ipv4 unicast neighbors 192.11.104.13
advertised-routes
Network                Next Hop                From                AS Path
Route Distinguisher: 214:104 (default for vrf BGP)
192.3.104.0/24          192.11.104.11          214.0.0.3           214 104?
192.4.104.0/24          192.11.104.11          214.0.0.3           214 104?
192.6.104.0/24          192.11.104.11          214.0.0.3           214 104?
192.104.6.6/32          192.11.104.11          214.0.0.3           214 104?
192.104.14.14/32        192.11.104.11          214.0.0.3           214 104?
```

The issue with using an RPL to set SoO on ingress is that the automatic filtering that SoO provides is then lost. We would need to manually match the SoO in an outbound RPL for that peer to drop the routes. This is a design consideration of SoO by making it more configuration intensive at the cost of having additional granularity. On XRv1, specifying a neighbor-level SoO automatically works bidirectionally, so XRv1's configuration is complete.

```
! XRv2
route-policy RPL_TO_CSR5
  if extcommunity soo matches-every (214:513) then
    drop
  else
    pass
  endif
end-policy

router bgp 214
vrf BGP
  address-family ipv4 unicast
  neighbor 192.12.104.5
  address-family ipv4 unicast
  route-policy RPL_TO_CSR5 out
```

After applying this new policy, XRv2 cannot advertise XRv3's loopback to CSR5. Neither CSR5 nor XRv3 have routes to one another's loopback, which is expected when the backdoor is down.

```
RP/0/0/CPU0:XRv2#show bgp vrf BGP ipv4 unicast neighbors 192.12.104.5
advertised-routes
Network                Next Hop                From                AS Path
Route Distinguisher: 214:104 (default for vrf BGP)
192.3.104.0/24          192.12.104.12          214.0.0.4           214 104?
192.4.104.0/24          192.12.104.12          214.0.0.4           214 104?
192.6.104.0/24          192.12.104.12          214.0.0.4           214 104?
192.104.6.6/32          192.12.104.12          214.0.0.4           214 104?
192.104.14.14/32        192.12.104.12          214.0.0.4           214 104?
```

```
R5#show bgp vpnv4 unicast vrf BGP 192.104.13.13/32
% Network not in table
```

```
RP/0/0/CPU0:XRv3#show bgp vrf BGP ipv4 unicast 192.104.5.5/32
% Network not in table
```

Notice that IPv6 connectivity is totally unaffected when the backdoor is down. SoO is enabled per AFI, unlike EIGRP, so we can have different “site-based” topologies for IPv4 and IPv6. Both CSR5 and XRv3 have IPv6 routes towards one another’s loopbacks.

```
R5#show bgp vpnv6 unicast vrf BGP ::192:104:13:13/128
BGP routing table entry for [214:104]::192:104:13:13/128, version 1737
Paths: (1 available, best #1, table BGP)
  Not advertised to any peer
  Refresh Epoch 1
  214 104
    FD00:192:12:104::12 (FE80::12) (via vrf BGP) from FD00:192:12:104::12
    (214.0.0.12)
      Origin incomplete, localpref 100, valid, external, best
      rx pathid: 0, tx pathid: 0x0
```

```
RP/0/0/CPU0:XRv3#show bgp vrf BGP ipv6 unicast ::192:104:5:5/128 | begin
Paths
Paths: (1 available, best #1)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  214 214
    fd00:192:11:104::11 from 192.11.104.11 (214.0.0.11)
      Origin incomplete, localpref 100, valid, external, best, group-best,
import-candidate
      Received Path ID 0, Local Path ID 1, version 126
```

We can also use an approach similar to that tested in the EIGRP section, whereby a site “guards” itself with the same SoO around the perimeter routers to prevent routes from that site entering again, be it via the SP network or backdoor links. On the left side of the network, we will first begin by enabling SoO on the PE devices. CSR6 and XRv4 will be considered in different “sites” so that they can talk over MPLS or the backdoor. We will use the per-neighbor method on CSR3 and the route-map method on CSR4. We also advertise the PE-CE link into BGP so we can test the feature as we did for EIGRP.

```
! CSR3
router bgp 214
  address-family ipv4 vrf BGP
    network 192.3.104.0
    neighbor 192.3.104.14 soo 214:3
  address-family ipv6 vrf BGP
    network FD00:192:3:104::/64
    neighbor FD00:192:3:104::14 soo 214:3
```

```

! CSR4
route-map RM_SOO_IN permit 10
  set extcommunity soo 214:4

address-family ipv4 vrf BGP
  network 192.4.104.0
  neighbor 192.4.104.6 route-map RM_SOO_IN in
address-family ipv6 vrf BGP
  network FD00:192:4:104::/64
  neighbor FD00:192:4:104::6 route-map RM_SOO_IN in

```

Unlike XR, which has no command to verify per-neighbor SoO, XE will show this under the neighbor details. As expected, when using the route-map, it will not.

```

R3#show bgp vpnv4 unicast vrf BGP neighbors 192.3.104.14 | include v4|Origin
  Address family IPv4 Unicast: advertised and received
  For address family: VPNv4 Unicast
  Translates address family IPv4 Unicast for VRF BGP
  neighbor Site-of-Origin is SoO:214:3

```

```

R3#show bgp vpnv6 unicast vrf BGP neighbors fd00:192:3:104::14 | include
v6|Origin
  Address family IPv6 Unicast: advertised and received
  For address family: VPNv6 Unicast
  Translates address family IPv6 Unicast for VRF BGP
  neighbor Site-of-Origin is SoO:214:3

```

Looking at CSR3's view of the connected PE-CE link between itself and XRv4, we can see both SoO impositions are not quite working. The remotely learned route from the eBGP peer (XRv4) has the proper SoO applied, but the connected route via the network statement does not. Unlike EIGRP, SoO is being applied to routes learned from a peer, not routes recursing out of a given interface. BGP has no concept of interfaces.

```

R3#show bgp vpnv6 unicast vrf BGP FD00:192:3:104::/64
BGP routing table entry for [214:104]FD00:192:3:104::/64, version 84
Paths: (2 available, best #2, table BGP)
  Advertised to update-groups:
    12          11
  Refresh Epoch 1
  104
  FD00:192:3:104::14 (FE80::14) (via vrf BGP) from FD00:192:3:104::14
  (14.14.14.14)
    Origin incomplete, metric 0, localpref 100, valid, external
    Extended Community: SoO:214:3 RT:214:104
    mpls labels in/out 3045/nolabel
    rx pathid: 0, tx pathid: 0
  Refresh Epoch 1

```

```

Local
:: (via vrf BGP) from 0.0.0.0 (214.0.0.3)
  Origin IGP, metric 0, localpref 100, weight 32768, valid, sourced,
local, best
  Extended Community: RT:214:104
  mpls labels in/out 3045/nolabel(BGP)
  rx pathid: 0, tx pathid: 0x0

```

Applying a route-map to the BGP network statement does not work, although one would think that it would manually adjust the attributes of the prefix. Instead, we can use the interface-level command that is protocol-agnostic to solve this problem. For a quick side-demo, we will set a bogus SoO just to see if it accidentally overrides the BGP-level neighbor one for routes learned via that interface. Fortunately, it does not. We know we will have the exact same problem on CSR4, so we will correct the problem there also.

```

! CSR3 and CSR4
route-map RM_SOO_IN permit 10
  set extcommunity soo 214:999

interface GigabitEthernet2.104
  ip vrf sitemap RM_SOO_IN

R3#show bgp vpnv6 unicast vrf BGP FD00:192:3:104::/64
[snip]
  Local
    :: (via vrf BGP) from 0.0.0.0 (214.0.0.3)
      Origin IGP, metric 0, localpref 100, weight 32768, valid, sourced,
local, best
      Extended Community: SoO:214:999 RT:214:104
      mpls labels in/out 3011/nolabel(BGP)
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  104
  FD00:192:3:104::14 (FE80::14) (via vrf BGP) from FD00:192:3:104::14
(14.14.14.14)
    Origin incomplete, metric 0, localpref 100, valid, external
    Extended Community: SoO:214:3 RT:214:104
    mpls labels in/out 3011/nolabel
    rx pathid: 0, tx pathid: 0

```

We will change the SoO back to 214:3 as expected (not shown) and verify it quickly. Now, the PE-CE link and the eBGP-learned route have the proper SoO.

```

R3#show bgp vpnv6 unicast vrf BGP FD00:192:3:104::/64 | include from|SoO
:: (via vrf BGP) from 0.0.0.0 (214.0.0.3)
  Extended Community: SoO:214:3 RT:214:104

```

```
FD00:192:3:104::14 (FE80::14) (via vrf BGP) from FD00:192:3:104::14
(14.14.14.14)
Extended Community: SoO:214:3 RT:214:104
```

Since we know the locally-originated route is the best-path, CSR3 can advertise this to CSR4. It carries the SoO attribute. CSR4 learns 3 routes; one directly from CSR3 (RR), another reflected from XRv2 (RR), and one from the eBGP peer CSR6. The best-path is the one from CSR3 since the peer ID 214.0.0.3 is less than 214.0.0.12 (worst tiebreaker). This is because the BGP RID of 214.0.0.3 is compared against the originator ID of the same value from XRv2, which results in a tie. The AS path length is 0 which immediately disqualifies the eBGP path.

```
R4#show bgp vpnv6 unicast vrf BGP fd00:192:3:104::/64
BGP routing table entry for [214:104]FD00:192:3:104::/64, version 2317
BGP Bestpath: cost-community-ignore
Paths: (3 available, best #2, table BGP)
  Advertised to update-groups:
    5
  Refresh Epoch 1
  Local
    ::FFFF:214.0.0.3 (metric 10) (via default) from 214.0.0.12 (214.0.0.12)
      Origin IGP, metric 0, localpref 100, valid, internal
      Extended Community: SoO:214:3 RT:214:104
      Originator: 214.0.0.3, Cluster list: 0.0.3.12
      mpls labels in/out nlabel/3011
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 3
  Local
    ::FFFF:214.0.0.3 (metric 10) (via default) from 214.0.0.3 (214.0.0.3)
      Origin IGP, metric 0, localpref 100, valid, internal, best
      Extended Community: SoO:214:3 RT:214:104
      mpls labels in/out nlabel/3011
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 2
  104
    FD00:192:4:104::6 (FE80::6) (via vrf BGP) from FD00:192:4:104::6
    (6.6.6.6)
      Origin incomplete, localpref 100, valid, external
      Extended Community: SoO:214:4 RT:214:104
      rx pathid: 0, tx pathid: 0
```

CSR4 then advertises this to CSR6, who should ideally see SoO:214:3. CSR6 doesn't see any SoO information on the route received from CSR4. Recall that XE does not send communities by default, and XR does not send communities to eBGP neighbors by default.

```
R6#show bgp vpnv6 unicast vrf BGP fd00:192:3:104::/64
BGP routing table entry for [214:104]FD00:192:3:104::/64, version 1587
Paths: (2 available, best #2, table BGP)
```

```

Advertised to update-groups:
  1
Refresh Epoch 1
214
  FD00:192:4:104::4 (FE80::4) (via vrf BGP) from FD00:192:4:104::4
(214.0.0.4)
  Origin IGP, localpref 100, valid, external
  rx pathid: 0, tx pathid: 0
Refresh Epoch 1
Local
  FD00:192:6:104::14 (via vrf BGP) from FD00:192:6:104::14 (14.14.14.14)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  rx pathid: 0, tx pathid: 0x0

```

We will quickly enable the sending of extended communities to eBGP peers on CSR3 and CSR4.

```

! CSR3
router bgp 214
  address-family ipv6 vrf BGP
    neighbor FD00:192:3:104::14 send-community extended

! CSR4
router bgp 214
  address-family ipv6 vrf BGP
    neighbor FD00:192:4:104::6 send-community extended

```

Now, we can see the SoO on CSR6 as expected. We also check the PE-CE link between CSR4/CSR6 on XRv4; it can see SoO:214:4 as imposed by CSR4. These connected-route impositions required the interface-level site-map command since the BGP-level SoO configuration does not affect locally-originated routes.

```

R6#show bgp vpnv6 unicast vrf BGP fd00:192:3:104::/64
BGP routing table entry for [214:104]FD00:192:3:104::/64, version 1587
Paths: (2 available, best #2, table BGP)
  Advertised to update-groups:
    1
Refresh Epoch 1
214
  FD00:192:4:104::4 (FE80::4) (via vrf BGP) from FD00:192:4:104::4
(214.0.0.4)
  Origin IGP, localpref 100, valid, external
  Extended Community: SoO:214:3
  rx pathid: 0, tx pathid: 0
Refresh Epoch 1
Local
  FD00:192:6:104::14 (via vrf BGP) from FD00:192:6:104::14 (14.14.14.14)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  rx pathid: 0, tx pathid: 0x0

```



```

RP/0/0/CPU0:XRv4#show bgp vrf BGP ipv6 unicast fd00:192:4:104::/64 | begin
Paths
Paths: (2 available, best #2)
  Advertised to CE peers (in unique update groups):
    fd00:192:3:104::3
  Path #1: Received by speaker 0
  Not advertised to any peer
  214
    fd00:192:3:104::3 from fd00:192:3:104::3 (214.0.0.3)
      Origin IGP, localpref 100, valid, external, group-best, import suspect
      Received Path ID 0, Local Path ID 0, version 0
      Extended community: SoO:214:4
  Path #2: Received by speaker 0
  Advertised to CE peers (in unique update groups):
    fd00:192:3:104::3
  Local
    fd00:192:6:104::6 from fd00:192:6:104::6 (6.6.6.6)
      Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, import suspect
      Received Path ID 0, Local Path ID 1, version 71

```

Because those connected routes were technically originated in AS 214, the AS path check will pass without needing “allowas-in” or “as-override”. Shutting down CSR6’s PE-CE link proves that CSR6 can still learn that connected route with SoO:214:4 from XRv4. Normally, this is the situation we are trying to prevent, since CSR4 is effectively learning a route from its local “site”.

```

R6#show bgp vpnv6 unicast vrf BGP fd00:192:4:104::/64
BGP routing table entry for [214:104]FD00:192:4:104::/64, version 1656
Paths: (1 available, best #1, table BGP)
  Not advertised to any peer
  Refresh Epoch 1
  214
    FD00:192:6:104::14 (via vrf BGP) from FD00:192:6:104::14 (14.14.14.14)
      Origin IGP, localpref 100, valid, internal, best
      Extended Community: SoO:214:4
      rx pathid: 0, tx pathid: 0x0

```

Next, we need to configure the SoO on the backdoor peerings for IPv6 at a minimum, as well as ensure extended-communities are being exchanged more liberally since SoO is being actively used by PE and CE routers. CSR6 will identify as being in site 4 while XRv4 will be in site 3. XR automatically sends communities inside of iBGP, so it only needs to be explicitly enabled with the eBGP PE.

```

! XRv4
router bgp 214
  vrf BGP
    neighbor fd00:192:3:104::3

```

```

    remote-as 214
    address-family ipv6 unicast
        send-extended-community-ebgp
neighbor fd00:192:6:104::6
    remote-as 104
    address-family ipv6 unicast
        site-of-origin 214:3

! CSR6
router bgp 104
    address-family ipv6 vrf BGP
        neighbor FD00:192:4:104::4 send-community extended
        neighbor FD00:192:6:104::14 send-community extended
        neighbor FD00:192:6:104::14 soo 214:4

```

XR doesn't allow this to be committed and claims the SoO is an eBGP-only configuration. However, XE takes the code, so we will attempt it. According to this, the third SoO technique is not supported on XR.

```

! XRv4
neighbor fd00:192:6:104::6
    address-family ipv6 unicast
        site-of-origin 214:3
!!% Change would result in internal neighbor (fd00:192:6:104::6 VRF BGP) with
external-only config

```

We verify that the configuration was successful on CSR6. Despite multiple BGP hard resets, CSR6 still learns the route regardless of whether the BGP SoO is set or not. XR warned us that this wasn't supported, but XE seems to accept the configuration and gives no indication of a problem, either in show or debug commands. This suggests that the "perimeter" technique is not supported on either platform and is likely specific to EIGRP deployments.

```

R6#show bgp vpnv6 unicast vrf BGP neighbors FD00:192:6:104::14 | include
v6|Origin
    Address family IPv6 Unicast: advertised and received
    For address family: VPNv6 Unicast
    Translates address family IPv6 Unicast for VRF BGP
    neighbor Site-of-Origin is SoO:214:4

```

```

R6#show bgp vpnv6 unicast vrf BGP fd00:192:4:104::/64
BGP routing table entry for [214:104]FD00:192:4:104::/64, version 7
Paths: (1 available, best #1, table BGP)
    Not advertised to any peer
    Refresh Epoch 1
    214
    FD00:192:6:104::14 (via vrf BGP) from FD00:192:6:104::14 (14.14.14.14)
        Origin IGP, localpref 100, valid, internal, best
        Extended Community: SoO:214:4

```

```
rx pathid: 0, tx pathid: 0x0
```

In summary, it appears that BGP SoO is best used in the strict sense of identifying sites so that intra-site traffic does not transit the SP network. Usage of the site-map command is very rare with BGP and was used for demonstration purposes only, however using it in this way does speak to its flexibility. There is another handy use for SoO which can be used to reduce suboptimal routing. Considering, at present, routes learned via CSR3 have SoO:214:3 and routes via CSR4 have SoO:214:4, we have a “tag” on routes based on which PE they were learned from. When all links in the network are up, CSR4 learns routes behind the remote CE (the one to which it is not connected) via eBGP. This means that traffic routed to CSR4 destined for networks behind XRv4 will transit the (possibly) slow backdoor link. It may be desirable for CSR4 to prefer the iBGP route via CSR3 because that PE-CE link (CSR3/XRv4) is much faster, where CSR4 is just a backup. Following this scenario, we can adjust the best-path selection algorithm based on prefixes to achieve this. Matching on SoO is a simple and dynamic way, especially if it is already being used for loop control in other parts of the network.

```
R4#show bgp vpnv6 unicast vrf BGP ::192:104:14:14/128
BGP routing table entry for [214:104]::192:104:14:14/128, version 2350
BGP Bestpath: cost-community-ignore
Paths: (2 available, best #1, table BGP)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  104
    FD00:192:4:104::6 (FE80::6) (via vrf BGP) from FD00:192:4:104::6
    (6.6.6.6)
      Origin incomplete, localpref 100, valid, external, best
      Extended Community: SoO:214:4 RT:214:104
      mpls labels in/out 4055/nolabel
      rx pathid: 0, tx pathid: 0x0
    Refresh Epoch 5
    104
      ::FFFF:214.0.0.3 (metric 10) (via default) from 214.0.0.3 (214.0.0.3)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Extended Community: SoO:214:3 RT:214:104
      mpls labels in/out 4055/3009
      rx pathid: 0, tx pathid: 0
```

Since the BGP peer over iBGP is VPNv6, and the eBGP peer is IPv6 unicast inside of a VRF, we cannot just apply a route-map to the VPNv6 peer if other VRFs are using the same SoO. For example, if SoO:214:3 is being used by VRF EIGRP, for example, the route-map could have unintended effects. In this case, we are safe, but it is a future consideration. The route-map effectively says “routes from site 3 should be preferred via CSR3”, which is a very logical decision.

```
! CSR4
ip extcommunity-list standard EXTCOML_SOO_3 permit soo 214:3
```

```

route-map RM_BGP_CSR3_IN permit 10
  match extcommunity EXTCOML_SOO_3
  set local-preference 200
route-map RM_BGP_CSR3_IN permit 1000

router bgp 214
  address-family vpnv6
    neighbor 214.0.0.3 route-map RM_BGP_CSR3_IN in

```

After a soft reset, we verify the local preference update was successful. Now CSR4 prefers the path over MPLS; we verify with traceroute.

```

R4#show bgp vpnv6 unicast vrf BGP ::192:104:14:14/128
[snip]
104
  ::FFFF:214.0.0.3 (metric 10) (via default) from 214.0.0.12 (214.0.0.12)
    Origin incomplete, metric 0, localpref 100, valid, internal
    Extended Community: SoO:214:3 RT:214:104
    Originator: 214.0.0.3, Cluster list: 0.0.3.12
    mpls labels in/out nolabel/3009
    rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
104
  FD00:192:4:104::6 (FE80::6) (via vrf BGP) from FD00:192:4:104::6
  (6.6.6.6)
    Origin incomplete, localpref 100, valid, external
    Extended Community: SoO:214:4 RT:214:104
    rx pathid: 0, tx pathid: 0
  Refresh Epoch 6
104
  ::FFFF:214.0.0.3 (metric 10) (via default) from 214.0.0.3 (214.0.0.3)
    Origin incomplete, metric 0, localpref 200, valid, internal, best
    Extended Community: SoO:214:3 RT:214:104
    mpls labels in/out nolabel/3009
    rx pathid: 0, tx pathid: 0x0

R4#traceroute vrf BGP ipv6 ::192:104:14:14
Type escape sequence to abort.
Tracing the route to ::192:104:14:14
 1 FD00:192:3:104::3 [MPLS: Label 3009 Exp 0] 4 msec 2 msec 3 msec
 2 ::192:104:14:14 [AS 104] 4 msec 4 msec 3 msec

```

The downside of this approach is that now CSR4 will prefer CSR3 for all traffic going into AS 104. Adding some kind of non-overlapping attributes (possibly prefixes, RTs, other custom communities, etc) to the route-map match clauses can add granularity to this routing enhancement. Applying this policy everywhere would make sense, but I demonstrate it only on CSR4 for brevity.

```

R4#show bgp vpnv6 unicast vrf BGP ::192:104:6:6/128
[snip]
104
::FFFF:214.0.0.3 (metric 10) (via default) from 214.0.0.3 (214.0.0.3)
Origin incomplete, metric 0, localpref 200, valid, internal, best
Extended Community: SoO:214:3 RT:214:104
mpls labels in/out nolabel/3016
rx pathid: 0, tx pathid: 0x0

R4#traceroute vrf BGP ipv6 ::192:104:6:6
Type escape sequence to abort.
Tracing the route to ::192:104:6:6
 1 FD00:192:3:104::3 [MPLS: Label 3016 Exp 0] 5 msec 4 msec 2 msec
 2 FD00:192:3:104::14 4 msec 4 msec 3 msec
 3 FD00:192:6:104::6 [AS 104] 5 msec 5 msec 5 msec

```

The reachability script is below. The only failure will be between CSR5 and XRv3 since the backdoor is down and their PEs assert that those routers belong to the same “site”. This was to demonstrate SoO specifically. Bringing it up yields full reachability.

```

! CSR5 and CSR6
tclsh
foreach x {
192.104.5.5
192.104.6.6
192.104.13.13
192.104.14.14
::192:104:5:5
::192:104:6:6
::192:104:13:13
::192:104:14:14
} { ping vrf BGP $x source loopback104 repeat 3 timeout 1 }

```

38.1.6 Static routing

Static routing is another possible way to achieve PE-CE routing. Most service providers will offer either BGP or static routing as options due to their widespread support and simplicity. Looking at CSR4 and XRv2, all we have to do is add a VRF-aware static route and redistribute it into BGP per AFI.

```

! CSR4
ip route vrf STATIC 192.105.6.6 255.255.255.255 Gig2.105 192.4.105.6
ipv6 route vrf STATIC ::192:105:6:6/128 Gig2.105 FD00:192:4:105::6

router bgp 214
 address-family ipv4 vrf STATIC
   redistribute static
 address-family ipv6 vrf STATIC
   redistribute static

```

```

! XRv2
router static
  vrf STATIC
    address-family ipv4 unicast
      192.105.5.5/32 GigabitEthernet0/0/0/0.105 192.12.105.5
    address-family ipv6 unicast
      ::192:105:5:5/128 GigabitEthernet0/0/0/0.105 fd00:192:12:105::5

router bgp 214
  vrf STATIC
    rd 214:105
    address-family ipv4 unicast
      redistribute static
    address-family ipv6 unicast
      redistribute static

```

Using some lesser-known static routing show commands, we can verify the static routes inside of the VRF, per AFI.

```

RP/0/0/CPU0:XRv2#show static vrf STATIC ipv4 unicast topology
VRF: STATIC Table Id: 0xe0000018 AFI: IPv4 SAFI: Unicast
Prefix/Len      Interface      Nexthop      Object      Metrics
192.105.5.5/32  GigabitEthernet0_0_0_0.105 192.12.105.5  None        [0/0/1/0]

RP/0/0/CPU0:XRv2#show static vrf STATIC ipv6 unicast topology
VRF: STATIC Table Id: 0xe0800018 AFI: IPv6 SAFI: Unicast
Prefix/Len      Interface      Nexthop      Object      Metrics
::192:105:5:5/128  GigabitEthernet0_0_0_0.105 fd00:192:12:105::5  None        [0/0/1/0]

R4#show ip static route vrf STATIC
Codes: M - Manual static, A - AAA download, N - IP NAT, D - DHCP,
      [snip]
Codes in []: A - active, N - non-active, B - BFD-tracked, D - Not Tracked, P
- permanent
Static local RIB for STATIC
M 192.105.6.6/32 [1/0] via GigabitEthernet2.105 192.4.105.6 [A]

R4#show ipv6 static vrf STATIC
IPv6 Static routes Table - STATIC
Codes: * - installed in RIB, u/m - Unicast/Multicast only
      [snip]
* ::192:105:6:6/128 via FD00:192:4:105::6, GigabitEthernet2.105, distance 1

```

A quick check on XRv2 shows all of the remote static routes inside of BGP with the proper VPNv4 next hops for both AFIs. Based on this, we can assume all of the static-to-BGP redistribution was successful on all 4 PEs within VRF STATIC.

```

RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf STATIC | begin Network
Network      Next Hop      Metric LocPrf Weight Path

```

```

Route Distinguisher: 214:105 (default for vrf STATIC)
*> 192.105.5.5/32      192.12.105.5          0          32768 ?
*>i192.105.6.6/32     214.0.0.4              0      100      0 ?
*>i192.105.13.13/32   214.0.0.11             0      100      0 ?
*>i192.105.14.14/32   214.0.0.3              0      100      0 ?

RP/0/0/CPU0:XRv2#show bgp vpnv6 unicast vrf STATIC | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 214:105 (default for vrf STATIC)
*> ::192:105:5:5/128  fd00:192:12:105::5
                                0          32768 ?
*>i::192:105:6:6/128  214.0.0.4              0      100      0 ?
*>i::192:105:13:13/128
                        214.0.0.11             0      100      0 ?
*>i::192:105:14:14/128
                        214.0.0.3              0      100      0 ?

```

We also need to ensure the CE routers have static routes pointing towards the PE. A simple default for IPv4 and IPv6 would work, but we will use a coarse aggregate for variety. Only configurations on XRv3 and CSR5 are shown for brevity. Assuming this MPLS network was also the mechanism by which CEs reached the Internet, a default route is most appropriate. If not, a coarse aggregate encompassing all of the WAN subnets would work well, with a default pointing elsewhere towards the Internet.

```

! CSR6
ip route vrf STATIC 192.0.0.0 255.0.0.0 GigabitEthernet2.105 192.12.105.12
ipv6 route vrf STATIC ::192:0:0:0/80 GigabitEthernet2.105 FD00:192:12:105::12

! XRv3
router static
vrf STATIC
  address-family ipv4 unicast
    192.0.0.0/8 GigabitEthernet0/0/0/0.105 192.11.105.11
  address-family ipv6 unicast
    ::192:0:0:0/80 GigabitEthernet0/0/0/0.105 fd00:192:11:105::11

```

There isn't much more to test at this point, so we will verify reachability before continuing. Running this script from the XE CEs is a quick way to ensure all 4 sites in the STATIC VPN can communicate.

```

! CSR5 and CSR6
tclsh
foreach x {
192.105.5.5
192.105.6.6
192.105.13.13
192.105.14.14
::192:105:5:5
::192:105:6:6

```

```

::192:105:13:13
::192:105:14:14
} { ping vrf STATIC $x source loopback105 repeat 3 timeout 1 }

```

As a quick note, SoO can also apply to static routes provided they point traffic out of the interface to which the SoO is applied. XR does not support this, but XE does. This would only be useful if BGP was used somewhere in the VPN, as it was for IS-IS, which is where the filtering would happen. Even if we use the same SoO on CSR3 and CSR4, CSR6 and XRv4 will still be able to talk to one another. BGP will import the routes into the proper table, but since static routing is used, there is no possible way for the SoO to be “advertised” into the customer domain.

```

! CSR3 and CSR4
route-map RM_STATIC_SOO permit 10
  set extcommunity soo 214:105

interface GigabitEthernet2.105
  ip vrf sitemap RM_STATIC_SOO

```

We verify that SoO:214:105 has been applied by both CSR3 and CSR4 on their locally redistributed static routes. The information is exchanged, but no filtering occurs since there is nothing being advertised to the CE routers, as described above. Without a PE-CE protocol, the SoO could be used as a route-tag as seen in the BGP section for best-path adjustment, but in that case, using ordinary communities would be more appropriate.

```

R3#show bgp vpnv6 unicast vrf STATIC ::192:105:6:6/128
BGP routing table entry for [214:105]::192:105:6:6/128, version 411
Paths: (1 available, best #1, table STATIC)
  Advertised to update-groups:
    13
  Refresh Epoch 7
  Local, (Received from a RR-client)
    ::FFFF:214.0.0.4 (metric 10) (via default) from 214.0.0.4 (214.0.0.4)
    Origin incomplete, metric 0, localpref 100, valid, internal, best
    Extended Community: SoO:214:105 RT:214:105
    mpls labels in/out nolabel/4027
    rx pathid: 0, tx pathid: 0x0

```

```

R3#show bgp vpnv6 unicast vrf STATIC ::192:105:14:14/128
BGP routing table entry for [214:105]::192:105:14:14/128, version 410
Paths: (1 available, best #1, table STATIC)
  Advertised to update-groups:
    13
  Refresh Epoch 1
  Local
    FD00:192:3:105::14 (via vrf STATIC) from 0.0.0.0 (214.0.0.3)
    Origin incomplete, metric 0, localpref 100, weight 32768, valid,
    sourced, best

```



```
Extended Community: SoO:214:105 RT:214:105
mpls labels in/out 3005/nolabel
rx pathid: 0, tx pathid: 0x0
```

```
R6#ping vrf STATIC ::192:105:14:14 source ::192:105:6:6
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to ::192:105:14:14, timeout is 2 seconds:
Packet sent with a source address of ::192:105:6:6%STATIC
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 3/6/16 ms
```

38.1.7 RIP

RIP is a very uncommon PE-CE option but it is supported in XE and XR. XR does not support RIPng (IPv6) at all, and because of the high density of XR in the network, RIPng is not tested. Below are the PE configurations for RIPv2 on an XE and XR PE. When redistributing from BGP, we can set the metric to transparent which allows RIP to copy the BGP MED value when importing routes from BGP. We won't need this command in XR as it is automatic so long as we don't specify a hard metric.

```
! CSR4
router rip
  address-family ipv4 vrf RIP
    redistribute bgp 214 metric transparent
    network 192.4.106.0
    no auto-summary
    version 2

! XRv2
router rip
  vrf RIP
    interface GigabitEthernet0/0/0/0.106
    redistribute bgp 214
```

The CE configurations are nearly identical, except there is no BGP redistribution and RIP is enabled on the loopbacks/backdoors. The backdoors are currently shutdown.

```
! CSR6
router rip
  address-family ipv4 vrf RIP
    network 192.4.106.0
    network 192.6.106.0
    network 192.106.6.0
    no auto-summary
    version 2

! XRv3
router rip
  vrf RIP
```

```

interface Loopback106
  passive-interface
interface GigabitEthernet0/0/0/0.106
interface GigabitEthernet0/0/0/0.206

```

Checking XRv2, we can see all of the RIP loopbacks in BGP VPNv4, which means the RIP advertisement from CE to PE, and the redistribution from RIP, were successful. All of the PE-CE transit links are included by default as well.

```

RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf RIP | begin Network
  Network                Next Hop                Metric LocPrf Weight Path
Route Distinguisher: 214:106 (default for vrf RIP)
*>i192.3.106.0/24        214.0.0.3                0      100      0 ?
*>i192.4.106.0/24        214.0.0.4                0      100      0 ?
*>i192.11.106.0/24       214.0.0.11               0      100      0 ?
*> 192.12.106.0/24       0.0.0.0                  0              32768 ?
*> 192.106.5.5/32        192.12.106.5             1              32768 ?
*>i192.106.6.6/32        214.0.0.4                1      100      0 ?
*>i192.106.13.13/32      214.0.0.11               1      100      0 ?
*>i192.106.14.14/32      214.0.0.3                1      100      0 ?

```

Although unrelated to RIP, we will perform some cleanup by removing all of these connected routes. This is a good practice for EIGRP and OSPF also. Doing so is simple; a deny-all route-map/RPL applied to connected redistribution will take explicit control of the connected routes and deny them. Only true RIP-learned routes will be permitted since that redistribution is unfiltered. This solution scales better than a “white-list” prefix-filter under RIP since any RIP-learned prefix is now allowed by default.

```

! CSR3 and CSR4
route-map RM_DENY_ALL deny 10

router bgp 214
  address-family ipv4 vrf RIP
    redistribute connected route-map RM_DENY_ALL
    redistribute rip

! XRv1 and XRv2
route-policy RPL_DENY_ALL
  drop
end-policy

router bgp 214
  vrf RIP
    address-family ipv4 unicast
      redistribute connected route-policy RPL_DENY_ALL
      redistribute rip

```

Checking XRv2 again, all we see are the loopbacks without the PE-CE transit links, as expected.

```

RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf RIP | begin Network
  Network                Next Hop                Metric LocPrf Weight Path
Route Distinguisher: 214:106 (default for vrf RIP)
*> 192.106.5.5/32        192.12.106.5                1              32768 ?
*>i192.106.6.6/32        214.0.0.4                  1      100        0 ?
*>i192.106.13.13/32     214.0.0.11                 1      100        0 ?
*>i192.106.14.14/32     214.0.0.3                  1      100        0 ?

```

For additional complexity, we will open backdoor connections between XRv4/CSR6 and CSR5/XRv3. Because RIP is hop-count based, the CEs will prefer this backdoor over the MPLS network. Also note that our fancy “redistribute connected” filter is no longer effective since transit links can be learned via RIP now; we will ignore this as it is outside the scope of this test (we could easily filter it from RIP updates). We will pretend that CSR6 and XRv4 actually are in the same site and should never reach one another via MPLS. SoO seems like it might work here, so we will test it. We will re-use some old route-maps that have the same SoO from older testing.

```

! CSR3 and CSR4
interface GigabitEthernet2.106
 ip vrf sitemap RM_STATIC_SO

```

Checking CSR3, it has a locally-originated route (redistributed from RIP) and an iBGP learned one from CSR4. As expected, both of them have the same SoO.

```

R3#show bgp vpnv4 unicast vrf RIP 192.106.14.14/32
BGP routing table entry for 214:106:192.106.14.14/32, version 443
Paths: (2 available, best #2, table RIP)
  Advertised to update-groups:
    14
  Refresh Epoch 2
  Local, (Received from a RR-client)
    214.0.0.4 (metric 10) (via default) from 214.0.0.4 (214.0.0.4)
      Origin incomplete, metric 2, localpref 100, valid, internal
      Extended Community: SoO:214:105 RT:214:106
      mpls labels in/out 3065/4025
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  Local
    192.3.106.14 (via vrf RIP) from 0.0.0.0 (214.0.0.3)
      Origin incomplete, metric 1, localpref 100, weight 32768, valid,
sourced, best
      Extended Community: SoO:214:105 RT:214:106
      mpls labels in/out 3065/nolabel
      rx pathid: 0, tx pathid: 0x0

```

To speed up the convergence process, we can shut down the backdoor link on both CSR6 and XRv4 concurrently. SoO actually does work for RIP filtering; CSR6 and XRv4 do not have routes for one another's loopback anymore, despite the PEs having the routes in their local VPN tables. Debugging RIP on CSR3, we do not see CSR6's loopback being advertised to XRv4 at all. CSR3 did properly redistribute this route from BGP into RIP, as the database shows.

```
R3#show ip rip database vrf RIP 192.106.6.6 255.255.255.255
192.106.6.6/32      redistributed
    [2] via 214.0.0.4,
```

```
! CSR3
RIP: sending v2 update to 224.0.0.9 via GigabitEthernet2.106 (192.3.106.3)
RIP: build update entries
    192.5.106.0/24 via 0.0.0.0, metric 2, tag 0
    192.11.106.0/24 via 0.0.0.0, metric 3, tag 0
    192.12.106.0/24 via 0.0.0.0, metric 3, tag 0
    192.106.5.5/32 via 0.0.0.0, metric 2, tag 0
    192.106.13.13/32 via 0.0.0.0, metric 2, tag 0
```

```
RP/0/0/CPU0:XRv4#show route vrf RIP 192.106.6.6
% Network not in table
```

CSR5 and XRv3 will be in “different” sites but should prefer MPLS over the backdoor. We can use an offset list on the CE router backdoor interfaces to make this link less preferable. For example, CSR5 is currently preferring the backdoor link to reach XRv3's loopback.

```
R5#show ip route vrf RIP 192.106.13.13
Routing Table: RIP
Routing entry for 192.106.13.13/32
    Known via "rip", distance 120, metric 1
    Redistributing via rip
    Last update from 192.5.106.13 on GigabitEthernet2.206, 00:00:19 ago
    Routing Descriptor Blocks:
    * 192.5.106.13, from 192.5.106.13, 00:00:19 ago, via GigabitEthernet2.206
        Route metric is 1, traffic share count is 1
```

I am surprised to see that XR even has a site-of-origin command for RIP. We will use another traditional multi-site SoO design on these routers where each PE router has a different SoO. This means that a routing update loop will stop at the originating PE if the route manages to come “full circle”, but this assumes RIP can somehow carry SoO (which it does not appear to). There is no backdoor filtering. This design would be more relevant for EIGRP than anything else since EIGRP can carry SoO.

```
! CSR5
router rip
    address-family ipv4 vrf RIP
        offset-list 0 in 8 GigabitEthernet2.206
```

```

! XRv3
route-policy RPL_ADD_8
  add rip-metric 8
end-policy

router rip
vrf RIP
  interface GigabitEthernet0/0/0/0.206
    route-policy RPL_ADD_8 in

```

On the PEs, we need to configure the SoO on the PE-CE link as well. This completes the SoO design.

```

! XRv1
router rip
vrf RIP
  interface GigabitEthernet0/0/0/0.106
    site-of-origin 214:9013

! XRv2
router rip
vrf RIP
  interface GigabitEthernet0/0/0/0.106
    site-of-origin 214:9005

```

Verifying some BGP routes on XRv2, we can see the loopback CSR5 with SoO values of 214:9013 and 214:9005, respectively. One was locally originated and one was iBGP-learned. These values were imposed by the PEs at the redistribution point.

```

RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf RIP 192.106.5.5/32 | begin Local$
Local
  192.12.106.5 from 0.0.0.0 (214.0.0.12)
    Origin incomplete, metric 1, localpref 100, weight 32768, valid,
redistributed, best, group-best, import-candidate
    Received Path ID 0, Local Path ID 1, version 1358
    Extended community: SoO:214:9005 RT:214:106
  Path #2: Received by speaker 0
  Not advertised to any peer
  Local, (Received from a RR-client)
    214.0.0.11 (metric 10) from 214.0.0.11 (214.0.0.11)
    Received Label 91031
    Origin incomplete, metric 10, localpref 100, valid, internal, import-
candidate, imported
    Received Path ID 0, Local Path ID 0, version 0
    Extended community: SoO:214:9013 RT:214:106
    Source VRF: RIP, Source Route Distinguisher: 214:106

```

There is a race condition regarding XRv2's view of XRv3's loopback. If it learns the RIP route from CSR5 first, it will install it into the RIB and advertise it into BGP. We can simulate this by shutting down the

BGP peering to XRv1 for a minute or so. In this case, XRv2 routes the traffic to CSR5 and then over the backdoor. This implies that CSR5's route is via the slow back door, which is not what we want.

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf RIP 192.106.13.13/32 | begin
Local$
  Local
    192.12.106.5 from 0.0.0.0 (214.0.0.12)
      Origin incomplete, metric 10, localpref 100, weight 32768, valid,
redistributed, best, group-best, import-candidate
      Received Path ID 0, Local Path ID 1, version 1420
      Extended community: SoO:214:9005 RT:214:106
    Path #2: Received by speaker 0
    Not advertised to any peer
    Local, (Received from a RR-client)
      214.0.0.11 (metric 10) from 214.0.0.11 (214.0.0.11)
        Received Label 91012
        Origin incomplete, metric 1, localpref 100, valid, internal, import-
candidate, imported
        Received Path ID 0, Local Path ID 0, version 0
        Extended community: SoO:214:9013 RT:214:106
        Source VRF: RIP, Source Route Distinguisher: 214:106
```

```
R5#show ip route vrf RIP 192.106.13.13
Routing Table: RIP
Routing entry for 192.106.13.13/32
  Known via "rip", distance 120, metric 9
  Redistributing via rip
  Last update from 192.5.106.13 on GigabitEthernet2.206, 00:00:05 ago
  Routing Descriptor Blocks:
  * 192.5.106.13, from 192.5.106.13, 00:00:05 ago, via GigabitEthernet2.206
    Route metric is 9, traffic share count is 1
```

If XRv2 learns the BGP route first, it will install it in the RIB, advertise it to CSR5 with a low metric, and CSR5 will install it. We can simulate this by breaking the backdoor link on CSR5's side for a minute or so. This is a more ideal end-state since the customer is using MPLS over the slow backdoor.

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf RIP 192.106.13.13/32 | begin
Local$
  Local, (Received from a RR-client)
    214.0.0.11 (metric 10) from 214.0.0.11 (214.0.0.11)
      Received Label 91012
      Origin incomplete, metric 1, localpref 100, valid, internal, best,
group-best, import-candidate, imported
      Received Path ID 0, Local Path ID 1, version 1432
      Extended community: SoO:214:9013 RT:214:106
      Source VRF: RIP, Source Route Distinguisher: 214:106

R5#show ip route vrf RIP 192.106.13.13
```

Routing Table: RIP

Routing entry for 192.106.13.13/32

Known via "rip", distance 120, metric 2

Redistributing via rip

Last update from 192.12.106.12 on GigabitEthernet2.106, 00:00:25 ago

Routing Descriptor Blocks:

* 192.12.106.12, from 192.12.106.12, 00:00:25 ago, via GigabitEthernet2.106

Route metric is 2, traffic share count is 1

Solutions to prevent this situation are numerous: metric based ingress filtering on the PEs or egress filtering on the CEs to prevent remote routes from even hitting the MPLS network would be best. Unfortunately, we can't use the RPL to match RIP metrics, otherwise this would be a clean solution. We can match prefixes directly and filter them, which scales poorly but is effective. We will use ingress filtering on XRv1 and XRv2 to ensure routes that came over the backdoor (we can tell by the huge metric) cannot be installed in the routing table. The downside of this approach is that if a PE-CE link fails, the PE "diagonal" from a CE will not have reachability, which is a rare case. We use opposing logic in the RPLs for variety.

```
! XRv1
route-policy RPL_RIP_FILTER
  if not destination in (192.106.13.13/32) then
    drop
  else
    pass
  endif
end-policy
```

```
! XRv2
route-policy RPL_RIP_FILTER
  if destination in (192.106.13.13/32) then
    pass
  else
    drop
  endif
end-policy
```

```
! XRv1 and XRv2
router rip
vrf RIP
interface GigabitEthernet0/0/0/0.106
  route-policy RPL_RIP_FILTER in
```

To test it, we can break the BGP connection between XRv1 and XRv2 again. The CE routers will utilize the slow backdoor to communicate and advertise one another's loopbacks to the PEs. The PEs should reject it, ultimately having no routes to the diagonally-opposed CEs. Only the local loopbacks are allowed.

```
RP/0/0/CPU0:XRv1#show route vrf RIP rip
```

```
R 192.106.13.13/32 [120/1] via 192.11.106.13, 02:49:50,  
GigabitEthernet0/0/0/0.106
```

```
RP/0/0/CPU0:XRv2#show route vrf RIP rip
```

```
R 192.106.5.5/32 [120/1] via 192.12.106.5, 00:01:27,  
GigabitEthernet0/0/0/0.106
```

The backdoor link continues to work as an alternate path when the MPLS network fails.

```
R5#ping vrf RIP 192.106.13.13 source 192.106.5.5  
Type escape sequence to abort.  
Sending 5, 100-byte ICMP Echos to 192.106.13.13, timeout is 2 seconds:  
Packet sent with a source address of 192.106.5.5  
!!!!!  
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/3/4 ms
```

When BGP comes back online, both PEs only see the BGP learned router, ensuring that the PEs immediately prefer the MPLS network again without question.

```
R5#show ip cef vrf RIP 192.106.13.13  
192.106.13.13/32  
  nexthop 192.12.106.12 GigabitEthernet2.106
```

```
RP/0/0/CPU0:XRv3#show cef vrf RIP ipv4 192.106.5.5/32  
192.106.5.5/32, version 105, internal 0x1000001 0x0 (ptr 0xa144ad74) [1], 0x0  
(0xa142e250), 0x0 (0x0)  
local adjacency 192.11.106.11  
  Prefix Len 32, traffic index 0, precedence n/a, priority 2  
  via 192.11.106.11, GigabitEthernet0/0/0/0.106, 5 dependencies, weight 0,  
class 0 [flags 0x0]  
  path-idx 0 NHID 0x0 [0xa0ec734c 0x0]  
  next hop 192.11.106.11  
  local adjacency
```

Finally, we test connectivity between all sites. CSR6 and XRv4 cannot communicate only because the backdoor is down (PE imposes the same SoO), but otherwise, there is full reachability.

```
! CSR5 and CSR6  
tclsh  
foreach x {  
192.106.5.5  
192.106.6.6  
192.106.13.13  
192.106.14.14  
} { ping vrf RIP $x source loopback106 repeat 3 timeout 1 }
```

38.2 VRF label modes

When using VPNv4/v6 BGP to allocate labels for VPN prefixes, we are often accustomed to seeing a unique VPN label per customer prefix. The logic is that this label serves as a demultiplexer value; when the label is received from the MPLS core, the egress LSR performs a single LFIB lookup on a single label in a single table. There is no need to look at the IPv4/v6 destination since the local label that was allocated will determine the IPv4/v6 next-hop within the VPN, as well as represent a specific customer prefix. This is an intelligent approach, but comes with scalability limitations. The most obvious example is transporting the Internet prefixes inside of a central-service L3VPN. Allocating ~350,000 labels is more than one third of the available MPLS label space, not to mention increased burden on the router in terms of memory (large LIB, LFIB, etc). New options were invented to help increase efficiency with respect to label resource management.

The first and oldest alternative was to allocate labels on a per-VRF basis. For each customer, there will be a single, common VPN label allocated by BGP for VPNv4/v6. The drawback of this approach is that it violates one of the original designs of MPLS. When the aggregate VPN label arrives on an egress PE from the core, it could represent any prefix within a given VPN, so the router must perform a second FIB (not LFIB) lookup on the IPv4/v6 destination address to determine how to route it. In cases where there is only a single PE-CE link per VPN, this is obviously inefficient/unnecessary and could affect performance. When there are multiple PE-CE links in the same VPN, the inefficiency remains but it is necessary since the next-hop CE may vary depending on the VPN prefix. Below is the original output of the BGP allocated labels without modification. We see a unique VPN label for each prefix, which is the default.

```
R7#show bgp vpnv4 unicast vrf TOP labels | include 88\.
 88.7.7.7/32      0.0.0.0      7020/nolabel(TOP)
 88.125.160.0/28  10.7.10.10   7012/nolabel
 88.125.160.16/28 10.7.10.10   7013/nolabel
 88.125.160.32/28 10.7.10.10   7014/nolabel
 88.125.160.48/28 10.7.10.10   7015/nolabel
 88.125.160.64/27 10.7.10.10   7016/nolabel
 88.125.160.96/27 10.7.10.10   7017/nolabel
[snip]
```

We have the flexibility to change this per protocol (VPNv4 or VPNv6) or for all protocols, as well as per VRF or for all VRFs. We will enable it on CSR7 inside VRF TOP for VPNv4 only. Notice that CSR7 learns several 88.0.0.0/8 routes from CSR10, and advertises a local loopback inside the same VRF. Before issuing the command, each one had a different label as shown above. Afterwards, however, they all use the same label, regardless of the route's origination. The label is explicitly marked as an IPv4 VRF aggregate label for clarity.

```
! CSR7
mpls label mode vrf TOP protocol bgp-vpnv4 per-vrf
```

```
R7#show bgp vpnv4 unicast vrf TOP labels | include 88\.
 88.7.7.7/32      0.0.0.0      IPv4 VRF Aggr:7005/nolabel(TOP)
 88.125.160.0/28  10.7.10.10   IPv4 VRF Aggr:7005/nolabel
 88.125.160.16/28 10.7.10.10   IPv4 VRF Aggr:7005/nolabel
```

```

88.125.160.32/28 10.7.10.10 IPv4 VRF Aggr:7005/nolabel
88.125.160.48/28 10.7.10.10 IPv4 VRF Aggr:7005/nolabel
88.125.160.64/27 10.7.10.10 IPv4 VRF Aggr:7005/nolabel
[snip]

```

As a result, packets arriving for 88.7.7.7/32 and 88.125.160.0/28 must be looked up in the LFIB first. The LFIB basically says “this belongs to VRF TOP, but since it is an aggregate label, we don’t know how to forward the packet”. The label is popped to reveal the IPv4 packet. The LFIB entry contains no outgoing interface or next-hop. All it knows is that the IPv4 VRF TOP FIB must be consulted.

```

R7#show mpls forwarding-table labels 7005 detail
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id Switched     interface
7005       Pop Label IPv4 VRF[V]  0            aggregate/ TOP
          MAC/Encaps=0/0, MRU=0, Label Stack{}
          VPN route: TOP
          No output feature configured

```

The second VRF-aware FIB lookup tells us how to route the packet once the MPLS labels are removed. There is no way to carry this information inside of MPLS anymore. As an example, two packets could arrive with identical labels but were forwarded two different ways based on their IPv4 destinations.

```

R7#show ip cef vrf TOP 88.7.7.7
88.7.7.7/32
receive for Loopback7

R7#show ip cef vrf TOP 88.125.160.0
88.125.160.0/28
nexthop 10.7.10.10 GigabitEthernet2.570

```

The per-VRF label feature was a good “first attempt” at solving this problem. A better approach is to allocate labels per-next-hop, or as Cisco calls it, “per-CE”. Imagine the scenario described earlier with two CEs in the same VRF. We don’t necessarily need an aggregate label for the whole VRF, as this causes inefficiencies in the forwarding lookups. One label per next-hop means that the label can still demultiplex the traffic’s direction, saving the PE an IPv4/v6 FIB lookup. The CE can look at the IP packet and route the traffic as it normally would. This will also work well in our current situation of having a local loopback in the same VRF as customer routes. We can test this using VPNv6; below is the output before any configuration changes where labels are allocated per-prefix by default.

```

R7#show bgp vpnv6 unicast vrf TOP labels | section 2BAD:88
2BAD:88:125:160::/72
                        FD00:10:7:10::10
                        7007/nolabel
2BAD:88:125:160:100::/72
                        FD00:10:7:10::10

```

```

                                7009/nolabel
2BAD:88:125:160:200::/72
                                FD00:10:7:10::10
                                7010/nolabel
2BAD:88:125:160:300::/72
                                FD00:10:7:10::10
                                7011/nolabel

```

Specifically for this VRF (we only have one, so this doesn't matter for CSR7) we will enable the per-CE label allocation. Notice that these labels are not considered "aggregates"; they still have forwarding instructions, as the LFIB will show later. We can see that the next-hop of CSR10 uses label 7020 for all prefixes, while prefixes with next-hop :: (local) use label 7012. Local/aggregate routes would normally share the same label since an IPv4/v6 FIB lookup is required no matter what.

```

! CSR7
mpls label mode vrf TOP protocol bgp-vpnv6 per-ce

R7#show bgp vpnv6 unicast vrf TOP labels | section 2BAD::?88
2BAD::88:7:7/128 :: 7012/nolabel (TOP)
2BAD:88:125:160::/72
                                FD00:10:7:10::10
                                7020/nolabel
2BAD:88:125:160:100::/72
                                FD00:10:7:10::10
                                7020/nolabel
2BAD:88:125:160:200::/72
                                FD00:10:7:10::10
                                7020/nolabel
2BAD:88:125:160:300::/72
                                FD00:10:7:10::10
                                7020/nolabel

```

The LFIB outputs are different for per-CE versus per-VRF. The first entry for label 7020 is an ordinary LFIB entry for a PE router with the exception of the "prefix" field. A "next-hop-ID" is assigned, which we know is CSR10, and is mapped to this specific local label. I cannot find a command to show the next-hop bindings and the relevance of the ID number 3, and Cisco does not discuss it in their documentation. It is not terribly important; the LFIB entry clearly shows CSR10 as the next-hop anyway.

```

R7#show mpls forwarding-table labels 7020 detail
Local      Outgoing   Prefix          Bytes Label  Outgoing   Next Hop
Label      Label      or Tunnel Id    Switched     interface
7020       No Label   nh-id(3)        0            Gi2.570    FE80::10
MAC/Encaps=18/18, MRU=1504, Label Stack{}
005056A9F961005056A9EA7781000DF286DD
No output feature configured

```

Below is the normal L3VPN output for a local/aggregate route. The prefix ID for the loopback prefix is known since the next-hop is local and removing the VPN label is required anyway.

```
R7#show mpls forwarding-table labels 7012 detail
Local      Outgoing  Prefix      Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id Switched      interface
7012       Pop Label  2BAD::88:7:7/128[V]  \
                                         0          aggregate/TOP
      MAC/Encaps=0/0, MRU=0, Label Stack{}
      VPN route: TOP
      No output feature configured
```

All of these label mode modifications can be viewed by checking the VRF details as well. The per-VRF option even shows the label value since it is authoritative for the entire VRF.

```
R7#show vrf detail TOP | include ^Address|label_alloc
Address family ipv4 unicast (Table ID = 0x1):
  VRF label allocation mode: per-vrf (Label 7005)
Address family ipv6 unicast (Table ID = 0x1E000001):
  VRF label allocation mode: per-ce
Address family ipv4 multicast not active
```

We know that the per-CE feature allocates labels per next-hop. We can check the BGP next-hops to see how many are in VRF TOP, which is a good indication of how many labels we need. This doesn't account for local/aggregate prefixes. Note that any route with a zero next-hop (0.0.0.0 for IPv4 or :: for IPv6) will get a label per-prefix since the router cannot reliably determine the next-hop. This example is a little artificial in this way, but it still demonstrates the feature.

```
R7#show bgp vpnv6 unicast vrf TOP nexthops
# Paths  Nexthop Address
Route Distinguisher: 214:7 (default for vrf TOP)
  20     ::FFFF:214.0.0.8
  7      FD00:10:7:10::10 (FE80::10) (TOP)
```

Nothing is networking is free, and these optimizations come with several limitations. Carrier supporting Carrier (CSC) is not a valid design when using these label aggregation features. Likewise, BGP PIC and iBGP multipath are not advertised to work either. The per-CE feature has even more limitations: routes can only be imported from BGP and cannot be eBGP multi-hop. To demonstrate some of these restrictions, we will try to enable VRF-aware LDP on the VRF TOP interface to support CSC. We can also try to the IPv4 labeled-unicast inside BGP, but we get the same message.

```
R7(config)#int gig2.570
R7(config-subif)#mpls ip
% Cannot configure CSC for vrf TOP, please remove the per-VRF label mode
configuration before configuring CSC
```

```

R7(config)#router bgp 214
R7(config-router)#address-family ipv4 vrf TOP
R7(config-router-af)#neighbor 10.7.10.10 send-label
% Cannot configure CSC for vrf TOP, please remove the per-VRF label mode
configuration before configuring CSC

```

We can configure three bogus static routes on CSR7 within the VRF. Two of them use CSR10's public address of FD00:10:7:10::10 and one of them uses FE80::10. Since the BGP next-hops are recursive using the LL address, the static route pointing to FE80::10 is allowed to share a label with the BGP prefixes from CSR10. This is confusing since the BGP label list for VPNv6 clearly shows the pre-recursive next-hop of the global address, but label 7020 is really associated with FE80::10 per the LFIB. The other two static routes use a new label bound to FD00:10:7:10::10 per the LFIB.

```

! CSR7
ipv6 route vrf TOP 2BAD::88:10:10/128 GigabitEthernet2.570 FD00:10:7:10::10
ipv6 route vrf TOP 2BAD::88:10:100/128 GigabitEthernet2.570 FD00:10:7:10::10
ipv6 route vrf TOP 2BAD::88:100:100/128 GigabitEthernet2.570 FE80::10

router bgp 214
 address-family ipv6 vrf TOP
 redistribute static

```

```

R7#show bgp vpnv6 unicast vrf TOP labels | section 2BAD::?88
 2BAD::88:7:7/128 :: 7012/nolabel(TOP)
 2BAD::88:10:10/128
                        FD00:10:7:10::10
                        7013/nolabel
 2BAD::88:10:100/128
                        FD00:10:7:10::10
                        7013/nolabel
 2BAD::88:100:100/128
                        FE80::10      7020/nolabel
 2BAD:88:125:160::/72
                        FD00:10:7:10::10
                        7020/nolabel
 2BAD:88:125:160:100::/72
                        FD00:10:7:10::10
                        7020/nolabel

[snip]

```

```

R7#show ipv6 static vrf TOP | begin distance
* 2BAD::88:10:10/128 via FD00:10:7:10::10, GigabitEthernet2.570, distance 1
* 2BAD::88:10:100/128 via FD00:10:7:10::10, GigabitEthernet2.570, distance 1
* 2BAD::88:100:100/128 via FE80::10, GigabitEthernet2.570, distance 1

```

```

R7#show mpls forwarding-table labels 7013
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop

```

Label	Label	or Tunnel Id	Switched	interface	
7013	No Label	nh-id(4)	0	Gi2.570	FD00:10:7:10::10

R7#show mpls forwarding-table labels 7020

Local	Outgoing	Prefix	Bytes	Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched		interface	
7020	No Label	nh-id(3)	0		Gi2.570	FE80::10

Because prefixes are aggregated by labels, per-prefix counters no longer function. Since the per-VRF and per-next-hop methods both collapse LFIB entries into single (or few) entries, only aggregate counters are collected. The LFIB has no way to differentiate and makes no attempt. We know label 7005 represents all of VRF TOP for IPv4, so we can check counters for that only.

R1#ping 88.125.160.65 source 82.125.161.129 repeat 10000

R7#show mpls forwarding-table labels 7005

Local	Outgoing	Prefix	Bytes	Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched		interface	
7005	Pop Label	IPv4 VRF[V]	244614			aggregate/TOP

Likewise for the per-next-hop behavior configured for VPNv6, we can only account for packets on a per-next-hop basis.

R1#ping 2BAD:88:125:160:300::1 source 2BAD:82:125:160:400::1 repeat 1000

R7#show mpls forwarding-table interface gig2.570

Local	Outgoing	Prefix	Bytes	Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched		interface	
7013	No Label	nh-id(4)	0		Gi2.570	FD00:10:7:10::10
7020	No Label	nh-id(3)	254998		Gi2.570	FE80::10

We can still send traffic for the bogus routes and temporarily create a new loopback on CSR10 to respond. The counters for the second next-hop will increment because technically the LFIB still received an MPLS packet and performed a pop operation.

R1#ping 2BAD::88:10:100 source 2BAD:82:125:160:400::1 repeat 100

R7#show mpls forwarding-table interface gig2.570

Local	Outgoing	Prefix	Bytes	Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched		interface	
7013	No Label	nh-id(4)	11800		Gi2.570	FD00:10:7:10::10
7020	No Label	nh-id(3)	299484		Gi2.570	FE80::10

There is one final option which takes the per-VRF method and enhances it. Aggregates created on the PE, along with any connected routes, will all share a single label. This allows the SP to leak specific routes that may need their own labels for multipath, add-path, per-prefix counters, or any other feature not

supported with per-VRF or per-CE labels. The aggregate/connected routes behave like per-VRF label aggregates. On CSR7, we configured these label mode options on a per-VRF, per-AFI basis. On CSR8, we will summarily configure the per-VRF connected/aggregate feature for all VRFs and all AFIs for variety. We will also configure some IPv4 and IPv6 aggregates within the VRFs. Last, we redistribute the connected PE-CE transit link so that we have a combination of many “interesting” routes.

```
! CSR8
mpls label mode all-vrfs protocol all-afs vrf-conn-aggr

router bgp 214
 address-family ipv4 vrf BOTTOM
   redistribute connected
   aggregate-address 82.125.160.64 255.255.255.192
   aggregate-address 82.125.160.0 255.255.255.192

 address-family ipv6 vrf BOTTOM
   redistribute connected
   aggregate-address 2BAD:82:125:160:200::/71
   aggregate-address 2BAD:82:125:160::/71
```

We can see that VPNv4 has allocated label 8025 as an aggregate label for the connected transit link and aggregate addresses. Because the aggregates did not suppress longer-matches, the component routes are present as well, and with per-prefix labels.

```
R8#show bgp vpnv4 unicast vrf BOTTOM labels
  Network           Next Hop           In label/Out label
Route Distinguisher: 214:8 (BOTTOM)
 10.1.8.0/24         0.0.0.0             IPv4 VRF Aggr:8025/nolabel (BOTTOM)
 82.125.160.0/28     10.1.8.1            8005/nolabel
 82.125.160.0/26     0.0.0.0             IPv4 VRF Aggr:8025/aggregate (BOTTOM)
 82.125.160.16/28    10.1.8.1            8006/nolabel
 82.125.160.32/28    10.1.8.1            8007/nolabel
 82.125.160.48/28    10.1.8.1            8008/nolabel
 82.125.160.64/27    10.1.8.1            8009/nolabel
 82.125.160.64/26    0.0.0.0             IPv4 VRF Aggr:8025/aggregate (BOTTOM)
```

The LFIB for this label entry is identical to that seen for the per-VRF mode. This design isn’t very effective since all of the longer-matches are unsuppressed, but realistically many would be suppressed, making use of the aggregated labels to reduce LIB size.

```
R8#show mpls forwarding-table labels 8025 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
8025       Pop Label  IPv4 VRF[V]     0            aggregate/BOTTOM
          MAC/Encaps=0/0, MRU=0, Label Stack{}
          VPN route: BOTTOM
```

No output feature configured

VPNv6 is nearly identical except contains a different set of prefixes. These use IPv6 aggregated labels, and in this case, all aggregate and connected routes share label 8019. The LFIB output is identical with the exception of the prefix ID indicating an IPv6 VRF aggregate.

```
R8#show bgp vpnv6 unicast vrf BOTTOM labels
  Network          Next Hop          In label/Out label
Route Distinguisher: 214:8 (BOTTOM)
  2BAD:82:125:160::/72
                        FD00:10:1:8::1  8036/nolabel
  2BAD:82:125:160::/71
                        ::              IPv6 VRF Aggr:8019/aggregate (BOTTOM)
  2BAD:82:125:160:100::/72
                        FD00:10:1:8::1  8037/nolabel
  2BAD:82:125:160:200::/72
                        FD00:10:1:8::1  8038/nolabel
  2BAD:82:125:160:200::/71
                        ::              IPv6 VRF Aggr:8019/aggregate (BOTTOM)
  FD00:10:1:8::/64 ::              IPv6 VRF Aggr:8019/nolabel (BOTTOM)
```

```
R8#show mpls forwarding-table labels 8019 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label      or Tunnel Id    Switched     interface
8019       Pop Label  IPv6 VRF[V]     0            aggregate/BOTTOM
          MAC/Encaps=0/0, MRU=0, Label Stack{}
          VPN route: BOTTOM
          No output feature configured
```

We can confirm the configuration and label value by checking the VRF details. Just like the per-VRF label mode, the per-VRF connected/aggregate option uses a single, predetermined label, so the VRF stores it.

```
R8#show vrf detail BOTTOM | include ^Address|label_alloc
Address family ipv4 unicast (Table ID = 0x2):
  VRF label allocation mode: vrf-conn-aggr (Label 8025)
Address family ipv6 unicast (Table ID = 0x1E000001):
  VRF label allocation mode: vrf-conn-aggr (Label 8019)
Address family ipv4 multicast not active
```

The feature is very similar on XR and is examined briefly. The configuration happens under BGP, which is a more appropriate place for it. On XRv1, we will enable the per-VRF mode for EIGRP IPv4 AFI and per-CE mode for the EIGRP IPv6 AFI. The topology is imperfect for validating these, but we can show that they are working.

```
! XRv1
router bgp 214
  vrf EIGRP
```



```

address-family ipv4 unicast
  label mode per-vrf
address-family ipv6 unicast
  label mode per-ce

```

After issuing these commands, we can verify it by checking the BGP label table. The label “type” shows whether we are using per-VRF, per-CE, or per-prefix. The default for IPv4/v6 and VPNV4/v6 AFI is per-prefix. Interestingly, we see per-VRF labels allocated for OSPF, EIGRP, and RIP, when we only configured it for EIGRP. We also see per-prefix labels for RD 214:101 and 214:106 (OSPF and RIP), but not RD 214:102 (EIGRP). The IOS XR documentation states *“Even if label mode is in use, per-vrf label is allocated for connected, aggregate, and local prefixes.”* This helps explain why we see aggregated, per-VRF labels in the BGP label table for VRFs we haven’t configured at all. In the case of OSPF, EIGRP, and RIP, they all included connected routes when redistributing into BGP, and so those routes were assigned the VRF aggregate label.

```

RP/0/0/CPU0:XRv1#show bgp label table

```

Label	Type	VRF/RD	Context
91005	IPv6 VRF Table	OSPF	–
91006	IPv6 VRF Table	EIGRP	–
91007	IPv4 VRF Table	OSPF	–
91008	IPv4 VRF Table	EIGRP	–
91009	IPv4 VRF Table	RIP	–
91010	IPv4 VRF Prefix	214:104	192.11.104.0/24
91011	IPv6 VRF Prefix	214:104	fd00:192:11:104::/64
91012	IPv4 VRF Prefix	214:106	192.106.13.13/32
91013	IPv4 VRF Prefix	214:104	192.104.13.13/32
91014	IPv4 VRF Prefix	214:101	192.101.13.13/32
91015	IPv6 VRF Prefix	214:104	::192:104:13:13/128
91017	IPv6 VRF Prefix	214:101	::192:101:13:13/128
91019	IPv4 VRF Prefix	214:105	192.105.13.13/32
91020	IPv4 VRF Prefix	214:104	192.5.104.0/24
91021	IPv6 VRF Prefix	214:104	fd00:192:5:104::/64
91022	IPv4 VRF Prefix	214:101	192.5.101.0/24
91023	IPv6 VRF Prefix	214:101	fd00:192:5:101::/64
91032	IPv6 VRF Prefix	214:101	::192:101:5:5/128
91033	IPv6 VRF Prefix	214:101	fd00:192:12:101::/64
91040	IPv6 Nexthop	EIGRP	1

For example, XRv1 has allocated label 91008 for EIGRP IPv6 prefixes that meet the condition outlined above, while using label 91040 for labels learned from each CE. The connected route uses the per-VRF label (green) while the EIGRP-learned routes from XRv3 (specifically FE80::13 as a next-hop) use the per-next-hop label (yellow). It is interesting to note that XR per-CE label allocation works with non-BGP protocols for PE-CE as we demonstrate below.

```

RP/0/0/CPU0:XRv1#show bgp vrf EIGRP ipv6 unicast labels | begin Network

```

Network	Next Hop	Rcvd Label	Local Label

```

Route Distinguisher: 214:102 (default for vrf EIGRP)
*>i::50:50:5:5/128      214.0.0.12      92025      nolabel
*>i::192:102:5:5/128    214.0.0.12      92010      nolabel
*>i::192:102:6:6/128    214.0.0.4       4052       nolabel
*> ::192:102:13:13/128
                        fe80::13        nolabel    91040
*>i::192:102:14:14/128
                        214.0.0.3      3028       nolabel
*>ifd00:192:3:102::/64
                        214.0.0.3      3027       nolabel
*>ifd00:192:4:102::/64
                        214.0.0.4      4005       nolabel
*> fd00:192:5:102::/64
                        fe80::13        nolabel    91040
*>ifd00:192:6:102::/64
                        214.0.0.3      3010       nolabel
*> fd00:192:11:102::/64
                        ::              nolabel    91006
*>ifd00:192:12:102::/64
                        214.0.0.12      92008      nolabel

```

For EIGRP IPv4, we don't see any per-prefix or next-hop labels, implying that all EIGRP IPv4 prefixes share the per-VRF label. This is the expected behavior based on our configuration.

```

RP/0/0/CPU0:XRv1#show bgp vrf EIGRP labels | begin Network
  Network      Next Hop      Rcvd Label      Local Label
Route Distinguisher: 214:102 (default for vrf EIGRP)
*>i50.50.5.5/32      214.0.0.12      92024      nolabel
*>i192.3.102.0/24     214.0.0.3       3055      nolabel
*>i192.4.102.0/24     214.0.0.4       4006      nolabel
*> 192.5.102.0/24     192.11.102.13   nolabel    91008
*>i192.6.102.0/24     214.0.0.3       3056      nolabel
*> 192.11.102.0/24     0.0.0.0         nolabel    91008
*>i192.12.102.0/24     214.0.0.12      92007      nolabel
*>i192.102.5.5/32     214.0.0.12      92012      nolabel
*>i192.102.6.6/32     214.0.0.4       4053      nolabel
*> 192.102.13.13/32    192.11.102.13   nolabel    91008
*>i192.102.14.14/32    214.0.0.3       3057      nolabel

```

In the case of OSPF, where we didn't change the label mode at all, we can see individual, per-prefix labels for all truly OSPF-learned routes. Routes locally originated into BGP, such as the connected PE-CE link to XRv3 or the sham-link endpoint, use the per VRF label 91007 identified in the label table viewed earlier. This is efficient because the VPN FIB needs to be consulted anyway, so allocating 1 label for N connected/aggregate routes conserves label resources.

```

RP/0/0/CPU0:XRv1#show bgp vrf OSPF labels | begin Network
  Network      Next Hop      Rcvd Label      Local Label

```

```

Route Distinguisher: 214:101 (default for vrf OSPF)
*>i101.0.0.3/32      214.0.0.3      3048      nolabel
*>i101.0.0.4/32      214.0.0.4      4026      nolabel
*> 101.0.0.11/32     0.0.0.0        nolabel    91007
*>i101.0.0.12/32     214.0.0.12     92005     nolabel
*>i192.3.101.0/24    214.0.0.3      3049      nolabel
*> 192.5.101.0/24    192.11.101.13  nolabel    91022
*>i192.6.101.0/24    214.0.0.3      3051      nolabel
*> 192.11.101.0/24   0.0.0.0        nolabel    91007
*> 192.12.101.0/24   192.11.101.13  nolabel    91025
*>i192.60.6.6/32     214.0.0.4      4021      nolabel
*> 192.101.5.5/32    192.11.101.13  nolabel    91026
* i                 214.0.0.12     92011     91026
*>i192.101.6.6/32    214.0.0.4      4011      nolabel
*> 192.101.13.13/32  192.11.101.13  nolabel    91014
*>i192.101.14.14/32  214.0.0.3      3054      nolabel

```

We can drill into the details of a single label as well. Using 91040, since it is the most interesting, we can see the table details displayed in verbose fashion, as well as the label's age. The label context contains a reference ID to the actual next-hop. We can check the nexthop-set with BGP to see what "1" means, and as expected, it references FE80::13.

```

RP/0/0/CPU0:XRv1#show bgp label 91040
BGP Label Entry: 91040
  Last Updated : MON 03 21:09:07.386 (00:33:19 ago)
  Label Type   : IPv6 Nexthop
  VRF/RD       : EIGRP
  Label Context: 1
  Refcount     : 2

```

```

RP/0/0/CPU0:XRv1#show bgp vrf EIGRP ipv6 unicast nexthop-set
  Resilient per-CE nexthop set, ID 1
  Number of nexthops 1, Label 91040, Flags 0x1
  Nexthops:
  fe80::13

```

Looking at a per-prefix label, we see similar information, with the addition of the prefix to which the label is bound. XE did not appear to support these kind of label-specific show commands.

```

RP/0/0/CPU0:XRv1#show bgp label 91020
BGP Label Entry: 91020
  Last Updated : MON 01 23:04:58.145 (1d22h ago)
  Label Type   : IPv4 VRF Prefix
  VRF/RD       : 214:104
  Label Context: 192.5.104.0/24
  Refcount     : 1

```

We can view the label allocations in summary form as well. Because XR supports so many label varieties, we can perform a quick check to see the quantity of each label allocated. As an example, one per-CE IPv6 label was allocated, which is used by the EIGRP IPv6 VPN. There are zero for IPv4 because we did not enable that anywhere.

```
RP/0/0/CPU0:XRv1#show bgp label summary
```

Label Alloc Type	Labels
GBL IPv4 Prefix	0
VRF IPv4 Prefix	0
VPN IPv4 Prefix	0
CE IPv4 Nexthop	0
VRF IPv4 Table	3
GBL IPv6 Prefix	0
VRF IPv6 Prefix	0
VPN IPv6 Prefix	0
CE IPv6 Nexthop	1
VRF IPv6 Table	2
ASBR Nexthop	0
TOTAL	6

Checking the data plane, we can see these labels programmed into the LFIB. Because the label refers to a remote VPN prefix, referencing the label by itself did not work, but using a label range did. We also look at a per-VRF label, which is identified as such in the LFIB, whereby the per-nexthop label has “No ID” that makes it unique. The aggregate label LFIB entry essentially directs the router to perform a VPN FIB lookup.

```
RP/0/0/CPU0:XRv1#show mpls forwarding labels 91039 91040
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
91040	Unlabelled	No ID	Gi0/0/0/0.102	fe80::13	0
	Aggregate	No ID	EIGRP		0

```
RP/0/0/CPU0:XRv1#show mpls forwarding labels 91007
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
91007	Aggregate	OSPF: Per-VRF Aggr[V]	OSPF	\	12488

On XRv2, we will perform a more complex example using RPL. XRv2 learns 4 total routes via BGP between IPv4 and IPv6 from CSR5. Currently, each one has a different, per-prefix label. Only examining IPv4, we want to use a per-VRF label for all non-host routes.

```
RP/0/0/CPU0:XRv2#show bgp label table | include 214:104
92013   IPv4 VRF Prefix  214:104          192.12.104.0/24
92014   IPv4 VRF Prefix  214:104          192.104.5.5/32
92015   IPv6 VRF Prefix  214:104          ::192:104:5:5/128
92016   IPv6 VRF Prefix  214:104          fd00:192:12:104::/64
```

Ideally, we should see 192.12.104.0/24 to the per-VRF value, while CSR5's loopback still has a per-prefix label. This might be useful to have different LFIB counters or for performance reasons, since per-VRF label requires a FIB (not LFIB) lookup after the bottom label is removed.

```
! XRv2
prefix-set PS_HOST_ROUTES
  0.0.0.0/0 ge 32
end-set

route-policy RPL_PER_PREFIX_IF_MATCH($PS)
  if destination in $PS then
    set label-mode per-prefix
  else
    set label-mode per-vrf
  endif
end-policy

router bgp 214
  vrf BGP
  address-family ipv4 unicast
  label mode route-policy RPL_PER_PREFIX_IF_MATCH(PS_HOST_ROUTES)
```

Looking at the label table again, we only see 3 routes with per-prefix labels within RD 214:104. The one IPv4 route remaining is CSR5's loopback, which is a host route, and should have a per-prefix label.

```
RP/0/0/CPU0:XRv2#show bgp label table | include 214:104
92014   IPv4 VRF Prefix  214:104          192.104.5.5/32
92015   IPv6 VRF Prefix  214:104          ::192:104:5:5/128
92016   IPv6 VRF Prefix  214:104          fd00:192:12:104::/64
```

We have a new entry in the table for the IPv4 VRF table for the BGP VRF. The label allocated was 92029, and the details indicate it is only a few minutes old. BGP also indicates that the PE-CE link to CSR5 and the backdoor link between CSR5 and XRv3 are both using this label.

```
RP/0/0/CPU0:XRv2#show bgp label table | include BGP
92029   IPv4 VRF Table      BGP      -
```

```
RP/0/0/CPU0:XRv2#show bgp label 92029
BGP Label Entry: 92029
  Last Updated : MON 03 22:03:21.159 (00:02:50 ago)
  Label Type   : IPv4 VRF Table
  VRF/RD       : BGP
  Refcount     : 2
```

```
RP/0/0/CPU0:XRv2#show bgp vrf BGP labels | include 92029
*> 192.5.104.0/24      192.12.104.5      nolabel      92029
*> 192.12.104.0/24     192.12.104.5      nolabel      92029
```

If we create a dummy prefix on CSR5 with a mask of /31, this route is also included in the per-VRF label. We can see the prefix bound to label 92029 on XRv2 as we expect.

```
! CSR5
interface Loopback104
 ip address 192.104.55.55 255.255.255.254 secondary
```

```
RP/0/0/CPU0:XRv2#show bgp vrf BGP labels | include 92029
*> 192.5.104.0/24      192.12.104.5      nolabel      92029
*> 192.12.104.0/24     192.12.104.5      nolabel      92029
*> 192.104.55.54/31    192.12.104.5      nolabel      92029
```

38.3 VRF selection for traffic leaking

Although route-leaking is a little more straightforward (discussed later), we can also forcibly move traffic between VRFs, VRF to global, and global to VRF using other techniques. IOS and IOS-XE platforms support three: VRF selection, VRF autoclassify, and VRF receive with PBR. All three are discussed, but only the last one works on the CSR1000v.

VRF selection does not appear to be supported on the CSR1000v as it is somewhat hardware specific. There isn't anything more to say about this since the command doesn't work. Normally, any traffic coming from the specified source would have its destination IP address looked up in the specified VRF, much like the PBR approach we will evaluate later.

```
R2(config)#vrf selection source 10.2.9.0 255.255.255.0 vrf VIA_XRV4
% VRF Select: failed to add config
```

VRF autoclassify should work, but the system tries to add dynamic route-maps with spaces in the name. I was not able to find a debug command that actually shows the route-map name, but I can see the system try to make them. This feature does not work as a result of the route-maps not being properly created. With the exception of the command that doesn't work, the configuration below is valid.

```
! CSR2
interface GigabitEthernet2.529
 description VRF SELECT USING SECONDARIES
```

```
encapsulation dot1Q 3529
ip address 10.2.9.2 255.255.255.0 secondary vrf VIA_XRV4
ip address 10.9.2.2 255.255.255.0 secondary vrf VIA_CSR6
ip address 2.2.2.2 255.255.255.0
```

```
R2(config-subif)#ip vrf autoclassify source
```

```
% Spaces not Allowed in route-map names
```

```
% Spaces not Allowed in route-map names
```

```
R2#debug route-map api
```

```
RMAP: App:IPVRF; Call to:rmap_api_reg_app() -- 3
```

```
RMAP: App:IPVRF; Call to: rmap_api_create() -- 3
```

```
RMAP: App:IPVRF; Call to: rmap_api_create() -- 3
```

Ultimately, no route-maps are made, IP VRF auto-classify is not listed as an input feature on that interface, and PBR is not active. The feature may not be fully supported on this IOS-XE version; always check the official Cisco release notes for your current version.

```
R2#show route-map dynamic
```

```
Current active dynamic routemaps = 0
```

```
R2#show ip interface gig2.529 | include Input_feat
```

```
Input features: MCI Check
```

```
R6#show ip policy
```

```
Interface      Route map
```

```
[no output]
```

What does work is the “receive VRF” selection using manual PBR. Before beginning, note that you must enable PBR on the interface or configure VRF Select before configuring the VRF receive feature. Since VRF Select is not supported, we will create a PBR policy.

```
R2(config-subif)#ip vrf receive VIA_XRV4
```

```
% Need to enable Policy Based Routing or VRF Select on the interface first
```

Specifically, we want to take traffic from 10.2.9.0/24 and use VRF VIA_XRV4 for the route lookup. We want to do the same for 10.9.2.0/24 but use VRF VIA_CSR6 for the route lookup. We can be even more specific; rather than simply select a VRF for a normal routing lookup, we can cross VRFs while also specifying a next-hop. To demonstrate this, the first case uses the ordinary VRF lookup; packets arriving with source 10.2.9.0/24 will be routed normally, but the destination address will be looked up in VIA_XRV4. For sources 10.9.2.0/24, the regular routing process is overridden and the next-hop is set to CSR6, but is also allowed to cross VRFs. The effect is the same, since we can take traffic from the global table (the interface is not in a VRF) and leak it into other tables.

```
! CSR2
```

```

ip access-list standard ACL_VIA_CSR6
  permit 10.9.2.0 0.0.0.255
ip access-list standard ACL_VIA_XRV4
  permit 10.2.9.0 0.0.0.255

route-map RM_VRF_SELECTION permit 10
  match ip address ACL_VIA_XRV4
  set vrf VIA_XRV4
route-map RM_VRF_SELECTION permit 20
  match ip address ACL_VIA_CSR6
  set ip vrf VIA_CSR6 next-hop 10.2.6.6

interface GigabitEthernet2.529
  encapsulation dot1Q 3529
  ip vrf receive VIA_XRV4
  ip vrf receive VIA_CSR6
  ip address 2.2.2.2 255.255.255.0
  ip policy route-map RM_VRF_SELECTION

```

The problem with this approach is that it doesn't consider the return traffic. Even if XRv4 and CSR6 had return routes for 10.2.9.0/24 and 10.9.2.0/24 respectively, CSR2 will receive them on VRF enabled interfaces, perform a route lookup on the destination, and drop the traffic. We can override this with PBR in the return path to leak traffic from a VRF to the global table. Using similar techniques, we use an extended ACL to match destinations (for the return traffic). For CSR6, we will instruct the policy to perform a normal route lookup in the global table, which means CSR2 will need a route to 10.9.2.0/24 via 2.2.2.9. For XRv4, we will use PBR to avoid that by statically defining the next-hop in the PBR policy to 2.2.2.9. We also specify the keyword "global" so that the lookup on that hard-coded next-hop happens outside of the incoming link VRF, which was VIA_XRV4. We don't need to specify any "receive" VRFs on those interfaces since the traffic isn't being considered for migration into another VRF, but just the global table.

```

! CSR2
ip access-list extended ACL_RETURN_CSR6
  permit ip any 10.9.2.0 0.0.0.255
ip access-list extended ACL_RETURN_XRV4
  permit ip any 10.2.9.0 0.0.0.255

route-map RM_FROM_XRV4 permit 10
  match ip address ACL_RETURN_XRV4
  set ip global next-hop 2.2.2.9
route-map RM_FROM_CSR6 permit 10
  match ip address ACL_RETURN_CSR6
  set global

ip route 10.9.2.0 255.255.255.0 GigabitEthernet2.529 2.2.2.9

interface GigabitEthernet2.526

```



```

vrf forwarding VIA_CSR6
ip policy route-map RM_FROM_CSR6

interface GigabitEthernet2.524
vrf forwarding VIA_XRV4
ip policy route-map RM_FROM_XRV4

```

This is certainly a lot of manual labor. Fortunately, the “set global” and “set vrf” commands are highly scalable since we can just select tables for lookups without statically defining next-hops. However, this technique assumes that the VRF table has a route that can forward the leaked packets. Any combination of static and dynamic routing can be used to achieve this. Before testing, I will clear all counters on the PBR route-maps and enabling PBR debugs.

```

R2#clear route-map counters RM_FROM_XRV4
R2#clear route-map counters RM_FROM_CSR6
R2#clear route-map counters RM_VRF_SELECTION

R2#debug ip policy
Policy routing debugging is on

```

A quick verification shows that PBR is properly applied on the correct interfaces. We also configure some basic secondary addresses (loopbacks would work also) along with a static default route. This is only needed so CSR9 can encapsulate the IP packets to CSR2’s MAC address, allowing PBR to take over once the layer 2 encapsulation is removed. These are effectively our test source addresses.

```

R2#show ip policy
Interface      Route map
Gi2.524        RM_FROM_XRV4
Gi2.526        RM_FROM_CSR6
Gi2.529        RM_VRF_SELECTION

! CSR9
interface GigabitEthernet2.529
encapsulation dot1Q 3529
ip address 10.9.2.9 255.255.255.0 secondary
ip address 10.2.9.9 255.255.255.0 secondary
ip address 2.2.2.9 255.255.255.0

ip route 0.0.0.0 0.0.0.0 GigabitEthernet2.529 2.2.2.2

```

Unfortunately, the CSR1000v (CSR2) is not revealing any PBR debugs at all when we send some test traffic from CSR9 to CSR6. We source the traffic from one of CSR9’s secondary addresses that is mapped to the VIA_CSR6 VPN. We can check the route-map counters for additional confirmation. Notice that no traffic was matched for the first SELECTION match clause, which was for VIA_XRV4. Likewise, no traffic has entered from XRV4 yet. The successful ping and route-map counters are indicative of a successful configuration.

```
R9#ping 10.2.6.6 source 10.9.2.9
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.2.6.6, timeout is 2 seconds:
Packet sent with a source address of 10.9.2.9
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/6/10 ms
```

```
R2#show route-map | include ^route-map|Policy
route-map RM_VRF_SELECTION, permit, sequence 10
  Policy routing matches: 0 packets, 0 bytes
route-map RM_VRF_SELECTION, permit, sequence 20
  Policy routing matches: 5 packets, 590 bytes
route-map RM_FROM_XRV4, permit, sequence 10
  Policy routing matches: 0 packets, 0 bytes
route-map RM_FROM_CSR6, permit, sequence 10
  Policy routing matches: 5 packets, 590 bytes
```

We retest from CSR9, this time sending traffic to XRV4. We will send more packets so the counters look different. Also note that there is nothing stopping us from using more advanced PBR matching, like TCP/UDP ports, DSCP, etc. Unlike auto-classify or VRF select, we gain more flexibility from this approach despite the poorly scaling configurations.

```
R9#ping 10.2.14.14 source 10.2.9.9 repeat 10
Type escape sequence to abort.
Sending 10, 100-byte ICMP Echos to 10.2.14.14, timeout is 2 seconds:
Packet sent with a source address of 10.2.9.9
!!!!!!!!!!!!
Success rate is 100 percent (10/10), round-trip min/avg/max = 2/12/19 ms
```

```
R2#show route-map | include ^route-map|Policy
route-map RM_VRF_SELECTION, permit, sequence 10
  Policy routing matches: 10 packets, 1180 bytes
route-map RM_VRF_SELECTION, permit, sequence 20
  Policy routing matches: 5 packets, 590 bytes
route-map RM_FROM_XRV4, permit, sequence 10
  Policy routing matches: 10 packets, 1180 bytes
route-map RM_FROM_CSR6, permit, sequence 10
  Policy routing matches: 5 packets, 590 bytes
```

38.4 VRF route leaking

Both XE and XR can leak routes between VRFs as well. BGP is typically required for this. On CSR6 and XRV4, we will leak routes from the global table to and from the different customer VRFs. Since we have been using CSR6 and XRV4 for CE routers, they have many VRFs with many diverse routes. Their links to CSR2 are in the global table and are currently used for the VRF traffic leaking with PBR. Both CSR6 and XRV4 have a static route (configured earlier) to 10.0.0.0/8 via CSR2 in their global tables. This is entirely disconnected from the MPLS L3VPN tests conducted earlier. There are two important side notes with

route leaking. First, the VRF must have an RD. If it doesn't the parser gives you an error message. This is because the routes are treated like VPN routes and need to be stored separately in the BGP table.

```
R6(config-vrf-af)#export ipv4 unicast 10 map RM_VRF_TO_GLOBAL
%vrf ISIS does not have "rd" configured, please configure "rd" before
configuring export route-map
```

Second, these commands only have visibility to routes in the BGP RIB. If the route isn't in global BGP, it cannot be imported to a VRF. If the route isn't in VPN BGP, it cannot be exported to the global table. For this reason, we will work with the pre-existing BGP VRF since it already has BGP-learned routes, which implies it has an RD configured. When configuring this feature, we can configure an upper-bound on the number of prefixes imported. Since we are moving routes between tables, they are being copied, so the boundary condition is a safety check. First, we will leak all VPNv4 BGP host routes within the BGP VPN into the global table. The usage of RT's is kind of odd since it makes no sense, but is actually required for the feature to work. If you forget the RTs, iBGP routes appear to work, but locally-originated and eBGP learned routes do not. Overall, the behavior is simply inconsistent if you forget the add RTs.

```
! CSR6
ip prefix-list PL_HOST_ROUTES seq 5 permit 0.0.0.0/0 ge 32

route-map RM_VRF_TO_GLOBAL permit 10
 match ip address prefix-list PL_HOST_ROUTES

vrf definition BGP
 rd 214:104
 address-family ipv4
   export ipv4 unicast 10 map RM_VRF_TO_GLOBAL
   route-target export 1:1
   route-target import 1:1
```

After applying this configuration, we see several host-routes in the BGP global table, as well as the routing table, since these are the best available routes. The routing table annotates that the next-hops are in VRF BGP, not the global table. Notice that all BGP paths are exported, not just the best paths. Since CSR6's local loopback prefix is directly connected, there is no next-hop, but the packet is delivered locally to the router.

```
R6#show bgp ipv4 unicast | begin Network
```

	Network	Next Hop	Metric	LocPrf	Weight	Path
* i	192.104.5.5/32	192.6.104.14		100	0	214 104 ?
*>		192.4.104.4			0	214 214 ?
*>	192.104.6.6/32	0.0.0.0	0		32768	?
* i	192.104.13.13/32	192.6.104.14		100	0	214 104 ?
*>		192.4.104.4			0	214 214 ?
*>i	192.104.14.14/32	192.6.104.14	0	100	0	?

```

R6#show ip route bgp | begin Gate
Gateway of last resort is not set
    192.104.5.0/32 is subnetted, 1 subnets
B       192.104.5.5 [20/0] via 192.4.104.4 (BGP), 00:01:56
    192.104.6.0/32 is subnetted, 1 subnets
B       192.104.6.6 is directly connected, 00:01:30, Loopback104
    192.104.13.0/32 is subnetted, 1 subnets
B       192.104.13.13 [20/0] via 192.4.104.4 (BGP), 00:01:56
    192.104.14.0/32 is subnetted, 1 subnets
B       192.104.14.14 [200/0] via 192.6.104.14 (BGP), 00:01:35

```

The details of the BGP route in the global table indicate that it was imported from VRF BGP, which includes the entire VPNv4 prefix, with the RD and VRF identified. All attributes of the route, to include communities, are carried over. Communities like SoO and RT are not terribly useful in global BGP.

```

R6#show bgp ipv4 unicast 192.104.5.5/32
BGP routing table entry for 192.104.5.5/32, version 3
Paths: (2 available, best #2, table default)
  Not advertised to any peer
  Refresh Epoch 1
  214 104, imported path from 214:104:192.104.5.5/32 (BGP)
    192.6.104.14 from 192.6.104.14 (14.14.14.14)
      Origin incomplete, localpref 100, valid, internal
      Extended Community: SoO:214:4 RT:1:1
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  214 214, imported path from 214:104:192.104.5.5/32 (BGP)
    192.4.104.4 from 192.4.104.4 (214.0.0.4)
      Origin incomplete, localpref 100, valid, external, best
      Extended Community: RT:1:1
      rx pathid: 0, tx pathid: 0x0

```

We can also move routes in the opposite direction, taking them from the global table into the VRF table. We will use a match-all route-map and not set a prefix limit. This is the most dangerous approach but is shown here for variety; it effectively permits all VPNv4 routes to be leaked to the global table. We must advertise the static 10.0.0.0/8 route into BGP and do so with a simple network statement. There is no need to specify RTs when importing to VRF from global.

```

! CSR6
route-map RM_MATCH_ALL permit 10

router bgp 104
  address-family ipv4
    network 10.0.0.0

vrf definition BGP
  address-family ipv4

```

```
import ipv4 unicast map RM_MATCH_ALL
```

First, we verify that advertising 10.0.0.0/8 into the global table was successful. The route is marked as “af-import” to signify that a VRF is importing this route. Then, we check the VRF BGP table for the route and find it. Since the route was already imported to VRF BGP, it cannot be imported again, explaining the “no-import” flag. It is interesting to note that the VRF treats this route as external despite having an AS path length of 0. I assume this is done so that the route can be advertised to iBGP peers.

```
R6#show bgp ipv4 unicast 10.0.0.0
BGP routing table entry for 10.0.0.0/8, version 5
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    10.2.6.2 from 0.0.0.0 (6.6.6.6)
      Origin IGP, metric 0, localpref 100, weight 32768, valid, sourced,
local, af-export(1), best
      rx pathid: 0, tx pathid: 0x0

R6#show bgp vpnv4 unicast vrf BGP 10.0.0.0
BGP routing table entry for 214:104:10.0.0.0/8, version 18
Paths: (1 available, best #1, table BGP)
  Advertised to update-groups:
    3          4
  Refresh Epoch 1
  Local, imported path from 10.0.0.0/8 (global)
    10.2.6.2 (via default) from 0.0.0.0 (6.6.6.6)
      Origin IGP, metric 0, localpref 100, weight 32768, valid, external, no-
import, no-import, best
      rx pathid: 0, tx pathid: 0x0
```

We can check the summary information of these import/export policies within the VRF BGP table. It shows the route-map name, AFI, and prefix counts. The default limit is 1000. Notice that the 4 prefixes (loopbacks) exported from VRF BGP to global only count as 4 and not 6; we learned 6 unique paths but this is a per-prefix counter, which does not account for alternate paths.

```
R6#show bgp vpnv4 unicast vrf BGP | include t_Map
Import Map: RM_MATCH_ALL, Address-Family: IPv4 Unicast, Pfx Count/Limit:
1/1000
Export Map: RM_VRF_TO_GLOBAL, Address-Family: IPv4 Unicast, Pfx Count/Limit:
4/10
```

At this point, we should have reachability from CSR9 to those leaked BGP loopbacks. We already verified that CSR6’s global table had the remote loopbacks, but we also have to verify that the VRF table has the 10.0.0.0/8 summary. The next-hop is in the global (or “default”) table which allows CSR6 to move packets across tables.

```
R6#show ip route vrf BGP 10.0.0.0
```

Routing Table: BGP

```
Routing entry for 10.0.0.0/8
```

```
Known via "bgp 104", distance 20, metric 0, type external
```

```
Last update from 10.2.6.2 00:09:52 ago
```

```
Routing Descriptor Blocks:
```

```
* 10.2.6.2 (default), from 0.0.0.0, 00:09:52 ago
```

```
Route metric is 0, traffic share count is 1
```

```
AS Hops 0
```

```
MPLS label: none
```

Verifying CSR5, we can see it now learns a route to 10.0.0.0/8 within the VPN. There is nothing special about this route as it looks just like any other MPLS L3VPN route from the PE. This aggregate route should allow CSR5 to reach CSR9.

```
R5#show bgp vpnv4 unicast vrf BGP 10.0.0.0
```

```
BGP routing table entry for 214:104:10.0.0.0/8, version 337
```

```
Paths: (1 available, best #1, table BGP)
```

```
Not advertised to any peer
```

```
Refresh Epoch 1
```

```
214 104
```

```
192.12.104.12 (via vrf BGP) from 192.12.104.12 (214.0.0.12)
```

```
Origin IGP, localpref 100, valid, external, best
```

```
rx pathid: 0, tx pathid: 0x0
```

We can test connectivity from CSR6 to CSR5, being sure to specify a valid source address on CSR6. The route lookup occurs in the global table, and CEF doesn't make any special note about the next-hop of CSR4 (Although we know 192.4.104.4 is within VRF BGP, not the global table).

```
R6#ping 192.104.5.5 source 10.2.6.6
```

```
Type escape sequence to abort.
```

```
Sending 5, 100-byte ICMP Echos to 192.104.5.5, timeout is 2 seconds:
```

```
Packet sent with a source address of 10.2.6.6
```

```
!!!!
```

```
Success rate is 100 percent (5/5), round-trip min/avg/max = 9/9/10 ms
```

```
R6#show ip cef 192.104.5.5 detail
```

```
192.104.5.5/32, epoch 2, flags [rib only nolabel, rib defined all labels]
```

```
recursive via 192.4.104.4
```

```
attached to GigabitEthernet2.104
```

When we test connectivity to XRv4's loopback, the ping fails. This is because the 10.0.0.0/8 route was imported as an eBGP route, which means it can be advertised to iBGP peers. The next-hop is unchanged by default. Normally, this would just mean that XRv4 would select the path via CSR3 instead. The key phrase below is "import dampened" which means there is some kind of routing problem with this BGP

prefix. We expected 10.2.6.2 to be inaccessible, as explained earlier, but this new “dampened” word is different.

```
RP/0/0/CPU0:XRv4#show bgp vpv4 unicast vrf BGP 10.0.0.0/8 | begin Paths
Paths: (2 available, best #1)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  214 104
    192.3.104.3 from 192.3.104.3 (214.0.0.3)
      Origin IGP, localpref 100, valid, external, best, group-best, import-
candidate, import dampened
      Received Path ID 0, Local Path ID 1, version 2786
  Path #2: Received by speaker 0
  Not advertised to any peer
  Local
    10.2.6.2 (inaccessible) from 192.6.104.6 (6.6.6.6)
      Origin IGP, metric 0, localpref 100, valid, internal, import dampened
      Received Path ID 0, Local Path ID 0, version 0
```

The issue is that the route to 10.2.6.2 is 10.0.0.0/8, which creates recursive routing logic error on XRv4. It generates occasional syslog messages to warn us about this. We would not have this issue if the next-hop was not covered by the aggregate in question.

```
! XRv4
ipv4_rib[1144]: %ROUTING-RIB-7-SERVER_ROUTING_DEPTH : Recursion loop looking
up prefix 10.2.6.2 in Vrf: "BGP" Tbl: "default" Safi: "Unicast" added by bgp
ipv4_rib[1144]: %ROUTING-RIB-5-TABLE_NH_DAMPED : Nexthops in Vrf: BGP Tbl:
default (0xe0000011) Safi: Unicast are getting damped
```

To fix it, we apply next-hop-self to CSR6’s neighbor configuration towards XRv4. XRv4 then learns this route via the backdoor, as expected, and reachability is achieved. Normally, we would never need next-hop-self in this situation, but since the route was viewed as eBGP, the next-hop was not changed when it was advertised to an iBGP neighbor.

```
! CSR4
router bgp 104
  address-family ipv4 vrf BGP
    neighbor 192.6.104.14 next-hop-self
```

```
RP/0/0/CPU0:XRv4#show bgp vpv4 unicast vrf BGP 10.0.0.0/8 | begin Paths
Paths: (1 available, best #1)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  Local
    192.6.104.6 from 192.6.104.6 (6.6.6.6)
```

```
Origin IGP, metric 0, localpref 100, valid, internal, best, group-best,
import-candidate, import suspect
Received Path ID 0, Local Path ID 1, version 3052
```

```
R6#ping 192.104.14.14 source 10.2.6.6
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.104.14.14, timeout is 2 seconds:
Packet sent with a source address of 10.2.6.6
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 3/5/14 ms
```

Tying this in with the existing VRF traffic leaking, CSR9 now has reachability to all 4 of the VRF BGP loopbacks. The summary results are shown below; provided CSR2 can somehow get the traffic back and forth between CSR9 and CSR6 (which it can), there should be no problems. CSR6 will perform route lookups in the global FIB on traffic coming inbound from CSR2, but the next-hops will be inside the BGP VRF (either towards CSR4 or XRv4). Returning traffic will always go to 10.9.2.9, which is in the global table. Again, the FIB does not make any special annotation of the VRF since it just needs to switch packets between interfaces and add layer 2 encapsulation as needed.

```
R6#show ip cef vrf BGP 10.9.2.9 detail
10.0.0.0/8, epoch 0, flags [rib only nolabel, rib defined all labels]
recursive via 10.2.6.2
attached to GigabitEthernet2.526
```

```
R9#ping 192.104.5.5 source 10.9.2.9
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 9/9/11 ms
```

```
R9#ping 192.104.6.6 source 10.9.2.9
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/5/7 ms
```

```
R9#ping 192.104.13.13 source 10.9.2.9
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 11/30/85 ms
```

```
R9#ping 192.104.14.14 source 10.9.2.9
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 3/6/15 ms
```

We will examine the same feature on XRv4 now. Fortunately, XR does not require the RT configuration to export routes from a VRF to a global table. Using the same logic as before (except with a parameterized RPL), we can match all host routes in the BGP VRF. We also need to specify a BGP RID since there are no available IPv4 unicast global loopbacks; without a RID, BGP cannot select a best-path as it cannot properly initialize. We also must initialize the global IPv4 unicast AFI on XRv4, which is

something we haven't done yet. The "else drop" stanza is technically unnecessary since the RPL will drop all routes not explicitly passed, but I add it for completeness.

```
! XRv4
vrf BGP
  address-family ipv4 unicast
    export to default-vrf route-policy RPL_LEAK(PS_VRF_TO_GLOBAL)

prefix-set PS_VRF_TO_GLOBAL
  0.0.0.0/0 ge 32
end-set

route-policy RPL_LEAK($PS)
  if destination in $PS then
    pass
  else
    drop
  endif
end-policy

router bgp 104
  bgp router-id 14.14.14.14
  address-family ipv4 unicast
```

Looking at the BGP details, we can see the 4 loopbacks inside the global table now. The BGP details of a specific prefix don't specify in what VRF the next-hop lies, but it does list the source VRF, which implies the next-hop is accessible inside that VRF. The route is marked as "imported" much like a normal VPNv4 route would when using RTs. The IPv4 unicast RIB specifies that the next-hop is in VRF BGP, which is correct. The outputs in XR are similar to XE and nicely detail the route leaking operation.

```
RP/0/0/CPU0:XRv4#show bgp ipv4 unicast | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
*> 192.104.5.5/32    192.3.104.3                      0 214 104 ?
*>i192.104.6.6/32    192.6.104.6              0    100      0 ?
*> 192.104.13.13/32  192.3.104.3                      0 214 104 ?
*> 192.104.14.14/32  0.0.0.0                    0              32768 ?
```

```
RP/0/0/CPU0:XRv4#show bgp ipv4 unicast 192.104.6.6/32 | begin Local
Local
  192.6.104.6 from 192.6.104.6 (6.6.6.6)
    Origin incomplete, metric 0, localpref 100, valid, internal, best,
group-best, import-candidate, imported
    Received Path ID 0, Local Path ID 1, version 3
    Source VRF: BGP, Source Route Distinguisher: 214:104
```

```
RP/0/0/CPU0:XRv4#show route 192.104.6.6/32
Routing entry for 192.104.6.6/32
  Known via "bgp 104", distance 200, metric 0, type internal
```

Routing Descriptor Blocks

192.6.104.6, from 192.6.104.6

Next hop in Vrf: "BGP", Table: "default", IPv4 Unicast, Table Id:

0xe0000011

Route metric is 0

No advertising protocols.

We can also see a list of imported routes within the IPv4 unicast AFI on a per-VRF or per-neighbor basis. We can specify the AFI and neighbor concurrently when verifying multi-homed VRF deployments.

```
RP/0/0/CPU0:XRv4#show bgp ipv4 unicast imported-routes vrf BGP | begin Network
```

Network	Neighbor	Route Distinguisher	Source VRF
*> 192.104.5.5/32	192.3.104.3	214:104	BGP
*>i192.104.6.6/32	192.6.104.6	214:104	BGP
*> 192.104.13.13/32	192.3.104.3	214:104	BGP
*> 192.104.14.14/32	0.0.0.0	214:104	BGP

```
RP/0/0/CPU0:XRv4#show bgp ipv4 unicast imported-routes neighbor 192.3.104.3 | begin Network
```

Network	Neighbor	Route Distinguisher	Source VRF
*> 192.104.5.5/32	192.3.104.3	214:104	BGP
*> 192.104.13.13/32	192.3.104.3	214:104	BGP

Next, we will configure XRv4 to import its static route of 10.0.0.0/8 from the global table to the BGP VRF table. The configuration is very similar to XE; there are no RTs required. We will use redistribution to advertise the static route into BGP for variety.

```
! XRv4
vrf BGP
  address-family ipv4 unicast
    import from default-vrf route-policy RPL_LEAK(PS_GLOBAL_TO_VRF)

prefix-set PS_GLOBAL_TO_VRF
  10.0.0.0/8
end-set

router bgp 104
  address-family ipv4 unicast
    redistribute static
```

After configuring this, we can verify that the static route was properly redistributed into BGP IPv4 unicast. After that, it becomes candidate for import into VRF BGP. We verify that the export operation was successful, and also note that it is treated as an eBGP route. XR specifically marks this as an “extranet” route since it was imported from another routing table; specifically, it lists the source VRF as

“default”, which means global. XRv4 also learns the 10.0.0.0/8 prefix via CSR3 (PE) and CSR6 (backdoor), but they are not best paths due to the weight attribute.

```
RP/0/0/CPU0:XRv4#show bgp ipv4 unicast 10.0.0.0/8 brief | begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
*> 10.0.0.0/8	10.2.14.2	0		32768	?

```
RP/0/0/CPU0:XRv4#show bgp vpnv4 unicast vrf BGP 10.0.0.0/8 brief | begin Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
Route Distinguisher: 214:104 (default for vrf BGP)					
*> 10.0.0.0/8	10.2.14.2	0		32768	?
*	192.3.104.3			0	214 104 i
* i	192.6.104.6	0	100	0	i

```
RP/0/0/CPU0:XRv4#show bgp vpnv4 unicast vrf BGP 10.0.0.0/8 | begin Local
```

```
Local
  10.2.14.2 from 0.0.0.0 (14.14.14.14)
    Origin incomplete, metric 0, localpref 100, weight 32768, valid,
extranet, best, group-best, import-candidate, imported
    Received Path ID 0, Local Path ID 1, version 3054
    Source VRF: default, Source Route Distinguisher: 0:0:0
Path #2: Received by speaker 0
Not advertised to any peer
214 104
  192.3.104.3 from 192.3.104.3 (214.0.0.3)
    Origin IGP, localpref 100, valid, external, group-best
    Received Path ID 0, Local Path ID 0, version 0
Path #3: Received by speaker 0
Not advertised to any peer
Local
  192.6.104.6 from 192.6.104.6 (6.6.6.6)
    Origin IGP, metric 0, localpref 100, valid, internal
    Received Path ID 0, Local Path ID 0, version 0
```

The only significant result of using redistribution instead of the network statement is that CSR3 still prefers the iBGP route to 10.0.0.0/8 from CSR4 over the eBGP route from XRv4. This is because origin IGP is preferred over incomplete. This isn't a big deal, but is interesting to note as it changes the traffic pattern.

```
R3#show bgp vpnv4 unicast rd 214:104 10.0.0.0/8
```

```
BGP routing table entry for 214:104:10.0.0.0/8, version 1800
```

```
Paths: (2 available, best #2, table BGP)
```

```
Advertised to update-groups:
```

```
13 14
```

```
Refresh Epoch 1
```

```
104
```

```
192.3.104.14 (via vrf BGP) from 192.3.104.14 (14.14.14.14)
```

```

Origin incomplete, localpref 100, valid, external
Extended Community: SoO:214:3 RT:214:104
rx pathid: 0, tx pathid: 0
Refresh Epoch 2
104, (Received from a RR-client)
214.0.0.4 (metric 10) (via default) from 214.0.0.4 (214.0.0.4)
Origin IGP, metric 0, localpref 100, valid, internal, best
Extended Community: SoO:214:4 RT:214:104
mpls labels in/out nolabel/4048
rx pathid: 0, tx pathid: 0x0

```

XR clearly annotates the next-hop VRF in the FIB. When performing a global lookup on XRv4 for CSR5's loopback, the VRF associated with the next-hop is displayed. This is probably just a cosmetic difference since XE also "knows" the VRF of the next-hop. The same is true for the VRF-aware FIB lookup towards 10.0.0.0/8, where the VRF containing the next-hop is "default".

```

RP/0/0/CPU0:XRv4#show cef ipv4 192.104.5.5/32
192.104.5.5/32, version 24, internal 0x5000001 0x0 (ptr 0xa14156f4) [1], 0x0
(0x0), 0x0 (0x0)
local adjacency 192.3.104.3
Prefix Len 32, traffic index 0, precedence n/a, priority 4
via 192.3.104.3, 3 dependencies, recursive, bgp-ext [flags 0x6020]
path-idx 0 NHID 0x0 [0xa14153f4 0x0]
next hop VRF - 'BGP', table - 0xe0000011
next hop 192.3.104.3 via 192.3.104.3/32

RP/0/0/CPU0:XRv4#show cef vrf BGP ipv4 10.0.0.0/8
10.0.0.0/8, version 4655, internal 0x5000001 0x0 (ptr 0xa1415674) [1], 0x0
(0x0), 0x0 (0x0)
local adjacency 10.2.14.2
Prefix Len 8, traffic index 0, precedence n/a, priority 3
via 10.2.14.2, 2 dependencies, recursive [flags 0x6000]
path-idx 0 NHID 0x0 [0xa1413674 0x0]
next hop VRF - 'default', table - 0xe0000000
next hop 10.2.14.2 via 10.2.14.2/32

```

Before testing connectivity from CSR9, we need to make one additional modification to the policies on CSR2. Currently, when packets arrive from CSR9 with a source inside 10.2.9.0/24, the router doesn't actually forwarding the traffic, but just sets a VRF for the route lookup. Sending traffic from CSR9's address in that range to any loopback will fail since the VIA_XRV4 table does not have a route for those prefixes.

```

R2#show route-map RM_VRF_SELECTION
route-map RM_VRF_SELECTION, permit, sequence 10
Match clauses:
ip address (access-lists): ACL_VIA_XRV4
Set clauses:

```

```

vrf VIA_XRV4
Policy routing matches: 98 packets, 11564 bytes
route-map RM_VRF_SELECTION, permit, sequence 20
Match clauses:
  ip address (access-lists): ACL_VIA_CSR6
Set clauses:
  ip vrf VIA_CSR6 next-hop 10.2.6.6
Policy routing matches: 99 packets, 11394 bytes

R2#show ip cef vrf VIA_XRV4 192.104.14.14
0.0.0.0/0
no route

```

To solve it, a VRF-aware static default route is added to CSR2. This means that after the policy-based lookup occurs, the VRF-aware FIB can forward traffic to XRv4. We could also run BGP or IGP with CSR2 to exchange routing information, but the static route is quick and easy.

```

! CSR2
ip route vrf VIA_XRV4 0.0.0.0 0.0.0.0 GigabitEthernet2.524 10.2.14.14

R2#show ip cef vrf VIA_XRV4 192.104.14.14
0.0.0.0/0
nexthop 10.2.14.14 GigabitEthernet2.524

```

CSR9 can now reach XRv14's loopback using 10.2.9.2 as a source, but no other loopbacks are reachable. This is due to the asymmetric routing that occurs. In this example, CSR9 cannot reach CSR6's loopback.

```

R9#ping 192.104.14.14 source 10.2.9.9
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.104.14.14, timeout is 2 seconds:
Packet sent with a source address of 10.2.9.9
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/5/19 ms

R9#ping 192.104.6.6 source 10.2.9.9
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.104.6.6, timeout is 2 seconds:
Packet sent with a source address of 10.2.9.9
.....
Success rate is 0 percent (0/5)

```

Considering that the traffic is probably reaching CSR6's loopback (the BGP routing is correct), the issue is with the return flow. We quickly verify that traffic reaching XRv4 for CSR6's loopback is routed to CSR6, as expected.

```

RP/0/0/CPU0:XRv4#show cef ipv4 192.104.6.6/32

```

```

192.104.6.6/32, version 26, internal 0x5000001 0x0 (ptr 0xa1416df4) [1], 0x0
(0x0), 0x0 (0x0)
local adjacency 192.6.104.6
Prefix Len 32, traffic index 0, precedence n/a, priority 4
via 192.6.104.6, 3 dependencies, recursive [flags 0x6000]
path-idx 0 NHID 0x0 [0xa1416b74 0x0]
next hop VRF - 'BGP', table - 0xe0000011
next hop 192.6.104.6 via 192.6.104.6/32

```

When CSR6 sends the reply packet back to 10.2.9.2, it follows its local static route towards CSR2.

```

R6#show ip cef vrf BGP 10.2.9.2
10.0.0.0/8
nexthop 10.2.6.2 GigabitEthernet2.526

```

The issue is that CSR2 does not have an inbound PBR policy to account for traffic to this destination. When the PBR lookup fails, CSR2 attempts to perform a normal VRF-aware FIB lookup, yet has no routing entry for it either.

```

R2#show route-map RM_FROM_CSR6
route-map RM_FROM_CSR6, permit, sequence 10
Match clauses:
ip address (access-lists): ACL_RETURN_CSR6
Set clauses:
global
Policy routing matches: 93 packets, 10842 bytes

```

```

R2#show access-list ACL_RETURN_CSR6
Extended IP access list ACL_RETURN_CSR6
10 permit ip any 10.9.2.0 0.0.0.255

```

```

R2#show ip cef vrf VIA_CSR6 10.2.9.2
0.0.0.0/0
no route

```

We can fix this problem by adding another entry to the ACL to account for 10.2.9.0/24 sources, which essentially permits asymmetric routing to work. More interestingly, we can add a static VRF-aware route with a global next-hop for this destination. When the PBR fails to match the incoming traffic to perform a lookup in the global VRF, CSR2 performs a lookup in the VRF VIA_CSR6 routing table. The static route is used, which essentially directs traffic for this destination towards CSR9's address in the global table.

```

! CSR2
ip route vrf VIA_CSR6 10.2.9.0 255.255.255.0 GigabitEthernet2.529 2.2.2.9
global

```

```

R2#show ip cef vrf VIA_CSR6 10.2.9.2
10.2.9.0/24

```

nexthop 2.2.2.9 GigabitEthernet2.529

If the asymmetry was happening in the other direction, we will have the same problem on CSR2's interface to XRv4. We can test this by sending traffic to XRv4's loopback using a source of 10.9.2.9, which is directed to CSR6. To verify the asymmetry, CSR6 performs a lookup in its global FIB and routes the packet to XRv4 over the backdoor. XRv4 performs a lookup in its VRF-aware FIB, processes the local packet, and routes the reply directly to CSR2.

```
R6#show ip cef 192.104.14.14
192.104.14.14/32
  nexthop 192.6.104.14 GigabitEthernet2.204
```

```
RP/0/0/CPU0:XRv4#show cef vrf BGP ipv4 192.104.14.14
192.104.14.14/32, version 17, attached, receive
  Prefix Len 32
  internal 0x3006041 (ptr 0xa1416274) [3], 0x0 (0xa13dfe84), 0x0 (0x0)
```

```
RP/0/0/CPU0:XRv4#show cef vrf BGP ipv4 10.9.2.9
10.0.0.0/8, version 4655, internal 0x5000001 0x0 (ptr 0xa1415674) [1], 0x0
(0x0), 0x0 (0x0)
  local adjacency 10.2.14.2
  Prefix Len 8, traffic index 0, precedence n/a, priority 3
  via 10.2.14.2, 2 dependencies, recursive [flags 0x6000]
  path-idx 0 NHID 0x0 [0xa1413674 0x0]
  next hop VRF - 'default', table - 0xe0000000
  next hop 10.2.14.2 via 10.2.14.2/32
```

CSR2's inbound policy-routing matches an ACL that only accounts for traffic sourced from 10.2.9.0/24. The routing entry we added earlier would cause the packet to loop until TTL expires, which is undesirable.

```
R2#show route-map RM_FROM_XRV4
route-map RM_FROM_XRV4, permit, sequence 10
  Match clauses:
    ip address (access-lists): ACL_RETURN_XRV4
  Set clauses:
    ip global next-hop 2.2.2.9
  Policy routing matches: 92 packets, 10840 bytes
```

```
R2#show access-lists ACL_RETURN_XRV4
Extended IP access list ACL_RETURN_XRV4
  10 permit ip any 10.2.9.0 0.0.0.255
```

```
R2#show ip cef vrf VIA_XRV4 10.9.2.0
0.0.0.0/0
  nexthop 10.2.14.14 GigabitEthernet2.524
```

Because this route-map clause manually sets the correct next-hop, we don't have to worry about changing the VIA_XRV4 routing table. Instead, we can simply invoke the pre-existing ACL_RETURN_XRV4 access-list in the same route-map entry. This will use Boolean OR logic, so that traffic destinations matching either list can be forwarding to 2.2.2.9. We are effectively extending the match logic in the route-map to account for both 10.2.9.0/24 and 10.9.2.0/24.

```
! CSR2
route-map RM_FROM_XRV4 permit 10
  match ip address ACL_RETURN_XRV4 ACL_RETURN_CSR6

R2#show route-map RM_FROM_XRV4
route-map RM_FROM_XRV4, permit, sequence 10
  Match clauses:
    ip address (access-lists): ACL_RETURN_XRV4 ACL_RETURN_CSR6
  Set clauses:
    ip global next-hop 2.2.2.9
  Policy routing matches: 95 packets, 11194 bytes
```

Next, we verify that CSR9 has reachability to all remote loopbacks from all valid sources. We can use a nested set of for loops to accomplish this. After enumerating all the possibilities for the destination, we nest another loop to iterate over the sources. The result is X * Y (8 total) verifications. The output is not shown, but it was successful for all ping targets.

```
! CSR9
tclsh
foreach x {
192.104.5.5
192.104.6.6
192.104.13.13
192.104.14.14
} { foreach y {
10.2.9.9
10.9.2.9
} { ping $x source $y repeat 3 timeout 1 } }
```

Regarding IPv6, XE only supports route-export from VRF to global. Importing from global to VRF does not appear supported in this code release. Only being able to go one-way seems awkward and incapable.

```
R6(config-vrf-af)#export ?
  ipv6  Address family based VRF export
  map   Route-map based VRF export

R6(config-vrf-af)#import ?
  map   Route-map based VRF import
```


We will test it briefly just to see the routes populated in the IPv6 global table, but nothing more. We will create an IPv6 prefix-list just to match the host routes inside of VRF BGP. We also need to include dummy RTs for this to work properly.

```
! CSR6
ipv6 prefix-list PL_HOST_ROUTES_V6 seq 5 permit ::/0 ge 128

route-map RM_VRF_TO_GLOBAL_V6 permit 10
 match ipv6 address prefix-list PL_HOST_ROUTES_V6

vrf definition BGP
 address-family ipv6
  export ipv6 unicast map RM_VRF_TO_GLOBAL_V6
  route-target export 1:1
  route-target import 1:1
```

As expected, we see all 4 loopbacks within the global table, including the non-best-paths seen by BGP. All of the BGP candidate routes are imported which is consistent with the IPv4 behavior. The VPNv6 BGP table for that VRF shows that 4 prefixes were exported out of a maximum of 1000. This is the same default as IPv4 and serves as a safety mechanism.

```
R6#show bgp ipv6 unicast | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
* i  ::192:104:5:5/128
      FD00:192:6:104::14
                                100      0 214 104 ?
*>
      FD00:192:4:104::4
                                0 214 214 ?
*>  ::192:104:6:6/128
      ::
                                0      32768 ?
*
      FD00:192:4:104::4
                                0 214 214 ?
* i  ::192:104:13:13/128
      FD00:192:6:104::14
                                100      0 214 104 ?
*>
      FD00:192:4:104::4
                                0 214 214 ?
* i  ::192:104:14:14/128
      FD00:192:6:104::14
                                0      100      0 ?
*>
      FD00:192:4:104::4
                                0 214 214 ?
```

```
R6#show bgp vpnv6 unicast vrf BGP | include t_Map
Export Map: RM_VRF_TO_GLOBAL_V6, Address-Family: IPv6 Unicast, Pfx
Count/Limit: 4/1000
```

The global routing table now shows these routes as BGP routes. For the connected loopback, we can see the % symbol is used to indicate that the interface is in a different VRF.

```
R6#show ipv6 route bgp | begin ^B
B   ::192:104:5:5/128 [20/0]
    via FE80::4, GigabitEthernet2.104
B   ::192:104:6:6/128 [20/0]
    via Loopback104%BGP, directly connected
B   ::192:104:13:13/128 [20/0]
    via FE80::4, GigabitEthernet2.104
B   ::192:104:14:14/128 [20/0]
    via FE80::4, GigabitEthernet2.104
```

An interesting note is that the VRF table selects XRv4 as the best-path to XRv4's loopback, which makes sense given the shorter AS path length. The global table selects CSR4 as the next-hop for this route (seen above, also) because the next-hop is inaccessible.

```
R6#show bgp vpnv6 unicast vrf BGP ::192:104:14:14/128
BGP routing table entry for [214:104]::192:104:14:14/128, version 64
Paths: (2 available, best #1, table BGP)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local
    FD00:192:6:104::14 (via vrf BGP) from FD00:192:6:104::14 (14.14.14.14)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: SoO:214:4 RT:1:1
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 2
  214 214
    FD00:192:4:104::4 (FE80::4) (via vrf BGP) from FD00:192:4:104::4
    (214.0.0.4)
      Origin incomplete, localpref 100, valid, external
      Extended Community: SoO:214:3 RT:1:1
      rx pathid: 0, tx pathid: 0
```

```
R6#show bgp ipv6 unicast ::192:104:14:14/128
BGP routing table entry for ::192:104:14:14/128, version 55
Paths: (2 available, best #2, table default)
  Not advertised to any peer
  Refresh Epoch 1
  Local, imported path from [214:104]::192:104:14:14/128 (BGP)
    FD00:192:6:104::14 (inaccessible) from FD00:192:6:104::14 (14.14.14.14)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Extended Community: SoO:214:4 RT:1:1
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 2
  214 214, imported path from [214:104]::192:104:14:14/128 (BGP)
```

```
FD00:192:4:104::4 (FE80::4) from FD00:192:4:104::4 (214.0.0.4)
  Origin incomplete, localpref 100, valid, external, best
  Extended Community: SoO:214:3 RT:1:1
  rx pathid: 0, tx pathid: 0x0
```

This doesn't make sense since both routes are connected to CSR6 and are in the VRF; either both should be inaccessible or both should be accessible. Neither one was imported into the global table. The only difference I can see is that the valid route puts the next-hop's LL address into parenthesis.

```
R6#show ipv6 route FD00:192:4:104::
% Route not found
```

```
R6#show ipv6 route FD00:192:6:104::
% Route not found
```

Adjusting the next-hop inbound to include LL addresses does not solve the problem. The LL address of FE80::14 is still absent from the iBGP route.

```
! CSR6
route-map RM_IPV6_NHOP_ADJUST permit 10
  set ipv6 next-hop FD00:192:6:104::14 FE80::14

router bgp 104
  address-family ipv6 vrf BGP
    neighbor FD00:192:6:104::14 route-map RM_IPV6_NHOP_ADJUST in

R6#show bgp vpnv6 unicast vrf BGP ::192:104:14:14/128 | include via vrf
  FD00:192:6:104::14 (via vrf BGP) from FD00:192:6:104::14 (14.14.14.14)
  FD00:192:4:104::4 (FE80::4) (via vrf BGP) from FD00:192:4:104::4
(214.0.0.4)
```

A last-ditch effort involves leaking that connected /64 route into the VRF to satisfy the route recursion. This actually does solve the problem, but is kind of a hack.

```
! CSR6
ipv6 prefix-list PL_HOST_ROUTES_V6 seq 10 permit FD00:192:6:104::/64

R6#show ipv6 route bgp | begin ^B
B  ::192:104:5:5/128 [20/0]
    via FE80::4, GigabitEthernet2.104
B  ::192:104:6:6/128 [20/0]
    via Loopback104%BGP, directly connected
B  ::192:104:13:13/128 [20/0]
    via FE80::4, GigabitEthernet2.104
B  ::192:104:14:14/128 [200/0]
    via FD00:192:6:104::14%BGP, GigabitEthernet2.104%BGP
B  FD00:192:6:104::/64 [20/0]
```

via GigabitEthernet2.204%BGP, directly connected

The feature appears more stable on XR. We will leak the IPv6 VPN loopbacks into the global table, re-using the RPL from earlier by feeding it a new prefix set. We also have to initialize the IPv6 unicast AFI under BGP.

```
! Xrv4
vrf BGP
  address-family ipv6 unicast
    export to default-vrf route-policy RPL_LEAK(PS_VRF_TO_GLOBAL_V6)

prefix-set PS_VRF_TO_GLOBAL_V6
  ::/0 ge 128
end-set

router bgp 104
  address-family ipv6 unicast
```

Once the IPv6 unicast AFI is finishing initializing and importing the routes, we can see them in the global table. All of the next-hops are reachable, so we don't have the same problem on XR as we did on XE. We can see all four of the routes have the correct next-hops, all have valid best-paths, and all were imported from the source VRF BGP.

```
RP/0/0/CPU0:XRv4#show bgp ipv6 unicast imported-routes vrf BGP | begin
Network
```

Network	Neighbor	Route Distinguisher	Source VRF
*> ::192:104:5:5/128	fd00:192:3:104::3	214:104	BGP
*>i ::192:104:6:6/128	fd00:192:6:104::6	214:104	BGP
*> ::192:104:13:13/128	fd00:192:3:104::3	214:104	BGP
*> ::192:104:14:14/128	::	214:104	BGP

The routing table displays the expected information as well. Remote routes, like CSR5's loopback, are reachable via the PE (CSR3). The "local site" route of CSR6's loopback is reachable via the backdoor link.

```
RP/0/0/CPU0:XRv4#show route ipv6 unicast bgp
B    ::192:104:5:5/128
      [20/0] via fe80::3 (nexthop in vrf BGP), 00:00:44,
GigabitEthernet0/0/0/0.104
B    ::192:104:6:6/128
      [200/0] via fd00:192:6:104::6 (nexthop in vrf BGP), 00:00:44
B    ::192:104:13:13/128
```

```

[20/0] via fe80::3 (nexthop in vrf BGP), 00:00:44,
GigabitEthernet0/0/0/0.104
B    ::192:104:14:14/128 is directly connected,
      00:00:44, Loopback104 (nexthop in vrf BGP)

```

We can also import global routes into a VRF on XR for IPv6. Using the connected route on XRv4's interface to CSR2, we re-use the RPL using a new prefix-set to import the route from global to VRF BGP. We need to ensure this route is advertised into BGP, or else the import will not work.

```

! XRv4
prefix-set PS_GLOBAL_TO_VRF_V6
  2001::/16 le 96
end-set

vrf BGP
  address-family ipv6 unicast
    import from default-vrf route-policy RPL_LEAK(PS_GLOBAL_TO_VRF_V6)

router bgp 104
  address-family ipv6 unicast
    network 2001:dead:beef:cafe::/64

```

First, we confirm the route was successfully advertised into the IPv6 unicast global BGP table via the network statement. Then, we confirm it was imported into the VPNv6 BGP table for VRF BGP using the import route-policy. Just like with IPv4, BGP calls this route an “extranet”, which has eBGP characteristics, except with an AS path length of 0 (route is local). The source VRF is identified as being the default table (global) with no RD.

```

RP/0/0/CPU0:XRv4#show bgp ipv6 unicast 2001::/16 longer-prefixes | begin
Network
  Network          Next Hop          Metric LocPrf Weight Path
*> 2001:dead:beef:cafe::/64
      ::              0              32768 i

RP/0/0/CPU0:XRv4#show bgp vpnv6 unicast vrf BGP 2001:dead:beef:cafe::/64 |
beg$
  Local
  :: from 0.0.0.0 (14.14.14.14)
    Origin IGP, metric 0, localpref 100, weight 32768, valid, extranet,
best, group-best, import-candidate, imported
    Received Path ID 0, Local Path ID 1, version 139
    Source VRF: default, Source Route Distinguisher: 0:0:0

```

From routers across the VPN, there is nothing special about this route, and it looks like a normal BGP route within the MPLS L3VPN. We can verify reachability from XRv4 to the other loopbacks using this route to ensure the import/export successfully achieves full reachability.

```

R5#show bgp vpnv6 unicast vrf BGP 2001:DEAD:BEEF:CAFE::/64
BGP routing table entry for [214:104]2001:DEAD:BEEF:CAFE::/64, version 2411
Paths: (1 available, best #1, table BGP)
  Not advertised to any peer
  Refresh Epoch 1
  214 104
    FD00:192:12:104::12 (FE80::12) (via vrf BGP) from FD00:192:12:104::12
    (214.0.0.12)
      Origin IGP, localpref 100, valid, external, best
      rx pathid: 0, tx pathid: 0x0

RP/0/0/CPU0:XRv4#ping ::192:104:5:5 source 2001:dead:beef:cafe::14
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/11/29 ms

RP/0/0/CPU0:XRv4#ping ::192:104:6:6 source 2001:dead:beef:cafe::14
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/4/9 ms

RP/0/0/CPU0:XRv4#ping ::192:104:13:13 source 2001:dead:beef:cafe::14
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/6/9 ms

```

38.5 L3VPN import/export maps

Both XE and XR support the ability to apply additional constraints to restrict VPN membership beyond a simple RT import/export policy. This allows the routers to advertise certain routes with additional exported route-targets, or to filter inbound routes more aggressively even if they match the RT-import configured. Essentially, it is a per-prefix granularity tool to refine VPN membership. For this test, we will use CSR7 and XRv2 as the PEs to exchange routes between VRF BGP and VRF TOP. First, we verify the existing RT import/export policies on both routers. CSR7 has an asymmetric policy where it imports and exports routes with different RTs. XRv2 uses RT:214:104 for both directions.

```

R7#show vrf detail TOP | include Address|VPN|RT
VRF TOP (VRF Id = 1); default RD 214:7; default VPNID <not set>
Address family ipv4 unicast (Table ID = 0x1):
  Export VPN route-target communities
  RT:214:7
  Import VPN route-target communities
  RT:214:8
Address family ipv6 unicast (Table ID = 0x1E000001):
  Export VPN route-target communities
  RT:214:7
  Import VPN route-target communities
  RT:214:8
Address family ipv4 multicast not active

```

```
RP/0/0/CPU0:XRv2#show vrf BGP
```

VRF	RD	RT	AFI	SAFI
BGP	214:104			
		import 214:104	IPV4	Unicast
		export 214:104	IPV4	Unicast
		import 214:104	IPV6	Unicast
		export 214:104	IPV6	Unicast

The first goal will be to get all IPv4 host routes from XRv2 VRF BGP into CSR7 VRF TOP without configuring CSR7. If we didn't have the host route restriction, we could just add another export RT to XRv2 with value RT:214:8. That configuration is shown below, but is uncommitted, since it is incorrect.

```
! XRv2 (not committed, incorrect)
vrf BGP
  address-family ipv4 unicast
    export route-target
      214:8
```

We know that CSR7 is already importing RT:214:8, and it would work. However, we need to be more granular and only attach RT:214:8 to host routes inside the VPN. Non-matching routes will not have RT:214:8 applied, which means CSR7 will not import them into VRF TOP. Using RPL, we can construct a flexible policy to do this. We already have a prefix-set matching IPv4 host routes from earlier, so we can reuse that. The policy adds (does not replace) another RT to the extended community list and simply passes other routes. The "passed" routes will only have the "normal" export RTs defined under the VRF.

```
! XRv2
vrf BGP
  address-family ipv4 unicast
    export route-policy RPL_IF_MATCH_SET_RT(PS_HOST_ROUTES, RT_214_8)

extcommunity-set rt RT_214_8
  214:8
end-set

route-policy RPL_IF_MATCH_SET_RT($PS, $RT)
  if destination in $PS then
    set extcommunity rt $RT additive
  else
    pass
  endif
end-policy
```

After applying this policy, we confirm that it is working locally. XRv2 learns multiple routes from CSR5, and two are shown below. The PE-CE link does not have the new RT but the host route does, implying that the policy worked.

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf BGP 192.12.104.0/24 | include
Extend
```

```
Extended community: SoO:214:513 RT:214:104
```

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf BGP 192.104.5.5/32 | include
Extend
```

```
Extended community: SoO:214:513 RT:214:8 RT:214:104
```

Of greater importance is verifying that these routes were imported on CSR7. We can use a regex to filter based on originating AS 104 to ensure only the host routes were allowed. Since XRv2 only originated one host route, it imposed RT:214:8 on that single route only. The route contains all information we would expect to see, including the new RT, MPLS label, and other components needed for reachability.

```
R7#show bgp vpnv4 unicast vrf TOP quote-regex "104$" | begin Network
Network           Next Hop           Metric LocPrf Weight Path
Route Distinguisher: 214:7 (default for vrf TOP)
*>i 192.104.5.5/32  214.0.0.12          0      100      0 104 ?
```

```
R7#show bgp vpnv4 unicast vrf TOP 192.104.5.5/32
BGP routing table entry for 214:7:192.104.5.5/32, version 728
Paths: (1 available, best #1, table TOP)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  104, imported path from 214:104:192.104.5.5/32 (global)
    214.0.0.12 (metric 20) (via default) from 214.0.0.12 (214.0.0.12)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: SoO:214:513 RT:214:8 RT:214:104
      mpls labels in/out nolaabel/92014
      rx pathid: 0, tx pathid: 0x0
```

At this point, we will not have reachability since XRv2 is not importing RT:214:7, which is exported by CSR7. At this point, we can either export RT:214:104 on CSR7 or import RT:214:7 on XRv2. The two approaches have different consequences. Exporting RT:214:104 on CSR7 means that all 4 of the PEs will be importing those routes, since all of them are importing RT:214:104 into VRF BGP. Importing RT:214:7 on XRv2 means that only XRv2 (not counting CSR8) will import those routes, which is probably cleaner. However, we add the additional requirement that XRv2 should not import all of the routes from CSR7 since there are several. We will attempt to match /26 routes with RT:214:7 set. It is important that the RPL test for RT:214:7 in the match statement or else the /26 filter would apply to all imported routes. Our requirement is to permit /26 prefixes with RT:214:7 while not applying this strict filter to other imported routes.

```
! XRv2
vrf BGP
  address-family ipv4 unicast
    import route-policy RPL_IF_MATCH_PASS(PS_LENGTH_26, RT_214_7)
```



```

extcommunity-set rt RT_214_7
    214:7
end-set

prefix-set PS_LENGTH_26
    0.0.0.0/0 ge 26 le 26
end-set

route-policy RPL_IF_MATCH_PASS($PS, $RT)
    if destination in $PS and extcommunity rt matches-every $RT then
        pass
    elseif extcommunity rt matches-every $RT then
        drop
    else
        pass
    endif
end-policy

```

After applying this configuration, XRv2 is not importing any routes from AS 88 into VRF BGP. Because XRv2 is a route-reflector, it already retains all the VPN routes, so we can verify that it is at least receiving them from CSR7 within RD:214:7 (RD, not RT). The RT of the route is RT:214:7 as expected. For some reason, the configuration did not work.

```

RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf BGP regexp 88$
[no output]

```

```

RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast rd 214:7 88.125.160.128/26
[snip]
88, (Received from a RR-client)
214.0.0.7 (metric 20) from 214.0.0.7 (214.0.0.7)
    Received Label 7005
    Origin IGP, metric 0, localpref 100, valid, internal, best, group-best,
import-candidate, not-in-vrf
    Received Path ID 0, Local Path ID 1, version 896
    Extended community: RT:214:7

```

The reason for this issue is in the order of operations. The import-map is processed only after the routes enter the VRF, and they enter the VRF via the “import route-target” command. Therefore, we must specify the RT manually in the VRF, then use the RPL to filter the routes further. This is also like the export-map which is processed after “export route-target”, which is why we must be careful not to overwrite RT values. We will correct our configuration below. The RPL is still correct in its three-stage logic: pass if RT:214:7 and /26, drop if RT:214:7 (non /26 prefixes from AS 88), pass otherwise (for unrelated imports).

```

! XRv2
vrf BGP

```

```
address-family ipv4 unicast
import route-target 214:7
```

Now, we can see the /26 routes from AS 88 in VRF BGP as expected. The RPL filtered all of the non /26 routes during the import process, but only routes already matching RT:214:7 were candidates to be compared against that filter.

```
RP/0/0/CPU0:XRv2#show bgp vpnv4 unicast vrf BGP regexp 88$ | begin Network
      Network                Next Hop                Metric LocPrf Weight Path
Route Distinguisher: 214:104 (default for vrf BGP)
*>i88.125.160.128/26          214.0.0.7                0      100      0 88 i
*>i88.125.160.192/26          214.0.0.7                0      100      0 88 i
```

We quickly check the routes on CSR5 and CSR10 to ensure the ultimate endpoints can reach one another. Then, we confirm the data-plane is functional using ping. CSR10 has the route to CSR5's loopback, and CSR5 has the routes to CSR1's /26 prefixes. Notice that we achieved this new reachability without configuring CSR7 at all.

```
R10#show bgp ipv4 unicast quote-regexp "104$" | begin Network
      Network                Next Hop                Metric LocPrf Weight Path
*> 192.104.5.5/32            10.7.10.7                0      100      0 214 104 ?
```

```
R5#show bgp vpnv4 unicast vrf BGP quote-regexp "88$" | begin Network
      Network                Next Hop                Metric LocPrf Weight Path
Route Distinguisher: 214:104 (default for vrf BGP)
*> 88.125.160.128/26          192.12.104.12            0      100      0 214 88 i
*> 88.125.160.192/26          192.12.104.12            0      100      0 214 88 i
```

```
R5#ping vrf BGP 88.125.160.129 source 192.104.5.5
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 8/9/11 ms
```

```
R5#ping vrf BGP 88.125.160.193 source 192.104.5.5
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 8/8/11 ms
```

We will examine the feature on XE as well, this time using IPv6 prefixes. Specifically, we will enable connectivity between XRv3's loopback and the /70's within AS 82 by only configuring CSR8. First, we will import XRv3's loopback into VRF BOTTOM using RT:214:104. We will do it incorrectly as we did on XR by trying to import the RT within the import-map; this is not supported on either platform.

```
! CSR8
ipv6 prefix-list PL_XRV13_LOOPBACK seq 5 permit ::192:104:13:13/128
ip extcommunity-list standard EXTCOML_RT_214_104 permit rt 214:104
```

```

route-map RM_VRF_IMPORT permit 10
  match extcommunity EXTCOML_RT_214_104
  match ipv6 address prefix-list PL_XRV13_LOOPBACK

vrf definition BOTTOM
  address-family ipv6
    import map RM_VRF_IMPORT

```

We can verify the configuration was successfully applied by checking the VRF details. BGP debugging reveals that the route is still rejected due to the import-RT policy not including RT:214:104. Even though the route-map clearly imports this RT, it is not accepted for VPN membership due to the order of operations. The import-map is only applied against routes already imported by the “route-target import” command.

```

R8#show vrf detail BOTTOM | begin v6
Address family ipv6 unicast (Table ID = 0x1E000001):
  Flags: 0x0
  Export VPN route-target communities
    RT:214:8
  Import VPN route-target communities
    RT:214:7 RT:214:103
  Import route-map: RM_VRF_IMPORT
  No global export route-map
  No export route-map
  VRF label distribution protocol: not configured
  VRF label allocation mode: vrf-conn-aggr (Label 8019)
Address family ipv4 multicast not active

```

```

R8#debug bgp vpnv6 unicast updates in
BGP(5): 214.0.0.3 rcvd [214:104]::192:104:13:13/128, label 91015 -- DENIED
due to: extended community not supported;

```

We can modify the configuration on CSR8 as we did on XRv2. We will remove the extcommunity-list containing the new RT and just import it the normal way. Then, we leave the filter in place to select the prefixes we want based on other attributes (other community values, prefix length, etc). This may create issues later when dealing with imports unrelated to RT:214:104, but we will continue anyway.

```

! CSR8
no ip extcommunity-list standard EXTCOML_RT_214_104 permit rt 214:104

route-map RM_VRF_IMPORT permit 10
  no match extcommunity EXTCOML_RT_214_104

vrf definition BOTTOM
  address-family ipv6
    route-target import 214:104

```

Now, we can see XRv3's loopback inside of the VRF table as expected. This means that the custom import was successful and that those other routes originating from AS 104 were filtered.

```
R8#show bgp vpnv6 unicast vrf BOTTOM quote-regexp "104$" | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 214:8 (default for vrf BOTTOM)
*>i  ::192:104:13:13/128
                                     ::FFFF:214.0.0.11
                                           0      100      0 104 ?
```

There is no reachability from AS 82 to XRv3's loopback since XRv1 is not importing RT:214:8, which CSR8 is exporting. We will configure CSR8 to export RT:214:104 so that XRv1 can import it. The danger here is that all PEs will import these routes, but will not have bidirectional connectivity since CSR8 only imported XRv3's loopback. In a real design, another RT would be used to exchange these routes, say RT:214:500, which XRv1 can import and CSR8 can append to the /70s it wants XRv1 to see. We will also take this opportunity to prove the order of operations when using export maps by intentionally forgetting the "additive" keyword.

```
! CSR8
ipv6 prefix-list PL_LENGTH_70 seq 5 permit ::/0 ge 70 le 70

route-map RM_VRF_EXPORT permit 10
 match ipv6 address prefix-list PL_LENGTH_70
 set extcommunity rt 214:104

vrf definition BOTTOM
 address-family ipv6
  export map RM_VRF_EXPORT
```

After applying this export, BGP needs to be cleared in order for the RT changes to have effect. First, we can validate that the configuration was successful by checking the VRF details.

```
R8#show vrf detail BOTTOM | begin v6
Address family ipv6 unicast (Table ID = 0x1E000001):
  Flags: 0x0
  Export VPN route-target communities
    RT:214:8
  Import VPN route-target communities
    RT:214:104
  Import route-map: RM_VRF_IMPORT
  No global export route-map
  Export route-map: RM_VRF_EXPORT
  VRF label distribution protocol: not configured
  VRF label allocation mode: vrf-conn-aggr (Label 8019)
Address family ipv4 multicast not active
```

Looking at the details of a /70 route, we can see the new RT:214:104 was applied, but the originally exported RT:214:8 was erased. This will allow all of the VRF BGP PE's to import this route, but CSR7 can no longer. This proves that the export-map happens after the "route-target export" command, overwriting whatever RTs were exported.

```
R8#show bgp vpnv6 unicast vrf BOTTOM 2BAD:82:125:160:400::/70
BGP routing table entry for [214:8]2BAD:82:125:160:400::/70, version 6
Paths: (1 available, best #1, table BOTTOM)
  Advertised to update-groups:
    5
  Refresh Epoch 1
  82
  FD00:10:1:8::1 (FE80::1) (via vrf BOTTOM) from 10.1.8.1 (82.125.160.1)
    Origin IGP, metric 0, localpref 100, valid, external, best
    Extended Community: RT:214:104
    mpls labels in/out 8016/nolabel
    rx pathid: 0, tx pathid: 0x0
```

This issue might be difficult to track down without regression testing, since reachability between XRv3 and CSR1 is now achieved, as shown below. XRv13 has the routes from AS 82, as expected, and has reachability as well.

```
RP/0/0/CPU0:XRv3#show bgp vrf BGP ipv6 unicast regexp "82$" | begin Network
      Network                Next Hop                Metric LocPrf Weight Path
Route Distinguisher: 214:104 (default for vrf BGP)
*> 2bad:82:125:160:400::/70
                                fd00:192:11:104::11
                                                0 214 82 i
*> 2bad:82:125:160:800::/70
                                fd00:192:11:104::11
                                                0 214 82 i
*> 2bad:82:125:160:c00::/70
                                fd00:192:11:104::11
                                                0 214 82 i

RP/0/0/CPU0:XRv3#ping vrf BGP 2BAD:82:125:160:400::1 source ::192:104:13:13
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/6/9 ms
```

Debugging on CSR7 reveals that the routes from AS 82 are now being denied as they no longer carry RT:214:8 which CSR7 expects to import. Other routes (the non /70 prefixes) are still imported to CSR7 since their RT values were never adjusted. The /72s are still successfully installed, for example.

```
R7#debug bgp vrf BGP vpnv6 unicast updates in
```

```
BGP(5): 214.0.0.12 rcvd [214:8]2BAD:82:125:160:C00::/70, label 8044 -- DENIED
due to: extended community not supported;
BGP(5): 214.0.0.12 rcvd [214:8]2BAD:82:125:160:800::/70, label 8043 -- DENIED
due to: extended community not supported;
BGP(5): 214.0.0.12 rcvd [214:8]2BAD:82:125:160:400::/70, label 8016 -- DENIED
due to: extended community not supported;
```

```
R7#show bgp vpv6 unicast vrf TOP quote-regexp "82$" | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 214:7 (default for vrf TOP)
*>i 2BAD:82:125:160::/72
      ::FFFF:214.0.0.8
                                0      100      0 82 i
*>i 2BAD:82:125:160:100::/72
      ::FFFF:214.0.0.8
                                0      100      0 82 i
*>i 2BAD:82:125:160:200::/72
      ::FFFF:214.0.0.8
                                0      100      0 82 i
*>i 2BAD:82:125:160:300::/72
      ::FFFF:214.0.0.8
                                0      100      0 82 i
```

To fix this, we can correct the route-map to use the “additive” keyword when setting a new RT. This is the most straightforward and logical way to solve the problem and is something we used on XRv2 earlier. However, I will solve it another way; perhaps we are not allowed to use the “additive” keyword. We can remove the generalized RT export from the VRF; we will apply the general export RT:214:8 to IPv4 only so that AFI is unaffected. Removing it from the top-level VRF configuration means the IPv6 AFI cannot inherit it. We will extend the IPv6 export route-map to include a default match-any clause that sets RT:214:8 on everything not previously matched. The /70s matched in the first clause are assigned both RT:214:104 and RT:214:8, ultimately receiving all the RTs they need. Non /70 routes only receive RT:214:8 as expected. We could have solved this by not having a second route-map clause and having left the “inheritance” RT inside the VRF as well, which would have been simpler.

```
! CSR8
vrf definition BOTTOM
  no route-target export 214:8
  address-family ipv4
    route-target export 214:8

route-map RM_VRF_EXPORT permit 10
  match ipv6 address prefix-list PL_LENGTH_70
  set extcommunity rt 214:8 214:104
route-map RM_VRF_EXPORT permit 20
  set extcommunity rt 214:8
```

We can verify this on CSR8 by looking at a /70 and /71, noticing that the /70 gets both RT values. One of the routes is eBGP learned while the other is a local aggregate; both are subject to the export-map adjustments.

```
R8#show bgp vpnv6 unicast vrf BOTTOM 2BAD:82:125:160:C00::/70
BGP routing table entry for [214:8]2BAD:82:125:160:C00::/70, version 8
Paths: (1 available, best #1, table BOTTOM)
  Advertised to update-groups:
    11
  Refresh Epoch 1
  82
  FD00:10:1:8::1 (FE80::1) (via vrf BOTTOM) from 10.1.8.1 (82.125.160.1)
    Origin IGP, metric 0, localpref 100, valid, external, best
    Extended Community: RT:214:8 RT:214:104
    mpls labels in/out 8016/nolabel
    rx pathid: 0, tx pathid: 0x0
```

```
R8#show bgp vpnv6 unicast vrf BOTTOM 2BAD:82:125:160:200::/71
BGP routing table entry for [214:8]2BAD:82:125:160:200::/71, version 21
Paths: (1 available, best #1, table BOTTOM)
  Advertised to update-groups:
    10      11
  Refresh Epoch 1
  Local, (aggregated by 214 214.0.0.8)
  :: (via default) from 0.0.0.0 (214.0.0.8)
    Origin IGP, localpref 100, weight 32768, valid, aggregated, local,
    atomic-aggregate, best
    Extended Community: RT:214:8
    mpls labels in/out IPv6 VRF Aggr:8019/aggregate(BOTTOM)
    rx pathid: 0, tx pathid: 0x0
```

Next, let's verify that CSR7 is able to import these routes. We don't need to see the route details; we can safely assume that so long as BGP selected a best-path, the routes are valid. The data-plane is expected to work since it worked for all other prefixes earlier.

```
R7#show bgp vpnv6 unicast vrf TOP quote-regexp "82$" | include /70
*>i 2BAD:82:125:160:400::/70
*>i 2BAD:82:125:160:800::/70
*>i 2BAD:82:125:160:C00::/70
```

There is one other issue with the RT inheritance on CSR8, and this is the result of an earlier change. When configuring RT import values at the top level and the AFI level, the AFI level takes precedence. Given this configuration, only RT:214:104 is imported into the IPv6 AFI. The RTs of 214:7 and 214:103 are no longer considered for import into the IPv6 AFI.

```
! CSR8
vrf definition BOTTOM
```

```
rd 214:8
route-target import 214:7
route-target import 214:103
address-family ipv6
    route-target import 214:104
```

```
R8#show vrf detail BOTTOM | begin v6
Address family ipv6 unicast (Table ID = 0x1E000001):
  Flags: 0x0
  No Export VPN route-target communities
  Import VPN route-target communities
    RT:214:104
  Import route-map: RM_VRF_IMPORT
  No global export route-map
  Export route-map: RM_VRF_EXPORT
  VRF label distribution protocol: not configured
  VRF label allocation mode: vrf-conn-aggr (Label 8019)
Address family ipv4 multicast not active
```

To correct this, we need to add RT:213:103 (for IS-IS testing completed earlier) and RT:214:7 (to import routes from CSR7) under the IPv6 AFI. Because the IPv4 AFI has not defined explicit RT-import policies, it can still inherit from the top-level configuration, so we won't delete those. We can tell this new configuration is working by looking at the VRF details.

```
! CSR8
vrf definition BOTTOM
rd 214:8
! route-target import 214:7
! route-target import 214:103
address-family ipv6
    route-target import 214:104
    route-target import 214:7
    route-target import 214:103
```

```
R8#show vrf detail BOTTOM | begin v6
Address family ipv6 unicast (Table ID = 0x1E000001):
  Flags: 0x0
  No Export VPN route-target communities
  Import VPN route-target communities
    RT:214:104          RT:214:7          RT:214:103
  Import route-map: RM_VRF_IMPORT
  No global export route-map
  Export route-map: RM_VRF_EXPORT
  VRF label distribution protocol: not configured
  VRF label allocation mode: vrf-conn-aggr (Label 8019)
Address family ipv4 multicast not active
```


Even after this, we still don't have the AS 88 routes in CSR8's VPN table anymore. The import-map we applied earlier is the problem; it indiscriminately drops anything that is not XRv3's loopback. We can tell by looking at the routes in the VPN table that are not from AS 82; there is only one. This is somewhat similar to the RPL on XRv2 where we needed to use multiple conditional matches to achieve the desired result.

```
R8#show bgp vpnv6 unicast vrf BOTTOM quote-regexp "[^82]" | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 214:8 (default for vrf BOTTOM)
*>i ::192:104:13:13/128
      ::FFFF:214.0.0.11
                                0      100      0 104 ?
```

Ideally, we need to allow all routes from AS 88. We have successfully imported RT:214:7 but the routes were filtered by the second-stage import-map. We can safely permit all routes from AS 88 because this route-map is only evaluated after the RT import process.

```
! CSR8
ip as-path access-list 88 permit 88$
```

```
route-map RM_VRF_IMPORT permit 10
  match ipv6 address prefix-list PL_XRV13_LOOPBACK
route-map RM_VRF_IMPORT permit 20
  match as-path 88
```

```
R8#show bgp vpnv6 unicast vrf BOTTOM quote-regexp "88$" | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 214:8 (default for vrf BOTTOM)
*>i 2BAD:88:125:160::/72
      ::FFFF:214.0.0.7
                                0      100      0 88 i
*>i 2BAD:88:125:160:100::/72
      ::FFFF:214.0.0.7
                                0      100      0 88 i
[snip]
```

Now that routing has been verified, we quickly test IPv6 reachability between AS 88 and XRv3's loopback. This proves that the "extranet" configuration was successful.

```
RP/0/0/CPU0:XRv3#ping vrf BGP 2BAD:82:125:160:400::1 source ::192:104:13:13
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/6/9 ms
```

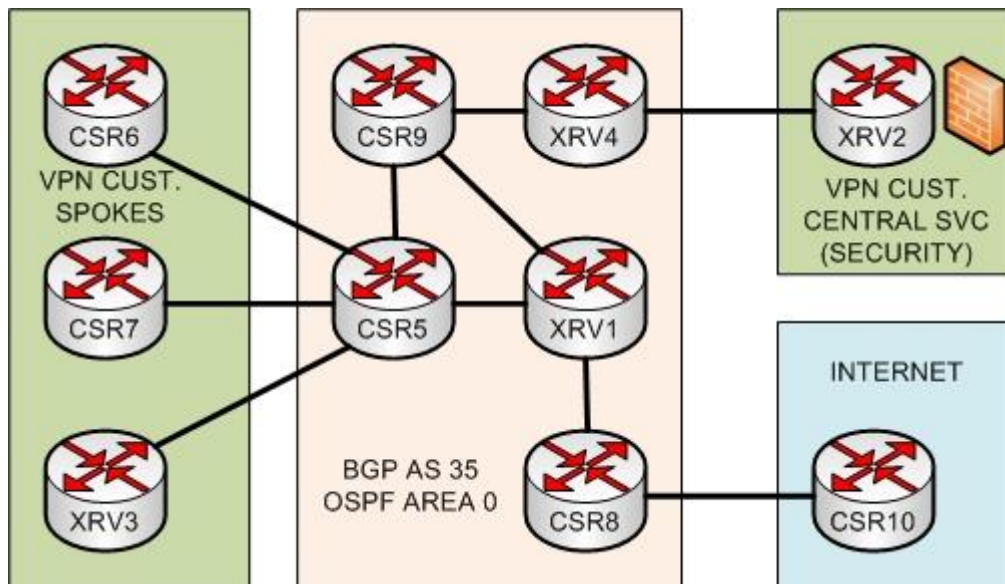
```
R10#ping 2BAD:82:125:160:C00::1 source 2BAD:88:125:160::1
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 6/6/7 ms
```

This section introduced many advanced L3VPN and VRF techniques. In summary:

1. Top-level VRF configuration of RTs in XE is convenient but confusing. XR doesn't even support it. Policies configured there are inherited to child AFIs only if the child AFIs don't have policies of their own. The inheritance is not additive and is overridden by the child policy as those are more specific. I recommend to only configure RTs at the AFI level for clarity and consistency with XR.
2. Import/export maps are processed after the import/export RT process, respectively. You cannot use the import-map to match RTs for import. The import-map applies to routes already passing the import-RT check and serves as a second-stage filter for additional granularity. This route-map/RPL has an implicit-deny at the end; it will reject non-matching routes from VRF import.
3. Export-map RT application will overwrite RTs written by "route-target export" unless the "additive" keyword is used. Prefixes not permitted by the export-map are not denied, but they are unmodified; they will have the normal export RTs configured in the VRF and nothing else. This can be considered an "implicit-pass" at the end of the route-map/RPL. This feature isn't meant for denying routes, only adjusting exported RTs.

38.6 Half-Duplex VRF (HDVRF)

The half-duplex VRF (HDVRF) feature is a minor option within the scope of L3VPN. Integrating this with the existing large-scale VRF lab previously would have been difficult, so a small lab has been developed specifically to test this feature. An HDVRF effectively defines two VRFs per interface: one upstream and one downstream. Like an isolated private VLAN, a single downstream VRF can service many customers but ensure they do not have direct connectivity across the local PE. Instead, traffic can hairpin across the network; although seemingly inefficient, this might be valuable for security reasons. Perhaps there is a centralized security checkpoint consisting of firewalls and intrusion prevention systems that inspects all spoke-to-spoke traffic on the network. HDVRF simplifies configuration on the local PE since two VRFs, the upstream and downstream VRFs, can service an unlimited number of customers without leaking traffic laterally. The feature is not currently supported on XR platforms or for IPv6 in general. Below is a basic L3VPN network a corporate headquarters site whereby XRv2 allows traffic to hairpin between spokes. The Internet peering point is access-only and spokes cannot use CSR10 for hairpinning traffic between one another. The core runs basic OSPFv2, BGP VPNv4, and LDP as shown below.



Before beginning, the parser returns an error when trying to configure HDVRF with IPv6. The parser tells you the VRF assignment is simply ignored. To avoid any unforeseen consequences of this unsupported configuration, the IPv6 AFI has been removed from all VRFs. There is no 6VPE in this lab, either.

```
R5(config-subif)#vrf forwarding UP downstream DOWN
% VRF forwarding ignored for IPv6 on HDVRF interface GigabitEthernet2.556
```

Before configuring the VRFs, we will perform a quick check of the existing IGP, LDP, and BGP infrastructure; the configurations are very basic and not shown for brevity. CSR5 sees 5 routers in the area with no DRs as all links are point-to-point. XRV1 and CSR9 each see 3 LDP neighbors which suggests all LDP sessions are up. XRV1 is the VPNv4 RR and has a valid BGP session to each PE.

```
R5#show ip ospf 35 0 database | begin ^Link
Link ID      ADV Router   Age         Seq#         Checksum Link count
35.0.0.5     35.0.0.5     204        0x80000017  0x004AEB  5
35.0.0.8     35.0.0.8     11         0x80000018  0x00AD29  3
35.0.0.9     35.0.0.9     1904       0x80000018  0x00EF5E  7
35.0.0.11    35.0.0.11    11         0x8000001B  0x005DED  7
35.0.0.14    35.0.0.14    1862       0x80000014  0x00E8D3  3
```

```
RP/0/0/CPU0:XRV1#show mpls ldp neighbor brief
Peer          GR  NSR  Up Time      Discovery   Addresses   Labels
              --  ---  -----
              ipv4  ipv6  ipv4  ipv6  ipv4  ipv6
-----
35.0.0.5:0    N  N    10:32:14     1    0     3     0     10    0
35.0.0.9:0    N  N    10:32:14     1    0     4     0     10    0
35.0.0.8:0    N  N    10:27:40     1    0     2     0     10    0
```

```
R9#show mpls ldp neighbor | include Peer
Peer LDP Ident: 35.0.0.5:0; Local LDP Ident 35.0.0.9:0
```

```
Peer LDP Ident: 35.0.0.11:0; Local LDP Ident 35.0.0.9:0
Peer LDP Ident: 35.0.0.14:0; Local LDP Ident 35.0.0.9:0
```

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast summary | begin ^Neigh
Neighbor      Spk    AS MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
35.0.0.5      0    35    706    659      24    0    0 10:30:48      0
35.0.0.8      0    35    699    641      24    0    0 10:28:20      0
35.0.0.14     0    35    635    641      24    0    0 10:29:32      0
```

CSR8 and XRv4 are basic L3VPN PE's with no fancy configuration. They use BGP to communicate with CSR10 and XRv2, respectively. The sessions are currently up and the PE's are learning routes from each CE eBGP peer. We can see the route counts: CSR8 learns one route while XRv4 learns several.

```
R8#show bgp vpnv4 unicast vrf CENTRAL summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down  State/PfxRcd
10.8.10.10    4      10     16     19      42    0    0 00:10:50      1
```

```
RP/0/0/CPU0:XRv4#show bgp vrf INTERNET summary | begin ^Neigh
Neighbor      Spk    AS MsgRcvd MsgSent   TblVer  InQ  OutQ  Up/Down  St/PfxRcd
10.12.14.12    0     12     27     27      26    0    0 00:23:09      7
```

CSR8 will only receive a default route from CSR10. The Internet default route is exported with RT:35:8; it makes sense that only a default route is provided for Internet access in this single-homed network.

```
R8#show bgp vpnv4 unicast vrf CENTRAL 0.0.0.0/0
BGP routing table entry for 35:100:0.0.0.0/0, version 18
Paths: (1 available, best #1, table CENTRAL)
  Advertised to update-groups:
    5
  Refresh Epoch 1
  10
    10.8.10.10 (via vrf CENTRAL) from 10.8.10.10 (10.10.10.10)
      Origin IGP, localpref 100, valid, external, best
      Extended Community: RT:35:8
      mpls labels in/out 8011/nolabel
      rx pathid: 0, tx pathid: 0x0
```

XRv4 learns intra-enterprise routes from XRv2. Because the spokes don't need all these routes, we use aggregation on XRv4. The AS-path information is applied to the aggregate which prevents AS 12 routers from accidentally installing it. Component routes are also suppressed and the aggregation configuration is shown for reference.

```
! XRv4
router bgp 35
  vrf CENTRAL
    address-family ipv4 unicast
      aggregate-address 12.24.0.0/16 as-set summary-only
```

```

RP/0/0/CPU0:XRv4#show bgp vpnv4 unicast vrf CENTRAL 12.24.0.0/16 | begin
Paths
Paths: (1 available, best #1)
  Advertised to peers (in unique update groups):
    35.0.0.11
  Path #1: Received by speaker 0
  Advertised to peers (in unique update groups):
    35.0.0.11
  12, (aggregated by 35 35.0.0.14)
    0.0.0.0 from 0.0.0.0 (35.0.0.14)
      Origin incomplete, localpref 100, weight 32768, valid, aggregated,
best, group-best, import-candidate
      Received Path ID 0, Local Path ID 1, version 63
      Extended community: RT:35:14
      Origin-AS validity: not-found (iBGP signalled)

```

We quickly check XRv1, the VPNv4 RR, to ensure it is learning and retaining these routes. This is a very clean way of providing Internet and Intranet connectivity to remote sites with only two routes.

```

RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast rd 35:100 | begin Network
  Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 35:100
*>i0.0.0.0/0        35.0.0.8              0      100      0 10 i
*>i12.24.0.0/16     35.0.0.14             100     0 12 ?

```

With the basic network verification complete, we will configure the HDVRF feature on CSR5. CSR6, CSR7, and XRv3 are spoke sites that all require Internet and Intranet access without direct spoke-to-spoke access. Without considering HDVRF, there are two general solutions to this problem:

1. Create three VRFs, each of which imports the RTs exported by CSR8 and XRv4, and exports a “spoke RT” that those routers can import. Spokes will have no connectivity unless the spoke routes are leaked somewhere upstream.
2. Create one VRFs which imports the RTs exported by CSR8 and XRv4 and exports a “spoke RT” that those routers can import. Spokes will have direct connectivity across CSR5 unless some kind of data-plane filter is applied.

Neither of these solutions work well in our environment to meet the requirements stated earlier. The first solution scales poorly since each spoke needs its own VRF, and spoke-to-spoke traffic is not inherent without additional work. The second is efficient from a forwarding perspective but “insecure” since XRv2 is the central point for all network security services. HDVRF is a combination of these two methods. We create two VRFs on CSR5 to represent upstream and downstream flows. Before adjusting any of the RT policies, we will configure the basic VRFs and apply them to the PE-CE interfaces.

```

! CSR5
vrf definition DOWN
  rd 35:6713

```

```

address-family ipv4

vrf definition UP
  rd 35:100
  address-family ipv4

interface GigabitEthernet2.553
  encapsulation dot1Q 3553
  vrf forwarding UP downstream DOWN

interface GigabitEthernet2.556
  vrf forwarding UP downstream DOWN

interface GigabitEthernet2.557
  vrf forwarding UP downstream DOWN

```

When we verify this configuration, we see that the DOWN VRF has a special “D” flag next to each interface. This specifies that the particular VRF is a downstream VRF on that interface. The lack of any flag specifies that the VRF is an upstream VRF. Below, we can see both UP and DOWN VRFs applied to the same interfaces.

```

R5#show vrf

```

Name	Default RD	Protocols	Interfaces
DOWN	35:6713	ipv4	Gi2.556 [D] Gi2.557 [D] Gi2.553 [D]
UP	35:100	ipv4	Gi2.556 Gi2.557 Gi2.553

We will need to exchange routing information with these spokes. Before we can do that, we must determine in which VRF the connected PE-CE subnets exist. This will determine in which VRF we must configure the PE-CE routing protocol. We quickly look at the VRF UP and DOWN routing tables to see that the connected routes are in the DOWN VRF. If we tried to configure BGP in the UP VRF, for example, there would be no route to the eBGP peer and the session would never form.

```

R5#show ip route vrf UP connected | include ^C_
[no output]

R5#show ip route vrf DOWN connected | include ^C_
C      10.5.6.0/24 is directly connected, GigabitEthernet2.556
C      10.5.7.0/24 is directly connected, GigabitEthernet2.557
C      10.5.13.0/24 is directly connected, GigabitEthernet2.553

```

Next, we configure BGP to each spoke. The BGP configuration is basic; the only noteworthy comment is that these sessions are in the DOWN VRF since that VRF contains the connected PE-CE subnets. We

quickly verify that the configuration was successful by verifying that the BGP sessions are up (CE configurations not shown).

```
! CSR5
router bgp 35
  template peer-session IPV4_UCAST_DOWN
    remote-as 6713
    password SPOKE
    ttl-security hops 1

  address-family ipv4 vrf DOWN
    neighbor 10.5.6.6 inherit peer-session IPV4_UCAST_DOWN
    neighbor 10.5.7.7 inherit peer-session IPV4_UCAST_DOWN
    neighbor 10.5.13.13 inherit peer-session IPV4_UCAST_DOWN
```

```
R5#show bgp vpnv4 unicast vrf DOWN summary | begin ^Neigh
Neighbor      V      AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
10.5.6.6      4      6713    756    765     81    0    0 11:17:40    1
10.5.7.7      4      6713    752    763     81    0    0 11:17:33    1
10.5.13.13    4      6713     49     72     81    0    0 00:45:15    1
```

Incoming routes learned by the PE are placed into the downstream VRF. This makes sense because BGP was configured in VRF DOWN and the connected subnets also exist there. A quick check of CSR5's VRF DOWN routing table reveals these three eBGP routes, one from each spoke.

```
R5#show ip route vrf DOWN bgp | include ^B_
B      100.6.6.6 [20/0] via 10.5.6.6, 00:54:27
B      100.7.7.7 [20/0] via 10.5.7.7, 00:54:27
B      100.13.13.13 [20/0] via 10.5.13.13, 00:54:25
```

Ultimately, both the Internet and Intranet routers will need reachability to these subnets, just like a normal L3VPN. Since the routes only exist in VRF DOWN, we will export RT:35:6713 onto these prefixes so that CSR8 and XRv4 can import them. Both the export and import processes are shown below.

```
! CSR5
vrf definition DOWN
  address-family ipv4
    route-target export 35:6713
```

```
! CSR8
vrf definition INTERNET
  address-family ipv4
    route-target import 35:6713
```

```
! XRv4
vrf CENTRAL
  address-family ipv4 unicast
```

```
import route-target
35:6713
```

We don't expect to have bidirectional connectivity since the routing advertisement is only complete in one direction. We verify that both CSR10 and XRv2 learn these spoke routes via BGP. From the perspective of CSR8 and XRv4, this is a normal L3VPN with no advanced features; they continue to import and export RTs as usual. The CENTRAL and INTERNET VRFs on these routers need not be HDVRF-aware since the feature is local to CSR5.

```
R10#show bgp ipv4 unicast 100.0.0.0/8 longer-prefixes | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
*> 100.6.6.6/32    10.8.10.8
*> 100.7.7.7/32    10.8.10.8
*> 100.13.13.13/32 10.8.10.8
                                0 35 6713 i
```

```
RP/0/0/CPU0:XRv2#show bgp ipv4 unicast 100.0.0.0/8 longer-prefixes | begin
Network
      Network      Next Hop      Metric LocPrf Weight Path
*> 100.6.6.6/32    10.12.14.14
*> 100.7.7.7/32    10.12.14.14
*> 100.13.13.13/32 10.12.14.14
                                0 35 6713 i
```

To complete the routing configuration, we introduce the half-duplex nature of the VRFs on CSR5.

Incoming traffic towards the PE will use the upstream VRF for its route lookup, which implies that VRF UP should be importing the remote L3VPN routes. In summary, VRF UP is importing upstream routes to control how clients route upstream. VRF DOWN is exporting downstream routes to control how upstream clients route back towards spokes. After importing the Internet and Intranet RTs into VRF UP on CSR5, we look at the VPNv4 routes across both VRFs. VRF UP has reachability to the Internet and Intranet sites, while VRF DOWN has reachability to the spokes. In a normal L3VPN configuration, one would think that these were entirely different topologies as reachability appears fragmented.

```
! CSR5
vrf definition UP
 address-family ipv4
   route-target import 35:8
   route-target import 35:14
```

```
R5#show bgp vpnv4 unicast all | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 35:100 (default for vrf UP)
*>i 0.0.0.0        35.0.0.8
*>i 12.24.0.0/16    35.0.0.14
                                0 100 0 10 i
                                100 0 12 ?
Route Distinguisher: 35:6713 (default for vrf DOWN)
*> 100.6.6.6/32    10.5.6.6
*> 100.7.7.7/32    10.5.7.7
*> 100.13.13.13/32 10.5.13.13
                                0
                                0 6713 i
                                0 6713 i
                                0 6713 i
```


At present, the spokes have no ability to reach any remote destinations since they haven't learned any routes from CSR5; VRF DOWN is not importing anything so nothing can be advertised to the spokes. As a quick fix, we will configure a static route on CSR6 only to achieve reachability.

```
! CSR6
ip route 0.0.0.0 0.0.0.0 10.5.6.5
```

When CSR6 sends traffic to CSR5, the FIB lookup happens in VRF UP regardless of whether VRF DOWN has a route or not. Two MPLS labels are imposed: the top label is XRv1's label for CSR8's loopback (transport) and the bottom label is CSR8's label for CSR10's loopback (VPN).

```
R5#show ip cef vrf DOWN 10.10.10.10
0.0.0.0/0
no route
```

```
R5#show ip cef vrf UP 10.10.10.10
0.0.0.0/0
  nexthop 35.5.11.11 GigabitEthernet2.551 label 91005 8010
```

```
R5#show mpls ldp bindings 35.0.0.8 32 neighbor 35.0.0.11
lib entry: 35.0.0.8/32, rev 20
  remote binding: lsr: 35.0.0.11:0, label: 91005
```

```
R5#show bgp vpnv4 unicast vrf UP labels
Network          Next Hop          In label/Out label
Route Distinguisher: 35:100 (UP)
  0.0.0.0          35.0.0.8          nolabel/8010
  12.24.0.0/16     35.0.0.14         nolabel/94023
```

XRv1 performs PHP and exposes label 8010 to CSR8. CSR8 removes all labels and delivers to the traffic to the Internet destination of 10.10.10.10. Once the label stack is imposed by CSR5, there is nothing special in the forwarding-plane.

```
RP/0/0/CPU0:XRv1#show mpls forwarding labels 91005
Local  Outgoing  Prefix          Outgoing  Next Hop        Bytes
Label  Label      or ID           Interface  Hop              Switched
-----
91005  Pop         35.0.0.8/32     Gi0/0/0/0.581 35.8.11.8      17575
```

```
R8#show mpls forwarding-table labels 8010 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
8010       No Label  0.0.0.0/0[V]    590          Gi2.580    10.8.10.10
MAC/Encaps=18/18, MRU=1504, Label Stack{}
005056A9F961005056A9FB1C81000DFC0800
VPN route: INTERNET
```

No output feature configured

Earlier, we confirmed that the Internet and Intranet sites already had ordinary L3VPN routes back to the spokes. When CSR10 replies, CSR8 imposes two labels in similar fashion to what CSR5 did above. Rather than trace the LSP from CSR8 to CSR5, we will skip to CSR5, which receives packets labeled 5004 after XRv1 performs PHP. CSR5 removes all labels and delivers to packets to CSR6 inside VRF DOWN. This makes sense because the VPN route originated inside VRF DOWN since there is where BGP was configured. It is also the VRF in which the connected next-hop resides.

```
R8#show ip cef vrf INTERNET 100.6.6.6/32
100.6.6.6/32
  nexthop 35.8.11.11 GigabitEthernet2.581 label 91000 5004
```

```
R5#show mpls forwarding-table labels 5004 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
5004       No Label  100.6.6.6/32 [V] 590          Gi2.556     10.5.6.6
          MAC/Encaps=18/18, MRU=1504, Label Stack{}
          005056A9DE0D005056A9DC6381000DE40800
          VPN route: DOWN
          No output feature configured
```

To ensure connectivity is functional, we quickly ping from CSR6 to CSR10 before continuing.

```
R6#ping 10.10.10.10 source 100.6.6.6
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.10.10.10, timeout is 2 seconds:
Packet sent with a source address of 100.6.6.6
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 8/9/11 ms
```

Rather than configure static routes on all spokes, we can advertise the expected BGP routes to them. This is accomplished by importing CSR8's and XRv4's RT values on VRF DOWN. Since VRF DOWN is never used for upstream route lookups, having these in the PE's VRF DOWN routing table means nothing. Of greater importance is being able to advertise them down to the spokes. Note that this step is optional and is not required if the CEs have other upstream routing mechanisms (static, etc), as we just demonstrated on CSR6. We quickly check to ensure the routes were imported.

```
! CSR5
vrf definition DOWN
  address-family ipv4
    route-target import 35:8
    route-target import 35:14
```

```
R5#show bgp vpnv4 unicast vrf DOWN quote-regexp "^[^6]" | begin Network
Network          Next Hop          Metric LocPrf Weight Path
```

```

Route Distinguisher: 35:6713 (default for vrf DOWN)
*>i 0.0.0.0 35.0.0.8 0 100 0 10 i
*>i 12.24.0.0/16 35.0.0.14 100 0 12 ?

```

Looking at the details of the default route as an example, we see interesting output. CSR5 claims that the route was imported directly from VRF UP, not from VPNv4 BGP. This is odd because VRF UP is not exporting RT:35:8 yet the output seems to suggest that. In reality, this means that VRF DOWN is “borrowing” the route from the upstream routing VRF so that it can advertise it downwards.

```

R5#show bgp vpnv4 unicast vrf DOWN 0.0.0.0/0
BGP routing table entry for 35:6713:0.0.0.0/0, version 21
Paths: (1 available, best #1, table DOWN)
  Advertised to update-groups:
    3
  Refresh Epoch 1
  10, imported path from 35:100:0.0.0.0/0 (UP)
    35.0.0.8 (metric 3) (via default) from 35.0.0.11 (35.0.0.11)
      Origin IGP, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:35:8
      Originator: 35.0.0.8, Cluster list: 35.0.0.11
      mpls labels in/out nlabel/8010
      rx pathid: 0, tx pathid: 0x0

```

As mentioned earlier, the local VRF DOWN RIB on CSR5 will now have these BGP routes, but they are irrelevant since they are not consulted when traffic is received from the CE. It is critical that the spokes learn these routes; we check CSR7 and XRv3 specifically. We also allowed the spokes to learn 12.24.0.0/16 since this was also “borrowed” from VRF UP to VRF DOWN. This is needed to route traffic towards the central services resources behind XRv2 in the case that the default route is not available.

```

R5#show ip route vrf DOWN bgp | include \[200
B* 0.0.0.0/0 [200/0] via 35.0.0.8, 00:04:50
B 12.24.0.0 [200/0] via 35.0.0.14, 00:04:50

R7#show ip route bgp | include ^B
B* 0.0.0.0/0 [20/0] via 10.5.7.5, 00:05:13
B 12.24.0.0 [20/0] via 10.5.7.5, 00:05:13

RP/0/0/CPU0:XRv3#show route bgp
B* 0.0.0.0/0 [20/0] via 10.5.13.5, 00:05:26
B 12.24.0.0/16 [20/0] via 10.5.13.5, 00:05:26

```

A quick set of tests confirms that this achieves connectivity between spokes and “hubs”. We don’t need to trace each LSP again since the forwarding plane is identical regardless of how the spokes learn their upstream routes.

```

R7#ping 10.10.10.10 source 100.7.7.7

```

```

[snip]
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 7/8/10 ms

R7#ping 12.24.48.1 source 100.7.7.7
[snip]
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 5/8/19 ms

RP/0/0/CPU0:XRv3#ping 10.10.10.10 source 100.13.13.13
[snip]
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/5/9 ms

RP/0/0/CPU0:XRv3#ping 12.24.48.1 source 100.13.13.13
[snip]
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/6/19 ms

```

Next, we will verify that hairpinning capability of HDVRF. Both CSR7 and XRv3 learn default routes from CSR5 inside the same VRF; under normal circumstances, traffic would be routed locally by CSR5 between these two endpoints. Since the routing lookup always happens in the upstream VRF, traffic will not be routed directly. In this example, we will trace the path from CSR7 to XRv3. When CSR7 sends packets to CSR5 for this destination it follows its default route.

```

R7#show ip cef 100.13.13.13
0.0.0.0/0
  nexthop 10.5.7.5 GigabitEthernet2.557

```

CSR5 performs a lookup in VRF UP and also finds that the default route is the best match. Traffic is tunneled inside MPLS all the way to CSR10, since that is where the default route was originated.

```

R5#show ip cef vrf UP 100.13.13.13
0.0.0.0/0
  nexthop 35.5.11.11 GigabitEthernet2.551 label 91005 8010

```

CSR10's longest-match for XRv3 is via CSR8 following a host route. Since CSR8 imported those downstream routes, but VRF UP didn't do it locally on CSR5, we have effectively created a large hairpin terminating on CSR10.

```

R10#show ip cef 100.13.13.13
100.13.13.13/32
  nexthop 10.8.10.8 GigabitEthernet2.580

```

CSR8 tunnels traffic inside MPLS back to CSR5 for forwards it to XRv3 inside VRF DOWN.

```
R8#show ip cef vrf INTERNET 100.13.13.13/32
```

```
100.13.13.13/32
```

```
nexthop 35.8.11.11 GigabitEthernet2.581 label 91000 5009
```

```
R5#show mpls forwarding-table labels 5009 detail
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Outgoing interface	Next Hop
5009	No Label	100.13.13.13/32[V]	\		
			1888	Gi2.553	10.5.13.13

MAC/Encaps=18/18, MRU=1504, Label Stack{}

005056A9EA54005056A9DC6381000DE10800

VPN route: DOWN

No output feature configured

Using traceroute inside CSR7 reveals the entire path. CSR10 terminates the hairpin and is highlighted, but it is clear that traffic between spokes cannot be routed locally by CSR5.

```
R7#traceroute 100.13.13.13 source 100.7.7.7
```

```
Type escape sequence to abort.
```

```
Tracing the route to 100.13.13.13
```

```
VRF info: (vrf in name/id, vrf out name/id)
```

```
 1 10.5.7.5 [AS 10] 4 msec 4 msec 3 msec
 2 35.5.11.11 [AS 10] [MPLS: Labels 91005/8010 Exp 0] 7 msec 5 msec 6 msec
 3 10.8.10.8 [AS 10] [MPLS: Label 8010 Exp 0] 16 msec 16 msec 15 msec
 4 10.8.10.10 [AS 10] 19 msec 11 msec 10 msec
 5 10.8.10.8 [AS 10] 10 msec 11 msec 10 msec
 6 35.8.11.11 [AS 10] [MPLS: Labels 91000/5009 Exp 0] 32 msec 51 msec 50 msec
 7 10.5.13.5 [AS 10] [MPLS: Label 5009 Exp 0] 21 msec 19 msec 18 msec
 8 10.5.13.13 [AS 10] 32 msec 28 msec 27 msec
```

This behavior is highly undesirable at present. Recall that CSR10 represents the Internet while XRv2 represents the central services site containing the security tools. Traffic is being hairpinned via an Internet router and totally bypassing the security features at XRv2. This is the result of VRF UP only having a default route to reach the spoke loopbacks. Instead, we must find a way to attract spoke-to-spoke traffic on XRv2. The simplest solution is a static null route advertised into BGP on XRv2. After configuring it, we verify that XRv1 (RR) learns this prefix.

```
! XRv2
router static
  address-family ipv4 unicast
    100.0.0.0/8 Null0

router bgp 12
  address-family ipv4 unicast
    network 100.0.0.0/8
```

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast rd 35:100 100.0.0.0/8 brief | begin
Network
  Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 35:100
*>i100.0.0.0/8      35.0.0.14          0      100      0 12 i
```

We quickly verify that all three spokes learn this prefix. Although the next-hop is the same as the default route, all of them will be following this new route to reach one another's loopbacks. I use three different verifications on the spokes for variety.

```
R6#show ip route 100.0.0.0 255.0.0.0
Routing entry for 100.0.0.0/8
  Known via "bgp 6713", distance 20, metric 0
  Tag 35, type external
  [snip]
```

```
R7#show ip cef 100.0.0.0/8
100.0.0.0/8
  nexthop 10.5.7.5 GigabitEthernet2.557
```

```
RP/0/0/CPU0:XRv3#show bgp ipv4 unicast 100.0.0.0/8 brief | begin Network
  Network          Next Hop          Metric LocPrf Weight Path
*> 100.0.0.0/8      10.5.13.5          0      35 12 i
```

Of paramount importance is the upstream VRF routing lookup on CSR5. Since the static null route on XRv2 is generic enough to cover all spoke loopbacks, yet more specific than the default route, it is preferred. Now traffic will hairpin through XRv2 as desired. Without performing a detailed verification, we know that label 94008 was allocated by XRv4; in this case, it represents XRv4's local VPN label for prefix 100.0.0.0/8.

```
R5#show ip cef vrf UP 100.6.6.6
100.0.0.0/8
  nexthop 35.5.9.9 GigabitEthernet2.559 label 9010 94008
```

```
R5#show ip cef vrf UP 100.7.7.7
100.0.0.0/8
  nexthop 35.5.9.9 GigabitEthernet2.559 label 9010 94008
```

```
R5#show ip cef vrf UP 100.13.13.13
100.0.0.0/8
  nexthop 35.5.9.9 GigabitEthernet2.559 label 9010 94008
```

Using traceroute from CSR7 again verifies this behavior. XRv2 is highlighted in the output below. At this point the HDVRF configuration is complete.

```
R7#traceroute 100.13.13.13 source 100.7.7.7
```

Type escape sequence to abort.

Tracing the route to 100.13.13.13

VRF info: (vrf in name/id, vrf out name/id)

```
 1 10.5.7.5 [AS 10] 4 msec 2 msec 2 msec
 2 35.5.9.9 [AS 10] [MPLS: Labels 9010/94008 Exp 0] 6 msec 6 msec 11 msec
 3 35.9.14.14 [AS 10] [MPLS: Label 94008 Exp 0] 19 msec 21 msec 20 msec
 4 10.12.14.12 [AS 10] 20 msec 15 msec 14 msec
 5 10.12.14.14 [AS 10] 14 msec 14 msec 15 msec
 6 35.9.14.9 [AS 10] [MPLS: Labels 9001/5009 Exp 0] 45 msec 56 msec 51 msec
 7 10.5.13.5 [AS 10] [MPLS: Label 5009 Exp 0] 22 msec 8 msec 11 msec
 8 10.5.13.13 [AS 10] 25 msec 27 msec 29 msec
```

Earlier I briefly discussed the need for the spokes to learn the prefix 12.24.0.0/16 despite having a default route. To prove its value, we temporarily shut down CSR10's BGP session to CSR8 (not shown). Because CSR10 represents the Internet, we fully expect that the spokes will no longer have Internet reachability. They should, however, continue to have central services and "hairpin" services via XRv2. Checking CSR7, we can see that it retains both of these BGP routes but no longer learns a default dynamically. Without this route, even if the Internet was operational, the central services subnets would be unreachable. This is more of a basic L3VPN/routing issue than an HDVRF issue, but integrating an intelligent routing design with HDVRF is important.

```
R7#show ip route bgp | include ^B
B      12.24.0.0 [20/0] via 10.5.7.5, 00:34:33
B      100.0.0.0/8 [20/0] via 10.5.7.5, 00:10:44
```

CSR7 will not have Internet access since there is no route, but has full access to XRv2's services. We can tell by the long latency on the spoke-to-spoke ping that traffic is still being hairpinned via XRv2.

```
R7#show ip cef 10.10.10.10
0.0.0.0/0
  no route
```

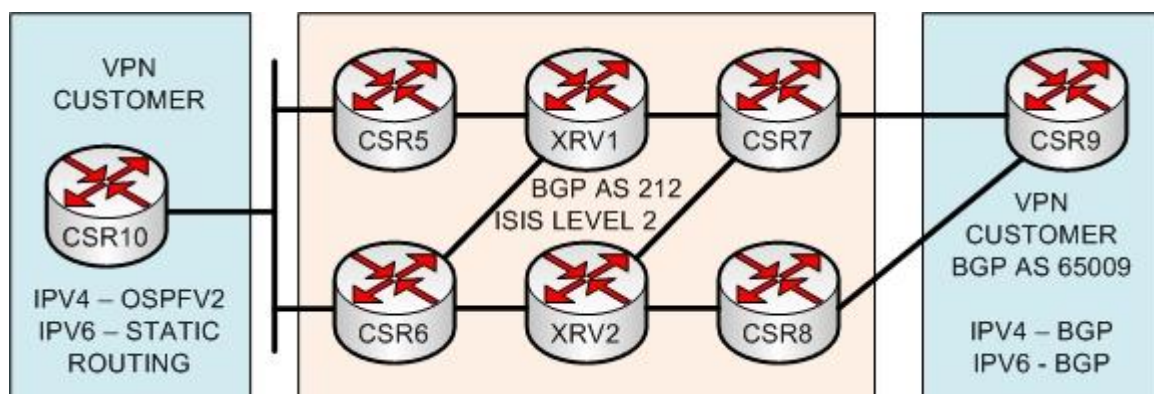
```
R7#ping 100.13.13.13 source 100.7.7.7
[snip]
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 9/25/39 ms
```

```
R7#ping 12.24.48.1 source 100.7.7.7
[snip]
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/6/13 ms
```

Additional Reading – Reference configurations "hdvrf"

38.7 BGP Local Convergence (VRF Local Protection)

This feature is difficult to classify by name since it is a combination of BGP, HA, L3VPN, and VRF features. It is designed to speed up BGP convergence on a local egress PE by protecting against a PE-CE link failure. Normally, when the PE-CE link fails, the eBGP session running over that link also fails. This will occur as a result of the “fast-external-falover” feature enabled by default if the link drops line-protocol. If the line-protocol doesn’t fail, such as would be the case in many Ethernet last-mile deployments, the BGP keepalive timer (or possibly BFD timer) must expire. Once this happens, the PE that suffered the link failure will withdraw the prefix from the BGP topology, which typically involves sending the MP_UNREACH_NLRI towards the RR. Assuming that BGP convergence is slow (that is to say, it takes time for the remote PEs to learn this update to select better paths), this can result in a significant amount of dropped traffic. Like TE-FRR in a way, VPN traffic destined for hosts behind the PE suffering the link failure will still be sent towards that PE. Here, the traffic would typically be blackholed while BGP was converging. With local protection, this egress PE will swap the VPN label to the VPN label of the backup egress PE and send the packet back into the MPLS core, effectively performing local repair until BGP converges. There are two key network design requirements in order to use this feature: the CE site must be multi-homed to at least two different PEs, and the primary egress PE must learn the backup PE’s VPN route. The second requirement can be achieved many ways: BGP additional-paths, diverse RDs, etc. The network diagram is shown below. CSR9 and CSR10 are in the same VPN and each one is multi-homed to two PEs.



Before beginning any advanced configuration, we verify the basic IGP, LDP, and BGP components. IS-IS level-2 is running on all nodes. A quick check of XRv1’s LSPDB shows 6 LSPs, each with the appropriate number of transit links. We also verify that each LSR advertises a single loopback into IGP. This quick check verifies the entire IGP topology quickly.

```
RP/0/0/CPU0:XRv1#show isis database detail | include "[RX]|Extended"
R5.00-00          0x0000000e  0x9ec1          1015          0/0/0
  Metric: 10      IS-Extended XRv1.00
  Metric: 0       IP-Extended 212.0.0.5/32
R6.00-00          0x0000000d  0x96a5          697          0/0/0
  Metric: 10      IS-Extended XRv2.00
  Metric: 10      IS-Extended XRv1.00
  Metric: 0       IP-Extended 212.0.0.6/32
```



```

R7.00-00          0x00000009  0x023a          443          0/0/0
  Metric: 10      IS-Extended XRv2.00
  Metric: 10      IS-Extended XRv1.00
  Metric: 0       IP-Extended 212.0.0.7/32
R8.00-00          0x00000009  0xa4b3          887          0/0/0
  Metric: 10      IS-Extended XRv2.00
  Metric: 0       IP-Extended 212.0.0.8/32
XRv1.00-00        * 0x0000000c  0xef36          1123         0/0/0
  Metric: 10      IS-Extended R6.00
  Metric: 10      IS-Extended R5.00
  Metric: 10      IS-Extended R7.00
  Metric: 0       IP-Extended 212.0.0.11/32
XRv2.00-00        0x00000009  0xd150          951          0/0/0
  Metric: 10      IS-Extended R6.00
  Metric: 10      IS-Extended R8.00
  Metric: 10      IS-Extended R7.00
  Metric: 0       IP-Extended 212.0.0.12/32

```

Since there are 6 LSRs in the network, we expect to see exactly 6 loopbacks. This is because each LSR is only allocating labels for host routes. Checking the core routers, we can see all LDP sessions are up with 6 IPv4-based labels exchanged between each set of peers.

```
RP/0/0/CPU0:XRv1#show mpls ldp neighbor brief
```

Peer	GR	NSR	Up Time	Discovery		Addresses		Labels	
				ipv4	ipv6	ipv4	ipv6	ipv4	ipv6
212.0.0.6:0	N	N	01:10:32	1	0	3	0	6	0
212.0.0.5:0	N	N	01:10:26	1	0	2	0	6	0
212.0.0.7:0	N	N	01:09:22	1	0	3	0	6	0

```
RP/0/0/CPU0:XRv2#show mpls ldp neighbor brief
```

Peer	GR	NSR	Up Time	Discovery		Addresses		Labels	
				ipv4	ipv6	ipv4	ipv6	ipv4	ipv6
212.0.0.6:0	N	N	01:12:04	1	0	3	0	6	0
212.0.0.7:0	N	N	01:12:04	1	0	3	0	6	0
212.0.0.8:0	N	N	01:12:04	1	0	2	0	6	0

XRv1 is the RR for both VPNv4 and VPNv6. We quickly verify that both sessions are up to all 4 PEs. Ignore the route counts for now as those will make sense once we verify VPN connectivity.

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast summary | begin ^Neigh
```

Neighbor	Spk	AS	MsgRcvd	MsgSent	ThlVer	InQ	OutQ	Up/Down	St/PfxRcd
212.0.0.5	0	212	96	137	45	0	0	01:12:46	1
212.0.0.6	0	212	92	136	45	0	0	01:12:33	1
212.0.0.7	0	212	118	141	45	0	0	00:27:34	1
212.0.0.8	0	212	115	139	45	0	0	00:27:39	1

```
RP/0/0/CPU0:XRv1#show bgp vpv6 unicast summary | begin ^Neigh
```

Neighbor	Spk	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	St/PfxRcd
212.0.0.5	0	212	96	137	38	0	0	01:12:49	1
212.0.0.6	0	212	92	136	38	0	0	01:12:35	1
212.0.0.7	0	212	118	141	38	0	0	00:27:37	1
212.0.0.8	0	212	115	139	38	0	0	00:27:41	1

Next, we quickly verify the PE-CE routing. The local protection feature supports all protocols (including static routing) except for IS-IS within the IPv4 AFI, while IPv6 only supports BGP and static routing. To test as many combinations as possible, we will use a variety of PE-CE configurations. CSR9 will use BGP for both IPv4 and IPv6. The configuration is not shown since it is very basic, but we quickly verify that all sessions are functional from the perspective of CSR9. The only interesting configuration is that CSR9 is configured to always compared BGP RID; this ensures that CSR7 is always the primary next-hop since the eBGP oldest-route comparison is skipped.

```
! CSR9
router bgp 65009
  bgp bestpath compare-routerid
```

```
R9#show bgp ipv4 unicast summary | begin ^Neigh
```

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
10.7.9.7	4	212	29	31	31	0	0	00:20:45	1
10.8.9.8	4	212	43	46	31	0	0	00:31:29	1

```
R9#show bgp ipv6 unicast summary | begin ^Neigh
```

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
FD00:10:7:9::7	4	212	29	30	34	0	0	00:20:49	1
FD00:10:8:9::8	4	212	42	45	34	0	0	00:31:30	1

CSR10 will use OSPFv2 for IPv4 and static routing for IPv6. Both CSR5 and CSR6, as PEs, are never allowed to be the OSPF DR. If CSR10 fails, there is no reason for CSR5 and CSR6 to form a neighborhood as no routes need to be exchanged laterally. The IPv6 static routing is basic; note that the PE-CE LAN segment does not have any routable IPv6 addressing as it is not required. CSR10 uses different administrative distances for selection. CSR5 is always the primary next-hop as a result.

```
! CSR5 and CSR6
interface GigabitEthernet2.550
  ip ospf priority 0

ipv6 route vrf 910 910:10:10:10::10/128 GigabitEthernet2.550 FE80::10

! CSR10
ipv6 route ::/0 GigabitEthernet2.550 FE80::6 8
ipv6 route ::/0 GigabitEthernet2.550 FE80::5 4
```

Because the local protection feature requires the primary PE (CSR5 and CSR7 in this case) to learn the iBGP route via the backup PE, we used unique RDs on all PEs. BGP additional-paths would have been

appropriate also, but given XE's limited support for this feature within VPNv4/v6, unique RD was a simpler solution. We quickly check XRv1, the RR, to ensure that each PE is using a different RD. CSR9 and CSR10 contribute one route each as shown below. It is between these customer host addresses that we will test local protection. The RDs are highlighted for clarity; they key is that they are all different.

```
RP/0/0/CPU0:XRv1#show bgp vpnv4 unicast | begin Network
  Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 212:9105
*>i9.10.0.10/32      212.0.0.5      20    100    0 ?
Route Distinguisher: 212:9106
*>i9.10.0.10/32      212.0.0.6      20    100    0 ?
Route Distinguisher: 212:9107
*>i9.10.0.9/32       212.0.0.7      0     100    0 65009 ?
Route Distinguisher: 212:9108
*>i9.10.0.9/32       212.0.0.8      0     100    0 65009 ?

RP/0/0/CPU0:XRv1#show bgp vpnv6 unicast | begin Network
  Network      Next Hop      Metric LocPrf Weight Path
Route Distinguisher: 212:9105
*>i910:10:10:10::10/128
                        212.0.0.5      0     100    0 ?
Route Distinguisher: 212:9106
*>i910:10:10:10::10/128
                        212.0.0.6      0     100    0 ?
Route Distinguisher: 212:9107
*>i910:9:9:9::9/128  212.0.0.7      0     100    0 65009 ?
Route Distinguisher: 212:9108
*>i910:9:9:9::9/128  212.0.0.8      0     100    0 65009 ?
```

Because all of our PE-CE links are Ethernet-based, BGP's fast-external-fallover will not be valuable for detecting PE-CE link failures in a timely manner. We will use BFD to achieve less than 3 seconds detection time; the relatively slow timer is used given the CSR1000v's 100 kbps rate-limit. The BFD configuration is not discussed in detail since there is a dedicated chapter for it. For brevity, only the CE configurations are shown, but the BFD configurations are very similar on both ends of the PE-CE link.

```
! CSR9
router bgp 65009
 neighbor 10.7.9.7 fall-over bfd single-hop
 neighbor 10.8.9.8 fall-over bfd single-hop
 neighbor FD00:10:7:9::7 fall-over bfd single-hop
 neighbor FD00:10:8:9::8 fall-over bfd single-hop

interface GigabitEthernet2.579
 bfd interval 900 min_rx 900 multiplier 3

interface GigabitEthernet2.589
 bfd interval 900 min_rx 900 multiplier 3
```

```

! CSR10
interface GigabitEthernet2.550
  bfd interval 900 min_rx 900 multiplier 3

router ospf 910
  bfd all-interfaces

ipv6 route static bfd GigabitEthernet2.550 FE80::5
ipv6 route static bfd GigabitEthernet2.550 FE80::6

```

From the CE routers, we quickly verify that BFD is operational. Each CE should have 4 operational BFD sessions, two per AFI.

```
R9#show bfd neighbors
```

```
IPv4 Sessions
```

NeighAddr	LD/RD	RH/RS	State	Int
10.7.9.7	4097/4097	Up	Up	Gi2.579
10.8.9.8	4101/4097	Up	Up	Gi2.589

```
IPv6 Sessions
```

NeighAddr	LD/RD	RH/RS	State	Int
FD00:10:7:9::7	2/1	Up	Up	Gi2.579
FD00:10:8:9::8	1/2	Up	Up	Gi2.589

```
R10#show bfd neighbors
```

```
IPv4 Sessions
```

NeighAddr	LD/RD	RH/RS	State	Int
10.5.10.5	4097/4097	Up	Up	Gi2.550
10.5.10.6	4098/4097	Up	Up	Gi2.550

```
IPv6 Sessions
```

NeighAddr	LD/RD	RH/RS	State	Int
FE80::5	1/1	Up	Up	Gi2.550
FE80::6	2/1	Up	Up	Gi2.550

Quickly note that a positive side effect of removing CSR5 and CSR6 from the DR election is that they do not establish a BFD session with one another. OSPF only establishes BFD sessions to the DR and BDR as discussed in the BFD chapter.

```
R6#show bfd neighbors client ospf
```

```
IPv4 Sessions
```

NeighAddr	LD/RD	RH/RS	State	Int
10.5.10.10	4097/4098	Up	Up	Gi2.550

To test connectivity over the basic L3VPN, we use traceroute for brevity. Notice that CSR9 will always use CSR7 and CSR10 will always use CSR5, assuming CSR7 and CSR5 are up. These PEs are used for

ingress and egress by default; when comparing iBGP paths, the ingress PEs will select the egress PE with the lower BGP RID. This was done intentionally to converge the network deterministically and without advanced BGP best-path adjustments.

```
R10#traceroute 9.10.0.9 source 9.10.0.10
Type escape sequence to abort.
Tracing the route to 9.10.0.9
VRF info: (vrf in name/id, vrf out name/id)
 1 10.5.10.5 6 msec 4 msec 4 msec
 2 212.5.11.11 [MPLS: Labels 91003/7008 Exp 0] 8 msec 8 msec 7 msec
 3 10.7.9.7 [MPLS: Label 7008 Exp 0] 16 msec 16 msec 14 msec
 4 10.7.9.9 20 msec 11 msec 14 msec
```

```
R9#traceroute 9.10.0.10 source 9.10.0.9
Type escape sequence to abort.
Tracing the route to 9.10.0.10
VRF info: (vrf in name/id, vrf out name/id)
 1 10.7.9.7 5 msec 5 msec 3 msec
 2 212.7.11.11 [MPLS: Labels 91000/5001 Exp 0] 8 msec 8 msec 8 msec
 3 10.5.10.5 [MPLS: Label 5001 Exp 0] 10 msec 16 msec 13 msec
 4 10.5.10.10 22 msec 15 msec 10 msec
```

Before enabling local protection on the PEs, we will verify that the primary PEs (CSR5 and CSR7), at a minimum, have the iBGP paths from the backup PEs (CSR6 and CSR8). These do not need to be installed as “BGP backup paths” in the literal sense, but the routers need to have them. For variety, we check the local CE route on CSR5 using IPv4 and CSR7 using IPv6. The key word is “local”; this feature only provides FRR service for traffic egressing the core towards the customer. The primary paths are highlighted in yellow and the backup paths are in green. Notice that the label operations for the primary path is a basic imposition, while the backup path requires a VPN label swap. This makes sense since the backup PE changed the VPN next-hop, so BGP must perform a label swap to that backup PE’s local label. This label swapping intelligence has nothing to do with local protection as this is a fundamental component of BGP/MPLS interaction.

```
R5#show bgp vpnv4 unicast vrf 910 9.10.0.10/32
BGP routing table entry for 212:9105:9.10.0.10/32, version 35
Paths: (2 available, best #2, table 910)
  Advertised to update-groups:
    1
  Refresh Epoch 1
  Local, imported path from 212:9106:9.10.0.10/32 (global)
    212.0.0.6 (metric 20) (via default) from 212.0.0.11 (212.0.0.11)
      Origin incomplete, metric 20, localpref 100, valid, internal
      Extended Community: RT:212:910 OSPF DOMAIN ID:0x0005:0x0000038E0200
        OSPF RT:0.0.0.0:5:1 OSPF ROUTER ID:10.5.10.6:0
      Originator: 212.0.0.6, Cluster list: 212.0.0.11
      mpls labels in/out 5001/6033
```

```

    rx pathid: 0, tx pathid: 0
Refresh Epoch 1
Local
10.5.10.10 (via vrf 910) from 0.0.0.0 (212.0.0.5)
    Origin incomplete, metric 20, localpref 100, weight 32768, valid,
sourced, best
    Extended Community: RT:212:910 OSPF DOMAIN ID:0x0005:0x0000038E0200
    OSPF RT:0.0.0.0:5:1 OSPF ROUTER ID:10.5.10.5:0
    mpls labels in/out 5001/nolabel
    rx pathid: 0, tx pathid: 0x0

R7#show bgp vpnv6 unicast vrf 910 910:9:9:9::9/128
BGP routing table entry for [212:9107]910:9:9:9::9/128, version 38
Paths: (2 available, best #1, table 910)
    Advertised to update-groups:
        5
Refresh Epoch 2
65009
    FD00:10:7:9::9 (FE80::9) (via vrf 910) from FD00:10:7:9::9 (9.10.0.9)
    Origin incomplete, metric 0, localpref 100, valid, external, best
    Extended Community: RT:212:910
    mpls labels in/out 7001/nolabel
    rx pathid: 0, tx pathid: 0x0
Refresh Epoch 1
65009, imported path from [212:9108]910:9:9:9::9/128 (global)
    ::FFFF:212.0.0.8 (metric 20) (via default) from 212.0.0.11 (212.0.0.11)
    Origin incomplete, metric 0, localpref 100, valid, internal
    Extended Community: RT:212:910
    Originator: 212.0.0.8, Cluster list: 212.0.0.11
    mpls labels in/out 7001/8015
    rx pathid: 0, tx pathid: 0

```

Before enabling local protection, we will break the link between CSR7 and CSR9 by changing the VLAN encapsulation (not shown). BGP update debugging is enabled on CSR7 for VPNv4 and VPNv6. Although CSR7 immediately installs the backup VPN route via CSR8, it withdraws its own route immediately to trigger a BGP convergence event. This is a good thing; there is no reason for CSR7 to continue advertising its ability to be an egress PE since this is no longer true.

```

R7#debug bgp vpnv4 unicast updates out
BGP updates debugging is on (outbound) for address family: VPNv4 Unicast

BGP(4): Revise route installing 1 of 1 routes for 9.10.0.9/32 ->
212.0.0.8(910) to 910 IP table
BGP(4): (base) 212.0.0.11 send unreachable (format) 212:9107:9.10.0.9/32
BGP(4): 212.0.0.11 rcv UPDATE about 212:9107:9.10.0.9/32 -- withdrawn, label
524288

R7#debug bgp vpnv6 unicast updates out

```

BGP updates debugging is on (outbound) for address family: VPNv4 Unicast
VPNv6 Unicast

```
BGP(5): Revise route installing [212:9107]910:9:9:9::9/128 ->
::FFFF:212.0.0.8 (::) to IPv6 910 table
BGP(5): (base) 212.0.0.11 send unreachable (format)
[212:9107]910:9:9:9::9/128
BGP(5): 212.0.0.11 rcv UPDATE about [212:9107]910:9:9:9::9/128 -- withdrawn,
label 524288
```

The fundamental issue is more subtle; CSR7 also immediately deallocates its local labels for those prefixes since it does not consider itself an ingress PE at all. This means that any traffic that may already be transiting the core will be dropped when CSR7 receives it, causing a short-term black hole. Until BGP converges around the failure, CSR7 will continue to drop packets received from the core. No matter how fast BGP converges, the packets that are already transiting the core would still be dropped.

```
R7#show bgp vpnv4 unicast vrf 910 labels
  Network          Next Hop      In label/Out label
Route Distinguisher: 212:9107 (910)
  9.10.0.9/32       212.0.0.8       nolabel/8012
  9.10.0.10/32      212.0.0.6       nolabel/6033
                   212.0.0.5       nolabel/5001
```

```
R7#show bgp vpnv6 unicast vrf 910 labels
  Network          Next Hop      In label/Out label
Route Distinguisher: 212:9107 (910)
  910:9:9:9::9/128  ::FFFF:212.0.0.8 nolabel/8015
  910:10:10:10::10/128
                        ::FFFF:212.0.0.6 nolabel/6020
                        ::FFFF:212.0.0.5 nolabel/5028
```

Local protection doesn't change the operation of BGP routing so much as it changes the label retention mechanism. Rather than immediately deallocating the local label for that prefix, the egress PE will retain it for 5 minutes, giving BGP plenty of time to converge. Packets that arrive on the egress PE will have their VPN label swapped per the LFIB, and any additional transport labels will be pushed as well. This behavior is similar to the PLR in TE-FRR and is only meant to be a temporary backup while BGP converges. The feature does not have any knobs and is simple to enable for IPv4 and IPv6; we enable it on all PEs. Technically, the feature only needs to be enabled on the primary PE, but we can enable it everywhere in case we decide to modify the BGP best-path selection algorithm later to prefer CSR6 or CSR8. The feature isn't explicitly supported in XR, but using BGP additional-paths, you could protect PE-CE links as seen in the "add-path" section. Once configured, we verify it by checking the VRF details.

! CSR5, CSR6, CSR7, and CSR8

```

vrf definition 910
  address-family ipv4
    protection local-prefixes
  address-family ipv6
    protection local-prefixes

R7#show vrf detail 910 | include ^Addr|Local
Address family ipv4 unicast (Table ID = 0x2):
  Local prefix protection enabled
Address family ipv6 unicast (Table ID = 0x1E000001):
  Local prefix protection enabled
Address family ipv4 multicast not active

```

Because BGP converges extremely fast in such a small network with so few prefixes, we need to slow it down. Using the “advertisement-interval” on each PE towards the RR, we can effectively limit how frequently the PEs can send successive updates/withdrawals across the network. We will introduce a long timer of 180 seconds (3 minutes) onto each PE. This means that the failure of a PE-CE link could take up to 3 minutes to converge, giving us time to verify that local protection works.

```

! CSR5, CSR6, CSR7, and CSR8
router bgp 212
  address-family vpnv4
    neighbor 212.0.0.11 advertisement-interval 180
  address-family vpnv6
    neighbor 212.0.0.11 advertisement-interval 180

```

After adjusting the advertisement interval, I quickly flap CSR9’s loopback. We can check the VPNv4/v6 update-groups on CSR7 to watch it count down as a result of seeing this instability. For the next ~3 minutes, CSR7 cannot send any kind of BGP updates to XRv2. We also ensure that CSR10’s VPN traffic is still egressing through CSR7. The VPN label allocated by CSR7 is 7003; this is the label we are trying to retain until BGP converges.

```

R7#show bgp vpnv4 unicast all update-group 212.0.0.11 | include Minimum
  Minimum time between advertisement runs is 180 seconds (expires in 176
seconds)

```

```

R10#traceroute 9.10.0.9 source 9.10.0.10
Type escape sequence to abort.
Tracing the route to 9.10.0.9
VRF info: (vrf in name/id, vrf out name/id)
 1 10.5.10.5 4 msec 5 msec 4 msec
 2 212.5.11.11 [MPLS: Labels 91003/7003 Exp 0] 8 msec 7 msec 7 msec
 3 10.7.9.7 [MPLS: Label 7003 Exp 0] 12 msec 21 msec 15 msec
 4 10.7.9.9 19 msec 11 msec 10 msec

```


Moving quickly, we change the VLAN tag on CSR9 again to break the PE-CE link (not shown). Because CSR7 still has about 2 minutes remaining on its advertisement-interval, CSR7 cannot notify XRv1 about this change quite yet. Traffic from CSR10 will continue to flow towards CSR7 until the network is notified about the change. CSR7 knows that 232.0.0.8 (CSR8) is the correct next-hop for actually delivering the traffic. The difference is that its local label of 7003 is swapped to CSR8's local label of 8001 and sent towards 212.0.0.8. Without local protection, label 7003 would have been immediately deallocated.

```
R7#show bgp vpnv4 unicast all update-group 212.0.0.11 | include Minimum
Minimum time between advertisement runs is 180 seconds (expires in 123
seconds)
```

```
R7#show bgp vpnv4 unicast vrf 910 9.10.0.9/32
BGP routing table entry for 212:9107:9.10.0.9/32, version 12
Paths: (1 available, best #1, table 910)
Flag: 0x820
  Advertised to update-groups: (Pending Update Generation)
    9
  Refresh Epoch 1
  65009, imported path from 212:9108:9.10.0.9/32 (global)
    212.0.0.8 (metric 20) (via default) from 212.0.0.11 (212.0.0.11)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:212:910
      Originator: 212.0.0.8, Cluster list: 212.0.0.11
      mpls labels in/out 7003/8001
      rx pathid: 0, tx pathid: 0x0
```

Since CSR8 is multiple hops away, CSR7 must add additional MPLS encapsulation. The route to CSR8's loopback is learned via IGP, so an LDP label is used. In this case, label 92001 is pushed atop the stack to create {92001 8001}. Like a TE-FRR PLR, CSR7 adds additional labels to tunnel traffic towards the correct egress PE. This is why the primary PE must have the backup PE's route; if CSR7 didn't already have this iBGP path, local protection would not be useful.

```
R7#show ip route 212.0.0.8
Routing entry for 212.0.0.8/32
  Known via "isis", distance 115, metric 20, type level-2
  Redistributing via isis 212
  Last update from 212.7.12.12 on GigabitEthernet2.572, 02:21:12 ago
  Routing Descriptor Blocks:
  * 212.7.12.12, from 212.0.0.8, 02:21:12 ago, via GigabitEthernet2.572
    Route metric is 20, traffic share count is 1
```

```
R7#show mpls ldp bindings 212.0.0.8 32 neighbor 212.0.0.12
lib entry: 212.0.0.8/32, rev 10
  remote binding: lsr: 212.0.0.12:0, label: 92001
```

XRv2 performs PHP to expose label 8001 to CSR8. CSR8 removes all labels and delivers traffic to CSR9 as a normal egress PE.

```
RP/0/0/CPU0:XRv2#show mpls forwarding labels 92001
```

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
92001	Pop	212.0.0.8/32	Gi0/0/0/0.582	212.8.12.8	56767

```
R8#show mpls forwarding-table labels 8001 detail
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
8001	No Label	9.10.0.9/32[V]	3672	Gi2.589	10.8.9.9

MAC/Encaps=18/18, MRU=1504, Label Stack{}

005056A9D672005056A9FB1C81000E050800

VPN route: 910

No output feature configured

We can use traceroute from CSR10 to confirm this new path. Notice that label 7003 is swapped for label 8001 at CSR7 as shown below.

```
R10#traceroute 9.10.0.9 source 9.10.0.10
```

Type escape sequence to abort.

Tracing the route to 9.10.0.9

VRF info: (vrf in name/id, vrf out name/id)

```

 1 10.5.10.5 7 msec 4 msec 4 msec
 2 212.5.11.11 [MPLS: Labels 91003/7003 Exp 0] 9 msec 8 msec 7 msec
 3 212.7.11.7 [MPLS: Label 7003 Exp 0] 27 msec 31 msec 30 msec
 4 212.7.12.12 [MPLS: Labels 92001/8001 Exp 0] 31 msec 31 msec 31 msec
 5 10.8.9.8 [MPLS: Label 8001 Exp 0] 20 msec 17 msec 15 msec
 6 10.8.9.9 20 msec 10 msec 15 msec
```

At some point, CSR7's advertisement-interval will expire, which will trigger the much-needed BGP reconvergence event. CSR5 will be forced to select CSR8 as the best egress point; we can determine this by checking the VRF-aware FIB on CSR5. Label 8001 is at the bottom of the stack, which we already know is CSR8's VPNv4 label for CSR9's loopback. Traceroute from CSR10 also confirms this new LSP as there is no swapping of the VPN label anymore.

```
R5#show ip cef vrf 910 9.10.0.9/32
```

```
9.10.0.9/32
```

```
  nexthop 212.5.11.11 GigabitEthernet2.551 label 91004 8001
```

```
R10#traceroute 9.10.0.9 source 9.10.0.10
```

Type escape sequence to abort.

Tracing the route to 9.10.0.9

VRF info: (vrf in name/id, vrf out name/id)

```
 1 10.5.10.5 6 msec 4 msec 4 msec
```

```

2 212.5.11.11 [MPLS: Labels 91004/8001 Exp 0] 9 msec 8 msec 9 msec
3 212.7.11.7 [MPLS: Labels 7009/8001 Exp 0] 32 msec 31 msec 31 msec
4 212.7.12.12 [MPLS: Labels 92001/8001 Exp 0] 31 msec 31 msec 31 msec
5 10.8.9.8 [MPLS: Label 8001 Exp 0] 16 msec 16 msec 15 msec
6 10.8.9.9 24 msec 10 msec 10 msec

```

Since the advertisement-interval was 2 minutes less than the time local protection retains local labels (always 5 minutes), we wait a bit longer before checking CSR7. After those 5 minutes expire, we can see that label 7003 has been totally deallocated as local protection is assumed to no longer be required.

```

R7#show bgp vpnv4 unicast vrf 910 9.10.0.9/32
BGP routing table entry for 212:9107:9.10.0.9/32, version 13
Paths: (1 available, best #1, table 910)
  Not advertised to any peer
  Refresh Epoch 1
  65009, imported path from 212:9108:9.10.0.9/32 (global)
    212.0.0.8 (metric 20) (via default) from 212.0.0.11 (212.0.0.11)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:212:910
      Originator: 212.0.0.8, Cluster list: 212.0.0.11
      mpls labels in/out nodelabel/8001
      rx pathid: 0, tx pathid: 0x0

```

We will quickly test the feature using IPv6 on the other side of the network. Because static routing is in use, we cannot flap CSR10's loopback to achieve the desired effect. Instead, we will quickly remove and re-add CSR5's static IPv6 route to CSR10's loopback. This has the same effect as it will trigger a BGP update from CSR5 and start the advertisement-interval. We check CSR5 to ensure the interval is counting down while also ensuring CSR9 still routes via CSR5 to reach CSR10. Note the original VPNv6 label value of 5032.

```

R5#show bgp vpnv6 unicast all update-group 212.0.0.11 | include Minimum
  Minimum time between advertisement runs is 180 seconds (expires in 178
seconds)

```

```

R9#traceroute ipv6
Target IPv6 address: 910:10:10:10::10
Source address: 910:9:9:9::9
[snip]
 1 FD00:10:7:9::7 3 msec 4 msec 5 msec
 2 ::FFFF:212.0.0.11 [MPLS: Labels 91000/5032 Exp 0] 7 msec 8 msec 9 msec
 3 ::FFFF:212.5.11.5 [MPLS: Label 5032 Exp 0] 22 msec 17 msec 20 msec
 4 910:10:10:10::10 [AS 212] 24 msec 14 msec 16 msec

```

Next, we break the PE-CE link between CSR5 and CSR10. Since this is a multi-access network, we must pay attention as to where we break the link. Doing so on the CE side would break everything, so we must break it on CSR5. We change the VLAN tag and wait about 3 seconds for BFD to detect the failure

(not shown). Once the link fails, we immediately check the VPNv6 BGP entry to ensure the local label value of 5032 has not been deallocated. CSR5 is performing the VPN label swap operation as expected.

```
R5#show bgp vpnv6 unicast vrf 910 910:10:10:10::10/128
BGP routing table entry for [212:9105]910:10:10:10::10/128, version 17
Paths: (1 available, best #1, table 910)
Flag: 0x820
  Advertised to update-groups: (Pending Update Generation)
    2
  Refresh Epoch 1
  Local, imported path from [212:9106]910:10:10:10::10/128 (global)
    ::FFFF:212.0.0.6 (metric 20) (via default) from 212.0.0.11 (212.0.0.11)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: RT:212:910
      Originator: 212.0.0.6, Cluster list: 212.0.0.11
      mpls labels in/out 5032/6027
      rx pathid: 0, tx pathid: 0x0
```

We can confirm this by checking the LFIB details. In the past, label 5032 was removed and the raw IPv6 traffic was delivered directly to CSR10. A broken PE-CE link requires CSR5 to swap the VPN label to 6027, CSR6's label for the same VPN route, then add a transport label to tunnel to packets across the core.

```
R5#show mpls forwarding-table labels 5032 detail
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop
Label      Label     or Tunnel Id    Switched     interface
5032       6027      910:10:10:10::10/128[V]  \
                                     1320          Gi2.551      212.5.11.11
MAC/Encaps=18/26, MRU=1496, Label Stack{91001 6027}
005056A92DC6005056A9DC6381000DDF8847 163790000178B000
VPN route: 910
No output feature configured
```

We can confirm the path using traceroute on CSR9 again. Notice that XRv1 is seen in the path twice; like TE-FRR, this is a valid design for short bursts of traffic and does not qualify as a “loop”. Also notice the label activity at CSR5; label 5032 is received and label stack {91001 6027} is sent back into the core as verified above. This behavior appears identical between IPv4 and IPv6.

```
R9#traceroute ipv6
Target IPv6 address: 910:10:10:10::10
Source address: 910:9:9:9::9
[snip]
 1 FD00:10:7:9::7 4 msec 4 msec 5 msec
 2 ::FFFF:212.0.0.11 [MPLS: Labels 91000/5032 Exp 0] 8 msec 8 msec 10 msec
 3 ::FFFF:212.5.11.5 [MPLS: Label 5032 Exp 0] 8 msec 22 msec 23 msec
 4 ::FFFF:212.0.0.11 [MPLS: Labels 91001/6027 Exp 0] 23 msec 22 msec 23 msec
 5 ::FFFF:212.6.11.6 [MPLS: Label 6027 Exp 0] 22 msec 27 msec 23 msec
 6 910:10:10:10::10 [AS 212] 23 msec 21 msec 15 msec
```

For a brief period of about 2 minutes, we can see that BGP has converged but local protection is still enabled on CSR5. This makes sense since 2 minutes is the difference between the advertisement interval and the local protection label retention timer. CSR9 now routes directly via CSR6, indicating BGP has converged correctly. CSR7 uses label 6027 directly at imposition rather than relying on CSR5 to swap it.

```
R9#traceroute ipv6
Target IPv6 address: 910:10:10:10::10
Source address: 910:9:9:9::9
[snip]
 1 FD00:10:7:9::7 3 msec 5 msec 4 msec
 2 ::FFFF:212.0.0.11 [MPLS: Labels 91001/6027 Exp 0] 113 msec 8 msec 8 msec
 3 ::FFFF:212.6.11.6 [MPLS: Label 6027 Exp 0] 18 msec 18 msec 17 msec
 4 910:10:10:10::10 [AS 212] 28 msec 22 msec 15 msec

R5#show bgp vpnv6 unicast vrf 910 labels
[snip]
 910:10:10:10::10/128
                        ::FFFF:212.0.0.6
                        5032/6027
```

Checking CSR5's label allocations shortly thereafter, label 5032 has been deallocated and traffic is now flowing solely through CSR6 as verified above.

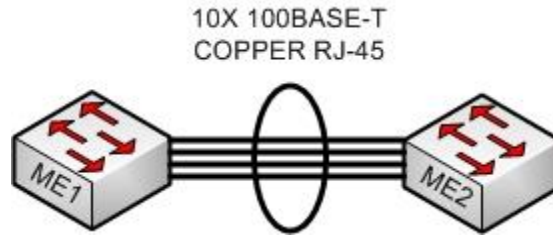
```
R5#show bgp vpnv6 unicast vrf 910 labels
[snip]
 910:10:10:10::10/128
                        ::FFFF:212.0.0.6
                        nolabel/6027
```

Additional Reading – Reference configurations "vrf-local-prot"

39. Describe, implement, and troubleshoot Layer 2 failure detection

39.1 Link Aggregation Control Protocol (LACP)

LACP is a way to bond links together to create larger logical channels. This provides two key benefits: high availability and increased bandwidth. HA is inherent because if some number of links in the bundle fail, the remaining links serve as backups automatically. Bandwidth is increased by way of load-sharing across the links, and there are many algorithms to accomplish this. This lab uses two hardware ME-3400-24TS-A switches with ten links between them.



LACP is supported on ENI and NNI interfaces only, so the port-type is set to NNI for simplicity. The configuration of LACP is very simple; we specify the LACP bond number and the mode of operation. The channel-protocol is an optional field which ensures no other protocols (like Port Aggregation Protocol, PAgP) get negotiated. LACP active mode means that the local switch actively sends LACPDUs to try and establish the bond, while passive mode means a switch will respond to them. Two switches both running in passive mode cannot form LACP bonds, but two switches in active mode can. Both sides are configured as active on all ten ports. I also configure this as a trunk port to carry multiple VLANs if we decide to test that later, although LACP bonds can be access ports (or layer 3 routed ports). The configuration at the port level can be identical on both sides and for all ports in the bond. Since these ports were originally UNI ports, we also have to “no shutdown” them, as UNI ports are shutdown by default in this particular Metro IOS version.

```
! ME1 and ME2
interface range FastEthernet0/13 - 22
  port-type nni
  switchport mode trunk
  channel-protocol lacp
  channel-group 1 mode active
  spanning-tree portfast trunk
  no shutdown
```

With this very basic configuration, we can see that the bond comes up with no issues. Looking at ME1, we see that the first 8 ports are actively used in the bond, while the remaining 2 are in hot-standby mode. LACP supports up to 16 links in a bond, but only 8 can be used at once, with up to 8 standby links. PAgP and static Etherchannel do not support the hot-standby feature. The “SU” flags in the output indicate that the port-channel is in use and that it is switched (running at layer 2). ME2 has the exact same output and is not shown.

```
ME1#show etherchannel summary
Flags:  D - down                P - bundled in port-channel
        I - stand-alone s - suspended
        H - Hot-standby (LACP only)
        R - Layer3            S - Layer2
        U - in use           f - failed to allocate aggregator

        M - not in use, minimum links not met
        u - unsuitable for bundling
        w - waiting to be aggregated
```

d - default port

Number of channel-groups in use: 1
Number of aggregators: 1

Group	Port-channel	Protocol	Ports
1	Po1 (SU)	LACP	Fa0/13 (P) Fa0/14 (P) Fa0/15 (P) Fa0/16 (P) Fa0/17 (P) Fa0/18 (P) Fa0/19 (P) Fa0/20 (P) Fa0/21 (H) Fa0/22 (H)

Most of the time, the command above is the only command you really need to verify and troubleshoot LACP. We will examine advanced commands as well since they are not so well-known. First, we can look at the LACP neighbors. The device ID is the neighbor's MAC address which is also used for the STP bridge-ID MAC address component. We can prove this by checking the STP information on ME2. We can see all ten of the ports have the same neighbor ID since there is no multi-homed, multi-chassis LACP being tested here. All ports are using "slow" LACPDU's; this happens automatically once an Etherchannel is stable, which is a 30 second PDU timer. The "Age" column resets to 0 when an LACPDU is received and counts up to 30. During the initial setup, "fast" LACPDU's are sent which is once per second. We also see the port numbers are listed in hexadecimal which are just sequential numbers assigned to ports; this is similar to the STP port ID and is used as a tie-breaker for selecting active links (more on this later). The port state is a code, again similar to STP, which shows what the port is doing. We can see code 0x3D probably relates to forwarding while code 0x5 relates to hot-standby (or a general link-down state).

ME1#show lacp 1 neighbor

Flags: S - Device is requesting Slow LACPDU's
F - Device is requesting Fast LACPDU's
A - Device is in Active mode P - Device is in Passive mode

Channel group 1 neighbors

Partner's information:

Port	Flags	LACP port Priority	Dev ID	Age	Admin key	Oper Key	Port Number	Port
Fa0/13	SA	32768	04c5.a453.d800	22s	0x0	0x1	0x110	0x3D
Fa0/14	SA	32768	04c5.a453.d800	27s	0x0	0x1	0x111	0x3D
Fa0/15	SA	32768	04c5.a453.d800	2s	0x0	0x1	0x112	0x3D
Fa0/16	SA	32768	04c5.a453.d800	24s	0x0	0x1	0x113	0x3D
Fa0/17	SA	32768	04c5.a453.d800	3s	0x0	0x1	0x114	0x3D
Fa0/18	SA	32768	04c5.a453.d800	18s	0x0	0x1	0x115	0x3D
Fa0/19	SA	32768	04c5.a453.d800	24s	0x0	0x1	0x116	0x3D
Fa0/20	SA	32768	04c5.a453.d800	25s	0x0	0x1	0x117	0x3D
Fa0/21	SA	32768	04c5.a453.d800	24s	0x0	0x1	0x118	0x5
Fa0/22	SA	32768	04c5.a453.d800	3s	0x0	0x1	0x119	0x5

ME2#show spanning-tree vlan 1 | section Bridge

Bridge ID Priority 32769 (priority 32768 sys-id-ext 1)

```

Address      04c5.a453.d800
Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec
Aging Time   300 sec

```

If we change a few ports on ME2 to be in passive mode (we can have a variety of modes on the same switch for the same bond), we can see the output changes to reflect that. Changing ports 13 and 14 on ME2 means that ME1 sees these peers as passive now, while all other ports are still active.

```

! ME2
interface range FastEthernet0/13 - 14
  channel-group 1 mode passive

ME1#show lacp 1 neighbor
Flags:  S - Device is requesting Slow LACPDUs
        F - Device is requesting Fast LACPDUs
        A - Device is in Active mode          P - Device is in Passive mode
Channel group 1 neighbors
Partner's information:

```

Port	Flags	LACP port Priority	Dev ID	Age	Admin key	Oper Key	Port Number	Port
Fa0/13	SP	32768	04c5.a453.d800	16s	0x0	0x1	0x110	0x3C
Fa0/14	SP	32768	04c5.a453.d800	15s	0x0	0x1	0x111	0x3C
Fa0/15	SA	32768	04c5.a453.d800	19s	0x0	0x1	0x112	0x3D
Fa0/16	SA	32768	04c5.a453.d800	2s	0x0	0x1	0x113	0x3D

```

[snip]

```

LACP has the concept of a system-ID, which is loosely analogous to the STP bridge-ID. It is computed the same way; a 2 byte administratively-defined priority is prepended to the MAC address of the switch, and the lower value is considered the decision-maker in terms of which links are used for forwarding. Below, we can see the system-ID on both ME1 and ME2. Both have the default priority of 32768 and their switch global MAC addresses. ME1 has the lower MAC address and, as such, is the decision-maker on which links get aggregated.

```

ME1#show lacp sys-id
32768, 001d.4692.f700

ME2#show lacp sys-id
32768, 04c5.a453.d800

```

For variety, we will configure ME2 to be the LACP decision-maker by adjusting its system priority globally, then confirming the change. Changing the decision-maker, in this case from ME1 to ME2, will cause the bond to flap, so don't do it in production.

```

! ME2
lacp system-priority 16384

```



```
ME2#show lacp sys-id
16384, 04c5.a453.d800
```

We can change the port-priority to select which ports should be used for forwarding. Much like changing the STP port-priority on an upstream switch to influence root port selection on a downstream switch, this is configured on a per-interface basis to select usable ports. Our example has 10 ports, yet only 8 are used for forwarding. The 8 ports with the lowest port-ID (which is the concatenation of the port-priority and the port-number) are used for forwarding. That is why the first 8 ports are used, since their port-numbers as seen above are lower. We will reduce the port priorities of ports 21 and 22 on ME1 to observe the effect; we see that nothing changes.

```
! ME1
interface range FastEthernet0/21 - 22
 lacp port-priority 16384
```

Looking at the internal details of the bond, ME1 shows that the new port-priority was configured correctly yet the ports are stuck in hot-standby. This is because ME1 is no longer the decision-maker on which ports are used for forwarding in the bond as it has the higher LACP system-ID. This is equivalent to configuring port-priority on a downstream STP switch on the root port and expecting it to do something. Only ME2 can determine which ports can be used for forwarding, so ME1's port-priorities for this channel are meaningless.

```
ME1#show lacp internal | include 2[12]
Fa0/21    SA      hot-sby  16384      0x1      0x1      0x118      0x5
Fa0/22    SA      hot-sby  16384      0x1      0x1      0x119      0x5
```

We will use a similar configuration on ME2 to achieve the desired effect. This change also causes ports in the bond to flap, so don't do it in production. ME1 still shows its local port-priority of 16384, which is meaningless, but the ports are now active in the bundle since ME2 decided to do this. I show outputs from both ME1 and ME2 for clarity here; we can see that the port-priority column is a reflection of the local configuration only. Now, the next worse LACP port-IDs are placed into hot-standby mode, which are ports 19 and 20.

```
! ME2
interface range FastEthernet0/21 - 22
 lacp port-priority 8192
```

```
ME1#show lacp internal
[snip]
```

Port	Flags	State	LACP port Priority	Admin Key	Oper Key	Port Number	Port State
Fa0/13	SA	bndl	32768	0x1	0x1	0x110	0x3D
Fa0/14	SA	bndl	32768	0x1	0x1	0x111	0x3D
Fa0/15	SA	bndl	32768	0x1	0x1	0x112	0x3D
Fa0/16	SA	bndl	32768	0x1	0x1	0x113	0x3D

Fa0/17	SA	bndl	32768	0x1	0x1	0x114	0x3D
Fa0/18	SA	bndl	32768	0x1	0x1	0x115	0x3D
Fa0/19	SA	hot-sby	32768	0x1	0x1	0x116	0x5
Fa0/20	SA	hot-sby	32768	0x1	0x1	0x117	0x5
Fa0/21	SA	bndl	16384	0x1	0x1	0x118	0x3D
Fa0/22	SA	bndl	16384	0x1	0x1	0x119	0x3D

```
ME2#show lacp internal
[snip]
```

Port	Flags	State	LACP port Priority	Admin Key	Oper Key	Port Number	Port State
Fa0/13	SP	bndl	32768	0x1	0x1	0x110	0x3C
Fa0/14	SP	bndl	32768	0x1	0x1	0x111	0x3C
Fa0/15	SA	bndl	32768	0x1	0x1	0x112	0x3D
Fa0/16	SA	bndl	32768	0x1	0x1	0x113	0x3D
Fa0/17	SA	bndl	32768	0x1	0x1	0x114	0x3D
Fa0/18	SA	bndl	32768	0x1	0x1	0x115	0x3D
Fa0/19	SA	hot-sby	32768	0x1	0x1	0x116	0x5
Fa0/20	SA	hot-sby	32768	0x1	0x1	0x117	0x5
Fa0/21	SA	bndl	8192	0x1	0x1	0x118	0x3D
Fa0/22	SA	bndl	8192	0x1	0x1	0x119	0x3D

The “neighbor” command seen earlier reflects the remote peer configuration. On ME1, for example, we can see the remote port-priority on those newly-bundled ports. This is in contrast to the “internal” command which reflects the local configuration.

```
ME1#show lacp neighbor | include 2[12]
Fa0/21    SA      8192    04c5.a453.d800  17s    0x0    0x1    0x118    0x3D
Fa0/22    SA      8192    04c5.a453.d800   4s    0x0    0x1    0x119    0x3D
```

The detail version of the “internal” and “neighbor” commands will display a more verbose version of the tabular data for local and remote LACP fields, respectively. Snippets are shown below, but very little new information is provided, so we don’t analyze it. In this example, we look at ME2’s port 13 details (neighbor) and ME1’s port 13 details (internal), both from ME1’s perspective.

```
ME1#show lacp neighbor detail
[snip]
```

Port	Partner System ID	Partner Port Number	Age	Partner Flags
Fa0/13	16384, 04c5.a453.d800	0x110	18s	SP

LACP Partner Port Priority	Partner Oper Key	Partner Port State
32768	0x1	0x3C

```
Port State Flags Decode:
Activity:   Timeout:   Aggregation:   Synchronization:
```

Passive	Long	Yes	Yes
Collecting:	Distributing:	Defaulted:	Expired:
Yes	Yes	No	No

```
ME1#show lacp internal detail
[snip]
```

Port	Actor	Actor	Actor
	System ID	Port Number	Age
Fa0/13	32768, 001d.4692.f700	0x110	11s
	SA		

LACP Actor	Actor	Actor
Port Priority	Oper Key	Port State
32768	0x1	0x3D

Port State Flags Decode:

Activity:	Timeout:	Aggregation:	Synchronization:
Active	Long	Yes	Yes

Collecting:	Distributing:	Defaulted:	Expired:
Yes	Yes	No	No

All of these details represent the same fields that are carried in LACPDUs during the bond negotiation. A short debugging effort on ME1 reveals some of these LACPDUs being sent. The “p-pri” and “s-pri” fields represent the port and system priorities, respectively, and are show in hexadecimal (0x8000 = 32768).

```
! ME1
ME1#debug lacp packet
Link Aggregation Control Protocol packet debugging is on

! ME1
00:59:15: LACP :lacp_bugpak: Send LACP-PDU packet via Fa0/13
00:59:15: LACP : packet size: 124
00:59:15: LACP: pdu: subtype: 1, version: 1
00:59:15: LACP: Act: tlv:1, tlv-len:20, key:0x1, p-pri:0x8000, p:0x110, p-
state:0x3D,
s-pri:0x8000, s-mac:001d.4692.f700
```

Assuming I unplug one of the links actively in the bundle, the hot-standby links will take over for it. I unplug port 13, which means port 19 is the next most desirable port for forwarding. Notice that the line protocol is moved from down to up across interfaces in about one second. Port 20 still remains down, as LACP is holding it in this state given its high port-ID compared to the remaining ports.

```
! ME1
01:03:34: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/13,
changed state to down
01:03:35: %LINK-3-UPDOWN: Interface FastEthernet0/13, changed state to down
```

```
01:03:35: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/19,  
changed state to up
```

```
ME1#show interfaces fastEthernet 0/20 | include line_proto  
FastEthernet0/20 is up, line protocol is down (notconnect)
```

A quick check of the LACP internal information shows the first port as down, which interestingly has the same code of 0x5 as hot-standby. Port 19 is now in the bundle while port 20 remains in hot-standby.

```
ME1#show lacp internal  
[snip]
```

Port	Flags	State	LACP port Priority	Admin Key	Oper Key	Port Number	Port State
Fa0/13	SA	down	32768	0x1	0x1	0x110	0x5
Fa0/14	SA	bndl	32768	0x1	0x1	0x111	0x3D
Fa0/15	SA	bndl	32768	0x1	0x1	0x112	0x3D
Fa0/16	SA	bndl	32768	0x1	0x1	0x113	0x3D
Fa0/17	SA	bndl	32768	0x1	0x1	0x114	0x3D
Fa0/18	SA	bndl	32768	0x1	0x1	0x115	0x3D
Fa0/19	SA	bndl	32768	0x1	0x1	0x116	0x3D
Fa0/20	SA	hot-sby	32768	0x1	0x1	0x117	0x5
Fa0/21	SA	bndl	16384	0x1	0x1	0x118	0x3D
Fa0/22	SA	bndl	16384	0x1	0x1	0x119	0x3D

We plug port 13 in before continuing. Now that we have tested the high-availability and channel formation details, we will discuss the bandwidth benefits. These benefits are not specific to LACP and apply to all types of Ethernet link aggregation. Because this FastEthernet Etherchannel (FEC, not to be confused with MPLS forwarding equivalence class) consists of 8 active members, it has an aggregated bandwidth of 800 Mbps. This is important from an IGP perspective, because if we were running routing over this single logical link, we should prefer it more than a single 100 Mbps link, but less than a single 1 Gbps link. In the case of switching, STP takes this new bandwidth into account automatically; we know that classic STP uses a cost of 19 for 100 Mbps ports and 4 for 1 Gbps port. The cost of this 800 Mbps FEC is 5, which sounds right. This value of 5 is much better than 19 but just slightly worse than 4.

```
ME1#show interfaces port-channel 1 | include BW  
MTU 1998 bytes, BW 800000 Kbit/sec, DLY 100 usec,
```

```
ME1#show spanning-tree interface port-channel 1  
Vlan          Role Sts Cost          Prio.Nbr Type  
-----  
VLAN0001      Desg FWD 5           128.64  P2p
```

To achieve the load-sharing, Etherchannels can select from a variety of algorithms, and I show the context-sensitive help below with added comments. The term “IP” always includes IPv4, but may include IPv6 depending on the platform. Non-IP traffic (PPPoE, MPLS, CLNS, etc) is always load balanced based on MAC addresses since the IP-based load sharing doesn’t make sense.

```
ME1(config)#port-channel load-balance
dst-ip      Dst IP Addr ! Hashed based on destination IP address only
dst-mac     Dst Mac Addr ! Hashed based on destination MAC address only
src-dst-ip  Src XOR Dst IP Addr ! Hased based on source/dest IP address
src-dst-mac Src XOR Dst Mac Addr ! Hased based on source/dest MAC address
src-ip      Src IP Addr ! Hashed based on source IP address only
src-mac     Src Mac Addr ! Hashed based on source MAC address only
```

The load-balancing method is configured globally and the default on this platform is “src-mac”. We can verify this with a show command, and we see that both IP and non-IP traffic uses this method.

```
ME1#show etherchannel load-balance
EtherChannel Load-Balancing Configuration:
src-mac
```

```
EtherChannel Load-Balancing Addresses Used Per-Protocol:
Non-IP: Source MAC address
IPv4: Source MAC address
```

We can quickly test to see if this works. Each switch has a management SVI in VLAN 1, and the IP addresses for ME1 and ME2 are 10.0.10.91 and 10.0.10.92, respectively. We can test, using an exec command, to see which link is used when ME1 sends traffic to ME2 from this SVI, and vice versa. First, we need to record the MAC addresses of the SVIs.

```
ME1#show interfaces vlan 1 | include bia
Hardware is EtherSVI, address is 001d.4692.f740 (bia 001d.4692.f740)

ME2#show interfaces vlan 1 | include bia
Hardware is EtherSVI, address is 04c5.a453.d840 (bia 04c5.a453.d840)
```

Next, we issue the “test” commands on ME1 and ME2 to determine which member links will be used for these flows. Notice that the ports between switches don’t have to be the same, since the load-balancing decisions only affect how traffic is transmitted. Traffic can be received on any bundled link, since each switch can have a different load-sharing policy for traffic transmission.

```
ME1#test etherchannel load-balance interface port-channel 1 mac
001d.4692.f740 04c5.a453.d840
Would select Fa0/21 of Po1

ME2#test etherchannel load-balance interface port-channel 1 mac
04c5.a453.d840 001d.4692.f740
Would select Fa0/14 of Po1
```

Since the current load-sharing mechanism only cares about the source MAC, the destination MAC can change but the outgoing interface remains the same. We do a few bogus tests on ME1 to prove this.

```
ME1#test etherchannel load-balance interface port-channel 1 mac
001d.4692.f740 04c5.a453.2bad
Would select Fa0/21 of Po1
```

```
ME1#test etherchannel load-balance interface port-channel 1 mac
001d.4692.f740 04c5.a453.deed
Would select Fa0/21 of Po1
```

```
ME1#test etherchannel load-balance interface port-channel 1 mac
001d.4692.f740 04c5.a453.beef
Would select Fa0/21 of Po1
```

As a data-plane test, I will send many packets from ME2 to ME1, then check the counters while the flow is happening. We check port 14 on ME2 and port 21 on ME1 to see the packets increasing quickly.

```
ME2#ping 10.0.10.91 size 1500 repeat 1000000
```

```
ME1#show interfaces fastEthernet 0/21 | include packets_output
7055 packets output, 10339446 bytes, 0 underruns
```

```
ME2#show interfaces fastEthernet 0/14 | include packets_output
7127 packets output, 10368100 bytes, 0 underruns
```

For further proof, we can look at all the output packets on ME1 within the bond. It is obvious that one port has the most packets by far. Other ports are accounting for LACPDUs being transferred back and forth, along with any other flows that may be going between a different set of MAC addresses.

```
ME1#show interfaces | include packets_output
[snip]
2762 packets output, 237064 bytes, 0 underruns
341 packets output, 75578 bytes, 0 underruns
2932 packets output, 235918 bytes, 0 underruns
300 packets output, 64138 bytes, 0 underruns
306 packets output, 64906 bytes, 0 underruns
299 packets output, 63744 bytes, 0 underruns
290 packets output, 57346 bytes, 0 underruns
271 packets output, 50786 bytes, 0 underruns
7060 packets output, 10340618 bytes, 0 underruns
245 packets output, 40138 bytes, 0 underruns
```

We will test 2 other load sharing methods. ME1 will use the combination of source/destination IP while ME2 uses the combination of source/destination MAC. Notice that ME1 uses source/destination MAC load-sharing automatically for non-IP traffic, while ME2 uses this technique for both.

! ME1

```
port-channel load-balance src-dst-ip
```

```
! ME2
```

```
port-channel load-balance src-dst-mac
```

```
ME1#show etherchannel load-balance
```

```
EtherChannel Load-Balancing Configuration:  
    src-dst-ip
```

```
EtherChannel Load-Balancing Addresses Used Per-Protocol:
```

```
Non-IP: Source XOR Destination MAC address
```

```
    IPv4: Source XOR Destination IP address
```

```
ME2#show etherchannel load-balance
```

```
EtherChannel Load-Balancing Configuration:  
    src-dst-mac
```

```
EtherChannel Load-Balancing Addresses Used Per-Protocol:
```

```
Non-IP: Source XOR Destination MAC address
```

```
    IPv4: Source XOR Destination MAC address
```

With these new methods, we may get different results for our load sharing on both switches. ME1 coincidentally selects port 21 again while ME2 selects port 13. Notice that ME1 used IP addresses for input while ME2 used MAC addresses.

```
ME1#test etherchannel load-balance interface port-channel 1 ip 10.0.10.91  
10.0.10.92
```

```
Would select Fa0/21 of Po1
```

```
ME2#test etherchannel load-balance interface port-channel 1 mac  
04c5.a453.d840 001d.4692.f740
```

```
Would select Fa0/13 of Po1
```

We will clear counters on both switches and perform a short ping test of 100 packets. We can see that both switches have just over 100 packets (add in a few recent LACPDUs) on their selected outgoing interfaces.

```
ME2#ping 10.0.10.91 size 1500 repeat 100
```

```
ME1#show interfaces fastEthernet 0/21 | include packets_output  
    103 packets output, 152450 bytes, 0 underruns
```

```
ME2#show interfaces fastEthernet 0/13 | include packets_output  
    102 packets output, 152056 bytes, 0 underruns
```

As a note, when using the “test” command with an IP-based load-sharing scheme, the parser warns you that the result is not accurate for IP traffic. For example, if we used the MAC addresses as input on ME1,

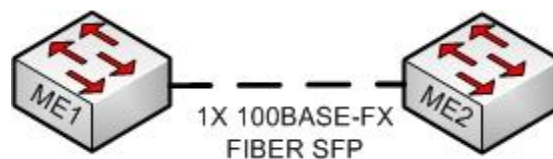
we get a different output interface (port 13), which may not be true for IP traffic. Since ME1 does use this mechanism for non-IP traffic, this command is still valuable, but you must be diligent to provide the proper inputs depending on the traffic type. MPLS traffic exiting ME1 towards ME2 would actually use port 13, as an example.

```
ME1#test etherchannel load-balance interface port-channel 1 mac
001d.4692.f740 04c5.a453.d840
Configured load-balance is "src-dst-ip", results may not be accurate for IP
traffic
Would select Fa0/13 of Po1
```

Additional Reading – Reference configurations "lACP"

39.2 Uni-Directional Link Detection (UDLD)

UDLD is a feature that is designed to detect unidirectional links. It is generally used on fiber links where there are explicit and separate transit (TX) and receive (RX) fiber lines. Each set of TX/RX lines is called a “pair” and ideally, both TX and RX directions are functional. When one of the lines fails, the overall link does not perform as expected, and UDLD helps mitigate this. The network diagram is very simple and consists of two hardware Cisco ME-3400-24TS-A switches connected with a single fiber pair. The SFPs used are 100BaseFX, which is 100 Mbps.



First, we will configure the interfaces to support a basic fiber connection between the two. I hardcode the duplex to full since sometimes, in my experience, the FX SFPs will negotiate half-duplex. We cannot set the speed on these interfaces as this is controlled by the SFP hardware. UDLD is not yet enabled, but CDP is by default, and we will use that for verification assistance.

```
! ME1 and ME2
interface GigabitEthernet0/1
  description UDLD TEST
  port-type nni
  duplex full
```

We verify the basic link characteristics first. We see a 100Base-FX SFP with a speed of 100 Mbps, along with full-duplex operation. The interface hardware components appear to be working properly.

```
ME1#show interfaces gigabitEthernet 0/1 | include media_type|line_proto
GigabitEthernet0/1 is up, line protocol is up (connected)
Full-duplex, 100Mb/s, link type is auto, media type is 100BaseFX SFP
```



```
ME2#show interfaces gigabitEthernet 0/1 | include media_type|line_proto
GigabitEthernet0/1 is up, line protocol is up (connected)
Full-duplex, 100Mb/s, link type is auto, media type is 100BaseFX SFP
```

We can also see that CDP sees neighbors in both directions, which is a quick indication that the fiber pair is working. We are also able to ping between management SVIs, which is another verification method.

```
ME1#show cdp neighbors gigabitEthernet 0/1
Capability Codes: R - Router, T - Trans Bridge, B - Source Route Bridge
                  S - Switch, H - Host, I - IGMP, r - Repeater, P - Phone,
                  D - Remote, C - CVTA, M - Two-port Mac Relay
```

Device ID	Local Intrfce	Holdtme	Capability	Platform	Port ID
ME2	Gig 0/1	145	S I	ME-3400-2	Gig 0/1

```
ME2#show cdp neighbors gigabitEthernet 0/1
```

```
[snip]
```

Device ID	Local Intrfce	Holdtme	Capability	Platform	Port ID
ME1	Gig 0/1	136	S I	ME-3400-2	Gig 0/1

```
ME1#ping 10.0.10.92
```

```
Type escape sequence to abort.
```

```
Sending 5, 100-byte ICMP Echos to 10.0.10.92, timeout is 2 seconds:
```

```
!!!!
```

```
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/4/9 ms
```

Now that we are certain the link is functional, we will enable UDLD. There are two modes of operation: normal and aggressive. In normal mode, when a unidirectional link is detected, UDLD does not take action. The port forwarding states will behave according to the STP topology. When a port is determined to be bi-directional (functional), once the UDLD message hold-down time expires without having seen UDLD messages from the remote end, the port is considered to be unidirectional. UDLD can be enabled globally or per-interface. When enabled globally, it only applies to fiber ports, but you can enable it on copper ports at the interface level. ME1 uses the global command and ME2 uses the port command; both are using UDLD normal mode. We can also disable UDLD on specific ports when it is enabled globally; on ME1, I demonstrate this by disabling UDLD on Gig0/2, a dummy port. In both cases, UDLD normal mode is enabled on Gig0/1 only on both switches.

```
! ME1
```

```
udld enable
```

```
interface GigabitEthernet0/2
```

```
udld port disable
```

```
! ME2
```

```
interface GigabitEthernet0/1
```

```
udld port
```

We can see the state of UDLD by checking the neighbors. The device name shown in the neighbor table is the remote device's serial number, which we prove by checking ME2.

```
ME1#show udld neighbors
```

Port	Device Name	Device ID	Port ID	Neighbor State
----	-----	-----	-----	-----
Gi0/1	FOC1510V51X	1	Gi0/1	Bidirectional

```
ME2#show version | include System_serial
```

```
System serial number      : FOC1510V51X
```

Often times, looking at the neighbor summary is sufficient for UDLD verification, but specifying an interface prints more detail. Specifically, we can see the message interval and timeout which are discussed later.

```
ME1#show udld gigabitEthernet 0/1
```

```
Interface Gi0/1
```

```
---
```

```
Port enable administrative configuration setting: Follows device default
```

```
Port enable operational state: Enabled
```

```
Current bidirectional state: Bidirectional
```

```
Current operational state: Advertisement - Single neighbor detected
```

```
Message interval: 15
```

```
Time out interval: 5
```

```
Entry 1
```

```
---
```

```
Expiration time: 34
```

```
Device ID: 1
```

```
Current neighbor state: Bidirectional
```

```
Device name: FOC1510V51X
```

```
Port ID: Gi0/1
```

```
Neighbor echo 1 device: FOC1132U4V2
```

```
Neighbor echo 1 port: Gi0/1
```

```
Message interval: 15
```

```
Time out interval: 5
```

```
CDP Device name: ME2
```

We can test UDLD by simply unplugging one of the lines on one of the pairs. Since the link is currently in the bidirectional state, this will change shortly, which should trigger UDLD. Unplugging the RX line on ME2 means that the ME2 interface goes down but ME1 stays up. There is nothing of interest to test on ME2 since the interface is down, but on ME1, we can see the UDLD and CDP neighbors are gone. However, the interface stays up, and is forwarding per STP. UDLD in normal mode isn't terribly useful and it doesn't generate a syslog message, either.

```
ME1#show udld neighbors
```

Port	Device Name	Device ID	Port ID	Neighbor State
----	-----	-----	-----	-----

[no output]

```
ME1#show cdp neighbors gigabitEthernet 0/1
```

[snip]

Device ID	Local Intrfce	Holdtme	Capability	Platform	Port ID

[no output]

```
ME1#show spanning-tree vlan 1
```

VLAN0001

Spanning tree enabled protocol rstp

Root ID	Priority	32769
	Address	001d.4692.f700

This bridge is the root

Hello Time	2 sec	Max Age 20 sec	Forward Delay 15 sec

Bridge ID	Priority	32769	(priority 32768 sys-id-ext 1)
	Address	001d.4692.f700	
	Hello Time	2 sec	Max Age 20 sec Forward Delay 15 sec
	Aging Time	300 sec	

Interface	Role	Sts	Cost	Prio.Nbr	Type
-----	----	---	-----	-----	-----

Gi0/1	Desg	FWD	19	128.1	P2p

Next, we will restore fiber connectivity and enable UDLD aggressive mode. This is also enabled globally or per-port, as we see on ME1 and ME2. UDLD remains disabled on ME1 Gig0/2 explicitly, so only Gig0/1 on both switches is UDLD-enabled. I also enable err-disable recovery on both switches so that after a UDLD failure is fixed, the port can be restored automatically.

```
! ME1
```

```
errdisable recovery cause udld
errdisable recovery interval 30
udld port aggressive
```

```
! ME2
```

```
errdisable recovery cause udld
errdisable recovery interval 30
```

```
interface GigabitEthernet0/1
  udld port aggressive
```

ME1 sees ME2 as a bidirectional neighbor again, except this time in aggressive mode. Aggressive mode will err-disable unidirectional links so that alternative paths can be used by STP and other protocols. This

only occurs after aggressive mode tries to recover the link by sending more frequent UDLD messages across; err-disable is the result of the recovery failure.

```
ME1#show udld neighbors
```

Port	Device Name	Device ID	Port ID	Neighbor State
Gi0/1	FOC1510V51X	1	Gi0/1	Bidirectional

```
ME1#show udld gigabitEthernet 0/1
```

```
Interface Gi0/1
```

```
---
```

```
Port enable administrative configuration setting: Follows device default
```

```
Port enable operational state: Enabled / in aggressive mode
```

```
Current bidirectional state: Bidirectional
```

```
Current operational state: Advertisement - Single neighbor detected
```

```
Message interval: 15
```

```
Time out interval: 5
```

```
[snip]
```

Now, we unplug ME2 RX connection the break bidirectional connectivity, ME1's interface stays up until UDLD detects the broken connectivity. UDLD aggressive-mode prints syslog messages to notify us about this failure, then err-disables the port.

```
! ME1
```

```
02:49:49: %UDLD-4-UDLD_PORT_DISABLED: UDLD disabled interface Gi0/1,
```

```
aggressive mode failure detected
```

```
02:49:49: %PM-4-ERR_DISABLE: udld error detected on Gi0/1, putting Gi0/1 in  
err-disable state
```

```
02:49:49: %LINEPROTO-5-UPDOWN: Line protocol on Interface Vlan1, changed  
state to down
```

```
02:49:50: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/1,  
changed state to down
```

The port is now unusable; since there are no alternative paths, there are no interfaces in the "up" state on either switch, so connectivity is totally broken. If there were alternative paths, those would be used instead.

```
ME1#show interfaces gigabitEthernet 0/1 status
```

Port	Name	Status	Vlan	Duplex	Speed	Type
Gi0/1	UDLD TEST	err-disabled	1	full	100	100BaseFX SFP

```
ME1#show spanning-tree vlan 1
```

```
Spanning tree instance(s) for vlan 1 does not exist.
```

The err-disable recovery timer tries to recover the port every 30 seconds, so plugging the cable back in restores the port, and thus the UDLD error clears. We can see that in 6 seconds, the err-disable process

will attempt to recover this port, and the log messages from a successful recovery are shown below. Without the auto-recovery configured, manual “shutdown” and “no shutdown” commands are required. Alternatively, the “udld reset” command can be used.

```
ME1#show errdisable recovery | begin reason
Interface      Errdisable reason      Time left(sec)
-----
Gi0/1          udld                    6

! ME1
%PM-4-ERR_RECOVER: Attempting to recover from udld err-disable state on Gi0/1
%LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/1, changed
state to up
```

We will break the link again to capture more debugging information. We can watch this UDLD behavior in action by looking at UDLD events on ME1 (or whichever switch still has its RX line connected). Before the failure, we see UDLD messages being exchanged every 15 seconds as expected.

```
ME1#debug udld events
UDLD events debugging is on

! ME1
02:58:36: New_entry = 55128D8 (Gi0/1)
02:58:36: Found an entry from same device (Gi0/1)
02:58:36: Cached entries = 2 (Gi0/1)
02:58:36: Entry (0x62359C0) deleted: 1 entries cached
02:58:36: Cached entries = 1 (Gi0/1)
02:58:36: Checking if multiple neighbors (Gi0/1)
02:58:36: Single neighbor detected (Gi0/1)
02:58:36: Checking if link is bidirectional (Gi0/1)
02:58:36: Found my own ID pair in 2way conn list (Gi0/1)

02:58:51: New_entry = 62359C0 (Gi0/1)
02:58:51: Found an entry from same device (Gi0/1)
02:58:51: Cached entries = 2 (Gi0/1)
02:58:51: Entry (0x55128D8) deleted: 1 entries cached
02:58:51: Cached entries = 1 (Gi0/1)
02:58:51: Checking if multiple neighbors (Gi0/1)
02:58:51: Single neighbor detected (Gi0/1)
02:58:51: Checking if link is bidirectional (Gi0/1)
02:58:51: Found my own ID pair in 2way conn list (Gi0/1)
```

Once we unplug the cable, the output changes. ME1 shows no output for about 45 seconds (3 times the hello interval) then declares that all neighbors on Gig0/1 have aged out. Then, UDLD aggressive mode sends 8 hellos 1 second apart to try and recover the link. After that, the link is considered down and is err-disabled.

```

! ME1
03:02:20: allNeighborsAgedOutEvent during link up. (Gi0/1)
03:02:20: Phase set from ADV to LUP because all neighbors aged out (Gi0/1)
03:02:20: prev = 0 entry = 55128D8 next = 0 exp_time = 0 (Gi0/1)
03:02:20: udsb->cache = 0x532E274 (Gi0/1)
03:02:20: timeout timer = 7 (Gi0/1)
03:02:21: timeout timer = 6 (Gi0/1)
03:02:22: timeout timer = 5 (Gi0/1)
03:02:23: timeout timer = 4 (Gi0/1)
03:02:24: timeout timer = 3 (Gi0/1)
03:02:25: timeout timer = 2 (Gi0/1)
03:02:26: timeout timer = 1 (Gi0/1)
03:02:27: timeout timer = 0 (Gi0/1)
03:02:27: Phase set to udld advertisement from phase udld_link_up in
aggressive mode after all neighbors aged out. (Gi0/1)
03:02:27: %UDLD-4-UDLD_PORT_DISABLED: UDLD disabled interface Gi0/1,
aggressive mode failure detected
03:02:27: %PM-4-ERR_DISABLE: udld error detected on Gi0/1, putting Gi0/1 in
err-disable state
03:02:27: Port UDLD set error disabled (Gi0/1)

```

We can plug the line back in and perform a manual CLI reset to fix the network so we can test it once more with the UDLD packet debugs. We would also wait for err-disable recovery, but the manual CLI reset is faster and more appropriate for testing.

```

ME1#udld reset
1 ports shutdown by UDLD were reset.

ME1#show interfaces gigabitEthernet 0/1 status
Port      Name      Status      Vlan    Duplex  Speed Type
Gi0/1     UDLD TEST connected    1       full    100 100BaseFX SFP

```

With the network stable again, I break the network once more by unplugging the RX line from ME2. Packet debugging has been enabled on ME1 so we can see the UDLD activity, as well as aggressive mode trying to recover the link. The first two probes are ordinary messages and are sent 15 seconds apart. Once UDLD sees the issues, it issues 8 “resynch” messages in quick succession, once per second, until it flushes the UDLD state from the database. I personally think aggressive mode is a much better option than normal mode. I would not consider this “high” availability since it takes almost a minute to detect and bypass the failure.

```

ME1#debug udld packets
UDLD packets debugging is on

! ME1
03:09:48: UDLD send probe message, flags = rec_timeout (Gi0/1)
03:09:48: P (Gi0/1)
03:10:03: UDLD send probe message, flags = rec_timeout (Gi0/1)

```

```

03:10:03: P (Gi0/1)
03:10:17: UDLD send probe message, flags = rec_timeout | resynch (Gi0/1)
03:10:17: Pr (Gi0/1)
03:10:18: UDLD send probe message, flags = rec_timeout | resynch (Gi0/1)
03:10:18: Pr (Gi0/1)
03:10:19: UDLD send probe message, flags = rec_timeout | resynch (Gi0/1)
03:10:19: Pr (Gi0/1)
03:10:20: UDLD send probe message, flags = rec_timeout | resynch (Gi0/1)
03:10:20: Pr (Gi0/1)
03:10:21: UDLD send probe message, flags = rec_timeout | resynch (Gi0/1)
03:10:21: Pr (Gi0/1)
03:10:22: UDLD send probe message, flags = rec_timeout | resynch (Gi0/1)
03:10:22: Pr (Gi0/1)
03:10:23: UDLD send probe message, flags = rec_timeout | resynch (Gi0/1)
03:10:23: Pr (Gi0/1)
03:10:24: UDLD send probe message, flags = rec_timeout | resynch (Gi0/1)
03:10:24: Pr (Gi0/1)
03:10:25: UDLD send flush message, flags = 0 (Gi0/1)
03:10:25: F (Gi0/1)
03:10:25: %UDLD-4-UDLD_PORT_DISABLED: UDLD disabled interface Gi0/1,
aggressive mode failure detected

```

The only knob that UDLD has available is how often UDLD messages are sent in steady-state operation. This can speed up UDLD “convergence”. The default is 15 seconds, but we can adjust it to any number between 1 and 90. I adjust it to 10 on ME1 only, since the timers don’t have to match. Once we set this, the details reveal it. ME1 uses a timer of 10, while it learns that ME2 uses the default timer of 15.

```

! ME1
udld message time 10

ME1#show udld gigabitEthernet 0/1 | include Message|Device
Message interval: 10
  Device ID: 1
  Device name: FOC1510V51X
  Message interval: 15
  CDP Device name: ME2

```

UDLD packet debugging reveals this new timer in use. I omit the received probe information since it is verbose and not valuable for this verification. We can see ME1 sending probes 10 seconds apart now.

```

ME1#debug udld packets
UDLD packets debugging is on

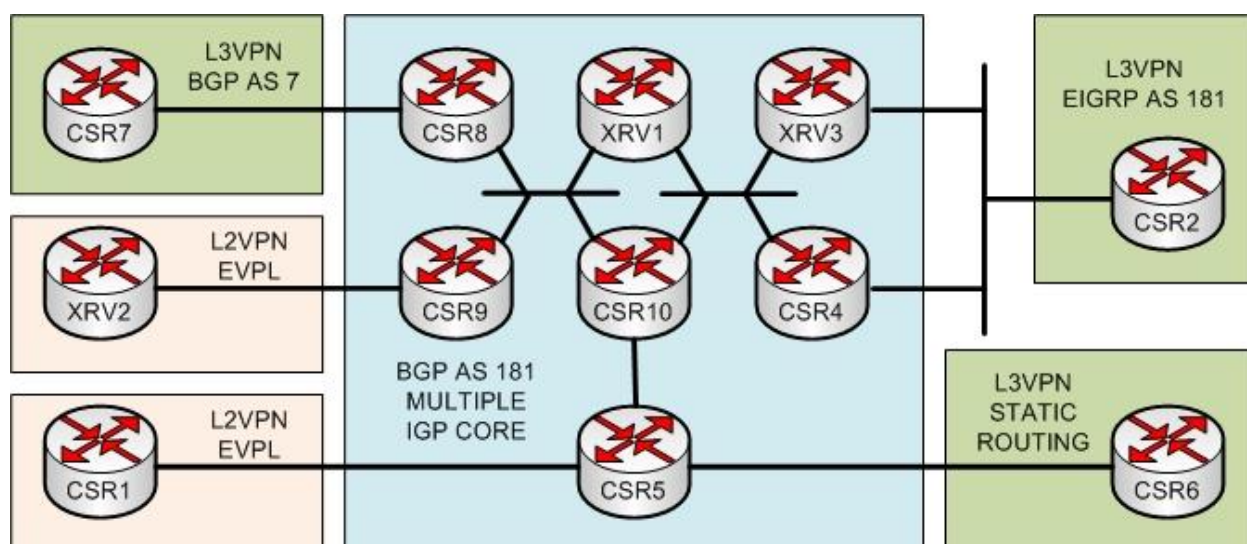
03:15:29: UDLD send probe message, flags = rec_timeout (Gi0/1)
03:15:29: P (Gi0/1)
03:15:39: UDLD send probe message, flags = rec_timeout (Gi0/1)
03:15:39: P (Gi0/1)

```

40. Describe, implement, and troubleshoot Layer 3 failure detection

40.1 Individual Protocol Hello packets

Most protocols have their own neighbor discovery mechanism, which is typically a hello packet of sorts. This is true for EIGRP, OSPF, IS-IS, RSVP, FHRPs, BGP, LDP, PIM, and others. There are typically two timers associated with each: an interval that determines how frequently to send hellos, and a hold/dead timer that says how long one can go without receiving one before declaring the neighbor dead. The timer behavior varies significantly between protocols and is discussed in this section. The network diagram includes XE and XR routers to see the configuration/verification on both platforms.



First, we will examine OSPF timers briefly. The hello and dead timers must match on all peers sharing an interface/network. This can be changed per-link. The hello/dead timers are easily understood and there are no magic tricks with OSPF. If the dead-interval is not specified, it is automatically assumed to be four times the hello interval. We will adjust the hello/dead timers to 2/6 seconds on the OSPF LAN.

```
! CSR8, CSR9, CSR10
interface GigabitEthernet2.518
 ip ospf dead-interval 6
 ip ospf hello-interval 2

! XRV1
router ospf 181
 area 0
 interface GigabitEthernet0/0/0/0.518
  dead-interval 6
  hello-interval 2
```


A quick debug on XRv1 and CSR8 shows hellos being send and received as expected. XR shows more detailed output to include the DR/BDR information, timers, and other pieces of information.

```
RP/0/0/CPU0:XRv1#debug ospf 181 hello gig0/0/0/0.518
ospf[1018]: Send hello to 224.0.0.5 area 0 on GigabitEthernet0/0/0/0.518 from
181.18.90.11 (nbr/if state 3/7)
ospf[1018]: Send hello pkt pri 0 options 0x12 DR 181.18.90.9 BDR 181.18.90.10
hello 2 dead 6 netmask 255.255.255.0, vrf default vrfid 0x60000000
ospf[1018]: Rcv hello from 181.0.0.9 area 0 from GigabitEthernet0/0/0/0.518
181.18.90.9 (nbr/if state 3/7) vrf default vrfid 0x60000000
ospf[1018]: Rcv hello pkt pri 5 options 0x12 DR 181.18.90.9 BDR 181.18.90.10
hello 2 dead 6 netmask 255.255.255.0 (nbr_dr 181.18.90.9 nbr_bdr
181.18.90.10)
ospf[1018]: End of hello processing
ospf[1018]: Rcv hello from 181.0.0.8 area 0 from GigabitEthernet0/0/0/0.518
181.18.90.8 (nbr/if state 3/7) vrf default vrfid 0x60000000
ospf[1018]: Rcv hello pkt pri 0 options 0x12 DR 181.18.90.9 BDR 181.18.90.10
hello 2 dead 6 netmask 255.255.255.0 (nbr_dr 181.18.90.9 nbr_bdr
181.18.90.10)
ospf[1018]: End of hello processing
ospf[1018]: Rcv hello from 181.0.0.10 area 0 from GigabitEthernet0/0/0/0.518
181.18.90.10 (nbr/if state 3/7) vrf default vrfid 0x60000000
ospf[1018]: Rcv hello pkt pri 3 options 0x12 DR 181.18.90.9 BDR 181.18.90.10
hello 2 dead 6 netmask 255.255.255.0 (nbr_dr 181.18.90.9 nbr_bdr
181.18.90.10)
ospf[1018]: End of hello processing
```

```
R8#debug ip ospf hello
OSPF-181 HELLO Gi2.518: Rcv hello from 181.18.90.11 area 0 181.18.90.11
OSPF-181 HELLO Gi2.518: Send hello to 224.0.0.5 area 0 from 181.18.90.8
OSPF-181 HELLO Gi2.518: Rcv hello from 181.0.0.10 area 0 181.18.90.10
OSPF-181 HELLO Gi2.518: Rcv hello from 181.0.0.9 area 0 181.18.90.9
```

We will configure CSR9 to have a different set of timers so that the neighborhood fails. Both XE and XR show these failures in the debug messages. Both XE and XR output uses “R” and “C” for Remote and Configured, respectively. The remote value is what the peer is using, while configured is the local value. We can see that CSR9 is using a dead time of 13 and a hello time of 6, while CSR8 expects a dead time of 6 and a hello time of 2. The subnet mask also much match on broadcast links, and in this case, it does.

```
! CSR9
interface GigabitEthernet2.518
 ip ospf dead-interval 6
 ip ospf hello-interval 13

! CSR8
OSPF-181 HELLO Gi2.518: Rcv hello from 181.0.0.9 area 0 181.18.90.9
OSPF-181 HELLO Gi2.518: Mismatched hello parameters from 181.18.90.9
```

```
OSPF-181 HELLO Gi2.518: Dead R 13 C 6, Hello R 6 C 2 Mask R 255.255.255.0 C 255.255.255.0
```

```
! XRv1
Rcv hello from 181.0.0.9 area 0 from GigabitEthernet0/0/0/0.518 181.18.90.9
(nbr/if state 3/7) vrf default vrfid 0x60000000
Mismatched hello parameters from 181.18.90.9
Dead R 13 C 6, Hello R 6 C 2 Mask R 255.255.255.0 C 255.255.255.0
hello from 181.0.0.9 area 0 failed validation
```

Before continuing, we will set CSR9 back to the proper values (not shown). We will quickly verify the timers on CSR9 as well as the neighbors. CSR9 is a DR/BDR candidate and should have full adjacencies with all nodes on the segment.

```
R9#show ip ospf interface gig2.518 | include Timer
Timer intervals configured, Hello 2, Dead 6, Wait 6, Retransmit 5
```

```
R9#show ip ospf neighbor gig2.518
```

Neighbor ID	Pri	State	Dead Time	Address	Interface
181.0.0.8	0	FULL/DROTHER	00:00:05	181.18.90.8	Gig2.518
181.0.0.10	3	FULL/BDR	00:00:05	181.18.90.10	Gig2.518
181.18.90.11	0	FULL/DROTHER	00:00:04	181.18.90.11	Gig2.518

For the fastest possible OSPF detection, we can set the dead timer to “minimal”, which is 1 second. We then specify how many hellos packets should be sent within that 1 second interval. A smaller number means less frequent hello packets. Below, we set the dead interval to 1 second with the hello interval to 1/3 of a second, or 333ms. OSPF can send hellos as fast as 50ms apart using a hello-multiplier of 20 (1/20th of a second). To ensure I don’t violate the CSR1000v 100 kbps bandwidth constraint, I only configure this on the link between CSR5 and CSR10. We also verify the timers are correct and that a neighbor has formed. We can verify this on one side, since a timer mismatch implies no neighbor. Notice the very short dead timer.

```
! CSR5 and CSR10
interface GigabitEthernet2.550
ip ospf dead-interval minimal hello-multiplier 3
```

```
R5#show ip ospf interface gig2.550 | include Timer
Timer intervals configured, Hello 333 msec, Dead 1, Wait 1, Retransmit 5
```

```
R5#show ip ospf neighbor gig2.550
```

Neighbor ID	Pri	State	Dead Time	Address	Interface
181.0.0.10	0	FULL/ -	990 msec	181.5.10.10	GigabitEthernet2.550

OSPFv3 is similar. We will use a hello timer of 2 and a dead timer of 8 for routers on the LAN. We quickly verify the timers as well.

```
! CSR8, CSR9, and CSR10
interface GigabitEthernet2.518
  ospfv3 181 hello-interval 2
  ospfv3 181 dead-interval 8
```

```
! XRv1
router ospfv3 181
  interface GigabitEthernet0/0/0/0.518
    dead-interval 8
    hello-interval 2
```

```
R9#show ospfv3 interface gig2.518 | include Timer
Timer intervals configured, Hello 2, Dead 8, Wait 8, Retransmit 5
```

```
RP/0/0/CPU0:XRv1#show ospfv3 interface gig0/0/0/0.518 | include Timer
Timer intervals configured, Hello 2, Dead 8, Wait 8, Retransmit 5
```

Because OSPFv2 fast hellos existed before BFD (discussed later) and OSPFv3 was created later, there was no good reason to support OSPFv3 fast hellos. Thus, like EIGRP, the slowest hello-interval is 1 second. By only specifying this, the dead-timer is automatically 4 times this value, which is 4 seconds.

```
! CSR5 and CSR10
interface GigabitEthernet2.550
  ospfv3 181 hello-interval 1
```

```
R10#show ospfv3 interface gig2.550 | include Timer
Timer intervals configured, Hello 1, Dead 4, Wait 4, Retransmit 5
```

IS-IS has a slightly different behavior with respect to matching timers. Timers do not need to match on both sides, and the IS-IS hellos carry a “hold-time” value. This value tells a peer “if you don’t receive a hello from me within this time period, consider me dead”. In this way, a different set of timers can be established per node. By default, the hold-time is 3 times the hello time, which is considered the hello-multiplier. Like many IS-IS commands, we can optionally specify the level to which these hellos apply. Since XRv1 is the DIS, the logic changes to act like the “minimal” configuration. That is to say, the hello interval is divided by the multiplier, but only on the DIS. If we want a 2 second hello interval from the DIS, we can use 6/3. Below, we configure different hello/hold timers on all 4 routers.

```
! XRv1
router isis 181
  interface GigabitEthernet0/0/0/0.513
    hello-interval 6
    hello-multiplier 3
```

```
! XRv3
router isis 181
```

```

interface GigabitEthernet0/0/0/0.513
  hello-interval 2 level 2

! CSR4
interface GigabitEthernet2.513
  isis hello-interval minimal
  isis hello-multiplier 3

! CSR10
interface GigabitEthernet2.513
  isis hello-interval 6 level-2

```

Based on this configuration, XRv1 uses 2/6, XRv3 uses 2/3 (3 is the default hello-multiplier), CSR4 uses 333ms/1 (similar to OSPF fast hellos), and CSR10 uses 6/18. The verification for this is very loose; we cannot clearly see the timer configuration for either XE or XR. We can only look at the interface details to see when the “next” hello packet will be sent.

```

R4#show clns interface gig2.513 | include Hello
Next IS-IS LAN Level-2 Hello in 196 milliseconds

```

```

R10#show clns interface gig2.513 | include Hello
Next IS-IS LAN Level-2 Hello in 3 seconds

```

```

RP/0/0/CPU0:XRv3#show isis interface gig0/0/0/0.513 | include IIH
Next LAN IIH in:      1 s

```

```

RP/0/0/CPU0:XRv1#show isis interface gig0/0/0/0.513 | include IIH
Next LAN IIH in:      461 ms

```

Likewise, when looking at a neighbor, we can see the hold-time as it counts down, but not the neighbors configured value. From the DIS’ perspective, CSR4’s hold-time is sub-second so the value shows as 0. CSR10 has a hold time of 18 (6 * 3) so a number like 16 makes sense. XRv3’s hold time is 6 seconds (2 * 3) so a number like 5 makes sense.

```

RP/0/0/CPU0:XRv1#show isis neighbors
IS-IS 181 neighbors:

```

System Id	Interface	SNPA	State	Holdtime	Type	IETF-NSF
R4	Gi0/0/0/0.513	0050.56a9.2c57	Up	0	L2	Capable
R10	Gi0/0/0/0.513	0050.56a9.f961	Up	16	L2	Capable
XRv3	Gi0/0/0/0.513	0050.56a9.ea54	Up	5	L2	Capable

To prove the awkwardness of the DIS timers, we can enable IS-IS hello debugging on XRv1 and CSR10. XRv1 will receive many hellos from CSR4 because it has a sub second hello interval (omitted). We can see the hello packets being originated from XRv1 using the DIS’ hostname-based NET (XRv1.01), and being sent every ~2 seconds despite configuring a 6 second hello interval (6 / 3).

```

RP/0/0/CPU0:XRv1#debug isis adjacencies
13:58:13.380 : isis[1010]: RECV L2 LAN IIH from GigabitEthernet0/0/0/0.513
SNPA 0050.56a9.ea54: System ID XRv3, Holdtime 6, LAN ID XRv1.01, length 90
13:58:13.490 : isis[1010]: RECV L2 LAN IIH from GigabitEthernet0/0/0/0.513
SNPA 0050.56a9.2c57: System ID R4, Holdtime 1, LAN ID XRv1.01, length 90
[snip, R4 hellos]
13:58:13.960 : isis[1010]: SEND L2 LAN IIH on GigabitEthernet0/0/0/0.513:
isis_osi_send_lan_hello 1491 LAN ID XRv1.01, 3 neighbors, Holdtime 6s, Length
90
13:58:14.150 : isis[1010]: RECV L2 LAN IIH from GigabitEthernet0/0/0/0.513
SNPA 0050.56a9.2c57: System ID R4, Holdtime 1, LAN ID XRv1.01, length 90
[snip, R4 hellos]
13:58:15.000 : isis[1010]: RECV L2 LAN IIH from GigabitEthernet0/0/0/0.513
SNPA 0050.56a9.ea54: System ID XRv3, Holdtime 6, LAN ID XRv1.01, length 90
[snip, R4 hellos]
13:58:15.550 : isis[1010]: SEND L2 LAN IIH on GigabitEthernet0/0/0/0.513:
isis_osi_send_lan_hello 1491 LAN ID XRv1.01, 3 neighbors, Holdtime 6s, Length
90
13:58:15.660 : isis[1010]: RECV L2 LAN IIH from GigabitEthernet0/0/0/0.513
SNPA 0050.56a9.2c57: System ID R4, Holdtime 1, LAN ID XRv1.01, length 90
[snip, R4 hellos]
13:58:16.890 : isis[1010]: RECV L2 LAN IIH from GigabitEthernet0/0/0/0.513
SNPA 0050.56a9.f961: System ID R10, Holdtime 18, LAN ID XRv1.01, length 90
13:58:17.010 : isis[1010]: RECV L2 LAN IIH from GigabitEthernet0/0/0/0.513
SNPA 0050.56a9.ea54: System ID XRv3, Holdtime 6, LAN ID XRv1.01, length 90
[snip, R4 hellos]
13:58:17.540 : isis[1010]: SEND L2 LAN IIH on GigabitEthernet0/0/0/0.513:
isis_osi_send_lan_hello 1491 LAN ID XRv1.01, 3 neighbors, Holdtime 6s, Length
90

```

CSR10 receives hellos from all other neighbors (shown by MAC address) which all show the DIS' NET minus the area (0000.0000.0011.01). Hellos can be jittered up to 25% so the interval is not exactly 2 seconds. Also, we see that CSR10 sends a single hello for every ~3 received from the DIS as its interval is 3 times slower.

```

R10#show ip arp 181.13.40.11
Protocol Address Age (min) Hardware Addr Type Interface
Internet 181.13.40.11 34 0050.56a9.9c60 ARPA Gig2.513

R10#debug isis adj-packets
14:01:47.932: ISIS-Adj: Sending L2 LAN IIH on GigabitEthernet2.513, length 90
14:01:48.074: ISIS-Adj: Rec L2 IIH from 0050.56a9.9c60
(GigabitEthernet2.513), cir type L2, cir id 0000.0000.0011.01, length 90,
ht(6)
14:01:48.074: ISIS-Adj: tid 0
14:01:48.074: ISIS-Adj: ,2
14:01:48.074: ISIS-Adj: he_knows_us 1, old state 0, new state 0, level 2
[snip, other hellos]

```

```

14:01:49.694: ISIS-Adj: Rec L2 IIH from 0050.56a9.9c60
(GigabitEthernet2.513), cir type L2, cir id 0000.0000.0011.01, length 90,
ht(6)
14:01:49.694: ISIS-Adj: tid 0
14:01:49.694: ISIS-Adj: ,2
14:01:49.694: ISIS-Adj: he_knows_us 1, old state 0, new state 0, level 2
14:01:51.554: ISIS-Adj: Rec L2 IIH from 0050.56a9.9c60
(GigabitEthernet2.513), cir type L2, cir id 0000.0000.0011.01, length 90,
ht(6)
14:01:51.554: ISIS-Adj: tid 0
14:01:51.554: ISIS-Adj: ,2
14:01:51.554: ISIS-Adj: he_knows_us 1, old state 0, new state 0, level 2

```

EIGRP behaves similarly to IS-IS with respect to its per-neighbor treatment of hello/hold timers. The difference is that there isn't a pseudonode concept since EIGRP is not link-state, so the additional complexity observed in IS-IS doesn't apply. EIGRP also does not have a sub-second hello option, so the fastest possible failure detection is 1 second; hello timer is 1 second with a 1 second hold-time. This is probably highly unstable and a hold-time of at least 2 seconds would be preferred in this case. There is no difference between IPv4 and IPv6, so only IPv4 is examined here. XRv3 and CSR2 have timer adjustments shown below, while CSR4 uses the default timers of 5/15 (no configuration changes).

```

! CSR2
router eigrp L3
  address-family ipv4 unicast autonomous-system 181
    af-interface default
      hello-interval 4
      hold-time 16

! XRv3
router eigrp L3
  vrf L3
    address-family ipv4
      interface GigabitEthernet0/0/0/0.523
        hello-interval 3
        hold-time 9

```

We can quickly verify the configuration was successful by checking the interface details. Notice that all of the timers are different.

```

R4#show eigrp address-family ipv4 vrf L3 interfaces detail | include Hello-
Hello-interval is 5, Hold-time is 15

```

```

R2#show eigrp address-family ipv4 interfaces detail | include Hello-
Hello-interval is 4, Hold-time is 16

```

```

RP/0/0/CPU0:XRv3#show eigrp vrf L3 ipv4 interfaces detail | include Hello
Hello interval is 3 sec, hold time is 9 sec

```

Checking the EIGRP neighbors on CSR2, we can see the per-neighbor hold times, just like IS-IS. XRv3 has a hold time of 9 seconds, so a value of 8 makes sense. CSR4 has a hold time of 16 seconds, so a value of 12 makes sense. Similarly for XRv3, CSR4 and CSR2 both have reasonable hold times (that is, values less than or equal to their configured hold times), which shows a correct configuration.

```
R2#show eigrrp address-family ipv4 neighbors
EIGRP-IPv4 VR(L3) Address-Family Neighbors for AS(181)
H   Address                Interface      Hold  Uptime    SRTT    RTO    Q    Seq
                               (sec)                (ms)    Cnt  Num
1   10.23.4.13              Gi2.523       8     01:18:02   51     306    0   162
0   10.23.4.4                Gi2.523      12     1d22h     27     162    0   110
```

```
RP/0/0/CPU0:XRv3#show eigrrp vrf L3 ipv4 neighbors
IPv4-EIGRP VR(L3) Neighbors for AS(181) VRF L3
H   Address                Interface      Hold  Uptime    SRTT    RTO    Q    Seq
                               (sec)                (ms)    Cnt  Num
1   10.23.4.4                Gi0/0/0/0.523 10     02:43:02  505     3030    0   118
0   10.23.4.2                Gi0/0/0/0.523 13     02:43:02    1       200    0    79
```

EIGRP hello debugging on CSR2 also shows packets being sent and received. Packets from CSR4 are 5 seconds apart (yellow), while packets received from XRv13 are 3 seconds apart (green). Packets sent from CSR2 are transmitted every 4 seconds, but only one is shown here (cyan).

```
! CSR2
20:52:09.736: EIGRP: received packet with MD5 authentication, key id = 1
20:52:09.736: EIGRP: Received HELLO on Gi2.523 - paklen 60 nbr 10.23.4.4
20:52:09.736:   AS 181, Flags 0x0:(NULL), Seq 0/0 interfaceQ 0/0 iidbQ
un/rely 0/0 peerQ un/rely 0/0
20:52:10.298: EIGRP: received packet with MD5 authentication, key id = 1
20:52:10.298: EIGRP: Received HELLO on Gi2.523 - paklen 60 nbr FE80::13
20:52:10.298:   AS 181, Flags 0x0:(NULL), Seq 0/0 interfaceQ 0/0 all iidbQ
un/rely 0/0 peerQ un/rely 0/0
20:52:10.597: EIGRP: received packet with MD5 authentication, key id = 1
20:52:10.597: EIGRP: Received HELLO on Gi2.523 - paklen 60 nbr 10.23.4.13
20:52:10.597:   AS 181, Flags 0x0:(NULL), Seq 0/0 interfaceQ 0/0 iidbQ
un/rely 0/0 peerQ un/rely 0/0
20:52:11.160: EIGRP: received packet with MD5 authentication, key id = 1
20:52:11.160: EIGRP: Received HELLO on Gi2.523 - paklen 60 nbr FE80::4
20:52:11.160:   AS 181, Flags 0x0:(NULL), Seq 0/0 interfaceQ 0/0 iidbQ
un/rely 0/0 peerQ un/rely 0/0
20:52:13.456: EIGRP: received packet with MD5 authentication, key id = 1
20:52:13.456: EIGRP: Received HELLO on Gi2.523 - paklen 60 nbr 10.23.4.13
20:52:13.456:   AS 181, Flags 0x0:(NULL), Seq 0/0 interfaceQ 0/0 iidbQ
un/rely 0/0 peerQ un/rely 0/0
20:52:13.548: EIGRP: Sending HELLO on Gi2.523 - paklen 66
20:52:13.548:   AS 181, Flags 0x0:(NULL), Seq 0/0 interfaceQ 0/0 iidbQ
un/rely 0/0
```

```
20:52:14.681: EIGRP: received packet with MD5 authentication, key id = 1
20:52:14.681: EIGRP: Received HELLO on Gi2.523 - paklen 60 nbr 10.23.4.4
20:52:14.681:   AS 181, Flags 0x0:(NULL), Seq 0/0 interfaceQ 0/0 iidbQ
un/rely 0/0 peerQ un/rely 0/0
```

Output from XR is verbose, especially with authentication enabled. Sent packets are shown in cyan, with received packets shows in yellow and green from CSR4 and CSR2, respectively.

```
! XRv3
20:58:11.402 : eigrp[1002]: IPv4-EIGRP(L3-181): interface
GigabitEthernet0_0_0_0.523, keychain KC_GEN_AUTH, Found live authentication
key(id = 1, alg = 3)
20:58:11.402 : eigrp[1002]: IPv4-EIGRP(L3-181): interface
GigabitEthernet0_0_0_0.523, packet being sent with key(id = 1)
20:58:11.402 : eigrp[1002]: IPv4-EIGRP(L3-181): Sending HELLO on
Gi0/0/0/0.523
20:58:11.402 : eigrp[1002]: IPv4-EIGRP(L3-181): AS 181, Flags 0x00000000, Seq
0/0 iidbQ un/rely 0/0 ddb Gr/Nsf/NiP 1/1/0

20:58:13.201 : eigrp[1002]: IPv4-EIGRP(L3-181): interface
GigabitEthernet0_0_0_0.523, received authenticated packet, key id = 1
20:58:13.201 : eigrp[1002]: IPv4-EIGRP(L3-181): interface
GigabitEthernet0_0_0_0.523, pkt authentication, local key(id = 1)
20:58:13.201 : eigrp[1002]: IPv4-EIGRP(L3-181): interface
GigabitEthernet0_0_0_0.523, packet authentication key id = 1, authentication
data is valid
20:58:13.201 : eigrp[1002]: IPv4-EIGRP(L3-181): Received HELLO on
Gi0/0/0/0.523 nbr 10.23.4.4
20:58:13.201 : eigrp[1002]: IPv4-EIGRP(L3-181): AS 181, Flags 0x00000000, Seq
0/0 iidbQ un/rely 0/0 ddb Gr/Nsf/NiP 1/1/0 peer
Cr/0/NIn/0/NiA/0/NiP/0/Eot/0/uNS/1/IGr/0/LAck/118 peerQ un/rely 0/0

20:58:13.321 : eigrp[1002]: IPv4-EIGRP(L3-181): interface
GigabitEthernet0_0_0_0.523, received authenticated packet, key id = 1
20:58:13.321 : eigrp[1002]: IPv4-EIGRP(L3-181): interface
GigabitEthernet0_0_0_0.523, pkt authentication, local key(id = 1)
20:58:13.321 : eigrp[1002]: IPv4-EIGRP(L3-181): interface
GigabitEthernet0_0_0_0.523, packet authentication key id = 1, authentication
data is valid
20:58:13.321 : eigrp[1002]: IPv4-EIGRP(L3-181): Received HELLO on
Gi0/0/0/0.523 nbr 10.23.4.2
20:58:13.321 : eigrp[1002]: IPv4-EIGRP(L3-181): AS 181, Flags 0x00000000, Seq
0/0 iidbQ un/rely 0/0 ddb Gr/Nsf/NiP 1/1/0 peer
Cr/0/NIn/0/NiA/0/NiP/0/Eot/0/uNS/1/IGr/0/LAck/79 peerQ un/rely 0/0

20:58:13.981 : eigrp[1002]: IPv4-EIGRP(L3-181): interface
GigabitEthernet0_0_0_0.523, keychain 'KC_GEN_AUTH' Valid authentication keys
found (id = 1)
```



```

20:58:13.981 : eigrp[1002]: IPv4-EIGRP(L3-181): interface
GigabitEthernet0_0_0_0.523, keychain KC_GEN_AUTH, Found live authentication
key(id = 1, alg = 3)
20:58:13.981 : eigrp[1002]: IPv4-EIGRP(L3-181): interface
GigabitEthernet0_0_0_0.523, packet being sent with key(id = 1)
20:58:13.981 : eigrp[1002]: IPv4-EIGRP(L3-181): Sending HELLO on
Gi0/0/0/0.523
20:58:13.981 : eigrp[1002]: IPv4-EIGRP(L3-181): AS 181, Flags 0x00000000, Seq
0/0 iidbQ un/rely 0/0 ddb Gr/Nsf/NiP 1/1/0

```

RIP timers aren't really used for failure detection since RIP packets actually carry routes; it doesn't have a dedicated neighbor discovery/maintenance mechanism. We can change the frequency at which the routing advertisements are sent, invalidated, and flushed. We use RIP and RIPvng as the PE-CE routing protocol between CSR5 and CSR6 for a quick demonstration. The only caveat is that the remote hello timer must be less than the local invalid timer. If it isn't, then the routing advertisements will come in too slowly, and the receiving router will have periods of downtime since the routers are flushed too quickly. Below is an example of this phenomenon. CSR5 uses the default timers (30/180/180/240) while CSR6 uses new, aggressive timers. CSR6 sends updates every 5 seconds and flushes received updates after 15 seconds. CSR5 is sending updates every 30 seconds yet CSR6 can only retain the for 15 seconds. For the remaining ~15 seconds, CSR6 will have no routes from CSR5. CSR6's advertisements are shown in yellow, and the received routes and subsequent downtime is shown in green.

```

! CSR6
router rip
  timers basic 5 15 15 20

R6#debug ip rip
R6#debug ip routing
22:39:53.022: RIP: received v2 update from 10.5.6.5 on GigabitEthernet2.556
22:39:53.022:      10.7.7.7/32 via 0.0.0.0 in 8 hops
22:39:53.022: RT: updating rip 10.7.7.7/32 (0x0) :
10.5.6.5 Gi2.556 0 1048578
22:39:53.022: RT: add 10.7.7.7/32 via 10.5.6.5, rip metric [120/8]
22:39:55.023: RIP: sending v2 flash update to 224.0.0.9 via
GigabitEthernet2.556 (10.5.6.6)
22:39:55.023: RIP: build flash update entries - suppressing null update
22:39:56.369: RIP: sending v2 update to 224.0.0.9 via GigabitEthernet2.556
(10.5.6.6)
22:39:56.369: RIP: build update entries
22:39:56.369:      10.6.6.6/32 via 0.0.0.0, metric 1, tag 0
22:39:56.369:      10.66.66.66/32 via 0.0.0.0, metric 1, tag 0
22:40:01.324: RIP: sending v2 update to 224.0.0.9 via GigabitEthernet2.556
(10.5.6.6)
22:40:01.325: RIP: build update entries
22:40:01.325:      10.6.6.6/32 via 0.0.0.0, metric 1, tag 0
22:40:01.325:      10.66.66.66/32 via 0.0.0.0, metric 1, tag 0

```

```

22:40:05.711: RIP: sending v2 update to 224.0.0.9 via GigabitEthernet2.556
(10.5.6.6)
22:40:05.711: RIP: build update entries
22:40:05.711: 10.6.6.6/32 via 0.0.0.0, metric 1, tag 0
22:40:05.711: 10.66.66.66/32 via 0.0.0.0, metric 1, tag 0
22:40:10.062: RIP: sending v2 update to 224.0.0.9 via GigabitEthernet2.556
(10.5.6.6)
22:40:10.062: RIP: build update entries
22:40:10.062: 10.6.6.6/32 via 0.0.0.0, metric 1, tag 0
22:40:10.062: 10.66.66.66/32 via 0.0.0.0, metric 1, tag 0
22:40:14.570: RIP: sending v2 update to 224.0.0.9 via GigabitEthernet2.556
(10.5.6.6)
22:40:14.570: RIP: build update entries
22:40:14.570: 10.6.6.6/32 via 0.0.0.0, metric 1, tag 0
22:40:14.570: 10.66.66.66/32 via 0.0.0.0, metric 1, tag 0
22:40:17.555: RT: del 10.7.7.7 via 10.5.6.5, rip metric [120/8]
22:40:17.555: RT: delete subnet route to 10.7.7.7/32

```

To fix it, we mirror the timers on CSR5 under the VRF. Additionally, we configure “fast” timers on both nodes for RIPng as well. Also note that RIPng is not VRF away by default and requires a global command to enable that feature set (this would have been done during the initial L3VPN setup, but I draw attention to it here).

```

! CSR5
ipv6 rip vrf-mode enable

router rip
 address-family ipv4 vrf L3
   timers basic 5 15 15 20

ipv6 router rip L3
 address-family ipv6 vrf L3
   timers 5 15 15 20

! CSR6
ipv6 router rip CUST
 timers 5 15 15 20

```

We can verify updates are being passed back and forth successfully. We can see packets being sent and received 5 seconds apart, which is correct. Updates in green are being sent from CSR6 and updates in yellow are received by CSR6 from CSR5.

```

R6#debug ipv6 rip
22:48:44.530: RIPng [Gi2.556, default VRF]: process "CUST" is sending a
multicast update.
22:48:44.530: src=FE80::6
22:48:44.530: dst=FF02::9 (GigabitEthernet2.556)
22:48:44.530: sport=521, dport=521, length=72

```

```

22:48:44.530:      command=2, version=1, mbz=0, #rte=3
22:48:44.530:      tag=0, metric=1, prefix>::10:6:6:6/128
22:48:44.530:      tag=0, metric=1, prefix=FD00:10:5:6::/64
22:48:44.530:      tag=0, metric=1, prefix>::10:66:66:66/128
22:48:44.530: RIPng [default VRF]: a message has been received.

22:48:44.893: RIPng [default VRF]: a message has been received.
22:48:44.893: RIPng [Gi2.556, default VRF]: response received from FE80::5
for process "CUST".
22:48:44.893:      src=FE80::5 (GigabitEthernet2.556)
22:48:44.893:      dst=FF02::9
22:48:44.893:      sport=521, dport=521, length=32
22:48:44.893:      command=2, version=1, mbz=0, #rte=2
22:48:44.893:      tag=0, metric=1, prefix=FD00:10:5:6::/64
22:48:44.893:      tag=0, metric=8, prefix>::10:7:7:7/128

22:48:49.530: RIPng [Gi2.556, default VRF]: process "CUST" is sending a
multicast update.
22:48:49.530:      src=FE80::6
22:48:49.530:      dst=FF02::9 (GigabitEthernet2.556)
22:48:49.530:      sport=521, dport=521, length=72
22:48:49.530:      command=2, version=1, mbz=0, #rte=3
22:48:49.530:      tag=0, metric=1, prefix>::10:6:6:6/128
22:48:49.530:      tag=0, metric=1, prefix=FD00:10:5:6::/64
22:48:49.530:      tag=0, metric=1, prefix>::10:66:66:66/128
22:48:49.530: RIPng [default VRF]: a message has been received.

22:48:49.893: RIPng [default VRF]: next RIB walk in 10000.
22:48:49.893: RIPng [default VRF]: a message has been received.
22:48:49.893: RIPng [Gi2.556, default VRF]: response received from FE80::5
for process "CUST".
22:48:49.893:      src=FE80::5 (GigabitEthernet2.556)
22:48:49.893:      dst=FF02::9
22:48:49.893:      sport=521, dport=521, length=32
22:48:49.893:      command=2, version=1, mbz=0, #rte=2
22:48:49.893:      tag=0, metric=1, prefix=FD00:10:5:6::/64
22:48:44.893:      tag=0, metric=8, prefix>::10:7:7:7/128

```

HSRP timers must match for the process to converge, but they can be configured differently. Whichever device is the active router for a given group will propagate its timers to the standby nodes. In this way, the configuration need not match, since the HSRP process ensures the timers are carried in the hello messages. This is similar to STP where the timers of the root bridge are carried in BPDUs for all non-root switches to honor, but individually switches can have entirely different timer settings. The default HSRP timers are 3/10 for hello/hold, so we will modify it to 4/12 on XRv3, while leaving the CSR4 timers unchanged.

```

! XRv3
router hsrp

```

```

interface GigabitEthernet0/0/0/0.523
  address-family ipv4
    hsrp 234 version 2
    timers 4 12

```

We can clearly see that XRv3 is the active router based on its higher priority (configuration not shown). CSR4 is the standby router but has inherited the new timers. XRv3 also shows us the configured timers as well, so comparing the configured and real-time timers is easier.

```

RP/0/0/CPU0:XRv3#show hsrp gig0/0/0/0.523 234
IPv4 Groups:

```

```

          P indicates configured to preempt.
          |
Interface      Grp  Pri P State Active addr Standby addr Group addr
Gi0/0/0/0.523  234 105 P Active local      10.23.4.4   10.23.4.254

```

```

R4#show standby gig2.524 234 | include State|Hello
State is Standby
Hello time 4 sec, hold time 12 sec

```

```

RP/0/0/CPU0:XRv3#show hsrp gig0/0/0/0.523 234 detail | include [Hh]ello
Hellotime 4000 msec holdtime 12000 msec
Configured hellotime 4000 msec holdtime 12000 msec

```

Since CSR4 is the active router for IPv6 hosts, we will change its timers using the smallest possible values, which is 15 ms and 50 ms. XRv3 accepts this value as it is learned via HSRP exchanges via the active router. Since HSRP runs different instances for IPv4 and IPv6, the two topologies can achieve rudimentary load-sharing (and entirely different configurations) in this way.

```

! CSR4
interface GigabitEthernet2.524
  standby 236 timers msec 15 msec 50

```

```

R4#show standby gig2.524 236 | include State|Hello
State is Active
Hello time 15 msec, hold time 50 msec

```

```

RP/0/0/CPU0:XRv3#show hsrp gig0/0/0/0.523 236 detail | include Hello
Hellotime 15 msec holdtime 50 msec

```

For debugging purposes, HSRP 236 is temporarily disabled to avoid flooding the log buffer/console. We will look at the IPv4 exchanges. Even with detailed debugging, there isn't much to see. The packets contain their timers, priorities, interface MAC addresses, and virtual IP addresses. The in/out hello packets show the source IP of the packet received as the destination is multicast (224.0.0.102 for HSRPv2). We can see that packets are sent ~4 seconds apart in either direction.

```

R4#debug standby packets hello detail
23:05:30.380: HSRP: Gi2.524 Grp 234 Hello out 10.23.4.4 Standby pri 100 vIP
10.23.4.254
23:05:30.380:      hel 4000 hol 12000 id 0050.56a9.2c57
23:05:31.951: HSRP: Gi2.524 Grp 234 Hello in 10.23.4.13 Active pri 105 vIP
10.23.4.254
23:05:31.951:      hel 4000 hol 12000 id 0050.56a9.ea54
23:05:33.819: HSRP: Gi2.524 Grp 234 Hello out 10.23.4.4 Standby pri 100 vIP
10.23.4.254
23:05:33.819:      hel 4000 hol 12000 id 0050.56a9.2c57
23:05:35.970: HSRP: Gi2.524 Grp 234 Hello in 10.23.4.13 Active pri 105 vIP
10.23.4.254
23:05:35.970:      hel 4000 hol 12000 id 0050.56a9.ea54

```

XRv3's debugging is very similar to CSR4's, showing the hello packets and their contents. One interesting difference is that XR doesn't appear to show outgoing HSRP packets, only incoming ones.

```

RP/0/0/CPU0:XRv3#debug hsrp packets hello detail
23:10:06.699 : hsrp[1095]: SB234: Gi0/0/0/0.523 Hello in 10.23.4.4
Standby pri 100 hel 4000 hol 12000 ip 10.23.4.254 ver 2
23:10:10.239 : hsrp[1095]: SB234: Gi0/0/0/0.523 Hello in 10.23.4.4
Standby pri 100 hel 4000 hol 12000 ip 10.23.4.254 ver 2
23:10:13.539 : hsrp[1095]: SB234: Gi0/0/0/0.523 Hello in 10.23.4.4
Standby pri 100 hel 4000 hol 12000 ip 10.23.4.254 ver 2

```

BGP timers can be configured differently on both sides, but the hold-time must match (after negotiation) for the session to form. BGP sessions use hello/hold timers of 60/180 by default on Cisco routers. Before doing anything, we will verify the default timers on CSR8. I show internal and external VPNv4 sessions and a single external VPNv6 session to prove it.

```

R8#show bgp vpnv4 unicast all neighbors | include ^BGP|hold_time
BGP neighbor is 7.7.7.7, vrf L3, remote AS 7, external link
  Last read 00:00:17, last write 00:00:32, hold time is 180, keepalive
interval is 60 seconds
BGP neighbor is 181.0.0.4, remote AS 181, internal link
  Last read 00:00:28, last write 00:00:37, hold time is 180, keepalive
interval is 60 seconds
BGP neighbor is 181.0.0.5, remote AS 181, internal link
  Last read 00:00:16, last write 00:00:43, hold time is 180, keepalive
interval is 60 seconds
BGP neighbor is 181.0.0.13, remote AS 181, internal link
  Last read 00:00:10, last write 00:00:46, hold time is 180, keepalive
interval is 60 seconds

R8#show bgp vpnv6 unicast vrf L3 neighbors fd00:10:7:8::7 | include hold_time
  Last read 00:00:00, last write 00:00:22, hold time is 180, keepalive
interval is 60 seconds

```

The hold-time is negotiated as part of the BGP session establishment where the smaller (faster) values are preferred. We can set a lower interval on XRv3 to prove this. A side effect of this is that the router who acknowledged the adjustment (CSR8, in this case) adjusts its keepalive interval to 1/3 of the new, smaller hold-time. This means the hello-interval decreases proportionally to the hold-time. In this case, XR follows suit and also uses 58 seconds despite being configured for 60.

```
! XRv3
router bgp 181
 neighbor 181.0.0.8
  timers 60 175
```

```
R8#show bgp vpnv4 unicast all neighbors 181.0.0.13 | include hold_time
Last read 00:00:37, last write 00:00:37, hold time is 175, keepalive
interval is 58 seconds
```

```
RP/0/0/CPU0:XRv3#show bgp vpnv4 unicast neighbors 181.0.0.8 | include hold
time
Hold time is 175, keepalive interval is 58 seconds
Configured hold time: 175, keepalive: 60, min acceptable hold time: 3
```

A more extreme example would be setting much lower timers, such as 10/40, and observing the remote side. We configure 10/40 timers on CSR8 towards CSR5, who then calculates its new keepalive timer as 13 seconds. Like in XR, there is a difference between configured and actual hold times. When a local node makes explicit changes to a timer, the two sets of timers are shown together. Unlike XR, it seems that XE is less generous; it still uses its configured keepalive timer despite CSR5 adjusting its timer proportionally. Asymmetric timers are typically undesirable and add complexity while yielding little gain.

```
! CSR8
router bgp 181
 neighbor 181.0.0.5 timers 10 40
```

```
R5#show bgp vpnv4 unicast all neighbors 181.0.0.8 | include hold_time
Last read 00:00:00, last write 00:00:03, hold time is 40, keepalive
interval is 13 seconds
```

```
R8#show bgp vpnv4 unicast all neighbors 181.0.0.5 | include hold_time
Last read 00:00:06, last write 00:00:08, hold time is 40, keepalive
interval is 10 seconds
Configured hold time is 40, keepalive interval is 10 seconds
```

As a security mechanism to protect the router's control-plane from too many BGP keepalives, we can specify a minimum neighbor hold-time. This allows a BGP speaker to reject sessions that have overly aggressive hold-timers; this is ideal for PE routers. By default, this is set to 0, allowing any hold-down timers. We can verify this by checking CSR5, as an example.

```
R8#show bgp vpnv4 unicast all neighbors 181.0.0.5 | include Minimum
```

Minimum holdtime from neighbor is 0 seconds

We will configure CSR8, as a PE, to ensure CSR7 is using at least a 60 second hold-down timer. Because we have not changed CSR7 yet, and we know the default was 180, we can assume CSR7 will negotiate a hold-time of 60 when peering with CSR8. This is an acceptable hold-time. We verify these timers on CSR7 and CSR8; again we see that XE is stubborn and uses its locally configured values, despite the remote peer calculating its keepalive to be 1/3 of the negotiated hold-down timers.

```
! CSR8
router bgp 181
 address-family ipv6 vrf L3
  neighbor FD00:10:7:8::7 timers 15 60 60
```

```
R8#show bgp vpnv6 unicast vrf L3 neighbors FD00:10:7:8::7 | include
hold_?time
  Last read 00:00:15, last write 00:00:06, hold time is 60, keepalive
interval is 15 seconds
  Configured hold time is 60, keepalive interval is 15 seconds
  Minimum holdtime from neighbor is 60 seconds
```

```
R7#show bgp ipv6 unicast neighbors FD00:10:7:8::8 | include hold_?time
  Last read 00:00:00, last write 00:00:09, hold time is 60, keepalive
interval is 20 seconds
```

Assuming CSR7 wanted to use an aggressive hold-time, the PE would not allow it, and the session does not form. Both CSR7 and CSR8 generate BGP log messages to detail this error. CSR7 tries to use a hold-time of 59 seconds which is “too fast” for CSR8. The syslog messages make this error very apparent.

```
! CSR7
router bgp 181
  neighbor FD00:10:7:8::8 timers 15 59
```

```
! CSR8
%BGP-3-NOTIFICATION: sent to neighbor FD00:10:7:8::7 active 2/6 (unacceptable
hold time) 0 bytes
```

```
! CSR7
%BGP-3-NOTIFICATION: received from neighbor FD00:10:7:8::8 active 2/6
(unacceptable hold time) 0 bytes
```

This isn't very useful for the customer, since they legitimately may not know what the hold-time is supposed to be. The BGP debugs don't help, either, so the customer would need to contact the provider. Assume that the issue was fixed and that CSR7 is now using a hold-time of 60 seconds again. Debugging on CSR7 towards CSR8, we can see CSR8 sending keepalives to CSR7 every ~15 seconds (yellow). CSR7 is sending keepalives to CSR8 every ~20 seconds as well (green), and these values don't need to match.

```
R7#debug bgp ipv6 unicast keepalives
```

```
00:28:56.997: BGP: FD00:10:7:8::8 received KEEPALIVE, length (excl. header) 0
```

```
00:29:02.756: BGP: ses global FD00:10:7:8::8 (0x7F401571BBC8:1) Keep alive timer fired.
```

```
00:29:02.756: BGP: FD00:10:7:8::8 KEEPALIVE requested  
(bgp_keepalive_timer_expired)
```

```
00:29:02.756: BGP: ses global FD00:10:7:8::8 (0x7F401571BBC8:1) service  
keepalive IO request.
```

```
00:29:02.756: BGP: FD00:10:7:8::8 KEEPALIVE write request serviced in BGP_IO
```

```
00:29:10.310: BGP: FD00:10:7:8::8 received KEEPALIVE, length (excl. header) 0
```

```
00:29:20.164: BGP: ses global FD00:10:7:8::8 (0x7F401571BBC8:1) Keep alive timer fired.
```

```
00:29:20.164: BGP: FD00:10:7:8::8 KEEPALIVE requested  
(bgp_keepalive_timer_expired)
```

```
00:29:20.165: BGP: ses global FD00:10:7:8::8 (0x7F401571BBC8:1) service  
keepalive IO request.
```

```
00:29:20.165: BGP: FD00:10:7:8::8 KEEPALIVE write request serviced in BGP_IO
```

```
00:29:24.648: BGP: FD00:10:7:8::8 received KEEPALIVE, length (excl. header) 0
```

For completeness, we will do the same debugging on XRv3. Since XR is more generous than XE, it has reduced its keepalive timer to 58 seconds, along with CSR8, so we should be keepalives at approximately the same interval. Keepalives sent to CSR8 occur every ~58 seconds (yellow), and keepalives received from CSR8 also occur every ~58 seconds as expected (green). This simplifies debugging by ensuring that both the hello and hold timers are synchronized.

```
RP/0/0/CPU0:XRv3#debug bgp keepalive 181.0.0.8
```

```
00:36:25.824 : bgp[1052]: [default-ka]: Keepalive timer expired for 181.0.0.8
```

```
00:36:25.824 : bgp[1052]: [default-ka]: Keepalive timer started for  
181.0.0.8(loc 5)
```

```
00:36:25.824 : bgp[1052]: [default-ka]: KEEPALIVE sent to 181.0.0.8
```

```
00:36:35.864 : bgp[1052]: [default-iord]: KEEPALIVE received from 181.0.0.8
```

```
00:37:23.841 : bgp[1052]: [default-ka]: Keepalive timer expired for 181.0.0.8
```

```
00:37:23.841 : bgp[1052]: [default-ka]: Keepalive timer started for  
181.0.0.8(loc 5)
```

```
00:37:23.841 : bgp[1052]: [default-ka]: KEEPALIVE sent to 181.0.0.8
```

```
00:37:31.160 : bgp[1052]: [default-iord]: KEEPALIVE received from 181.0.0.8
```

PIM hellos are also very straightforward and the hold-time is non-configurable. It is automatically 3.5 times the hello timer received from each neighbor. As such, the timers can be completely different like EIGRP and IS-IS. The default hello timer is 30; we reduce it to 15 on CSR4 and 3 on XRv3 for IPv4/v6.

! CSR4


```

interface GigabitEthernet2.524
  ip pim query-interval 15
  ipv6 pim hello-interval 15

! Xrv3
router pim
vrf L3
  address-family ipv4
    hello-interval 3
  address-family ipv6
    hello-interval 3

```

We can quickly confirm the configuration was correct by checking the interface details on both routers for both address-families.

```

R4#show ip pim vrf L3 interface detail | include Hello
PIM Hello/Query interval: 15 seconds
PIM Hello packets in/out: 34334/36527

```

```

R4#show ipv6 pim vrf L3 interface gig2.524
Interface          PIM    Nbr    Hello    DR
                  Count  Intvl  Prior
Gi2.524            on     1      15      1
  Address: FE80::4
  DR      : FE80::13

```

```

RP/0/0/CPU0:XRv3#show pim vrf L3 ipv4 interface gig0/0/0/0.523
PIM interfaces in VRF L3
Address          Interface          PIM    Nbr    Hello    DR    DR
                  Count  Intvl  Prior
10.23.4.13      GigabitEthernet0/0/0/0.523  on     2      3      1      this system

```

```

RP/0/0/CPU0:XRv3#show pim vrf L3 ipv6 interface gig0/0/0/0.523
PIM interfaces in VRF L3
Interface          PIM    Nbr    Hello    DR
                  Count  Intvl  Prior
GigabitEthernet0/0/0/0.523  on     2      3      1
  Primary Address : fe80::13
  DR : this system

```

We will debug the IPv4 PIM hellos to show the variance in timers. Yellow represents CSR4 sending PIM hellos every 15 seconds, while green represents receiving PIM hellos from XRv3 every 3 seconds. Of note, IPv6 PIM requires non-LL addressing, unlike many other IPv6 protocols, and the PIMv6 hello debug is very verbose.

```

R4#debug ip pim vrf L3 hello
23:31:48.447: PIM(1): Send periodic v2 Hello on GigabitEthernet2.524 with
GenID = 2707292449

```

```

23:31:48.536: PIM(1): Received v2 hello on GigabitEthernet2.524 from
10.23.4.13
23:31:48.536: PIM(1): Ignored unknown option 2 in PIM Hello packet
23:31:48.536: PIM(1): Neighbor (10.23.4.13) Hello GENID = 21332

23:31:51.635: PIM(1): Received v2 hello on GigabitEthernet2.524 from
10.23.4.13
23:31:51.635: PIM(1): Ignored unknown option 2 in PIM Hello packet
23:31:51.635: PIM(1): Neighbor (10.23.4.13) Hello GENID = 21332

23:31:54.735: PIM(1): Received v2 hello on GigabitEthernet2.524 from
10.23.4.13
23:31:54.735: PIM(1): Ignored unknown option 2 in PIM Hello packet
23:31:54.735: PIM(1): Neighbor (10.23.4.13) Hello GENID = 21332

23:31:57.840: PIM(1): Received v2 hello on GigabitEthernet2.524 from
10.23.4.13
23:31:57.840: PIM(1): Ignored unknown option 2 in PIM Hello packet
23:31:57.840: PIM(1): Neighbor (10.23.4.13) Hello GENID = 21332

23:32:00.945: PIM(1): Received v2 hello on GigabitEthernet2.524 from
10.23.4.13
23:32:00.945: PIM(1): Ignored unknown option 2 in PIM Hello packet
23:32:00.945: PIM(1): Neighbor (10.23.4.13) Hello GENID = 21332

23:32:03.197: PIM(1): Send periodic v2 Hello on GigabitEthernet2.524 with
GenID = 2707292449

```

The XR debugs are concise, so much so that receiving a PIM hello doesn't even show the originator's IP address. It only indicates that a packet has been "read". Considering that this "read" event occurs ~15 seconds apart and there is only one other PIM router on the segment, we can assume this is from CSR4(yellow). The XRV3 originated packets are ~3 seconds apart, as expected (green).

```

RP/0/0/CPU0:XRv3#debug pim vrf L3 ipv4 io
23:34:46.048 : pim[1160]: [16] Read packet paklen 58 hdrlen 20
23:34:46.048 : pim[1160]: [16] Read 58 bytes of data

23:34:46.498 : pim[1160]: [16] pim_net_send: app_offset 20, nw_offset 0
23:34:46.498 : pim[1160]: [16] Sending packet, src 10.23.4.13, dest
224.0.0.13, datasize 42

23:34:49.598 : pim[1160]: [16] pim_net_send: app_offset 20, nw_offset 0
23:34:49.598 : pim[1160]: [16] Sending packet, src 10.23.4.13, dest
224.0.0.13, datasize 42

23:34:52.697 : pim[1160]: [16] pim_net_send: app_offset 20, nw_offset 0

```

```
23:34:52.697 : pim[1160]: [16] Sending packet, src 10.23.4.13, dest
224.0.0.13, datasize 42
```

```
23:34:55.797 : pim[1160]: [16] pim_net_send: app_offset 20, nw_offset 0
23:34:55.797 : pim[1160]: [16] Sending packet, src 10.23.4.13, dest
224.0.0.13, datasize 42
```

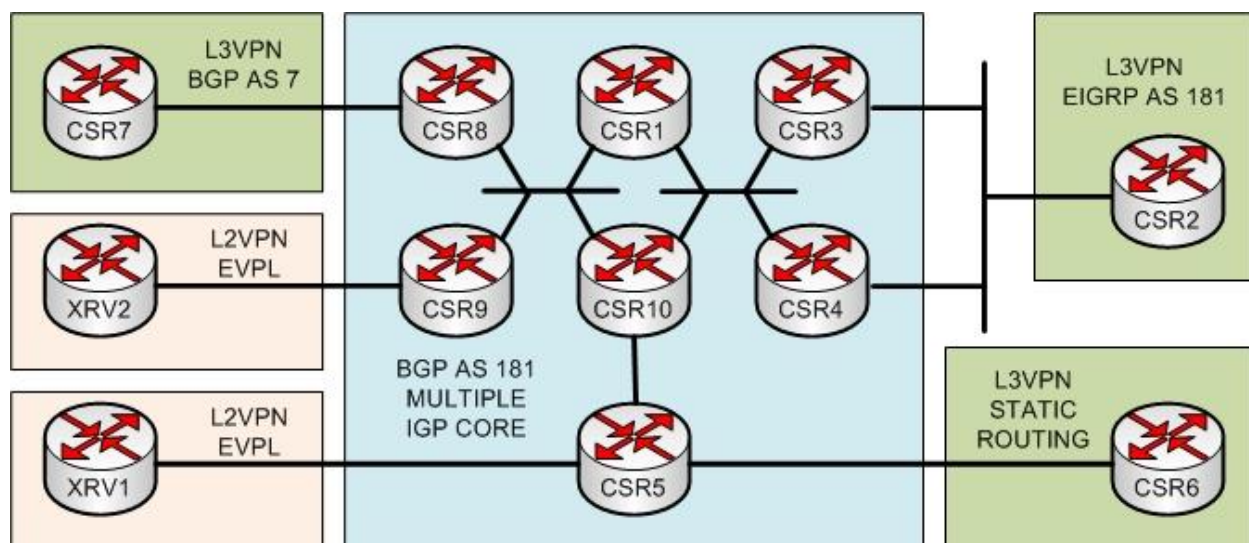
```
23:34:58.897 : pim[1160]: [16] pim_net_send: app_offset 20, nw_offset 0
23:34:58.897 : pim[1160]: [16] Sending packet, src 10.23.4.13, dest
224.0.0.13, datasize 42
```

```
23:35:00.607 : pim[1160]: [16] Read packet paklen 58 hdrln 20
23:35:00.607 : pim[1160]: [16] Read 58 bytes of data
```

Additional Reading – Reference configurations "hello"

40.2 Bidirectional Forwarding Detection (BFD)

Bidirectional Forwarding Detection (BFD) is a lightweight keepalive protocol design to reduce dead peer detection time across layer 2 networks. It is used primarily on networks that do not rely on line-protocol for interface status (like Ethernet). BFD uses a pair of UDP ports for transport and allows other protocols to "register" to it. BFD relies on a single session between its neighbors so that protocol-specific hello messages don't need to be aggressively tuned for fast fall-over. Individual protocols can be configured to use BFD; by default, none of them do. Some of these client protocols include EIGRP, OSPFv2/3, IS-IS, BGP, RSVP-TE, PIM, and xconnect. BFD uses echo messages to test reachability between neighbors and control messages for signaling. BFD does not perform neighbor detection; it relies on the registered protocol to do that, then runs probes based on those discovered IPv4/v6 neighbor addresses. The network diagram is shown below; large LANs are used for certain protocols rather than P2P links to demonstrate BFD behavior. It is similar to the previous section without XRv in the core as XRv does not support BFD (XRv1 is swapped with CSR1 and XRv3 is replaced by CSR3).



The echo function is on by default if you just enable BFD at link level using "bfd interval". Echoes are supported in BFD version 1, where version 0 only supported control packets. We can tell if echo is enabled by checking the BFD neighbor details. The OSPF LAN is configured using this method; the difference between echo and control packets are discussed later. One interesting note about BFD with OSPF is that, on broadcast networks, BFD sessions only form to the DR/BDR. In this case, CSR1 and CSR8 are not eligible to participate in the DR election, and thus do not form BFD peers with one another. Only CSR9 and CSR10 can be the BD/BDR.

```
! CSR1, CSR8, CSR9, CSR10
interface GigabitEthernet2.518
  bfd interval 900 min_rx 900 multiplier 3
```

```
R1#show bfd neighbors client ospf
```

```
IPv4 Sessions
```

NeighAddr	LD/RD	RH/RS	State	Int
181.18.90.9	4103/4100	Up	Up	Gi2.518
181.18.90.10	4104/4106	Up	Up	Gi2.518

```
R8#show bfd neighbors client ospf
```

```
IPv4 Sessions
```

NeighAddr	LD/RD	RH/RS	State	Int
181.18.90.9	4098/4097	Up	Up	Gi2.518
181.18.90.10	4100/4103	Up	Up	Gi2.518

```
R9#show bfd neighbors client ospf
```

```
IPv4 Sessions
```

NeighAddr	LD/RD	RH/RS	State	Int
181.18.90.1	4100/4103	Up	Up	Gi2.518
181.18.90.8	4097/4098	Up	Up	Gi2.518
181.18.90.10	4098/4101	Up	Up	Gi2.518

```
R10#show bfd neighbors client ospf
```

```
IPv4 Sessions
```

NeighAddr	LD/RD	RH/RS	State	Int
181.5.10.5	4102/4097	Up	Up	Gi2.550
181.18.90.1	4106/4104	Up	Up	Gi2.518
181.18.90.8	4103/4100	Up	Up	Gi2.518
181.18.90.9	4101/4098	Up	Up	Gi2.518

We can also check the OSPF process to ensure BFD is enabled. From CSR10, we can check an OSPF neighbor, interface, or process to see if BFD is enabled or not. BFD needs to be bound to OSPF (or any other protocol) on a per-interface basis. Using the command below under the OSPF process registers OSPF to BFD on all interfaces, which is typically desired.

```
! CSR1, CSR8, CSR9, CSR10
router ospf 181
  bfd all-interfaces
```

```
R10#show ip ospf neighbor 181.0.0.1 | include BFD
    In the area 0 via interface GigabitEthernet2.518, BFD enabled
```

```
R10#show ip ospf interface gig2.518 | include BFD
    Transmit Delay is 1 sec, State BDR, Priority 3, BFD enabled
```

```
R10#show ip ospf 181 | include BFD
    BFD is enabled
```

A quick check of the same routers shows similar results for OSPFv3 (assuming BFD is enabled for all interfaces). In this topology, CSR8 and CSR9 cannot be DR/BDR, and thus do not form BFD peers with one another. More accurately, because they do not form OSPF neighbors, the OSPF process does not inform BFD that a session should be created.

```
! CSR1, CSR8, CSR9, CSR10
router ospfv3 181
 address-family ipv6 unicast
  bfd all-interfaces
```

```
R1#show bfd neighbors client ospfv3
```

```
IPv6 Sessions
```

NeighAddr	LD/RD	RH/RS	State	Int
FE80::8	2/2	Up	Up	Gi2.518
FE80::9	1/2	Up	Up	Gi2.518
FE80::10	3/4	Up	Up	Gi2.518

```
R8#show bfd neighbors client ospfv3
```

```
IPv6 Sessions
```

NeighAddr	LD/RD	RH/RS	State	Int
FE80::1	2/2	Up	Up	Gi2.518
FE80::10	1/3	Up	Up	Gi2.518

```
R9#show bfd neighbors client ospfv3
```

```
IPv6 Sessions
```

NeighAddr	LD/RD	RH/RS	State	Int
FE80::1	2/1	Up	Up	Gi2.518
FE80::10	1/2	Up	Up	Gi2.518

```
R10#show bfd neighbors client ospfv3
```

```
IPv6 Sessions
```

NeighAddr	LD/RD	RH/RS	State	Int
FE80::1	4/3	Up	Up	Gi2.518
FE80::5	1/1	Up	Up	Gi2.550
FE80::8	3/1	Up	Up	Gi2.518
FE80::9	2/1	Up	Up	Gi2.518

Likewise, we check the OSPFv3 show commands to verify BFD is enabled.

```
R10#show ospfv3 neighbor 181.0.0.1 | include BFD
    In the area 0 via interface GigabitEthernet2.518, BFD enabled
```

```
R10#show ospfv3 interface gig2.518 | include BFD
    Transmit Delay is 1 sec, State DR, Priority 5, BFD enabled
```

```
R10#show ospfv3 181 | include BFD
    BFD is enabled
```

The output below shows that the echo feature is enabled using the timers we configured. This is an ideal configuration because the echoes are what make BFD lightweight on any router. From CSR9, we will look at CSR1 within the IPv4 AF. We see that OSPF is registered to BFD.

```
R9#show bfd neighbors ipv4 181.18.90.1 details | include ^Sess|^Regist
Session state is UP and using echo function with 900 ms interval.
Session Host: Software
Registered protocols: OSPF CEF
```

Interestingly, BFD echoes do not appear to be supported for IPv6 addresses. We will further reinforce this theory when we test BGP using non-LL addressing.

```
R8#show bfd neighbors ipv6 fe80::10 details | include echo
Session state is UP and not using echo function.
```

The BFD control packets are not lightweight like echoes and cannot be offloaded to hardware in all platforms. Below is a control packet on UDP port 3784 (0xEC8) and an echo on UDP port 3785 (0xEC9). The source UDP port for all BFD packets appears to be 49152 (0xC0000). The control ports are shown in cyan and the echo ports are shown in pink. The echo always has the source and destination IP set to the sender's address; this is done so the receiver can immediately loop the packet back out and not process it locally. The IP addressing is highlighted in green in both packets below. Notice the destination MAC of the echo is still CSR9's MAC which allows the frame to be delivered at layer 2, while the source is always CSR1's MAC (yellow/grey). The control packet is actually destined to CSR9 like a regular packet would and is punted to the CPU (RSP/RP) for processing.

```
R9#show interfaces gigabitEthernet 2 | include bia
    Hardware is CSR vNIC, address is 0050.56a9.d672 (bia 0050.56a9.d672)
```

```
R9#show ip arp 181.18.90.1
Protocol  Address          Age (min)  Hardware Addr  Type   Interface
Internet  181.18.90.1        38        0050.56a9.1aaa  ARPA   GigabitEth2.518
```

```
R9#monitor capture CAP match ipv4 protocol udp any any range 3784 3785
R9#show monitor capture CAP buffer detailed
    0  70    0.000000  181.18.90.1  -> 181.18.90.9      UDP
0000:  005056A9 D6720050 56A91AAA 81000DBE .PV..r.PV.....
```

```

0010: 080045C0 00340990 0000FF11 9339B512 ..E..4.....9..
0020: 5A01B512 5A09C000 0EC80020 8E8E20C0 Z...Z..... ..
0030: 03180000 10010000 1002000F 4240000F .....B@..

  6   58      0.173987 181.18.90.1 -> 181.18.90.1      UDP
0000: 005056A9 D6720050 56A91AAA 81000DBE .PV..r.PV.....
0010: 080045C0 00280991 0000FF11 934CB512 ..E..(.....L..
0020: 5A01B512 5A01C000 0EC90014 3BDC0000 Z...Z.....;...
0030: 00000000 10010001 C6F6 .....

```

We can confirm that we are receiving both echoes and control packets by counting the matches. **Note:** **uRPF and ICMP redirects must be disabled for echoes to work, for obvious reasons.** The router is smart enough to not send ICMP redirects even if you don't explicit disable it, but uRPF will break this feature.

```

R9#show monitor capture CAP buffer detailed | count 0EC8
Number of lines which match regexp = 64

```

```

R9#show monitor capture CAP buffer detailed | count 0EC9
Number of lines which match regexp = 140

```

We will simulate a failure on the LAN by configuring an inbound ACL on CSR8 to drop BFD echoes. We will debug BFD events on CSR8 to monitor the change. CSR8 shows the echo failure, notifies all of its clients, and then the OSPF neighbors fail. CSR8 has neighbors to CSR9/CSR10, the DR/BDR, so BFD tracks those.

```

! CSR8
ip access-list extended ACL_DENY_BFD_ECHO
deny    udp any any eq 3785
permit ip any any

```

```

interface GigabitEthernet2.518
ip access-group ACL_DENY_BFD_ECHO in

```

```

R8#debug bfd event
BFD-DEBUG Event: V1 FSM ld:4098 handle:2 event:ECHO FAILURE state:UP (0)
BFD-DEBUG EVENT: bfd_session_destroyed, proc:OSPF, handle:2 act
BFD-DEBUG Event: notify client(CEF) IP:181.18.90.9, ld:4098, handle:2,
event:DOWN, cp independent failure (0)
BFD-DEBUG Event: notify client(OSPF) IP:181.18.90.9, ld:4098, handle:2,
event:DOWN, cp independent failure (0)
%OSPF-5-ADJCHG: Process 181, Nbr 181.0.0.9 on GigabitEthernet2.518 from FULL
to DOWN, Neighbor Down: BFD node down

```

```

BFD-DEBUG Event: V1 FSM ld:4100 handle:4 event:RX DOWN state:UP (0)
BFD-DEBUG EVENT: bfd_session_destroyed, proc:OSPF, handle:4 act
BFD-DEBUG Event: notify client(CEF) IP:181.18.90.10, ld:4100, handle:4,
event:DOWN, cp independent failure (0)

```

```
BFD-DEBUG Event: notify_client(OSPF) IP:181.18.90.10, ld:4100, handle:4,
event:DOWN, cp independent failure (0)
%OSPF-5-ADJCHG: Process 181, Nbr 181.0.0.10 on GigabitEthernet2.518 from FULL
to DOWN, Neighbor Down: BFD node down
```

Removing the ACL, we see the sessions come back up. CSR8 can successfully receive BFD packets, and then initializes the BFD neighbor relationship. OSPF forms its neighbor first as it needs to tell BFD that it requests fast detection. When BFD comes up, it notifies OSPF that it can begin using BFD state.

```
! CSR8
%OSPF-5-ADJCHG: Process 181, Nbr 181.0.0.9 on GigabitEthernet2.518 from
LOADING to FULL, Loading Done
BFD-DEBUG EVENT: bfd_session_created, proc:OSPF, idb:GigabitEthernet2.518
handle:2 act
BFD-DEBUG Event: V1 FSM ld:4118 handle:4 event:RX INIT state:DOWN (0)
BFD-DEBUG Event: notify_client(CEF) IP:181.18.90.10, ld:4118, handle:4,
event:UP, (0)
BFD-DEBUG Event: notify_client(OSPF) IP:181.18.90.10, ld:4118, handle:4,
event:UP, (0)
BFD-DEBUG Event: V1 FSM ld:4118 handle:4 event:RX UP state:UP (0)

%OSPF-5-ADJCHG: Process 181, Nbr 181.0.0.10 on GigabitEthernet2.518 from
LOADING to FULL, Loading Done
BFD-DEBUG EVENT: bfd_session_created, proc:OSPF, idb:GigabitEthernet2.518
handle:4 act
BFD-DEBUG Event: V1 FSM ld:4117 handle:2 event:RX INIT state:DOWN (0)
BFD-DEBUG Event: notify_client(CEF) IP:181.18.90.9, ld:4117, handle:2,
event:UP, (0)
BFD-DEBUG Event: notify_client(OSPF) IP:181.18.90.9, ld:4117, handle:2,
event:UP, (0)
BFD-DEBUG Event: V1 FSM ld:4117 handle:2 event:RX UP state:UP (0)
```

A quick repeat of the same test, except filtering only the control traffic, yields similar results. BFD still does the exact same thing, except the failure event is based on not being able to detect a BFD peer at all. Losing echoes was a different failure event, but the result is identical. In most designs, the echo timer is much faster than the control timer, so a loss of control packets would take longer to discover. Upon failure, BFD notifies its clients who immediately tear down their neighbor sessions.

```
! CSR8
ip access-list extended ACL_DENY_BFD_CONTROL
deny  udp any any eq 3784
permit ip any any

interface GigabitEthernet2.518
ip access-group ACL_DENY_BFD_CONTROL in

! CSR8
```



```

BFD-DEBUG Event: V1 FSM ld:4118 handle:4 event:DETECT TIMER EXPIRED state:UP
(0)
BFD-DEBUG EVENT: bfd_session_destroyed, proc:OSPF, handle:4 act
BFD-DEBUG Event: notify client(CEF) IP:181.18.90.10, ld:4118, handle:4,
event:DOWN, (0)
BFD-DEBUG Event: notify client(OSPF) IP:181.18.90.10, ld:4118, handle:4,
event:DOWN, (0)
%OSPF-5-ADJCHG: Process 181, Nbr 181.0.0.10 on GigabitEthernet2.518 from FULL
to DOWN, Neighbor Down: BFD node down

BFD-DEBUG Event: V1 FSM ld:4117 handle:2 event:DETECT TIMER EXPIRED state:UP
(0)
BFD-DEBUG EVENT: bfd_session_destroyed, proc:OSPF, handle:2 act
BFD-DEBUG Event: notify client(CEF) IP:181.18.90.9, ld:4117, handle:2,
event:DOWN, (0)
BFD-DEBUG Event: notify client(OSPF) IP:181.18.90.9, ld:4117, handle:2,
event:DOWN, (0)
%OSPF-5-ADJCHG: Process 181, Nbr 181.0.0.9 on GigabitEthernet2.518 from FULL
to DOWN, Neighbor Down: BFD node down

```

Upon restoration, the process is reversed, and when OSPF forms neighbors, it notifies BFD. When BFD completes its neighbor exchange, it notifies OSPF that it is now operational. This process is identical to the previous example in terms of debug and is not shown again. We quickly verify it using show commands.

```

R8#show bfd neighbors client ospf
IPv4 Sessions

```

NeighAddr	LD/RD	RH/RS	State	Int
181.18.90.9	4117/4109	Up	Up	Gi2.518
181.18.90.10	4118/4115	Up	Up	Gi2.518

Next, we look at BFD with IS-IS. The echo functionality was not explicitly enabled on the IS-IS LAN using the BFD-template construct. BFD makes no assumptions about your configuration when using templates; in this case, echoes are not used, so all of the BFD packets are control packets. If you use the "interval" command in a bfd-template without explicitly enabling "echo", only BFD control packets are affected by the timer specification. BFD still operates and is effective, but the lightweight behavior is reduced. Some high-end platforms can offload the entire BFD process to linecards, but echo offloading works on almost all routers since it is just based on routing. Despite having a DIS, IS-IS still forms neighbors will all nodes on the segment, or at least that is what IS-IS tells BFD to do. I also enable basic authentication, which is self explanatory. BFD dampening is like BGP or event dampening; continuous flaps will cause BFD to penalize an unstable neighbor. We do not analyze BFD dampening in detail since the logic is identical to BGP dampening (discussed in the security section).

```

! CSR1, CSR3, CSR4, CSR10
key chain KC_BFD_AUTH
key 1

```

```

key-string BFD_AUTH

bfd-template single-hop BFD_AUTH_DAMP
interval min-tx 900 min-rx 900 multiplier 3
authentication sha-1 keychain KC_BFD_AUTH
dampening 3 1500 3000 5

interface GigabitEthernet2.513
bfd template BFD_AUTH_DAMP

```

We can verify the “full mesh” of neighbors, as directed by IS-IS, and also verify that echoes are not enabled. Echoes must be enabled bidirectionally in order for BFD to negotiate them. Also notice that since IS-IS works with both IPv4 and IPv6, we can see the IPv6 peers by checking the IS-IS clients. For brevity, I just show CSR1 and CSR3, but the neighbors are fully meshed.

```

R1#show bfd neighbors client isis
IPv4 Sessions

```

NeighAddr	LD/RD	RH/RS	State	Int
181.13.40.3	4099/4105	Up	Up	Gi2.513
181.13.40.4	4097/4105	Up	Up	Gi2.513
181.13.40.10	4101/4104	Up	Up	Gi2.513

```

IPv6 Sessions

```

NeighAddr	LD/RD	RH/RS	State	Int
FE80::3	4100/4106	Up	Up	Gi2.513
FE80::4	4098/4106	Up	Up	Gi2.513
FE80::10	4102/4105	Up	Up	Gi2.513

```

R3#show bfd neighbors client isis
IPv4 Sessions

```

NeighAddr	LD/RD	RH/RS	State	Int
181.13.40.1	4105/4099	Up	Up	Gi2.513
181.13.40.4	4097/4097	Up	Up	Gi2.513
181.13.40.10	4102/4099	Up	Up	Gi2.513

```

IPv6 Sessions

```

NeighAddr	LD/RD	RH/RS	State	Int
FE80::1	4106/4100	Up	Up	Gi2.513
FE80::4	4098/4098	Up	Up	Gi2.513
FE80::10	4104/4100	Up	Up	Gi2.513

```

R3#show bfd neighbors ipv4 181.13.40.10 details | include ^Sess|^Regist
Session state is UP and not using echo function.
Session Host: Software
Registered protocols: ISIS CEF

```

```

R3#show bfd neighbors ipv6 fe80::10 details | include ^Sess|^Regist
Session state is UP and not using echo function.

```

Session Host: Software

Registered protocols: ISIS CEF

We can verify BFD is enabled by checking the IS-IS process as well. The commands clearly show support for both IPv4 and IPv6 address families.

```
R10#show clns is-neighbors detail | include ^R[0-9]|BFD
```

```
R1          Gi2.513      Up      L2    5          R1.01          Phase V
  BFD enabled: (MTID:0, ipv4) (MTID:2, ipv6)
R3          Gi2.513      Up      L2    0          R1.01          Phase V
  BFD enabled: (MTID:0, ipv4) (MTID:2, ipv6)
R4          Gi2.513      Up      L2    0          R1.01          Phase V
  BFD enabled: (MTID:0, ipv4) (MTID:2, ipv6)
```

```
R10#show isis neighbors detail | include ^R[0-9]|BFD
```

```
R1          L2    Gi2.513      181.13.40.1      UP      9          R1.01
  Remote BFD Support:TLV (MTID:0, IPV4) (MTID:2, IPV6)
  BFD enabled: (MTID:0, ipv4) (MTID:2, ipv6)
R3          L2    Gi2.513      181.13.40.3      UP      24         R1.01
  Remote BFD Support:TLV (MTID:0, IPV4) (MTID:2, IPV6)
  BFD enabled: (MTID:0, ipv4) (MTID:2, ipv6)
R4          L2    Gi2.513      181.13.40.4      UP      26         R1.01
  Remote BFD Support:TLV (MTID:0, IPV4) (MTID:2, IPV6)
  BFD enabled: (MTID:0, ipv4) (MTID:2, ipv6)
```

```
R10#show clns interface gig2.513 | include BFD
```

```
  BFD enabled: (MTID:0, ipv4) (MTID:2, ipv6)
```

A quick look at CSR3's capture shows only packets with varying source/destination addresses. If there were many echoes, we would see packets destined to themselves at layer 3 arriving on CSR3's interface from the other 3 routers on the segment. We see no such packets, which is a good indication that BFD echoes are not enabled on this LAN segment.

```
R3#show monitor capture CAP buffer brief
```

#	size	timestamp	source		destination	protocol
0	98	0.000000	181.13.40.1	->	181.13.40.10	UDP
1	98	0.009002	181.13.40.10	->	181.13.40.1	UDP
2	98	0.020995	181.13.40.4	->	181.13.40.10	UDP
3	98	0.033995	181.13.40.1	->	181.13.40.10	UDP
4	98	0.041990	181.13.40.4	->	181.13.40.10	UDP
5	98	0.042997	181.13.40.1	->	181.13.40.3	UDP
6	98	0.050001	181.13.40.4	->	181.13.40.1	UDP
7	98	0.057996	181.13.40.10	->	181.13.40.1	UDP
8	98	0.069989	181.13.40.3	->	181.13.40.10	UDP
9	98	0.077999	181.13.40.3	->	181.13.40.1	UDP
10	98	0.085994	181.13.40.10	->	181.13.40.1	UDP

11	98	0.100993	181.13.40.1	->	181.13.40.10	UDP
12	98	0.108988	181.13.40.3	->	181.13.40.10	UDP
13	98	0.116998	181.13.40.10	->	181.13.40.1	UDP
14	98	0.124994	181.13.40.3	->	181.13.40.1	UDP

We can also use output filtering to help prove the point. Searching for the hexadecimal control-port in the packet details, we have many hits. Searching for the echo port yields zero hits.

```
R3#show monitor capture CAP buffer detail | count 0EC8
Number of lines which match regexp = 924
```

```
R3#show monitor capture CAP buffer detail | count 0EC9
Number of lines which match regexp = 0
```

A quick detailed look at one of the packets clearly shows it is a control packet. This particular packet was sent from CSR1 to CSR3.

```
R3#show interfaces gigabitEthernet 2 | include bia
Hardware is CSR vNIC, address is 0050.56a9.8ccf (bia 0050.56a9.8ccf)
```

```
R3#monitor capture CAP match ipv4 protocol udp any any range 3784 3785
```

```
R3#show monitor capture CAP buffer detail
79 98 0.900955 181.13.40.1 -> 181.13.40.3 UDP
0000: 005056A9 8CCF0050 56A91AAA 81000DB9 .PV....PV.....
0010: 080045C0 00501F7A 0000FF11 E143B50D ..E..P.z.....C..
0020: 2801B50D 2803C000 0EC8003C 2A4520C4 (... (...<*E .
0030: 03340000 10030000 1001000D BBA0000D .4.....
```

We will correct this behavior using HSRP and PIM next. Enabling "echo" in the template on CSR3 and CSR4 for the VPN session sends echoes. CSR2 also has it enabled. Adjusting the "slow-timer" to 5 seconds, we can slow down the rate at which we send control messages. The timers are shown within the details and are different than the echo timer. The default slow-timer for control messages is 1 second, which is close to our BFD echo timer of 900 ms. We also configure basic HSRP for IPv4 and IPv6 within this VPN so that HSRP can be bound to BFD. HSRP can re-use the BFD key-chain, as can EIGRP. As a final touch, PIM is enabled also, just to demonstrate that it too can register to BFD.

```
! CSR3, CSR4
key chain KC_GEN_AUTH
key 1
key-string GEN_AUTH

bfd slow-timers 5000

bfd-template single-hop BFD_ECHO_VPN
interval min-tx 900 min-rx 900 multiplier 3
```

```

authentication md5 keychain KC_GEN_AUTH
echo

ip multicast-routing vrf L3 distributed
ipv6 multicast-routing vrf L3
standby bfd all-interfaces

interface GigabitEthernet2.523 (or 524)
 ip pim sparse-mode
 ip pim bfd
 standby version 2
 standby 234 ip 10.23.4.254
 standby 234 preempt
 standby 234 authentication md5 key-chain KC_GEN_AUTH
 standby 236 ipv6 FE80::254
 standby 236 priority 105
 standby 236 preempt
 standby 236 authentication md5 key-chain KC_GEN_AUTH
 ipv6 pim bfd
 bfd template BFD_ECHO_VPN

! CSR2
bfd-template single-hop BFD_ECHO_VPN
 interval min-tx 900 min-rx 900 multiplier 3
 authentication md5 keychain KC_GEN_AUTH
 echo

interface GigabitEthernet2.523
 bfd template BFD_ECHO_VPN

```

We quickly verify that CSR3 and CSR4 see two peers each, one another and CSR2. We also verify echoes have been negotiated as expected with a 5 second control-message (slow-timer) interval. We can see that EIGRP, HSRP, and PIM are bound to BFD for both IPv4 and IPv6 successfully. Again, we see that the echo function does not appear to be supported for IPv6 LL addressing. Because of this, the TX/RX intervals are 900ms, not 5000ms, since the control messages are actually used for dead peer detection.

The slow-timer only applies to control messages when echoes are negotiated.

```

R3#show bfd neighbors vrf L3 client pim ipv4 details | include
^Sess|^Regist|xInt
Session state is UP and using echo function with 900 ms interval.
Session Host: Software
MinTxInt: 5000000, MinRxInt: 5000000, Multiplier: 3
Received MinRxInt: 5000000, Received Multiplier: 3
Registered protocols: EIGRP CEF HSRP PIM

```

```

R3#show bfd neighbors vrf L3 client pim ipv6 details | include
^Sess|^Regist|xInt
Session state is UP and not using echo function.

```

Session Host: Software

MinTxInt: 900000, MinRxInt: 900000, Multiplier: 3

Received MinRxInt: 900000, Received Multiplier: 3

Registered protocols: EIGRP CEF HSRP PIM

For completeness, below are several show commands to verify that BFD is enabled for EIGRP, PIM, and HSRP, both IPv4 and IPv6.

R3#show eigrp address-family ipv4 vrf L3 neighbors detail | begin BFD

BFD sessions

NeighAddr	Interface
10.23.4.4	GigabitEthernet2.523
10.23.4.2	GigabitEthernet2.523

R3#show eigrp address-family ipv6 vrf L3 neighbors detail | begin BFD

BFD sessions

NeighAddr	Interface
FE80::4	GigabitEthernet2.523
FE80::2	GigabitEthernet2.523

R3#show ip pim vrf L3 interface detail | include BFD

PIM BFD: enabled

R3#show ip pim vrf L3 interface bfd state-on

Address	Interface	Ver/	Nbr	Query	BFD	DR
DR		Mode	Count	Intvl		Prior
10.23.4.3	GigabitEthernet2.523	v2/S	1	30	On	1
10.23.4.4						

R3#show ipv6 pim vrf L3 interface bfd state-on

Interface	PIM	Nbr	Hello	DR	BFD
		Count	Intvl	Prior	
Gi2.523	on	1	30	1	on
Address: FE80::3					
DR	:	FE80::4			

R3#show standby neighbors

HSRP neighbors on GigabitEthernet2.523

10.23.4.4

No active groups

Standby groups: 234

BFD enabled

FE80::4

Active groups: 236

No standby groups

BFD enabled

Capturing for about 10 seconds on CSR3's VPN interface, we can capture the control messages. Since we know echoes have the same IPv4 source/destination, we can assume that traffic going to CSR3 from CSR4 (or CSR2) is not an echo. The output below shows the control packets from CSR4 to CSR3 over that period. The time between them is about 4 seconds, which is close enough to the timer we configured.

```
R3#show monitor capture CAP buffer brief | include 10.23.4.4.*10.23.4.3
 83   94   1.689011  10.23.4.4    ->  10.23.4.3    UDP
302   94   5.766018  10.23.4.4    ->  10.23.4.3    UDP
543   94   9.755017  10.23.4.4    ->  10.23.4.3    UDP
```

The vast majority of packets are echoes. The promiscuous ESXi port-groups show all of the echoes on the LAN, but the point is that we have reduced the control traffic while still keeping performance good and convergence fast.

```
R3#show monitor capture CAP buffer brief
-----
#    size  timestamp      source          destination    protocol
-----
 0    58    0.000000    10.23.4.2      -> 10.23.4.2      UDP
 1    58    0.055997    10.23.4.2      -> 10.23.4.2      UDP
 2    58    0.057004    10.23.4.2      -> 10.23.4.2      UDP
 3    94    0.061001    10.23.4.3      -> 10.23.4.2      UDP
 4    94    0.076000    10.23.4.4      -> 10.23.4.2      UDP
 5    58    0.091990    10.23.4.2      -> 10.23.4.2      UDP
 6    58    0.119989    10.23.4.2      -> 10.23.4.2      UDP
 7    58    0.131997    10.23.4.2      -> 10.23.4.2      UDP
 8    58    0.152992    10.23.4.2      -> 10.23.4.2      UDP
 9    58    0.165000    10.23.4.2      -> 10.23.4.2      UDP
10   94    0.178000    10.23.4.4      -> 10.23.4.2      UDP
11   58    0.192998    10.23.4.2      -> 10.23.4.2      UDP
12   58    0.204991    10.23.4.2      -> 10.23.4.2      UDP
13   58    0.209996    10.23.4.2      -> 10.23.4.2      UDP
```

Now that we understand echoes and control packets, we will examine the actual timer values between CSR5 and CSR10 before continuing. Many others have suggested that the naming of these timers is difficult to interpret, and it is. The number following the “bfd interval” command is the “min_tx” value, followed by “min_rx”. When exchanging control packets, the potential BFD peers exchange these values (just like HSRP). The local min_tx is compared against the received min_rx from the peer. The larger (slower) value wins and is used as the hello interval from the local router. This is a unidirectional timer and may vary in both directions. Below, CSR5 has its min_tx timer set to 910 ms and its min_rx timer set to 890 ms. CSR10 uses values 900 and 920 for min_tx and min_rx, respectively.

```
! CSR5
interface GigabitEthernet2.550
  bfd interval 910 min_rx 890 multiplier 3
```

```
! CSR10
interface GigabitEthernet2.550
  bfd interval 900 min_rx 920 multiplier 3
```

From CSR5's perspective, its local min_tx (910) is compared to CSR10's min_rx (920) to determine how quickly it should send hellos. Since CSR9 is not willing to accept hellos any faster than 920ms, CSR5 will use a value of 920. Note that the "TxInt" and "RxInt" timers in the output below do not represent the echo timers; these are for the control packets discussed earlier.

```
R5#show bfd neighbors ipv4 181.5.10.10 details | include echo|xInt
Session state is UP and using echo function with 920 ms interval.
MinTxInt: 1000000, MinRxInt: 1000000, Multiplier: 3
Received MinRxInt: 1000000, Received Multiplier: 3
```

In the opposite direction, CSR9 compares its local min_tx (900) to CSR5's min_rx (890). Since CSR9 cannot send any faster than 900ms, this value is used.

```
R10#show bfd neighbors ipv4 181.5.10.5 details | include echo|xInt
Session state is UP and using echo function with 900 ms interval.
MinTxInt: 1000000, MinRxInt: 1000000, Multiplier: 3
Received MinRxInt: 1000000, Received Multiplier: 3
```

We can manually disable echoes on this link between CSR5 and CSR10. To be a minimalist, I will only disable echoes on CSR5, which has the added benefit of disabling echoes for anything trying to establish BFD sessions with CSR5. We also look at the "Hello" value since this is the result of the local min_tx versus remote min_rx comparison. CSR5 selects 920 ms, which is CSR10's min_rx, since it is greater than its local min_tx. CSR10 selects 900 ms, which is its local min_tx, since it is greater than CSR5's remote min_rx. Because echoes were disabled, the "TxInt" and "RxInt" timers now reflect the configured intervals. As mentioned earlier, using control packets for dead peer detection generally undesirable as echoes are lighter weight.

```
! CSR5
interface GigabitEthernet2.550
  no bfd echo
```

```
R5#show bfd neighbors ipv4 181.5.10.10 details | include echo|xInt|Hello
Session state is UP and not using echo function.
MinTxInt: 910000, MinRxInt: 890000, Multiplier: 3
Received MinRxInt: 920000, Received Multiplier: 3
Holddown (hits): 2061(0), Hello (hits): 920(6600)
```

```
R10#show bfd neighbors ipv4 181.5.10.5 details | include echo|xInt|Hello
Session state is UP and not using echo function.
MinTxInt: 900000, MinRxInt: 920000, Multiplier: 3
Received MinRxInt: 890000, Received Multiplier: 3
Holddown (hits): 2196(0), Hello (hits): 900(6623)
```


Next, we will examine BFD multi-hop sessions, which carry some additional restrictions. RFC 5883 states that the echo function must not be used over multi-hop BFD sessions. This is because the intermediate hops would route the packets directly back to the source, so only control packets exist using port 4784 (0x12B0) are shown below. You cannot configure BFD echoes on a template applied to a multi-hop session. Below is a capture inbound on CSR9 to see the BFD packets from CSR5. I would assume this means that BFD for pseudowires, when used in this way, is generally undesirable unless the platform can hardware-offload the packets. BFD can run inside VCCV in some versions/platforms, but not the CSR1000v. The BFD-map command assigns a template to a specific source/destination pair in a specified VRF (as needed). When a BFD session is established that matches a BFD-map, the longest-match rules apply, and a template is used to build the BFD session. I use interesting masks in this example just to illustrate the point, but using /32s for explicit multi-hop links is the most straightforward. The significant benefit of multi-hop sessions is that it allows BFD to track layer 3 paths, not just layer 2 paths.

```
! Both CSR5 and CSR9
bfd-template multi-hop BFD_L2VPN
  interval min-tx 900 min-rx 900 multiplier 3

template type pseudowire TMP_VPLS
  encapsulation mpls
  control-word include
  signaling protocol ldp

l2vpn vfi context VPLS
  autodiscovery bgp signaling ldp template TMP_VPLS

! CSR5 only
bfd map ipv4 181.0.0.0/28 181.0.0.0/29 BFD_L2VPN

template type pseudowire TMP_VPLS
  monitor peer bfd local interface Loopback0

! CSR9 only
bfd map ipv4 181.0.0.5/32 181.0.0.9/32 BFD_L2VPN

template type pseudowire TMP_VPLS
  monitor peer bfd
```

The verification process for this is two steps. First, we need to verify the BFD neighbor is up and that xconnect is registered to it. Then, we will verify that the PW details include BFD state information.

```
R9#show bfd neighbors multihop-ipv4 181.0.0.5 details | include ^Sess|^Regist
Session state is UP and not using echo function.
Session Host: Software
Registered protocols: Xconnect
```

```
R9#show l2vpn atom vc vcid 1112 detail | include BFD
Fast failure detection (BFD): up
  BFD dataplane status received      : Not sent
  BFD peer monitor status received   : No fault
```

We can check each BFD-map for a list of BFD neighbors that are using the map attributes. Show/debug commands indicate that the “L” and “R” stand for local and remote, respectively, and the “D” stands for “discriminator”. The numbers are reversed on either side of the BFD peering and the BFD-map references the LD when describing the sessions that inherit from it.

```
R9#show bfd map internal
Destination[vrf]: 181.0.0.5/32
Source[vrf]: 181.0.0.9/32
Template: BFD_L2VPN
Authentication(Type/Keychain): invalid_type/
Adjacencies associated to map:
  LD: 4111, Ref: 0x7F8A337A9600
```

```
R9#show bfd neighbors multihop-ipv4
IPv4 Multihop Sessions
NeighAddr[vrf]          LD/RD          RH/RS          State
181.0.0.5              4111/4100        Up             Up
```

We can verify this is true by checking the other side as well. We see that 4111 is CSR9’s LD which maps to CSR5’s RD. The opposite is true for the value 4100 (LD of CSR5, RD of CSR9).

```
R5#show bfd map internal
Destination[vrf]: 181.0.0.0/28
Source[vrf]: 181.0.0.0/29
Template: BFD_L2VPN
Authentication(Type/Keychain): invalid_type/
Adjacencies associated to map:
  LD: 4100, Ref: 0x7FB78F42DE10
```

```
R5#show bfd neighbors multihop-ipv4
IPv4 Multihop Sessions
NeighAddr[vrf]          LD/RD          RH/RS          State
181.0.0.9              4100/4111        Up             Up
```

The BFD session for this PW is just a normal IP/UDP flow between two endpoints. The end router sees it as IP, not MPLS, simply because the penultimate hop performs PHP. This is different than the VCCV-based BFD (not supported on CSR1000v) which would be carried in the PW-ACH, implying MPLS encapsulation from end-to-end.

```
R9#monitor capture CAP match ipv4 protocol udp 181.0.0.5/32 181.0.0.9/32 eq
4784
```

```

R9#show monitor capture CAP buffer detailed
 14   70   0.394029  181.0.0.5    ->  181.0.0.9    UDP
0000:  005056A9 D6720050 56A9F961 81000DBE  .PV..r.PV..a....
0010:  080045C0 00340C51 0000FE11 4599B500  ..E...4.Q....E...
0020:  0005B500 0009C000 12B00020 07B420C0  .....
0030:  03180000 10020000 1005000D BBA0000D  .....

```

We can simulate a failure by blocking these multi-hop BFD packets in the transit path. I will do it on CSR9 once all the MPLS transport labels are removed, revealing the IP packet.

```

! CSR9
ip access-list extended ACL_DENY_MHOP_BFD
deny  udp any any eq 4784
permit ip any any

interface GigabitEthernet2.518
ip access-group ACL_DENY_MHOP_BFD in

```

Debugging AToM status events on CSR5, we can see that there is a fault with sending messages to CSR9. CSR5 generates a status message to show this, as does CSR9. Both routers update their PW status to down due to a fast failure detection (FFD) error, a sub-process within the PW process. The Cisco documentation does not detail what the “A” means in the state machine, but I assume it is some kind of administrative hold-down on the PW due to a sub-process like FFD dictating the PW remain down. It could also mean “active” since BFD keeps trying to re-establish the session so long as it has a client.

```

R5#debug l2vpn atom vc status event
AToM[181.0.0.9, 1112]:  FFD          : DOWN(pw-tx-fault)
AToM[181.0.0.9, 1112]:  .. FFD          : DOWN(pw-tx-fault)
AToM[181.0.0.9, 1112]:  .. Overall      : DOWN(pw-rx-fault)
AToM[181.0.0.9, 1112]:  .. Update AC for status change from UP
AToM[181.0.0.9, 1112]:  .. Send AC STATUS (DOWN(pw-rx-fault))
AToM[181.0.0.9, 1112]:  .. Send notify(DOWN)
AToM[181.0.0.9, 1112]:  .. FFD          : DOWN(pw-tx-fault)
AToM[181.0.0.9, 1112]:  .. Overall      : DOWN(pw-tx-fault)
[snip]
%XCONNECT-5-PW_STATUS: MPLS peer 181.0.0.9 vcid 1112, VC state DOWN, PW Err

```

```

R5#show l2vpn atom vc vcid 1112 detail | include BFD
Fast failure detection (BFD): down
BFD dataplane status received          : Not sent
BFD peer monitor status received        : DOWN(pw-tx-fault)

```

```

R5#show l2vpn atom vc vcid 1112 detail | include state_machine
Label/status state machine              : established, LruRrA

```

We see similar output on CSR9. Both routers think there is a TX problem with the remote end since the issue is in the transit path, not related to a local AC.

```

R9#show l2vpn atom vc vcid 1112 detail | include BFD
Fast failure detection (BFD): down
  BFD dataplane status received      : Not sent
  BFD peer monitor status received   : DOWN(pw-tx-fault)

R9#show l2vpn atom vc vcid 1112 detail | include state_machine
Label/status state machine          : established, LruRrA

```

Before continuing, we will remove the ACL from CSR9. With debugging still enabled on CSR5, we can see the FFD process shows no fault and brings the PW back online.

```

! CSR5
AToM[181.0.0.9, 1112]: FFD      : No fault
AToM[181.0.0.9, 1112]: .. Remote   : DOWN(pw-tx-fault)
AToM[181.0.0.9, 1112]: .. Overall  : DOWN(pw-tx-fault)
AToM[181.0.0.9, 1112]: .. Update AC for status change from DOWN PW(rx/tx
faults)
AToM[181.0.0.9, 1112]: .. Send AC STATUS(DOWN(pw-tx-fault))
AToM[181.0.0.9, 1112]: .. Send notify(UP)
[snip]
%XCONNECT-5-PW_STATUS: MPLS peer 181.0.0.9 vcid 1112, VC state UP

```

```

R5#show l2vpn atom vc vcid 1112 detail | include BFD
Fast failure detection (BFD): up
  BFD dataplane status received      : Not sent
  BFD peer monitor status received   : No fault

R5#show l2vpn atom vc vcid 1112 detail | include state_machine
Label/status state machine          : established, LruRru

```

BGP can also use BFD in single or multi-hop fashion. Multi-hop BFD does not appear to work with IPv6 though, so we will use multi-hop for IPv4 and single-hop for IPv6 between CSR7 (CE) and CSR8 (PE). Because echoes are not supported on any kind of multi-hop connections, I will enable them for the IPv6 single-hop connection. The reason for enabling BGP graceful restart (GR) and the check-control-plane failure (C-bit) are described later.

```

! CSR7 and CSR8
key chain KC_MHOP
  key 1
    key-string MHOP

bfd-template multi-hop BFD_MHOP
  interval min-tx 900 min-rx 900 multiplier 3
  authentication meticulous-sha-1 keychain KC_MHOP

bfd-template single-hop BFD_SHOP

```

```

interval min-tx 900 min-rx 900 multiplier 3
authentication meticulous-sha-1 keychain KC_MHOP
echo

! CSR8 only
bfd map ipv4 vrf L3 7.7.7.7/32 vrf L3 10.7.8.8/32 BFD_MHOP

router bgp 181
  bgp graceful-restart
  address-family ipv4 vrf L3
    neighbor 7.7.7.7 disable-connected-check
    neighbor 7.7.7.7 fall-over bfd multi-hop check-control-plane-failure

address-family ipv6 vrf L3
  neighbor FD00:10:7:8::7 fall-over bfd single-hop

ip route vrf L3 7.7.7.7 255.255.255.255 GigabitEthernet2.578 10.7.8.7

! CSR7 only
bfd map ipv4 10.7.8.8/32 7.7.7.7/32 BFD_MHOP

router bgp 7
  bgp graceful-restart
  neighbor 10.7.8.8 fall-over bfd multi-hop check-control-plane-failure
  neighbor FD00:10:7:8::8 fall-over bfd single-hop

```

We can check to ensure the sessions are up from CSR8 within the VRF. Neither session is using echoes; for the third time, we have seen a lack of echo support for IPv6 despite being configured for it. We can also verify that BGP is registered to BFD properly. The multi-hop session does not register CEF as a client since the lookup is recursive.

```

R8#show bfd neighbors multihop-ipv4 vrf L3 7.7.7.7 details | include
^Sess|^Regist
Session state is UP and not using echo function.
Session Host: Software
Registered protocols: BGP

R8#show bfd neighbors vrf L3 ipv6 FD00:10:7:8::7 details | include
^Sess|^Redist
Session state is UP and not using echo function.
Session Host: Software
Registered protocols: BGP CEF

```

We can quickly check BGP to ensure BFD is configured for this peer. CSR8 has it within VPNv4 (a customer VRF) and CSR7 has it in the global IPv4 unicast AFI. I cannot explain why CSR8 doesn't show that the BFD session is up or why the C-bit check is not displayed, because they are operational. I assume this is a cosmetic decision specific to the VPNv4 show command.

```
R8#show bgp vpnv4 unicast vrf L3 neighbors 7.7.7.7 | include BFD
BFD is configured.
```

```
R7#show bgp ipv4 unicast neighbors 10.7.8.8 | include BFD
BFD is configured. BFD peer is Up. Using BFD to detect fast fallover (multi-hop) with c-bit check-control-plane-failure.
```

For completeness, we can check the LD/RD and BFD map information. CSR7 has an LD of 4098 which is used to create a neighbor to CSR8. CSR8 sees that LD as its RD.

```
R7#show bfd neighbors multihop-ipv4 10.7.8.8
IPv4 Multihop Sessions
NeighAddr[vrf]                LD/RD                RH/RS                State
10.7.8.8                      4098/4099                Up                    Up
```

```
R7#show bfd map internal
Destination[vrf]: 10.7.8.8/32
Source[vrf]: 7.7.7.7/32
Template: BFD_MHOP
Authentication(Type/Keychain): meticulous-sha-1/KC_MHOP
Authentication keys:
  Send - ID: 1, String: MHOP
  Accept - ID: 1, String: MHOP
Adjacencies associated to map:
  LD: 4098, Ref: 0x7F4015FEBAC8
```

```
R8#show bfd neighbors multihop-ipv4 vrf L3 7.7.7.7
IPv4 Multihop Sessions
NeighAddr[vrf]                LD/RD                RH/RS                State
7.7.7.7 [L3]                  4099/4098                Up                    Up
```

With BFD properly configured, we will examine the C-bit and its significance. This is neither straightforward nor well-documented. First, we will ensure the GR capability was exchanged PE-CE. BGP GR is covered in detail in the High Availability (HA) section.

```
R7#show bgp ipv4 unicast neighbors 10.7.8.8 | include Grace
Graceful Restart Capability: advertised and received
Graceful-Restart is enabled, restart-time 120 seconds, stalepath-time 360 seconds
```

```
R8#show bgp vpnv4 unicast vrf L3 neighbors 7.7.7.7 | include Grace
Graceful Restart Capability: advertised and received
Graceful-Restart is enabled, restart-time 120 seconds, stalepath-time 360 seconds
```

The BFD process can set the C-bit, or control-plane bit (not to be confused with the PW C-bit, or control-word bit). The C-bit will be set to 0 when BFD is run in software or 1 when it runs in hardware on a linecard. This allows your BFD peer to determine whether or not a failure of BFD control traffic is really an indication of a node failure. For example, if a node stops receiving BFD control packets and the peer's C-bit is 0, it is safe to assume that BFD was not running on the linecard, and maybe the RSP was restarted. The linecard can continue passing traffic, which is the basis of non-stop forwarding (NSF). BFD does not instruct BGP to purge routes since the data-plane is still operable. If a node stops receiving BFD control packets and the C-bit is 1, this means the entire BFD process was running on that linecard, so chances are the whole linecard is inoperable. BFD will instruct BGP to purge all routes in this case. CSR1000v does not have separate RSPs and linecards (BFD runs in software) so the C-bit should remain zero. By default, BGP totally ignores this bit, and the "check-control-plane-failure" option tells BGP to look at it. This can prevent black holes when the linecard actually does fail at the same time the RSP does. BFD tracks the last control packet's information; we can see the C-bit is zero for reasons already described. Also note "my/your" values the session discriminators seen earlier.

```
R8#show bfd neighbors multihop-ipv4 vrf L3 7.7.7.7 details | begin Last_pack
Last packet: Version: 1                - Diagnostic: 0
          State bit: Up                - Demand bit: 0
          Poll bit: 0                  - Final bit: 0
          C bit: 0
          Multiplier: 3                - Length: 52
          My Discr.: 4098              - Your Discr.: 4099
          Min tx interval: 900000      - Min rx interval: 900000
          Min Echo interval: 0
```

This feature is also supported in IS-IS. I enable it on CSR3 and CSR4, although no show commands seem to exist to verify it. Considering all BFD sessions thus far have been in "software", the C-bit doesn't mean much and will always be zero on the CSR1000v.

```
! CSR3 and CSR4
router isis 181
  bfd check-ctrl-plane-failure
```

Back to BGP, we will test IPv6 between CSR7 and CSR8. We already verified that BFD sees BGP as a client for IPv6. We quickly verify this using BGP show commands on both CSR7 and CSR8.

```
R8#show bgp vpv6 unicast vrf L3 neighbors FD00:10:7:8::7 | include BFD
BFD is configured. BFD peer is Up. Using BFD to detect fast fallover
(single-hop).
```

```
R7#show bgp ipv6 unicast neighbors FD00:10:7:8::8 | include BFD
BFD is configured. BFD peer is Up. Using BFD to detect fast fallover
(single-hop).
```

We will use an IPv6 ACL to drop single-hop BFD control packets for IPv6, which should break the eBGP connection to CSR8. We also enable BFD event and BGP IPv6 unicast debugging. When CSR7 cannot receive control packets anymore, it cannot detect CSR8, and tears down the BFD session. It notifies all clients (BGP and CEF in this case) that the session is down.

```
! CSR7
ipv6 access-list ACL_DENY_BFD_CONTROL_V6
  deny udp any any eq 3784
  permit ipv6 any any

interface GigabitEthernet2.578
  ipv6 traffic-filter ACL_DENY_BFD_CONTROL_V6 in

R7#debug bgp ipv6 unicast
R7#debug bfd event
BFD-DEBUG Event: V1 FSM ld:4097 handle:2 event:DETECT TIMER EXPIRED state:UP
(0)
BGP-BFD: IPv6 Single hop Neighbor FD00:10:7:8::8 DOWN
BGP: FD00:10:7:8::8 Received event from BFD, event: Down, admin_down: FALSE
BGP: BFD DOWN event notification for nbr: FD00:10:7:8::8
BFD-DEBUG Event: resetting timestamps ld:4097 handle:2 (0)
BFD-DEBUG Event: notify client(CEF) IP:FD00:10:7:8::8, ld:4097, handle:2,
event:DOWN, (0)
BFD-DEBUG Event: notify client(BGP) IP:FD00:10:7:8::8, ld:4097, handle:2,
event:DOWN, (0)
BGP: BFD adjacency for FD00:10:7:8::8 down
%BGP-5-NBR_RESET: Neighbor FD00:10:7:8::8 reset (BFD adjacency down)
BGP: ses global FD00:10:7:8::8 (0x7F4015FEC5D0:1) Reset (BFD adjacency down).
BGP: nbr_topo global FD00:10:7:8::8 IPv6 Unicast:base (0x7F4015FEC5D0:1) NSF
delete stale NSF not active
BGP: nbr_topo global FD00:10:7:8::8 IPv6 Unicast:base (0x7F4015FEC5D0:1) NSF
no stale paths state is NSF not active
BGP: nbr_topo global FD00:10:7:8::8 IPv6 Unicast:base (0x7F4015FEC5D0:1)
Resetting ALL counters.
BGP: FD00:10:7:8::8 closing
BGP: ses global FD00:10:7:8::8 (0x7F4015FEC5D0:1) Session close and reset
neighbor FD00:10:7:8::8 topostate
BGP: nbr_topo global FD00:10:7:8::8 IPv6 Unicast:base (0x7F4015FEC5D0:1)
Resetting ALL counters.
BGP: FD00:10:7:8::8 went from Established to Idle
%BGP-5-ADJCHANGE: neighbor FD00:10:7:8::8 Down BFD adjacency down
%BGP_SESSION-5-ADJCHANGE: neighbor FD00:10:7:8::8 IPv6 Unicast topology base
removed from session BFD adjacency down
BGP: ses global FD00:10:7:8::8 (0x7F4015FEC5D0:1) Removed topology IPv6
Unicast:base
BGP: ses global FD00:10:7:8::8 (0x7F4015FEC5D0:1) Removed last topology
BGP: nbr global FD00:10:7:8::8 Open active delayed 12288ms (35000ms max, 60%
jitter)
```



```
BFD-DEBUG EVENT: bfd_session_destroyed, proc:BGP, handle:2 act
BGP: FD00:10:7:8::8 unregistered with BFD
BGP: nbr global FD00:10:7:8::8 Active open failed - open timer running
```

When we bring the session back up (removal of ACL not shown for brevity), the BGP session notifies BFD and re-registers with it.

```
! CSR7
BFD-DEBUG Event: V1 FSM ld:4097 handle:2 event:RX INIT state:DOWN (0)
BFD-DEBUG Event: resetting timestamps ld:4097 handle:2 (0)
BFD-DEBUG Event: notify client(CEF) IP:FD00:10:7:8::8, ld:4097, handle:2,
event:UP, (0)
BFD-DEBUG EVENT: bfd_session_destroyed, proc:CEF, handle:2 act
BFD-DEBUG Event: V1 FSM ld:4097 handle:2 event:Session delete state:UP (0)
BFD-DEBUG Event: resetting timestamps ld:4097 handle:2 (0)
BGP: FD00:10:7:8::8 active went from Idle to Active
BGP: FD00:10:7:8::8 open active, local address FD00:10:7:8::7
[snip, BGP capability exchange, etc]
BGP: FD00:10:7:8::8 active went from OpenSent to OpenConfirm
BGP: FD00:10:7:8::8 active went from OpenConfirm to Established
BGP: ses global FD00:10:7:8::8 (0x7F401407F248:1) act Assigned ID
BGP-BFD: FD00:10:7:8::8 Trying bfd peer force single-hop session on
idb:GigabitEthernet2.578
BGP-BFD: FD00:10:7:8::8 Forcing single-hop peer session on
idb:GigabitEthernet2.578
BGP-BFD: FD00:10:7:8::8 doing single-hop peer session on
idb:GigabitEthernet2.578
BFD-DEBUG EVENT: bfd_session_created, proc:BGP, idb:GigabitEthernet2.578
handle:2 act
BGP: FD00:10:7:8::8 registered 1 with BFD session on interface:
GigabitEthernet2.578 and address type: 20 and session_handle: 2
BGP: FD00:10:7:8::8 Singlehop ipv6 peer session : PASSED
BGP: ses global FD00:10:7:8::8 (0x7F401407F248:1) Up
%BGP-5-ADJCHANGE: neighbor FD00:10:7:8::8 Up
BGP: FD00:10:7:8::8 Received event from BFD, event: Up, admin_down: TRUE
BGP: BFD UP event notification for nbr: FD00:10:7:8::8
BFD-DEBUG Event: resetting timestamps ld:4097 handle:2 (0)
BFD-DEBUG Event: notify client(CEF) IP:FD00:10:7:8::8, ld:4097, handle:2,
event:UP, (0)
BFD-DEBUG Event: notify client(BGP) IP:FD00:10:7:8::8, ld:4097, handle:2,
event:UP, (0)
BFD-DEBUG Event: V1 FSM ld:4097 handle:2 event:RX UP state:UP (0)
```

BGP keeps track of these failures on a per-neighbor basis. We can see both CSR7 and CSR8 claim that BFD was the reason for the last peer failure.

```
R8#show bgp vpnv6 unicast vrf L3 neighbors FD00:10:7:8::7 | include BFD
```

BFD is configured. BFD peer is Up. Using BFD to detect fast fallover (single-hop).

Last reset 00:11:13, due to BFD adjacency down of session 1

```
R7#show bgp ipv6 unicast neighbors FD00:10:7:8::8 | include BFD
```

BFD is configured. BFD peer is Up. Using BFD to detect fast fallover (single-hop).

Last reset 00:11:43, due to BFD adjacency down of session 1

BFD can also support static routes. The sessions cannot be used on PPPoE (dialer/virtual-template) interfaces but are supported on most other interface types. This is one of the most complicated BFD configurations, especially because the parser is inconsistent. It appears that only multi-hop sessions are available when using this inside VRFs for IPv4, so we will have to create multi-hop sessions from CSR5 to CSR6 despite them being connected. This means we need a multi-hop template and a BFD-map for the endpoints. We can make a very generic BFD map such that the endpoint matching (10.0.0.0/8) identical for both nodes. The only difference will be the presence of a VRF on CSR5. The routes in CSR5 are VRF aware while those in CSR6 are not. CSR5 has two VRF routes to reach remote CE networks and redistributes these into BGP for L3VPN service. BFD tracks the next-hop of those routes. One entry is marked as passive; this does not establish a BFD session, but simply tracks the non-passive one. If 10.5.6.6 becomes unavailable, it is assumed that 10.5.6.66 is also unavailable (in this case, a secondary address). Both the 10.6.6.6/32 and 10.66.66.66/32 routes can be withdrawn without having to use a different BFD peer per next-hop or per static route. CSR6 has a simplified configuration with just a default route to the next-hop, which is BFD tracked. **Pitfall: When using multi-hop static route sessions, you must never include the outgoing interface in the static route, only the IP addresses. For single-hop static route sessions, you must always include the outgoing interface in the static route.**

```
! CSR5
bfd-template multi-hop BFD_STATIC
  interval min-tx 900 min-rx 900 multiplier 3
  echo

bfd map ipv4 vrf L3 10.0.0.0/8 vrf L3 10.0.0.0/8 BFD_STATIC

ip route static bfd vrf L3 10.5.6.6 vrf L3 10.5.6.5 group L3_GROUP log
ip route static bfd vrf L3 10.5.6.66 vrf L3 10.5.6.5 group L3_GROUP passive
ip route vrf L3 10.6.6.6 255.255.255.255 10.5.6.6
ip route vrf L3 10.66.66.66 255.255.255.255 10.5.6.66

! CSR6
bfd-template multi-hop BFD_STATIC
  interval min-tx 900 min-rx 900 multiplier 3
  echo

bfd map ipv4 10.0.0.0/8 10.0.0.0/8 BFD_STATIC

ip route static bfd 10.5.6.5 10.5.6.6 log
```

```
ip route 0.0.0.0 0.0.0.0 10.5.6.5
```

The “log” keyword will generate log messages when the reachability status changes. This is useful for static routing since there is no IGP/BGP to generate logs for us. Flapping CSR6’s interface shows the down and up log messages. Despite configuring echo, the feature cannot work since technically this is a multi-hop session. There is no good reason why single-hop functionality is not available in a VRF (but is in the global table); this is the only reason we used a multi-hop session.

```
! CSR6
%IPRT-5-STATICROUTES_BFD_EVENT: BFD session Down, [Destination
Addr:10.5.6.5,Source Addr:10.5.6.6]
%IPRT-5-STATICROUTES_BFD_EVENT: BFD session Up, [Destination
Addr:10.5.6.5,Source Addr:10.5.6.6]
```

```
R6#show bfd neighbors client static-v4 details | include ^Sess|^Regist
Session state is UP and not using echo function.
Session Host: Software
Registered protocols: IPv4 Static
```

There is an entirely new set of show commands we can use to verify this as well. Since the ordinary “show ip route” command doesn’t reveal anything new, there has to be a way to verify BFD from the perspective of the supported client protocol. The commands below reveal more detail about the origin of a static route. They can end up in the RIB for a number of reasons: AAA, DHCP, PFR (OER), PPP, MPLS-TE (auto-route destination), etc. It is nice to see this kind of detail when troubleshooting “phantom” static routes. In this case, they are manually configured and tracked by BFD.

```
R5#show ip static route vrf L3
Codes: M - Manual static, A - AAA download, N - IP NAT, D - DHCP,
       G - GPRS, V - Crypto VPN, C - CASA, P - Channel interface processor,
       B - BootP, S - Service selection gateway
       DN - Default Network, T - Tracking object
       L - TL1, E - OER, I - iEdge
       D1 - Dot1x Vlan Network, K - MWAM Route
       PP - PPP default route, MR - MRIPv6, SS - SSLVPN
       H - IPe Host, ID - IPe Domain Broadcast
       U - User GPRS, TE - MPLS Traffic-eng, LI - LIIN
       IR - ICMP Redirect
Codes in [: A - active, N - non-active, B - BFD-tracked, D - Not Tracked, P
- permanent
Static local RIB for L3
M 10.6.6.6/32 [1/0] via 10.5.6.6 [AB]
M 10.66.66.66/32 [1/0] via 10.5.6.66 [AB]
```

```
R5#show ip static route bfd
Codes in [: R - Reachable, U - Unreachable, L - Loop, D - Not Tracked
IPv4 Static BFD Multihop Sessions:
Destination [vrf] : 10.5.6.6 [L3]
```

```
Source [vrf]      : 10.5.6.5      [L3]
Status           : [R]
Group Info       : L3_GROUP, Active
```

```
Destination [vrf] : 10.5.6.66     [L3]
Source [vrf]      : 10.5.6.5      [L3]
Status           : [R]
Group Info       : L3_GROUP, Passive
```

We can verify this on CSR6 also. The output is basically the same, except there is only one default route and no assigned association group.

```
R6#show ip static route | begin Static
Static local RIB for default
M 0.0.0.0/0 [1/0] via 10.5.6.5 [AB]
```

```
R6#show ip static route bfd 10.5.6.5
Codes in []: R - Reachable, U - Unreachable, L - Loop, D - Not Tracked
IPv4 Static BFD Multihop Sessions:
Destination [vrf] : 10.5.6.5
Source [vrf]      : 10.5.6.6
Status           : [R]
Group Info       :
```

From CSR5, we currently have both routes in the VRF table, meaning they are “active”. By creating an ACL on CSR6 to block the BFD control messages (echoes are not in use), this will break the BFD session and should withdraw both routes. Even though there isn’t a BFD session to 10.5.6.66, it is associated to a route that does have a BFD session and shares its fate.

```
R5#show ip route vrf L3 static | begin Gate
Gateway of last resort is not set
    10.0.0.0/8 is variably subnetted, 5 subnets, 2 masks
S      10.6.6.6/32 [1/0] via 10.5.6.6
S      10.66.66.66/32 [1/0] via 10.5.6.66
```

Debugging BFD and VRF-aware RIB changes, we apply the ACL to CSR6 to drop the BFD multi-hop control packets. BFD detects the failure and notifies “IPv4 Static” that the session failed. BFD generates a syslog message, and the RIB removes both 10.6.6.6/32 and 10.66.66.66/32 routes as a result of the failure.

```
! CSR6
ip access-list extended ACL_DENY_BFD_CONTROL
deny   udp any any eq 4784
permit ip any any

interface GigabitEthernet2.556
ip access-group ACL_DENY_BFD_CONTROL in
```

```

R5#debug bfd event
R5#debug ip routing vrf L3
BFD-DEBUG Event: V1 FSM ld:4102 handle:4 event:RX DOWN state:UP (0)
BFD-DEBUG Event: resetting timestamps ld:4102 handle:4 (0)
BFD-DEBUG Event: notify client(IPv4 Static) dst IP:10.5.6.6 src IP:10.5.6.5
ld:4102 handle:4, event:DOWN, cp independen (0)
%IPRT-5-STATICROUTES_BFD_EVENT: BFD session Down, [Destination
Addr:10.5.6.6, Source Addr:10.5.6.5]
RT(L3): del 10.6.6.6 via 10.5.6.6, static metric [1/0]
RT(L3): delete subnet route to 10.6.6.6/32
RT(L3): del 10.66.66.66 via 10.5.6.66, static metric [1/0]
RT(L3): delete subnet route to 10.66.66.66/32
BFD-DEBUG Event: V1 FSM ld:4102 handle:4 event:RX DOWN state:DOWN (0)

```

With the routes withdrawn, we can see the “A” for active turns into an “N” for non-active. These don’t show up in the RIB, but rather than search for them in the configuration, we can see them here at all times. Using IP SLA to track static routes would be comparable to using a protocol-specific hello timer; BFD is the newer and better option when static routes are configured bidirectionally.

```

R5#show ip static route vrf L3 | begin Codes_in
Codes in []: A - active, N - non-active, B - BFD-tracked, D - Not Tracked, P
- permanent
Static local RIB for L3
M 10.6.6.6/32 [1/0] via 10.5.6.6 [NB]
M 10.66.66.66/32 [1/0] via 10.5.6.66 [NB]

R5#show ip route vrf L3 static | begin Gateway
Gateway of last resort is not set
[no output]

```

Unlike the IGP, IPv4 Static can’t tell BFD when it comes back (no discovery mechanism), so BFD is in charge of telling IPv4 Static to go up and go down. The process is similar on CSR5 when we remove the ACL from CSR6, the only similarity to the IGP being that the BFD session comes fully “up” once the routes get added to the RIB. BFD went fully down when the routes were removed from the RIB also, seen earlier.

```

! CSR5
BFD-DEBUG Event: V1 FSM ld:4102 handle:4 event:RX UP state:INIT (0)
BFD-DEBUG Event: resetting timestamps ld:4102 handle:4 (0)
BFD-DEBUG Event: notify client(IPv4 Static) dst IP:10.5.6.6 src IP:10.5.6.5
ld:4102 handle:4, event:UP, (0)
%IPRT-5-STATICROUTES_BFD_EVENT: BFD session Up, [Destination
Addr:10.5.6.6, Source Addr:10.5.6.5]
RT(L3): updating static 10.6.6.6/32 (0x1) :
via 10.5.6.6 0 1048578

RT(L3): add 10.6.6.6/32 via 10.5.6.6, static metric [1/0]

```

```
RT(L3): updating static 10.66.66.66/32 (0x1) :
    via 10.5.6.66 0 1048578
RT(L3): add 10.66.66.66/32 via 10.5.6.66, static metric [1/0]
BFD-DEBUG Event: V1 FSM ld:4102 handle:4 event:RX UP state:UP (0)
```

Last, we examine IPv6 static routing with BFD. Interestingly, single-hop seems to be supported for IPv6 as we can specify an interface when selecting the VRF-aware option. Multi-hop is not even an option though, so the feature options between IPv4/v6 static are mutually exclusive (IPv4 multi-hop or IPv6 single-hop). We also don't have the fancy group configuration or logging features available for IPv6 Static. Otherwise, the configuration is very similar and does not warrant additional explanation. Notice that we must include the outgoing interface with the static routes since these are single-hop BFD peers; including the outgoing interface with multi-hop BFD peers results in a broken configuration.

```
! CSR5
ipv6 route static bfd vrf L3 GigabitEthernet2.556 FE80::6
ipv6 route vrf L3 ::10:6:6:6/128 GigabitEthernet2.556 FE80::6
ipv6 route vrf L3 ::10:66:66:66/128 GigabitEthernet2.556 FE80::6

! CSR6
ipv6 route static bfd GigabitEthernet2.556 FE80::5
ipv6 route ::/0 GigabitEthernet2.556 FE80::5
```

Suddenly, this session claims to be running in “hardware”, which is the first time I’ve seen this. Even stranger is that because this is supported in “hardware”, the C-bit is set, which one would expect if this was actually true (it obviously isn’t on a CSR1000v). The C-bit functionality and purpose was discussed in the BGP section.

```
R5#show bfd neighbors ipv6 fe80::6 details | include ^Sess|^Regist
Session state is UP and not using echo function.
Session Host: Hardware
Registered protocols: IPv6 Static CEF
```

```
R5#show bfd neighbors ipv6 fe80::6 details | begin Last_packet
Last packet: Version: 1                - Diagnostic: 0
              State bit: Up             - Demand bit: 0
              Poll bit: 0               - Final bit: 0
              C bit: 1
              Multiplier: 3             - Length: 24
              My Discr.: 1              - Your Discr.: 3
              Min tx interval: 900000   - Min rx interval: 900000
              Min Echo interval: 0
```

We can see IPv6 static routing is bound to this BFD neighbor on CSR5. We can also check the static routing details to see the two routes tracked by BFD.

```
R5#show ipv6 static vrf L3 bfd
```

IPv6 Static BFD Neighbors (Table - L3)

Codes: * - installed in RIB, u/m - Unicast/Multicast only
U - Per-user Static route, N - ND Static route
P - DHCP-PD Static route

Neighbor FE80::6/GigabitEthernet2.556 [Up]

```
* ::10:66:66:66/128 via FE80::6, GigabitEthernet2.556, distance 1
* ::10:6:6:6/128 via FE80::6, GigabitEthernet2.556, distance 1
```

R5#show ipv6 route vrf L3 static | begin ^S

```
S ::10:6:6:6/128 [1/0]
    via FE80::6, GigabitEthernet2.556
S ::10:66:66:66/128 [1/0]
    via FE80::6, GigabitEthernet2.556
```

Using the same mechanism as before, we will create an IPv6 ACL to block UDP port 3784, killing the BFD single-hop control traffic. Be sure not to confuse the single-hop and multi-hop ports.

```
! CSR6
ipv6 access-list ACL_DENY_BFD_CONTROL_V6
deny udp any any eq 3784
permit ipv6 any any

interface GigabitEthernet2.556
ipv6 traffic-filter ACL_DENY_BFD_CONTROL_V6 in
```

Debugging on CSR5 reveals very similar output with IPv6 as it did for IPv4. The only difference is that there is no syslog message generated since the feature is not supported for IPv6.

```
R5#debug bfd event
R5#debug ipv6 routing
BFD-DEBUG Event: V1 FSM ld:3 handle:5 event:RX DOWN state:UP (0)
BFD-DEBUG Event: resetting timestamps ld:3 handle:5 (0)
BFD-DEBUG Event: notify client(CEF) IP:FE80::6, ld:3, handle:5, event:DOWN,
cp independent failure (0)
BFD-DEBUG Event: notify client(IPv6 Static) IP:FE80::6, ld:3, handle:5,
event:DOWN, cp independent failure (0)
[IPv6 Static Handler]IPv6RT[L3]: static, Delete next-hop for
::10:66:66:66/128, FE80::6%L3, GigabitEthernet2.556%L3
[IPv6 Static Handler]IPv6RT[L3]: static, Delete ::10:66:66:66/128 from table
[IPv6 Static Handler]IPv6RT[L3]: static, Delete next-hop for ::10:6:6:6/128,
FE80::6%L3, GigabitEthernet2.556%L3
[IPv6 Static Handler]IPv6RT[L3]: static, Delete ::10:6:6:6/128 from table
[IPv6 RIB Event Handler]IPv6RT[L3]: Event: ::10:66:66:66/128, Del, owner
static, previous None
[IPv6 RIB Event Handler]IPv6RT[L3]: Event: ::10:6:6:6/128, Del, owner static,
previous None
BFD-DEBUG Event: V1 FSM ld:3 handle:5 event:RX DOWN state:DOWN (0)
```

The BFD session is down, so we don't expect to see those routes in the VRF RIB anymore. The IPv6 output doesn't have as many flags as IPv4, but it clearly shows the BFD session as being down. This implies that the routes should not be active in the RIB. The removal of the "*" seen on the routes (left side of the screen) earlier is an indication that these routes are not installed in the RIB; we verify that they are absent.

```
R5#show ipv6 static vrf L3 bfd | begin ^Neighbor
Neighbor FE80::6/GigabitEthernet2.556 [Down]
      ::10:66:66:66/128 via FE80::6, GigabitEthernet2.556, distance 1
      ::10:6:6:6/128 via FE80::6, GigabitEthernet2.556, distance 1
```

```
R5#show ipv6 route vrf L3 static | begin ^S
[no output]
```

The process is very similar in the opposite direction as one would expect. When BFD starts receiving packets again, it notifies IPv6-static so the routes can be re-added to the RIB. Once that is complete, BFD comes fully up.

```
! CSR6
BFD-DEBUG Event: V1 FSM ld:3 handle:5 event:RX UP state:INIT (0)
BFD-DEBUG Event: resetting timestamps ld:3 handle:5 (0)
BFD-DEBUG Event: notify client(CEF) IP:FE80::6, ld:3, handle:5, event:UP,
(0)
BFD-DEBUG Event: notify client(IPv6 Static) IP:FE80::6, ld:3, handle:5,
event:UP, (0)
[IPv6 Static Handler]IPv6RT[L3]: static, Route add ::10:66:66:66/128 [new
1/0]
[IPv6 Static Handler]IPv6RT[L3]: static, Added path
FE80::6/GigabitEthernet2.556 (A:0x1/F:0x0)
[IPv6 Static Handler]IPv6RT[L3]: static, Route add ::10:6:6:6/128 [new 1/0]
[IPv6 Static Handler]IPv6RT[L3]: static, Added path
FE80::6/GigabitEthernet2.556 (A:0x1/F:0x0)
[IPv6 RIB Event Handler]IPv6RT[L3]: Event: ::10:66:66:66/128, Add, owner
static, previous None
[IPv6 RIB Event Handler]IPv6RT[L3]: Event: ::10:6:6:6/128, Add, owner static,
previous None
BFD-DEBUG Event: V1 FSM ld:3 handle:5 event:RX UP state:UP (0)
```

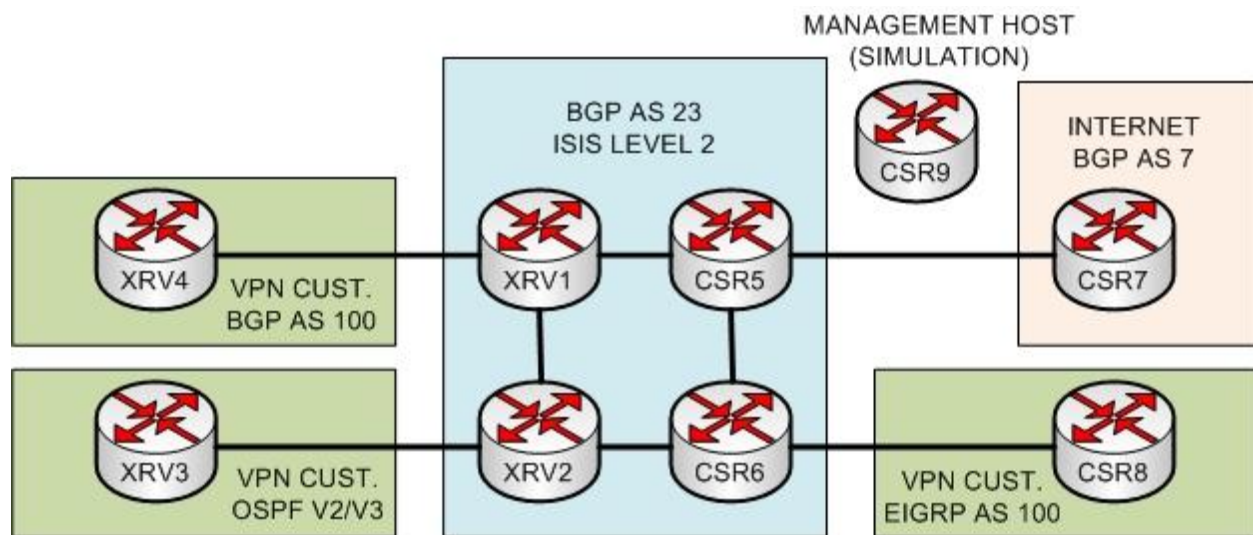
Additional Reading – Reference configurations "bfd"

41. Describe, implement, and troubleshoot control plane protection techniques

41.1 Control Plane Policing (CPP) in XE and Local Packet Transport Services (LPTS) in XR

CPP and LPTS are generally used to treat traffic destined to or originating from the router itself. This "treat" action is typically policing or dropping, but can also be QoS marking (for outbound traffic). CPP uses the standard MQC syntax and the policies can be applied inbound or outbound on the router's control-plane "subinterfaces". LPTS configuration is more like a stateless host firewall in which certain

exceptions are made dynamically as new features are configured. The network diagram below shows 4 MPLS routers servicing 4 remote sites. CSR7 represents a shared services site, like the Internet, while the other three CEs are part of an internal VPN. Different PE-CE routing protocols are used intentionally to test different CPP/LPTS filtering policies. CSR9 represents a management laptop that has an out-of-band network to reach each router. The 8 “production” routers in this lab have a separate VRF MGMT that is used for this purpose; all 9 devices share a common management LAN and IPv4/v6 subnet. The goal is to sufficiently secure the routers so they can only be managed by authorized hosts, on authorized interfaces, using authorized protocols. We will also secure the control-plane (routing, ARP, etc) to the best of our ability using CPP/LPTS. Note: Sometimes CPP is called control plane protection (CPPr), which is a more generic term that covers management plane protection also.



One of the simplest CPP / LPTS features is management plane protection (MPP). This is a way of specifying through which interfaces, and using which protocols, a router can be managed. Because every router in this lab has an OOBM interface in a separate VRF specifically for management, it would make sense to disallow management protocols on other interfaces. Before beginning, we can see that CSR5 is allowing telnet sessions from anywhere. We could secure this with VTY ACLs, but CPP is another option.

```
R7#telnet 10.5.7.5
Trying 10.5.7.5 ... Open
R5#
```

CSR5 can specify which interfaces allow telnet to be processed by the router for management purposes. In this case, only Gig1, the OOBM interface, should be used. The configuration for MPP on XE happens under the “host” subinterface. We can select from a list of protocols for each interface we list. After applying this configuration, CSR7 can no longer telnet to CSR5. The TCP SYN is not rejected, but simply discarded, as generating a TCP RST takes CPU cycles.

```
! CSR5
```

```
control-plane host
management-interface GigabitEthernet1 allow ssh tftp snmp telnet
```

```
R7#telnet 10.5.7.5
Trying 10.5.7.5 ...
% Connection timed out; remote host not responding
```

This does not affect transit traffic; telnet can still pass through the router, and CSR7 is able to access CSR6 as an example.

```
R7#telnet 10.6.8.6 /source-interface loopback0
Trying 10.6.8.6 ... Open
R6#
```

Let's assume that CSR7 should be allowed to SSH into CSR6, but only Gig1 should support all protocols supported by the router. Though the parser says BEEP and TLS1 are options, IOS-XE notifies us that they are not supported. The full configuration is also shown below; it also reveals that typing "allow" by itself is just a macro for manually enabling all available management services.

```
R6(config-cp-host)#management-interface GigabitEthernet1 allow
TLS1 is not a supported management protocol in IOSXE. It will be ignored as
if it is not configured.
BEEP is not a supported management protocol in IOSXE. It will be ignored as
if it is not configured.
```

```
! CSR6
control-plane host
management-interface GigabitEthernet1 allow ftp http https ssh tftp snmp
beep telnet t11
management-interface GigabitEthernet2.568 allow ssh
```

External routers can SSH into CSR6 via Gig2.568 (as the target IP), but can no longer telnet. CSR9, the authorized management station, can do both.

```
R7#telnet 10.6.8.6 /source-interface lo0
Trying 10.6.8.6 ...
% Connection timed out; remote host not responding
```

```
R8#ssh -l test 10.6.8.6
Password:
```

```
R9#telnet 10.31.5.206
Trying 10.31.5.206 ... Open
R6#
```

```
R9#ssh -l test 10.31.5.206
Password:
```

This functionality can be verified using a single show command. Outputs from both CSR5 and CSR6 are shown below. It includes packet counters also.

```
R5#show management-interface
Management interface GigabitEthernet1
      Protocol      Packets processed
      ssh           0
      tftp           0
      snmp           0
      telnet        70
```

```
R6#show management-interface
Management interface GigabitEthernet1
      Protocol      Packets processed
      ftp           0
      http          0
      https         0
      ssh           30
      tftp          0
      snmp          0
      beep          0
      telnet        22
      t11           0
```

```
Management interface GigabitEthernet2.568
      Protocol      Packets processed
      ssh           30
```

The policing aspect of CPP is also powerful. This is commonly used to prevent common DoS attacks like ARP or ping flooding. My personal favorite outbound functionality is to police the rate at which ICMP unreachables are sent. Both XE and XR have one-line commands for this, but filtering the type of ICMP unreachable is valuable to achieve greater granularity in high-security networks. This example shows portions of a configuration that is used to protect CSR5 from ICMP unreachable CPU-hogging. Rather than disable ICMP unreachables on the transit interfaces, we allow the unreachables needed for PMTUd (packet-too-big), along with the familiar traceroute unreachables (port unreachable and time exceeded). This allows MPLS administrators to traceroute internally for troubleshooting, but ensures the routers don't give away "too" much information. It also allows customers to traceroute across the MPLS network to check their own connectivity within their VPN. The flexibility of MQC is beneficial because needing to enable/disable a new unreachable means modifying an ACL, which is straightforward. The syntax is nearly identical for IPv4 and IPv6, making it easy to read, operate, and troubleshoot. This is also an effective security mechanism when tied in with disabling MPLS TTL propagation (added on all four core routers but not shown).

```
! CSR5
ip access-list extended ACL_ALLOWED_ICMP_UNREACHABLES
```

```

    permit icmp any any time-exceeded
    permit icmp any any port-unreachable
    permit icmp any any packet-too-big
ip access-list extended ACL_DENIED_ICMP_UNREACHABLES
    permit icmp any any unreachable

ipv6 access-list ACL_ALLOWED_ICMPV6_UNREACHABLES
    permit icmp any any time-exceeded
    permit icmp any any port-unreachable
    permit icmp any any packet-too-big
ipv6 access-list ACL_DENIED_ICMPV6_UNREACHABLES
    permit icmp any any unreachable

class-map match-any CMAP_ALLOWED_ICMP_UNREACHABLES
    match access-group name ACL_ALLOWED_ICMP_UNREACHABLES
    match access-group name ACL_ALLOWED_ICMPV6_UNREACHABLES

class-map match-any CMAP_DENIED_ICMP_UNREACHABLES
    match access-group name ACL_DENIED_ICMP_UNREACHABLES
    match access-group name ACL_DENIED_ICMPV6_UNREACHABLES

policy-map PMAP_CPP_OUT
    class CMAP_ALLOWED_ICMP_UNREACHABLES
        police rate 10 pps burst 10 packets
        conform-action transmit
        exceed-action drop
    class CMAP_DENIED_ICMP_UNREACHABLES
        police rate 1 pps conform-action drop

```

The above configuration polices the allowed unreachable described earlier to 10 packets per second with no inter-packet limit until the limit of 10 is reached. Because IOS-XE does not have the “drop” keyword like IOS does, we can police the prohibited unreachable to any rate and set the conform action to drop. This means that, regardless of the generation rate, 100% of packets matching this class are dropped. We can further extend this policy to do some basic QoS marking for our outbound management traffic. In networks with only in-band management mechanisms, sometimes it isn’t always known from which interface a device will be managed. Using CPP (or local PBR) can obviate the need to make per-interface policies and risk double-classifying transit traffic. Anytime the router generates a “management” packet like syslog, TFTP, RADIUS, or FTP, it will be marked DSCP CS2 (DSCP 16).

```

! CSR5
ip access-list extended ACL_LOCAL_MGMT_TRAFFIC
    permit udp any any eq syslog
    permit udp any eq syslog any
    permit udp any any range 1812 1813
    permit udp any range 1812 1813 any
    permit udp any any eq tftp
    permit udp any eq tftp any

```

```

permit tcp any any range ftp-data ftp
permit tcp any range ftp-data ftp any

class-map match-all CMAP_LOCAL_MGMT_TRAFFIC
match access-group name ACL_LOCAL_MGMT_TRAFFIC

policy-map PMAP_CPP_OUT
class CMAP_LOCAL_MGMT_TRAFFIC
set dscp cs2

```

The policy now contains the ICMP unreachable limiting and QoS marking features. It is applied under the control-plane stanza and can be verified similar to a normal policy-map. Packets not classified by this policy are ignored (untreated).

```

! CSR5
control-plane
service-policy output PMAP_CPP_OUT

```

```

R5#show policy-map control-plane output | include Class|packet
Class-map: CMAP_ALLOWED_ICMP_UNREACHABLES (match-any)
  0 packets, 0 bytes
    rate 10 pps, burst 10 packets
    conformed 0 packets, 0 bytes; actions:
    exceeded 0 packets, 0 bytes; actions:
Class-map: CMAP_DENIED_ICMP_UNREACHABLES (match-any)
  0 packets, 0 bytes
    rate 1 pps, burst 1 packets
    conformed 0 packets, 0 bytes; actions:
    exceeded 0 packets, 0 bytes; actions:
Class-map: CMAP_LOCAL_MGMT_TRAFFIC (match-all)
  0 packets, 0 bytes
Class-map: class-default (match-any)
  16 packets, 6823 bytes

```

We will perform several tests to ensure the router takes actions in accordance with the CPP policies. First, we will traceroute to CSR5 from XRv1. CSR5 should originate “port unreachable” messages for each probe sent by XRv1. Since three probes were sent, three unreachables were generated.

```

RP/0/0/CPU0:XRv1#traceroute 23.0.0.5
Type escape sequence to abort.
Tracing the route to 23.0.0.5

```

```

1  23.5.11.5 0 msec 0 msec 0 msec

```

```

R5#show policy-map control-plane output | include Class|packet
Class-map: CMAP_ALLOWED_ICMP_UNREACHABLES (match-any)
  3 packets, 168 bytes
    rate 10 pps, burst 10 packets

```

```

    conformed 3 packets, 168 bytes; actions:
    exceeded 0 packets, 0 bytes; actions:
Class-map: CMAP_DENIED_ICMP_UNREACHABLES (match-any)
  0 packets, 0 bytes
    rate 1 pps, burst 1 packets
    conformed 0 packets, 0 bytes; actions:
    exceeded 0 packets, 0 bytes; actions:
Class-map: CMAP_LOCAL_MGMT_TRAFFIC (match-all)
  0 packets, 0 bytes
Class-map: class-default (match-any)
  150 packets, 57709 bytes

```

Traceroute across CSR5, even from within a VRF, is subject to the same checks. This time, CSR5 originates the “time-exceeded” unreachable for CSR7 since XRv4 is farther away. XRv4 will generate the port unreachables, but that is regular transit traffic to the core routers.

```

R7#traceroute 10.14.14.14 source loopback0
Type escape sequence to abort.
Tracing the route to 10.14.14.14
VRF info: (vrf in name/id, vrf out name/id)
 1 10.5.7.5 1 msec 1 msec 1 msec
 2 23.5.11.11 [MPLS: Label 91004 Exp 0] 4 msec 3 msec 3 msec
 3 10.11.14.14 10 msec 13 msec 13 msec

R5#show policy-map control-plane output | include Class|packet
Class-map: CMAP_ALLOWED_ICMP_UNREACHABLES (match-any)
  6 packets, 336 bytes
    rate 10 pps, burst 10 packets
    conformed 6 packets, 336 bytes; actions:
    exceeded 0 packets, 0 bytes; actions:
Class-map: CMAP_DENIED_ICMP_UNREACHABLES (match-any)
  0 packets, 0 bytes
    rate 1 pps, burst 1 packets
    conformed 0 packets, 0 bytes; actions:
    exceeded 0 packets, 0 bytes; actions:
Class-map: CMAP_LOCAL_MGMT_TRAFFIC (match-all)
  0 packets, 0 bytes
Class-map: class-default (match-any)
  276 packets, 106709 bytes

```

A static default route on CSR7 allows it to send bogus traffic to CSR5. We expect CSR5 to not originate any “host unreachable” or “network unreachable” messages, and expect to see five denials in the CPP counters. The catch-all of denying all other ICMP unreachables also includes these; this makes the policy simple by not having to enumerate every denied unreachable in “blacklist” form. Though these 5 packets conformed to the policer’s CIR, the conform-action is drop.

```

R7(config)#ip route 0.0.0.0 0.0.0.0 10.5.7.5

```

```
R7#ping 1.2.3.4
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 1.2.3.4, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
```

```
R5#show policy-map control-plane output | include Class|packet
  Class-map: CMAP_ALLOWED_ICMP_UNREACHABLES (match-any)
    6 packets, 336 bytes
      rate 10 pps, burst 10 packets
      conformed 6 packets, 336 bytes; actions:
      exceeded 0 packets, 0 bytes; actions:
  Class-map: CMAP_DENIED_ICMP_UNREACHABLES (match-any)
    5 packets, 280 bytes
      rate 1 pps, burst 1 packets
      conformed 5 packets, 280 bytes; actions:
      exceeded 0 packets, 0 bytes; actions:
  Class-map: CMAP_LOCAL_MGMT_TRAFFIC (match-all)
    0 packets, 0 bytes
  Class-map: class-default (match-any)
    529 packets, 205423 bytes
```

Next, we will configure remote logging on CSR5 to send logs to CSR6 as a bogus syslog server. These packets should be marked DSCP CS2 by the router regardless of which interface or VRF they exit. CSR6 will use EPC to monitor these packets inbound. The TOS is 0x40 (0100 0000, decimal 64, or DSCP CS2) as shown by the packet dump. You can clearly see the log message inside the payload of the packet shown in green. I did this twice for sanity, which is why two packets are counted in the CPP policy. The policy could also include an IPv6 ACL to include IPv6 management ports, but I intentionally left that out to demonstrate that you can be selective.

```
! CSR5
logging host 23.0.0.6
```

```
R5#show policy-map control-plane output | include Class|packet
  Class-map: CMAP_ALLOWED_ICMP_UNREACHABLES (match-any)
    6 packets, 336 bytes
      rate 10 pps, burst 10 packets
      conformed 6 packets, 336 bytes; actions:
      exceeded 0 packets, 0 bytes; actions:
  Class-map: CMAP_DENIED_ICMP_UNREACHABLES (match-any)
    5 packets, 280 bytes
      rate 1 pps, burst 1 packets
      conformed 5 packets, 280 bytes; actions:
      exceeded 0 packets, 0 bytes; actions:
  Class-map: CMAP_LOCAL_MGMT_TRAFFIC (match-all)
    2 packets, 222 bytes
  Class-map: class-default (match-any)
```

992 packets, 387000 bytes

R6#show monitor capture CAP buffer dump

```
1
0000: 005056A9 DE0D0050 56A9DC63 81000DE4 .PV....PV..c....
0010: 08004540 006F0003 0000FF11 872B1705 ..E@.o.....+..
0020: 06051700 0006CC24 0202005B B6F83C31 .....$...[...<1
0030: 38393E32 31373A20 2A417567 20313820 89>217: *Aug 18
0040: 32323A31 393A3433 2E383533 3A202553 22:19:43.853: %S
0050: 59532D35 2D434F4E 4649475F 493A2043 YS-5-CONFIG_I: C
0060: 6F6E6669 67757265 64206672 6F6D2063 onfigured from c
0070: 6F6E736F 6C652062 7920636F 6E736F6C onsole by consol
0080: 65e
```

To quickly verify the IPv6 components of this policy, I will clear the counters and re-test with traceroute. From CSR7, we will send traffic to XrV4 and note that CSR5 generates three “time-exceeded” messages.

R5#clear control-plane

R7#traceroute

Protocol [ip]: ipv6

Target IPv6 address: ::10:14:14:14

Source address: ::77:7:7:7

Type escape sequence to abort.

Tracing the route to ::10:14:14:14

```
1 2001:10:5:7::5 1 msec 1 msec 1 msec
2 2001:10:11:14::1 [MPLS: Label 91005 Exp 0] 7 msec 3 msec 5 msec
3 ::10:14:14:14 [AS 100] 154 msec 3 msec 3 msec
```

R5#show policy-map control-plane output | include Class|packet

Class-map: CMAP_ALLOWED_ICMP_UNREACHABLES (match-any)

3 packets, 288 bytes

rate 10 pps, burst 10 packets

conformed 3 packets, 288 bytes; actions:

exceeded 0 packets, 0 bytes; actions:

Class-map: CMAP_DENIED_ICMP_UNREACHABLES (match-any)

0 packets, 0 bytes

rate 1 pps, burst 1 packets

conformed 0 packets, 0 bytes; actions:

exceeded 0 packets, 0 bytes; actions:

Class-map: CMAP_LOCAL_MGMT_TRAFFIC (match-all)

0 packets, 0 bytes

Class-map: class-default (match-any)

92 packets, 35528 bytes

Adding a static IPv6 default route to CSR7, we see that the router does not create “host unreachable” or “network unreachable”, as programmed.


```

! CSR7
ipv6 route ::/0 2001:10:5:7::5

R7#ping ::1:2:3:4 source lo0
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to ::1:2:3:4, timeout is 2 seconds:
Packet sent with a source address of ::77:7:7:7
.....
Success rate is 0 percent (0/5)

R5#show policy-map control-plane output | include Class|packet
  Class-map: CMAP_ALLOWED_ICMP_UNREACHABLES (match-any)
    3 packets, 288 bytes
      rate 10 pps, burst 10 packets
      conformed 3 packets, 288 bytes; actions:
      exceeded 0 packets, 0 bytes; actions:
  Class-map: CMAP_DENIED_ICMP_UNREACHABLES (match-any)
    5 packets, 740 bytes
      rate 1 pps, burst 1 packets
      conformed 5 packets, 740 bytes; actions:
      exceeded 0 packets, 0 bytes; actions:
  Class-map: CMAP_LOCAL_MGMT_TRAFFIC (match-all)
    0 packets, 0 bytes
  Class-map: class-default (match-any)
    214 packets, 82735 bytes

```

One final test on CSR5 is to test the policer for the unreachable. CSR6 will send 15 of traceroute probes and expect to see the 11th one dropped (green). CSR5 responds with 10 responses (yellow), drops one, and responds with the remaining 4. No other classes were matched, as expected. The total is now 18 packets; 3 from earlier plus 15 just now, and of those 15, 1 was in excess of the 10 pps rate.

```

R6#traceroute 23.0.0.5 probe 15
Type escape sequence to abort.
Tracing the route to 23.0.0.5
VRF info: (vrf in name/id, vrf out name/id)
 1 23.5.6.5 2 msec 1 msec 0 msec 1 msec 1 msec 1 msec 0 msec 1 msec 1 msec 0
msec * 1 msec 1 msec 0 msec 1 msec

```

```

R5#show policy-map control-plane output | include Class|packet
  Class-map: CMAP_ALLOWED_ICMP_UNREACHABLES (match-any)
    18 packets, 1128 bytes
      rate 10 pps, burst 10 packets
      conformed 17 packets, 1072 bytes; actions:
      exceeded 1 packets, 56 bytes; actions:
  Class-map: CMAP_DENIED_ICMP_UNREACHABLES (match-any)
    5 packets, 740 bytes
      rate 1 pps, burst 1 packets

```

```

    conformed 5 packets, 740 bytes; actions:
    exceeded 0 packets, 0 bytes; actions:
Class-map: CMAP_LOCAL_MGMT_TRAFFIC (match-all)
  1 packets, 111 bytes
Class-map: class-default (match-any)
  2349 packets, 912636 bytes

```

We can also use CPP for inbound policing. For example, CSR5 and CSR6 already know what protocols must be accepted inbound (BGP, LDP, etc), so we can manually configure a policy to only allow those. As we will see later, LPTS on XR does this automatically, but CPP on XE does not. A router can have both inbound and outbound policies concurrently, but we will configure the inbound policy on CSR6 for variety. Specifically, the goals will be to allow BGP sessions to 23.0.0.5 (source port 179 only, since the RR is using dynamic peer-groups). LDP discovery (UDP) will be allowed for anyone in the 23.0.0.0/12 subnet with LDP sessions (TCP) allowed for 23.0.0.0/24. The LDP flows are unlimited for now, but we want to have a placeholder in the policy to limit it later. BGP is limited to 64 kbps and also must be DSCP CS6. Additionally, critical non-IP flows, such as ARP and IS-IS, will be policed to an aggregate of 32 kbps. All other BGP and LDP should be denied, with remaining unknown traffic allowed to pass. If you make a comprehensive policy, it will take longer to program and validate, so we will limit it to a few protocols. A “drop by default” policy is always more effective but requires more careful planning.

```

! CSR6
ip access-list extended ACL_ALLOWED_BGP
  permit tcp host 23.0.0.5 eq bgp host 23.0.0.6 dscp cs6
ip access-list extended ACL_ALLOWED_LDP
  permit udp 23.0.0.0 0.15.255.255 eq 646 host 224.0.0.2 eq 646
  permit tcp 23.0.0.0 0.255.255.255 host 23.0.0.6 eq 646
  permit tcp 23.0.0.0 0.255.255.255 eq 646 host 23.0.0.6
ip access-list extended ACL_DENIED_BGP_LDP
  permit tcp any any eq bgp
  permit tcp any eq bgp any
  permit tcp any any eq 646
  permit tcp any eq 646 any
  permit udp any any eq 646
  permit udp any eq 646 any

class-map match-all CMAP_ALLOWED_BGP
  match access-group name ACL_ALLOWED_BGP
class-map match-any CMAP_CRITICAL_NON_IP
  match protocol arp
class-map match-all CMAP_ALLOWED_LDP
  match access-group name ACL_ALLOWED_LDP
class-map match-all CMAP_DENIED_BGP_LDP
  match access-group name ACL_DENIED_BGP_LDP

policy-map CMAP_CPP_IN
  class CMAP_ALLOWED_LDP
    ! No action currently, just match the packets

```

```

class CMAP_ALLOWED_BGP
  police cir 64000
class CMAP_CRITICAL_NON_IP
  police cir 32000
class CMAP_DENIED_BGP_LDP
  police cir 8000 conform-action drop

control-plane
service-policy input CMAP_CPP_IN

```

Attempting to match any kind of NBAR protocols, even IS-IS, is not allowed. The only “match protocol” option that appears to be supported is ARP, so we will police that only. That is why IS-IS is removed from the configuration above despite the desire to specify it. I tried to use “cls_is” instead of “isis” but the CSR1000v does not support that for any application.

```

! With class-map "match protocol isis"
R6(config-cp)#service-policy input CMAP_CPP_IN
error: failed to install policy map CMAP_CPP_IN

```

```

R6(config-cmap)#match protocol cls_is
'match protocol cls_is' is not supported on this platform

```

With the network in steady state, we don’t see any packet drops. We see plenty of LDP/BGP traffic, along with other flows in the network.

```

R6#show policy-map control-plane input | include Class|packets
Class-map: CMAP_ALLOWED_LDP (match-all)
92 packets, 7456 bytes
Class-map: CMAP_ALLOWED_BGP (match-all)
8 packets, 700 bytes
  conformed 8 packets, 700 bytes; actions:
  exceeded 0 packets, 0 bytes; actions:
Class-map: CMAP_CRITICAL_NON_IP (match-any)
  0 packets, 0 bytes
  conformed 0 packets, 0 bytes; actions:
  exceeded 0 packets, 0 bytes; actions:
Class-map: CMAP_DENIED_BGP_LDP (match-all)
  0 packets, 0 bytes
  conformed 0 packets, 0 bytes; actions:
  exceeded 0 packets, 0 bytes; actions:
Class-map: class-default (match-any)
232 packets, 70701 bytes

```

First, we will “clear arp” several times in quick succession to see if we can trip the policer. Probably not, but at least we see matches on the class. On a router with hundreds of interfaces (or a lower CIR), this would be easier. Of greater importance is confirming that the CPP policy is matching the ARP traffic.

```
R6#show policy-map control-plane input class CMAP_CRITICAL_NON_IP | include
packet
    311 packets, 19120 bytes
    conformed 311 packets, 19120 bytes; actions:
    exceeded 0 packets, 0 bytes; actions:
```

Next, we will clear BGP and LDP to ensure they can re-establish successfully from a cold state. This is an important step to ensure your ACLs are configured correctly and are not working coincidentally due to current session formation. We should see several more BGP and LDP hits. We also verify that the BGP and LDP neighbors came up successfully.

```
R6#show policy-map control-plane input class CMAP_ALLOWED_BGP | include
packet
    31 packets, 3954 bytes
    conformed 31 packets, 3954 bytes; actions:
    exceeded 0 packets, 0 bytes; actions:
```

```
R6# show policy-map control-plane input class CMAP_ALLOWED_LDP | include
packet
    319 packets, 26186 bytes
```

```
R6#show bgp vpnv4 unicast all summary | include 23.0.0.5
23.0.0.5      4      23      20      5      11      0      0 00:00:58      4
```

```
R6#show mpls ldp neighbor | include Peer
Peer LDP Ident: 23.0.0.5:0; Local LDP Ident 23.0.0.6:0
Peer LDP Ident: 23.0.0.12:0; Local LDP Ident 23.0.0.6:0
```

Next, we will configure a BGP peer from CSR8 to CSR6. We expect this to be rejected; the “DENIED” class-map matches these entries and drops the packets.

```
! CSR8
router bgp 8
  bgp log-neighbor-changes
  neighbor 10.6.8.6 remote-as 23
```

```
R6#show policy-map control-plane input class CMAP_DENIED_BGP_LDP | include
packets
    4 packets, 248 bytes
    conformed 4 packets, 248 bytes; actions:
    exceeded 0 packets, 0 bytes; actions:
```

We will shutdown that BGP peer on CSR8, clear CPP counters on CSR6, and configure a targeted LDP session on CSR8 to repeat a similar test for LDP. We expect this to be discarded just like the BGP session.

```
R6#clear control-plane
```

```

! CSR8
mpls ldp neighbor 10.6.8.6 targeted ldp

R6#show policy-map control-plane input class CMAP_DENIED_BGP_LDP | include
packets
    2 packets, 160 bytes
        conformed 2 packets, 160 bytes; actions:
        exceeded 0 packets, 0 bytes; actions:

```

The MPP functionality in XR, as a component of LPTS, has the additional capability to specify interfaces as in-band or out-of-band. OOB interfaces can never be used for transit, even if there is a valid routing path, while in-band interfaces can. Typically, the OOBM interface will allow more management protocols on the device, while in-band may allow none or very few. In the example below, the OOB interface allows all protocols provided they are sourced from within the specified subnets. The inband interface only specifies the link to XRv4 and allows telnet sessions sourced from XRv4's loopback only, arriving on that directly-connected interface.

```

! XRv11
control-plane
management-plane
inband
    interface GigabitEthernet0/0/0/0.541
        allow Telnet peer
        address ipv4 10.14.14.14

out-of-band
    interface MgmtEth0/0/CPU0/0
        allow all peer
        address ipv4 10.31.5.128/25
        address ipv6 FD00:10:31:5::209

```

Before testing MPP, we can verify that the LPTS tables are correct. Note that you must enable the management services in question (telnet, SNMP, SSH, etc) before you'll see an LPTS entry. If the service is disabled, LPTS has no reason to create a permissive exception for it. The TCP entries are shown first. Notice how the OOB interface allows telnet from the specified subnets, while the inband interface only allows it from XRv14's (CE) loopback. You can think of these entries like reflexive ACLs in a way.

```

RP/0/0/CPU0:XRv1#show lpts bindings l4-protocol 6 brief
@ - Indirect binding; Sc - Scope
  Location  Clnt Sc L3   L4     VRF-ID   Interface   Local-Address,Port Remote-
Address,Port
-----
0/0/CPU0   TCP  LR  IPV4 TCP    default  any         23.0.0.11,48860 23.0.0.5,179
0/0/CPU0   TCP  LR  IPV4 TCP    default  any         23.0.0.11,179 23.0.0.5
0/0/CPU0   TCP  LR  IPV4 TCP    default  any         23.0.0.11,15666 23.0.0.5,646
0/0/CPU0   TCP  LR  IPV4 TCP    default  any         23.0.0.11,646 23.0.0.12,41852
0/0/CPU0   TCP  LR  IPV4 TCP    default  any         any,646 23.0.0.12

```

0/0/CPU0	TCP	LR	IPv4	TCP	C	any	10.11.14.11,179	10.11.14.14
0/0/CPU0	TCP	LR	IPv4	TCP	C	Gi0/0/0/0.541	any,23	10.14.14.14
0/0/CPU0	TCP	LR	IPv4	TCP	C	any	10.11.14.11,179	10.11.14.14,21987
0/0/CPU0	TCP	LR	IPv4	TCP	MGMT	Mg0/0/CPU0/0	any,23	10.31.5.128/25
0/0/CPU0	TCP	LR	IPv6	TCP	MGMT	Mg0/0/CPU0/0	any,23	fd00:10:31:5::209

The UDP entries are similar. I have enabled SNMP on XRv1; UDP ports 161 and 162 are now accessible from the specified subnets in the OOB configuration. Notice that no in-band interfaces can manage the router via SNMP which is consistent with our MPP configuration.

```
RP/0/0/CPU0:XRv1#show lpts bindings l4-protocol 17 brief
```

```
@ - Indirect binding; Sc - Scope
```

Location	Clnt	Sc	L3	L4	VRF-ID	Interface	Local-Address,Port	Remote-Address,Port

0/0/CPU0	UDP	LR	IPv4	UDP	default	Gi0/0/0/0.512	224.0.0.2,646	any
0/0/CPU0	UDP	LR	IPv4	UDP	default	Gi0/0/0/0.551	224.0.0.2,646	any
0/0/CPU0	UDP	LR	IPv4	UDP	default	any	any,496	any
0/0/CPU0	UDP	LR	IPv4	UDP	MGMT	Mg0/0/CPU0/0	any,161	10.31.5.128/25
0/0/CPU0	UDP	LR	IPv4	UDP	MGMT	Mg0/0/CPU0/0	any,162	10.31.5.128/25
0/0/CPU0	UDP	LR	IPv6	UDP	MGMT	Mg0/0/CPU0/0	any,162	fd00:10:31:5::209
0/0/CPU0	UDP	LR	IPv6	UDP	MGMT	Mg0/0/CPU0/0	any,161	fd00:10:31:5::209

Some quick data tests show that our policy works. XRv14 can telnet only when sourced from the loopback0 and routing over the directly-connected link (MPP in-band on XRv1). Unlike CPP, this LPTS feature does send TCP RSTs since it's more like a stateless firewall or access-list.

```
RP/0/0/CPU0:XRv4#telnet 10.11.14.11
```

```
Trying 10.11.14.11...
```

```
telnet: Unable to connect to remote host: Connection refused
```

```
telnet[65692]: %IP-telnet-6-CONNECT : Couldn't connect to remote host
10.11.14.11(Connection refused).
```

```
RP/0/0/CPU0:XRv4#telnet 10.11.14.11 source-interface loopback0
```

```
Trying 10.11.14.11...
```

```
Use specified source interface(Loopback0).
```

```
Use 10.14.14.14 as local address.
```

```
Connected to 10.11.14.11.
```

All routers on the OOB LAN can telnet to XRv1, including CSR9, using IPv4. All of them have IPv4 addresses in the 10.31.5.128/25 range. Using IPv6, however, only CSR9 can. LPTS can be very flexible in this way without having to define complex ACLs.

```
R9#telnet 10.31.5.211
```

```
Trying 10.31.5.211 ... Open
```

```
R7#telnet 10.31.5.211 /vrf MGMT
```

```
Trying 10.31.5.211 ... Open
```

```
R9#telnet FD00:10:31:5::211
Trying FD00:10:31:5::211 ... Open

R7#telnet FD00:10:31:5::211 /vrf MGMT
Trying FD00:10:31:5::211 ...
% Connection refused by remote host
```

Several other LPTS entries are created totally dynamically. Below are the LPTS entries on XRv1 for BGP and LDP. The BGP sessions are preconfigured and known, but the LDP ones are built based on the UDP link-discovery mechanism. We can see the BGP session to CSR5 (RR) and XRv4 in 2 separate VRFs. Notice there are 4 entries; one is specified including the source port of the TCP client, while the other is a generic “listener” in a traditional TCP server role. When a new client is configured, the corresponding “listener” entry is added to the LPTS table. In routers that actually have line-cards, these entries are policed individually; a generic listener is scrutinized more heavily than an established session, for example. The show commands to see the policed rates aren’t available in XRv.

```
RP/0/0/CPU0:XRv1#show lpts bindings l4-protocol 6 brief | include 179
0/0/CPU0    TCP    LR  IPV4  TCP    default  any          23.0.0.11,48860 23.0.0.5,179
0/0/CPU0    TCP    LR  IPV4  TCP    default  any          23.0.0.11,179 23.0.0.5
0/0/CPU0    TCP    LR  IPV4  TCP    C        any          10.11.14.11,179 10.11.14.14
0/0/CPU0    TCP    LR  IPV4  TCP    C        any          10.11.14.11,179 10.11.14.14,21987
```

LDP works similarly. There are established sessions to CSR5 and XRv12 in the global table which are shown with their fully-qualified TCP port lists. The third entry is odd; this was created only because XRv1 is the TCP server in this connection, whereas for CSR5 it was the client. With XRv2 as a TCP client, I believe that LPTS assumes that TCP client reserves the right to tear-down and reestablish the session using a new port, which would explain the more liberal entry. Again, in a hardware platform, this would be policed more heavily than the established sessions.

```
RP/0/0/CPU0:XRv1#show lpts bindings l4-protocol 6 brief | include 646
0/0/CPU0    TCP    LR  IPV4  TCP    default  any          23.0.0.11,15666 23.0.0.5,646
0/0/CPU0    TCP    LR  IPV4  TCP    default  any          23.0.0.11,646 23.0.0.12,41852
0/0/CPU0    TCP    LR  IPV4  TCP    default  any          any,646 23.0.0.12
```

There isn’t a keyword for revealing it, but configuring IS-IS automatically allows CLNS traffic to be accepted by the control-plane. Many of the specific filters aren’t in play, since CLNS routing isn’t used.

```
RP/0/0/CPU0:XRv1#show lpts bindings brief | include CLNS
0/0/CPU0    CLNS  LR  CLNS  -      default  Gi0/0/0/0.512 - -
0/0/CPU0    CLNS  LR  CLNS  -      default  Gi0/0/0/0.551 - -
```

As a best practice, it usually makes sense to police totally unknown BGP/LDP entries (the equivalent of BGP/LDP “any any”) to zero, effectively dropping the traffic. There are some corner cases where not applying this draconian policer could be valuable, such as dynamically established LDP targeted session

of BGP dynamic neighbors (not supported in XR yet), but they are generally rare. Again, this cannot be demonstrated here since XRv doesn't support the syntax.

Personally, I use the "show lpts bindings" command to verify most LPTS MPP configurations, but two other concepts are important. There is an iFIB, or internal FIB, that exists on an individual linecard. This determines how packets are to be punted, and where. Some things can be punted to the linecard CPU, such as ARP and BFD, which saves resources. Other traffic like IGP/BGP updates need to go to the Route-Switch Processors (RSPs, the main CPUs), which is beyond the linecard through the router fabric. Here is a trimmed-down output of some iFIB entries. Because there are no linecard CPUs, everything is "delivered" to the RSP. If BFD worked on XRv, we could test it in conjunction with LPTS, but it doesn't.

```
RP/0/0/CPU0:XRv1#show lpts ifib brief
Slice      VRF-ID    L4        Interface  Dlvr       Local-Address,Port Remote-
Address,Port
-----
RAWIP4     default  IGMP      any        0/0/CPU0   any any
RAWIP4     default  RSVP      any        0/0/CPU0   any any
RAWIP4     default  RSVP      Gi0/0/0/0.512 0/0/CPU0   any any
RAWIP4     default  RSVP      Gi0/0/0/0.551 0/0/CPU0   any any
RAWIP4     default  L2TPV3    any        0/0/CPU0   any any
RAWIP6     default  ICMP6     any        0/0/CPU0   any,MLDLQUERY any
RAWIP6     default  ICMP6     any        0/0/CPU0   any,LSTNRREPORT any
RAWIP6     default  ICMP6     any        0/0/CPU0   any,MLDLSTNRDN any
RAWIP6     default  ICMP6     any        0/0/CPU0   any,LSTNRREPORT any
BGP4       C        TCP       any        0/0/CPU0   10.11.14.11,179 10.11.14.14,21987
BGP4       C        TCP       any        0/0/CPU0   10.11.14.11,179 10.11.14.14
BGP4       default  TCP       any        0/0/CPU0   23.0.0.11,179 23.0.0.5
BGP4       default  TCP       any        0/0/CPU0   23.0.0.11,48860 23.0.0.5,179
UDP4       default  UDP       Gi0/0/0/0.512 0/0/CPU0   224.0.0.2,646 any
UDP4       default  UDP       Gi0/0/0/0.551 0/0/CPU0   224.0.0.2,646 any
UDP4       *        UDP       Mg0/0/CPU0/0 0/0/CPU0   any,161 10.31.5.128/25
UDP4       *        UDP       Mg0/0/CPU0/0 0/0/CPU0   any,162 10.31.5.128/25
[snip]
```

The pre-iFIB (piFIB) shows the entries before they are added to the iFIB. This is the first level of filtering that happens and the iFIB entries are populated next. This shows additional delivery options such as reassemble (all IPv4 fragments) or drop ICMP pings, timestamps, and mask requests. The iFIB will deliver traffic after the piFIB has performed dropping or reassembly as needed.

```
RP/0/0/CPU0:XRv1#show lpts pifib brief
```

```
* - Any VRF; I - Local Interest;
```

```
X - Drop; R - Reassemble;
```

Type	VRF-ID	L4	Interface	Deliver	Local-Address,Port	Remote-
Address,Port						
ISIS	default	-	Gi0/0/0/0.512	0/0/CPU0	- -	
ISIS	default	-	Gi0/0/0/0.551	0/0/CPU0	- -	
ISIS	*	-	any	0/0/CPU0	- -	
IPv4_frag	*	any	any	R	any any	


```

IPv4_echo * ICMP any XI any,ECHO any
IPv4 default IGMP any 0/0/CPU0 any any
IPv4 default TCP any 0/0/CPU0 23.0.0.11,15666 23.0.0.5,646
IPv4 default TCP any 0/0/CPU0 23.0.0.11,48860 23.0.0.5,179
IPv4 default TCP any 0/0/CPU0 23.0.0.11,179 23.0.0.5
IPv4 default TCP any 0/0/CPU0 23.0.0.11,646 23.0.0.12,31388
IPv4 default TCP any 0/0/CPU0 any,646 23.0.0.12
IPv4 default UDP Gi0/0/0/0.512 0/0/CPU0 224.0.0.2,646 any
IPv4 default UDP Gi0/0/0/0.551 0/0/CPU0 224.0.0.2,646 any
IPv4 default UDP any 0/0/CPU0 any,496 any
IPv4 default RSVP any 0/0/CPU0 any any
IPv4 default RSVP Gi0/0/0/0.512 0/0/CPU0 any any
IPv4 default RSVP Gi0/0/0/0.551 0/0/CPU0 any any
IPv4 default L2TPV3 any 0/0/CPU0 any any
IPv4 * ICMP any 0/0/CPU0 any,ECHOREPLY any
IPv4 * ICMP any XI any,TSTAMP any
IPv4 * ICMP any XI any,MASKREQ any

```

Additional Reading – Reference configurations "cpp-lpts"

42. Describe, implement, and troubleshoot logging and SNMP security

42.1 Logging

This feature is discussed/tested in detail in the security section. Logging enhances security by notifying network administrators of a problem in the network. Logging on ACL matches and other critical events allows administrators (and machines) to correlate events and identify security vulnerabilities.

42.2 SNMP security

This feature is discussed/tested in detail in the security section. In summary, there are three common versions of SNMP. SNMPv1 used community strings, much like SNMPv2c, and both provide very little security. SNMPv3 introduces authentication and encryption for SNMP traffic, along with usernames and groups. These groups are assigned specific views into the network and the SNMPv3 architecture allows for very granular control of permissions. ACLs can also be used to restrict access for both IPv4 and IPv6.

43. Describe, implement, and troubleshoot timing

PTP and SyncE are not supported on the CSR1000v or XRv at this time as confirmed by Cisco engineering. Only NTP is analyzed from a configuration and troubleshooting perspective. PTP and SyncE basic operation and architectures are described in text only. Below is a brief comparison between NTP and PTP. SyncE doesn't directly compare to these IP-based protocols and their distribution mechanisms.

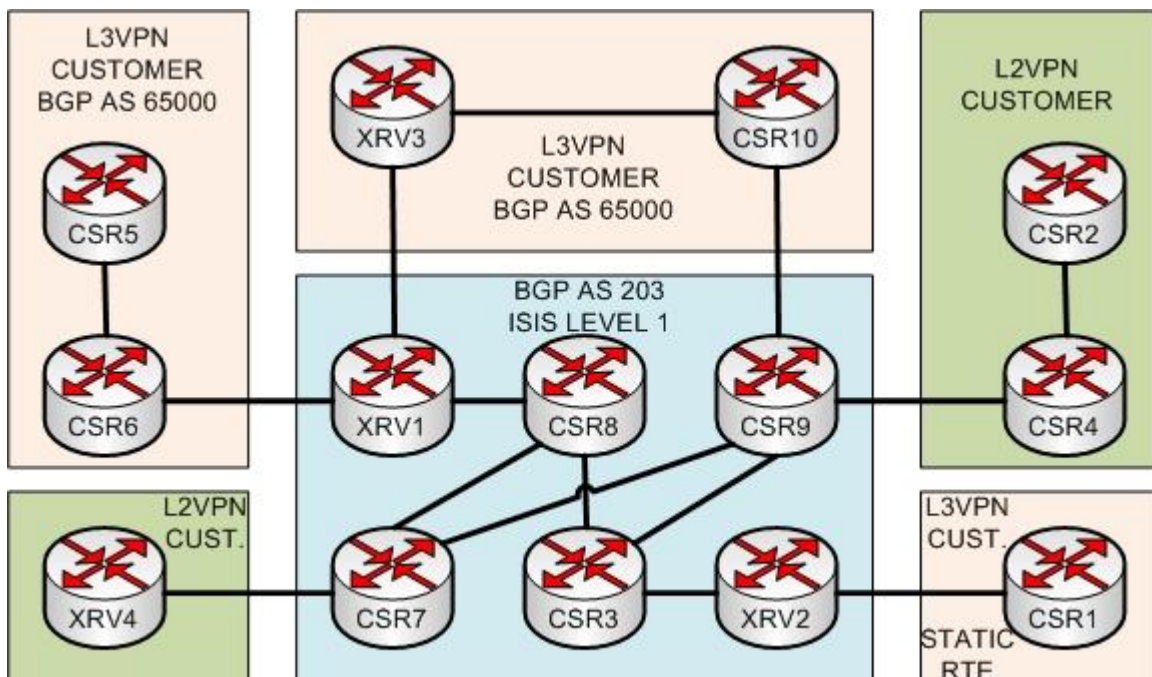
Characteristic	NTP	PTP
Accuracy	1 ms – 10 ms	100 ns – 100 us
Server (master) type	Hardware or software	Hardware only
Sync method	Client pull from server	Server push to client
In-path adjustments	None	Yes, via transparent blocks
Cost	Inexpensive	Expensive
Monitoring metrics	Minimal	Many

43.1 Network Time Protocol (NTP)

NTP is the oldest network timing protocol and is used in many networks to synchronize time to millisecond accuracy using IPv4/v6 as transport. It uses UDP port 123 and comes in 5 versions:

1. Version 0: Introduced round-trip delay and offset calculations, 10-100 ms accuracy.
2. Version 1: Formalized client-server and peer-to-peer models
3. Version 2: Introduced authentication
4. Version 3: Introduced algorithms to select the “best” server when multiple clocks disagreed, along with broadcast mode. Introduced NTP control messages as well.
5. Version 4: Introduced more accurate clock and various other improvements. This is the current Cisco default and can achieve 1-10 ms accuracy, ten times better than the original versions. It is backwards compatible as far as NTPv2; some Internet time sources still use NTPv2.

NTP uses “strata” to measure the accuracy of a clock, with stratum 0 being the best (most accurate) and stratum 16 being unreachable. Each network hop away from stratum N is assumed to be stratum N + 1. Cisco routers can be made NTP masters, but this is not recommended unless there is a direct attachment to a stratum 0 clocking device (not a network connection). NTP can operate in many modes (symmetric active/passive, client-service, broadcast, multicast, etc) and supports both IPv4 and IPv6. The network diagram is shown below and shows an SP core providing L3VPN and L2VPN services to various customers.



The SP core will use a combination of client-server and symmetric-active modes. CSR3 and CSR8 are the NTP masters for the core, with CSR8 having a lower stratum and thus being preferred. CSR8 supports IPv4 only while CSR3 is dual stack. Because authentication is such a simple topic in NTP, it is included

during the initial configuration. Of note, **NTP servers authenticate to clients**, which seems backwards when compared to other protocols. The logic is that the clients must be certain that they are receiving the correct time; that is to say, they should not accept time from untrusted sources. The “ntp authenticate” command tells the router (only needed on clients or symmetric-active peers) to actually authenticate a server/peer. The key must be defined, trusted, and applied to a server/peer to be effective. The default stratum is 8, which is used by CSR3. CSR3 uses the “prefer” keyword, which is meaningless in this context, and is used as a “tie breaker” when there are multiple NTP servers/peers. All other things being equal, such as strata, time quality, and other measurements, the “prefer” option selects this server/peer over the others.

```
! CSR3
ntp authentication-key 203 md5 NTP_CORE
ntp authenticate
ntp trusted-key 203
ntp master
ntp peer 203.0.0.8 key 203 prefer source Loopback0

! CSR8
ntp logging
ntp authentication-key 203 md5 NTP_CORE
ntp authenticate
ntp trusted-key 203
ntp master 4
ntp peer 203.0.0.3 key 203 source Loopback0
```

It may take tens of minutes for the local router to synchronize its clock, even as a master. This makes NTP very time-consuming to test. CSR3 and CSR8 should both claim to be synchronized. Most of the information is extraneous (low-level details), so filtering based on the synchronization state is often sufficient. Both clocks have a local-host IP address of 127.127.1.1 which implies they are local masters.

```
R3#show ntp status
Clock is synchronized, stratum 8, reference is 127.127.1.1
nominal freq is 250.0000 Hz, actual freq is 250.0000 Hz, precision is 2**10
ntp uptime is 2181200 (1/100 of seconds), resolution is 4000
reference time is D9AC9B12.07AE1490 (03:29:22.030 UTC Wed Sep 23 2015)
clock offset is 0.0000 msec, root delay is 0.00 msec
root dispersion is 2.40 msec, peer dispersion is 1.20 msec
loopfilter state is 'CTRL' (Normal Controlled Loop), drift is 0.000000000 s/s
system poll interval is 16, last update was 15 sec ago.
```

```
R8#show ntp status | include sync
Clock is synchronized, stratum 4, reference is 127.127.1.1
```

For faster convergence, we can adjust the min and max poll timers. These are NTP “hello” interval boundaries that determine how frequently peers can be polled. The values are read in terms of 2^x where x is the input value. The default for min/max is 6/10, which yields a 64 second minpoll and 1024

second (17 minute) maxpoll. We can adjust these to be faster on both routers. Before the adjustment, we can confirm the default values. The minpoll must be less or equal to the maxpoll.

```
R8#show ntp associations detail | include stratum|minpoll
127.127.1.1 configured, ipv4, our_master, sane, valid, stratum 3
minpoll = 4, maxpoll = 4
203.0.0.3 configured, ipv4, authenticated, insane, invalid, stratum 8
minpoll = 6, maxpoll = 10
```

```
! CSR8
ntp peer 203.0.0.3 key 203 minpoll 4 maxpoll 5 source Loopback0
```

```
! CSR3
ntp peer 203.0.0.8 key 203 minpoll 4 maxpoll 5 source Loopback0
```

We can confirm the change using the same show command as above, and also use debugging to verify the packet exchange. We can see the packets are about ~18 seconds apart which is close to 16 (2^4).

```
R8#show ntp associations detail | include stratum|minpoll
127.127.1.1 configured, ipv4, our_master, sane, valid, stratum 3
minpoll = 4, maxpoll = 4
203.0.0.3 configured, ipv4, authenticated, insane, invalid, stratum 8
minpoll = 4, maxpoll = 5
```

```
! CSR8
03:40:47.202: NTP message sent to 203.0.0.3, from interface 'Loopback0'
(203.0.0.8).
03:40:52.032: NTP message received from 203.0.0.3 on interface 'Loopback0'
(203.0.0.8).
03:41:05.202: NTP message sent to 203.0.0.3, from interface 'Loopback0'
(203.0.0.8).
03:41:08.031: NTP message received from 203.0.0.3 on interface 'Loopback0'
(203.0.0.8).
03:41:23.203: NTP message sent to 203.0.0.3, from interface 'Loopback0'
(203.0.0.8).
03:41:25.032: NTP message received from 203.0.0.3 on interface 'Loopback0'
(203.0.0.8).
```

Looking more closely at the association details, the two servers successfully authenticate one another but fail to acknowledge the accuracy of one another's clock. Unfortunately, the debugs do not appear to offer any valuable information as the configuration appears correct. The insane/invalid message is typically shown when the offset is less than -1000 ms or greater than +1000 ms (so +/- 1 second). Clearly our offset is within the bounds at -2 ms, yet the two cannot synchronize. Looking at the detailed output below, we confirm the small offset and successful authentication. The "reach" variable is an octal representation of the last 8 packets received. The number is in octal, so 377 is 11111111 in binary. This means that all of the last 8 packets were successfully received (right to left). A value of 367 is 11110111

would indicate that the fourth newest packet was dropped, for example. We also see the poll intervals at 2^4 (16) for both the remote and local NTP speakers. The mode is symmetric-active since both routers have a “peer” statement for one another.

```
R3#show ntp associations detail
203.0.0.8 configured, ipv4, authenticated, insane, invalid, stratum 4
ref ID 127.127.1.1      , time D9ACA199.33F7CF68 (03:57:13.203 UTC Wed Sep 23
2015)
our mode active, peer mode active, our poll intvl 16, peer poll intvl 16
root delay 0.00 msec, root disp 2.18, reach 377, sync dist 8.81
delay 6.00 msec, offset -2.0000 msec, dispersion 2.58, jitter 0.97 msec
precision 2**10, version 4
assoc id 25175, assoc name 203.0.0.8
assoc in packets 1181, assoc out packets 1184, assoc error packets 17
org time 00000000.00000000 (00:00:00.000 UTC Mon Jan 1 1900)
rec time D9ACA19A.33F7CF68 (03:57:14.203 UTC Wed Sep 23 2015)
xmt time D9ACA19A.33F7CF68 (03:57:14.203 UTC Wed Sep 23 2015)
filtdelay =      7.00      7.00      6.00      6.00      6.00      6.00      7.00      7.00
filtoffset =     -1.50     -1.50     -2.00     -1.00     -2.00     -1.00     -1.50     -1.50
filtererror =      1.95      2.19      2.43      2.67      2.91      3.15      3.39      3.63
minpoll = 3, maxpoll = 5
```

The output is very similar on CSR8, showing no indication of failure with the offset, authentication, packet loss, or poll interval. The peers remain unsynchronized, though. This is likely related to two different masters trying to synchronize with slightly different times.

```
R8#show ntp associations detail
203.0.0.3 configured, ipv4, authenticated, insane, invalid, stratum 8
ref ID 127.127.1.1      , time D9ACA292.07AE1490 (04:01:22.030 UTC Wed Sep 23
2015)
our mode active, peer mode active, our poll intvl 16, peer poll intvl 16
root delay 0.00 msec, root disp 2.21, reach 377, sync dist 8.58
delay 5.00 msec, offset 0.5000 msec, dispersion 2.66, jitter 1.14 msec
precision 2**10, version 4
assoc id 53783, assoc name 203.0.0.3
assoc in packets 86, assoc out packets 89, assoc error packets 2
[snip]
```

We can test symmetric passive mode instead, which involves configuring the peer on only one side (passive router must be the lower stratum router, CSR8). We can see the peer is now dynamic from CSR3’s perspective and the local mode is passive. “Reach” indicates the past 3 packets were received, but the first 5 were not (00000111); technically since this is a new peer, only 3 packets have been exchanged, and this number will approach 377 over time. The symmetric mode feature, both active and passive, appears to be broken in this version of the CSR1000v between two masters.

! CSR3

```
no ntp peer 203.0.0.8
```

```
R3#show ntp associations detail
```

```
203.0.0.8 dynamic, ipv4, authenticated, insane, invalid, stratum 4  
ref ID 127.127.1.1 , time D9ACA659.33B64630 (04:17:29.202 UTC Wed Sep 23  
2015)  
our mode passive, peer mode active, our poll intvl 16, peer poll intvl 16  
root delay 0.00 msec, root disp 2.19, reach 7, sync dist 1945.52  
delay 6.00 msec, offset -2.0000 msec, dispersion 1939.32, jitter 1.00 msec
```

The remaining 4 SP routers will reference both NTP servers using IPv4 for CSR8 and IPv6 for CSR3. The XR configuration is very similar to XE except it goes into an NTP stanza. The update-calendar command is also available on IOS/XE hardware routers which tells NTP to update the hardware clock based on the software clock. Even though XRv clearly doesn't have a hardware clock, it still has a concept of a one, along with the appropriate command syntax. CSR does not.

```
RP/0/0/CPU0:XRv1#show calendar
```

```
Wed Sep 23 04:23:15 UTC 2015
```

```
RP/0/0/CPU0:XRv1#show clock
```

```
04:23:17.544 UTC Wed Sep 23 2015
```

```
! CSR7 and CSR9
```

```
ntp logging  
ntp authentication-key 203 md5 NTP_CORE  
ntp authenticate  
ntp trusted-key 203  
ntp server 203.0.0.8 key 203 maxpoll 5 source Loopback0  
ntp server ::203:0:0:3 key 203 maxpoll 5 source Loopback0
```

```
! XRv1 and XRv2
```

```
ntp  
 authentication-key 203 md5 NTP_CORE  
 authenticate  
 trusted-key 203  
 server 203.0.0.8 version 4 key 203 minpoll 4 maxpoll 5 source Loopback0  
 server ipv6 ::203:0:0:3 version 4 key 203 minpoll 4 maxpoll 5 source  
 Loopback0  
 update-calendar  
 log-internal-sync
```

More oddities from NTP are present when we verify CSR7 for synchronization. It has access to a stratum 4 and stratum 8 clock, yet it synchronizes to the worse stratum clock. It also displays a bogus IPv4 address which must be some representation of ::203:0:0:3, the IPv6 address of the current master. Clearly CSR8 has a been stratum but was not selected. The offsets are very close and no received packets were lost. It appears that having multiple, unsynchronized NTP masters in the network concurrently breaks many things.

```
R7#show ntp status | include sync
```

```
Clock is synchronized, stratum 9, reference is 177.221.165.15
```

```
R7#show ntp associations
```

address	ref clock	st	when	poll	reach	delay	offset	disp
+~203.0.0.8	127.127.1.1	4	10	32	377	2.000	-1.000	2.150
*~::203:0:0:3	127.127.1.1	8	4	32	377	4.000	0.000	2.117

* sys.peer, # selected, + candidate, - outlyer, x falseticker, ~ configured

Digging deeper, CSR7 seems to claim that CSR8 has association packet errors, and this might be the reason why it cannot be synchronized. Unlike CSR3, CSR7 actually considers CSR8 as sane/valid, which generally means it should be the master given the lower stratum. The debug does not seem to indicate anything extraordinary, either. It might be specific to NTP with IPv4 since the IPv6 features appear to work more reliably.

```
R7#show ntp associations detail | include stratum|assoc
```

```
203.0.0.8 configured, ipv4, authenticated, sane, valid, stratum 4
```

```
assoc id 6518, assoc name 203.0.0.8
```

```
assoc in packets 2965, assoc out packets 2965, assoc error packets 60
```

```
::203:0:0:3 configured, ipv6, authenticated, our_master, sane,valid,stratum 8
```

```
assoc id 6520, assoc name ::203:0:0:3
```

```
assoc in packets 1406, assoc out packets 1416, assoc error packets 0
```

```
R7#debug ntp all
```

```
04:32:04.726: NTP message received from 203.0.0.8 on interface 'Loopback0' (203.0.0.7).
```

```
04:32:04.726: NTP Core(DEBUG): ntp_receive: message received
```

```
04:32:04.726: NTP Core(DEBUG): ntp_receive: peer is 0x7F4015E78278, next action is 1.
```

Verifying the XRv is simple and appears to be working. CSR8 is correctly selected as the NTP master given the lower stratum of 4 (versus CSR3's stratum of 8). The outputs are almost identical to XE. We can clearly see that CSR8 is the current master, so there is no need to troubleshoot these associations. This reinforces the possibility of an issue with IPv4 NTP on XE routers.

```
RP/0/0/CPU0:XRv1#show ntp status | include sync
```

```
Clock is synchronized, stratum 5, reference is 203.0.0.8
```

```
RP/0/0/CPU0:XRv1#show ntp associations
```

address	ref clock	st	when	poll	reach	delay	offset	disp
+~::203:0:0:3	127.127.1.1	8	28	32	377	0.98	4.628	2.416
*~203.0.0.8	127.127.1.1	4	21	32	377	0.98	1.157	2.427

We can add additional security to NTP by applying access-lists. The lists are processed in the following order, from least restrictive to most restrictive:

1. Peer: Identifies NTP speakers who are allowed to update your clock
2. Serve: Identifies NTP speakers who can be updated by you, and can issue control queries
3. Serve-only: Identifies NTP speakers who can be updated by you, but cannot query
4. Query-only: Identifies NTP speakers who can query you, but not receive time

A deny entry placed in any ACL will automatically deny that level of privilege and everything beneath it. For example, an explicit “deny any” at the end of a “peer” ACL will deny all NTP to service to any NTP speaker not permitted in the peer ACL. The other three categories would also be denied as the ACLs are processed in the sequence outlined above. This feature is best illustrated with some examples. We will first secure our NTP masters, CSR3 and CSR8. CSR8 only has to worry about IPv4; the peer list specifies that only CSR3 is allowed to update CSR8’s clock. The serve list specifies that anyone with address 203.0.0.0 – 203.0.0.15 (loopbacks in the core) can receive time and query the master. CSR3’s ACLs follow the same logic but includes IPv6 also. Because no IPv6 peer list has been defined, no IPv6 NTP speaker is allowed to update CSR3’s clock, which makes sense since the only symmetric peer is IPv4-only (CSR8).

```
! CSR8
ip access-list standard ACL_NTP_PEER
  permit 203.0.0.3

ip access-list standard ACL_NTP_SERVE
  permit 203.0.0.0 0.0.0.15

ntp access-group peer ACL_NTP_PEER
ntp access-group serve ACL_NTP_SERVE

! CSR3
ip access-list standard ACL_NTP_PEER
  permit 203.0.0.8

ip access-list standard ACL_NTP_SERVE
  permit 203.0.0.0 0.0.0.15

ipv6 access-list ACL_NTP_SERVE_IPV6
  permit ipv6 ::203:0:0:0/120 host ::203:0:0:3

ntp access-group peer ACL_NTP_PEER
ntp access-group serve ACL_NTP_SERVE
ntp access-group ipv6 serve ACL_NTP_SERVE_IPV6
```

There isn’t a good way to verify this configuration is working other than checking the ACL counters. On CSR3, we can see counters on all three lists. Currently, CSR3 has no IPv4 clients, but at one time it did, which is why there are 421 matches compared to the thousands of matches in other categories.


```
R3#show access-lists | section NTP
Standard IP access list ACL_NTP_PEER
  10 permit 203.0.0.8 (1653 matches)
Standard IP access list ACL_NTP_SERVE
  10 permit 203.0.0.0, wildcard bits 0.0.0.15 (421 matches)
IPv6 access list ACL_NTP_SERVE_IPV6
  permit ipv6 ::203:0:0:0/120 host ::203:0:0:3 (6609 matches) sequence 10
```

When a packet is denied, NTP logging can help reveal it for troubleshooting. To illustrate the case above, CSR3's IPv6 serve list is augmented with a "deny" statement for XRv1 as a client. While this won't affect communication to CSR8 or other clients, it breaks all XRv1's ability to receive time.

```
! CSR3
ipv6 access-list ACL_NTP_SERVE_IPV6
  deny ipv6 host ::203:0:0:11 host ::203:0:0:3
  permit ipv6 ::203:0:0:0/120 host ::203:0:0:3

R3#show access-list ACL_NTP_SERVE_IPV6
IPv6 access list ACL_NTP_SERVE_IPV6
  deny ipv6 host ::203:0:0:11 host ::203:0:0:3 (1 match) sequence 5
  permit ipv6 ::203:0:0:0/120 host ::203:0:0:3 (6763 matches) sequence 10

! CSR3
NTP Core (NOTICE): ntp_receive: dropping message: restricted..
```

We can also secure the clients using similar logic. Since the clients should never serve time to anyone, a "deny any" list can be used to ensure the serve, serve-only, and query-only functions are totally disabled. The only "peers" who can update the client clocks are CSR3 and CSR8. Since we have two functions (peer and serve) using two routed protocols (IPv4 and IPv6), we need 4 access-lists. **Pitfall: the NTP ACLs on XR can never process the destination addresses and these MUST be "any".** Specifying the loopback or any other local address causes the ACL entry to be unmatched, resulting in NTP synchronization failures. The XE IPv4 ACL is standard, so the issue is moot, but the XE IPv6 ACL does work with destination addresses as demonstrated below. Technically, there is no need for a serve list, since the implicit deny from the peer-ACL will ensure that the serve, serve-only, and query-function functionality is disabled. Having the explicit ACL can be useful for counting ACL entry hits, which is valuable for troubleshooting. The minimalist approach would only require two "peer" lists. The "log" option is also supported for NTP for additional detail, but is not included in this configuration.

```
! CSR7 and CSR9
ip access-list standard ACL_NTP_PEER
  permit 203.0.0.8

ip access-list standard ACL_NTP_SERVE
  deny any

ipv6 access-list ACL_NTP_PEER_IPV6
```

```

permit ipv6 host ::203:0:0:3 host ::203:0:0:7 ! (or :9 for CSR9)

ipv6 access-list ACL_NTP_SERVE_IPV6
deny ipv6 any any

ntp access-group peer ACL_NTP_PEER
ntp access-group serve ACL_NTP_SERVE
ntp access-group ipv6 peer ACL_NTP_PEER_IPV6
ntp access-group ipv6 serve ACL_NTP_SERVE_IPV6

! XRv1 and XRv2
ipv4 access-list ACL_NTP_PEER
10 permit ipv4 host 203.0.0.8 any

ipv4 access-list ACL_NTP_SERVE
10 deny ipv4 any any

ipv6 access-list ACL_NTP_PEER_IPV6
10 permit ipv6 host ::203:0:0:3 any

ipv6 access-list ACL_NTP_SERVE_IPV6
10 deny ipv6 any any

ntp
access-group ipv4 peer ACL_NTP_PEER
access-group ipv4 serve ACL_NTP_SERVE
access-group ipv6 peer ACL_NTP_PEER_IPV6
access-group ipv6 serve ACL_NTP_SERVE_IPV6

```

We can confirm packet counters for the permit statements with none for the deny statements, indicating a proper configuration. These clients are not allow to serve time or control messages to anyone, but can be updated from the two NTP masters as specified in the ACL.

```

R9#show access-lists
Standard IP access list ACL_NTP_PEER
 10 permit 203.0.0.8 (2588 matches)
Standard IP access list ACL_NTP_SERVE
 10 deny any
IPv6 access list ACL_NTP_PEER_IPV6
 permit ipv6 host ::203:0:0:3 host ::203:0:0:9 (2588 matches) sequence 10
IPv6 access list ACL_NTP_SERVE_IPV6
 deny ipv6 any any sequence 10

RP/0/0/CPU0:XRv2#show access-list ipv4
ipv4 access-list ACL_NTP_PEER
 10 permit ipv4 host 203.0.0.8 any (11 matches)
ipv4 access-list ACL_NTP_SERVE
 10 deny ipv4 any any

```

```
RP/0/0/CPU0:XRv2#show access-list ipv6
ipv6 access-list ACL_NTP_PEER_IPV6
 10 permit ipv6 host ::203:0:0:3 any (10 matches)
ipv6 access-list ACL_NTP_SERVE_IPV6
 10 deny ipv6 any any
```

NTP has many other minor features associated with it. One of them is the concept of an “orphan”. An NTP orphan scenario occurs when all NTP servers are unreachable. Ideally, synchronizing to UTC time is desirable, but when this is impossible, synchronizing to something common between nodes in an NTP “mesh” is the next best thing. Cisco recommends enabling this command on all servers and clients, and ensuring clients can reach all of the orphan servers. XR does not appear to support this feature, so we will enable it on the XE core routers. Cisco also recommends taking the maximum stratum in the network and adding 2 to it, which yields 11 in our case. The orphan server will not be enabled until the current stratum is worse than the orphan value (if sync is lost, the stratum will be 16, which is always worse than the orphan value).

```
! CSR3, CSR7, CSR8, CSR9
ntp orphan 11
```

We can test this by temporarily disabling “ntp master” on CSR3 and CSR8 (not shown). The clocks on these routers remain synchronized, but do so at stratum 11 and use a new reference, 127.0.0.1.

```
R8#show ntp status | include sync
Clock is synchronized, stratum 11, reference is 127.0.0.1
```

Interestingly, we see that orphan mode appears to work well between CSR3 and CSR8, likely because there isn’t a competing set of master clocks. CSR3 synchronizes to CSR8 as the orphan master, and symmetric passive mode is still in effect. Notice that the stratum is 12 on CSR3, which is one number greater than the orphan stratum identified on CSR8.

```
R3#show ntp status | include sync
Clock is synchronized, stratum 12, reference is 203.0.0.8
```

```
R3#show ntp associations detail | include stratum|our_mode
203.0.0.8 dynamic, ipv4, authenticated, our_master, sane, valid, stratum 11
our mode passive, peer mode active, our poll intvl 64, peer poll intvl 32
```

CSR9, as a client, has similar output. Because the clocks are still skewing, CSR9 thinks that CSR8 has the wrong time and selects CSR3, despite the clocks being about the same. Regardless, we can see that orphan mode is “working”. Routers autonomously select orphan servers, so I suppose it is possible that CSR9 selected CSR3 over CSR8 coincidentally. The details of this selection aren’t very clear (or relevant).

```
R9#show ntp associations detail | include stratum|our_mode
```

```
203.0.0.8 configured, ipv4, authenticated, insane, invalid, stratum 11
our mode client, peer mode server, our poll intvl 32, peer poll intvl 32
```

```
::203:0:0:3 configured, ipv6, authenticated, our_master, sane, valid, stratum
12
our mode client, peer mode server, our poll intvl 32, peer poll intvl 32
```

Another minor option is the NTP panic update. This alerts the user when the configured and received times are more than 1000 seconds (~16 minutes) apart.

```
! CSR7, CSR9
ntp panic update
```

We set CSR3's clock to be a totally wrong time; CSR9 shows a log message saying that the peer's clock is "way too off". The values in the first message are 0 but I believe this is a cosmetic choice as Cisco states it should show the actual values. Oddly, CSR9 still synchronizes to CSR3, and I assume this will change over time. It calls CSR8 insane/invalid despite a .5 ms offset, and calls CSR3 sane/valid despite thousands of seconds of offset.

```
R3#clock set 00:00:00 26 sep 2015
```

```
! CSR9
NTP Core (INFO): ::203:0:0:3 F68A 8A sys_peer
NTP Core (ERROR): Time correction of 0. seconds exceeds sanity limit of 0.
seconds.
Set clock manually to the correct UTC time.
NTP Core (ERROR): Peer's clock is way too off. Local clock not updated
```

```
R9#show ntp associations detail | include stratum|offset
203.0.0.8 configured, ipv4, authenticated, insane, invalid, stratum 11
delay 3.00 msec, offset 0.5000 msec, dispersion 2.04, jitter 0.97 msec
filtoffset = 0.00 0.50 0.50 0.00 0.00 0.50 0.50 0.00
::203:0:0:3 configured, ipv6, authenticated, our_master, sane, valid, stratum
11
delay 1.00 msec, offset 36612020.5000 msec, dispersion 2.03, jitter 0.97 msec
filtoffset = 3661202 3661202 3661202 3661202 3661202 3661202 3661202 3661202
```

Before continuing, we enable NTP master on CSR3/CSR8 again to disable orphan mode. More options include the burst and iburst features that can be applied to an NTP server or peer and allow NTP to synchronize faster. These are enabled by default in newer code releases. Burst will send 8 packets, as opposed to 2, to a peer that is newly reachable. This is done to converge in a few minutes rather than tens of minutes. The other option, iburst, performs identically except does so for the "initial" burst. This happens when a peer becomes unreachable and, as a client, we want to discover it. Both are considered inappropriate when communicating to Internet NTP servers as it can appear like an attack, yet I am not sure how to disable it in Cisco IOS. The "normal-sync" option only applies to symmetric-mode (peer) and doesn't even show up in the configuration when applied. We won't evaluate this any further.

```
R8(config)#ntp server 1.1.1.1 ?
burst      Send a burst when peer is reachable (Default)
iburst     Send a burst when peer is unreachable (Default)
```

We can also specify the maximum and minimum distance that adjust how synchronization occurs. “Maxdistance” is misleading; with an input range of 1 – 16, one would assume this is the “worst possible stratum” value so that NTP peers don’t accidentally synchronize with terrible clocks. The command does something entirely different; it sets the number of packets required for synchronization. The default used to be 1, but is now 8. Increasing this value means that more packets are required for synchronization to occur. This increases network load but potentially increases reliability. We will use a more liberal value of 4 so that synchronization occurs faster but potentially less accurately.

```
! CSR7 and CSR9
ntp maxdistance 4
```

Mindistance, however, is not well documented and is configured in the range from 1 – 500 with unspecified units. We won’t configure it; non-Cisco documentation indicates that it is the sum of the offset and jitter, measured in seconds, but the command is absent from the XE 3S command reference.

```
R7(config)#ntp mindistance ?
<1-500>    Minimum distance to consider for clockhop
```

Another interesting command is the “passive” option. Cisco documentation suggests that it is used to accept passive associations. No explicit mention is made about symmetric-passive, and since CSR3 is a symmetric passive peer of CSR8 (for IPv6) and did not require this command, I am unsure of its purpose. The context-sensitive help does not provide any assistance either. It is not configured anywhere and is shown below for reference.

```
R3(config)#ntp ?
passive          NTP passive mode
```

NTP can also limit to number of associations. This is valuable in many scenarios. For CSR7 and CSR9, they should have exactly two associations, none of which are dynamic, so using max = 2 would protect them from learning a dynamic peer. For peers that only have dynamic associations (symmetric passive, broadcast, multicast), setting an upper-bound can conserve router resources, ultimately increasing security. The command is also supported on XR, so we configure it on XRv1 and XRv2.

```
! CSR7 and CSR9
ntp max-associations 2
```

```
! XRv1 and XRv2
ntp
max-associations 2
```

After completing all of the NTP configuration within the core, we can quickly verify that all of the routers still have time synchronization before continuing. We can also check the packet statistics to make sure our security mechanisms did not cause any errors. We will check the NTP masters for packet statistics as they have the most NTP traffic; we see no errors. **Pitfall: After a reboot, XRv NTP appears to never synchronize despite the clock skewing towards accurate time. You must delete all NTP configuration (no ntp), then reapply. It will synchronize almost immediately.**

```
R3#show ntp status | include sync
Clock is synchronized, stratum 8, reference is 127.127.1.1
```

```
R7#show ntp status | include sync
Clock is synchronized, stratum 5, reference is 203.0.0.8
```

```
R8#show ntp status | include sync
Clock is synchronized, stratum 4, reference is 127.127.1.1
```

```
R9#show ntp status | include sync
Clock is synchronized, stratum 5, reference is 203.0.0.8
```

```
RP/0/0/CPU0:XRv1#show ntp status | include sync
Clock is synchronized, stratum 5, reference is 203.0.0.8
```

```
RP/0/0/CPU0:XRv2#show ntp status | include sync
Clock is synchronized, stratum 5, reference is 203.0.0.8
```

```
R3#show ntp packets
Ntp In packets           : 25584
Ntp Out packets          : 25584
Ntp bad version packets  : 0
Ntp protocol error packets : 0
```

```
R8#show ntp packets
Ntp In packets           : 25581
Ntp Out packets          : 25588
Ntp bad version packets  : 0
Ntp protocol error packets : 0
```

Next, we will examine NTP multicast delivery. CSR1 is the NTP server for the customer L3VPN and is serving time to CSR5, CSR6, CSR10, and XRv3. CSR1 has both an IPv4 and IPv6 multicast NTP service available. The global NTP configuration is nothing new. We must configure NTP as a multicast sender on the PE-CE link so that XRv2 can register the source with the C-RP. We configure both an IPv4 and IPv6 multicast stream and re-use authentication keys for simplicity.

```
! CSR1
ntp logging
ntp authentication-key 65000 md5 NTP_MCAST
```

```

ntp trusted-key 65000
ntp master

interface GigabitEthernet2.512
 ntp multicast FF08::1:2:3 key 65000
 ntp multicast 225.1.2.3 key 65000

```

We also must configure receivers. Since NTP is using an IGMP membership report behind the scenes, we must enable PIM on the NTP client interface. It also makes sense to configure this on loopbacks for routers that have multiple transit links like CSR10 and XRv3. CSR5 is the only router requesting IPv4-only timing service.

```

! All three CSR clients
ip multicast-routing distributed

ntp logging
ntp authentication-key 65000 md5 NTP_MCAST
ntp authenticate
ntp trusted-key 65000

! CSR5
interface GigabitEthernet2.556
 ip pim sparse-mode
 ntp multicast client 225.1.2.3

! CSR6 and CSR10
ipv6 multicast-routing

interface Loopback0
 ip pim sparse-mode
 ntp multicast client 225.1.2.3
 ntp multicast client FF08::1:2:3

! XRv3
multicast-routing
 address-family ipv4
  interface all enable

 address-family ipv6
  interface all enable

ntp
 interface Loopback0
  multicast client 225.1.2.3
 authentication-key 65000 md5 NTP_MCAST
 authenticate
 trusted-key 65000
 update-calendar

```

```
log-internal-sync
```

We can see that CSR6 sees membership reports from CSR5 and CSR6 (itself). Note that NTP normally uses group 224.0.1.1 but I demonstrate a new group for variety. CSR6 has the correct (*,G) entry with the RP as XRv3 (configuration not shown) with both the local loopback and CSR5 in the OIL. The (S,G) entry, once CSR1 starts sending, is also accounted for.

```
R6#show ip igmp groups 225.1.2.3
IGMP Connected Group Membership
Group Address      Interface                Uptime    Expires    Last Reporter
Group Accounted
225.1.2.3          GigabitEthernet2.556    20:08:57  00:02:32   10.5.6.5
225.1.2.3          Loopback0                20:09:11  00:02:53   10.6.6.6
```

```
R6#show ip mroute 225.1.2.3 | begin \
(*, 225.1.2.3), 20:11:01/stopped, RP 10.13.13.13, flags: SJCL
Incoming interface: GigabitEthernet2.561, RPF nbr 10.6.11.11
Outgoing interface list:
Loopback0, Forward/Sparse, 20:10:59/00:02:01
GigabitEthernet2.556, Forward/Sparse, 20:10:46/00:02:44

(10.1.12.1, 225.1.2.3), 19:51:09/00:02:09, flags: LJT
Incoming interface: GigabitEthernet2.561, RPF nbr 10.6.11.11
Outgoing interface list:
GigabitEthernet2.556, Forward/Sparse, 19:51:09/00:02:44
Loopback0, Forward/Sparse, 19:51:09/00:02:01
```

We can check the flow of traffic by checking the packet counts against the (S,G) entry.

```
R6#show ip mroute 225.1.2.3 10.1.12.1 count | begin ^Group
Group: 225.1.2.3, Source count: 1, Packets forwarded: 1090, Packets received:
1090
Source: 10.1.12.1/32, Forwarding: 1089/0/114/0, Other: 1089/0/0
```

Checking CSR5, we can see that NTP is receiving packets and is synchronized to CSR1.

```
R5#show ntp packets
Ntp In packets          : 1104
Ntp Out packets         : 10
Ntp bad version packets : 0
Ntp protocol error packets : 0
```

```
R5#show ntp status | include sync
Clock is synchronized, stratum 9, reference is 10.1.12.1
```


Note that CSR6 does not sync to the IPv4 address but rather uses the IPv6 address. Debugging reveals that CSR6 doesn't think it is an IPv4 NTP client when it clearly is. **It does not appear supported to use IPv4 multicast on a loopback for NTP client (check the release notes for your version).** This works fine on XRV and IPv6 works fine on both platforms. Notice that the debug also shows a successfully processed IPv6 multicast packet. We already saw that CSR6 was locally joining the IPv4 multicast group, so we also prove that it is locally joining the IPv6 group as well.

```
R6#show ipv6 mld groups ff08::1:2:3
MLD Connected Group Membership
Group Address                Interface      Uptime    Expires
FF08::1:2:3                  Loopback0     00:52:06  never
```

```
R6#debug ntp packet
NTP IPv4 multicast message discarded: not an NTP multicast client for
225.1.2.3
NTP message received from FD00:10:1:12::1 on interface 'Loopback0',
(FF08::1:2:3), table 0.
```

```
NTP Core(DEBUG): ntp_receive: message received
NTP Core(DEBUG): ntp_receive: peer is 0x00000000, next action is 6.
NTP IPv6 message sent to FD00:10:1:12::1, from FE80::21E:BDFF:FE69:6200,
table = 0, interface Loopback0.
```

```
R6#show ntp status | include sync
Clock is synchronized, stratum 9, reference is 65.190.62.223
```

After awhile, CSR6 will eventually show that the IPv6 peer is reachable and the clock will synchronize as shown above. Also note that the local mode is "xcast client", which I assume means multicast OR broadcast client. The peer FD00:10:1:12::1 is the NTP server, as expected. The detailed output also shows the correct (not weird IPv4-ish) IPv6 address of the dynamically-discovered NTP server.

```
! CSR6
NTP Core (INFO): FD00:10:1:12::1 6024 84 reachable
NTP Core (INFO): 0.0.0.0 C02C 0C clock_step
NTP Core (INFO): FD00:10:1:12::1 763A 8A sys_peer
NTP Core (NOTICE): Clock is synchronized.
```

```
R6#show ntp associations detail | include stratum|our_mode
FD00:10:1:12::1 dynamic, ipv6, authenticated, our_master, sane, valid,
stratum 8
our mode xcast client, peer mode server, our poll intvl 64, peer poll intvl
64
```

Likewise, CSR10 has the same problem as CSR6 and will not accept IPv4 multicast on the loopback. Again, it shows the IPv6-translated version of the IPv4 address.

```
R10#show ntp status | include sync
Clock is synchronized, stratum 9, reference is 65.190.62.223
```

```
R10#show ntp packets
Ntp In packets : 277
Ntp Out packets : 3
Ntp bad version packets : 0
Ntp protocol error packets : 0
```

XRv3 is not an IPv6 client since it actually does work with IPv4 on the loopback. Below we can see the IPv4 NTP server used, along with the association details. I am not sure what “our mode control” means, but we know XRv3 is functioning as an NTP IPv4 multicast client.

```
RP/0/0/CPU0:XRv3#show ntp status | include sync
Clock is synchronized, stratum 9, reference is 10.1.12.1
```

```
RP/0/0/CPU0:XRv3#show ntp associations detail | utility egrep 'stratum|^our'
10.1.12.1 dynamic, authenticated, our_master, stratum 8
our mode control, peer mode bdcast, our poll intvl 1024, peer poll intvl 1024
```

Last, we will examine broadcast distribution using NTP. This is simpler than multicast because it cannot be routed but can touch all hosts on a LAN. It is also limited to a single broadcast domain, so we will use CSR4 as the master for the L2VPN customer. For extra security, we use different authentication keys for CSR2 and XRv4. We can also specify the directed broadcast address if we choose versus 255.255.255.255 which is the default. The client configuration is similar to multicast except there is no need to specify a group.

```
! CSR4
ntp logging
ntp authentication-key 24 md5 NTP_BCAST
ntp authentication-key 414 md5 NTP_L2VPN
ntp trusted-key 414
ntp trusted-key 24
ntp master

interface GigabitEthernet2.524
 ntp broadcast key 24 destination 10.2.4.255

interface GigabitEthernet2.549
 ntp broadcast key 414

! CSR2
ntp logging
ntp authentication-key 24 md5 NTP_BCAST
ntp authenticate
ntp trusted-key 24
```

```

interface GigabitEthernet2.524
 ntp broadcast client

! XRv4
ntp
 interface GigabitEthernet0/0/0/0.574
  broadcast client

 authentication-key 414 md5 NTP_L2VPN
 authenticate
 trusted-key 414
 log-internal-sync

```

Since CSR4 is sourcing these broadcasts from its LAN interfaces towards both peers, the NTP server IP address is different on each LAN, but they all reference CSR4.

```

R2#show ntp status | include sync
Clock is synchronized, stratum 9, reference is 10.2.4.4

RP/0/0/CPU0:XRv4#show ntp status | include sync
Clock is synchronized, stratum 9, reference is 10.4.14.4

```

The details indicate that the mode is “xcast client” where CSR4 is a server. Because this “xcast” string is shown for multicast and broadcast, I assume that the NTP underpinning don’t differentiate, and only higher layer protocols like IGMP, MLD, and PIM really make the difference. Like symmetric passive and multicast clients, the association was dynamic since “10.2.4.4” does not exist in CSR2’s configuration anywhere. The output is similar for XRv except that it says the local mode is “control”, which is a little confusing.

```

R2#show ntp associations detail | include stratum|our_mode
10.2.4.4 dynamic, ipv4, authenticated, our_master, sane, valid, stratum 8
our mode xcast client, peer mode xcast server, our poll intvl 64, peer poll
intvl 64

RP/0/0/CPU0:XRv4#show ntp associations detail | utility egrep 'stratum|^our'
10.4.14.4 dynamic, authenticated, our_master, stratum 8
our mode control, peer mode bdcast, our poll intvl 1024, peer poll intvl 1024

```

The CSR1000v also supports Simple NTP (SNTP). This is very similar to NTP from a network perspective; it supports unicast, multicast, and broadcast transport, uses UDP port 123, and can be authenticated. CSR1000v does not support SNTP ACLs or multicast/broadcast server, only client. SNTP uses simpler, less CPU intensive time algorithms (simple time stepping) and is meant to be used on lower-end platforms or small devices. SNTP is also a good fit for simple applications where accuracy and reliability are not terribly important. It is recommended to only be used at the extremities of the network and certainly not in the core of the NTP distribution tree. XR does not support this feature at all. Rather than

extensively test it, the available commands are shown below, and there is no support for interface level commands. SNTP and NTP are mutually exclusive on a router and cannot be configured concurrently.

```
R4(config)#sntp ?
authenticate      Authenticate time sources
authentication-key Authentication key for trusted time sources
broadcast         Configure SNTP broadcast services
logging           Enable SNTP message logging
multicast         Configure SNTP multicast services
server            Configure SNTP server
source-interface  Configure interface for source address
trusted-key       Key numbers for trusted time sources
```

Additional Reading – Reference configurations "ntp"

43.2 1588v2 (Precision Time Protocol(PTP))

Precision Time Protocol (PTP), also called IEEE 1588-2008 or IEEE 1588v2, is used to convey time and frequency information across an IP network. It was created because existing mechanisms, such as NTP and GPS, were insufficient for all needs. Putting a GPS receiver at every node is unaffordable and sometimes not physically possible, and NTP cannot achieve accuracy beyond milliseconds in many networks. PTP defines a master-slave relationship for timing distribution much like NTP. Clocks are identified as “ordinary” or “boundary”. Ordinary clocks are typically endpoints with a single network connection (host device) which can be a master or slave clock. A boundary clock has multiple network connections and synchronizes time across network segments (router). Both ordinary and boundary locks can be identified as master clocks. Boundary clocks are like NTP clocks that are nodes in a tree where masters are upstream and slaves are downstream. There is a concept of grandmaster as well, where the grandmaster is the root timing reference from which timing is derived. IEEE 1588-2008 introduced the concept of a transparent clock which modifies PTP messages transiting it to account for network latency time for packets in transit. This helps the downstream clocks to have more accurate timing; NTP has no mechanism for adjusting transit NTP packets.

PTP uses several different messages to synchronize time. Messages are categorized as “event” or “general” messages. Events are time-sensitive as the timestamps directly affect the timing of the PTP nodes (sync, delay, etc). General messages are important to PTP, but their time sensitivity is insignificant (announce, management, signaling, etc).

- a. **Sync, delay_req, follow up, delay_resp:** Core synchronization messages to communicate time related information between PTP clocks.
- b. **Announce:** Used by the best master clock (BMC) algorithm to select the best clock. It can build a clock hierarchy based on “quality” and select the grandmaster clock.
- c. **Management:** Used to monitor, configure, and maintain PTP clocks.
- d. **Signaling:** Used for non-time-critical communications between clocks.

The BMC algorithm takes many components into account.

- a. **Identifier:** Unique identifier for a clock. This is based on the MAC address of the device.
- b. **Quality:** This is based on expected timing deviation and other advanced math.
- c. **Priority:** Administratively assigned precedence to select a grandmaster, which is two 8-bit priority fields. The first priority is the big “hammer” which is evaluated first. The second priority is meant to be a tie-breaker if all other things are equal. That is to say, if all time-related evaluations are the same, this priority can select the grandmaster. If this priority is also a tie, the MAC address is used to select the grandmaster as a last resort.
- d. **Variance:** Clock’s estimate of its own stability based on the existing PTP reference.

The original specification used multicast addresses for IPv4 and IPv6 PTP messaging, but a unicast negotiation option was added later. The packets are UDP encapsulated using port 319 for events and port 320 for general messages. PTP clocks can use unicast or unicast negotiation modes, in addition to multicast. The difference between the unicast options is that unicast mode requires the master to have all of the slaves configured so that it can send “sync” messages to them. Unicast negotiation mode requires slaves to register with the masters by sending negotiation messages, which scales better and is like the classic NTP design (server need not explicitly identify all clients).

The simplest PTP architecture would consist of a single grandmaster clock and several ordinary slave clocks. The ordinary slave clocks would directly reference time from the grandmaster. This would be a “tree” architecture which is simple but has a single point of failure. PTP can also be partially meshed so that other master clocks can peer with one another, like the NTP symmetric mode options. This is more resilient but care must be taken to ensure the grandmaster is properly identified. Sample configurations from Cisco are shown below; notice that the unicast negotiation master does not specify a clock destination since the slave will negotiate its own address as a clock destination. Also note that boundary clocks are only supported in unicast negotiation mode.

```
!! Unicast Negotiation Mode
```

```
! Master Clock
ptp clock ordinary domain 1
tod R0 ntp
input 1pps R0
clock-port MASTER master
transport ipv4 unicast interface loopback 0 negotiation
```

```
! Slave clock
ptp clock ordinary domain 1
tod R0 ntp
output 1pps R0
clock-port SLAVE slave
transport ipv4 unicast interface loopback 0 negotiation
clock source 10.1.1.1
```

```

! Boundary clock
ptp clock boundary domain 1
clock-port SLAVE slave
transport ipv4 unicast interface loopback 0 negotiation
clock source 10.1.1.1
clock-port MASTER master
transport ipv4 unicast interface loopback 1 negotiation

!! Unicast Mode

! Master Clock
ptp clock ordinary domain 1
tod R0 ntp
input 1pps R0
clock-port MASTER master
transport ipv4 unicast interface loopback 0
clock destination 20.1.1.1

! Slave clock
ptp clock ordinary domain 1
tod R0 ntp
output 1pps R0
clock-port SLAVE slave
transport ipv4 unicast interface loopback 0
clock source 10.1.1.1

```

43.3 Synchronous Ethernet (SyncE)

Synchronous Ethernet allows transferring clock signals over the Ethernet physical layer. Ethernet interfaces that support SyncE should also be capable of generating their own synchronization if the reference is lost (internal timing). The SyncE signal is filtered/regenerated at layer 1 and is expected to achieve clock accuracies of +/- 4.6 parts per million (ppm). The ppm measurement is commonly used for low-level timing details since no crystal is perfect. After 1,000,000 units of time, SyncE Ethernet equipment clocks (EEC) will be at 999,995.4 through 1,000,004.6. This is highly accurate.

SyncE encodes its information in the Ethernet Synchronization Message Channel (ESMC) within an Ethernet frame. The destination MAC of the frame is 0180.c200.0002 and uses ethertype 0x8809 (slow protocols) and deeper within the header lies the ESMC. Notice that Ethernet OAM also uses this Ethertype as it is not specific to SyncE, but any “slow” Ethernet protocols. Not counting the Ethernet encapsulation details, the ESMC is a 4 byte TLV byte that carries a Synchronization Status Message (SSM) code. This is a 4 bit field which carries various codes about the synchronization state of the SyncE network by carrying the quality level (QL). It is also designed to help stop synchronization loops.

SyncE can achieve timing in one of four ways:

1. **External:** The EEC obtains its timing from some external source like GPS. This is used in typical deployments.

2. **Line:** NE obtains time by deriving it from an input signal.
3. **Through:** The TX outputs from one interface are synchronized with the RX inputs of the opposite interface.
4. **Internal:** Typically used when a clock is in the holdover state after losing synchronization. It could also be present when there is no other clock available.

SyncE also supports a tree architecture where there is a single master clock that distributed time downstream to clients. There is a single point of failure but no possibility of a synchronization loop. SyncE can also be deployed in a ring, which provides some redundancy but introduces the risk of a synchronization loop. Last, it can be meshed, like PTP and NTP, but there could be potentially many loops despite the added redundancy. An example SyncE configuration from Cisco is shown below.

```
interface GigabitEthernet0/0/0
  synchronous mode
  clock source line
  clock wait-to-restore 720

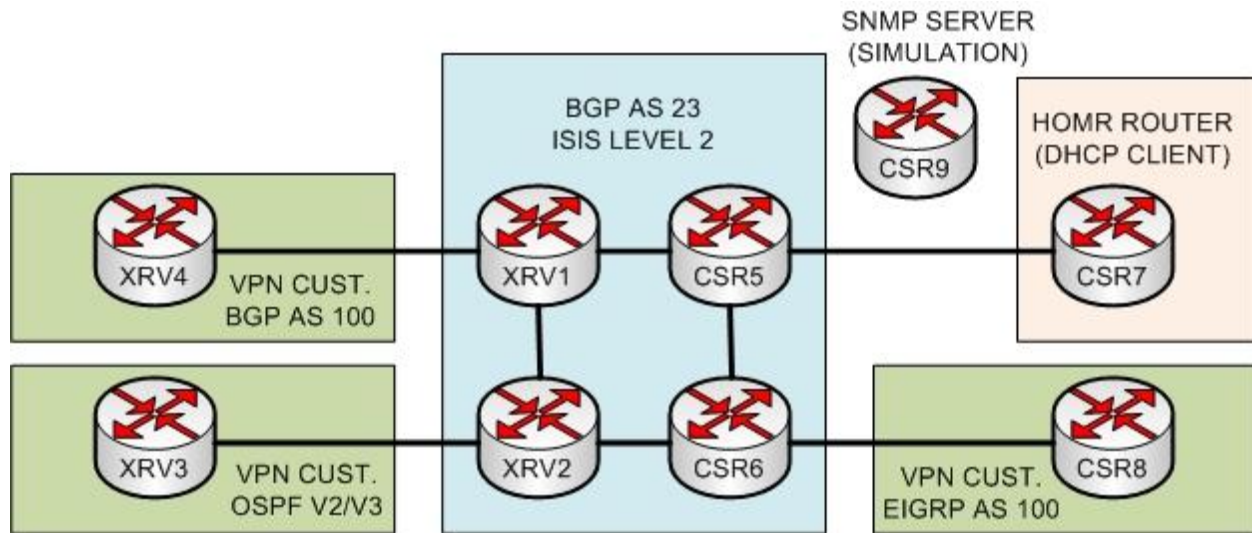
interface GigabitEthernet1/0/0
  synchronous mode
  clock source line

network-clock synchronization automatic
network-clock input-source 1 external 0/0/0 2m
network-clock input-source 2 external 1/0/0 2m

network-clock output-source line 1 interface GigabitEthernet0/0/0 external
0/0/0 2m
network-clock output-source line 1 interface GigabitEthernet1/0/0 external
1/0/0 2m
```

44. Describe, implement, and troubleshoot SNMP traps, RMON, EEM, and EPC

This section covers some traffic capturing and troubleshooting tools used by many kinds of networks. The network diagram used here is similar to that used in CPP/LPTS. The significant difference is that BGP has been removed from CSR5/CSR7 as CSR7 now gets a DHCP-issued address from CSR5. CSR7 represents a residential customer router for demonstrating advanced EEM. The OOBM laptop simulator, CSR9, now represents an SNMP server within the OOBM network.



Additional Reading – Reference configurations "cap-alarms"

44.1 SNMP traps

This section will focus heavily on SNMP traps and informs. SNMP polling is briefly examined in the security and IP SLA sections. The entire purpose of SNMP traps is to asynchronously notice a network management system (NMS) that some event has occurred. This is an unsolicited message that a device generates and sends without having received an SNMP poll. The fundamental difference between traps and informs is that informs must be acknowledged by the NMS. SNMPv2 introduced informs initially as this was not considered for the initial SNMPv1 rollout. We will first configure SNMPv2c (v2 with community string support) on XRV1 and CSR5. Note: Using the XE command "snmp-server enable traps" enables all traps the device supports. This is typically not appropriate; it is recommended to select the most important traps for your environment. Below is a reasonable example. We specify mostly traps that relate to MPLS L3VPN, including some IGP/BGP information. We can also specify which traps are sent to which servers; in this case, any IGP-related traps are sent to the IPv6 server, while BGP/LDP/MPLS traps are sent to the IPv4 server. Because the trap source is in a VRF, it is assumed that the destination route lookup on the SNMP trap/inform will be conducted in that VRF.

```
! CSR5
snmp-server trap-source GigabitEthernet1
snmp-server contact nrusso
snmp-server enable traps bgp state-changes
snmp-server enable traps isis
snmp-server enable traps mpls ldp session-up session-down
snmp-server enable traps mpls traffic-eng
snmp-server enable traps mpls vpn vrf-up vrf-down
snmp-server host FD00:10:31:5::209 vrf MGMT informs version 2c
  INFORM_COMM_STR isis
snmp-server host 10.31.5.209 vrf MGMT version 2c TRAP_COMM_STR bgp mpls-ldp
  mpls-traffic-eng mpls-vpn
```


We can confirm the configuration by viewing the hosts. The “host” in this context refers to the SNMP server, or NMS, that will be receiving the traps/informs.

```
R5#show snmp host
Notification host: FD00:10:31:5::209    udp-port: 162    VRFName: MGMT    type:
inform
user: INFORM_COMM_STR    security model: v2c

Notification host: 10.31.5.209    udp-port: 162    VRFName: MGMT    type: trap
user: TRAP_COMM_STR    security model: v2c
```

If we clear the IS-IS neighbors, CSR5 tries to send SNMP informs (which require acknowledgements) to the NMS using its IPv6 address. The requests are transmitted once every minute by default. The retry counter is 3 by default, resulting in 4 total informs when unacknowledged. Using “debug snmp packets”, we can see the packets originated by the device.

```
R5#clear isis *

R5#debug snmp packet
SNMP : Packet sent via UDP to FD00:10:31:5::209.162
SNMP: Inform request, reqid 1, errstat 0, erridx 0
  sysUpTime.0 = 36189655
  snmpTrapOID.0 = ciiAdjacencyChange
  ciiNotificationEntry.13 = 2
  ciiCircEntry.2 = 18
  ciiNotificationEntry.1 = 00 00 00 00 00 06 00 00
  ciiNotificationEntry.9 = 1
```

Clearing IS-IS again generates new inform requests, each of which expires after one minute. They are generated about 1 second apart.

```
R5#clear isis *
R5#show snmp pending
req id: 29, dest: 7FB3150DC7F0, V2C community: INFORM_COMM_STR, Expires in 56 secs
req id: 30, dest: 7FB3150DC7F0, V2C community: INFORM_COMM_STR, Expires in 56 secs
req id: 31, dest: 7FB3150DC7F0, V2C community: INFORM_COMM_STR, Expires in 57 secs
req id: 32, dest: 7FB3150DC7F0, V2C community: INFORM_COMM_STR, Expires in 58 secs
```

We can tune this more aggressively to ensure the router doesn’t keep sending the message minutes after an event has occurred. In this case, the router will send a single inform and wait 5 seconds for a reply.

```
! CSR5
snmp-server inform timeout 5
snmp-server inform retries 0
```

Clearing LDP neighbors has a similar affect, except these are traps sent to the IPv4 server. There are no retransmissions or acknowledgements for traps; they are “fire and forget” messages. Notice that this trap is more verbose/readable than the IS-IS one. We can even see the up/down messages and TCP session endpoints in the message body. If we check for any pending replies, there are none, as traps are not acknowledged. We also see the SNMP interface ID of 19, which is the link facing XRv11. This shows where the neighbor failed.

```
R5#clear mpls ldp neighbor 23.0.0.11
%LDP-5-CLEAR_NBRs: Clear LDP neighbors (23.0.0.11) by console
%LDP-5-NBRCHG: LDP Neighbor 23.0.0.11:0 (2) is DOWN (User cleared session
manually)

! CSR5
SNMP: Queuing packet to 10.31.5.209
SNMP: V2 Trap, reqid 20, errstat 0, erridx 0
  sysUpTime.0 = 36264940
  snmpTrapOID.0 = MPLS-LDP-MIB::mplsLdpSessionDown
  MPLS-LDP-MIB::mplsLdpSessionEntry.1.23.0.0.5.0.0.386206475.23.0.0.11.0.0 = 1
  MPLS-LDP-MIB::mplsLdpSessionEntry.5.23.0.0.5.0.0.386206475.23.0.0.11.0.0 =
36264940
  MPLS-LDP-MIB::mplsLdpSesStatsEntry.1.23.0.0.5.0.0.386206475.23.0.0.11.0.0 =0
  MPLS-LDP-MIB::mplsLdpSesStatsEntry.2.23.0.0.5.0.0.386206475.23.0.0.11.0.0 =0
  ifIndex.19 = 19
SNMP: Packet sent via UDP to 10.31.5.209

! Neighbor comes back up

%LDP-5-NBRCHG: LDP Neighbor 23.0.0.11:0 (1) is UP
SNMP: Queuing packet to 10.31.5.209
SNMP: V2 Trap, reqid 22, errstat 0, erridx 0
  sysUpTime.0 = 36265217
  snmpTrapOID.0 = MPLS-LDP-MIB::mplsLdpSessionUp
  MPLS-LDP-MIB::mplsLdpSessionEntry.1.23.0.0.5.0.0.386206475.23.0.0.11.0.0 = 5
  MPLS-LDP-MIB::mplsLdpSessionEntry.5.23.0.0.5.0.0.386206475.23.0.0.11.0.0 =
36265217
  MPLS-LDP-MIB::mplsLdpSesStatsEntry.1.23.0.0.5.0.0.386206475.23.0.0.11.0.0 =0
  MPLS-LDP-MIB::mplsLdpSesStatsEntry.2.23.0.0.5.0.0.386206475.23.0.0.11.0.0 =0
  ifIndex.19 = 19
SNMP: Packet sent via UDP to 10.31.5.209

R5#show snmp pending
[no output]

R5#show snmp mib ifmib ifindex | include 19
GigabitEthernet2.551: Ifindex = 19
```

The XR configuration is very similar. The downside is that there does not appear a way to associate certain traps/informs with a certain NMS, so all NMS' will get all configured traps/informs. In this case, the IPv4 host gets all the information as traps, including the IS-IS information that it didn't get on CSR5.

```
! XRv1
snmp-server vrf MGMT
  host 10.31.5.209 traps version 2 TRAP_COMM_STR
  host fd00:10:31:5::209 informs version 2 INFORM_COMM_STR

snmp-server contact nrusso
snmp-server trap-source MgmtEth0/0/CPU0/0
snmp-server inform pending 5
snmp-server inform retries 0
snmp-server traps isis adjacency-change
snmp-server traps bgp updown
snmp-server traps mpls traffic-eng cisco
snmp-server traps mpls ldp up
snmp-server traps mpls ldp down
snmp-server traps mpls l3vpn vrf-up
snmp-server traps mpls l3vpn vrf-down
```

For XRv1, we will do verification in a less destructive way. We can enable the ordinary SNMP packet debugging, as well as debugging for “test-trap” messages that we locally generate. XE also has the ability to use the “test” command for SNMP trap generation, but we demonstrate it on XR since the behavior is similar. We enable both debugs, then generate a trap to signal an LDP “session-down” event.

```
RP/0/0/CPU0:XRv1#debug snmp test-trap
RP/0/0/CPU0:XRv1#debug snmp packet
RP/0/0/CPU0:XRv1#snmp test trap routing mpls ldp session-down
```

First, the “test-trap” debugs show their output from the software processing of the test trap. Apparently this test command selected CSR5 as the neighbor to fake as being down for the purposes of reporting (probably because it had the least uptime since we flapped it earlier). The output is verbose but the message is clear.

```
! XRv1
RP/0/0/CPU0: test_trap[65713]: Debug init Done
RP/0/0/CPU0: test_trap[403]: Trap ID: 25
RP/0/0/CPU0: test_trap[403]: make_test_traps: input args 0x0 0x0 (null)
(null)
RP/0/0/CPU0:test_trap[403]: get_first_mpls_ldp_indices: Getting first LDP
session
RP/0/0/CPU0:test_trap[403]: get_first_mpls_ldp_indices: First MPLS LDP
entity_ldp_id=23.0.0.11.0.0, index=385875973 peer_ldp_id=23.0.0.5.0.0
RP/0/0/CPU0:test_trap[403]: make_test_trap_mpls_ldp: Formatting MPLS LDP Trap
for ent_ldp_id=23.0.0.11.0.0 index=385875973 peer_ldp_id=23.0.0.5.0.0
```

```

RP/0/0/CPU0:test_trap[403]: make_test_trap_mpls_ldp: local_id_addr=0x1700000B
local_id_inst=0x00000000 peer_id_addr =0x17000005 peer_id_inst =0x00000000
RP/0/0/CPU0:test_trap[403]: make_test_trap_mpls_ldp:
oid_str=1.3.6.1.2.1.10.166.4.1.3.3.1.2.23.0.0.11.0.0.385875973.23.0.0.5.0.0
len=67
RP/0/0/CPU0:test_trap[403]: make_test_trap_mpls_ldp: made OID
RP/0/0/CPU0:test_trap[403]: make_test_trap_mpls_ldp: got session_state
RP/0/0/CPU0:test_trap[403]: make_test_trap_mpls_ldp: got disc time
RP/0/0/CPU0:test_trap[403]: make_test_trap_mpls_ldp: got type_errors
RP/0/0/CPU0:test_trap[403]: make_test_trap_mpls_ldp: got tlv_errors
RP/0/0/CPU0:test_trap[403]: make_test_trap_mpls_ldp: Making varbinds for
state=5, disc_time=17460782 type_err=0 tlv_err=0
RP/0/0/CPU0:test_trap[403]: make_test_trap_mpls_ldp: Sending
mplsLdpSessionDown Trap

```

Shortly thereafter, the real SNMP packet debug shows that the packets were actually sent to the NMS' configured. We can see the trap-source being set properly, as we configured.

```

! XRv1
RP/0/0/CPU0:snmpd[1128]: t11 snmp_trap_addr_get:snmp trap-source configured
:fd00:10:31:5::211
RP/0/0/CPU0:snmpd[1128]: t11 snmp_send_pdu_udp_transport: Q-ing INFORM[sp
162, dp 41472]
RP/0/0/CPU0:snmpd[1128]: t11 snmp_trap_addr_get:snmp trap-source configured
:fd00:10:31:5::211
RP/0/0/CPU0:snmpd[1128]: t11 snmp_trap_addr_get:snmp trap-source configured
:10.31.5.211
RP/0/0/CPU0:snmpd[1128]: t11 snmp_send_pdu_udp_transport: Q-ing TRAP[sp 161,
dp 41472]
RP/0/0/CPU0:snmpd[1128]: t11 snmp_trap_addr_get:snmp trap-source configured
:10.31.5.211
RP/0/0/CPU0:snmpd[1128]: t11 SENT[I] 238 bytes to fd00:10:31:5::209:162 from
fd00:10:31:5::211:162, vrfid 0x60000003, tblid 0xe0800012
RP/0/0/CPU0:snmpd[1128]: t11 SENT[T] 236 bytes to 10.31.5.209:162 from
10.31.5.211:161, vrfid 0x60000003, tblid 0xe0000012

```

One oddity is that XRv reports the source-port of the inform is 162 and the source-port of the trap is 161, but we expected both to have dest-ports of 162. However, the end of the debug shows us that the ports were set properly at the last second. This is probably just a cosmetic debug issue or an internal software operation. Notice SENT[I], meaning inform, was sent to the IPv6 host, while SENT[T] (trap), was sent to the IPv4 host. This is the expected behavior.

SNMPv3 traps are very similar. This is a more secure model of SNMPv3 that is examined in detail, from a security perspective, in the security section. The only difference is that instead of assigning a community string to an SNMP server, you assign a user. CSR6 uses authentication only (no encryption) while XRv2 uses both. For brevity, we will only configure these routers to report link-up and link-down events, both of which can be “simulated” on both XE and XR.

```

! CSR6
snmp-server group AUTH_GROUP v3 auth
snmp-server contact nrusso
snmp-server enable traps snmp linkdown linkup
snmp-server host 10.31.5.209 vrf MGMT traps version 3 auth JLI
snmp-server host FD00:10:31:5::209 vrf MGMT informs version 3 auth JLI
! The following line does not show in the running-config
snmp-server user JLI AUTH_GROUP v3 auth sha AUTHPASS

! XRv2
snmp-server vrf MGMT
  host 10.31.5.209 traps version 3 priv NRUSSO
  host fd00:10:31:5::209 informs version 3 priv NRUSSO

snmp-server user NRUSSO PRIV_GROUP v3 auth sha encrypted 0227316F23362E127F
priv des56 encrypted 1335253B3D3C251918
snmp-server group PRIV_GROUP v3 priv
snmp-server traps snmp linkup
snmp-server traps snmp linkdown

```

First, we will conduct a basic verification of our users and groups on both routers. These commands are quick ways to ensure your users and groups are spelled correctly and have the correct attributes.

```

R6#show snmp group | begin AUTH_GROUP
groupname: AUTH_GROUP                security model:v3 auth
contextname: <no context specified>  storage-type: nonvolatile
readview : vldefault                 writeview: <no writeview
specified>
notifyview: *tv.00000001.00000000.00000000.0
row status: active

```

```

R6#show snmp user JLI
User name: JLI
Engine ID: 800000090300005056A96CBE
storage-type: nonvolatile          active
Authentication Protocol: SHA
Privacy Protocol: None
Group-name: AUTH_GROUP

```

```

RP/0/0/CPU0:XRv2#show snmp group
groupname: PRIV_GROUP                security model:usm
readview : vldefault                 writeview:
notifyview:
row status: active

```

```

RP/0/0/CPU0:XRv2#show snmp users
User Name: NRUSSO

```

```
EngineId: localSnmplD
storage-type: nonvolatile      active
```

Using the “test” command along with “debug snmp packet”, we will ensure CSR6 sends traps/informs to its configured NMS’. This time, both NMS’ should get notified of an event since I did not specify per=host traps/informs when configuring the servers. Oddly, SNMPv3 test informs don’t seem to work on XE. I only see traps being generated. For SNMPv2c, the “test traps” on XR also included informs.

```
R6#debug snmp packet
R6#test snmp trap snmp linkdown
Generating Link Down trap

! CSR6
SNMP: Queuing packet to 10.31.5.209
SNMP: V2 Trap, reqid 12, errstat 0, erridx 0
sysUpTime.0 = 29336782
snmpTrapOID.0 = snmpTraps.3
ifIndex.1 = 1
ifDescr.1 = GigabitEthernet1
ifType.1 = 6
lifEntry.20.1 = Link Down Trap
SNMP: Packet sent via UDP to 10.31.5.209
```

SNMPv3 does not appear to work on XRv; with traps or informs, I see neither one generated during the test. The OS acknowledges the trap but does not appear to put a packet on the wire.

```
RP/0/0/CPU0:XRv2#snmp test trap interface link-down
RP/0/0/CPU0:: test_trap[65713]: Debug init Done
RP/0/0/CPU0:: test_trap[403]: Trap ID: 4
RP/0/0/CPU0:: test_trap[403]: make_test_traps: input args 0x0 0x0 (null)
(null)
RP/0/0/CPU0:: test_trap[403]: Sending LinkDown Trap for ifIndex :1
```

44.2 Remote Monitor (RMON) in XE and logging correlation in XR

RMON is a simple way to generate alarms (log messages, traps, etc) when certain thresholds are crossed. Common uses for this are measuring CPU utilization, interface load, and any arbitrary SNMP MIB parameter. Some switches support RMON at the interface level for traffic measuring/discovery of sorts, but routers typically do not. RMON is comprised primarily of two components: events and alarms. An event is assigned a number determines “what to do” when an alarm is tripped. These events don’t actually measure anything, they are really more like “actions” than events. RMON alarms measure a specific SNMP variable against rising and/or falling threshold, and invoke an RMON event when either threshold is crossed.

First, we will define some events (which are like actions). These are not tied to any alarms yet, and we can reuse them across many alarms. Event 1 creates a syslog message and sends a trap to the

configured SNMP servers (SNMP must be configured already). The second event issues a log message and specifies a process owner and description. We can view the events with a simple command shown below. Don't worry about the timers/log entries yet.

```
! CSR6
rmon event 1 log trap JLI
rmon event 2 log description "EVENT 2 WAS CALLED" owner nrusso
```

```
R6#show rmon events
Event 1 is active, owned by nrusso
Description is
Event firing causes log and trap to community JLI,
last event fired at 0y0w5d,23:21:02,
Current uptime      0y0w5d,23:24:15
Current log entries:
  index  uptime              description
  1      0y0w5d,23:21:02
```

```
Event 2 is active, owned by nrusso
Description is EVENT 2 WAS CALLED
Event firing causes log,
last event fired at 0y0w5d,23:13:46,
Current uptime      0y0w5d,23:24:15
Current log entries:
  index  uptime              description
  1      0y0w5d,23:13:46      EVENT 2 WAS CALLED
```

Next, we will define some alarms. There are “alarms” and “hc-alarms”. The only difference is that HC (high capacity) alarms are used to measure 64 bit values, while regular alarms measure 32 bit values. Cisco states that interfaces less than 20 Mbps in speed must use 32 bit objects, interfaces between 20 Mbps and 650 Mbps can use either 32 or 64 bit objects, and interfaces greater than 650 Mbps must use 64 bit objects. In our case of having all Gigabit Ethernet interfaces, we “should” use 64 bit values but 32 bit values also seem to work. We will define an HC-alarm to measure input bytes on the interface towards CSR5. First, we must determine the interface by looking at the SNMP ifmib to see how interfaces have been enumerated by SNMP.

```
R6#show snmp mib ifmib ifindex | include 556
GigabitEthernet2.556-mpls layer: Ifindex = 19
GigabitEthernet2.556: Ifindex = 18
```

We will now configure SNMP to “remember” those interface assignments across reloads, otherwise RMON is worthless. When the router reloads, these SNMP ifmib indices may change unless we issue the command below.

```
! CSR6
snmp ifmib ifindex persist
```

Last, we create the RMON hc-alarm. There are several numerical components in the alarm highlighted in different colors for clarity. The first number is the alarm ID, which is an arbitrary number (yellow) used to reference the alarm. The number following the object to monitor is the sample rate in seconds; this is how often the device polls the counter to perform measurements (cyan). The “delta” option means that the operation should compare the new measurement against the previous one; “delta” is typically used to measure values that only move in one direction (things that only increase or decrease, NOT rates). The rising-threshold and falling-threshold values represent hexadecimal numbers whereby the alarm is triggered when the sampled data is above or below the configured threshold, respectively (pink). In this case, if the input bytes is more than 16384 (0x4000) bytes in a 10 second period, the rising-threshold is triggered. When it falls below 8192 (0x2000) bytes in a 10 second period, the falling-threshold is triggered. The “actions” taken when those thresholds are crossed are the RMON events defined earlier (green). We can tie in different actions when each threshold is crossed, as shown below.

```
! CSR6
rmon hc-alarm 10 ifHCInOctets.18 10 delta rising-threshold 04000 1 falling-
threshold 02000 2 owner nrusso
```

We can verify the configuration of the alarm using a simple command. Notice that it displays the thresholds in decimal, which is convenient for humans trying to troubleshoot RMON. Unfortunately, this show command doesn’t show you the HC-alarm ID (number 10 in yellow above), so if you had many alarms, you cannot easily differentiate them with this command.

```
R6#show rmon hc-alarms
Monitors ifHCInOctets.18 every 10 second(s)
Taking delta samples, last value was 3291
Rising threshold Low is 16384, Rising threshold Hi is 0,
    assigned to event 1
Falling threshold Low is 8192, Falling threshold Hi is 0,
    assigned to event 2
On startup enable rising or falling alarm
```

On CSR5, we will begin flooding pings to CSR6 to trip the rising-threshold (not shown). We can see the syslog message and subsequent SNMP trap (using “debug snmp packet”) when this event is triggered. Despite having only SNMPv3 configured, the trap is sent as SNMPv2c, and I cannot find a way to change this. It is also only sent to the IPv4 server, presumably because the IPv6 server is only sending informs, and perhaps the RMON process can only generate literal traps. We can see the hex value 0x4816 (decimal 18454) corresponds with the actual input bytes measured, which is greater than 0x4000, the rising-threshold.

```
! CSR6
%RMON-5-HCRISINGTRAP: Rising trap is generated because the value of
ifHCInOctets.18 exceeded the rising-threshold hc value 16384
```



```

SNMP: Queuing packet to 10.31.5.209
SNMP: V2 Trap, reqid 18, errstat 0, erridx 0
  sysUpTime.0 = 51606242
  snmpTrapOID.0 = hcAlarmMIB.2.0.1
  hcAlarmIndex.10 = 10
  hcAlarmVariable.10 = ifHCInOctets.18
  hcAlarmSampleType.10 = 2
  hcAlarmAbsValue.10 = 0x000004816
  hcAlarmRisingEventIndex.10 = 16384
  hcAlarmRisingThreshAbsValueLo.10 = 16384
  hcAlarmRisingThreshAbsValueHi.10 = 0
  hcAlarmRisingThresholdValStatus.1 = 2
SNMP: Packet sent via UDP to 10.31.5.209

```

When CSR5 stops bursting traffic, the RMON alarm will fall below the falling-threshold and trigger event 2. This only creates a syslog with no SNMP trap.

```

! CSR6
%RMON-5-HCFALLINGTRAP: Falling trap is generated because the value of
ifHCInOctets.18 has fallen below the falling-threshold hc value 8192

```

Every time a threshold is crossed and an event is referenced, RMON keeps track of when it was called and stores it in memory. We can see that information by looking at the RMON events again. I have tripped the threshold several times while testing, which is why there are so many log entries per event. Notice there is no description with event 1 log entries, but there is with event 2.

```

R6#show rmon events
Event 1 is active, owned by nrusso
Description is
Event firing causes log and trap to community JLI,
last event fired at 0y0w5d,23:39:11,
Current uptime      0y0w5d,23:41:05
Current log entries:
  index  uptime              description
  1      0y0w5d,23:21:02
  2      0y0w5d,23:39:11
Event 2 is active, owned by nrusso
Description is EVENT 2 WAS CALLED
Event firing causes log,
last event fired at 0y0w5d,23:40:11,
Current uptime      0y0w5d,23:41:05
Current log entries:
  index  uptime              description
  1      0y0w5d,23:13:46      EVENT 2 WAS CALLED
  2      0y0w5d,23:18:53      EVENT 2 WAS CALLED
  3      0y0w5d,23:23:21      EVENT 2 WAS CALLED
  4      0y0w5d,23:40:11      EVENT 2 WAS CALLED

```

The next example shows measuring bandwidth on the same interface (Gig2.556). This might be valuable if there is some NMS or automation system that changes interface bandwidth as a way to adjust QoS shapers or IGP convergence semi-dynamically. Since the SNMP parameters are not always obvious, one would have to know in advance what the OID is for a given parameter, or reference the documentation. We will reuse RMON events 1 and 2 just to show that it can be done, but new events can be customized per RMON alarm. Notice this is a regular RMON alarm since the HC counters are typically limited to interfaces (in/out bytes, packets, etc). The alarm ID is 20 (yellow) and since we are sampling a rate (a value that doesn't always increase or decrease), we use the absolute method. The rising-threshold is 800 Mbps and falling-threshold is 700 Mbps, where crossing either one fires event 1 or 2, respectively.

```
! CSR6
rmon alarm 20 ifSpeed.18 10 absolute rising-threshold 800000000 1 falling-
threshold 700000000 2
```

We can verify the alarm was configured properly by looking at RMON alarms. Unlike HC-alarms, we can see the alarm-ID, which is valuable. Notice that failing to specify an owner defaults to using "config" as opposed to a specific user (grey).

```
R6#show rmon alarms
Total number of alarm trap failures 0
Alarm 20 is active, owned by config
Monitors ifSpeed.18 every 10 second(s)
Taking absolute samples, last value was 1000000000
Rising threshold is 800000000, assigned to event 1
Falling threshold is 700000000, assigned to event 2
On startup enable rising or falling alarm
```

When we configure this alarm, RMON evaluates the current state of the interface and sees its speed is 1 Gbps. This crosses the rising threshold, so event 1 is fired. This creates a syslog message and SNMP trap.

```
! CSR6
%RMON-5-RISINGTRAP: Rising threshold has been crossed because the value of
ifSpeed.18 exceeded the rising-threshold value 800000000
SNMP: Queuing packet to 10.31.5.209
SNMP: V2 Trap, reqid 24, errstat 0, erridx 0
sysUpTime.0 = 51898167
snmpTrapOID.0 = rmon.0.1
alarmEntry.1.20 = 20
alarmEntry.3.20 = ifSpeed.18
alarmEntry.4.20 = 1
alarmEntry.5.20 = 1000000000
alarmEntry.7.20 = 800000000
SNMP: Packet sent via UDP to 10.31.5.209
```

When we change the bandwidth to a lower value, when RMON sees this again (based on sample interval), it will trigger the falling-threshold event. Only the syslog message is generated.

```
! CSR6
interface GigabitEthernet2.556
 bandwidth 123456
```

```
%RMON-5-FALLINGTRAP: Falling threshold has been crossed because the value of
ifSpeed.18 has fallen below the falling-threshold value 700000000
```

```
R6#show rmon alarms
Total number of alarm trap failures 0
Alarm 20 is active, owned by config
Monitors ifSpeed.18 every 10 second(s)
Taking absolute samples, last value was 123456000
Rising threshold is 800000000, assigned to event 1
Falling threshold is 700000000, assigned to event 2
On startup enable rising or falling alarm
```

Because events are shared across multiple alarms, any alarm triggered that event gets added to the event log. This can complicate troubleshooting as events and alarms are not mapped 1:1. For example, on event 2, index 4, this was the last time we tested the HC-alarm with packet counters. Several minutes later, we began testing the bandwidth measurement, which added new entries to the same log.

```
R6#show rmon events | begin Event_2
Event 2 is active, owned by nrusso
Description is EVENT 2 WAS CALLED
Event firing causes log,
last event fired at 0y0w6d,00:11:40,
Current uptime      0y0w6d,00:14:25
Current log entries:
  index  uptime                description
  1      0y0w5d,23:13:46      EVENT 2 WAS CALLED
  2      0y0w5d,23:18:53      EVENT 2 WAS CALLED
  3      0y0w5d,23:23:21      EVENT 2 WAS CALLED
  4      0y0w5d,23:40:11      EVENT 2 WAS CALLED
  5      0y0w6d,00:00:36      EVENT 2 WAS CALLED
  6      0y0w6d,00:02:27      EVENT 2 WAS CALLED
  7      0y0w6d,00:02:37      EVENT 2 WAS CALLED
  8      0y0w6d,00:10:51      EVENT 2 WAS CALLED
  9      0y0w6d,00:11:40      EVENT 2 WAS CALLED
```

XR logging correlation is a sizable topic with many fancy options. It is sometimes referred to as a subcomponent of the IOS XR Alarm Management and Debugging Event System (ALDEMS). We will examine the basic configuration and perform detailed verification; this is the closest thing to RMON I can find within XR. The general idea is to correlate logging messages that are related to the same event. For example, if the interface between XRv1 and CSR5 fails, this will trigger a link-down message. It will

also trigger an IS-IS neighbor down, LDP neighbor down, and possibly BGP neighbor down. We probably don't need to see all four of these messages, since the root-cause of the issue was the interface failing, which allows administrators to focus troubleshooting efforts. We know it is not a problem specific to IGP, LDP, or BGP. To simulate this, I will first configure basic remote logging on XRv1 to send messages to CSR9. I also enable logging for IS-IS, LDP, and the link in question, as these are disabled by default.

```
! XRv1
logging 10.31.5.209 vrf MGMT severity info

interface GigabitEthernet0/0/0/0.551
  logging events link-status

mpls ldp
  log
  neighbor

router isis 23
  log adjacency changes
  log pdu drops
```

Next, I will shut down the subinterface towards CSR5. Notice that the IS-IS and LDP neighbors go down immediately, but the BGP peer stays up because of an alternate IGP path via XRv2. This may not always be the case (for example, XRv2's link to CSR6 is down), but we can still identify the potential BGP failure as a by-product of the link failure. In this example, we assume the BGP failure was also related to this link-down event.

```
! XRv1
ifmgr[225]: %PKT_INFRA-LINK-5-CHANGED : Interface GigabitEthernet0/0/0/0.551,
changed state to Administratively Down

isis[1010]: %ROUTING-ISIS-5-ADJCHANGE : Adjacency to R5
(GigabitEthernet0/0/0/0.551) (L2) Down, Interface state down

mpls_ldp[1048]: %ROUTING-LDP-5-NBR_CHANGE : VRF 'default' (0x60000000),
Neighbor 23.0.0.5:0 is DOWN (Interface state down)
```

As an example, the BGP looks like this. This is not part of the current test, but here for reference.

```
bgp[1052]: %ROUTING-BGP-5-ADJCHANGE : neighbor 23.0.0.5 Down - BGP
Notification sent, hold time expired (VRF: default) (AS: 23)
```

XRv1 is currently sending all of these logs to CSR9. CSR9 is using EPC inbound on its OOBM management interface. Looking at the packet dumps, we can see those three syslog messages arrive at CSR9: interface, IS-IS, and LDP failure messages.

```
R9#show monitor capture CAP buffer dump
```

26

```

0000: 005056A9 7AC20050 56A92DC6 08004500 .PV.z..PV.-...E.
0010: 00BE381A 00001E11 44340A1F 05D30A1F ..8.....D4.....
0020: 05D10202 020200AA 63B43C31 38393E33 .....c.<189>3
0030: 39333A20 52502F30 2F302F43 5055303A 93: RP/0/0/CPU0:
0040: 41756720 32322031 353A3230 3A31312E Aug 22 15:20:11.
0050: 34323220 3A206966 6D67725B 3232355D 422 : ifmgr[225]
0060: 3A202550 4B545F49 4E465241 2D4C494E : %PKT INFRA-LIN
0070: 4B2D352D 4348414E 47454420 3A20496E K-5-CHANGED : In
0080: 74657266 61636520 47696761 62697445 terface Gigabite
0090: 74686572 6E657430 2F302F30 2F302E35 thernet0/0/0/0.5
00A0: 35312C20 6368616E 67656420 73746174 51, changed stat
00B0: 6520746F 2041646D 696E6973 74726174 e to Administrat
00C0: 6976656C 7920446F 776E200A ively Down .

```

27

```

0000: 005056A9 7AC20050 56A92DC6 08004500 .PV.z..PV.-...E.
0010: 00BE391A 00001E11 43340A1F 05D30A1F ..9.....C4.....
0020: 05D10202 020200AA E7B83C31 38393E33 .....<189>3
0030: 39343A20 52502F30 2F302F43 5055303A 94: RP/0/0/CPU0:
0040: 41756720 32322031 353A3230 3A31312E Aug 22 15:20:11.
0050: 34323220 3A206973 69735B31 3031305D 422 : isis[1010]
0060: 3A202552 4F555449 4E472D49 5349532D : %ROUTING-ISIS-
0070: 352D4144 4A434841 4E474520 3A204164 5-ADJCHANGE : Ad
0080: 6A616365 6E637920 746F2052 35202847 jacency to R5 (G
0090: 69676162 69744574 6865726E 6574302F igabitEthernet0/
00A0: 302F302F 302E3535 31292028 4C322920 0/0/0.551) (L2)
00B0: 446F776E 2C20496E 74657266 61636520 Down, Interface
00C0: 73746174 6520646F 776E200A state down .

```

28

```

0000: 005056A9 7AC20050 56A92DC6 08004500 .PV.z..PV.-...E.
0010: 00C43A1A 00001E11 422E0A1F 05D30A1F ...:.....B.....
0020: 05D10202 020200B0 46F13C31 38393E33 .....F.<189>3
0030: 39353A20 52502F30 2F302F43 5055303A 95: RP/0/0/CPU0:
0040: 41756720 32322031 353A3230 3A31312E Aug 22 15:20:11.
0050: 34343220 3A206D70 6C735F6C 64705B31 442 : mpls_ldp[1
0060: 3034385D 3A202552 4F555449 4E472D4C 048]: %ROUTING-L
0070: 44502D35 2D4E4252 5F434841 4E474520 DP-5-NBR_CHANGE
0080: 3A205652 46202764 65666175 6C742720 : VRF 'default'
0090: 28307836 30303030 30303029 2C204E65 (0x60000000), Ne
00A0: 69676862 6F722032 332E302E 302E353A ighbor 23.0.0.5:
00B0: 30206973 20444F57 4E202849 6E746572 0 is DOWN (Inter
00C0: 66616365 20737461 74652064 6F776E29 face state down)
00D0: 200A .

```

Before continuing, we will no-shutdown this interface on XRv1 to bring the network back online (not shown).

```
RP/0/0/CPU0:XRv1#show interfaces gig0/0/0/0.551 brief
```

Intf Name	Intf State	LineP State	Encap Type	MTU (byte)	BW (Kbps)
Gi0/0/0/0.551	up	up	802.1Q	1518	1000000

It would be convenient if the administrator could correlate these logs as treat them like a single event. It is important to quantify what constitutes a single event; for example, an interface going down and the BGP peer failing 3 hours later are probably not related. For this reason, we specify a timeout (ms) whereby the counter begins when the first log message is seen. The first message does not have to be the root-cause (the interface down event), but naturally, it often is. When specifying the alarms, the format is CATEGORY/GROUP/CODE. The format of a syslog is always CATEGORY-GROUP-SEVERITY-CODE. For example, "MGBL-SYS-5-CONFIG_I" and "ROUTING-LDP-5-NBR_CHANGE" both have hyphens to separate the fields. The logging correlator does not look at the severity. The rule below shows all four messages broken up by CATEGORY/GROUP/CODE with spaces in between as the parser requires. The timeout is set to 300000 ms, or 300 seconds, or 5 minutes.

```
! XRv1
logging correlator rule LINK_DOWN type nonstateful
  timeout 300000
  rootcause PKT_INFRA LINK CHANGED
  nonrootcause
    alarm ROUTING BGP ADJCHANGE
    alarm ROUTING ISIS ADJCHANGE
    alarm ROUTING LDP NBR_CHANGE
```

We can verify the rule was configured properly using basic show commands. The root-cause has a "root" code type while the others are considered "leaf". The difference between non-stateful and stateful rules is as follows. Non-stateful rules prevent all leaf messages from going to the syslog server, at any point, assuming the correlation occurs (i.e., the root-cause is seen). The leaf messages only remain in the correlation buffers. Stateful rules allows leaf messages to go to the syslog process if the root-cause alarm clears, which is useful for detecting non-root-cause conditions even after the root-cause is no longer present.

```
! XRv1
RP/0/0/CPU0:XRv1#show logging correlator rule LINK_DOWN
Rule Name : LINK_DOWN
Type      : Non Stateful
Source    : User
Timeout   : 300000          Rule State: RULE_APPLIED_ALL
Rootcause Timeout : None
Context Correlation : disabled
Reissue Non Bistate : N/A
Reparent  : N/A
Alarms :
```

Code	Type:	Category	Group	Message
	Root:	PKT_INFRA	LINK	CHANGED
	Leaf:	ROUTING	BGP	ADJCHANGE
	Leaf:	ROUTING	ISIS	ADJCHANGE
	Leaf:	ROUTING	LDP	NBR_CHANGE

```

Apply Locations:      None
Apply Contexts:      None

```

Number of buffered alarms : 0

These rules can be applied directly to the router or subcomponents therein. In XRv, it doesn't make sense to do anything other than "all-of-router", so we do that. We can also put the rule inside of a ruleset to achieve greater scalability when making multiple rules for a single location. Although we only have one rule (no benefit to rulesets), I use this configuration method anyway. More rules could be added to "LINK_RULESET" later.

```

! XRv1
logging correlator ruleset LINK_RULESET
  rulename LINK_DOWN

logging correlator apply ruleset LINK_RULESET
  all-of-router

```

We can verify that the ruleset has been applied, and also quickly see how many messages are in the correlation buffer. Presently, it is empty, as we expect.

```

RP/0/0/CPU0:XRv1#show logging correlator ruleset LINK_RULESET
Rule Set Name :   LINK_RULESET
Rules: LINK_DOWN                               : Applied

RP/0/0/CPU0:XRv1#show logging correlator info
Buffer-Size      Percentage-Occupied
655360           0.00

```

Next, we shut down the link again. Notice that only one message is sent to the syslog server. We only see the root cause; the leaf messages are never sent to the syslog process internal to XRv1 for nonstateful rules (assuming root cause is seen) and are instead stored in the correlation buffer. Since this is the only log message sent to syslog, we verify on CSR9 that no other logs were received. CSR9 is not explicitly told that IS-IS and LDP also failed as those are non-root-cause events.

```

! XRv1
ifmgr[225]: %PKT_INFRA-LINK-5-CHANGED : Interface GigabitEthernet0/0/0/0.551,
changed state to Administratively Down

R9#show monitor capture CAP buffer dump

```

43

```

0000: 005056A9 7AC20050 56A92DC6 08004500 .PV.z...PV.-...E.
0010: 00BE491A 00001E11 33340A1F 05D30A1F ..I.....34.....
0020: 05D10202 020200AA 64A53C31 38393E34 .....d.<189>4
0030: 31303A20 52502F30 2F302F43 5055303A 10: RP/0/0/CPU0:
0040: 41756720 32322031 353A3437 3A34312E Aug 22 15:47:41.
0050: 32393920 3A206966 6D67725B 3232355D 299 : ifmgr[225]
0060: 3A202550 4B545F49 4E465241 2D4C494E : %PKT INFRA-LIN
0070: 4B2D352D 4348414E 47454420 3A20496E K-5-CHANGED : In
0080: 74657266 61636520 47696761 62697445 terface GigabitE
0090: 74686572 6E657430 2F302F30 2F302E35 thernet0/0/0/0.5
00A0: 35312C20 6368616E 67656420 73746174 51, changed stat
00B0: 6520746F 2041646D 696E6973 74726174 e to Administrat
00C0: 6976656C 7920446F 776E200A ively Down .

```

As a preliminary check, we can check the correlation buffer utilization again to ensure the “Percentage-Occupied” is not 0.0 anymore. We should have some information present.

```

RP/0/0/CPU0:XRv1#show logging correlator info
Buffer-Size      Percentage-Occupied
655360           0.43

```

There are several ways to check the logs. First, we can look for logging correlations specific to a rule. The output is difficult to read, but the first field is the correlation ID, followed by the sub-log number. The CID is 1, and there are sub logs 1, 2, and 3. We can see 1.1 is the root cause while 1.2 and 1.3 are leaf events. These are all correlated as part of CID 1.

```

RP/0/0/CPU0:XRv1#show logging correlator buffer rule-name LINK_DOWN
#C_id.id:Rule Name:Source :Context:      Time      : Text
RP/0/0/CPU0:XRv1#show logging correlator buffer rule-name LINK_DOWN
#1.1 :LINK_DOWN:RP/0/0/CPU0:              :Aug 22 15:47:41.299
2015:ifmgr[225]: %PKT_INFRA-LINK-5-CHANGED : Interface GigabitEthernet0/0/0/0.551,
changed state to Administratively Down
#1.2 :LINK_DOWN:RP/0/0/CPU0:              :Aug 22 15:47:41.299
2015:isis[1010]: %ROUTING-ISIS-5-ADJCHANGE : Adjacency to R5
(GigabitEthernet0/0/0/0.551) (L2) Down, Interface state down
#1.3 :LINK_DOWN:RP/0/0/CPU0:              :Aug 22 15:47:41.319
2015:mpls_ldp[1048]: %ROUTING-LDP-5-NBR_CHANGE : VRF 'default' (0x60000000), Neighbor
23.0.0.5:0 is DOWN (Interface state down)

```

Assuming we didn’t know the correlation ID and wanted to find it without looking at the correlation buffer first, we can query the general logging events. Because the root cause was sent to the syslog process, syslog is aware of this one message along with its CID. Searching for string with “Administ” in them, we find a log at the bottom with a number following the colon. This is the CID.

```

RP/0/0/CPU0:XRv1#show logging events buffer all-in-buffer | include Administ
[snip]

```



```
#776      :1      :RP/0/0/CPU0:Aug 22 15:47:41.299 2015:ifmgr[225]: %PKT_INFRA-  
LINK-5-CHANGED : Interface GigabitEthernet0/0/0/0.551, changed state to  
Administratively Down
```

We can also query the correlation buffer based on the CID, if we know it. This output is identical to the output examined earlier, just referenced differently. The root cause is highlighted for clarity. The leaf messages will be in this buffer until manually purged, but CSR9 (syslog server) will never see them.

```
RP/0/0/CPU0:XRv1#show logging correlator buffer correlationID 1  
#C_id.id:Rule Name:Source :Context:                Time                : Text  
#1.1      :LINK_DOWN:RP/0/0/CPU0:                  :Aug 22 15:47:41.299  
2015:ifmgr[225]: %PKT_INFRA-LINK-5-CHANGED : Interface GigabitEthernet0/0/0/0.551,  
changed state to Administratively Down  
#1.2      :LINK_DOWN:RP/0/0/CPU0:                  :Aug 22 15:47:41.299  
2015:isis[1010]: %ROUTING-ISIS-5-ADJCHANGE : Adjacency to R5  
(GigabitEthernet0/0/0/0.551) (L2) Down, Interface state down  
#1.3      :LINK_DOWN:RP/0/0/CPU0:                  :Aug 22 15:47:41.319  
2015:mpls_ldp[1048]: %ROUTING-LDP-5-NBR_CHANGE : VRF 'default' (0x60000000), Neighbor  
23.0.0.5:0 is DOWN (Interface state down)
```

If we repeat the process quickly once more, we see only one syslog appears on XRv1's console. The remaining logs were buffered as we can quickly see by seeing the size of the buffer increase.

```
! XRv1  
:ifmgr[225]: %PKT_INFRA-LINK-5-CHANGED : Interface  
GigabitEthernet0/0/0/0.551, changed state to Administratively Down
```

```
RP/0/0/CPU0:XRv1#show logging correlator info  
Buffer-Size      Percentage-Occupied  
655360           0.86
```

We can determine the correlator ID by looking at the syslog buffer, and then check the correlation buffer for the entries within that CID. Over time, looking at the buffer by rulename will not scale since each event is a separation correlation (provided they are not inside of one another's timeout window).

```
RP/0/0/CPU0:XRv1#show logging events buffer last 50 | include PKT_INFRA-LINK  
#836      :3      :RP/0/0/CPU0:Aug 22 16:19:04.860 2015:ifmgr[225]: %PKT_INFRA-  
LINK-5-CHANGED : Interface GigabitEthernet0/0/0/0.551, changed state to  
Administratively Down
```

```
RP/0/0/CPU0:XRv1#show logging correlator buffer correlationID 3  
#C_id.id:Rule Name:Source :Context:                Time                : Text  
#3.1      :LINK_DOWN:RP/0/0/CPU0:                  :Aug 22 16:19:04.860  
2015:ifmgr[225]: %PKT_INFRA-LINK-5-CHANGED : Interface GigabitEthernet0/0/0/0.551,  
changed state to Administratively Down  
#3.2      :LINK_DOWN:RP/0/0/CPU0:                  :Aug 22 16:19:04.860  
2015:isis[1010]: %ROUTING-ISIS-5-ADJCHANGE : Adjacency to R5  
(GigabitEthernet0/0/0/0.551) (L2) Down, Interface state down
```

```
#3.3 :LINK_DOWN:RP/0/0/CPU0: :Aug 22 16:19:04.880
2015:mpls_ldp[1048]: %ROUTING-LDP-5-NBR_CHANGE : VRF 'default' (0x60000000), Neighbor
23.0.0.5:0 is DOWN (Interface state down)
```

When non-root-cause messages are caused by events outside the scope of the root cause message, they still get forced into the correlation buffer. During this “purgatory” period, the router waits for the specified timeout for the root-cause to occur. As an example, flapping CSR5’s interface to XRv1 a few times creates several IS-IS/LDP messages that are not related to an interface flap on XRv1’s side. To XRv1, these could be legitimate IS-IS/LDP issues since its interface is stable. The correlation buffer shows 6 logs, all correlated under CID 4, with no root cause. A quick way to tell is that the category for all of these rules is ROUTING, where the PKT_INFRA is the category of the root-cause for this particular correlation rule.

```
RP/0/0/CPU0:XRv1#show logging correlator buffer rule-name LINK_DOWN
[snip]
#4.1 :LINK_DOWN:RP/0/0/CPU0: :Aug 22 16:31:21.840
2015:isis[1010]: %ROUTING-ISIS-5-ADJCHANGE : Adjacency to R5
(GigabitEthernet0/0/0/0.551) (L2) Up, New adjacency
#4.2 :LINK_DOWN:RP/0/0/CPU0: :Aug 22 16:31:23.450
2015:mpls_ldp[1048]: %ROUTING-LDP-5-NBR_CHANGE : VRF 'default' (0x60000000), Neighbor
23.0.0.5:0 is UP (IPv4 connection)
#4.3 :LINK_DOWN:RP/0/0/CPU0: :Aug 22 16:33:22.671
2015:mpls_ldp[1048]: %ROUTING-LDP-5-NBR_CHANGE : VRF 'default' (0x60000000), Neighbor
23.0.0.5:0 is DOWN (Received Notification message from peer: Hold Timer Expired)
#4.4 :LINK_DOWN:RP/0/0/CPU0: :Aug 22 16:33:48.080
2015:isis[1010]: %ROUTING-ISIS-5-ADJCHANGE : Adjacency to R5
(GigabitEthernet0/0/0/0.551) (L2) Down, Holdtime expired
#4.5 :LINK_DOWN:RP/0/0/CPU0: :Aug 22 16:34:31.727
2015:isis[1010]: %ROUTING-ISIS-5-ADJCHANGE : Adjacency to R5
(GigabitEthernet0/0/0/0.551) (L2) Up, New adjacency
#4.6 :LINK_DOWN:RP/0/0/CPU0: :Aug 22 16:34:34.016
2015:mpls_ldp[1048]: %ROUTING-LDP-5-NBR_CHANGE : VRF 'default' (0x60000000), Neighbor
23.0.0.5:0 is UP (IPv4 connection)
```

Once the timeout expires (up to 5 minutes), all of these logs will be sent to the syslog process in order. Syslog directs them to the console and to the remote logging host, CSR9. They are all printed in quick succession, so in the event of LPTS (on a hardware router), ensure that any outbound syslog-related policers account for bursty syslog traffic towards the syslog server.

```
! XRv1
isis[1010]: %ROUTING-ISIS-5-ADJCHANGE : Adjacency to R5
(GigabitEthernet0/0/0/0.551) (L2) Up, New adjacency

mpls_ldp[1048]: %ROUTING-LDP-5-NBR_CHANGE : VRF 'default' (0x60000000),
Neighbor 23.0.0.5:0 is UP (IPv4 connection)

mpls_ldp[1048]: %ROUTING-LDP-5-NBR_CHANGE : VRF 'default' (0x60000000),
Neighbor 23.0.0.5:0 is DOWN (Received Notification message from peer: Hold
Timer Expired)
```

```
isis[1010]: %ROUTING-ISIS-5-ADJCHANGE : Adjacency to R5  
(GigabitEthernet0/0/0/0.551) (L2) Down, Holdtime expired
```

```
isis[1010]: %ROUTING-ISIS-5-ADJCHANGE : Adjacency to R5  
(GigabitEthernet0/0/0/0.551) (L2) Up, New adjacency
```

```
mpls_ldp[1048]: %ROUTING-LDP-5-NBR_CHANGE : VRF 'default' (0x60000000),  
Neighbor 23.0.0.5:0 is UP (IPv4 connection)
```

The entries are removed from the correlator buffer at this point. There was no real correlation since the root-cause was never seen in conjunction with the leaf messages.

```
RP/0/0/CPU0:XRv1#show logging correlator buffer correlationID 4  
NO records matching query found
```

If you need to clean out the correlator buffer to make extra space, you can do so selectively per CID or simply erase everything.

```
RP/0/0/CPU0:XRv1#clear logging correlator delete ?  
<0-4294967294> Correlation ID  
all-in-buffer Delete all messages
```

Alternatively, you can adjust the buffer size if space is running low. I do the opposite (to save memory) and shrink it significantly. As a result, the buffer is almost 70% full despite having only a few correlations.

```
RP/0/0/CPU0:XRv1(config)#logging correlator buffer-size 8192
```

```
RP/0/0/CPU0:XRv1#show logging correlator info  
Buffer-Size      Percentage-Occupied  
8192              69.14
```

44.3 Embedded Event Manager (EEM)

EEM is a lightweight, in-line scripting tool built into the XE and XR for automating certain tasks on the router. It can match a variety of inputs (CLI commands, syslog, counters, timers, IP SLA events, tracked objects, etc) and perform a variety of actions (CLI actions, syslog, counter increment/decrement, IP SLA probes, etc). We will look at some examples on XE only since XR does not have an applet feature; all of its scripts are actual TCL files which I assume is beyond the scope of CCIE SP.

The first example is on CSR5. The goal is to convince the network administrator to use “show running-config” less in an effort to learn more advanced commands. After using show run three times, a syslog message tells the user to use it less often. Two scripts are used; one counts the number of “show run” commands, and the other tests for that timer equaling 3, and then prints a log message. A note on EEM sync is warranted. With sync enabled, the EEM applet runs before the original command is executed. Sync disabled means the two run in parallel. The only time EEM applets need sync is when the command

being entered is going to generate output that the script cares about. When not using sync, you can optionally skip the command altogether, which means it never gets issued. In our case, whether the router increments the counter first or issues the “show run” command first is not relevant since the two happen at almost the exact same time. The second script tracks the counter equaling 3; when it does, the syslog message is issued and the timer is reset to 0.

```
! CSR5
event manager applet MATCH_SHOW_RUN
  event cli pattern "show run" sync no skip no
  action 020 counter name "SHOW_RUN_COUNTER" op inc value 1

event manager applet YELL_AT_USER
  event counter name SHOW_RUN_COUNTER entry-val 3 entry-op eq exit-val 0 exit-
op eq
  action 010 syslog priority critical msg "Stop using show run all the time"
  action 020 counter name "SHOW_RUN_COUNTER" op set value 0
```

We can test it by issuing “show run” three times. Notice that the EEM script expands the entire command, and the parser does too, so any short-hand command used is still matched.

```
R5#show run | include nothing
R5#show running | include nothing
R5#sh running-config | include nothing
%HA_EM-2-LOG: YELL_AT_USER: Stop using show run all the time
```

Using EEM debugs, we can see the counter increment when show run is issued. Currently, the counter is at 2. The next stanza shows it at 3, which triggers the entrance criteria based on our counter to create a syslog message. The script resets the counter to 0 before exiting.

```
R5#debug event manager policydir
EEM: policy_dir xml builtin: name:_cli_msg value:show running-config
fh_fd_counter_event_modify: re=0x7FB3151CE480
*** thresh_val = 3, exit_thresh_val 0
*** current_val = 2, monitored = 1

EEM: policy_dir xml builtin: name:_cli_msg value:show running-config
fh_fd_counter_event_modify: re=0x7FB3151CE480
*** thresh_val = 3, exit_thresh_val 0
*** current_val = 3, monitored = 0

EEM: policy_dir xml builtin: name:_counter_name value:SHOW_RUN_COUNTER
fh_fd_counter_event_modify: re=0x7FB3151CE480
*** thresh_val = 3, exit_thresh_val 0
*** current_val = 0, monitored = 1
```

The next set of scripts on CSR5 will check for a route to 23.0.0.12/32. If this exact route is not reachable, an SNMP trap and email will be sent to the NMS. This could be useful on a platform that is running a feature that the built-in SNMP traps don't natively support in the parser. There are many other EEM scripts to solve this, but I am making it complicated intentionally. The first script tracks an IP SLA to see if XRv2's loopback is responding. If it's not, the script does a "show ip route" and displays the output to the user. Then, it publishes an application event to a specific sub-system that a second script looks for as an event. The second script notifies the NMS of the fault. This basic software engineering hierarchy could allow many different "check route" scripts to invoke the same "notify" script rather than define the notification action themselves. Note that we also have to update our SNMP configuration to send EEM traps; we will send them to the IPv6 SNMP server only.

```
! CSR5
snmp-server enable traps event-manager
snmp-server host FD00:10:31:5::209 vrf MGMT informs version 2c
INFORM_COMM_STR isis event-manager

event manager applet CHECK_ROUTE_XRV12
  event track 1 state down
  action 005 cli command "enable"
  action 010 cli command "show ip route 23.0.0.12"
  action 020 puts "$_cli_result"
  action 030 publish-event sub-system 12 type 2012 arg1 "route lost"

event manager applet NOTIFY_NMS
  event application sub-system 12 type 2012
  action 010 snmp-trap strdata "LOST_ROUTE_XRV12"
  action 020 mail server "10.31.5.209" to "nrusso@ccie.sp" from "r5@ccie.sp"
  subject "LOST_ROUTE_XRV2" source-interface GigabitEthernet1 vrf MGMT

track 1 ip sla 1 reachability

ip sla 1
  icmp-echo 23.0.0.12 source-ip 23.0.0.5
  threshold 1000
  timeout 1000
  frequency 1
ip sla schedule 1 life forever start-time now
```

To trigger the event, I add a null route for XRv2's loopback to CSR5. With EEM debugs enabled, we can see most of the activity. First, EEM enters privileged-exec mode to issue the "show ip route" command. It then prints this string to the console. Normally, the output would be "Network not in table" but adding the null route returns a matching /32, as expected.

```
! CSR5
ip route 23.0.0.12 255.255.255.255 null0
```

```

R5#debug event manager action cli
%TRACK-6-STATE: 1 ip sla 1 reachability Up -> Down
%HA_EM-6-LOG: CHECK_ROUTE_XRV12 : DEBUG(cli_lib) : : CTL : cli_open called.
%HA_EM-6-LOG: CHECK_ROUTE_XRV12 : DEBUG(cli_lib) : : OUT : R5>
%HA_EM-6-LOG: CHECK_ROUTE_XRV12 : DEBUG(cli_lib) : : IN : R5>enable
%HA_EM-6-LOG: CHECK_ROUTE_XRV12 : DEBUG(cli_lib) : : OUT : R5#
%HA_EM-6-LOG: CHECK_ROUTE_XRV12 : DEBUG(cli_lib) : : IN : R5#show ip route
23.0.0.12
%HA_EM-6-LOG: CHECK_ROUTE_XRV12 : DEBUG(cli_lib) : : OUT : Routing entry for
23.0.0.12/32
%HA_EM-6-LOG: CHECK_ROUTE_XRV12 : DEBUG(cli_lib) : : OUT : Known via
"static", distance 1, metric 0 (connected)
%HA_EM-6-LOG: CHECK_ROUTE_XRV12 : DEBUG(cli_lib) : : OUT : Routing
Descriptor Blocks:
%HA_EM-6-LOG: CHECK_ROUTE_XRV12 : DEBUG(cli_lib) : : OUT : * directly
connected, via Null0
%HA_EM-6-LOG: CHECK_ROUTE_XRV12 : DEBUG(cli_lib) : : OUT : Route metric
is 0, traffic share count is 1
%HA_EM-6-LOG: CHECK_ROUTE_XRV12 : DEBUG(cli_lib) : : OUT : R5#
%HA_EM-6-LOG: CHECK_ROUTE_XRV12:
Routing entry for 23.0.0.12/32
Known via "static", distance 1, metric 0 (connected)
Routing Descriptor Blocks:
* directly connected, via Null0
Route metric is 0, traffic share count is 1
%HA_EM-6-LOG: CHECK_ROUTE_XRV12 : DEBUG(cli_lib) : : CTL : cli_close called.

```

Next, the SNMP inform is generated and sent to CSR9. Notice the string data is what we configured in the script.

```

! CSR5
SNMP: Inform request, reqid 37, errstat 0, erridx 0
sysUpTime.0 = 44822526
snmpTrapOID.0 = cEventManagerMIB.0.2
ceemHistoryEventEntry.2.1 = 1
ceemHistoryEventEntry.3.1 = 0
ceemHistoryEventEntry.4.1 = 0
ceemHistoryEventEntry.5.1 = 0
ceemHistoryEventEntry.6.1 =
ceemHistoryEventEntry.7.1 = applet: NOTIFY_NMS
ceemHistoryEventEntry.9.1 = 0
ceemHistoryEventEntry.10.1 = 0
ceemHistoryEventEntry.11.1 = LOST_ROUTE_XRV12
ceemHistoryEventEntry.13.1 = 0
ceemHistoryEventEntry.14.1 = 0
ceemHistoryEventEntry.15.1 = 0
ceemHistoryEventEntry.16.1 = 0
SNMP : Packet sent via UDP to FD00:10:31:5::209.162

```

Because there is no SMTP server, the email fails, but the debug shows that the script is trying to send it, which is the correct/expected behavior.

```
! CSR5
%HA_EM-6-LOG: fh_send_mail: : DEBUG(smtp_lib) : <?xml version="1.0"
encoding="UTF-8"
?><fh_smtp_args><fh_smtp_vrf_name>MGMT</fh_smtp_vrf_name><fh_smtp_src>Gigabit
Ethernet1</fh_smtp_src><fh_smtp_port>25</fh_smtp_port><fh_smtp_secure>0</fh_s
smtp_secure></fh_smtp_args>
%HA_EM-6-LOG: NOTIFY_NMS : DEBUG(smtp_lib) : smtp_connect_attempt: 1
%HA_EM-6-LOG: fh_smtp_connect: src: : DEBUG(smtp_lib) : GigabitEthernet1
%HA_EM-6-LOG: fh_smtp_connect: : DEBUG(smtp_lib) : intf name

tty is now going through its death sequence
%HA_EM-6-LOG: NOTIFY_NMS : DEBUG(smtp_lib) : smtp_connect connect fail 261
%HA_EM-6-LOG: NOTIFY_NMS : DEBUG(smtp_lib) : fh_smtp_connect failed at
attempt 1
[snip]
%HA_EM-6-LOG: NOTIFY_NMS : DEBUG(smtp_lib) : smtp_connect_attempt: 5
%HA_EM-6-LOG: fh_smtp_connect: src: : DEBUG(smtp_lib) : GigabitEthernet1
%HA_EM-6-LOG: fh_smtp_connect: : DEBUG(smtp_lib) : intf name
%HA_EM-6-LOG: NOTIFY_NMS : DEBUG(smtp_lib) : smtp_connect connect fail 261
%HA_EM-6-LOG: NOTIFY_NMS : DEBUG(smtp_lib) : fh_smtp_connect failed at
attempt 5
%HA_EM-3-FMPD_SMTP: Error occurred when sending mail to SMTP server:
10.31.5.209 : error in connecting to SMTP server
%HA_EM-3-FMPD_ERROR: Error executing applet NOTIFY_NMS statement 020
```

The last set of scripts will go on CSR7. This is now a DHCP client which represents a residential router connected directly to a PE at layer 3. The PE is the DHCP server, but the customer wants to dynamically update inbound ACLs based on the IP address assigned by the SP. The customer is running NAT, and since inbound ACLs are processed first, almost all of the traffic arriving for CSR7's LAN will be destined for the outside address. There are several complex ACLs/object-groups that are meant to increase security to allow common residential services, such as web browsing, SSH, DNS, and IPSec. Using the regexp action, we can capture values from within parenthesis and store them in variables. The regexp is applied against a string (in our case, the result from the CLI) and we use this to parse valuable output.

```
! CSR7
event manager applet UPDATE_WAN_IP
  event none sync yes
  action 100 cli command "enable"
  action 110 cli command "show ip interface brief | include
GigabitEthernet2.557"
  action 120 regexp " ([0-9.]+) " "$_cli_result" IP_REGEX CUR_IP_ADDR
  action 130 cli command "show ip access-list WAN_IN | include OBJGRP"
  action 140 regexp "per.* ([0-9.]+)" "$_cli_result" ACL_REGEX ACL_IP_ADDR
```

```

action 150 string equal "$CUR_IP_ADDR" "$ACL_IP_ADDR"
action 160 if $_string_result eq "1" goto 500
action 300 cli command "configure terminal"
action 310 cli command "ip access-list extended ACL_WAN_IN"
action 320 cli command "no 10"
action 330 cli command "10 permit object-group OBJGRP_PPS_IN any host
$CUR_IP_ADDR"
action 340 cli command "no 1000"
action 350 cli command "1000 deny ip any host $CUR_IP_ADDR log"
action 360 syslog priority warnings msg "ACL IP address changed from
$ACL_IP_ADDR to $CUR_IP_ADDR, please verify correct network operation."
action 490 exit 1
action 500 syslog priority informational msg "No change to DHCP IP address,
exiting."

```

We will run the script manually first to ensure it works. The current IP address on the WAN facing link is shown below. The script should take this IP and allow anything from the specified object-group (not shown) to enter the residential network.

```

R7#show ip interface brief | include 2.557
GigabitEthernet2.557    10.5.7.102      YES DHCP    up

```

There is an error in the script which can be revealed by using EEM debugging. This shows the input/output of commands and responses between EEM and the parser. The script is asking for an ACL called "WAN_IP" when the ACL is called "ACL_WAN_IP". The parser returns no output which is the same result one would see if the command were issued manually from the CLI. As such, the script cannot populate the variable I have defined, and the operations that rely on it will fail. This debugging is very useful to capture small typos (run-time errors) when writing EEM scripts.

```

R7#debug event-manager action cli
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : IN : R7#show ip access-list
WAN_IN | include OBJGRP
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : OUT : R7#
%HA_EM-3-FMPD_UNKNOWN_ENV: fh_parse_var: could not find environment variable:
ACL_IP_ADDR

```

The correction is easily made and is shown below. The benefit of having action "labels" is that it is easy to make changes to particular lines of code, much like sequence numbers in an ACL or route-map.

```

! CSR7
event manager applet UPDATE_WAN_IP
  no action 130 cli command "show ip access-list WAN_IN | include OBJGRP"
  action 130 cli command "show ip access-list ACL_WAN_IN | include OBJGRP"

```

The script only makes changes when the current IP address (on the interface) is different from the IP address inside the ACL. Assuming they are the same, nothing changes.


```

! CSR5
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : CTL : cli_open called.
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : OUT : R7>
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : IN : R7>enable
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : OUT : R7#
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : IN : R7#show ip interface
brief | include GigabitEthernet2.557
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : OUT : GigabitEthernet2.557
10.5.7.102 YES DHCP up up
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : OUT : R7#
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : IN : R7#show ip access-list
ACL_WAN_IN | include OBJGRP
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : OUT : 10 permit object-
group OBJGRP_PPS_IN any host 10.5.7.102 (42 matches)
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : OUT : R7#
%HA_EM-6-LOG: UPDATE_WAN_IP: No change to DHCP IP address, exiting.
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : CTL : cli_close called.

```

Assuming the two did not match, as shown below, the ACL gets updated with the current IP address. Rather than paste pages of debug, I only show the relevant parts.

```

! CSR7
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : IN : R7#show ip interface
brief | include GigabitEthernet2.557
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : OUT : GigabitEthernet2.557
10.5.7.102 YES DHCP up
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : IN : R7#show ip access-list
ACL_WAN_IN | include OBJGRP
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : OUT : 10 permit ip any
any (12 matches)

```

The script did not detect the string “10.5.7.102” inside a permit statement within the ACL. Now EEM will fix the ACL to match the DHCP address.

```

! CSR7
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : IN : R7#configure terminal
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : OUT : Enter configuration
commands, one per line. End with CNT
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : IN : R7(config)#ip access-
list extended ACL_WAN_IN
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : IN : R7(config-ext-nacl)#no
10
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : IN : R7(config-ext-nacl)#10
permit object-group OBJGRP_PPS_IN any host 10.5.7.102
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : IN : R7(config-ext-nacl)#no
1000

```

```
%HA_EM-6-LOG: UPDATE_WAN_IP : DEBUG(cli_lib) : : IN : R7(config-ext-
nacl)#1000 deny ip any host 10.5.7.102 log
%HA_EM-4-LOG: UPDATE_WAN_IP: ACL IP address changed from 1000 to 10.5.7.102,
please verify correct network operation.
```

We can verify the ACL was changed. Pinging from in->out should work, but telnet should not. A few basic tests confirm this.

```
R7#show ip access-lists ACL_WAN_IN
Extended IP access list ACL_WAN_IN
 10 permit object-group OBJGRP_PPS_IN any host 10.5.7.102 (41 matches)
 1000 deny ip any host 10.5.7.102 log (9 matches)
 2000 permit udp any any eq bootpc

R7#ping 10.14.14.14 source lo0
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.14.14.14, timeout is 2 seconds:
Packet sent with a source address of 192.168.7.7
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/5/17 ms

R7#telnet 10.14.14.14 /source-interface lo0
%FMANFP-6-IPACCESSLOGP: F0: fman_fp_image: list ACL_WAN_IN denied tcp
10.14.14.14(23) -> 10.5.7.102(4096), 1 packet
```

We can synchronously automate this procedure by wrapping it with a cron timer script. This will run the script every day at 0600 and 1800. The cron timer works as follows: minute, hour, day, month, day of week. In this case, at 00 minutes, 06 and 18 hours, every day, every month, every day of the week, execute this action. Cron is a sizable Linux-oriented topic with many different options, but the basics are shown below. When this script runs, it will simply call the other script, as the main script has no associated event. Notice that actions are alphanumeric labels, not numbers, so they are sorted alphabetically ASCII-wise.

```
! CSR7
event manager applet SCHEDULER_WAN_IP
 event timer cron name CRON_TWICE_DAILY cron-entry "0 6,18 * * *"
 action 100 syslog priority informational msg "Starting routine IP check ..."
 action 110 cli command "enable"
 action 120 cli command "event manager run UPDATE_WAN_IP"
 action zzz syslog priority informational msg "Update complete."
```

I also added the following script to allow things to work when the DHCP server decides to assign a new address. This helps automate the process with an asynchronous check. The syslog messages are shown below, along with the ACL verification. Since the log messages contain their script names, it's easy to see which logs came from which scripts when you have nested policies. I use all ASCII labels to show that it is indeed sorted by ASCII decimal numbers (0-9, A-Z, a-z, not including special characters). In summary, this

script will invoke the UPDATE_WAN_IP script when DHCP issues a syslog indicating the WAN IP has changed. If, for some reason, that syslog does not appear, the timer-based script SCHEDULER_WAN_IP will run twice daily as a fail-safe.

```
! CSR7
event manager applet DHCPEVENT_WAN_IP
  event syslog occurs 1 pattern "assign.*DHCP.*address"
  action A syslog priority informational msg "DHCP indicates IP activity,
starting IP check ..."
  action Z cli command "enable"
  action a cli command "event manager run UPDATE_WAN_IP"
  action z syslog priority informational msg "Update complete."

%DHCP-6-ADDRESS_ASSIGN: Interface GigabitEthernet2.557 assigned DHCP address
10.5.7.104, mask 255.255.255.0, hostname R7
%HA_EM-6-LOG: DHCPEVENT_WAN_IP: DHCP indicates IP activity, starting IP check
...
%HA_EM-4-LOG: UPDATE_WAN_IP: ACL IP address changed from 2000 to 10.5.7.104,
please verify correct network operation.
%HA_EM-6-LOG: DHCPEVENT_WAN_IP: Update complete.

R7#show ip access-lists ACL_WAN_IN
Extended IP access list ACL_WAN_IN
  10 permit object-group OBJGRP_PPS_IN any host 10.5.7.104
  1000 deny ip any host 10.5.7.104 log
  2000 permit udp any any eq bootpc (2 matches)
```

A quick way to verify that user-written EEM scripts are registered (valid events, etc) is to query the EEM policy database. This shows the entire script, which can be long (omitted), but it's valuable to see the scripts in summary view.

```
R7#show event manager policy registered user
No.  Class    Type    Event Type    Trap    Time Registered    Name
1    applet    user    none          Off     DAY MON 20 23:26:26 2015  UPDATE_WAN_IP
  policyname {UPDATE_WAN_IP} sync {yes}
  maxrun 20.000
  action 100 cli command "enable"
[snip]

2    applet    user    timer cron    Off     DAY MON 20 23:27:24 2015  SCHEDULER_WAN_IP
  name {CRON_TWICE_DAILY} cron entry {0 6,18 * * *}
  maxrun 20.000
  action 100 syslog priority informational msg "Starting routine IP check ..."
[snip]

3    applet    user    syslog        Off     DAY MON 20 23:27:25 2015  DHCPEVENT_WAN_IP
  occurs 1 pattern {assign.*DHCP.*address}
  maxrun 20.000
```

```

    action A syslog priority informational msg "DHCP indicates IP activity, starting IP
check ..."
[snip]

```

Complex scripts like this can be very difficult to troubleshoot since there does not appear to be a way to show or debug commands to see the values of these locally-defined variables. For example, the logical comparisons and regexp operations are totally invisible to the user, it seems. Creating an environment (global scope) variable is visible, however. I believe that the variables used in the script are deallocated once the script completes, much like local variables in any procedural program language, so viewing them would be difficult. This is a side note and not relevant to the testing at hand. Example environment variables as shown below.

```

! CSR7
event manager environment ENV_VAR hello

R7#show event manager environment all
No.   Name                               Value
1     ENV_VAR                           hello

```

We can write a tiny script to display the value of this variable. This demonstrates the value of environment variables as they can be used to “pass” information between sequential scripts in lieu of true parameterization. One script could set the variable, another could manipulate it, and a third could display the results.

```

! CSR7
event manager applet SHOW_ENV_VAR
    event none
    action 010 puts "$ENV_VAR"

R7#event manager run SHOW_ENV_VAR
hello

```

44.4 Embedded Packet Capture (EPC)

EPC is a feature on IOS and XE routers that can capture packets entering or exiting between interfaces. It's a lightweight and no-frills tool, and the exported captures from EPC can be viewed as normal PCAP files on popular sniffer applications like Wireshark. This is valuable when trying to capture packets for inspection or troubleshooting. Because fancy tools like Wireshark are (likely) not available on CCIE lab exams, I prefer to use the hexadecimal buffer dumps to identify traffic. It works in both the CEF and process switching paths, unlike “debug ip packet”. EPC does come at a performance penalty, so use it sparingly. Up to this point, we have used EPC extensively for verification in the data-plane.

The most basic EPC policy matches all traffic and specifies a set of interfaces. For example, to measure all inbound traffic on CSR5 within the MPLS core, we can use this EPC. We verify it immediately after to ensure the filter (match any) and interfaces are correct. Because we have not started this capture, the

state is “Inactive”. There are no limits on the buffers length (bad idea in production) and no additional filters/samplers are applied.

```
! CSR5 (from exec mode)
monitor capture CORE match any
monitor capture CORE interface gig2.551 in
monitor capture CORE interface gig2.556 in
```

```
R5#show monitor capture CORE
Status Information for Capture CORE
```

Target Type:

Interface: GigabitEthernet2.551, Direction: in

Interface: GigabitEthernet2.556, Direction: in

Status : Inactive

Filter Details:

Capture all packets

Buffer Details:

Buffer Type: LINEAR (default)

Limit Details:

Number of Packets to capture: 0 (no limit)

Packet Capture duration: 0 (no limit)

Packet Size to capture: 0 (no limit)

Packet sampling rate: 0 (no sampling)

Similar to BGP dampening verification, we can also see the parameters of the EPC explicitly, to include defaults. These are the exact CLI commands, so you can copy/edit/paste as necessary. The buffer size of 10 is measured in MB, not packets. This is very handy for making quick modifications as you can copy/paste it into a text editor, make changes, and copy/paste the result directly into the CLI.

```
R5#show monitor capture CORE parameter
  monitor capture CORE interface GigabitEthernet2.551 in
  monitor capture CORE interface GigabitEthernet2.556 in
  monitor capture CORE match any
  monitor capture CORE buffer size 10
  monitor capture CORE limit pps 1000
```

We can start the capture, verify that it has started. This also creates a syslog message.

```
R5#monitor capture CORE start
%BUFCAP-6-ENABLE: Capture Point CORE enabled.
```

```
R5#show monitor capture CORE | include Status
Status Information for Capture CORE
  Status : Active
```

There are four ways to view the buffer, each showing varying levels of detail. The first is the summary view which talks about the buffer’s length, how many packets are in the buffer, at which rate do packets

enter the buffer, etc. No packet-level information is shown and this is only useful when trying to get a general idea of how much traffic is being captured by the filter.

```
R5#show monitor capture CORE buffer
buffer size (KB) : 10240
buffer used (KB) : 128
packets in buf   : 184
packets dropped  : 0
packets per sec  : 1
```

The next level of detail reveals a summary of all packet/frame headers. This is useful to troubleshoot basic reachability when layers of encapsulation/payloads are not relevant. It is especially valuable when the traffic is IPv4/v6 since those headers are clearly shown. Though MAC addresses are shown for non-IPv4/v6 Ethertypes, most administrators don't memorize MAC addresses on their network devices.

```
R5#show monitor capture CORE buffer brief
-----
#    size    timestamp      source                destination          protocol
-----
 0    82      0.000000      23.0.0.12             -> 23.0.0.5             ICMP
 1    80      0.048993      23.5.6.6               -> 224.0.0.2            UDP
 2    74      0.814005      23.5.6.6               -> 23.5.6.5             ICMP
 3    82      1.116998      23.0.0.12             -> 23.0.0.5             ICMP
 4    88      1.664018      23.5.11.11            -> 224.0.0.2            UDP
 5 1517      2.067989      00:50:56:A9:DE:0D     -> 09:00:2B:00:00:05    LLC
[snip]
```

Next, we can see additional detail of buffer entries. This is useful for seeing layers of encapsulation for the first 64 bytes of each packet; for IPv4, this is usually enough to see a few layers of encapsulation and the beginning of the payload. For IPv6, it may not be sufficient. As an example, I highlight the IP source/destination addresses in the first header as hexadecimal digits. The beginning of the packet's payload is shown in green; seeing this indicates that the entire packet encapsulation is visible and there is no need to progress to the next level of detail when trying to verify encapsulations.

```
R5#show monitor capture CORE buffer detail
-----
#    size    timestamp      source                destination          protocol
-----
 0    82      0.000000      23.0.0.12             -> 23.0.0.5             ICMP
0000: 005056A9 DC630050 56A99C60 81000DDF  .PV..c.PV..`....
0010: 08004500 00404713 0000FF01 46991700  ..E..@G.....F...
0020: 000C1700 00050000 A9E245EF 00010000  ....E.....
0030: 000023D4 D9AFABCD ABCDABCD ABCDABCD  ..#.....

 1    80      0.048993      23.5.6.6               -> 224.0.0.2            UDP
0000: 01005E00 00020050 56A9DE0D 81000DE4  ..^....PV.....
```

```

0010: 080045C0 003E0000 00000111 BBE21705 ..E..>.....
0020: 0606E000 00020286 0286002A C6290001 .....*)...
0030: 001E1700 00060000 01000014 00000000 .....

  2   74   0.814005   23.5.6.6       -> 23.5.6.5       ICMP
0000: 005056A9 DC630050 56A9DE0D 81000DE4 .PV..c.PV.....
0010: 080045C0 00383A1D 0000FF01 46D31705 ..E..8:.....F...
0020: 06061705 06050303 15840000 00004500 .....E.
0030: 00540382 0000FE11 85071705 06051700 .T.....

```

The most detailed view is the buffer dump. This is the raw hexadecimal information with no additional detail (no size calculations, timestamps, ASCII interpretation, etc). This is useful for troubleshooting large encapsulation stacks or payload (application) issues. Again, I highlight the IP source/destination addresses in the first packet. This command is only useful if you are comfortable reading hexadecimal, which I recommend for any CCIE lab exam.

```

R5#show monitor capture CORE buffer dump
0
  0000: 005056A9 DC630050 56A99C60 81000DDF .PV..c.PV..`....
  0010: 08004500 00404713 0000FF01 46991700 ..E..@G.....F...
  0020: 000C1700 00050000 A9E245EF 00010000 .....E.....
  0030: 000023D4 D9AFABCD ABCDABCD ABCDABCD ..#.....
  0040: ABCDABCD ABCDABCD ABCDABCD ABCD0005 .....
  0050: 0500 ..

1
  0000: 01005E00 00020050 56A9DE0D 81000DE4 ..^....PV.....
  0010: 080045C0 003E0000 00000111 BBE21705 ..E..>.....
  0020: 0606E000 00020286 0286002A C6290001 .....*)...
  0030: 001E1700 00060000 01000014 00000000 .....
  0040: 04000004 000F0000 04010004 17000006 .....

[snip]

```

We can stop the buffer after we have collected the information we need. All of the aforementioned show commands can be used while the buffer is running or stopped.

```

R5#monitor capture CORE stop
%BUFCAP-6-DISABLE: Capture Point CORE disabled.

R5#show monitor capture CORE | include Status
Status Information for Capture CORE
  Status : Inactive

```

First, we will look at traffic filters. The simplest method is to apply them in-line with the policy. On CSR6, we will capture LDP messaging (transit or otherwise) arriving from XRv2 as long as it comes from within the SP address block. We will also limit the number of packets captured to 6. The buffer is linear, which means the 7th packet is not captured at all.

```

! CSR6 (in exec mode)
monitor capture CORE2 interface gig2.562 in match ipv4 protocol udp
23.0.0.0/8 any eq 646
monitor capture CORE2 limit packets 6

```

```

R6#show monitor capture CORE2
Status Information for Capture CORE2
  Target Type:
    Interface: GigabitEthernet2.562, Direction: in
    Status : Inactive
  Filter Details:
    IPv4
    Source IP: 23.0.0.0/8
    Destination IP: any
    Protocol: udp
    Destination port(s): = 646
  Buffer Details:
    Buffer Type: LINEAR (default)
  Limit Details:
    Number of Packets to capture: 6
    Packet Capture duration: 0 (no limit)
    Packet Size to capture: 0 (no limit)
    Packet sampling rate: 0 (no sampling)

```

We start the buffer and wait a few seconds for it to fill up with LDP messages. It doesn't take long with a buffer length of 6 packets.

```

R6#monitor capture CORE2 start
%BUFCAP-6-ENABLE: Capture Point CORE2 enabled.

```

The capture is automatically stopped when the packet count is reached. We can look at the buffer brief to see 6 LDP packets. In case there is any doubt that these are LDP packets, the buffer detail shows us some of the hexadecimal dump, and we can extract the UDP destination port 646 (0x286) from the messages.

```

! CSR5
%BUFCAP-6-DISABLE_ASYNC: Capture Point CORE2 disabled. Reason : Packet Count Reached

```

```

R6#show monitor capture CORE2 buffer brief
-----
#    size    timestamp      source                destination    protocol
-----
0    88      0.000000    23.6.12.12          -> 224.0.0.2      UDP
1    88      4.475018    23.6.12.12          -> 224.0.0.2      UDP
2    88      9.648013    23.6.12.12          -> 224.0.0.2      UDP

```


3	88	14.108988	23.6.12.12	->	224.0.0.2	UDP
4	88	18.475018	23.6.12.12	->	224.0.0.2	UDP
5	88	22.901006	23.6.12.12	->	224.0.0.2	UDP

R6#show monitor capture CORE2 buffer detailed

#	size	timestamp	source	destination	protocol
0	88	0.000000	23.6.12.12	-> 224.0.0.2	UDP
0000:	01005E00	00020050	56A90E6F	8100CDEA	..^....PV..o....
0010:	080045C0	0046407F	00000111	75541706	..E..F@.....uT..
0020:	0C0CE000	00020286	02860032	FDF50001	2....
0030:	00261700	000C0000	0100001C	00000000	.&.....

We can also measure traffic destined specifically to the control-plane. We will measure BGP traffic in this way; CSR6 is potentially in the transit path for XRv2's BGP session to CSR5, so by capturing the control-plane only, the transit traffic is ignored. We will use the ACL for matching BGP traffic, versus an in-line filter, and set a time limit of 2 minutes (120 seconds). We can specify matching in "both" directions as well. You must configure the ACL before attaching it to EPC, as shown below.

```
! CSR6
monitor capture BGPCP control-plane both access-list ACL_BGP limit duration
120
```

Non-existent access-list specified
Failed to attach existing ACL

Assuming the ACL exists (matches TCP port 179), the binding is successful.

```
! CSR6
ip access-list extended ACL_BGP
 permit tcp any eq bgp any
 permit tcp any any eq bgp
```

```
monitor capture BGPCP control-plane both access-list ACL_BGP limit duration
120
```

```
R6#show monitor capture BGPCP
Status Information for Capture BGPCP
Target Type:
  Interface: Control Plane, Direction : both
  Status : Inactive
Filter Details:
  Access-list: ACL_BGP
Buffer Details:
  Buffer Type: LINEAR (default)
Limit Details:
  Number of Packets to capture: 0 (no limit)
  Packet Capture duration: 120
```

```
Packet Size to capture: 0 (no limit)
Packet sampling rate: 0 (no sampling)
```

```
R6#show access-list ACL_BGP
Extended IP access list ACL_BGP
 10 permit tcp any eq bgp any
 20 permit tcp any any eq bgp
```

We start the capture and it runs for 2 minutes. It automatically disables itself after that time period. The reason for the ~2.5 minute time difference is because the capture starts once the first packet is received, so it took about ~24 seconds for the first packet to be captured at time = 0.

```
! Time 18:43:26.385
R6#monitor capture BGPCP start
%BUFCAP-6-ENABLE: Capture Point BGPCP enabled.

! Time 18:45:50.504
%BUFCAP-6-DISABLE_ASYNC: Capture Point BGPCP disabled. Reason : Time Duration
Reached
```

The buffer brief shows 8 packets received over the capture period. We can see all of the traffic is between CSR6 and the RR (CSR5), with nothing else captured. Notice that the first packet in every capture is timestamped at time = 0. As mentioned above that capture does not start until the first matching packet is captured.

```
R6#show monitor capture BGPCP buffer brief
-----
#    size    timestamp      source          destination    protocol
-----
0    93      0.000000      23.0.0.6        -> 23.0.0.5      TCP
1    78      2.198039      23.0.0.5        -> 23.0.0.6      TCP
2    97      24.112985     23.0.0.5        -> 23.0.0.6      TCP
3    74      24.314030     23.0.0.6        -> 23.0.0.5      TCP
4    93      52.228036     23.0.0.6        -> 23.0.0.5      TCP
5    78      53.832009     23.0.0.5        -> 23.0.0.6      TCP
6   185      73.417022     23.0.0.5        -> 23.0.0.6      TCP
7    74      73.418029     23.0.0.6        -> 23.0.0.5      TCP
```

The detailed buffer information can show us the TCP port of 179 also. The TCP session table tells us that packets sent to CSR5 should have destination port 179 (0xB3) and source port 26511 (0x678F). We can confirm these ports are correct below by checking the TCP TCB for this particular BGP session.

```
R6#show tcp brief | include 179
7F4AA56B52E8  23.0.0.6.26511          23.0.0.5.179          ESTAB
```

```
R6#show monitor capture BGPCP buffer detailed
-----
```

#	size	timestamp	source	destination	protocol
0	93	0.000000	23.0.0.6	-> 23.0.0.5	TCP
0000:	00000000	00000000	00000000	080045C0	E.
0010:	004F0B2A	4000FF06	41B41700	00061700	.O.*@...A.....
0020:	0005678F	00B35AB2	075A23A5	3A7BA018	..g...Z...Z#..{..
0030:	3AC2784C	00001312	D0BF5878	101E6018	..xL.....Xx..`.

You can also use MQC class-maps to match traffic. NBAR is not currently supported, so I would consider this functionality limited when ACLs are available. DSCP matching also does not work. It could be useful as a way to test outbound QoS policies in advance, but right now the MQC support in EPC seems very limited. For this example, I configure a class-map to match large ping packets that arrive from CSR5. This combines a simple ACL with the packet length matching capability of the class-map. I also consider a packet per second (pps) limit which can conserve resources on the router. The circular buffer means that once the buffer is full (I reduced size to 2 MB), the oldest entries are overwritten by newer ones. This is unlike a linear buffer where new packets are rejected once the buffer is full as the capture stops automatically. Other examples of circular buffers include the XE syslog and Linux kernel ring buffers.

```
R6#show monitor capture BIGPING parameter
  monitor capture BIGPING interface GigabitEthernet2.556 in
  monitor capture BIGPING class-map CMAP_BIG_PING
  monitor capture BIGPING buffer size 2 circular
  monitor capture BIGPING limit pps 1000
```

```
! CSR6
ip access-list extended ACL_PING
  permit icmp any any echo

class-map match-all CMAP_BIG_PING
  match access-group name ACL_PING
  match packet length min 1000 max 1500
```

When we send packets less than 1000 bytes from CSR5 towards CSR6, they are not captured. These packets do not satisfy the minimum length condition of the class-map.

```
R5#ping 23.5.6.6 timeout 0 size 900
Type escape sequence to abort.
Sending 5, 900-byte ICMP Echos to 23.5.6.6, timeout is 0 seconds:
.....
```

```
R6#show monitor capture BIGPING buffer
buffer size (KB) : 2048
buffer used (KB) : 0
packets in buf   : 0
packets dropped  : 0
packets per sec  : 0
```

When we send packets that are greater or equal to 1000 bytes, or less than or equal to 1500 bytes ($1000 \leq x \leq 1500$), the packets are matched. The reason I tested this extensively was to prove that the limits of the match, 1000 and 1500, are inclusive. We sent exactly 15 packets across 3 pings and saw exactly 15 packets captured. The detailed output shows the different lengths as well.

```
R5#ping 23.5.6.6 timeout 0 size 1100
Type escape sequence to abort.
Sending 5, 1100-byte ICMP Echos to 23.5.6.6, timeout is 0 seconds:
.....
```

```
R5#ping 23.5.6.6 timeout 0 size 1000
Type escape sequence to abort.
Sending 5, 1000-byte ICMP Echos to 23.5.6.6, timeout is 0 seconds:
.....
```

```
R5#ping 23.5.6.6 timeout 0 size 1500
Type escape sequence to abort.
Sending 5, 1000-byte ICMP Echos to 23.5.6.6, timeout is 0 seconds:
.....
```

```
R6#show monitor capture BIGPING buffer
buffer size (KB) : 2048
buffer used (KB) : 128
packets in buf   : 15
packets dropped  : 0
packets per sec  : 0
```

```
R6#show monitor capture BIGPING buffer brief
```

#	size	timestamp	source	destination	protocol
0	1118	0.000000	23.5.6.5	-> 23.5.6.6	ICMP
[snip, 4 more copies]					
5	1018	51.474026	23.5.6.5	-> 23.5.6.6	ICMP
[snip, 4 more copies]					
10	1518	151.242043	23.5.6.5	-> 23.5.6.6	ICMP
[snip, 4 more copies]					

The last topic includes exporting these captures to remote file systems. This is commonly used to TFTP captured data to a management station for PCAP analysis (Wireshark, automated process, etc). Because I do not have a TFTP server, I will copy the file to bootflash as a .pcap file. We can read the file from memory directly (shown below) or save it offline until we set up a TFTP server later.

```
R6#monitor capture BIGPING export bootflash:bigping.pcap
Exported Successfully
```

```
R6#more bootflash:bigping.pcap
```

```

00000000: A1B2C3D4 00020004 00000000 00000000 !2CT .... ....
00000010: 00010000 00000001 55D8CD5E 000D9841 .... UXM^ ...A
00000020: 0000045E 0000045E 005056A9 DE0D0050 ...^ ...^ .PV) ^...P
00000030: 56A9DC63 81000DE4 08004500 044C554B V)\c ...d ..E. .LUK
00000040: 0000FF01 28511705 06051705 06060800 .... (Q.. ....
00000050: 9AB753EA 00000000 00002440 DDB4ABCD .7Sj .... ..$@ ]4+M
00000060: ABCDABCD ABCDABCD ABCDABCD ABCDABCD +M+M +M+M +M+M +M+M
[snip]

```

Assuming we had a remote file system, the syntax is nearly identical, except we need to specify the server address. The process fails because I don't have a server online.

```

R6#monitor capture BIGPING export tftp://10.99.99.99/bigping.pcap
.....Failed to Export: Failed to create export file

```

Real XR routers with hardware linecards have some additional packet-capturing abilities, but XRv is limited. Using the “monitor interface” family of commands, we can see some real-time statistics. The command by itself displays the following screen. The screen updates every 2 seconds, and the user can use the commands listed at the bottom to control things. “Freeze” and “thaw” will stop and restart the refreshes, respectively. Most of the other commands are self-explanatory (quit, clear, etc). While this is running, we start a steady stream of pings from CSR5. We can see the increase in InBytes on Gig0/0/0/0.551 as a result. This counts against both the main and subinterfaces when this occurs.

```

RP/0/0/CPU0:XRv1#monitor interface
XRv1                      Monitor Time: 00:00:06                SysUptime: 411:29:22
Protocol:General
Interface                  In (bps)          Out (bps)          InBytes/Delta      OutBytes/Delta
Gi0/0/0/0.512              2000/ 0%          1000/ 0%           112.1M/260         111.1M/88
Gi0/0/0/0.541              0/ 0%             0/ 0%              1.6M/0             1.7M/0
Gi0/0/0/0.551              4000/ 0%          4000/ 0%           113.7M/25094       113.1M/25150
Gi0/0/0/0                  13000/ 0%         6000/ 0%           1.3G/28615         232.2M/25238
Mg0/0/CPU0/0              0/ --%            0/ --%             6.1M/0             1.5M/0

Quit='q',      Clear='c',      Freeze='f', Thaw='t',
Next set='n',  Prev set='p', Bytes='y',  Packets='k'
(General='g',  IPv4 Uni='4u',  IPv4 Multi='4m', IPv6 Uni='6u', IPv6 Multi='6m')

```

More useful to me is seeing the packet counters versus the byte counters. Use the “k” command to view packets instead. This can be used for rudimentary traffic flow verification.

```

XRv1                      Monitor Time: 00:01:02                SysUptime: 411:38:21
Protocol:General
Interface                  In (pps)          Out (pps)          InPkts/Delta       OutPkts/Delta
Gi0/0/0/0.512              1                 0                 189810/2            181722/0
Gi0/0/0/0.541              0                 0                 19059/0             18916/0
Gi0/0/0/0.551              33                34                221845/0            229986/2

```

Gi0/0/0/0	37	34	3.9M/12	820750/2
Mg0/0/CPU0/0	0	0	65089/0	17759/0

We can also specify a specific interface to monitor for a more detailed view. With the ping from CSR5 still going, we examine Gig0/0/0/0.551. We can see the input/output rate of this ping flow is about 32 kbps at this point. Using the “i” command allows you to specify another interface without having to exit. There is no difference between the brief and detail views in XRv, either. The error statistics at the bottom would be particularly valuable when trying to assess how frequently those errors are occurring, and on which interfaces, in real time.

```
RP/0/0/CPU0:XRv1#monitor interface gig0/0/0/0.551
XRv1                               Monitor Time: 00:00:24           SysUptime: 411:34:58
GigabitEthernet0/0/0/0.551 is up, line protocol is up
Encapsulation 802.1Q
```

Traffic Stats:(2 second rates)			Delta
Input	Packets:	213458	198
Input	pps:	31	
Input	Bytes:	115045242	24961
Input	Kbps (rate):	31	(0%)
Output	Packets:	221388	201
Output	pps:	32	
Output	Bytes:	114412648	23625
Output	Kbps (rate):	31	(0%)

Errors Stats:		
Input	Total:	0
Input	CRC:	0
Input	Frame:	0
Input	Overrun:	0
Output	Total:	0
Output	Underrun:	0

```
Quit='q', Freeze='f', Thaw='t', Clear='c', Interface='i',
Next='n', Prev='p'
```

```
Brief='b', Detail='d', Protocol(IPv4/IPv6)='r'
```

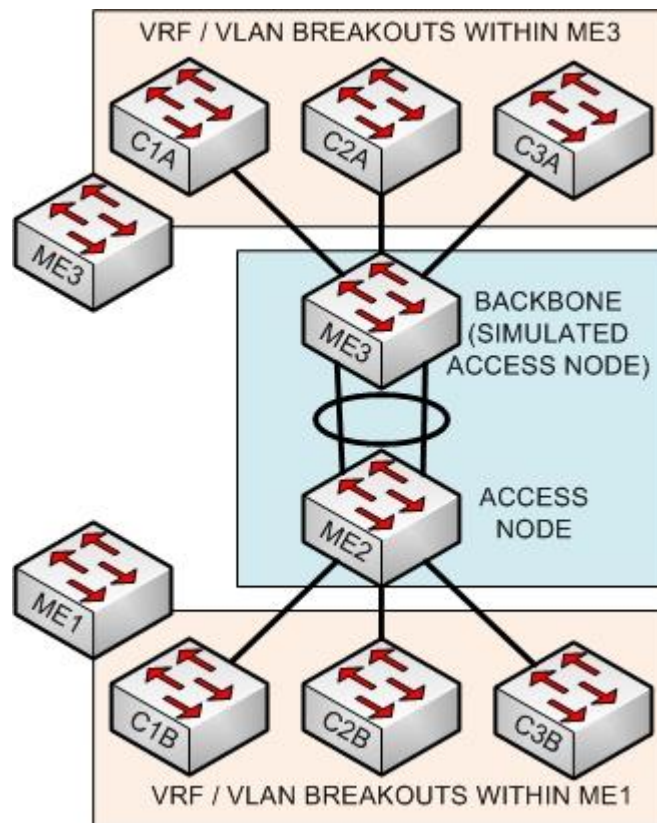
45. Describe, implement, and troubleshoot port mirroring protocols

Because not all of these features are reasonably testable on all platforms, two labs are used. SPAN and RSPAN are tested using the TR-101 VLAN lab since it is a functional QinQ network with 3 Cisco ME-3400-24TS-A switches. ERSPAN will use a CSR1000v / XRv lab since we can avoid the complexities of hypervisor switches with IP-encapsulated sessions.

45.1 Switch port analyzer (SPAN)

SPAN (also called local SPAN) is a feature that allows a network device to capture traffic on a switch port and flood it out of another port for evaluation. This is very useful for troubleshooting when a packet

sniffer is attached to the “mirror” or “span” port. It also has security applications for intrusion detection systems (IDS). The feature can specify a number of source interfaces or VLANs along with a number of destinations interfaces (not VLANs). You also cannot mix port and VLAN sources in the same session. The configurations for this lab are rolled into the RSPAN session as well and are available at the end of the RSPAN session.



We will connect our sniffer to port 24 on ME2 (not shown in diagram) and simply “no shutdown” the port. The port-type does not matter when SPAN is used since the line-protocol is forced down. The switchport mode and VLANs also make no difference for a basic SPAN configuration. We will configure a basic SPAN session to look at bidirectional traffic on the backbone LACP bond to ME3 and mirror it to port 24.

```
! ME2
interface FastEthernet0/24
  description LOCAL SPAN
  no shutdown

monitor session 1 source interface Po1
monitor session 1 destination interface Fa0/24

ME2#show monitor session 1
Session 1
```

```

-----
Type                        : Local Session
Source Ports                :
    Both                    : Po1
Destination Ports          : Fa0/24
    Encapsulation           : Native
    Ingress                 : Disabled

```

We also verify that the port state is “monitoring”; this actually qualifies as line-protocol down.

```

ME2#show interfaces fastEthernet 0/24 status
Port      Name           Status      Vlan    Duplex  Speed Type
Fa0/24    LOCAL SPAN        monitoring  1       a-full  a-100 10/100BaseTX

```

We send some pings from ME3 to ME1 within VRF C1 (10.255.10.3 to 10.255.10.1). Below is a snapshot of what Wireshark sees. Notice that it captures the L2/L3 header with the proper C-MACs and C-IP addresses, but no VLAN encapsulation. Above, we see that the encapsulation is native, which means the original encapsulation is discarded by the SPAN process when replicating the traffic.

```

+ Frame 1: 114 bytes on wire (912 bits), 114 bytes captured (912 bits) on interface 0
+ Ethernet II, Src: CiscoInc_0b:16:c1 (00:1f:9d:0b:16:c1), Dst: CiscoInc_92:f7:41 (00:1d:46:92:f7:41)
+ Internet Protocol Version 4, Src: 10.255.10.3 (10.255.10.3), Dst: 10.255.10.1 (10.255.10.1)
+ Internet Control Message Protocol

```

To correct this, we can instruct SPAN to replicate the encapsulation towards the sniffer. I think there is a problem with the OS driver on my sniffer because it only shows the inner-most C-VLAN tag (value 10), when we know there is an S-tag on these frames as well. At a minimum, we see a difference in output which shows at some additional encapsulation, although ideally we would see all of it.

```

! ME2
monitor session 1 source interface Po1
monitor session 1 destination interface Fa0/24 encapsulation replicate

```

```

ME2#show monitor session 1
Session 1
-----
Type                        : Local Session
Source Ports                :
    Both                    : Po1
Destination Ports          : Fa0/24
    Encapsulation           : Replicate
    Ingress                 : Disabled

```

```

+ Frame 60: 118 bytes on wire (944 bits), 118 bytes captured (944 bits) on interface 0
+ Ethernet II, Src: CiscoInc_0b:16:c1 (00:1f:9d:0b:16:c1), Dst: CiscoInc_92:f7:41 (00:1d:46:92:f7:41)
+ 802.1Q Virtual LAN, PRI: 0, CFI: 0, ID: 10
+ Internet Protocol Version 4, Src: 10.255.10.3 (10.255.10.3), Dst: 10.255.10.1 (10.255.10.1)
+ Internet Control Message Protocol

```


We can also use the sniffer laptop like a regular host. Traffic coming from the sniffer can be placed into an “ingress” VLAN of our choice. I create a new VLAN 169 and apply a link-local IPv4 address to it. This will be in the same IP subnet as the NIC driver that is running Wireshark (169.254.0.1/24). In order to make the SVI go up, I have to add VLAN 169 to some port that is up, so I add it to the backbone LACP bond. This is clearly for testing purposes and generally you would have already have more ports in VLAN 169 if you were putting a host into that VLAN. We can specify the ingress replication as well in the event that the host was tagging traffic, but in this case, it is not.

```
! ME2
vlan 169
  name SPAN_INGRESS

interface Vlan169
  ip address 169.254.0.2 255.255.255.0

monitor session 1 source interface Po1
monitor session 1 destination interface Fa0/24 encapsulation replicate
ingress untagged vlan 169

interface Port-channel1
  switchport trunk allowed vlan add 169

ME2#show monitor session 1
Session 1
-----
Type                : Local Session
Source Ports        :
    Both            : Po1
Destination Ports   : Fa0/24
Encapsulation       : Replicate
    Ingress         : Enabled, default VLAN = 169
Ingress encaps      : Untagged
```

To test it, I will try to ping ME2’s SVI from the sniffer laptop. It appears to work, so I verify the ARP entry in the sniffer against the SVI MAC address to confirm.

```
C:\Users\Nick>ping 169.254.0.2
Pinging 169.254.0.2 with 32 bytes of data:
Request timed out.
Reply from 169.254.0.2: bytes=32 time=1ms TTL=255
Reply from 169.254.0.2: bytes=32 time=6ms TTL=255
Reply from 169.254.0.2: bytes=32 time=2ms TTL=255

Ping statistics for 169.254.0.2:
    Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
```

Minimum = 1ms, Maximum = 6ms, Average = 3ms

```
C:\Users\Nick>arp -a
```

```
[snip]
```

```
Interface: 169.254.0.1 --- 0x6
```

Internet Address	Physical Address	Type
169.254.0.2	04-c5-a4-53-d8-41	dynamic

```
ME2#show interfaces vlan 169 | include bia
```

```
Hardware is EtherSVI, address is 04c5.a453.d841 (bia 04c5.a453.d841)
```

Last, we can specify which VLANs matter to us. When sniffing a trunk port, one may not want to see every VLAN, but rather an arbitrary subset. On ME2, we will capture on S-tagged traffic in VLAN 101 (which includes C1 and C2 but not C3) as well as VLANs 103 – 105. Since C3 traffic is inside VLAN 102, we should not see that traffic. The word “filter” uses positive logic and really means “match” as opposed to “remove”.

```
! ME2
```

```
monitor session 1 filter vlan 101 , 103 - 105
```

```
ME2#show monitor session 1
```

```
Session 1
```

```
-----
```

Type	: Local Session
Source Ports	:
Both	: Po1
Destination Ports	: Fa0/24
Encapsulation	: Replicate
Ingress	: Enabled, default VLAN = 169
Ingress encap	: Untagged
Filter VLANs	: 101,103-105

To test it, I send a single packet in each customer VRF across the network. The first two packets have an S-tag of 101 with C-tags of 10 and 20. The last packet has an S-tag of 102 with a C-tag of 30. All of this QinQ configuration was done in the TR-101 section.

```
ME3#ping vrf C1 10.255.10.1 repeat 1
```

```
Type escape sequence to abort.
```

```
Sending 1, 100-byte ICMP Echos to 10.255.10.1, timeout is 2 seconds:
```

```
!
```

```
Success rate is 100 percent (1/1), round-trip min/avg/max = 8/8/8 ms
```

```
ME3#ping vrf C2 10.255.20.1 repeat 1
```

```
Type escape sequence to abort.
```

```
Sending 1, 100-byte ICMP Echos to 10.255.20.1, timeout is 2 seconds:
```

```
!
```

```
Success rate is 100 percent (1/1), round-trip min/avg/max = 8/8/8 ms
```

```
ME3#ping vrf C3 10.255.30.1 repeat 1
Type escape sequence to abort.
Sending 1, 100-byte ICMP Echos to 10.255.30.1, timeout is 2 seconds:
!
Success rate is 100 percent (1/1), round-trip min/avg/max = 8/8/8 ms
```

I also adjust my Wireshark columns to display the VLAN ID in the summary view rather than the length. For some reason, Wireshark won't update the column header, but that column outlined in red represents the VLAN ID. Again, despite us matching on VLAN 101, we can only see the C-tags due to a driver issue (probably). At least this confirms that VLAN 101 is actually being used, otherwise the SPAN configuration would be invalid. Trying to match C-tags on that backbone LACP bond, would example, would yield unexpected/undesirable results. In this case, it would yield no results since VLANs 10, 20, and 30 are not used in the provider network at all. Of greatest importance is that traffic to 10.255.30.1 was not captured since the S-tag was 102, which was not candidate for SPAN treatment on ME2.

No.	Time	Source	Destination	Protocol	Length	Info
10	178.544803000	10.255.10.3	10.255.10.1	ICMP	10	Echo (ping) request
10	178.545638000	10.255.10.1	10.255.10.3	ICMP	10	Echo (ping) reply
20	179.986309000	10.255.20.3	10.255.20.1	ICMP	20	Echo (ping) request
20	179.987776000	10.255.20.1	10.255.20.3	ICMP	20	Echo (ping) reply

45.2 Remote SPAN (RSPAN)

RSPAN is very similar to SPAN but adds one capability; the ability to have the SPAN destination be a VLAN. Local SPAN can read traffic from VLANs or ports, but only mirror it to local interfaces. RSPAN can read from VLANs or ports as well, but flood the traffic into a specially-defined RSPAN VLAN. These VLANs still participate in STP and exist in the VLAN database, but have MAC address learning disabled. This makes sense considering the traffic is being forced into a VLAN with destination MAC addresses that don't exist in that VLAN normally.

Continuing from the SPAN lab, we have a Wireshark sniffer plugged into port 24 on ME2. SPAN and RSPAN can coexist, and we will prove it by not removing the SPAN session we configured earlier. The second monitor session will be an RSPAN session, and this ME platform only supports a maximum of 2 sessions. If you try to configure more, the parser prevents it.

```
ME2(config)#monitor session 3 source interface fast0/5
% Platform can support a maximum of 2 source sessions
```

First, we need to define the RSPAN VLANs. We can define as many as we wish, and these VLANs must be transportable across the network between the source and destination. Specifically, we will capture traffic coming from C3 on ME3, put it into an RSPAN VLAN, send it to ME2, and mirror it to our sniffer connected to port 24. Since VLANs 101 – 109 are already defined and allowed on the backbone LACP bond, I will define VLANs 108 and 109 as RSPAN VLANs. We only need one for this test, but I demonstrate that multiple RSPAN VLANs can exist, which is valuable when there are multiple different

remote captures is happening concurrently between different sets of switches. I add names for these VLANs for clarity, but it isn't required. The "remote-span" option is required for these VLANs to operate as RSPAN VLANs; this is the keyword that effectively disables MAC address learning for these VLANs.

```
! ME2 and ME3
vlan 108
  name RSPAN_PRIMARY
  remote-span

vlan 109
  name RSPAN_ALTERNATE
  remote-span
```

Don't be fooled by the MAC address learning show command; this command represents what is explicitly configured with the "mac address-table learning" command. The RSPAN VLANs do actually have MAC learning disabled, so don't use this command to verify as it is misleading.

```
ME2#show mac address-table learning vlan 108
VLAN      Learning Status
----      -
108       yes
```

```
ME2#show mac address-table learning vlan 109
VLAN      Learning Status
----      -
109       yes
```

RSPAN is configured on both the source and destination ends. The source (where traffic is captured) will read from a port or VLAN and mirror traffic into a VLAN. Port 22 is ME3's link to the "customer" switch C3 (really a jumper) and is a dot1q-tunnel port. We can perform captures on this port on ingress only, as an example. I also capture outbound-only on the LACP bond to ME2. In this way, we should see the same packet twice in Wireshark, and ideally with different VLAN encapsulations (incoming on port 22, outgoing on port-channel 1). The traffic is sent into RSPAN VLAN 108, which means 109 is unused.

```
! ME3
monitor session 2 source interface Fa0/22 rx
monitor session 2 source interface Po1 tx
monitor session 2 destination remote vlan 108

ME3#show monitor session 2
Session 2
-----
Type                               : Remote Source Session
Source Ports                       :
  RX Only                         : Fa0/22
  TX Only                         : Po1
```

```
Dest RSPAN VLAN : 108
```

ME2 has to receive traffic from the RSPAN VLAN and copy it to a destination port. Since two monitor sessions cannot share a destination port, we will move the cable from port 24 to port 23 on ME2.

```
ME2(config)#monitor session 2 destination interface fastEthernet 0/24
% Interface(s) Fa0/24 already configured as monitor destinations in other
monsessions
```

The port configuration is very simple and is shown below. I configure encapsulation replicate although it doesn't help us at all. This is the biggest downside of RSPAN; since traffic is carried inside an RSPAN VLAN, the originate encapsulation is always lost.

```
! ME2
interface FastEthernet0/23
  description LOCAL SPAN
  no shutdown

monitor session 2 source remote vlan 108
monitor session 2 destination interface Fa0/23 encapsulation replicate
```

We will send one packet in each customer VRF from ME3 to ME1 as a test. The results are successful but are omitted for brevity.

```
ME3#ping vrf C1 10.255.10.1 repeat 1
ME3#ping vrf C2 10.255.20.1 repeat 1
ME3#ping vrf C3 10.255.30.1 repeat 1
```

Despite having captured on ingress from the customer and egress to the AN on ME3, we see no encapsulation at all. The "length" column is really the VLAN tag, and being blank, there are no tags on these packets. We sent 3 packets yet see four entries. The first two were captured outbound on the LACP bond to ME2. The third packet is shown twice because it was captured once on ingress from C3 and again on egress on the LACP bond. There are no replies captured for any ping because traffic is only being captured in one general direction (ME3 to ME1).

No.	Time	Source	Destination	Protocol	Length	Info
	0.000000000	10.255.10.3	10.255.10.1	ICMP		Echo (ping) request
	5.081038000	10.255.20.3	10.255.20.1	ICMP		Echo (ping) request
	9.248830000	10.255.30.3	10.255.30.1	ICMP		Echo (ping) request
	9.248885000	10.255.30.3	10.255.30.1	ICMP		Echo (ping) request

We quickly check ME2 and ME3 CAM tables to ensure there are no MAC addresses learned in VLAN 108. It is not well-known that this is a workaround for disabling MAC learning in a VLAN in general. On a switch that may not support the "mac address-table learning" command, defining a VLAN as RSPAN will

have a similar effect. This might be useful for EPL or EVPL services where learning all of the MAC addresses across a P2P Ethernet circuit is inefficient for the carrier's switches.

```
ME2#show mac address-table dynamic vlan 108
      Mac Address Table
```

```
-----
Vlan    Mac Address      Type      Ports
----    -
[no output]
```

```
ME3#show mac address-table dynamic vlan 108
      Mac Address Table
```

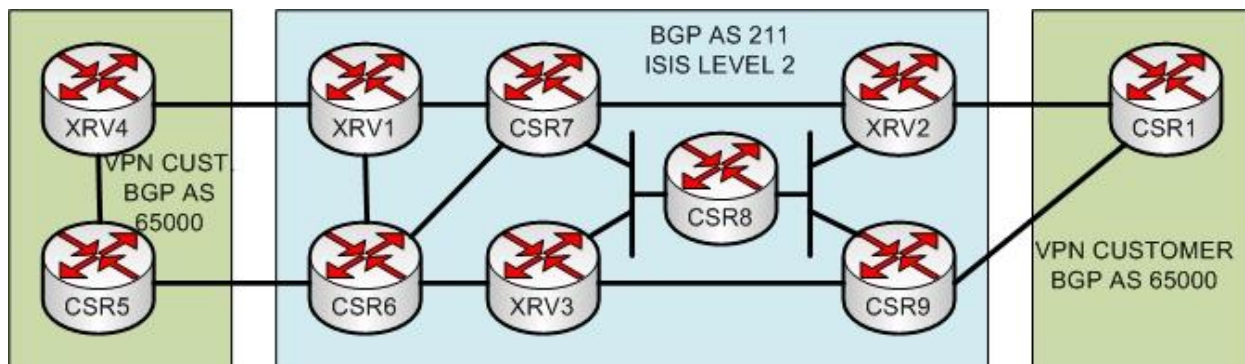
```
-----
Vlan    Mac Address      Type      Ports
----    -
[no output]
```

Additional Reading – Reference configurations "span-rspan"

45.3 Encapsulated RSPAN (ERSPAN)

ERSPAN is meant to behave just like RSPAN except is used when the mirrored traffic must be transported over a layer 3 network. In RSPAN, we identified special RSPAN VLANs with MAC learning disabled so that those VLANs could transport mirrored traffic across a switched network. ERSPAN performs a similar concept but uses GRE encapsulation to transport mirrored traffic over IPv4. Since the feature is newer than SPAN/RSPAN, it is not available in the ME3400 platforms, so I use an existing, large-scale CSR1000v/XRv network to demonstrate it.

This lab is based on the MPLS IP/MTU minor features from earlier. The diagram is the same but there are some minor changes with the configuration. Familiarize yourself with this network topology by first reading the aforementioned chapter. The existing network is very involved with many advanced features configured.



We will pretend that CSR10 has a sniffer host connected to its Gig3 port, but this is unnecessary since we just connect CSR9's Gig3 to a bogus port-group in ESXi, then use outbound EPC locally on CSR9 to see

mirrored traffic. For capturing traffic responsibly, I delete CSR7's Gig2.567 interface and use Gig3 instead. This is plugged into an access-port (untagged port-group) on the same vSwitch as the general-purpose trunk on VLAN 3567. Thus, CSR6 and CSR7 have reachability over this link, but CSR7 uses a dedicated "physical" interface. Below is a quick snapshot of the IS-IS neighbors on CSR7, and we can see CSR6 is reachable via Gig3. The configuration is not shown as it is identical to the Gig2.567 configuration from the MPLS IP/MTU lab.

```
R7#show isis neighbors
```

```
Tag 211:
```

System Id	Type	Interface	IP Address	State	Holdtime	Circuit Id
R6	L2	Gi3	211.6.7.6	UP	26	01
R8	L2	Gi2.578	211.7.8.8	UP	9	R8.02
XRv1	L2	Gi2.571	211.7.11.11	UP	21	00
XRv2	L2	Gi2.572	211.7.12.12	UP	25	00
XRv3	L2	Gi2.578	211.7.8.13	UP	21	R8.02

On CSR9, all we need to do is no-shutdown the port. I add a description for clarity; this simple configuration is similar to SPAN and RSPAN where the destination port doesn't require anything fancy.

```
! CSR9
interface GigabitEthernet3
  description ERSPAN DESTINATION TEST PORT
  no shutdown
```

Just like with RSPAN, we must configure the source and destination. An ERSPAN source is the point where traffic is captured, and the destination is where it is sent (destination terminates a session). We will configure the destination first since it is less involved. We define the ERSPAN session and specify the proper type. We can select a set of destination interfaces as well; in this case, the test port is used. Each ERSPAN session as a session ID to demultiplex the ERSPAN sessions when one device has multiple sessions configured between the same set of IP addresses. The IP address on the destination session is the tunnel destination; that is, it is on this IP address that the router expects to receive ERSPAN-mirrored traffic. Sessions require "no-shutdown" as well.

```
! CSR9
monitor session 1 type erspan-destination
  no shutdown
  description LINK-BETWEEN-CSR6-AND-CSR7
  destination interface Gi3
  source
    erspan-id 1023
    ip address 211.0.0.9
```

The show command for this basically just echoes the configuration details. We can verify that everything is correct; also, note the MTU as we will be verifying this later. ERSPAN is accounting for 36 bytes of encapsulation.

```
R9#show monitor session 1
```

```
Session 1
```

```
-----
```

```
Type : ERSPAN Destination Session
Status : Admin Enabled
Description : LINK-BETWEEN-CSR6-AND-CSR7
Destination Ports : Gi3
Source IP Address : 211.0.0.9
Source ERSPAN ID : 1023
MTU : 1464
```

The ERSPAN source configuration looks and feels like a GRE tunnel. We configure the origin IP address (source) and the session IP address (destination). The ERSPAN ID has to match as well, and we can optionally specify the session VRF, TTL, and DSCP values. These options are equivalent to the features normally available on a GRE tunnel interface. I adjust some of these parameters for variety. The source interface is Gig3, which is connected to CSR6.

```
! CSR7
description CSR6-CSR7-TO-CSR9
source interface Gi3
destination
  erspan-id 1023
  ip address 211.0.0.9
  ip dscp 16
  ip ttl 8
  origin ip address 211.0.0.7
```

Because the ERSPAN source is more involved, we can verify it with the detail command. Towards the bottom, we can verify our minor options with this command. The MTU is computed at 1464 as well.

```
R7#show monitor session 1 detail
```

```
Session 1
```

```
-----
```

```
Type : ERSPAN Source Session
Status : Admin Enabled
Description : CSR6-CSR7-TO-CSR9
Source Ports :
  RX Only : None
  TX Only : None
  Both : Gi3
Source VLANs :
  RX Only : None
  TX Only : None
  Both : None
Source EFPs :
  RX Only : None
```



```

TX Only          : None
Both             : None
Destination Ports : None
Filter VLANs     : None
Source IP Address : None
Source IP VRF    : None
Source ERSPAN ID : None
Destination IP Address : 211.0.0.9
Destination IP VRF : None
MTU              : 1464
Destination ERSPAN ID : 1023
Origin IP Address : 211.0.0.7
IP QOS DSCP      : 16
IP TTL           : 8

```

Looking at CSR9's capture, we can see a lot of information, even within the first few seconds of the capture. All of the traffic is source from either CSR6 or CSR7 on the link we are capturing. CSR9 is receiving the GRE packets, de-encapsulating the traffic, and sending it out of port Gig3. Most of this idle traffic is likely IS-IS, LDP, and possibly BGP.

```
R9#show monitor capture CAP buffer brief
```

#	size	timestamp	source	destination	protocol
0	93	0.000000	211.0.0.6	-> 211.0.0.7	TCP
1	74	0.200978	211.0.0.7	-> 211.0.0.6	TCP
2	104	0.380977	00:50:56:A9:DE:0D	-> 09:00:2B:00:00:05	LLC
3	76	1.275987	211.6.7.7	-> 224.0.0.2	UDP
4	93	1.605016	211.0.0.7	-> 211.0.0.6	TCP
5	76	1.805003	211.6.7.6	-> 224.0.0.2	UDP
6	74	1.807002	211.0.0.6	-> 211.0.0.7	TCP
7	60	2.652011	00:50:56:A9:25:E6	-> 09:00:2B:00:00:04	LLC
8	72	4.695008	211.0.0.7	-> 211.0.0.6	TCP

CSR7 has three EMCP paths to reach 211.0.0.9 which is the tunnel destination. According to the parser, traffic should be sent to CSR8 using label 8016.

```
R7#show ip cef 211.0.0.9/32
```

```
211.0.0.9/32
```

```

  nexthop 211.7.8.8 GigabitEthernet2.578 label 8016
  nexthop 211.7.8.13 GigabitEthernet2.578 label 93002
  nexthop 211.7.12.12 GigabitEthernet2.572 label 92004

```

```
R7#show ip cef exact-route 211.0.0.7 211.0.0.9
```

```

211.0.0.7 -> 211.0.0.9 => label 8016 TAG adj out of GigabitEthernet2.578, addr
211.7.8.8

```

However, a quick EPC outbound on CSR7 reveals that traffic is being sent to XRv3. This is interesting as I have no explanation as to why this would happen. If CSR7 is encapsulating traffic locally from 211.0.0.7 to 211.0.0.9 and not consulting the FIB, I am not sure what table it is consulting. Nonetheless, we can examine the EPC below, and we know it is going to XRv3 due to the destination MAC address and label value of 93002 (yellow). The IP header has TOS 64, which is DSCP 16 in cyan. The TTL is 8 and the IP protocol is 47 (pink). In green are the tunnel source and destination addresses, which are 211.0.0.7 and 211.0.0.9, respectively. A new packet type has been identified for ERSPAN, and it is 0x88BE (yellow shows the entire GRE header) Also notice that sequence numbers are enabled for the ERSPAN session per the 0x1 flag in the GRE header (this is why the GRE header is 8 bytes). Since GRE has to specify what it is carrying, this new field is important. Inside the ERSPAN payload, some additional information is carried, such as the ERSPAN ID of 1023 (grey). Later, the captured layer 2 frame is shown; in this case, the destination MAC address is highlighted in cyan and this was probably an LDP hello with destination IP address 224.0.0.2. We can tell this based on the MAC address.

```
R7#show ip arp 211.7.8.13
Protocol  Address          Age (min)  Hardware Addr  Type   Interface
Internet  211.7.8.13       220       0050.56a9.ea54  ARPA   Gig2.578

R7#show monitor cap CAP buffer detailed
 0  134      0.000000  00:50:56:A9:EA:77 -> 00:50:56:A9:EA:54 MPLS unicast
0000:  005056A9 EA540050 56A9EA77 81000DFA .PV..T.PV..w....
0010:  884716B4 A5084540 00700000 4000082F .G....E@.p..@../
0020:  CC0ED300 0007D300 00091000 88BE0000 .....
0030:  4C111001 03FF0000 00000100 5E000002 L.....^....
```

There isn't much more to ERSPAN than this. The packet breakdown shown above is pretty self-explanatory, though I admit I guessed the ERSPAN-ID would have been the GRE tunnel key. Instead, ERSPAN adds 8-bytes of additional overhead to carry important session information used by the ERSPAN process. Now, we can see why MTU 1464 makes sense: 1500 – 20 IP – 8 GRE – 8 ERSPAN = 1464 bytes.

Additional Reading – Reference configurations "erspan"

46. Describe, implement, and troubleshoot Netflow and IPFIX

FNF is the successor to ordinary Netflow that is used primarily to measure, aggregate, and export traffic in the network. It can be used for security, billing, capacity planning, or general network management purposes. I personally use it often for troubleshooting; you can apply it quickly to an interface and see the fields of your choice. It is similar to EPC in this way, except it is part of the configuration and thus is more permanent. FNF is broken into three main components:

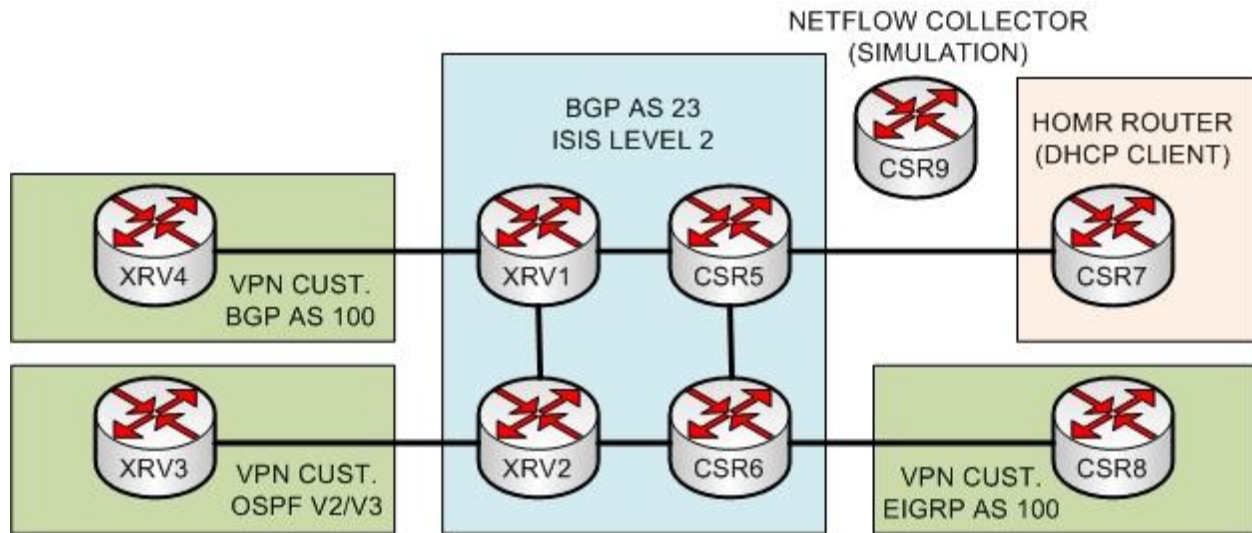
- a. **Record:** This determines what is being matched, somewhat like a class-map in MQC. Individual fields are specified here that comprise the entry. Fields can be identified as “key” (“match” command) or “nonkey” (“collect” command). The difference is that key fields are attributes that identify a flow; if two packets arrive on a router and all N of the key fields are identical, the two

packets are part of the same flow. If any key field differs, the packets are considered different flows. Nonkey fields are allowed to be different within the same flow. For example, a telnet session from A to B will likely have the same IP protocol, TOS, source/destination IP addresses and ports for its journey. Even if these packets are received by remote routers in different input interfaces (which are often nonkey), the traffic is considered one flow. Receiving traffic on different interfaces might be due to load-sharing schemes in the network core. A different source port, for example, would probably indicate a new telnet connection between the same set of hosts, which is legitimately a separate flow.

- b. **Exporter:** Defines the mechanism by which FNF records are sent to a collector. This includes source/destination IP address and port information, Netflow version, and other parameters that should match what the collector is expecting.
- c. **Monitor:** The monitor object contains a record (required) and an exporter (optional). The monitor also specifies the size of the cache and various timeout values. Two important timeout values are the “inactive” and “active” timers. Inactive starts counting from the moment a flow stops being seen by the device; this is the moment that the flow is considered inactive. The flow is purged from the cache after this timeout expires. The active timer is the maximum length of time a single flow is kept in the cache. If the length of the flow exceeds this time, the record is exported and purged, and then a new record starts to capture all packets after the active timer’s expiration. There are different cache types also, most of which are self-explanatory, that determine when flows are purged from the cache.

One critical change in v9 that is different from v5 is that it is template-based. Users do not modify templates directly, but templates are created based off of the information the user wishes to export. For example, a record may match 4 fields. Those fields may be variable in length; IPv4 addresses are 4 bytes, IPv6 addresses are 16, VLAN IDs are 2 (not counting the 0x8100 ethertype), etc. The exporter assembles all this information into a template so that records received from the exporter are intelligible and arranged in a deterministic fashion. Every template has an ID which is used internally to the Netflow process to identify a template.

The network diagram is shown below and is very similar to the alarm/capture test. CSR9 is an OOBM host used for Netflow collection while the majority of the Netflow configuration will be done on CSR5.



Note: Netflow does not appear to be supported at all in XRv versions 5.3.0 or 5.3.1.

```
RP/0/0/CPU0:XRv1(config)#flow?
flowspec
```

Additional Reading – Reference configurations "netflow"

46.1 Flexible Netflow (FNF)

FNF is supported on a per-interface basis for IPv4, IPv6, and MPLS traffic. First, we will create a basic IPv4 policy with all key fields on CSR5. This will capture the specified information from each IPv4 packet received from CSR7.

```
! CSR5
flow record FLOW_RECORD_IPV4
 match ipv4 dscp
 match ipv4 protocol
 match ipv4 source address
 match ipv4 destination address
 match transport source-port
 match transport destination-port
 match interface input
```

For now, we will not configure an exporter. Wrapping this record inside a monitor is the next step. We will tune the active/inactive timers as well as the cache size for variety. Using an inactive timer of 60 seconds gives us more time to see flows in the cache before they are quickly removed; this is very useful for troubleshooting.

```
! CSR5
flow monitor FLOW_MONITOR_IPV4
 cache timeout inactive 60
```

```
cache timeout active 300
cache entries 2048
record FLOW_RECORD_IPV4
```

The third and final step is to apply this to the interface facing CSR7. You can use this to specify unicast or multicast as well, plus apply any samplers (discussed later). At present, we see no records in the cache.

```
! CSR5
interface GigabitEthernet2.557
 ip flow monitor FLOW_MONITOR_IPV4 input
```

```
R5#show flow monitor FLOW_MONITOR_IPV4 cache
Cache type:                Normal (Platform cache)
Cache size:                2048
Current entries:           0
Flows added:               0
Flows aged:               0
```

There are no cache entries to display.

Now, we will send some ping/telnet traffic from CSR5 to XRv4 (not shown). The inactive timer gives us 60 seconds from the time the flow ends before it is purged from the cache, giving me time to view the results. The default is only 15 seconds which makes FNF less valuable for troubleshooting unless you tune it. There are three formats in which to view the information: record, table, and CSV. “Record” prints each field line-by-line, which is the most verbose. “Table” prints them arrayed horizontally with each row being an entry, which is my personal favorite. Beware that if you have many fields (more than ~6), table is a poor choice since it will wrap around the terminal screen. “CSV” is the least verbose and is good for pasting into a spreadsheet that understands CSV lists. All three versions are shown below; “Record” is the default and is shown first.

```
R5#show flow monitor FLOW_MONITOR_IPV4 cache
Cache type:                Normal (Platform cache)
Cache size:                2048
Current entries:           2
High Watermark:           2

Flows added:               2
Flows aged:               0
```

```
IPV4 SOURCE ADDRESS:      10.5.7.105
IPV4 DESTINATION ADDRESS: 10.14.14.14
TRNS SOURCE PORT:         0
TRNS DESTINATION PORT:    2048
INTERFACE INPUT:          Gi2.557
IP DSCP:                   0x00
IP PROTOCOL:              1
```

```

IPV4 SOURCE ADDRESS:      10.5.7.105
IPV4 DESTINATION ADDRESS: 10.14.14.14
TRNS SOURCE PORT:         4096
TRNS DESTINATION PORT:    23
INTERFACE INPUT:          Gi2.557
IP DSCP:                  0x30
IP PROTOCOL:              6

```

```

R5#show flow monitor FLOW_MONITOR_IPV4 cache format table
[snip]

```

IPV4 SRC ADDR	IPV4 DST ADDR	TRNS SRC PORT	TRNS DST PORT	INTF INPUT	IP DSCP	IP PROT
10.5.7.105	10.14.14.14	0	2048	Gi2.557	0x00	1
10.5.7.105	10.14.14.14	4096	23	Gi2.557	0x30	6

```

R5#show flow monitor FLOW_MONITOR_IPV4 cache format csv
[snip]

```

```

IPV4 SRC ADDR,IPV4 DST ADDR,TRNS SRC PORT,TRNS DST PORT,INTF INPUT,IP DSCP,IP PROT
10.5.7.105,10.14.14.14,0,2048,Gi2.557,0x00,1
10.5.7.105,10.14.14.14,4096,23,Gi2.557,0x30,6

```

If we come back a few minutes later, we can see both flows are gone due to aging. The cache shows 2 aged flows and the reason why.

```

R5#show flow monitor FLOW_MONITOR_IPV4 cache
Cache type:                      Normal (Platform cache)
Cache size:                      2048
Current entries:                 0
High Watermark:                 2

Flows added:                    2
Flows aged:                     2
- Inactive timeout ( 60 secs)  2

```

There are no cache entries to display.

Next, we will configure IPv6 FNF. It is identical to IPv4 except we can include IPv6 specific fields like flow-label if we choose. I will introduce the FNF exporter here. We will create a policy to account for the IPv6 flow label (key), and some layer 2 information as well (nonkey). We can use the layer 2 matches for IPv4, IPv6, or MPLS. Because it is a lower OSI layer, the data-link fields apply to any layer 3 protocol. We will remove the IPv6 DSCP and IP protocol matches for variety.

```

! CSR5
flow record FLOW_RECORD_IPV6
match ipv6 flow-label
match ipv6 source address
match ipv6 destination address
match transport source-port
match transport destination-port

```

```

match interface input
collect datalink source-vlan-id
collect datalink mac source address input

```

We will create an exporter to send these IPv6 records we collect to the FNF collector, CSR9. I can specify the TOS and TTL of these export messages also; I increase the TOS to CS6 and decrease the TTL to 1 since CSR9 is one hop away. This TTL adjustment is for security reasons. The destination is in the MGMT VRF and I can explicitly specify a source, although in this case it doesn't matter. The "option vrf-table" command allows FNF to capture the VRF table information from which the packets were received, which is VRF SHR. This could be useful in an SP environment on a PE router. By default, FNFv9 is used with UDP port 9995. I will name this exporter with a string containing "COMMON" because I can reuse it for other monitors; chances are you don't have a different collector for each traffic type (IPv4, IPv6, and MPLS).

```

! CSR5
flow exporter FLOW_EXPORTER_COMMON_V9
 destination 10.31.5.209 vrf MGMT
 source GigabitEthernet1
 dscp 48
 ttl 1
 option vrf-table

```

We can verify the configuration of the exporter using the command below. All of this is information that can be configured under the exporter, and seeing it this way also reveals the defaults. The least common known thing about FNF is the default transport is UDP 9995; ensure any data-plane filters permit UDP 9995 from network devices to the FNF collector.

```

R5#show flow exporter FLOW_EXPORTER_COMMON_V9
Flow Exporter FLOW_EXPORTER_COMMON_V9:
  Description:                User defined
  Export protocol:             NetFlow Version 9
  Transport Configuration:
    Destination IP address:    10.31.5.209
    VRF label:                 MGMT
    Source IP address:         10.31.5.205
    Source Interface:          GigabitEthernet1
    Transport Protocol:        UDP
    Destination Port:          9995
    Source Port:               54907
    DSCP:                      0x30
    TTL:                       1
    Output Features:           Used
  Options Configuration:
    vrf-table (timeout 600 seconds)

```

The monitor for IPv6 is slightly different. We use the IPv6 record with the COMMON exporter defined above, where IPv4 does not have an exporter currently. We will set the cache type to permanent which

means entries are never purged, so tuning the timers is not necessary. We also apply this as an IPv6 flow monitor to the interface facing CSR7. The monitor also specifies that only unicast traffic is candidate for capture, which is good on links where IGP is running (don't need to capture LL multicast, etc). Currently there is nothing in the cache.

```
! CSR5
flow monitor FLOW_MONITOR_IPV6
  exporter FLOW_EXPORTER_COMMON_V9
  cache type permanent
  record FLOW_RECORD_IPV6

! CSR5
interface GigabitEthernet2.557
  ipv6 flow monitor FLOW_MONITOR_IPV6 unicast input
```

```
R5#show flow monitor FLOW_MONITOR_IPV6 cache
Cache type: Permanent (Platform cache)
Cache size: 200000
Current entries: 0

Flows added: 0
Updates sent ( 1800 secs) 0
```

There are no cache entries to display.

Next, CSR7 will send ping/telnet traffic to XRv14 again using IPv6 addresses (not shown). In order to make it readable, I will use the record format, although table/CSV are also valid. Because we are no longer capturing DSCP or IP protocol, these flows almost look the same. The only differentiator at this point is source/destination port. Also notice that the data-link fields are in lower-case while all other fields are in upper-case; this denotes nonkey versus key fields.

```
R5#show flow monitor FLOW_MONITOR_IPV6 cache format record
Cache type: Permanent (Platform cache)
Cache size: 200000
Current entries: 0

Flows added: 0
Updates sent ( 1800 secs) 0

IPV6 FLOW LABEL: 0
IPV6 SOURCE ADDRESS: 2001:10:5:7::7
IPV6 DESTINATION ADDRESS: ::10:14:14:14
TRNS SOURCE PORT: 15668
TRNS DESTINATION PORT: 23
INTERFACE INPUT: Gi2.557
layer 2 source vlan id: 3557
datalink mac source address input: 0050.56A9.EA77
```



```

IPV6 FLOW LABEL:                0
IPV6 SOURCE ADDRESS:            2001:10:5:7::7
IPV6 DESTINATION ADDRESS:       ::10:14:14:14
TRNS SOURCE PORT:               0
TRNS DESTINATION PORT:          32768
INTERFACE INPUT:                Gi2.557
layer 2 source vlan id:         3557
datalink mac source address input: 0050.56A9.EA77

```

These entries will never age out. Over time, the IPv6 ND packets will enter the cache also. Presently, there are 6 flows in the cache. I show them in CSV format below just you can count the 6 rows.

```

R5#show flow monitor FLOW_MONITOR_IPV6 cache format csv
IPV6 FLOW LABEL,IPV6 SRC ADDR,IPV6 DST ADDR,TRNS SRC PORT,TRNS DST PORT,INTF
INPUT,l2 src vlan id,datalink mac src addr input
0,FE80::7,FF02::1,0,34304,Gi2.557,3557,0050.56A9.EA77
0,2001:10:5:7::7,::10:14:14:14,15668,23,Gi2.557,3557,0050.56A9.EA77
0,FE80::7,FE80::5,0,34560,Gi2.557,3557,0050.56A9.EA77
0,2001:10:5:7::7,::10:14:14:14,0,32768,Gi2.557,3557,0050.56A9.EA77
0,2001:10:5:7::7,FE80::5,0,34816,Gi2.557,3557,0050.56A9.EA77
0,FE80::7,FE80::5,0,34816,Gi2.557,3557,0050.56A9.EA77

```

We can manually clear any cache, regardless of type, using manual commands. When this occurs, the records are consolidated into UDP 9995 packets and sent to the collector. The “timeout update” parameter under the flow monitor controls this export interval, which is 30 minutes (1800 seconds) by default.

```
R5#clear flow monitor FLOW_MONITOR_IPV6 cache
```

```

R5#show flow monitor FLOW_MONITOR_IPV6 cache
Cache type:                Permanent (Platform cache)
Cache size:                200000
Current entries:           6
High Watermark:           6
Flows added:              6
Updates sent                ( 1800 secs) 0

```

```
There are no cache entries to display.
```

We can see the statistics of packets (4) and records (6) sent to the collector. Records are typically sent to the collector when they expire, but in permanent caches, that never happens. This has a separate timer because the information still remains in the FNF device, but the collector just receives updates, not purged records.

```
R5#show flow exporter FLOW_EXPORTER_COMMON_V9 statistics
```

```

Flow Exporter FLOW_EXPORTER_COMMON_V9:
  Packet send statistics (last cleared 00:16:10 ago):
    Successfully sent: 4 (622 bytes)

Client send statistics:
  Client: Option options vrf-id-name-table
    Records added: 6
    - sent: 6
    Bytes added: 216
    - sent: 216

  Client: Flow Monitor FLOW_MONITOR_IPV6
    Records added: 0
    Bytes added: 0

```

Last, we will examine MPLS FNF. The general logic is the same, except we won't be matching IPv4/v6 headers inside the FNF record, we will be matching MPLS label information. You can match up to 6 labels deep, but in our architecture, even with TE-FRR, we would have a maximum of 3. The labels in the stack will be the key fields, and I will collect packet and byte counts as nonkey fields. This means that we can see how many packets/bytes are identified with each flow.

```

! CSR5
flow record FLOW_RECORD_MPLS
  match mpls label 1 details
  match mpls label 2 details
  match mpls label 3 details
  collect counter bytes
  collect counter packets

```

There are two types of caches we have not tested yet. The "immediate" type ages out entries as soon as they are seen; for our purposes, this would not be helpful since we don't have a FNF collector where we can see the information. All other timers do not apply when using this mode. We will use the new "synchronized" cache type for our MPLS packets. This feature was specifically introduced to protect against "export storms". The synchronized interval is basically a timer that exports records when the timer expires, but in large caches, we may want to spread those exports over time to as not to storm the Netflow collector (NFC). This is especially true in networks where many devices are synchronized to export records at the same time. We will configure a 30 second interval with spreading over the first 10 seconds of the next interval.

```

! CSR5
flow monitor FLOW_MONITOR_MPLS
  exporter FLOW_EXPORTER_COMMON_V9
  cache timeout synchronized 30 export-spread 10
  cache type synchronized
  record FLOW_RECORD_MPLS
  ip flow monitor FLOW_MONITOR_IPV4 input

```

```
ipv6 flow monitor FLOW_MONITOR_IPV6 input
```

We will also configure a basic sampler. This collects 1 out of N packets based on some self-explanatory algorithm (fixed, random, etc) and adds them to the cache. This is useful for conserving resources on the router as now it only needs to care about 1/nth of the traffic arriving per flow. The sampler is applied at the interface level when we apply the monitor; it is not nested inside the FNF constructs defined earlier.

```
! CSR5
sampler FLOW_SAMPLER_MPLS
mode random 1 out-of 4
```

I illustrate one significant drawback of FNF; when you make an error or otherwise need to make a change, you have to undo many things. When I try to apply this policy to the interface facing CSR6, I am notified that I am missing a key field in the record to support export spreading. To correct it, I must remove the monitor from the interface and the record from the flow, make the correction, and reconfigure the linkages.

```
! CSR5
interface GigabitEthernet2.556
 mpls flow monitor FLOW_MONITOR_MPLS sampler FLOW_SAMPLER_MPLS output

% Flow Monitor: Flow Monitor 'FLOW_MONITOR_MPLS' [MONITOR:FLOW_MONITOR_MPLS]
Spread interval requires 'timestamp absolute monitor start' field to be
configured as a key field in the record.
```

After making the corrections, I apply the policy to both MPLS interfaces on CSR5. The cache is currently empty.

```
! CSR5
interface GigabitEthernet2.556
 mpls flow monitor FLOW_MONITOR_MPLS sampler FLOW_SAMPLER_MPLS output
```

```
R5#show flow monitor FLOW_MONITOR_MPLS cache
Cache type:                               Synchronized (Platform cache)
Cache size:                               200000
Current entries:                           0
High Watermark:                           0

Flows added:                              0
Flows aged:                               0
Synchronized timeout (secs):               30
```

There are no cache entries to display.

I open up two telnet sessions from CSR7. One goes to XRv4 and one goes to XRv3. One uses IPv4 and one uses IPv6, but that is totally irrelevant because we are only capturing MPLS-encapsulated packets.

Using the record format, I can see both flows. We will examine the label stacks shortly. Notice that the byte/packet counters are quite low; this is because we are only sampling 25% of the traffic.

```
R7#telnet 10.13.13.13 /source loopback0
```

```
R7#telnet ::10:14:14:14
```

```
R5#show flow monitor FLOW_MONITOR_MPLS cache
```

```
Cache type:                               Synchronized (Platform cache)
Cache size:                               200000
Current entries:                           2
High Watermark:                           2

Flows added:                               6
Flows aged:                               4
Synchronized timeout (secs):              30
```

```
MPLS LABEL 1:          6000   /6
MPLS LABEL 2:          92005  /6*
MPLS LABEL 3:           0     /0
TIMESTAMP MONITOR START: 22:49:00.000
counter bytes:          891
counter packets:        13
```

```
MPLS LABEL 1:          91005  /6*
MPLS LABEL 2:           0     /0
MPLS LABEL 3:           0     /0
TIMESTAMP MONITOR START: 22:49:00.000
counter bytes:          749
counter packets:         9
```

The first flow has a two label stack. The 92000-range of labels are allocated by XRv2, so we assume this is the VPNv4 label for 10.13.13.13. The label 6000 is the LDP label from CSR6 to reach XRv2's loopback 23.0.0.12/32.

```
R5#show bgp vpnv4 unicast vrf SHR 10.13.13.13/32 | include label
      mpls labels in/out nlabel/92005
```

```
R5#show mpls ldp bindings 23.0.0.12 32 neighbor 23.0.0.6
lib entry: 23.0.0.12/32, rev 34
remote binding: lsr: 23.0.0.6:0, label: 6000
```

We can confirm this by checking the FIB for the exact CEF match as well. Since CSR5 can load-share between XRv1 and CSR6, we prove that CSR6 was indeed selected. CSR7's post-NAT IP address can be gleaned from the DHCP binding database on CSR5 (CSR7 DHCP client-id is 0BADBEEF from the EEM lab).

```
R5#show ip dhcp binding | include beef
10.5.7.105      0bad.beef      MON 22 2015 11:01 PM      Automatic  Active      Gig2.557
```

```
R5#show ip cef vrf SHR exact-route 10.5.7.105 10.13.13.13
10.5.7.105 -> 10.13.13.13 => label 92005 label 6000*TAG adj out of
GigabitEthernet2.556, addr 23.5.6.6
```

The second flow has a single label from XRv1, presumably the VPNv6 label. There is no transport label due to PHP as the routers are directly connected (CSR5 is the penultimate hop).

```
R5#show bgp vpnv6 unicast vrf SHR ::10:14:14:14/128 | include label
mpls labels in/out nolabel/91005
```

```
R5#show mpls ldp bindings 23.0.0.11 32 neighbor 23.0.0.11
lib entry: 23.0.0.11/32, rev 6
remote binding: lsr: 23.0.0.11:0, label: imp-null
```

```
R5#show ipv6 cef vrf SHR ::10:14:14:14/128
::10:14:14:14/128
nexthop 23.5.11.11 GigabitEthernet2.551 label 91005
```

In both flows, also notice the MPLS EXP set to 6, which is correct for telnet traffic. The asterisk following some labels likely means that the S-bit is set, but I have asked Cisco to confirm this with no answer. It is not well documented. I have personally witnessed the asterisk on top-most labels in other relatively new versions as well. Below I show the final result of the cache after all entries have been exported per the synchronization timer. All of the flows have aged and been exported, and presumably spread over the first ten seconds of the next interval.

```
R5#show flow monitor FLOW_MONITOR_MPLS cache
Cache type:                               Synchronized (Platform cache)
Cache size:                               200000
Current entries:                           0
High Watermark:                           2

Flows added:                              16
Flows aged:                               16
Synchronized timeout (secs):              30
```

There are no cache entries to display.

Looking at the details of the exporter templates, we can see what the FNF records will look like. For each flow monitor (assuming it has a different record), a new template is created to fit the recorded components into a datagram of sorts. The second and third tables display the templates for IPv6 and MPLS respectfully. The tables also show the length of each field in bytes, plus the offset from the beginning of the recorded data. The record size is shown in the header data and is also measured in bytes. There is also a table for the “options” we enabled which allows the input VRF information to be included in the exported packets.

R5#show flow exporter FLOW_EXPORTER_COMMON_V9 templates

Flow Exporter FLOW_EXPORTER_COMMON_V9:

Client: Option options vrf-id-name-table

Exporter Format: NetFlow Version 9

Template ID : 257

Source ID : 6

Record Size : 40

Template layout

Field	Type	Offset	Size
v9-scope system	1	0	4
routing vrf input	234	4	4
routing vrf name	236	8	32

Client: Flow Monitor FLOW_MONITOR_IPV6

Exporter Format: NetFlow Version 9

Template ID : 257

Source ID : 256

Record Size : 51

Template layout

Field	Type	Offset	Size
ipv6 flow-label	31	0	3
ipv6 source address	27	3	16
ipv6 destination address	28	19	16
transport source-port	7	35	2
transport destination-port	11	37	2
datalink source-vlan-id	58	39	2
datalink mac source address input	56	41	6
interface input snmp	10	47	4

Client: Flow Monitor FLOW_MONITOR_MPLS

Exporter Format: NetFlow Version 9

Template ID : 258

Source ID : 512

Record Size : 25

Template layout

Field	Type	Offset	Size
mpls label 1 details	70	0	3
mpls label 2 details	71	3	3
mpls label 3 details	72	6	3
timestamp absolute monitoring-interval	359	9	8
counter bytes	1	17	4

counter packets		2		21		4	

A final verification command that is very powerful shows us the policies applied to each interface. We can see the monitor name, direction, address-family, and any kind of sampler applied. You can apply multiple monitors on the same interface in both directions, which I demonstrate in some capacity below.

```
R5#show flow interface
Interface GigabitEthernet2.551
  FNF:  monitor:      FLOW_MONITOR_MPLS
        direction:    Output
        traffic(mpls): sampler FLOW_SAMPLER_MPLS
Interface GigabitEthernet2.556
  FNF:  monitor:      FLOW_MONITOR_MPLS
        direction:    Output
        traffic(mpls): sampler FLOW_SAMPLER_MPLS
Interface GigabitEthernet2.557
  FNF:  monitor:      FLOW_MONITOR_IPV4
        direction:    Input
        traffic(ip):   on
  FNF:  monitor:      FLOW_MONITOR_IPV6
        direction:    Input
        traffic(ipv6): on
```

46.2 IPFIX

IP Flow Information eXport is meant to standardize the mechanism by which flows are exported. Although Cisco allowed many other vendors to implement its Netflow solution, IPFIX adds many improvements over the original specification. Specifically, it is designed to work in secure environments using Datagram Transport Layer Security (DTLS) connectivity between collector and measurement devices versus ordinary UDP. It also calls for use of Stream Control Transmission Protocol (SCTP), Partially-Reliable SCTP (PR-SCTP), or TCP for transport to the collector; the best IPFIX designs are supported to use reliable transport.

IPFIX is defined in RFC 5470 and introduces some new terms to Netflow. Existing definitions like traffic flow, flow key, record, and others were defined earlier and are not repeated. New terms are below.

Observation Point: A place in the network where packets can be observed. It could be a shared medium (LAN), single port on a device, or a set of ports on many devices. This would be the place where Netflow is actually configured. The concept of OP's is hierarchical as a line-card could be one OP where each port on that LC are also OPs.

Observation Domain: This is the largest set of OPs available (hierarchically, the largest collection); thus, an observation domain consists of many OPs. The line-card itself might be the OD if the ports are all OPs,

and each OD has a domain ID which uniquely identifies the domain. In this way, exported records have traceability.

IPFIX is broken down into three main processes which occur sequentially:

1. **Metering process:** The inputs for this process include packet headers and other relevant Netflow data at an OP, along with any packet treatment at the OP (for example, a routing decision that yields a specific outgoing interface, assuming that field was captured). The output of this process is a flow record.
2. **Exporting Process:** This process sends flow records to one or more collecting processes. The devices that host exporting processes are called “exporters”. This is where transport details, such as ports and protocols, are specified.
3. **Collecting Process:** This process receives flow records from one or more exporting processes. The collecting process might take some action based on the records, store them, or simply discard them. The RFC does not specify an action. A device that hosts collecting processes is called a “collector” and would be the target of the aforementioned export sessions.

In terms of configuration, XE only appears to support UDP at this time and none of the recommended security features. Below, we set the export-protocol to IPFIX (sometimes called Netflow v10) and adjust the UDP port for variety.

```
! CSR5
flow exporter FLOW_EXPORTER_IPFIX
 destination 10.31.5.209 vrf MGMT
 source GigabitEthernet1
 transport udp 9996
 export-protocol ipfix

R5#show flow exporter FLOW_EXPORTER_IPFIX
Flow Exporter FLOW_EXPORTER_IPFIX:
  Description:                User defined
  Export protocol:             IPFIX (Version 10)
  Transport Configuration:
    Destination IP address:    10.31.5.209
    VRF label:                 MGMT
    Source IP address:         10.31.5.205
    Source Interface:          GigabitEthernet1
    Transport Protocol:        UDP
    Destination Port:          9996
    Source Port:               54711
    DSCP:                      0x0
    TTL:                       255
    Output Features:           Used
```


We apply this exporter to the IPv4 monitor, but in order to do it, we must remove the flow monitor from all interfaces first. After reapplying it, we can ensure it was added properly (removal/re-add not shown).

```
R5#show flow interface gig2.557 | section IPV4
```

```
  FNF:  monitor:      FLOW_MONITOR_IPV4
        direction:    Input
        traffic(ip):   on
```

```
R5#show flow monitor FLOW_MONITOR_IPV4 | include Exporter
```

```
  Flow Exporter:      FLOW_EXPORTER_IPFIX
```

We can also look at the dynamically created templates for this exporter. The template ID is 259 and there is a new column called “Ent.ID”. This was not present in the v9 templates. I assume this is for future capability; RFC 6759 suggests this is an “Enterprise ID” used for layer 7 flow identification.

```
R5#show flow exporter FLOW_EXPORTER_IPFIX templates
```

```
Flow Exporter FLOW_EXPORTER_IPFIX:
```

```
Client: Flow Monitor FLOW_MONITOR_IPV4
```

```
  Exporter Format: IPFIX (Version 10)
```

```
  Template ID      : 259
```

```
  Source ID        : 768
```

```
  Record Size      : 18
```

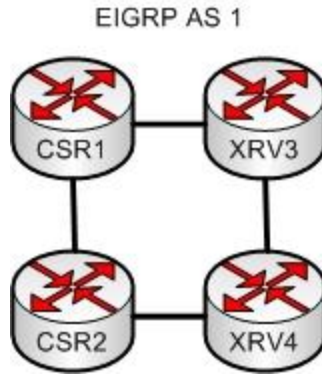
```
  Template layout
```

Field	ID	Ent.ID	Offset	Size
ipv4 source address	8		0	4
ipv4 destination address	12		4	4
ip dscp	195		8	1
ip protocol	4		9	1
transport source-port	7		10	2
transport destination-port	11		12	2
interface input snmp	10		14	4

47. Describe, implement, and troubleshoot IP SLA

47.1 Basic IP SLA probes, responders, features, and configurations

IP SLA is a feature that is used to actively probe network targets to gather desired statistics. This commonly includes basic reachability, latency, jitter, and even more advanced application operations like DNS lookups or HTTP GETs. In many deployments, a basic ICMP-echo operation is used to test reachability, and depending on the result, perform some action. This document will detail many of the lesser known/advanced features also. The square brackets will encompass a number in the description of each test to denote the IP SLA identifier, such as [ID 1]. Numbers will not be re-used on different routers for additional exclusivity and clarity. The network was kept simple since there are differences between IOS and XE code trains with respect to support IP SLA probes.



In this test, we have 2 CSR and 2 XRV routers in a square so that we can test all the combinations of IP SLA probers and responders. EIGRP is enabled on all nodes for basic reachability between loopbacks (not shown). A distribute-list is applied everywhere so that only loopbacks are advertised; this is the functional equivalent of OSPF's "prefix-suppression" feature. We will begin with a simple probe on CSR1 to track reachability to CSR2's loopback via the connected interface [ID 1]. We use the "source-interface" to set the source IP of the packet, but this does not force the probe to exit via that interface.

```

! CSR1
ip sla 1
  icmp-echo 153.2.2.2 source-interface GigabitEthernet2.512
  frequency 5
ip sla schedule 1 life forever start-time now

```

The reason this probe actually tracks the proper connected interfaces is because of our EIGRP distribute-list; CSR2 does not have a route back to the CSR1-CSR2 subnet if the link fails or is shutdown, so the probe fails. It isn't a fail-safe way to test a link; normally we would ping the connected interface, but for demonstration purposes, I used this roundabout method. ICMP-echoes are the simplest form of IP SLA and will not be discussed in greater detail.

```

R1#show ip sla statistics 1 | include Number
Number of successes: 3
Number of failures: 0

```

When I break the link between CSR1 and CSR2 (not shown), the probe fails. CSR1 still has reachability to the probe target but CSR2 cannot reply.

```

R1#show ip sla statistics 1 | include Number
Number of successes: 5
Number of failures: 15

```

TCP and UDP based probes are also very useful. ICMP is easy and automatic because routers send echo-replies when an echo is destined to a local interface. With TCP/UDP, more coordination is required, such as listening on specific layer 4 ports. IP SLA has an automatic (and also a manual) mechanism for

responding to TCP/UDP probes; the aptly-named IP SLA “responder” feature. When enabled with no options on CSR2, it will automatically reply to any IP SLA operation targeted to the router upon which the responder was configured.

```
! CSR2
ip sla responder
```

This raises the question of how the router determines the difference between an IP SLA probe and an ordinary data flow. IP SLA uses UDP port 1967 for signaling/control when using TCP/UDP probes. These control packets are sent first to establish communications with the remote end, then the data flow using the specific TCP/UDP ports follow. A “TCP-connect” operation [ID 2] has been configured on CSR1 to test TCP reachability to CSR2’s loopback on port 55555.

```
! CSR1
ip sla 2
  tcp-connect 153.2.2.2 55555 source-ip 153.1.1.1
  timeout 5000
  frequency 5
ip sla schedule 2 life forever start-time now
```

Enabling “debug ip sla trace” on CSR1 reveals the general operation. The control message is sent first to notify the IP SLA responder about the probe details as we configured them. This includes the target IP addresses, layer 4 port, and other information. The IP SLA responder process then expects a TCP packet with appropriate destination IP address and port information as specified in the control message. The TCP connect probe succeeds with a round-trip time (RTT) of 68 ms. Without the responder enabled on CSR2, the router would not have been listening on TCP 55555 and the packet would have been rejected.

```
R1#debug ip sla trace
IP SLAs(2) Scheduler: Starting an operation
IP SLAs(2) CtrlMsg: Sending msg, ver=1, id=15, len=52, cmd=3, ip=153.2.2.2,
port=55555, duration=5000ms
IP SLAs(2) CtrlMsg: Receive status = 0
IP SLAs(2) tcpConnect operation: Sending tcp packet
IP SLAs(2) tcpConnect operation: RTT=68

R1#show ip sla statistics 2 | include RTT|Number
Round Trip Time (RTT) for Index 2
  Latest RTT: 68 milliseconds
Number of successes: 32
Number of failures: 0
```

Because this control signaling is dynamic, some extra security would be useful. We can authenticate the control messages, too. Simply apply a key-chain in global configuration mode to the IP SLA process. In this example, we apply the key-chain only to CSR1, and the operation fails. Notice that the trace debug

also gives us a non-zero receive status, which in many cases indicates an error. Although it doesn't tell us the exact reason, we can use the show command to reveal the fault.

! CSR1

```
key chain KC_IP_SLA
  key 1
    key-string IP_SLA

ip sla key-chain KC_IP_SLA
```

```
R1#debug ip sla trace
IP SLAs(2) CtrlMsg: Receive status = 2
```

```
R1#show ip sla statistics 2 | include return|Number
Latest operation return code: Authentication failure
Number of successes: 52
Number of failures: 7
```

To see the error in real-time via debug, the routers support another debug specific to error messages.

```
R1#debug ip sla error
IPSLA-OPER_ERROR: OPER: 2 Processing video control: status = SLA_AUTH_FAIL
```

On the responder side, the debug reveals an authentication failure as well. The control message contained two commands: one for the probe details and one for authentication. On the XE routers, you must use the digit "0" after the debug command to invoke IP SLA responder debugging (not shown).

```
R2#debug ip sla trace
Ver: 1 ID: 82 Len: 52
cmd: command: RTT_CMD_TCP_CONN_ENABLE, ip: 153.2.2.2, port: 55555, duration:
5000
cmd: command: RTT_CMD_AUTH
IP SLAs responder: authentication failure
```

Once we add the same authentication configuration to CSR2 (not shown), the responder side, everything works as expected. The authentication configuration is identical on the probe sender and responder. The debug on CSR2 shows that the "hash insert" occurred, indicating successful authentication. The hash is removed after the operation completes. Some platforms will not show this debug, though, and "debug ip sla error" can be used instead to troubleshoot failures.

```
R2#debug ip sla trace
Ver: 1 ID: 128 Len: 52
cmd: command: RTT_CMD_TCP_CONN_ENABLE, ip: 153.2.2.2, port: 55555, duration:
5000
cmd: command: RTT_CMD_AUTH
IP SLAs hash insert : 153.2.2.2 55555
```

```
IP SLAs RESP: starting expTimer and hashTimer, duration = 5000
cleaning up port (153.2.2.2,55555)
IP SLAs hash remove: 153.2.2.2 55555
```

Another useful command to view IP SLA responder statistics is shown below. Notice that it shows successful IP SLA sources (probers) as well as invalid ones. It even reveals that the root of the issue was an authentication failure. This is handy for troubleshooting unexpected IP SLA failures.

```
R2#show ip sla responder
IP SLAs Responder is: Enabled
Number of control message received: 106 Number of errors: 50
Recent sources:
  153.1.1.1 [00:42:49.223 UTC Fri Mar 1 2002]
  153.1.1.1 [00:42:44.215 UTC Fri Mar 1 2002]
  153.1.1.1 [00:42:39.227 UTC Fri Mar 1 2002]
  153.1.1.1 [00:42:34.215 UTC Fri Mar 1 2002]
  153.1.1.1 [00:42:29.223 UTC Fri Mar 1 2002]
Recent error sources:
  153.1.1.1 [00:42:29.223 UTC Fri Mar 1 2002]    RTT_AUTH_FAIL
  153.1.1.1 [00:42:24.227 UTC Fri Mar 1 2002]    RTT_AUTH_FAIL
  153.1.1.1 [00:42:19.207 UTC Fri Mar 1 2002]    RTT_AUTH_FAIL
  153.1.1.1 [00:42:14.215 UTC Fri Mar 1 2002]    RTT_AUTH_FAIL
```

Let's test in the UDP-echo probes reverse direction. We know that we need IP SLA responder on CSR1, but we can also use a manual method. The authentication configured above was only for the control messages. We can configure IP SLA probes to suppress this control mechanism and only send traffic on the designated port, which will be 55556 in this example [ID 3]. Without the control, dynamic responses are not possible, and CSR1 must be configured for a specific IP address and layer 4 port pairing to which responses are allowed. **Key point: The IP address and port specified here are the destination address and the destination port of the probe.**

```
! CSR1
ip sla responder udp-echo ipaddress 153.1.1.1 port 55556

! CSR2
ip sla 3
  udp-echo 153.1.1.1 55556 control disable
  frequency 5
ip sla schedule 2 life forever start-time now
```

Notice that CSR2 does not send any control messages, just the UDP packet. This is also valuable for generating synthetic traffic on a specific TCP/UDP port; without control messages, the TCP/UDP packets are sent immediately. When testing QoS or ACLs, for example, generating traffic on key TCP/UDP ports without dealing with the IP SLA control mechanisms is made easy using this method.

```
R2#debug ip sla trace
```

```

IP SLAs(3) Scheduler: Starting an operation
IP SLAs(3) udpEcho Operation: VRF:          lookup table id: 0
IP SLAs hash insert : 153.1.2.2 55682
IP SLAs(3) udpEcho Operation: Sending udp packet
IP SLAs(3) udpEcho Operation: RTT=20
IP SLAs hash remove: 153.1.2.2 55682

```

CSR1 does not display any debug output this time because IP SLA responder (dynamic mode) is disabled. There are no control messages to process or control replies to send. CSR1 only responds to the specific UDP port and IP address pair specified. Instead, we can verify the static responder using the basic show command.

```

R1#show ip sla responder
IP SLAs Responder is: Disabled

```

```

udpEcho Responder:
  IP Address      Port
  153.1.1.1      55556

```

Finally, we can verify the probes are actually successful on CSR2.

```

R2#show ip sla statistics 3 | include Number
Number of successes: 41
Number of failures: 0

```

By setting a memory threshold on IP SLA, you can limit the number of probes that can be created. This is valuable when you are using some dynamic entities that might create many probes at will, such as Performance Routing (PFR) or some kind of SDN/PCE controller. In this test, we will check how many free bytes we have for “processor”, set the IP SLA memory threshold to that value, then try to make a new IP SLA. The router tells us we have reached our maximum limit (it doesn’t tell us exactly why).

```

R1#show memory statistics

```

	Head	Total (b)	Used (b)	Free (b)
Processor	6613F920	154915328	20544804	134370524
I/O	F400000	12582912	4124712	8458200

```

R1(config)#ip sla low-memory 134370524
R1(config)#ip sla 999
%Already at System Maximum Number of Entries of 16

```

If we clear or lower the limit, we can create new IP SLAs. On IOS XE, this limit does not seem to be as strict, as the memory management intelligence on XE is superior to that of IOS. On XE, you can still create many IP SLAs as their memory is shared differently.

```

R1(config)#no ip sla low-memory
R1(config)#ip sla 999

```

```
R1(config-ip-sla)#
```

To see the default low-memory watermark, as well as other general IP SLA information (probe counts supported operations, chronological information, etc) use the following command. The default low watermark is about 20% of the total memory, so IP SLA can continue allocating memory until that watermark is reached. Notice that the difference between the estimated system max (24775) and the number of configurable operations (24759) is equal to the number of entries configured (16). We didn't configure 16 real probes yet; I simply added dummy probes (not shown) to use more memory. Likewise, the sum of the active, pending, and inactive entries will always equal the number of entries configured.

```
R1#show ip sla application
      IP Service Level Agreements
Version: Round Trip Time MIB 2.2.0, Infrastructure Engine-II
Time of last change in whole IP SLAs: 00:59:39.879 UTC Fri Mar 1 2002
Estimated system max number of entries: 24775

Estimated number of configurable operations: 24759
Number of Entries configured      : 16
Number of active Entries          : 0
Number of pending Entries         : 0
Number of inactive Entries        : 16

      Supported Operation Types
Type of Operation to Perform: dhcp
Type of Operation to Perform: dlsw
Type of Operation to Perform: dns
[snip, the list is very long]

IP SLAs low memory water mark: 32954225
```

The output from IOS XE boxes is more concise but displays the same type of information.

```
R1#show ip sla application
      IP Service Level Agreements
Version: Round Trip Time MIB 2.2.0, Infrastructure Engine-III

Supported Operation Types:
icmpEcho, path-echo, path-jitter, udpEcho, tcpConnect, http
dns, udpJitter, dhcp, ftp, lsp Group, lspPing, lspTrace
802.lagEcho VLAN, EVC, Port, 802.lagJitter VLAN, EVC, Port
pseudowirePing, y1731Delay, y1731Loss, udpApp, wspApp, mcast
generic

[snip]
```

IP SLA can also be configured to notify the SNMP-server process via traps for any relevant log messages. The SNMP-server process also needs to be configured to enable "ipsla" traps in order for it to work. This

configuration assumes you already have an SNMP-server somewhere with a trap-host configured; I quickly specify CSR2 as a dummy SNMP server. Using SNMP with IP SLA also requires a reaction-configuration, which tells IP SLA what to do what a certain probe result is seen. In this example, we reuse IP SLA ID 2 (TCP-connect operation) and tell IP SLA to generate a trap when the connection fails, and to do so upon the first failure. The logger works in the opposite direction and will log when the probe starts working again, but it does not log successive failures or successes.

```
! CSR1
ip sla logging traps
snmp-server enable traps ipsla
snmp-server host 153.2.2.2 COMMUNITY

ip sla reaction-configuration 2 react connectionLoss action-type trapOnly
threshold-type immediate
```

Let's start the probe then shut down CSR2's loopback while debugging SNMP packets. Starting the probe and shutting down the loopback are not shown for brevity. The debugs are shown in green with syslog messages in yellow.

```
R1#debug snmp packet
%RTT-4-OPER_CLOSE: condition occurred, entry number = 2
SNMP: Queuing packet to 153.2.2.2
SNMP: V1 Trap, ent rttMonNotificationsPrefix, addr 153.1.1.1, gentrap 6,
spectrap 1
[snip]

%RTT-3-IPSLATHRESHOLD: IP SLAs(2): Threshold Occurred for connectionLoss
```

If we bring CSR2's loopback back up, we should see another reaction event. Specifically, the syslog messages tells us the reaction event was "cleared", whereas the failure of the probe used "occurred". Another trap is sent to notify the SNMP server of the state change.

```
! CSR1
%RTT-4-OPER_CLOSE: condition cleared, entry number = 2
SNMP: Queuing packet to 153.2.2.2
SNMP: V1 Trap, ent rttMonNotificationsPrefix, addr 153.1.2.1, gentrap 6,
spectrap 1
[snip]
%RTT-3-IPSLATHRESHOLD: IP SLAs(2): Threshold Cleared for connectionLoss
```

As a quick discussion, the reaction-configuration has many other configuration parameters. Below are a few configuration snippets of useful reactions but I'm not going to paste outputs from their tests. The first example reacts to RTT when it rises above 5000 ms or falls below 200 ms. It sends an SNMP trap (seen earlier) and also triggers a second event (to be discussed shortly). The thresholds, both rising and falling, are only breached if 2 out of every 3 probes cross the threshold.


```
! Sample
ip sla reaction-configuration 1 react rtt threshold-value 5000 200 action-
type trapAndTrigger threshold-type xOfy 2 3
```

The next example reacts when a probe times out. Notice there is no threshold-value for the timeout variable; “timeout” is a yes/no result, unlike RTT. No SNMP trap is sent, but a follow-on action is triggered. The threshold is only crossed when the average probe result over 5 attempts differs from the status quo. If the probe is up, and there are 3 consecutive failures, then regardless of the previous or next 2 attempts, the reaction occurs due to probe failure.

```
! Sample
ip sla reaction-configuration 3 react timeout action-type triggerOnly
threshold-type average 5
```

Reaction-trigger is a tool to initiate different probes when a threshold is crossed. Below is an example; we re-use IP SLA ID 2 from earlier to test TCP-connection state to CSR2 loopback on port 55555. If this probe fails 2 times in a row, we do not want to issue a trap, but want to start IP SLA ID 1 to just ping CSR2’s loopback for 60 seconds. The logic is that even if we can’t established a TCP-connection, we can at least see if the address is reachable using ping. This implies that IP SLA ID 1’s start-time is “pending”, which means it will not begin until the IP SLA process automatically starts it. The IP SLA start/run times are not discussed in detail because they are self-explanatory, with the exception of the “pending” keyword. The syntax of the “reaction-trigger” command expects the source probe first [ID 2] and the follow-on probe second [ID 1].

```
! CSR1
ip sla reaction-trigger 2 1
ip sla schedule 1 life 60 start-time pending
ip sla reaction-configuration 2 react connectionLoss threshold-type
consecutive 2 action-type triggerOnly
```

IP SLA ID 2 appears to be working correctly so far, which means IP SLA ID 1 isn’t running.

```
R1#show ip sla statistics 2 | include Number
Number of successes: 15
Number of failures: 0
```

Let’s temporarily stop making CSR2 a dynamic IP SLA responder then watch the probe fail while debugging IP SLA trace.

```
! CSR2 (temporary)
no ip sla responder
```

We expect the ICMP-echo probe to work, though. Remember that we must wait for 2 consecutive TCP-connection failures before IP SLA will start the trigger probe.

```
R1#show ip sla statistics 2 | include Number|return_code
Latest operation return code: No connection
Number of successes: 93
Number of failures: 2
```

The trigger mechanism begins and we see a syslog message indicating that we crossed the rising threshold for connection loss. Also notice that the router set the start-time for IP SLA ID 1 to “now” after we configured it as pending initially. Note: the word “pending” may not show in the configuration, depending on the version, but the backup probe must be scheduled as shown earlier. The debug shows IP SLA ID 1 sending an ICMP echo to 132.2.2.2 (CSR2 loopback0) from 153.1.2.1 (CSR1 connected link) as expected.

```
! CSR1
%RTT-3-IPSLATHRESHOLD: IP SLAs(2): Threshold Occurred for connectionLoss
```

```
R1#debug ip sla trace 1
IPSLA-INFRA_TRACE:OPER:1 SLA_SCHED_START_TYPE_NOW
IPSLA-INFRA_TRACE:OPER:1 Starting an operation
IPSLA-OPER_TRACE:OPER:1 Starting icmpecho operation - destAddr=153.2.2.2,
sAddr=153.1.2.1
```

The ICMP-echo probe was successful, and we can see watch the lifetime decrement. After the IP SLA ID 1 probe ends, we can still view the statistics until we restart the probe or reboot the router. We see exactly 12 probes because we set the frequency to 5 seconds and ran the probe for 60 seconds only. The operation time to live is now zero, which means the probe has completed.

```
R1#show ip sla statistics 1 | include Number|^Operation
Number of successes: 2
Number of failures: 0
Operation time to live: 51 sec
```

```
R1#show ip sla statistics 1 | include Number|^Operation
Number of successes: 12
Number of failures: 0
Operation time to live: 0
```

IP SLA operations are not always just pass/fail. There are actually three non-error options that can be returned: OK (success), timeout (fail), and over threshold. The first two are self-explanatory and we have seen them many times already. Because there is a second probe timer value known as “threshold”, we can classify a probe as being somewhere between pass and fail. Threshold must be less or equal to timeout as it represents a more aggressive measurement. As an example, we can probe a link to check its status. If the RTT is greater than 100 ms, IP SLA considers this over threshold, and I might prefer alternative routing paths or take some actions. If the RTT is greater than 200 ms, I may consider the link unusable and take some other action. If the probe returns a value between 100 ms (threshold) and 200 ms (timeout), the probe is considered “over threshold”. The actions taken are a result of tracked objects

discussed in another section, but the timers themselves are functions of IP SLA. A somewhat useless IP SLA is used for demonstration here to avoid adding policers and data-plane traffic conditioning unrelated to this section. I have modified IP SLA ID 1 to have a threshold of 0 ms and a timeout of 5000 ms.

```
! CSR1
ip sla 1
  icmp-echo 153.2.2.2 source-interface GigabitEthernet2.512
  frequency 5
  threshold 0
  timeout 5000
ip sla schedule 1 life forever start-time now
```

We can confirm these changes using the show command below. With the probe running, we can see an RTT of 4 ms, which is over threshold but under timeout. Even with a very fast RTT of 4ms, the IP SLA fails. The “over-threshold” condition is considered a failure to IP SLA, much like the timeout, but IP SLA records this as “over threshold”.

```
R1#show ip sla configuration 1 | include Threshold|timeout
Operation timeout (milliseconds): 5000
Threshold (milliseconds): 0
```

```
R1#show ip sla statistics 1 | include Number|code|RTT
Round Trip Time (RTT) for Index 1
  Latest RTT: 4 milliseconds
Latest operation return code: Over threshold
Number of successes: 0
Number of failures: 5
```

We can demonstrate a forced timeout by reducing the timer to a very low number (0 in this case). Notice how an RTT is not even recorded because after the timeout is exceeded, IP SLA stops keeping track of that probe and considers it totally failed. This could occur when using IP SLA to test HTTP downloads as larger files may complete successfully but fail to do so in a timely manner. Although not terribly granular, this third option (pass, fail, or over threshold) is very useful. In later sections, we will see how object tracking can differentiate between “over threshold” and “timeout” conditions.

```
R1#show ip sla configuration 1 | include Threshold|timeout
Operation timeout (milliseconds): 0
Threshold (milliseconds): 0
```

```
R1#show ip sla statistics 1 | include Number|code|RTT
Round Trip Time (RTT) for Index 1
  Latest RTT: NoConnection/Busy/Timeout
Latest operation return code: Timeout
Number of successes: 0
Number of failures: 3
```

47.2 UDP-jitter and VOIP codec probes

The next probe to examine is the UDP-jitter. This special probe gives us extra information about the probe, such as jitter, delay, MOS, etc. It is still UDP-based and requires a similar responder configuration, but the synthetic traffic is very different from a UDP-echo probe. We configure a new IP SLA on CSR2 [ID 4] and a corresponding static IP SLA responder on CSR1. When configuring the IP SLA initially, we are presented with several numerical options, some of which are new.

```
R2(config-ip-sla)#udp-jitter 153.1.1.1 55557 ?
interval      Inter Packet Interval ! (T, 10 msec)
num-packets   Number of Packets to be transmitted ! (N, 10 pkts)
```

```
R2(config-ip-sla-jitter)#?
IP SLAs jitter Configuration Commands:
frequency          Frequency of an operation ! (F, 60 sec)
request-data-size  Request data size ! (S, 10 bytes)
```

Cisco's documentation summarizes this very concisely: "The UDP jitter operation sends N number of UDP packets, each of size S, T milliseconds apart, from a source device to a destination device, at a given frequency of F". I've added the letters to each variable just for visualization along with the default values; the parser doesn't show it. Keep in mind that "S" is the payload size, so the default packet size is 38 bytes (20 IP + 8 UDP + 10 payload). To make a more complicated example, we will make a probe that sends 200 packets, spaced 25 ms apart, every 10 seconds. Each packet should be exactly 100 bytes, therefore $S = 100 - 28 = 72$. We know that it takes 5 seconds just to send the packets ($200 * 25 \text{ ms} = 5000 \text{ ms}$), so it does not make sense to make F less than 5, although the parser does not seem to complain if you try. A UDP-jitter static responder is configured identically to a UDP-echo one; CSR1 just needs the destination IP address and port number in order to respond. The final configuration is shown below.

```
! CSR1
ip sla responder udp-echo ipaddress 153.1.1.1 port 55557

! CSR2
ip sla 4
  udp-jitter 153.1.1.1 55557 control disable num-packets 200 interval 25
  request-data-size 72
  frequency 10
ip sla schedule 2 life forever start-time now
```

The output from the statistics show command is very long now as the UDP-jitter operation attempts to collect several pieces of information. The full output is below. We can see a valid RTT, which we also achieved with the UDP-echo and ICMP-echo probe. All other fields are zero, which is not valuable. So

far, this probe has not given us any new information. Clearly the probe is capable of providing this information, so our configuration needs to be fixed.

```
R2#show ip sla statistics 4
Round Trip Time (RTT) for Index 4
  Latest RTT: 22 milliseconds
Latest operation start time: *01:21:34.675 UTC Fri Mar 1 2002
Latest operation return code: OK
RTT Values:
  Number Of RTT: 200      RTT Min/Avg/Max: 4/22/52 milliseconds
Latency one-way time:
  Number of Latency one-way Samples: 0
  Source to Destination Latency one way Min/Avg/Max: 0/0/0 milliseconds
  Destination to Source Latency one way Min/Avg/Max: 0/0/0 milliseconds
Jitter Time:
  Number of SD Jitter Samples: 0
  Number of DS Jitter Samples: 0
  Source to Destination Jitter Min/Avg/Max: 0/0/0 milliseconds
  Destination to Source Jitter Min/Avg/Max: 0/0/0 milliseconds
Packet Loss Values:
  Loss Source to Destination: 0      Loss Destination to Source: 0
  Out Of Sequence: 0 Tail Drop: 0
  Packet Late Arrival: 0 Packet Skipped: 0
Voice Score Values:
  Calculated Planning Impairment Factor (ICPIF): 0
  Mean Opinion Score (MOS): 0
Number of successes: 5
Number of failures: 0
Operation time to live: Forever
```

The key is knowing that the two IP SLA endpoints MUST be synchronized via NTP in order to resolve one-way latency. Cisco clearly states this in their documentation but it is commonly forgotten. I will make CSR1 an NTP master and CSR2 will reference time from CSR1. Before continuing, we ensure the routers synchronize time correctly.

```
! CSR1
ntp master
```

```
! CSR2
ntp server 153.1.1.1
```

```
R1#show ntp status | include ^sync
Clock is synchronized, stratum 10, reference is 127.127.7.1
```

```
R2#show ntp status | include ^sync
Clock is synchronized, stratum 11, reference is 153.1.1.1
```

Once NTP is synchronized, we now see populated one-way delay values, but not jitter.

```

R2#show ip sla statistics 4 | section Latency|Jitter
Latency one-way time:
    Number of Latency one-way Samples: 199
    Source to Destination Latency one way Min/Avg/Max: 1/15/42 milliseconds
    Destination to Source Latency one way Min/Avg/Max: 2/9/35 milliseconds
Jitter Time:
    Number of SD Jitter Samples: 0
    Number of DS Jitter Samples: 0
    Source to Destination Jitter Min/Avg/Max: 0/0/0 milliseconds
    Destination to Source Jitter Min/Avg/Max: 0/0/0 milliseconds

```

In order to record jitter values, you MUST use a dynamic IP SLA probe with a dynamic responder on the remote end. This requirement is not as well documented and is even easier to overlook. To prove it, a new IP SLA [ID 5] has been added to CSR1 to probe CSR2's loopback for delay/jitter measurements. It has the same parameters as the previous one but in the opposite direction because CSR2 was previously configured as a dynamic IP SLA responder. CSR2's configuration is shown again for reinforcement.

```

! CSR1
ip sla 5
    udp-jitter 153.2.2.2 55558 num-packets 200 interval 25
    request-data-size 72
    frequency 10
ip sla schedule 5 life forever start-time now

! CSR2
ip sla responder

```

Notice that CSR1 has the jitter information within the trace information for IP SLA ID 5, but CSR2 does not for IP SLA ID 4.

```

R1#debug ip sla trace
IP SLAs(5) jitter operation: V1, verifying sequence numbers
IP SLAs(5) jitter operation: seq=28, jitterOut=-2
IP SLAs(5) jitter operation: seq=28, jitterIn=3
IP SLAs(5) jitter operation: sourceSeq: 30, destSeq: 30
IP SLAs(5) jitter operation: ST: 5448089, RT: 5448087, DT: 0, CT: 5448103
IP SLAs(5) jitter operation: seq=29, rtt=14

R2#debug ip sla trace
IP SLAs(4) jitter operation: sourceSeq: 87, destSeq: 0
IP SLAs(4) jitter operation: ST: 5548401, RT: 5548435, DT: 0, CT: 5548429
IP SLAs(4) jitter operation: seq=86, rtt=28

```

Checking CSR1's statistics for this new probe, we can see that both latency and jitter are being properly resolved.

```

R1#show ip sla statistics 5 | section Jitter|Latency
Latency one-way time:
    Number of Latency one-way Samples: 112
    Source to Destination Latency one way Min/Avg/Max: 2/5/22 milliseconds
    Destination to Source Latency one way Min/Avg/Max: 22/31/54 milliseconds
Jitter Time:
    Number of SD Jitter Samples: 199
    Number of DS Jitter Samples: 199
    Source to Destination Jitter Min/Avg/Max: 0/6/36 milliseconds
    Destination to Source Jitter Min/Avg/Max: 0/4/16 milliseconds

```

One final interesting note is that if we break NTP synchronization, we still get jitter measurements (due to the dynamic responder), but no longer one-way delay measurements.

```

! CSR1
no ntp
ip sla restart 5

```

```

R1#show ip sla statistics 5 | section Jitter|Latency
Latency one-way time:
    Number of Latency one-way Samples: 0
    Source to Destination Latency one way Min/Avg/Max: 0/0/0 milliseconds
    Destination to Source Latency one way Min/Avg/Max: 0/0/0 milliseconds
Jitter Time:
    Number of SD Jitter Samples: 199
    Number of DS Jitter Samples: 199
    Source to Destination Jitter Min/Avg/Max: 0/4/28 milliseconds
    Destination to Source Jitter Min/Avg/Max: 0/4/12 milliseconds

```

In summary, there are 4 possible outcomes with respect to resolving latency and jitter using UDP-jitter IP SLAs. Personally, I would always have time synchronized at a minimum, and using a static/dynamic responder is really a design choice.

1. **Get statistics for both latency and jitter/MOS:** Need NTP sync with dynamic IP SLA responder
2. **Get statistics for jitter/MOS only:** No NTP sync with dynamic IP SLA responder
3. **Get statistics for latency only:** Need NTP sync with static IP SLA responder
4. **Get statistics for neither (RTT only):** No NTP sync with static IP SLA responder

You can also use UDP-jitter probes to test mean opinion score (MOS) for IP telephony purposes. This requires a dynamic IP SLA responder as mentioned above. Built-in codecs include G.729A, G.711alaw, and G.711ulaw. The table below shows the default values for these codecs as implemented by IP SLA, and in many cases, these are appropriate for tests since they represent the codecs accurately. I would recommend reducing F in a test environment, but not less than 20 because it takes 20 seconds for the operation to complete (1000 * 20ms). The relationship between S, T, N, and F is exactly the same as described earlier. You do NOT need to account for the RTP header as the codec defaults include the 12 bytes within the value S. You also do not have to use a well-known RTP port (UDP 16384 - 32767); any

port will work.

Codec	Payload size, bytes (S)	Interval btwn pkts, ms (T)	# pkts (N)	Frequency, sec (F)
G.711ulaw	160 + 12 RTP	20	1000	60
G.711alaw	160 + 12 RTP	20	1000	60
G.729A	20 + 12 RTP	20	1000	60

A new IP SLA has been configured on CSR1 [ID 6] running G.711ulaw to CSR2 (dynamic responder) with NTP synchronized. We should see latency, jitter, and MOS computations from this probe.

```
R1#show ntp status | include ^sync
Clock is synchronized, stratum 8, reference is 127.127.7.1
```

```
R2#show ntp status | include ^sync
Clock is synchronized, stratum 9, reference is 153.1.1.1
```

```
! CSR1
ip sla 6
  udp-jitter 153.2.2.2 55559 codec g711ulaw
  frequency 25
ip sla schedule 6 life forever start-time now
```

A general note about jitter samples is that, in a perfect network, the value should be N-1. The reason it is not exactly N is because the first packet is meaningless to measure jitter; you need a minimum of two because jitter is a delta between latencies. So with a jitter measurement count of 999 while N = 1000, we know our jitter calculation is as accurate as possible. The same rules apply with respect to NTP sync and static/dynamic responders as demonstrated earlier. You can have latency, jitter/MOS, both, or neither depending on the state of the network.

```
R1#show ip sla statistics 6 | section Latency|Jitter|MOS
Latency one-way time:
  Number of Latency one-way Samples: 23
  Source to Destination Latency one way Min/Avg/Max: 4/5/12 milliseconds
  Destination to Source Latency one way Min/Avg/Max: 32/36/43 milliseconds
Jitter Time:
  Number of SD Jitter Samples: 999
  Number of DS Jitter Samples: 999
  Source to Destination Jitter Min/Avg/Max: 0/4/37 milliseconds
  Destination to Source Jitter Min/Avg/Max: 0/4/20 milliseconds
MOS score: 4.34
```

Another interesting parameter in the codec configuration is the “advantage factor” or “A” according to ITU-T Recommendation G.113. This is an artificial handicap that can be applied which positively affects MOS or ICPIF (an impairment measurement). The mathematical details are beyond the scope of this document, but it’s like “expectation management” where a user may acknowledge that being on a cell-phone in a remote location will be lower quality than a local land-line call. Higher advantage factors

imply a more forgiving operator (greater handicap). To test this, we will re-use pieces of IP SLA ID 6 from CSR1 that was an ordinary VOIP probe. The default advantage factor is 0, which means no forgiveness. A new VOIP probe has been added to CSR1 [ID 8] with similar characteristics but an advantage factor of 15. Notice that the probe has not been started yet.

```
! CSR1
ip sla 8
  udp-jitter 153.2.2.2 55560 codec g711alaw advantage-factor 15
  frequency 25
```

I have applied a temporary 64 kbps policer on CSR2 on ingress that will cause about 20% packet loss over that link. This matches UDP ports between 50000 and 60000 which can capture all of the VOIP probes used so far. This is a rudimentary way of creating a “low quality phone call”.

```
! CSR2
ip access-list extended ACL_UDP_50000
  permit udp any any range 50000 60000

class-map match-all CMAP_UDP_JITTER
  match access-group name ACL_UDP_50000

policy-map PMAP_UDP_JITTER
  class CMAP_UDP_JITTER
    police 64000

interface GigabitEthernet2.512
  service-policy input PMAP_UDP_JITTER
```

First, we will run IP SLA ID 6 and see the result on MOS and ICPIF. We can see the policer has a large negative impact and the call is much lower quality that observed earlier. Only 818/1000 packets made it through with a large impairment factor as a result. The MOS has dropped substantially also.

```
! CSR1
ip sla schedule 6 life forever start-time now
```

```
R2#show policy-map interface input
FastEthernet0/1
```

```
Service-policy input: PMAP_UDP_JITTER

Class-map: CMAP_UDP_JITTER (match-all)
  1829 packets, 391406 bytes
  5 minute offered rate 12000 bps, drop rate 4000 bps
Match: access-group name ACL_UDP_50000
police:
  cir 64000 bps, bc 2000 bytes
  conformed 1433 packets, 306662 bytes; actions:
```

```

transmit
exceeded 396 packets, 84744 bytes; actions:
drop
conformed 26000 bps, exceed 7000 bps

R1#show ip sla statistics 6 | include Number_Of_RTT|MOS|ICPIF
Number Of RTT: 818      RTT Min/Avg/Max: 1/27/56 milliseconds
Calculated Planning Impairment Factor (ICPIF): 25
MOS score: 3.50

```

When we re-run the test with a heavily handicapped VOIP probe (A = 15), the results appear more positive despite losing a few more packets (only 773/1000 succeeded). The difference in the impairment factor is exactly 15 (25 – 10), which was our configured value of “A”. MOS improves slightly as well, and since the “O” in MOS stands for “opinion”, the advantage factor adds additional subjectivity to the measurement of voice quality.

```

! CSR1
ip sla schedule 8 life forever start-time now

R1#show ip sla statistics 8 | include Number_Of_RTT|MOS|ICPIF
Number Of RTT: 773      RTT Min/Avg/Max: 4/29/53 milliseconds
Calculated Planning Impairment Factor (ICPIF): 10
MOS score: 4.09

```

47.3 Advanced ICMP probes

This section will discuss the ICMP-jitter, path-jitter, and path-echo probes. These are ICMP-based probes that do not require any kind of IP SLA responder, much like the basic ICMP-echo probe. To be clear, the path-echo probe is essentially a traceroute probe, which uses UDP probes on ports starting at 33434 to determine the routing path. The context-sensitive help considers it an ICMP-based operation because the unreachable that come back are ICMP messages. Likewise, the path-jitter relies on traceroute functionality also. Compared to the UDP-jitter probes, the ICMP-jitter and path-jitter operations cannot resolve MOS or ICPIF.

ICMP-jitter works similarly to UDP-jitter except the default value for the inter-packet interval (T) is 20 ms (like the codec IP SLAs) versus 10 ms for a generic UDP-jitter probe. Our example on CSR1 [ID 9] uses N=25 and T=20. We can control the probe frequency (F) from the IP SLA also, but we cannot control the payload size (S). With our configured parameters for N and T, the probe will require 500 ms to run, so an F value of 2 seconds should be sufficient. Because a responder is never required, jitter measurements are always captured. So long as you don’t need VOIP-type parameter resolution, ICMP-jitter is a good option if the target device is inside of a different administrative domain (perhaps you can’t access it to enable IP SLA responder, or perhaps it isn’t a Cisco router at all). The only special requirement for ICMP-jitter is that ICMP timestamp-requests must transit from prober to target, and timestamp-replies must transit back. Ensure any data-plane filters in the path are configured appropriately. This feature is NOT

supported in XE and this test was conducted using IOS version 12.4T. For XE platforms, use the path-jitter or UDP-jitter options instead. These probes are examined later.

```
! CSR1
ip sla 9
  icmp-jitter 153.2.2.2 num-packets 25
  timeout 2000
  frequency 2
ip sla schedule 9 life forever start-time now

R1#show ip sla statistics 9 | section Latency|Jitter
Type of operation:      icmpJitter
Latency one-way time:
  Number of Latency one-way Samples: 3
  Source to Destination Latency one way Min/Avg/Max: 0/4/10
  Destination to Source Latency one way Min/Avg/Max: 24/27/30
Jitter Time:
  Number of SD Jitter Samples: 24
  Number of DS Jitter Samples: 24
  Source to Destination Jitter Min/Avg/Max: 0/2/8
  Destination to Source Jitter Min/Avg/Max: 0/3/13
```

Just like the UDP-jitter probe, if NTP is not synchronized, ICMP-jitter does not attempt to resolve one-way latency. I temporarily disable NTP on CSR1 (not shown) to demonstrate this.

```
R1#show ntp status
%NTP is not enabled.

R1#show ip sla statistics 9 | section Latency|Jitter
Type of operation:      icmpJitter
Latency one-way time:
  Number of Latency one-way Samples: 0
  Source to Destination Latency one way Min/Avg/Max: 0/0/0
  Destination to Source Latency one way Min/Avg/Max: 0/0/0
Jitter Time:
  Number of SD Jitter Samples: 24
  Number of DS Jitter Samples: 24
  Source to Destination Jitter Min/Avg/Max: 0/6/33
  Destination to Source Jitter Min/Avg/Max: 0/3/21
```

The next probe is the path-echo. This feature introduces some new parameters, including the ability to encode loose-source-routing (LSR) options into the IP header. CSR1 has a path-echo probe [ID 10] targeting CSR2's loopback and the connected interface towards CSR2 has been shutdown. The probe doesn't give us much information (with the ordinary show commands) other than whether the traceroute worked or not. It can be used for determining if traceroute works on a given path, which might be useful to continuously test firewall policies or other data-plane filters. EIGRP routing has been modified so that CSR2 has reachability to the transit link between CSR1 and XRv3 (not shown).

```

! CSR1
ip sla 10
  path-echo 153.2.2.2
  threshold 50
  timeout 100
  frequency 10
ip sla schedule 10 life forever start-time now

```

```

R1#traceroute 153.2.2.2
Type escape sequence to abort.
Tracing the route to 153.2.2.2
VRF info: (vrf in name/id, vrf out name/id)
 1 153.1.13.13 [MPLS: Label 93003 Exp 0] 5 msec 4 msec 3 msec
 2 153.13.14.14 [MPLS: Label 94003 Exp 0] 2 msec 4 msec 3 msec
 3 153.2.14.2 3 msec 3 msec 3 msec

```

```

R1#show ip sla statistics 10 | include Number
Number of successes: 23
Number of failures: 0

```

Next, I disable ICMP unreachable on CSR2 so that it cannot send a port-unreachable back to CSR1. CSR1 can see the intermediate hops (they send time-expired messages) but not the final hop.

```

! CSR2
interface GigabitEthernet2.524
  no ip unreachable

R1#traceroute 153.2.2.2
Type escape sequence to abort.
Tracing the route to 153.2.2.2
VRF info: (vrf in name/id, vrf out name/id)
 1 153.1.13.13 [MPLS: Label 93003 Exp 0] 5 msec 3 msec 5 msec
 2 153.13.14.14 [MPLS: Label 94003 Exp 0] 12 msec 3 msec 3 msec
 3 * * *
[snip]

```

This IP SLA probes takes a long time to actually resolve, just like a default traceroute. It sends 3 probes out a few seconds apart with increasing TTLs until TTL reaches 64. Here are a few debug samples while the probe was running. Personally, I don't consider this useful if you have to watch debugs all day to see the result.

```

R1#debug ip sla trace 10
IPSLA-OPER_TRACE:OPER:10 Return Value of T for target 0.0.0.0
IPSLA-OPER_TRACE:OPER:10 Sending Packet to 153.2.2.2 with ttl = 64
[snip]
IPSLA-INFRA_TRACE:OPER:10 Starting an operation
IPSLA-OPER_TRACE:OPER:10 Sending a pathEcho operation

```

```
IPSLA-OPER_TRACE:OPER:10 hop count = 2
IPSLA-OPER_TRACE:OPER:10 Sending Packet to 153.2.2.2 with ttl = 1
```

```
R1#show ip sla statistics 10 | include Number
Number of successes: 11
Number of failures: 0
```

To be discussed later is the concept of history and aggregated statistics. The path-echo relies entirely upon that to be useful. We will quickly re-test this probe and examine the aggregated statistics. The topology is a square again and CSR1 routes to CSR2 directly (link is brought up). We start the probe and allow it to succeed a few times. We can check its path status using a new command.

```
R1#traceroute 153.2.2.2 source 153.1.1.1
Type escape sequence to abort.
Tracing the route to 153.2.2.2
VRF info: (vrf in name/id, vrf out name/id)
 1 153.1.2.2 3 msec 1 msec 1 msec
```

```
R1#show ip sla statistics aggregated 10
IPSLAs aggregated statistics
```

```
IPSLA operation id: 10
Start Time Index: 01:46:53.375 [snip]
Path Index: 1
Hop in Path Index: 1
Type of operation: path-echo
Number of successes: 2
Number of failures: 0
Target Address 153.1.2.2
```

Let's leave the probe running but shutdown the link to CSR2. Now, we should route via XRv3 and XRv4. Notice that the probe has a new path index number, along with each hop in the path, plus the target address. Because we did not restart or delete the IP SLA completely, the old statistics remain. Each "path-index" represents a unique path in the network. In our case, index 1 was the direct CSR1 > CSR2 path, while index 2 is CSR1 > XRv3 > XRv4 > CSR2. The hop indices represent reach hop within the path-index construct.

```
R1#traceroute 153.2.2.2 source 153.1.1.1
Type escape sequence to abort.
Tracing the route to 153.2.2.2
VRF info: (vrf in name/id, vrf out name/id)
 1 153.1.13.13 [MPLS: Label 93003 Exp 0] 6 msec 3 msec 4 msec
 2 153.13.14.14 [MPLS: Label 94000 Exp 0] 3 msec 3 msec 3 msec
 3 153.2.14.2 4 msec 3 msec 4 msec
```

```
R1#show ip sla statistics aggregated 10
IPSLAs aggregated statistics
```

```
IPSLA operation id: 10
Start Time Index: 01:46:53.375 UTC [snip]
Path Index: 1
Hop in Path Index: 1
Type of operation: path-echo
Number of successes: 5
Number of failures: 0
Target Address 153.1.2.2
```

```
IPSLA operation id: 10
Start Time Index: 01:50:40.089 UTC [snip]
Path Index: 2
Hop in Path Index: 1
Type of operation: path-echo
Number of successes: 11
Number of failures: 0
Target Address 153.1.13.13
```

```
Start Time Index: 01:50:40.089 UTC [snip]
Path Index: 2
Hop in Path Index: 2
Type of operation: path-echo
Number of successes: 11
Number of failures: 0
Target Address 153.13.14.14
```

```
Start Time Index: 01:50:40.089 UTC [snip]
Path Index: 2
Hop in Path Index: 3
Type of operation: path-echo
Number of successes: 11
Number of failures: 0
Target Address 153.2.14.2
```

Similar to the path-echo is the path-jitter. This can also be controlled by the loose-source-routing (LSR) IP option, but will display jitter on a hop-by-hop basis. This can be useful for testing the jitter along each hop in the path cumulatively. This operation is a two-step process. First, the IP SLA performs a traceroute operation to determine the path. The IP addresses returned from traceroute are then used as targets for an ordinary ICMP-echo operation. For this test, I have shut down the link CSR1 and CSR2, forcing traffic to route through XRv3 and XRv4.

```
R1#traceroute 153.2.2.2 source loopback 0
```

```
Type escape sequence to abort.
Tracing the route to 153.2.2.2
```

```
 1 153.1.13.13 [MPLS: Label 93002 Exp 0] 44 msec 72 msec 72 msec
```

```
2 153.13.14.14 [MPLS: Label 94000 Exp 0] 84 msec 60 msec 84 msec
3 153.2.14.2 72 msec 56 msec 84 msec
```

The IP addresses returned from this traceroute are now targets of individual probes that are not explicitly shown. They are revealed with debug commands, but beware that the debugs are quite noisy with this operation. The debug states how many echoes it is sending to a specific hop, and shows every echo and echo-reply. The probe [ID 11] is shown below with no fancy options.

```
! CSR1
ip sla 11
  path-jitter 153.2.2.2 source-ip 153.1.1.1
  frequency 5
ip sla schedule 11 life forever start-time now
```

I've shown the first two echoes in yellow and green as a demonstration. Basically, this is an enhanced traceroute that discovers the path, probes each hop, and reports statistics per hop [ID 11]. The router must have reachability to individual hops. Even though the traceroute works without having reachability to the transit links, the path-jitter will fail to collect statistics for any hop for which reachability is lacking. Thus, the EIGRP distribute-lists remain disabled for this test. The debug output below shows one of the hops in the path and the probes going to it; I did not show all three. Yellow represents one probe while green represents a second.

```
R1#debug ip sla trace 11
  IP SLAs PathJitter(11): Sending 10 echos to 153.2.14.2
  IP SLAs PathJitter(11): send echo id 9511 (seq 0)
  IP SLAs PathJitter(11): send echo id 9512 (seq 1)
  IP SLAs PathJitter(11): recv reply id 9511 (seq 0)
  IP SLAs PathJitter(11): send echo id 9513 (seq 2)
  IP SLAs PathJitter(11): recv reply id 9512 (seq 1)
[snip]
```

After the probes are collected, the debug also reveals statistics about that hop. This is the same information available from the detailed statistics show command. Notice in the shown command below that all hops in the path are displayed as individual records. The output is very long so I omit redundant details to illustrate the point.

```
R1#debug ip sla trace 11
  IP SLAs PathJitter(11): Hop 3 Statistics:
  Source IP Address      : 153.1.1.1
  Hop IP Address         : 153.2.14.2
  Total Receives         : 10
  Discarded Samples      : 0
  Out of Sequence        : 0
  Packet Loss (Timeouts) : 0
  RTT Min                : 56
```

```
RTT Max          : 83
RTT Sum          : 708
[snip]
```

```
R1#show ip sla statistics 11 details
Round Trip Time (RTT) for   Index 11
  Latest RTT: 73 milliseconds
Latest operation start time: 00:19:09.211 UTC Fri Mar 1 2002
Latest operation return code: OK
Over thresholds occurred: FALSE
```

```
---- Path Jitter Statistics ----
```

Hop IP 153.1.13.13:

```
Round Trip Time milliseconds:
  Latest RTT: 20 ms
  Number of RTT: 10
  RTT Min/Avg/Max: 15/20/32 ms
  RTT Sum/Sum2: 206/4470
Jitter time milliseconds:
  Number of jitter: 9
  Jitter Min/Avg/Max: 3/5/9 ms
  Number of positive jitter: 5
  Positive jitter Min/Avg/Max: 3/3/5 ms
  Positive jitter Number/Sum/Sum2: 5/18/68
  Number of negative jitter: 4
  Negative jitter Min/Avg/Max: 7/7/9 ms
  Negative jitter Number/Sum/Sum2: 4/31/243
Packet Values:
  Packet Loss (Timeouts): 0
  Out of Sequence: 0
  Discarded Samples: 0
```

Hop IP 153.13.14.14:

```
Round Trip Time milliseconds:
  Latest RTT: 42 ms
  Number of RTT: 10
  RTT Min/Avg/Max: 35/42/56 ms
  RTT Sum/Sum2: 425/18431
[snip]
```

Hop IP 153.2.14.2:

```
Round Trip Time milliseconds:
  Latest RTT: 73 ms
  Number of RTT: 10
  RTT Min/Avg/Max: 59/73/95 ms
  RTT Sum/Sum2: 736/55168
[snip]
```


We can fine-tune the probe count and inter-packet interval; the defaults are 10 probes spaced 20 ms apart. We can also direct the IP SLA to only collect statistics for the target and not hops along the path. A new IP SLA [ID 12] has been configured on CSR1 to demonstrate these options. With “targetOnly”, the final destination is the only hop, and the router will not probe intermediate hops.

```
! CSR1
ip sla 12
  path-jitter 153.2.2.2 source-ip 153.1.1.1 num-packets 4 interval 25
  targetOnly
  frequency 5
ip sla schedule 2 life forever start-time now
```

The design choice for using this option versus an ordinary ICMP-jitter operation is that this relies on ordinary UDP traceroute probes and subsequent ICMP echoes; ICMP timestamp messages are not required. Data-plane filters may have exceptions for common utilities like ping/traceroute; such exceptions are less common for ICMP timestamp messages.

```
R1#debug ip sla trace
IP SLAs PathJitter(12): Starting a pathJitter operation
IP SLAs PathJitter(12): IP=153.2.2.2, Echos=4, Interval=25, Target
Only=Enabled
[snip]
IP SLAs PathJitter(12): Sending 4 echos to 153.2.2.2
[snip, echos sent and received]
IP SLAs PathJitter(12): Hop 1 Statistics:
      Source IP Address      : 153.1.1.1
      Hop IP Address        : 153.2.2.2
      Total Receives        : 4
[snip]
IP SLAs PathJitter(12): Operation complete

R1#show ip sla statistics 12
Round Trip Time (RTT) for   Index 12
      Latest RTT: 68 milliseconds
Latest operation start time: 00:32:41.067 UTC Fri Mar 1 2002
Latest operation return code: OK

---- Path Jitter Statistics ----

Hop IP 153.2.2.2:
Round Trip Time milliseconds:
      Latest RTT: 68 ms
      Number of RTT: 4
      RTT Min/Avg/Max: 48/68/95 ms
[snip]
```

47.4 MPLS probes

IP SLA is also a key component of MPLS OAM. Reference the “MPLS OAM” section of this document to

familiarize yourself with MPLS ping and traceroute operations before continuing. MPLS ping and traceroute operations are supported per LSP and the IP SLA configuration generally mirrors the normal exec commands for each of these features.

With CSR1 still routing to CSR2 via XRv3 and XRv4, let's quickly do an MPLS traceroute and ensure the LSP is in working order. This proves that the connected link between CSR1 and CSR2 is currently down. The output below shows 'L' for labeled output interface and a final exclamation mark indicating the LSP was constructed successfully. Note: "mpls oam" must be enabled on XE and XR platforms. It's on by default in XE but not XR. MPLS OAM is discussed more in another section, and these tests assume you have a general familiarity with MPLS OAM.

```
R1#traceroute mpls ipv4 153.2.2.2/32 source 153.1.1.1
Tracing MPLS Label Switched Path to 153.2.2.2/32, timeout is 2 seconds

Type escape sequence to abort.
 0 153.1.13.1 MRU 1500 [Labels: 93002 Exp: 0]
 L 1 153.1.13.13 MRU 1500 [Labels: 94000 Exp: 0] 16 ms
 L 2 153.13.14.14 MRU 1504 [Labels: implicit-null Exp: 0] 48 ms
 ! 3 153.2.14.2 68 ms
```

Now we configure an MPLS ping IP SLA operation to continuously test this LSP [ID 13]. Again, many of the extended options are not specific to IP SLA but are specific to MPLS OAM and are discussed later. Notice that the target of the probe is a FEC (IPv4 prefix and mask), not just an IP address.

```
! CSR1
ip sla 13
 mpls lsp ping ipv4 153.2.2.2 255.255.255.255 source-ipaddr 153.1.1.1
 frequency 5
ip sla schedule 13 life forever start-time now

R1#show ip sla statistics 13 | include Number
Number of successes: 3
Number of failures: 0
```

We will break the LSP by temporarily disabling LDP label advertisement on CSR2 completely. At this point, XRv4 has no label for 153.2.2.2/32 and cannot label-switch traffic towards CSR2. The IP SLA reports failures, which effectively alerts the user that the LSP has failed. ICMP-echo would report a false-positive in this case as IP reachability is not degraded.

```
! CSR2
no mpls ldp advertise-labels

R1#show ip sla statistics 13 | include Number
Number of successes: 10
Number of failures: 5
```

A quick MPLS traceroute and MPLS ping clearly reveals the problem; XRv4 is trying to use an unlabeled interface ("B" flag) to reach CSR2 and the IP SLA probe has properly reported it. Before continuing, we enable LDP label advertisement on CSR2 again to repair the MPLS network (not shown).

```
R1#traceroute mpls ipv4 153.2.2.2/32 source 153.1.1.1
Tracing MPLS Label Switched Path to 153.2.2.2/32, timeout is 2 seconds
[snip]
```

```
Type escape sequence to abort.
 0 153.1.13.1 MRU 1500 [Labels: 93002 Exp: 0]
L 1 153.1.13.13 MRU 1500 [Labels: 94000 Exp: 0] 16 ms
B 2 153.13.14.14 MRU 1504 [No Label] 48 ms
[snip]
```

```
R1#ping mpls ipv4 153.2.2.2/32 source 153.1.1.1
Sending 5, 100-byte MPLS Echos to 153.2.2.2/32,
      timeout is 2 seconds, send interval is 0 msec:
[snip]
Type escape sequence to abort.
BBBBB
```

Next we will configure the MPLS traceroute probe on CSR1 to trace the LSP to CSR2's loopback [ID 14]. Note that some of the IP SLA parameters for MPLS traceroute are similar to the path-echo probe examined earlier, such as statistics and history kept.

```
! CSR1
ip sla 14
 mpls lsp trace ipv4 153.2.2.2 255.255.255.255 force-explicit-null source-
ipaddr 153.1.1.1
 frequency 10
ip sla schedule 2 life forever start-time now
```

This probe works the same way except it tests an MPLS path. MPLS ping does not require MPLS OAM to be enabled on transit devices (like XRv3 and XRv4), but MPLS traceroute does. This is because intermediate hops must process the OAM messages and respond, rather than just label-switch them. The probe is currently working.

```
R1#show ip sla statistics 14 | include Number
Number of successes: 2
Number of failures: 0
```

Next, I temporarily disable MPLS OAM on XRv4. We expect the traceroute to fail when TTL=2 since XRv4 is the second hop in the path from CSR1 to CSR2.

```
! XRv4
no mpls oam
```

```
R1#traceroute mpls ipv4 153.2.2.2/32 source 153.1.1.1
[snip]
```

Type escape sequence to abort.

```
0 153.1.13.1 MRU 1500 [Labels: 93003 Exp: 0]
L 1 153.1.13.13 MRU 1500 [Labels: 94003 Exp: 0] 9 ms
. 2 *
! 3 153.2.14.2 5 ms
```

```
R1#debug ip sla trace 14
```

```
IPSLA-OPER_TRACE:OPER:14 Sending Packet to 153.2.2.2 with TTL value of 2
IPSLA-OPER_TRACE:OPER:14 Return Value of '.' for target 153.2.2.2
IPSLA-OPER_TRACE:OPER:14 Timeout Return value '.'
```

Just like the path-echo, the probe seems to never actually fail, and watching the debug can be time-consuming. Remember, the MPLS ping still works even with OAM disabled on the transit routers.

```
R1#ping mpls ipv4 153.2.2.2/32 source 153.1.1.1
[snip]
```

Type escape sequence to abort.

```
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/5/10 ms
Total Time Elapsed 26 ms
```

Using the aggregate statistics show commands, we see the LSP information hop-by-hop. The network has been restored with all links enabled and MPLS OAM active everywhere. The probe begins by following the one-hop LSP directly to CSR2. It's hardly an LSP given the implicit-null label, but it does qualify. Then, after shutting down that CSR1-CSR2 connected link, a second LSP is discovered via XRv3 and XRv4. These outputs are nearly identical to those seen in the path-echo except the type of probe is "lsp trace" versus "path-echo". I've highlighted the path indices and hop indices to illustrate. Notice there were a few failures of the LSP trace during IGP convergence (green), but this isn't significant.

```
R1#show ip sla statistics aggregated 14
IPSLAs aggregated statistics
```

```
IPSLA operation id: 14
Type of operation: lsp trace
Start Time Index: 02:22:22.507 UTC [snip]
Path Index: 1
Hop in Path Index: 1
Type of operation: lsp trace
Number of successes: 2
Number of failures: 2
Target Address 153.1.2.2

Start Time Index: 02:22:22.507 UTC [snip]
Path Index: 2
```

```
Hop in Path Index: 1
Type of operation: lsp trace
Number of successes: 5
Number of failures: 0
Target Address 153.1.13.13

Start Time Index: 02:22:22.507 UTC [snip]
Path Index: 2
Hop in Path Index: 2
Type of operation: lsp trace
Number of successes: 5
Number of failures: 0
Target Address 153.13.14.14

Start Time Index: 02:22:22.507 UTC [snip]
Path Index: 2
Hop in Path Index: 3
Type of operation: lsp trace
Number of successes: 5
Number of failures: 0
Target Address 153.2.14.2
```

47.5 Ethernet probes including ITU-T Y.1731 Basics and Performing Monitoring (PM)

The ITU-T defined Y.1731 for large-scale carrier Ethernet network management and performance monitoring (PM). This is almost identical in operation to IEEE 802.3ag (CFM) described earlier, except it uses different terms to describe certain maintenance components. The real benefit of Y.1731 is the addition of PM, which comes in the form of some additional signaling parameters and IP SLA probes. Although not specific to IP SLA probes, we will briefly discuss the Y.1731 terminology and configure some Y.1731 features before testing probes.

Maintenance Entity Group (MEG): This is equivalent to a CFM maintenance association (MA). A MEG/MA is a set of MEPs that are all part of the same service instance (MEG/MA identifier, or MEGID/MAID) and domain. Each MEP must have a unique MPID within the MEG/MA. The only difference is that a MAID can be up to 43 characters while a MEGID is limited to 13. The CCMs sent by MEPs within a MEG/MA carry the MEGID/MAID as seen in the Ethernet OAM section.

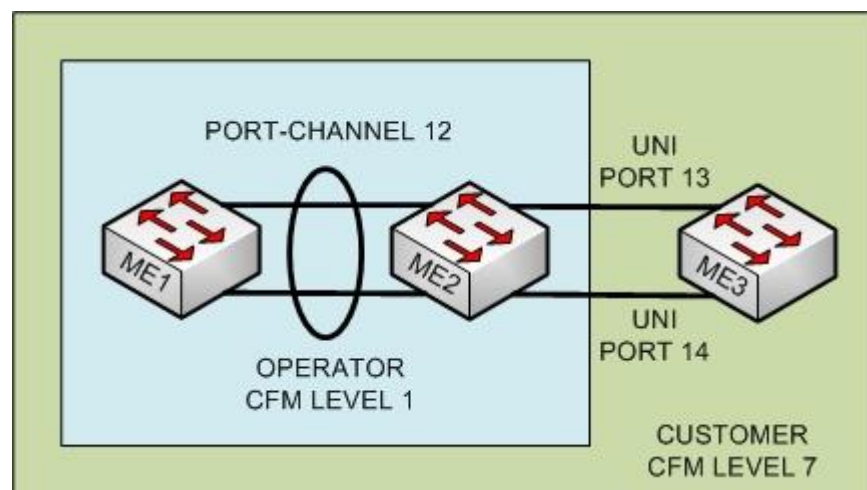
Alarm Indication Signal (AIS): Though discussed in the CFM section casually, this is actually a Y.1731 feature that enhances CFM. It is enabled by default and is used to signal alarms in a carrier Ethernet network. If a link fails, the switch that sees the failure will signal AIS to other switches in the MEG/MA.

Remote Defect Indication (RDI): This was also discussed in the CFM section and is a complementary function of AIS. The remote switches with MEPs in a MEG/MA will set the RDI bit in their CCMs when sending traffic back towards the failure. This assists with fault isolation and notifies other MEPs that a failure has occurred somewhere in the domain. The absence of RDI CCMs indicates no failure, and since RDI is carried in CCMs, CCMs must be enabled for RDI to work.

Lock status (LCK): This was not discussed in the CFM section and is another type of signaling using in Y.1731. Unlike AIS, LCK does not signify an error condition but is treated similarly as AIS upon receipt at a MEP. Data is interrupted and the Ethernet service breaks, but this allows the MEP to differentiate between a real failure (AIS) and an administrative decision (LCK). This is useful for troubleshooting or simply “shutting down” a MEP.

Service/server MEP (SMEP): Explained briefly, this is an interface which reports fault conditions whether it is actually a MEP or not. This is mostly relevant for verification and show commands.

We will use the CFM/E-LMI lab from earlier as it contains an existing CFM, OAM, and E-LMI architecture. Y.1731 can work alongside any of these protocols and no combination of them is mutually exclusive; Y.1731 and its PM feature set actually require CFM since the two are virtually the same.



Briefing reviewing the switched network, ME1 and ME2 are switches that are owned by an operator. Given limited hardware quantity, this is the “provider” network as well. ME2 is the UPE towards ME3, which is the CE, and provides Ethernet services on UNI ports 13 and 14. These service VLANs 13 and 14, respectively, which have CFM MEPs on ME1 (simulated remote CE) and ME3. ME1 would service the “other end” of these VLAN 13 and VLAN 14 connections if I had more hardware. Between ME2 and ME3, Ethernet OAM is running on port 13 only while E-LMI is running on both ports 13 and 14.

On ME2, we will adjust some basic timers on dummy ports. These settings can also be applied globally, and we can completely turn off AIS and LCK signaling (we won’t since the network is so small). On port 4, we totally disable AIS/LCK signaling, and on port 5, we adjust the timers. There are two options for timers, 60 seconds and 1 second. Cisco recommends 60 seconds when there are many VLANs in use, and 1 second when they are few VLANs in use; the default is 60 for both AIS and LCK. When an event occurs, AIS is signaled immediately, but follow on messages are sent at the specified interval. The default level at which AIS/LCK signaling is transmitted is the level of the MIP on that interface. Since these dummy links are “operator” links, I add them as level 1 MIPs. Port 5 adjusts the CFM level for AIS/LCK transmission as

well, and since MIPs will forward higher-level information, this allows for broader dissemination if desired.

```
! ME2
interface FastEthernet0/4
  ethernet cfm mip level 1 vlan 1-4094
  no ethernet cfm lck link-status
  no ethernet cfm ais link-status

interface FastEthernet0/5
  ethernet cfm mip level 1 vlan 1-4094
  ethernet cfm lck link-status period 1
  ethernet cfm lck link-status level 3
  ethernet cfm ais link-status period 1
  ethernet cfm ais link-status level 3
```

We can verify this by checking the service MEP (SMEP) show command. The confusing thing about Y.1731 configuration is that it is mostly rolled into the “CFM” parser construct. This streamlines configuration but makes it more challenging to differentiate between 802.1ag and Y.1731 feature sets.

```
ME2#show ethernet cfm smep interface f0/4
SMEP Settings:
-----
Interface: FastEthernet0/4
LCK-Status: Disabled
LCK Period: 60000 (ms)
Level to transmit LCK: Default
AIS-Status: Disabled
AIS Period: 60000 (ms)
Level to transmit AIS: Default
Defect Condition: No Defect
```

```
ME2#show ethernet cfm smep interface f0/5
SMEP Settings:
-----
Interface: FastEthernet0/5
LCK-Status: Enabled
LCK Period: 1000 (ms)
Level to transmit LCK: 3
AIS-Status: Enabled
AIS Period: 1000 (ms)
Level to transmit AIS: 3
Defect Condition: No Defect
```

On ME2, we will check the default parameters on the NNI LACP bond to ME1 and a UNI port to ME3 for completeness. We can see that the port-type has no affect on the default Y.1731 configurations as the AIS/LCK periods are still 60 seconds with the CFM level equal to the MIP level per interface.

```
ME2#show ethernet cfm smep interface port-channel 12
```

```
SMEP Settings:
```

```
-----
```

```
Interface: Port-channel12
```

```
LCK-Status: Enabled
```

```
LCK Period: 60000 (ms)
```

```
Level to transmit LCK: Default
```

```
AIS-Status: Enabled
```

```
AIS Period: 60000 (ms)
```

```
Level to transmit AIS: Default
```

```
Defect Condition: No Defect
```

```
ME2#show ethernet cfm smep interface fast0/13
```

```
SMEP Settings:
```

```
-----
```

```
Interface: FastEthernet0/13
```

```
LCK-Status: Enabled
```

```
LCK Period: 60000 (ms)
```

```
Level to transmit LCK: Default
```

```
AIS-Status: Enabled
```

```
AIS Period: 60000 (ms)
```

```
Level to transmit AIS: Default
```

```
Defect Condition: No Defect
```

We can test the Y.1731 LCK feature as well. Let's assume that ME3, as the customer, wants to administratively "shut down" service for VLAN 13 only. Rather than shut down interfaces, adjust trunk allowed lists, or unplug cables, ME3 can raise the LCK condition. Unlike the ping or cross-check features, this command is applied against a locally-configured MEP. Below, ME3 tries to shut down the remote MEP on ME1 within a given service instance. The parser notifies us that MPID 113 is not found, which suggests the feature is local-only.

```
ME3#ethernet cfm lck start mpid 113 domain CUSTOMER service number 13
```

```
No mep with mpid 113 is configured for Domain CUSTOMER vlan 13. Command  
aborted.
```

We correct the command to reference the local MPID of 313. The associated MEP has been errdisabled on a per-VLAN basis as a result of entering LCK condition.

```
ME3#ethernet cfm lck start mpid 313 domain CUSTOMER service number 13
```

```
01:33:07: %PM-4-ERR_DISABLE_VP: vlan oos error detected on Fa0/9, vlan 13.  
Putting in err-disable state.
```

```
01:33:07: %E_CFM-6-ENTER_LCK: local mep with mpid 313 level 7 VLAN 13 dir U  
Interface Fa0/9 enters LCK defect condition
```

We can verify the errdisable state and LCK condition using some show commands. Finding the LCK condition identified as a "fault" is challenging because it isn't technically an error, so CFM doesn't report

it as such. Instead, we can look at the details for the local MEP within the particular domain/service. This shows the “Defect condition” as LCK. The reason this doesn’t register as an “error” is because this isn’t an AIS condition, and no AIS/RDI is signaled. The interface clearly shows no RDI signal.

```
ME3#show interfaces fast0/9 status
Port      Name                Status      Vlan      Duplex  Speed  Type
Fa0/9     VLAN-BASED MEP      connected   vl-err-dis a-full  a-100  10/100BaseTX
```

```
ME3#show ethernet cfm errors
[no output]
```

```
ME3#show ethernet cfm maintenance-points local detail mep domain CUSTOMER
service number 13
Local MEPs:
-----
[snip]
LCK-Status: Enabled
LCK Period: 60000(ms)
LCK Expiry Threshold: 3.5
Level to transmit LCK: Default
Defect Condition: LCK
presentRDI: FALSE
[snip]
```

Unlike an AIS condition, there is no indication of a failure on ME1 at all. The remote MEP appears fully operational, no RDI is seen in the CCMs, and loopback messages (pings) can be exchanged. From the perspective of all remote MEPs, the LCK condition is local-only, which allows an administrator to break services without causing fault propagation through the MEG/MA.

```
ME1#show ethernet cfm errors
[no output]
```

```
ME1#show ethernet cfm maintenance-points remote detail mpid 313
Version: IEEE-CFM
MAC Address: 001f.9d0b.1680
Domain Name: CUSTOMER
MA Name: number 13
Level: 7
VLAN: 13
MPID: 313
Incoming Port(s): Port-channel12
CC Lifetime(sec): 35
Age of Last CC Message(sec): 2
CC Packet Statistics: 575/0 (Received/Error)
MEP interface status: Up
MEP port status: Up
Receive RDI: FALSE
```

```
ME1#ping ethernet mpid 313 domain CUSTOMER service number 13
Type escape sequence to abort.
Sending 5 Ethernet CFM loopback messages to 001f.9d0b.1680, timeout is 5
seconds:!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/8 ms
```

We can exit the LCK condition by using the exec command to “stop” locking the service. The interface leaves the err-disable state and the LCK condition is cleared from the local MEP details.

```
ME3#ethernet cfm lck stop mpid 313 domain CUSTOMER service number 13
01:51:12: %PM-4-ERR_RECOVER_VP: Attempting to recover from vlan oos err-
disable state on Fa0/9, vlan 13.
01:51:12: %E_CFM-6-EXIT_LCK: local mep with mpid 313 level 7 VLAN 13 dir U
Interface Fa0/9 exited LCK defect condition
```

```
ME3#show interfaces fast0/9 status
Port      Name              Status      Vlan    Duplex  Speed Type
Fa0/9     VLAN-BASED MEP    connected   trunk   a-full  a-100 10/100BaseTX
```

```
ME3#show ethernet cfm maintenance-points local detail mep domain CUSTOMER
service number 13 | include Defect|LCK
LCK-Status: Enabled
LCK Period: 60000(ms)
LCK Expiry Threshold: 3.5
Level to transmit LCK: Default
Defect Condition: No Defect
```

Now, we will configure basic Ethernet probes. These are less complex than Y.1731 PM probes but are a good starting point. As the customer, ME3 wants to ensure service reachability for VLAN 13 across the carrier network. We use a basic loopback message (ping) probe to test reachability to remote MEPs on ME1 [ID 1]. The look-and-feel of this probe is very similar to an ICMP-echo examined earlier. The target is a remote MPID with the specified domain using a specific VLAN. Inside the probe details, we can adjust minor parameters shown below, such as size and CoS. We can verify success/failure using the same show command seen earlier, which makes IP SLA very powerful for a wide range of service probes.

```
! ME3
ip sla 1
  ethernet echo mpid 113 domain CUSTOMER vlan 13
  cos 2
  request-data-size 1000
  threshold 5
  timeout 50
  frequency 5
ip sla schedule 1 life forever start-time now
```

```
ME3#show ip sla statistics 1
IPSLAs Latest Operation Statistics
```

```

IPSLA operation id: 1
Type of operation: ethernet-echo
    Latest RTT: 4 milliseconds
Latest operation start time: 02:01:20 UTC Mon Mar 1 1993
Latest operation return code: OK
RTT Values:
    Number Of RTT: 1                      RTT Min/Avg/Max: 4/4/4 milliseconds
Number of successes: 16
Number of failures: 0
Operation time to live: Forever

```

Additionally, ME3 can monitor jitter across the Ethernet network as well. This is comparable to the UDP-jitter probe since it returns one-way latency, jitter, and RTT. There is no concept of MOS or impairment factor with this probe, so those are not measured. The probe sends 40 frames 25 ms apart, which takes 1000 ms (1 second) to run. Notice that the statistics, just like UDP-jitter, can resolve jitter and RTT without NTP synchronization. One-way delay requires time synchronization; we have not configured NTP yet, so the probe below cannot capture latency.

```

! ME3
ip sla 2
    ethernet jitter mpid 114 domain CUSTOMER vlan 14 num-frames 40 interval 25
    frequency 10
ip sla schedule 2 life forever start-time now

ME3#show ip sla statistics 2
IPSLAs Latest Operation Statistics

IPSLA operation id: 2
Type of operation: ethernet-jitter
    Latest RTT: 3 milliseconds
Latest operation start time: 02:05:56 UTC Mon Mar 1 1993
Latest operation return code: OK
RTT Values:
    Number Of RTT: 40                      RTT Min/Avg/Max: 2/3/6 milliseconds
Latency one-way time:
    Number of Latency one-way Samples: 0
    Source to Destination Latency one way Min/Avg/Max: 0/0/0 milliseconds
    Destination to Source Latency one way Min/Avg/Max: 0/0/0 milliseconds
Jitter Time:
    Number of SD Jitter Samples: 39
    Number of DS Jitter Samples: 39
    Source to Destination Jitter Min/Avg/Max: 0/1/3 milliseconds
    Destination to Source Jitter Min/Avg/Max: 0/1/2 milliseconds
Packet Unprocessed: 0
Packet Loss Values:
    Loss Source to Destination: 0
    Loss Destination to Source: 0
    Out Of Sequence: 0          Tail Drop: 0

```

```
Packet Late Arrival: 0 Packet Skipped: 0
Number of successes: 3
Number of failures: 0
Operation time to live: Forever
```

We quickly configure NTP on all switches. Since we didn't have (or need) IP connectivity before, I also add SVIs for VLAN 14 on ME1 and ME3 to support this. ME1 and ME2 have reachability over VLAN 1 which I use for management. This will allow NTP to converge so that all switches are synchronized to ME1, and I use the fastest possible polling timers (2^4, or 16 seconds) to achieve this.

```
! ME1
interface Vlan14
 ip address 10.14.14.1 255.255.255.0

ntp master

! ME2
ntp server 10.0.10.91 min-poll 4 max-poll 4

! ME3
interface Vlan14
 ip address 10.14.14.3 255.255.255.0

ntp server 10.14.14.1 min-poll 4 max-poll 4
```

We confirm that NTP has synchronized all clocks, then check the IP SLA statistics on ME3 to ensure one-way delay can be measured. Since 40 packets are sent, all 40 can be used for latency measurements, and with jitter, exactly N-1 packets (39 in this case) can be used for jitter calculations as seen earlier.

```
ME1#show ntp status | include sync
Clock is synchronized, stratum 8, reference is 127.127.1.1
```

```
ME2#show ntp status | include sync
Clock is synchronized, stratum 9, reference is 10.0.10.91
```

```
ME3#show ntp status | include sync
Clock is synchronized, stratum 9, reference is 10.14.14.1
```

```
ME3#show ip sla statistics 2 | section Latency
Latency one-way time:
  Number of Latency one-way Samples: 40
  Source to Destination Latency one way Min/Avg/Max: 0/1/6 milliseconds
  Destination to Source Latency one way Min/Avg/Max: 1/1/2 milliseconds
```

With the basic probes operational, we will configure Y.1731 PM probes. These are very specific probes defined in Y.1731 which are unlike the generic probes we just configured. The generic probes would be best used in environments where not all devices were Y.1731-capable or where only basic reachability is

required. Y.1731 provides much more detailed information as part of the larger PM picture. There are a few different types of probes/tests worth discussing first. The detailed information returned by these probes is available for viewing within the CLI parser, as well as many external management tools.

Delay Measurement (ETH-DM): This measures frame delay and frame delay variation (latency and jitter, essentially). This is accomplished by sending ETH-DM frames from source to destination MEP, then receiving ETH-DM frames in reply, which is similar to the jitter probe we saw earlier. Specific DM measurement (DMM) and DM reply (DMR) messages are used for this exchange. Somewhat similar to statistical aggregation (another IP SLA feature), DM probes will tally delay measurements into bins for statistical analysis. While the ME3400 has the syntax options for one-way delay as well, only two-way delay is supported. One-way delay uses a special message (1DM) and is a simpler process.

Frame Loss Measurement (ETH-LM): This operation is an out-of-band signaling mechanism between two MEPs to keep track of transmitted/received frames. It relies on data traffic traversing the network, and the statistical information about those frames is exchanged between MEPs. Some number of frames are transmitted and received, and the difference between the two numbers is the frame loss. Frame loss measurement and reply (FLM/FLR) messages are used to signal between the controller MEP (prober) and responder MEP (target). The only traffic generated by this probe is meant to carry signaling information between MEPs about actual data traffic statistics. As such, this feature is only valid for measuring E-LINE (P2P) services. Though the syntax exists in the ME3400 parser for this feature, it isn't supported on the platform. There are better alternatives available within the Y.1731 specification.

Synthetic Loss Measurement (ETH-SLM): This is similar to ETH-LM in objective but not in operation. This measures frame loss by generating frames (like classic IP SLA probes) rather than monitoring data traffic (like Netflow or ETH-DM). SLM measurement (SLM) and SLM reply (SLR) frames are used to carry this information. SLM improves upon ETH-LM because it works in multipoint service deployments (E-LAN and E-TREE) and serves a dual purpose; the SLM/SLR messages are used as the synthetic traffic and also carry the performance information. While the ME3400 has the syntax options for ETH-LM as well (LMM), only ETH-SLM is supported. This is a single-ended operation that is managed by the controller MEP, and since the probes also carry the loss information, it supports multipoint Ethernet services.

On ME3, we will begin with a two-way delay measurement using Y.1731 DMM/DMR frames. Unlike most IP SLAs, there are many mandatory fields within the probe definition, including CoS and source MEP/MAC address. I use MPIDs rather than MAC addresses in these examples so that the configurations will work on other physical switches. We will measure two-way delay within VLAN 13 using CoS 5, presumably to troubleshoot a VOIP issue. We can adjust the frame size in increments of 4 starting at 64 (and going up to 384) and determine the inter-frame interval, which is valuable for certain applications. The DMM measurement cannot use a frame interval less than 1000 ms, but I configure that explicitly despite it being the default. The max-delay is effectively a timeout measured in ms. The aggregation interval determines how often to aggregate results, which will make more sense when we view the output. The Y.1731 distribution bucket configuration is very flexible as well. We can custom-define the number of buckets and their upper-bounds in microseconds. In this case, I create 5 buckets for both

delay (latency) and delay-variation (jitter): 0 – 999 us, 1000 – 2999 us, 3000 – 4999 us, 5000 – 6999 us, 7000+ us. We can use the integer “-1” to specify an infinite value, which makes sense for the final bucket’s upper-bound. This information is displayed nicely in the detailed show command seen later.

```
! ME3
ip sla 10
  ethernet y1731 delay DMM domain CUSTOMER vlan 13 mpid 113 cos 5 source mpid
  313
  max-delay 10
  frame size 128
  frame interval 1000
  aggregate interval 120
  distribution delay two-way 5 1000,3000,5000,7000,-1
  distribution delay-variation two-way 5 1000,3000,5000,7000,-1
ip sla schedule 10 life forever start-time now
```

First, we will verify the configuration, to include defaults, using the basic IP SLA show command. Notice that the probe thinks the clocks are not synchronized, which means one-way delay (latency) measurements are impossible with this current probe. We will revisit this later. Also notice the frequency is 120 seconds, which is a reflection of the aggregation interval.

```
ME3#show ip sla configuration 10
IP SLAs Infrastructure Engine-III
Entry number: 10
Owner:
Tag:
Operation timeout (milliseconds): 10
Ethernet Y1731 Delay Operation
Frame Type: DMM
Domain: CUSTOMER
Vlan: 13
Target Mpid: 113
Source Mpid: 313
CoS: 5
  Max Delay: 10
  Request size (Padding portion): 128
  Frame Interval: 1000
  Clock: Not In Sync
Threshold (milliseconds): 0
Schedule:
  Operation frequency (seconds): 120 (not considered if randomly scheduled)
  Next Scheduled Start Time: Start Time already passed
  Group Scheduled : FALSE
  Randomly Scheduled : FALSE
  Life (seconds): Forever
  Entry Ageout (seconds): never
  Recurring (Starting Everyday): FALSE
  Status of entry (SNMP RowStatus): Active
```

Statistics Parameters

```
Aggregation Period: 120
Frame offset: 1
Distribution Delay Two-Way:
  Number of Bins 5
  Bin Boundaries: 1000,3000,5000,7000,-1
Distribution Delay-Variation Two-Way:
  Number of Bins 5
  Bin Boundaries: 1000,3000,5000,7000,-1
```

History

```
Number of intervals: 2
```

The basic statistics for this probe are not very interesting. We can see the probe has been running for 35 seconds within this 120-second aggregation interval and the last returned code indicates success.

```
ME3#show ip sla statistics 10
IPSLAs Latest Operation Statistics

IPSLA operation id: 10
Delay Statistics for Y1731 Operation 10
Type of operation: Y1731 Delay Measurement
Latest operation start time: 03:53:16.659 UTC Mon Mar 1 1993
Latest operation return code: OK
Distribution Statistics:
Interval
  Start time: 03:53:16.659 UTC Mon Mar 1 1993
  Elapsed time: 35 seconds
  Number of measurements initiated: 26
  Number of measurements completed: 26
  Flag: OK
```

If you happen to use this command right after the 120 seconds expire, but before the next interval begins recording results, you will get invalid output. The date will read January 1, 1900 and all values will show 0 as the next probe iteration has not collected anything.

```
ME3#show ip sla statistics 10
IPSLAs Latest Operation Statistics

IPSLA operation id: 10
Delay Statistics for Y1731 Operation 10
Type of operation: Y1731 Delay Measurement
Latest operation start time: 00:00:00.000 UTC Mon Jan 1 1900
Latest operation return code: OK
Distribution Statistics:
Interval
  Start time: 00:00:00.000 UTC Mon Jan 1 1900
  Number of measurements initiated: 0
  Number of measurements completed: 0
```

Flag: OK

Of greater interest is the detailed output. This is where we can see our custom distribution buckets and their hit-counts, among other things. The first few stanzas of the output are identical to what we have already seen, so we won't analyze that again (output is omitted). We now get timestamps for the best and worst delay times for historical reference. I specifically customized the bins based on the latencies I knew I would see in the network so we could get better statistical distribution, rather than having everything land in one bucket. We can see that most of the time, the two way delay is between 1 ms and 3 ms, but so far, never greater than 5 ms. The delay-variation measurements include both positive and negative measurements (jitter can be positive or negative since consecutive packets are compared to one another). We can see that both positive and negative jitter measurements are almost always less than 3 ms.

```
ME3#show ip sla statistics 10 details
IPSLAs Latest Operation Statistics
IPSLA operation id: 10
[snip, same as summary output above]
```

Delay:

```
Number of TwoWay observations: 100
Min/Avg/Max TwoWay: 309/2330/3935 (microsec)
Time of occurrence TwoWay:
Min - 03:56:53.533 UTC Mon Mar 1 1993
Max - 03:55:38.413 UTC Mon Mar 1 1993
```

Bin TwoWay:

Bin Range (microsec)	Total observations
0 - < 1000	9
1000 - < 3000	65
3000 - < 5000	26
5000 - < 7000	0
7000 - < 4294967295	0

Delay Variance:

```
Number of TwoWay positive observations: 44
Min/Avg/Max TwoWay positive: 2/1207/3149 (microsec)
Time of occurrence TwoWay positive:
Min - 03:56:08.461 UTC Mon Mar 1 1993
Max - 03:55:38.413 UTC Mon Mar 1 1993
Number of TwoWay negative observations: 54
Min/Avg/Max TwoWay negative: 42/1009/2835 (microsec)
Time of occurrence TwoWay negative:
Min - 03:56:58.541 UTC Mon Mar 1 1993
Max - 03:56:03.453 UTC Mon Mar 1 1993
```

Bin TwoWay positive:

Bin Range (microsec)	Total observations

0 - < 1000	18
1000 - < 3000	25
3000 - < 5000	1
5000 - < 7000	0
7000 - < 4294967295	0

Bin TwoWay negative:

Bin Range (microsec)	Total observations
0 - < 1000	31
1000 - < 3000	23
3000 - < 5000	0
5000 - < 7000	0
7000 - < 4294967295	0

Most of the debugs of these features are hidden within the CFM command set, since Y.1731 PM has been rolled into CFM (generally speaking) within the parser. I show a few debug commands for completeness, but there isn't much value in them. Most of it is very low-level information not relevant for troubleshooting. I only use these for ensuring the probe is actually working (sending frames) without trying to decode the data. The packet debug clearly shows frames being transmitted.

```
ME3#debug ethernet cfm pm monitor
Ethernet PM Monitor debugging is on for all session
```

```
! ME3
04:08:59: EPM-MONITOR(session ID: 0) T[c] = 17 T[p] = 16 aggregated for start
data 17 max_data 40 wrap 0 set zero 0 730958933:479768178 730958933:480880203
730958933:480409279 730958933:481965923 730958934:486425548
730958934:487973025 730958934:487521795 730958934:490288286
```

```
ME3#debug ethernet cfm pm packets
Ethernet PM Packets debugging is on for all session
```

```
! ME3
04:06:36: EPM-PKT(session ID: 0) sec 2B918BCC nsec 226F7190
04:06:36: 00 1D 46 92 F7 00 00 1F 9D 0B 16 80 81 00
04:06:36: A0 0D 89 02 E0 2F 00 20 2B 91 8B CC 22 6F
04:06:36: 71 90 00 00 00 00 00 00 00 00 00 00 00 00
04:06:36: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
04:06:36: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
04:06:36: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
04:06:36: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
04:06:36: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
04:06:36: 00 00
04:06:36: EPM-PKT(session ID: 0) Number of frames xmit 80
```

For comparative assessment, we will create a similar probe except this time, we will notify Y.1731 that the clocks are synchronized [ID 11]. We will apply this to VLAN 14 and adjust some other minor options

for variety. Because the clocks are “in sync”, one-way delay buckets are now available for configuration, so I add them as well. Under the statistics parameters for the previous probe, we only saw buckets for two-way delay measurements; now we see delay and delay-variation measurements for one-way delay.

```
! ME3
ip sla 11
  ethernet y1731 delay DMM domain CUSTOMER vlan 14 mpid 114 cos 5 source mpid
314
  max-delay 15
  clock sync
  aggregate interval 90
  distribution delay one-way 3 1200,2400,-1
  distribution delay two-way 5 1000,3000,5000,7000,-1
  distribution delay-variation one-way 3 1200,2400,-1
  distribution delay-variation two-way 5 1000,3000,5000,7000,-1
ip sla
schedule 11 life forever start-time now
```

```
ME3#show ip sla configuration 11 | begin Statistics
```

Statistics Parameters

Aggregation Period: 90

Frame offset: 1

Distribution Delay One-Way:

Number of Bins 3

Bin Boundaries: 1200,2400,-1

Distribution Delay Two-Way:

Number of Bins 5

Bin Boundaries: 1000,3000,5000,7000,-1

Distribution Delay-Variation One-Way:

Number of Bins 3

Bin Boundaries: 1200,2400,-1

Distribution Delay-Variation Two-Way:

Number of Bins 5

Bin Boundaries: 1000,3000,5000,7000,-1

History

Number of intervals: 2

We quickly check to ensure the probe thinks the clock is synchronized and that the frequency is a reflection of the 90 second aggregation interval.

```
ME3#show ip sla configuration 11 | include Clock|freq
```

Clock: In Sync

Operation frequency (seconds): 90 (not considered if randomly scheduled)

Before looking at the output, there is a command we can use to verify the configured probes within the context of CFM. On the probing device, we can see the CFM level, VLAN ID, and MAC addresses used within a probe after the MPIDs are translated into brain MACs. This is useful to see which probes are in use; note that the “basic” probes are not shown here as this only displays Y.1731 PM probes.

```
ME3#show ethernet cfm pm
```

EPM-ID	SLA-ID	Lvl/Type/ID/Cos/Dir	Src-Mac-address	Dst-Mac-address
0	10	7/Vlan/13/5/Up	001f.9d0b.1680	001d.4692.f700
1	11	7/Vlan/14/5/Up	001f.9d0b.1680	001d.4692.f700

We can see details about any session by referencing the Ethernet PM ID (EPM-ID) which is allocated by the system. This lists all of the probe configuration parameters seen in the IP SLA verification, as well as the Y.1731-specific messages used. Since this is a delay measurement probe, DMM/DMR messages are used as discussed above, while the other messages are not. Recall that 1DM messages are not supported on this platform.

```
ME3#show ethernet cfm pm session detail 0
```

```
Session ID: 0
```

```
Sla Session ID: 10
```

```
Level: 7
```

```
Service Type: Vlan
```

```
Service Id: 13
```

```
Direction: Up
```

```
Source Mac: 001f.9d0b.1680
```

```
Destination Mac: 001d.4692.f700
```

```
Session Status: Active
```

```
MPID: 313
```

```
Tx active: yes
```

```
Rx active: yes
```

```
RP monitor Tx active: yes
```

```
RP monitor Rx active: yes
```

```
Timeout timer: stopped
```

```
Last clearing of counters: 00:00:00.000 UTC Mon Jan 1 1900
```

```
DMMs:
```

```
    Transmitted: 2770
```

```
DMRs:
```

```
    Rcvd: 2770
```

```
1DMs:
```

```
    Transmitted: 0
```

```
    Rcvd: 0
```

```
LMMs:
```

```
    Transmitted: 0
```

```
LMRs:
```

```
    Rcvd: 0
```

```
VSMs:
```

```
    Transmitted: 0
```

```
VSRs:
```

```
    Rcvd: 0
```

```
SLMs:
```

```
    Transmitted: 0
```

SLRs:

Rcvd: 0

The detailed output for this new probe, as expected, provides tons of useful information. With one-way delay being measured also, there is more detail available to the administrator. Technically, you could configure “clock sync” even if NTP was not configured, but this would lead to incorrect information on the switches, ultimately defeating the purpose of Y.1731 PM. This output spans several pages so comments are inserted throughout. Looking at the delay stanza, we now see 2 additional sub-sections for forward and backward one-way delay measurements. Earlier, we only had the two-way section, which didn’t require clock sync and was essentially an RTT measurement. I would have never guessed that my one-way latencies were so much different in this small lab network, but Y.1731 proves me wrong. If we sum ~800 us and ~1600 us timers, we get something close to ~2300 us which is the two-way delay average (values highlighted), so the math actually makes sense.

```
ME3#show ip sla statistics 11 details | begin ^Delay:
```

Delay:

Number of forward observations: 34

Min/Avg/Max forward: 28/859/1673 (microsec)

Time of occurrence forward:

Min - 04:18:25.606 UTC Mon Mar 1 1993

Max - 04:18:40.625 UTC Mon Mar 1 1993

Number of backward observations: 40

Min/Avg/Max backward: 550/1570/2528 (microsec)

Time of occurrence backward:

Min - 04:18:15.585 UTC Mon Mar 1 1993

Max - 04:18:20.594 UTC Mon Mar 1 1993

Number of TwoWay observations: 40

Min/Avg/Max TwoWay: 495/2281/3850 (microsec)

Time of occurrence TwoWay:

Min - 04:18:15.586 UTC Mon Mar 1 1993

Max - 04:18:05.570 UTC Mon Mar 1 1993

Next, we have three sets of delay bins per the probe configuration. We can further prove that the forward delay is much better than the backward delay. This might be cause for concern in a real network as it may indicate certain congestion at transmitting/receiving interfaces along the EVC. Two-way delay is about the same as it was for the previous probe in VLAN 13, as expected.

! ME3 continued

Bin forward:

Bin Range (microsec)	Total observations
0 - < 1200	23
1200 - < 2400	11
2400 - < 4294967295	0

Bin backward:

Bin Range (microsec)	Total observations

0 - < 1200	12
1200 - < 2400	25
2400 - < 4294967295	3

Bin TwoWay:

Bin Range (microsec)	Total observations
0 - < 1000	2
1000 - < 3000	27
3000 - < 5000	11
5000 - < 7000	0
7000 - < 4294967295	0

Next, we have the delay variance stanzas which measure jitter. This is even more verbose than the delay measurements since in addition to the direction of forward vs. backward, it introduces positive vs. negative. All 4 combinations are valid measurements which Y.1731 measures and records within the aggregation interval. Instead of 3 stanzas, there are now 6, each clearly marked. Unlike latency, the jitter appears to be more stable in both directions, generally hovering around 700-800 us.

! ME3 continued

Delay Variance:

Number of forward positive observations: 12

Min/Avg/Max forward positive: 65/711/1363 (microsec)

Time of occurrence forward positive:

Min - 04:18:40.625 UTC Mon Mar 1 1993

Max - 04:18:10.578 UTC Mon Mar 1 1993

Number of forward negative observations: 18

Min/Avg/Max forward negative: 51/603/1407 (microsec)

Time of occurrence forward negative:

Min - 04:18:20.593 UTC Mon Mar 1 1993

Max - 04:18:05.570 UTC Mon Mar 1 1993

Number of backward positive observations: 19

Min/Avg/Max backward positive: 172/834/1976 (microsec)

Time of occurrence backward positive:

Min - 04:18:20.594 UTC Mon Mar 1 1993

Max - 04:18:15.586 UTC Mon Mar 1 1993

Number of backward negative observations: 20

Min/Avg/Max backward negative: 158/786/1743 (microsec)

Time of occurrence backward negative:

Min - 04:18:20.594 UTC Mon Mar 1 1993

Max - 04:18:05.570 UTC Mon Mar 1 1993

Number of TwoWay positive observations: 21

Min/Avg/Max TwoWay positive: 36/806/2490 (microsec)

Time of occurrence TwoWay positive:

Min - 04:18:15.586 UTC Mon Mar 1 1993

Max - 04:18:05.570 UTC Mon Mar 1 1993

Number of TwoWay negative observations: 18

Min/Avg/Max TwoWay negative: 2/937/1982 (microsec)

Time of occurrence TwoWay negative:

Min - 04:18:20.594 UTC Mon Mar 1 1993
Max - 04:18:05.570 UTC Mon Mar 1 1993

Next, we see the bins for the delay variation measurements. Since delay variation is very low in this network, most of the hits are in the “best” bucket which makes characterization a little more difficult. A better configuration would be to create smaller buckets, perhaps of size 500 us, to achieve finer granularity. To correspond with the 6 summary stanzas above, there are 6 different buckets. 4 of them represent one-way delay measurements for all combinations of positive/negative and forward/backward. The remaining 2 are positive/negative two-way delay variation measurements.

! ME3 continued

Bin forward positive:

Bin Range (microsec)	Total observations
0 - < 1200	10
1200 - < 2400	2
2400 - < 4294967295	0

Bin forward negative:

Bin Range (microsec)	Total observations
0 - < 1200	16
1200 - < 2400	2
2400 - < 4294967295	0

Bin backward positive:

Bin Range (microsec)	Total observations
0 - < 1200	14
1200 - < 2400	5
2400 - < 4294967295	0

Bin backward negative:

Bin Range (microsec)	Total observations
0 - < 1200	17
1200 - < 2400	3
2400 - < 4294967295	0

Bin TwoWay positive:

Bin Range (microsec)	Total observations
0 - < 1000	15
1000 - < 3000	6
3000 - < 5000	0
5000 - < 7000	0
7000 - < 4294967295	0

Bin TwoWay negative:

Bin Range (microsec)	Total observations
0 - < 1000	9
1000 - < 3000	9
3000 - < 5000	0

```
5000 - < 7000 0
7000 - < 4294967295 0
```

Next, I configure a Y.1731 probe to attempt one-way delay measurement using 1DM messages [ID 12]. For simplicity, I don't adjust any other parameters since Cisco clearly states this is not supported on the ME3400. There is no need to specify clock synchronization since 1DM probes must assume this to be true, since their only purpose is one-way delay. Debugging IP SLA errors for this probe, we can see that the session never gets created. It does not register as a PM within CFM either; nothing happens.

```
! ME3
ip sla 12
  ethernet y1731 delay 1DM domain CUSTOMER vlan 13 mpid 113 cos 3 source mpid
  313
ip sla schedule 12 life forever start-time now
```

```
ME3#debug ip sla error 12
ME3(config)#ip sla schedule 12 life forever start-time now
04:49:48: IPSLA-OPER_ERROR: OPER: 12 Failed to Create Delay Session
```

```
ME3#show ethernet cfm pm
```

EPM-ID	SLA-ID	Lvl/Type/ID/Cos/Dir	Src-Mac-address	Dst-Mac-address
0	10	7/Vlan/13/5/Up	001f.9d0b.1680	001d.4692.f700
1	11	7/Vlan/14/5/Up	001f.9d0b.1680	001d.4692.f700

There is nothing in the configuration that shows an error, other than the probe simply not working. The statistics show commands work but produce no output. I assume this exists in the parser because other hardware platforms/line-cards may support it using a similar code base.

```
ME3#show ip sla configuration 12
IP SLAs Infrastructure Engine-III
Entry number: 12
Owner:
Tag:
Operation timeout (milliseconds): 5000
Ethernet Y1731 Delay Operation
Frame Type: 1DM
[snip]
```

```
ME3#show ip sla statistics 12
IPSLAs Latest Operation Statistics
```

```
IPSLA operation id: 12
[no output]
```

Now that we have tested frame delay and delay variation extensively, we will test frame loss. We will use the synthetic loss measurement (ETH-SLM) process for this, which introduces some new options. First, we can configure faster inter-frame intervals with SLM, so I set this to 100 ms versus 1 second used for DMM tested above. 100 ms is the fastest interval that SLM can use. I reduce the aggregate interval to 3 and configure the probe to store 3 intervals worth of traffic. Every 8 consecutive frames will be checked to measure loss, and I set the frame size (payload) to the maximum value. I use CoS 1 for the probe because this example can be used to measure the loss of scavenger traffic in the network. We quickly confirm the details of the probe by checking the configuration.

```
! ME3
ip sla 20
  ethernet y1731 loss SLM domain CUSTOMER vlan 14 mpid 114 cos 1 source mpid
314
  frame size 384
  frame interval 100
  history interval 3
  aggregate interval 30
  frame consecutive 8
ip sla schedule 20 life forever start-time now

ME3#show ip sla configuration 20 | begin ^Statistic
Statistics Parameters
  Aggregation Period: 30
  Frame consecutive: 8
  Frame consecutive loss-ratio: 10
  Availability algorithm: sliding-window
History
  Number of intervals: 3
```

Checking CFM, we can see this new entry in the table, which means it was successfully registered as a valid Y.1731 PM probe. The details indicate that SLM/SLR messages are being used. All other message counter fields are 0 and I omit them for brevity.

```
ME3#show ethernet cfm pm
```

EPM-ID	SLA-ID	Lvl/Type/ID/Cos/Dir	Src-Mac-address	Dst-Mac-address
0	10	7/Vlan/13/5/Up	001f.9d0b.1680	001d.4692.f700
1	11	7/Vlan/14/5/Up	001f.9d0b.1680	001d.4692.f700
2	20	7/Vlan/14/1/Up	001f.9d0b.1680	001d.4692.f700

```
ME3#show ethernet cfm pm session detail 2
Session ID: 2
Sla Session ID: 20
Level: 7
Service Type: Vlan
Service Id: 14
```



```
Direction: Up
Source Mac: 001f.9d0b.1680
Destination Mac: 001d.4692.f700
Session Status: Active
MPID: 314
[snip]
SLMs:
    Transmitted: 5477
SLRs:
    Rcvd: 5477
```

The detailed statistics are much less involved than delay and delay variation seen earlier. The basic statistics are omitted and I begin the output where the detailed information begins. The frame counts clearly show no loss in the network. The average frame loss ratio (FLR) is 0% as is the cumulative FLR. Minimums and maximums cannot be computed in a network with no loss. These statistics are identical for forward and backward measurements.

```
ME3#show ip sla statistics 20 details | begin Forward
```

Forward

```
Number of Observations 23
Available indicators: 23
Unavailable indicators: 0
Tx frame count: 230
Rx frame count: 230
Min/Avg/Max - (FLR % ): 0:9/000.00%/0:9
Cumulative - (FLR % ): 000.00%
Timestamps forward:
Min - 05:12:45.573 UTC Mon Mar 1 1993
Max - 05:12:45.573 UTC Mon Mar 1 1993
```

Backward

```
Number of Observations 23
Available indicators: 23
Unavailable indicators: 0
Tx frame count: 230
Rx frame count: 230
Min/Avg/Max - (FLR % ): 0:9/000.00%/0:9
Cumulative - (FLR % ): 000.00%
Timestamps backward:
Min - 05:12:45.573 UTC Mon Mar 1 1993
Max - 05:12:45.573 UTC Mon Mar 1 1993
```

To briefly introduce some frame loss, I shut down the LACP bond on ME1. This breaks much more than just Y.1731, but I use it as a quick way to break reachability to the remote MEP from ME3's perspective. Some frames should be lost and the Y.1731 statistics show the details. We can see the cumulative FLR increasing to a very high value, which indicates almost all traffic, is being lost. However, in the reverse direction we see no fault indication. Since the link is totally broken, the remote end is not seeing any traffic and cannot reply. In the forward direction, ME3 is sending many packets but getting no replies

after shutting down the LACP bond on ME1. With numbers like this, we can infer that the network is totally broken with no connectivity between MEs.

```
ME3#show ip sla statistics 20 details | begin Forward
```

```
Forward
```

```
Number of Observations 1
Available indicators: 1
Unavailable indicators: 0
Tx frame count: 120
Rx frame count: 4
Min/Avg/Max - (FLR % ): 0:3/000.00%/0:3
Cumulative - (FLR % ): 096.66%
Timestamps forward:
Min - 05:22:54.402 UTC Mon Mar 1 1993
Max - 05:22:54.402 UTC Mon Mar 1 1993
```

```
Backward
```

```
Number of Observations 1
Available indicators: 1
Unavailable indicators: 0
Tx frame count: 4
Rx frame count: 4
Min/Avg/Max - (FLR % ): 0:3/000.00%/0:3
Cumulative - (FLR % ): 000.00%
Timestamps backward:
Min - 05:22:54.402 UTC Mon Mar 1 1993
Max - 05:22:54.402 UTC Mon Mar 1 1993
```

Last, I attempt to configure ETH-LM which is the older mechanism of measuring frame loss using a new probe [ID 13]. The probe has no fancy options since the documentation states it isn't supported, so we don't expect it to work.

```
! ME3
ip sla 21
  ethernet y1731 loss LMM domain CUSTOMER vlan 14 mpid 114 cos 1 source mpid
  314
ip sla schedule 21 start-time now life forever
```

Just like with the 1DM probe, we will enable IP SLA debugging to confirm the feature is unsupported. IP SLA reports that it cannot create the loss session, and the Y.1731 PM session table confirms it.

```
ME3#debug ip sla error 21
ME3(config-sla-y1731-loss)#ip sla schedule 21 start-time now life forever
05:30:05: IPSLA-OPER_ERROR: OPER: 21 Failed to Create Loss Session
```

```
ME3#show ethernet cfm pm
```

```
-----
EPM-ID    SLA-ID    Lvl/Type/ID/Cos/Dir Src-Mac-address Dst-Mac-address
```

0	10	7/Vlan/13/5/Up	001f.9d0b.1680	001d.4692.f700
1	11	7/Vlan/14/5/Up	001f.9d0b.1680	001d.4692.f700
2	20	7/Vlan/14/1/Up	001f.9d0b.1680	001d.4692.f700

Configuration-wise, we see nothing wrong. The frames for this operation would be LMM/LMR as discussed earlier, assuming the feature was supported on the ME3400.

```
ME3#show ip sla configuration 21
IP SLAs Infrastructure Engine-III
Entry number: 21
Owner:
Tag:
Operation timeout (milliseconds): 5000
Ethernet Y1731 Loss Operation
Frame Type: LMM
[snip]
```

The last feature to examine is the IP SLA Ethernet-monitor. This feature automatically spawns probes for each of the MEPs within a MEG/MA. We can optionally exclude certain MEPs from this auto-discovery process or specify a subset of VLANs as well, but these are optional modifications. We specify a CFM domain and VLAN 13 for this example, plus optional threshold and CoS settings. The SLA is scheduled somewhat similar to IP SLA group scheduling (discussed in a dedicated section), since many sub-probes will be created.

```
! ME3
ip sla ethernet-monitor 30
  type echo domain CUSTOMER vlan 13
  cos 4
  threshold 500
ip sla ethernet-monitor schedule 30 schedule-period 30 start-time now
```

While configuring the probe, we can enable debugs to see what IP SLA is doing behind the scenes. After the probe is started, IP SLA considers it registered. Shortly thereafter, once it runs for the first time, it discovers a single RMEP inside VLAN 13 with MPID 113.

```
ME3#debug ip sla trace ethernet-monitor 30
IP SLAs Auto Ethernet debugging for entry 30 is on
```

```
! ME3
IPSLA-AUTOETH_TRACE:OPER:30 registered domain = CUSTOMER, vlan =
13,auto_probe = 30
IPSLA-AUTOETH_TRACE:OPER:30 Number of MPIDs discovered = 1
IPSLA-AUTOETH_TRACE:OPER:30 creating probe for domain = CUSTOMER, vlan = 13,
MPID = 113
```

Looking at the IP SLA summary, we see the original 2 probes (this does not show Y.1731 probes) with IDs 1 and 2. We then see a dynamic probe with ID 100001 which was spawned by the Ethernet monitor. This runs once every 30 seconds. This is a basic loopback message (LBM) probe, which is identical to an Ethernet-echo probe we evaluated first in this section. These spawned probes are not Y.1731-based.

```
ME3#show ip sla summary
```

```
IPSLAs Latest Operation Summary
```

```
Codes: * active, ^ inactive, ~ pending
```

ID	Type	Destination	Stats (ms)	Return Code	Last Run
*1	ethernet-ec ho	Domain:CUSTOMER pid:113 Vlan:13	M RTT=5	OK	1 second ago
*2	ethernet-ji tter	Domain:CUSTOMER pid:114 Vlan:14	M RTT=2	OK	1 second ago
*100001	ethernet-ec ho	Domain:CUSTOMER pid:113 Vlan:13	M RTT=3	OK	23 seconds ago

The statistics for the probe show that it is reachable. At a glance, this feature does not appear immediately useful, since the MEPs were already discovered by CFM. Rather than being used for discovery, it will save the administrator from having to manually configure probes. Furthermore, if MPIDs change in the network, the administrator won't have to adjust IP SLA echo probes if Ethernet-monitor is configured within that MEG/MA. The RTT time is probably the most valuable part of this output since the up/down status is CFM-managed.

```
ME3#show ip sla statistics 100001
```

```
IPSLAs Latest Operation Statistics
```

```
IPSLA operation id: 100001
```

```
Type of operation: ethernet-echo
```

```
Latest RTT: 3 milliseconds
```

```
Latest operation start time: 00:51:39 UTC Mon Mar 1 1993
```

```
Latest operation return code: OK
```

```
RTT Values:
```

```
Number Of RTT: 1
```

```
RTT Min/Avg/Max: 3/3/3 milliseconds
```

```
Number of successes: 13
```

```
Number of failures: 0
```

```
Operation time to live: Forever
```

The true value of this feature comes with the reaction-configuration. The details of this IP SLA feature as discussed in another section, but in short, this allows an administration to generate alarms when certain events occur. Ethernet-monitor reaction events only support SNMP traps and not trigger events, so we quickly enable SNMP traps for IP SLA along with IP SLA logging. I configure ME1 as a dummy SNMP

server to receive these traps so that we can see them originated by ME3 via SNMP packet debugs. I configure a basic reaction to timeout which reacts immediately and generates an SNMP trap.

```
! ME3
ip sla logging traps
snmp-server enable traps ipsla
snmp-server host 10.14.14.1 ETHMON ipsla

ip sla ethernet-monitor reaction-configuration 30 react timeout threshold-
type immediate action-type trapOnly
```

We can verify the configuration above by checking the SNMP host list (trap/inform servers) and verifying the reaction configuration was applied to the dynamic probe.

```
ME3#show snmp host
Notification host: 10.14.14.1    udp-port: 162    type: trap
user: ETHMON    security model: v1

ME3#show ip sla reaction-configuration 100001
Entry number: 100001
Index: 1
Reaction: timeout
Threshold Type: Immediate
Action Type: Trap only
```

To quickly test this, I remove VLAN 13 from the trunk allowed list on ME1 port 9. This generates an AIS alarm in CFM (unrelated to IP SLA), but more significantly it will break reachability to the RMEP. I enable SNMP packet debugging so we can confirm that an SNMP trap is sent when the IP SLA probe fails.

```
ME3#debug snmp packets
SNMP packet debugging is on

! ME1
interface FastEthernet0/9
 switchport trunk allowed vlan remove 13
```

When the timeout is first noticed, IP SLA is the first process to generate syslog messages to indicate the timeout condition for probe 100001. This triggers an SNMP trap which we can see below; this carries information about the alarm. Multiple traps might be sent based on the alarm, but I show one for brevity. We can clearly see the configuration is correct or else no traps would have appeared.

```
! ME3
01:04:33: %RTT-4-OPER_TIMEOUT: condition occurred, entry number = 100001
01:04:33: %RTT-3-IPSLATHRESHOLD: IP SLAs(100001): Threshold Occurred for
timeout
```

```

01:04:33: SNMP: Queuing packet to 10.14.14.1
01:04:33: SNMP: V1 Trap, ent rttMonNotificationsPrefix, addr 10.14.14.3,
gentrap 6, spectrap 5
rttMonCtrlAdminTag.100001 =
rttMonHistoryCollectionAddress.100001 =
00 00 00 00 03 08 73 2C 00 19 08 78 00 50 08 4C
04 19 88 B4 00 00 00 14 03 F4 BA 2C
rttMonReactVar.100001 = 7
rttMonReactOccurred.100001 = 1
rttMonReactValue.100001 = 1
rttMonReactThresholdRising.100001 = 0
rttMonReactThresholdFalling.100001 = 0
rttMonEchoAdminLSPSelector.100001 =
00 00 00 00 05 4D F9 00 00 00 00 01 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00

```

A few seconds later, the CFM errors are shown. This indicates that the remote MEP is no longer accessible which is signaled via DefRemoteCCM.

```

! ME3
01:04:36: %E_CFM-3-REMOTE_MEP_DOWN: Remote MEP mpid 113 vlan 13 MA name
number 13 in domain CUSTOMER changed state to down with event code TimeOut.
01:04:36: %E_CFM-6-ENTER_AIS: local mep with mpid 313 level 7 VLAN 13 dir U
Interface Fa0/9 enters AIS defect condition
01:04:38: %E_CFM-3-FAULT_ALARM: A fault has occurred in the network for the
local MEP having mpid 313 vlan 13 for service MA name number 13 with the
event code DefRemoteCCM.

```

Checking the IP SLA summary, the dynamic probe is gone since it is deleted when the remote MEP became inaccessible. Probe 1 was manually configured and was pinging the same remote MEP, and now it too reports a failure. The pings aren't even sent since the remote MEP doesn't exist within the specified CFM domain (that is, it is absent from the CFM database).

```

ME3#show ip sla summary
IPSLAs Latest Operation Summary
Codes: * active, ^ inactive, ~ pending

```

ID	Type	Destination	Stats (ms)	Return Code	Last Run
*1	ethernet-ec ho	Domain:CUSTOMER pid:113 Vlan:13	M RTT=0	Invalid Remote MPID	4 seconds ago
*2	ethernet-ji tter	Domain:CUSTOMER pid:114 Vlan:14	M RTT=3	OK	4 seconds ago

We will quickly repair the network by adding VLAN 13 back to ME1 port 9's trunk allowed list. A new dynamic probe is automatically created to begin monitoring the remote MEP once again. Since the reaction configuration was only tracking timeouts, IP SLA logs and SNMP traps are not generated for the restoration of service.

```
! ME1
interface FastEthernet0/9
 switchport trunk allowed vlan add 13
```

First, CFM clears all alarms since valid CCMs were received from this remote MEP.

```
! ME3
01:15:07: %E_CFM-6-EXIT_AIS: local mep with mpid 313 level 7 VLAN 13 dir U
Interface Fa0/9 exited AIS defect condition
01:15:07: %E_CFM-6-REMOTE_MEP_UP: Continuity Check message is received from a
remote MEP with mpid 113 vlan 13 MA name number 13 domain CUSTOMER interface
status Up event code Returning.
```

Next, we check the Ethernet-monitor to ensure a new probe was generated. The probe ID has been incremented, which is expected behavior when probes are added and removed. Probe ID 1 is also reporting a functional remote MEP again, which helps prove that the network has stabilized.

```
ME3#show ip sla summary
IPSLAs Latest Operation Summary
Codes: * active, ^ inactive, ~ pending
```

ID	Type	Destination	Stats (ms)	Return Code	Last Run
*1	ethernet-ec ho	Domain:CUSTOMER pid:113 Vlan:13	M RTT=3	OK	4 seconds ago
*2	ethernet-ji tter	Domain:CUSTOMER pid:114 Vlan:14	M RTT=3	OK	9 seconds ago
*100002	ethernet-ec ho	Domain:CUSTOMER pid:113 Vlan:13	M RTT=3	OK	13 seconds ago

Additional Reading – Reference configurations "eth-probes"

47.6 Miscellaneous probes

This section discusses less common probes, such as DHCP, HTTP, DNS, and FTP. The DHCP probe has very few options and is meant to test a DHCP server's (probe destination) ability to allocate IP addresses. In this example, we will assume there are some hosts on the LAN segment between CSR1 and XRv3. CSR1 wants to test to see if CSR2 has a matching pool for its connected interface on that LAN [ID

101]. While this is not a 100% fool-proof way of testing a DHCP server, in most cases it is fair to assume that if CSR2 has a pool that covers 153.1.13.1, it has a pool that covers the entire LAN subnet. The minimum configurable frequency value for DHCP operations is 10 seconds.

```
! CSR1
ip sla 101
  dhcp 153.2.2.2 source-ip 153.1.13.1
  frequency 10
ip sla schedule 101 life forever start-time now

! CSR2
ip dhcp pool LAN_1_13
  network 153.1.13.0 255.255.255.0
```

Once the probe is started, DHCP server debugs on CSR2 can reveal some packet details. Because the probe target may not be on the connected subnet, the DHCP prober promotes itself to a DHCP relay so that it can proxy the internally-generated DHCPDISCOVER messages to the DHCP server.

```
R2#debug ip dhcp server packet
R2#debug ip dhcp server event
DHCPD: DHCPDISCOVER received from client 0073.6161.2d31.3031.2c36 through
relay 153.1.13.1.
%DHCPD-4-PING_CONFLICT: DHCP address conflict: server pinged 153.1.13.1.
DHCPD: Adding binding to radix tree (153.1.13.2)
DHCPD: Adding binding to hash tree
DHCPD: assigned IP address 153.1.13.2 to client 0073.6161.2d31.3031.2c36.
DHCPD: Sending DHCP OFFER to client 0073.6161.2d31.3031.2c36 (153.1.13.2).
DHCPD: unicasting BOOTREPLY for client c201.0201.0000 to relay 153.1.13.1.
[snip, DHCP process continues]
```

Because IOS-based DHCP servers will attempt to ping an address before issuing it, the DHCP server allocates the next available address. Each time the probe runs, DHCP re-issues a new address, effectively iterating over the entire range of available addresses. After several iterations, we can look at the DHCP bindings on CSR2 and see the “probe’s” current address. We can see the number of DHCP successes is one less than the address assigned; this is due to the conflict since 153.1.13.1 could not be assigned. CSR1 doesn’t actually store the addresses anywhere, but CSR2 is allocating them correctly, which means the probe is successful.

```
R2#show ip dhcp binding | include 153\.
153.1.13.22 0073.6161.2d31.3031. Mar 01 2002 01:34 AM Automatic

R1#show ip sla statistics 101 | include Number
Number of successes: 21
Number of failures: 0
```

If we temporarily delete the DHCP pool for that LAN, the probe will fail. The source address specified in

the probe is the relay address and is also the input for the DHCP process. As a side note, DHCP option 82 (the information option) can also be inserted into the probe to carry the circuit-id, remote-id, or subnet-mask options. IOS DHCP server debugs and binding databases don't display this information, so a test is unwarranted. A real DHCP server would use this information as specified.

```
! CSR2
no ip dhcp pool LAN_1_13

R2#debug ip dhcp server packet
R2#debug ip dhcp server event
DHCPD: Sending notification of DISCOVER:
  DHCPD: htype 1 chaddr c201.0201.0000
  DHCPD: remote id 020a000099020e0200000000
  DHCPD: circuit id 00000000
DHCPD: DHCPDISCOVER received from client 0073.6161.2d31.3031.2c36 through
relay 153.1.13.1.
DHCPD: there is no address pool for 153.1.13.1.

R1#show ip sla statistics 101 | include Number
Number of successes: 0
Number of failures: 18
```

Next, we will examine the HTTP probe. This probe is generally used for HTTP GET requests but is very flexible in that it can handle raw requests also. HTTP syntax is beyond the scope of this test so the raw requests are not discussed in detail. CSR2 has been configured as an HTTP server with a few files on flash, route.txt and tech.txt. Let's quickly ensure that R1 can download these files.

```
R2#show ip route | redirect flash:route.txt

R2#show tech-support | redirect flash:tech.txt

R1#copy http://153.2.2.2/route.txt null:
Loading http://153.2.2.2/route.txt !
1118 bytes copied in 0.236 secs (4737 bytes/sec)

R1#copy http://153.2.2.2/tech.txt null:
Loading http://153.2.2.2/tech.txt !
236407 bytes copied in 10.764 secs (21963 bytes/sec)
```

We can clearly see the larger file took longer to download. Let's configure a pair of HTTP probes that downloads these files [ID 105, 106] using a threshold of 2 seconds with a timeout of 30 seconds

```
! CSR1
ip sla 105
  http get http://153.2.2.2/route.txt
```

```

threshold 2000
timeout 30000
ip sla schedule 105 life forever start-time now

ip sla 106
http get http://153.2.2.2/tech.txt
threshold 2000
timeout 30000
ip sla schedule 106 life forever start-time now

```

Given our previous discussion about threshold and timeout, we expect the HTTP probe downloading the route.txt file to be successful, while the probe downloading the tech.txt file should be over threshold (which counts as a failure).

```

R1#show ip sla statistics 105 | include Number|HTTP
Latest HTTP Transaction RTT: 208 ms
Number of successes: 3
Number of failures: 0

```

```

R1#show ip sla statistics 106 | include Number|HTTP
Latest HTTP Transaction RTT: 5645 ms
Number of successes: 0
Number of failures: 2

```

IP SLA HTTP operations are not smart enough to use configured HTTP usernames or passwords for authentication, unfortunately. If we enable local authentication on CSR2 with username/password http/http, we can see CSR1 can easily download a file given those credentials.

```

! CSR1
ip http client username http
ip http client password 0 http

! CSR2
ip http authentication local
username http privilege 15 password 0 http

R1#copy http://153.2.2.2/route.txt null:
Loading http://153.2.2.2/route.txt !
1118 bytes copied in 0.236 secs (4737 bytes/sec)

```

In IOS routers, we cannot figure usernames or passwords in the SLA definition, either. There appears to be no way to perform HTTP operations with authentication when using IP SLA at this time. An HTTP raw request might work but that is far beyond the scope of this document.

```

R1(config-ip-sla)#http get http://http:http@153.2.2.2/route.txt
% Bad URL

```

Even worse, in XE you can specify the username/password, but it still does not work.

```
R1(config-ip-sla)#http get http://http:http@153.2.2.2/route.txt
R1(config-ip-sla-http)#
```

Assuming we don't change the original URL string, notice how the probes are still trying to download the files but are not supplying a username or password. The most unfortunate part of this "feature" is that it leads to false positives. Notice how IP SLA ID 106 is suddenly succeeding because the HTTP operation technically completed very quickly. Earlier it was failing due to long download times, but now that the HTTP server is saying "You cannot download the file because you didn't authenticate", IP SLA considers this a success. The probe is effectively a TCP-connect operation at this point. In summary, don't use IP SLA HTTP probes to issue HTTP GET requests to servers that may require authentication. I cannot find a way to disable HTTP server authentication in XE, but it can be disabled in IOS.

```
R1#debug ip http all
its_urlhook url: /tech.txt, method 1
lds_urlhook, url=/tech.txt
Fri, 01 Mar 2002 00:24:48 GMT 153.1.13.1 /tech.txt auth_required
      Protocol = HTTP/1.0 Method = GET
```

```
R1#show ip sla statistics 106 | include Number|HTTP
Latest HTTP Transaction RTT: 111 ms
Number of successes: 2
Number of failures: 6
```

Just to prove my inability to bypass HTTP authentication, I've configured AAA and several authentication/authorization lists that say "none". This should effectively bypass HTTP authentication on XE routers and is applied to CSR2. It still does not work.

```
! CSR2
ip http authentication aaa login-authentication AAA_LOGIN_NONE
ip http authentication aaa exec-authorization AAA_EXEC_NONE
ip http authentication aaa command-authorization 15 AAA_COMMANDS_NONE

R1#debug ip http all
Sun, 05 MON 2015 18:23:44 GMT 153.1.13.1 /route.txt auth_required
      Protocol = HTTP/1.0 Method = GET
```

The DNS probe can be used to perform forward (A or AAAA record) or reverse (PTR record) lookups for a given hostname or IP address, respectively. The minimum frequency for a DNS operation is 9 seconds. Two new IP SLA probes [ID 110, 111] are built to test these two features. The first probe will request an IP address for the name "SLA_TEST.lab.local" while the second will request a hostname for "4.3.2.1". XE does not support DNS server capabilities (but does support the probe), so regular IOS 12.4T was used exclusively for this test.

```

! CSR1
ip sla 110
  dns SLA_TEST.lab.local name-server 153.2.2.2
  frequency 9
ip sla schedule 110 life forever start-time now

ip sla 111
  dns 4.3.2.1 name-server 153.2.2.2
  frequency 9
ip sla schedule 111 life forever start-time now

! CSR2 (IOS only)
ip host R2 153.2.2.2
ip host SLA_TEST.lab.local 1.2.3.4
ip host SLA_REVERSE.lab.local 4.3.2.1

```

I was not able to find any show commands to validate that these DNS lookups were successful, but the IP SLA debug reveals it. IP SLA is smart enough to determine whether an A/AAAA or PTR record resolution is required based on the user's input.

```

R1#debug ip sla trace
IP SLAs(110) Scheduler: saaSchedulerEventWakeup
IP SLAs(110) Scheduler: Starting an operation
IP SLAs(110) dns operation: Starting dns operation
IP SLAs(110) dns operation: Query name - SLA_TEST.lab.local
IP SLAs(110) dns operation: actual target queried = SLA_TEST.lab.local
IP SLAs(110) dns operation: Query return code - no error
IP SLAs(110) dns operation: received IP Address 1.2.3.4
IP SLAs(110) dns operation: RTT=73

IP SLAs(111) Scheduler: saaSchedulerEventWakeup
IP SLAs(111) Scheduler: Starting an operation
IP SLAs(111) dns operation: Starting dns operation
IP SLAs(111) dns operation: Query addr - 4.3.2.1
IP SLAs(111) dns operation: actual target queried = 1.2.3.4.in-addr.arpa
IP SLAs(111) dns operation: Query return code - no error
IP SLAs(111) dns operation: received name SLA_REVERSE.lab.local
IP SLAs(111) dns operation: RTT=59

```

An added benefit of using IP SLA for DNS testing is that I did not have to configure any name-servers or domain-names in global configuration. This means that I would not need to interrupt a working, production DNS configuration to test an unrelated DNS issue.

```

R1#show hosts | include domain
Default domain is not set

```

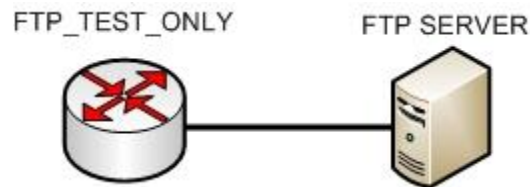
```

! This is a hidden command on regular IOS and XE, perhaps deprecated
R1#show ip name-servers

```

255.255.255.255

IOS routers used to support FTP server many versions ago. Rather than set up a new FTP server for this quick test, I will create a new topology using GNS3 and tap into my laptop as an FTP server. I am moderately confident that the FTP download probe works the same on XE as it would on classic IOS. So far, all of the miscellaneous probes have. Here is a small network diagram for clarity.



The aptly named router `FTP_TEST_ONLY` will download a small file from the FTP server manually, then test it with an IP SLA probe [ID 120]. The minimum frequency is 30 seconds and the default mode is passive FTP. The FTP server is 192.168.0.1 and the IOS router interface is 192.168.0.100. The FTP service requires a username/password of “ftp/ftp” to download files.

```
! FTP_TEST_ONLY
ip ftp username ftp
ip ftp password ftp

ip sla 120
  ftp get ftp://anonymous:test@192.168.0.1/config.txt
  frequency 30
ip sla schedule 120 life forever start-time now

FTP_TEST_ONLY#copy ftp://192.168.0.1/config.txt null:
Accessing ftp://192.168.0.1/config.txt...
Loading config.txt !
[OK - 881/4096 bytes]
881 bytes copied in 0.624 secs (1412 bytes/sec)
```

It looks like the Cisco decided to make the default FTP credentials “anonymous/test” for the IP SLA probe, even though I have an FTP username/password in the configuration. This is very similar to the HTTP probe issues we saw earlier where IP SLA is not talking to the FTP process.

```
FTP_TEST_ONLY#debug ip ftp
FTP: ---> USER anonymous
FTP: 331 Password required for anonymous
FTP: ---> PASS test
FTP: 530 Login or password incorrect!
FTP: ---> QUIT
FTP: 221 Goodbye
```

Interestingly, it automatically inserted that login information into the configuration. As expected, we can actually update it, unlike with HTTP. We will make a new probe for this [ID 121] using active FTP (versus passive FTP, the default). As expected, IP SLA ID 120 is failing due to authentication failure while IP SLA ID 121 is successfully downloading the target file. The key point is that the “ip ftp username” and “ip ftp password” commands in global configuration mode are totally ignored by the IP SLA FTP probes.

```
! FTP_TEST_ONLY
ip sla 120
  ftp get ftp://anonymous:test@192.168.0.1/config.txt
  frequency 30
ip sla schedule 120 life forever start-time now

ip sla 121
  ftp get ftp://ftp:ftp@192.168.0.1/config.txt mode active
  frequency 30
ip sla schedule 121 life forever start-time now

FTP_TEST_ONLY#show ip sla statistics 120 | include Number
Number of successes: 0
Number of failures: 3

FTP_TEST_ONLY#show ip sla statistics 121 | include Number
Number of successes: 4
Number of failures: 0
```

From this point on, IP SLA ID 120 is disabled. We will delete the “config.txt” file from the FTP server and see if IP SLA ID 121 returns errors. At the same time, we will confirm it’s actually broken with a manual download. The FTP operation appears quite reliable without any output deciphering required; the output below depicts newly-caused failures since the file no longer exists on the FTP server.

```
FTP_TEST_ONLY#copy ftp://192.168.0.1/config.txt null:
Accessing ftp://192.168.0.1/config.txt...
%Error opening ftp://192.168.0.1/config.txt (No such file or directory)

FTP_TEST_ONLY#show ip sla statistics 121 | include Number
Number of successes: 7
Number of failures: 3
```

47.7 Aggregated statistics, history, group scheduling, and miscellaneous features

These next three topics are not related to each other and could be considered miscellaneous IP SLA features, but they are important nonetheless.

IP SLA history as a feature is pretty self explanatory; it tracks IP SLA statistics over time and can aggregate results. The configuration of the feature is quite involved and there isn’t a simple “on” button for it. By default, aggregated statistics are collected for a probe. Often times, it is not feasible to collect every statistical component from every probe forever; of greater interest is probably the maximum RTT,

minimum RTT, sum of all RTTs (to compute the mean/average), and sum of all squares of the RTTs (to compute standard deviation). If a probe is run indefinitely in a production network, this statistical information can be very useful to diagnose network issues, plan for growth, etc.

IP SLA statistics introduces many new terms. A “bucket” is defined a range of values, measured in ms, inside which an RTT may lie. Each time the probe returns an RTT value that fits within a bucket, the bucket counter is incremented. For example, the default bucket width is 20 ms and assume we have a custom maximum of 4 buckets. The first bucket would be $[0 \leq b1 < 20]$, the second bucket would be $[20 \leq b2 < 40]$, the third bucket would be $[40 \leq b3 < 60]$, and the fourth bucket would be $[b \geq 60]$.

Below is a list of statistical aggregation-related options with their descriptions.

distributions-of-statistics-kept: controls the number of buckets (default 1). This is limited to ICMP echo, UDP echo, TCP-connect, Path-Echo, and DNS probes.

hours-of-statistics-kept: how many hours to collect before purging old results (default 2).

statistics-distribution-interval: controls the bucket width (default 20ms). This is limited to ICMP echo, UDP echo, TCP-connect, Path-Echo, and DNS probes. These are similar to the bins/buckets discussed in the Y.1731 PM section.

It would be nice to be able to collect all statistical information for a probe for a specific troubleshooting or measurement purpose; that is, statistical aggregation is not always desirable because it is very coarse. We can use “history enhanced” for collecting additional details. This specific collection feature is not on by default, but the aggregated data is automatically collected. The aforementioned parameters are tunable but do not “enable” aggregation as it is always on. Enhanced history introduces some new variables.

enhanced interval X buckets Y: X is measured in seconds and represents how wide a bucket is. Y is simply the number of buckets, and the product $X*Y$ is how long you can collect data. To collect data for one hour, you could make 60 buckets of 60 seconds each, for a total of $60*60=3600$ seconds. X must be greater or equal to the probe frequency. Default values: $X=900$ seconds (15 minutes), $Y=100$. Buckets are circular buffers and will rewrite the oldest data with the newest data. Not to be confused with “buckets” in statistical aggregation, these buckets store a collection of results over a period of time and do not attempt to sort them.

lives-kept: A life is the duration for which a probe runs. Often times we type “life forever” without thinking, but a probe that gets started/stopped regularly is said to be starting/ending “lives”. The default is 0, which effectively means history is disabled. You must set a non-zero value to enable history for a probe.

filter: Additional options exist as to when a history event should be recorded. Self-explanatory options include all, none, overThreshold, and failures (timeout exceeded). The default is none, so you have to pick another option to collect history. This is somewhat like an “on” button for the feature.

Rather than create new probes, we can tune our existing probes to collect statistics. Let's start with IP SLA ID 1, which was our basic ICMP-echo probe. Here is our task:

1. Measure RTTs across 4 buckets of width 25 ms for aggregated statistics. This assumes that most RTTs should be within 1 to 75 ms, since anything larger than that will end up in the final bucket.
2. Keep detailed statistics for 75 minutes
3. Purge all statistics when the probe is stopped and restarted
4. Maintain detailed history for each probe, remembering as far back as 15 probes
5. Record all events

To meet those requirements, the configuration would look like this. I've added in-line comments to help clarify. Regarding the "purge" requirement, only 1 "life" is kept, so restarting the probe will purge the old statistics once the probe restarts.

```
ip sla 1
 icmp-echo 153.2.2.2 source-interface GigabitEthernet2.512
 frequency 5
 history distributions-of-statistics-kept 4      ! 4 aggregation buckets ...
 history statistics-distribution-interval 25    ! of width 25 ms
 history lives-kept 1                          ! remember stats from 1 life
 history filter all                           ! record all events
 history enhanced interval 5 buckets 15        ! hist. details, 5*15 = 75 min
```

As expected, the statistical data is a lot to read for a human. What is interesting is the history information and aggregated statistics aren't both under the "history" show command tree. Technically aggregated statistics are not history, but in newer versions of code, they are configured under the "history" tree in the IP SLA configuration. Enhanced history is specific to exact probe results. Notice in the output below that the bucket index starts at 12 and goes to 26. This is because buckets "wrap", and after we hit bucket 16, we had to forget about bucket 1 by rewriting it. There are always up to 15 buckets, as we programmed. Because we set the history interval equal to the frequency, we essentially capture one sample per interval, which goes into a single bucket. The sample time is measured in absolute ms; notice that it increases by 5000 each row. The entry number is the IP SLA ID and the life resets when the probe is restarted by the system. A manual restart using "ip sla restart [id]" will always reset the life to 1.

```
R1#show ip sla history 1
      Point by point History
Entry    = Entry number
LifeI    = Life index
BucketI  = Bucket index
SampleI  = Sample index
SampleT  = Sample start time (milliseconds)
CompT    = RTT (milliseconds)
Sense    = Response return code
```


Entry	LifeI	BucketI	SampleI	SampleT	CompT	Sense	TargetAddr
1	1	12	1	77890478	1	1	153.2.2.2
1	1	13	1	77895478	1	1	153.2.2.2
1	1	14	1	77900478	1	1	153.2.2.2
1	1	15	1	77905478	1	1	153.2.2.2
1	1	16	1	77910478	1	1	153.2.2.2
1	1	17	1	77915478	1	1	153.2.2.2
1	1	18	1	77920478	1	1	153.2.2.2
1	1	19	1	77925478	1	1	153.2.2.2
1	1	20	1	77930478	1	1	153.2.2.2
1	1	21	1	77935478	1	1	153.2.2.2
1	1	22	1	77940478	1	1	153.2.2.2
1	1	23	1	77945478	1	1	153.2.2.2
1	1	24	1	77950478	1	1	153.2.2.2
1	1	25	1	77955478	1	1	153.2.2.2
1	1	26	1	77960478	1	1	153.2.2.2

The aggregated statistics used to be displayed in a tabular fashion but is no longer. At the top, we have a summary of the usual IP SLA statistics. It gives us a more detailed breakdown as to why failures occurred also. I've separated the buckets with a newline just to increase readability. The bucket starts by stating its range in ms, followed by the average latency (it really means RTT). Notice the sum of the RTT squares; because this could potentially be a huge number, it is split into two unsigned 32-bit integers. Assuming the high-order 32-bits are 0, the low order 32-bits represent the true sum of RTT squares. If the high-order bits are not 0, then you should convert both numbers to binary, concatenate them, then convert back to decimal. A machine-friendly operation would be to left-shift the high-order number 32 times, then add the low-order number for the same result. Notice that the other 3 buckets have no values because our RTT is very fast; all returned RTTs fall into the first bucket.

```
R1#show ip sla statistics aggregated 1 details
IPSLAs aggregated statistics
IPSLA operation id: 1
Type of operation: icmp-echo
Start Time Index: [snip]
RTT Values
    Number Of RTT: 46                      RTT Min/Avg/Max: 1/1/6 milliseconds
Number of successes: 46
Number of failures: 0
Number of over thresholds: 0
Failed Operations due to Disconnect/TimeOut/Busy/No Connection: 0/0/0/0
Failed Operations due to Internal/Sequence/Verify Error: 0/0/0
Failed Operations due to Control enable/Stats retrieve Error: 0/0

Distribution Statistics:
Bucket Range: 0 to < 25ms
Avg. Latency: 1 ms
Percent of Total Completions for this Range: 100 %
Number of Completions/Sum of Latency: 46/51
Sum of RTT squared low 32 Bits/Sum of RTT squared high 32 Bits: 81/0
```

Bucket Range: 25 to < 50ms

Avg. Latency: 0 ms
Percent of Total Completions for this Range: 0 %
Number of Completions/Sum of Latency: 0/0
Sum of RTT squared low 32 Bits/Sum of RTT squared high 32 Bits: 0/0

Bucket Range: 50 to < 75ms

Avg. Latency: 0 ms
Percent of Total Completions for this Range: 0 %
Number of Completions/Sum of Latency: 0/0
Sum of RTT squared low 32 Bits/Sum of RTT squared high 32 Bits: 0/0

Bucket Range: >=75 ms

Avg. Latency: 0 ms
Percent of Total Completions for this Range: 0 %
Number of Completions/Sum of Latency: 0/0
Sum of RTT squared low 32 Bits/Sum of RTT squared high 32 Bits: 0/0

Let's do another example with an HTTP download, reusing IP SLA ID 106 to download a large file. A quick manual test says this should take about 350 ms on a CSR1000v with the 100 kbps throughput limitation. Here are the requirements:

1. Measure RTTs within 8 buckets of maximal width for aggregated statistics
2. Keep statistics for four hours
3. Keep statistics through as many probe restarts as possible

Notice the requirements did not specify history or any of its affiliated parameters. These features are not supported for all probes, and HTTP happens to be one of them:

```
R1(config-ip-sla-http)#history filter all
%filter-for-history cannot be set for this probe
```

```
R1(config-ip-sla-http)#history lives-kept 2
%lives-of-history-kept cannot be set for this probe
```

```
R1(config-ip-sla-http)#history buckets-kept 3
%buckets-of-history-kept cannot be set for this probe
```

Here is the configuration given the aggregated statistics collection:

```
! CSR1
ip sla 106
  http get http://http:http@153.2.2.2/tech.txt
  threshold 2000
  timeout 30000
  history hours-of-statistics-kept 4
```

```
history distributions-of-statistics-kept 8
history statistics-distribution-interval 100
```

Unfortunately, we cannot test this in our current environment. Remember that XE refuses to let me disable HTTP authentication and no matter what, IP SLA will not issue the authentication parameters without a raw request. Let's look at the statistics anyway to confirm the configuration. I'm not going to show all 8 of the buckets. We do get some extra HTTP statistics since it relies both on TCP and sometimes DNS too. It is also interesting that the success counter increases despite the authentication failing on the server, as noted earlier.

```
R1#show ip sla statistics aggregated 106 details
IPSLAs aggregated statistics
```

```
IPSLA operation id: 106
Type of operation: http
Start Time Index: [sni[]
DNS RTT: 0
TCP Connection RTT: 9
HTTP Transaction RTT: 34
HTTP time to first byte: 42
DNS TimeOut: 0
TCP TimeOut: 0
Transaction TimeOut: 0
DNS Error: 0
TCP Error: 0
Transaction Error: 0
Number of successes: 9
Number of failures: 0
Number of over thresholds: 0
Failed Operations due to Disconnect/TimeOut/Busy/No Connection: 0/0/0/0
Failed Operations due to Internal/Sequence/Verify Error: 0/0/0
Failed Operations due to Control enable/Stats retrieve Error: 0/0
```

Distribution Statistics:

```
Bucket Range: 0 to < 100ms
Avg. Latency: 4 ms
Percent of Total Completions for this Range: 100 %
Number of Completions/Sum of Latency: 9/43
Sum of RTT squared low 32 Bits/Sum of RTT squared high 32 Bits: 211/0
```

```
Bucket Range: 100 to < 200ms
Avg. Latency: 0 ms
Percent of Total Completions for this Range: 0 %
Number of Completions/Sum of Latency: 0/0
Sum of RTT squared low 32 Bits/Sum of RTT squared high 32 Bits: 0/0
```

[snip, middle buckets omitted for brevity]

Bucket Range: >=700 ms

Avg. Latency: 0 ms

Percent of Total Completions for this Range: 0 %

Number of Completions/Sum of Latency: 0/0

Sum of RTT squared low 32 Bits/Sum of RTT squared high 32 Bits: 0/0

Although technically not history, some probes, such as path-echo and MPLS traceroute, have some additional statistical components. The amount of memory required to store all of the statistical data for this probe is the product of “history distributions-of-statistics-kept” and all three of the following fields. Increasing or reducing any of them results in more or less memory consumed, respectively.

paths-of-statistics-kept: Maximum number of statistics paths to capture (default unknown). Each unique path discovered by the probe counts as a different path and is assigned a unique index.

hops-of-statistics-kept: Maximum number of hops to capture (default 16). Each path index has list of hops associated with it, and this specifies the maximum number of hops to be stored per path index.

samples-of-history-kept: Maximum number of history samples to collect (default 2).

Because we have already examined the path/hop probe feature earlier, trying to see the effects of all of these variables is hard in a small network. We will look at the “hops” variable since that is easiest to see by adjusting an existing probe [ID 10] on CSR1. I’ve configured the maximum hops to 2, knowing that we actually have 3 hops to the destination. Notice that the aggregated statistics only show the hops to reach XRv4, but not the final destination as we saw earlier. The probe is not allowed to capture more than 2 hops per path index, so in this case, the path is incomplete.

```
! CSR1
ip sla 10
  path-echo 153.2.2.2
  threshold 50
  timeout 100
  frequency 10
  hops-of-statistics-kept 2
  samples-of-history-kept 10
ip sla schedule 10 life forever start-time now
```

```
R1#show ip sla statistics aggregated 10
IPSLAs aggregated statistics
IPSLA operation id: 10
Start Time Index: 01:40:52.017 UTC [snip]
Path Index: 1
```

Hop in Path Index: 1

Type of operation: path-echo

Number of successes: 2

Number of failures: 0

Target Address 153.1.13.13

```
Start Time Index: 01:40:52.017 UTC [snip]
Path Index: 1
Hop in Path Index: 2
Type of operation: path-echo
Number of successes: 2
Number of failures: 0
Target Address 153.13.14.14
```

Next, we progress to group scheduling. In name, this is also self-explanatory, and is a way to schedule multiple IP SLAs simultaneously. The parameters themselves can be confusing and require further explanation. Three new IP SLAs have been defined using the ICMP-echo probe [ID 152, 153, 154]. The three of them ping CSR2, XRv3, and XRv4 loopbacks and should be scheduled to run as a group but not at exactly the same time. There is nothing special about the probes themselves since this section focuses on the scheduling; notice that none of these probes have been scheduled.

```
! CSR1
ip sla 152
 icmp-echo 152.2.2.2 source-interface Loopback0
 frequency 10

ip sla 153
 icmp-echo 152.13.13.13 source-interface Loopback0
 frequency 12

ip sla 154
 icmp-echo 152.14.14.14 source-interface Loopback0
 frequency 15
```

We can use the “schedule-period” parameter to set the even interval at which probes are started. This overrides the individual probe frequency; take the schedule-period divided by the number of probes to get the new frequency (in this case, 60 sec / 3 probes = 20 sec). With timestamps enabled, we see the probes to each destination exactly 20 seconds apart. The entire group schedule [ID 150] is spaced 2 minutes apart (first set of three operations started at 22:26:28, the next one was 22:28:28) which is what the group frequency controls. This is a good approach to probe many destinations spread out over time to determine if there are any large scale failures. As a general restriction, the group frequency, just like the individual probe frequency, cannot be less than the timeout of any individual probe. The different probes are highlighted using different colors for clarify. Again, each one starts exactly 20 seconds apart.

```
! CSR1
ip sla group schedule 150 152-154 schedule-period 60 frequency 120

22:26:28: IPSLA-INFRA_TRACE:OPER:152 Starting an operation
22:26:28: IPSLA-OPER_TRACE:OPER:152 Starting icmpecho operation -
destAddr=153.2.2.2, sAddr=153.1.1.1
22:26:28: IPSLA-OPER_TRACE:OPER:152 ID:340, RTT=1
```

```

22:26:48: IPSLA-INFRA_TRACE:OPER:153 Starting an operation
22:26:48: IPSLA-OPER_TRACE:OPER:153 Starting icmpcho operation -
destAddr=153.13.13.13, sAddr=153.1.1.1
22:26:48: IPSLA-OPER_TRACE:OPER:153 ID:341, RTT=1

22:27:08: IPSLA-INFRA_TRACE:OPER:154 Starting an operation
22:27:08: IPSLA-OPER_TRACE:OPER:154 Starting icmpcho operation -
destAddr=153.14.14.14, sAddr=153.1.1.1
22:27:08: IPSLA-OPER_TRACE:OPER:154 ID:342, RTT=2

22:28:28: IPSLA-INFRA_TRACE:OPER:152 Starting an operation
22:28:28: IPSLA-OPER_TRACE:OPER:152 Starting icmpcho operation -
destAddr=153.2.2.2, sAddr=153.1.1.1
22:28:28: IPSLA-OPER_TRACE:OPER:152 ID:343, RTT=1

```

An alternative approach is to schedule them together. There is no scheduling period used for balanced distribution of probes; they are all start at the same time but still have their frequencies overwritten by the group schedule. In this case, the group frequency is 60 seconds, at which time all probes are issued at the same time. The timestamps are highlighted below to prove it. After 60 seconds, the scheduling process restarts.

```

! CSR1
ip sla group schedule 150 152-154 schedule-together frequency 60

22:41:36: IPSLA-INFRA_TRACE:OPER:152 Starting an operation
22:41:36: IPSLA-OPER_TRACE:OPER:152 Starting icmpcho operation -
destAddr=153.2.2.2, sAddr=153.1.1.1
22:41:36: IPSLA-OPER_TRACE:OPER:152 Sending ID: 359

22:41:36: IPSLA-INFRA_TRACE:OPER:153 Starting an operation
22:41:36: IPSLA-OPER_TRACE:OPER:153 Starting icmpcho operation -
destAddr=153.13.13.13, sAddr=153.1.1.1
22:41:36: IPSLA-OPER_TRACE:OPER:153 Sending ID: 360

22:41:36: IPSLA-INFRA_TRACE:OPER:154 Starting an operation
22:41:36: IPSLA-OPER_TRACE:OPER:154 Starting icmpcho operation -
destAddr=153.14.14.14, sAddr=153.1.1.1
22:41:36: IPSLA-OPER_TRACE:OPER:154 Sending ID: 361

22:41:36: IPSLA-OPER_TRACE:OPER:152 ID:359, RTT=1
22:41:36: IPSLA-OPER_TRACE:OPER:153 ID:360, RTT=1
22:41:36: IPSLA-OPER_TRACE:OPER:154 ID:361, RTT=2

22:42:36: IPSLA-INFRA_TRACE:OPER:152 Starting an operation
[snip]

```

You can combine the schedule-together or schedule-period options with a range of frequencies which is often called “random scheduling”. This can improve statistical metrics for measuring network performance because sampling at given intervals might yield biased results. When used with schedule-period, the initial probes are scheduled at random intervals within the bounds of the schedule-period (it is no longer a uniform spread). The specification of frequency is identical to the previous example except the interval is random within the specified range. In the “closest” case, each probe could be spaced up to 10 seconds apart with a frequency of 45 seconds, which is acceptable.

The initial scheduler starts ID 153 and ID 154 at different intervals after starting the first probe, which was ID 152. Notice that 154 had a smaller value, so it starts about a second before ID 153 in the debugs below. Each operation runs independently, and at the end, generates a new random frequency between 45 and 55 seconds. I did not enable msec granularity because it's not required to understand the technology. We can see that ID 152 restarts ~48 seconds later (23:04:22 to 23:05:10), ID 153 restarts ~51 seconds later (23:04:23 to 23:05:14), and ID 154 restarts about ~52 seconds later (23:04:23 to 23:05:15). At the end of the second set of probes where I have snipped most of the output, a new set of frequencies is generated, and the process continues. The frequency changes after each operation which is what achieves the pseudo-randomness.

```
! CSR1
ip sla group schedule 150 152-154 schedule-period 10 frequency range 45-55

R1#debug ip sla trace 150
23:04:22: IPSLA-OPER_TRACE:OPER:150 Group scheduling probe 153 after 1488
msec null
23:04:22: IPSLA-OPER_TRACE:OPER:150 Group scheduling probe 154 after 354 msec
null

23:04:22: IPSLA-INFRA_TRACE:OPER:152 Starting an operation
23:04:22: IPSLA-OPER_TRACE:OPER:152 Starting icmp echo operation -
destAddr=153.2.2.2, sAddr=153.1.1.1
23:04:22: IPSLA-OPER_TRACE:OPER:152 ID:424, RTT=1
23:04:22: IPSLA-INFRA_TRACE:OPER:152 frequency generated (msec) = 47097

23:04:23: IPSLA-INFRA_TRACE:OPER:154 Starting an operation
23:04:23: IPSLA-OPER_TRACE:OPER:154 Starting icmp echo operation -
destAddr=153.14.14.14, sAddr=153.1.1.1
23:04:23: IPSLA-OPER_TRACE:OPER:154 ID:425, RTT=1
23:04:23: IPSLA-INFRA_TRACE:OPER:154 frequency generated (msec) = 52681

23:04:24: IPSLA-INFRA_TRACE:OPER:153 Starting an operation
23:04:24: IPSLA-OPER_TRACE:OPER:153 Starting icmp echo operation -
destAddr=153.13.13.13, sAddr=153.1.1.1
23:04:24: IPSLA-OPER_TRACE:OPER:153 ID:426, RTT=1
23:04:24: IPSLA-INFRA_TRACE:OPER:153 frequency generated (msec) = 50427

23:05:10: IPSLA-INFRA_TRACE:OPER:152 Starting an operation
```

```
23:05:10: IPSLA-INFRA_TRACE:OPER:152 frequency generated (msec) = 52688
```

```
23:05:14: IPSLA-INFRA_TRACE:OPER:153 Starting an operation
```

```
23:05:14: IPSLA-INFRA_TRACE:OPER:153 frequency generated (msec) = 54178
```

```
23:05:15: IPSLA-INFRA_TRACE:OPER:154 Starting an operation
```

```
23:05:15: IPSLA-INFRA_TRACE:OPER:154 frequency generated (msec) = 45368
```

Random scheduling with schedule-together crashes the XE router 100% of the time, so I suppose I won't be testing it. This is CSR1000v version 3.13.2S. Random scheduling with schedule-period, tested above, appears to work well and never crashes the router.

```
R1(config)#ip sla group schedule 150 152-154 schedule-together frequency  
range 45-65 start-time now life forever
```

```
Exception to IOS Thread:
```

```
Frame pointer 0x7F2D7468D3A0, PC = 0x49A5A60
```

```
[snip]
```

IP SLA automatic provisioning is a feature where probes can be automatically spawned from a template and endpoint-list. The template resembles an IP SLA configuration minus the targets. The endpoint-list contains only the target information. This is similar to P2P TE Auto-Template (for auto-tunnel meshing) where the single interface spawns many P2P TE tunnels from a single interface. The feature can also be used for dynamic discovery by configuring a responder that can return a list of endpoints to clients requesting it. A use case could include several hundred branch nodes wanting to probe an unknown/constantly changing set of addresses to measure RTT. IP SLA auto-register is not supported on the CSR1000v (version 3.13.2S), despite the IOS-XE 3S documentation clearly stating that it should be. Likewise, creating auto templates is not possible.

```
R2(config)#ip sla responder auto-register AUTOREG
```

```
Auto Register is not supported.
```

```
R1#show ip sla auto template
```

```
[no output]
```

You can create endpoint-lists, which are useless since the feature isn't supported, and the show commands for the "auto" syntax tree do exist. These lists would serve as inputs to the auto-discovery process, assuming it was supported. I document the concept here in case future releases support it.

```
! CSR1
```

```
ip sla endpoint-list type ip TEST
```

```
ip-address 5.5.5.5 port 222
```

```
ip sla endpoint-list type ip LOOPBACKS
```

```
ip-address 153.2.2.2,153.13.13.13,153.14.14.14 port 55588
```



```

R1#show ip sla endpoint-list
Endpoint-list Name: TEST
  Description:
    List of IPV4 Endpoints:
    ip-address 5.5.5.5 port 222

Endpoint-list Name: LOOPBACKS
  Description:
    List of IPV4 Endpoints:
    ip-address 153.2.2.2,153.13.13.13,153.14.14.14 port 55588

R1(config)#ip sla auto
      ^
% Invalid input detected at '^' marker.

```

IP SLA Two-Way Active Measurement Protocol (TWAMP), an IETF standard protocol, is a way to perform monitoring using another mechanism similar to ordinary IP SLA. Cisco routers can be servers and session-reflectors, but not control-clients or session-senders. The control-clients start and stop sessions, kind of like probing IP SLA devices. The server and control-clients exchange TWAMP-Control messages which are comparable to the IP SLA control packets on UDP ports 1967 (v1) or 1167 (v2). The session-senders send TWAMP-Test (data measurement) packets to session-reflectors, which notify the other session-senders. Servers can accept connections from multiple clients also.

In case you didn't know TWAMP is an IETF protocol, the router reveals the RFC. This command exists to tell you which standards were implemented and which TWAMP roles are supported. This proves the claim earlier that the router is capable of only 2 of the 4 possible TWAMP roles.

```

R2#show ip sla twamp standards

```

Feature	Organization	Standard
TWAMP Server	IETF	RFC5357
TWAMP Reflector	IETF	RFC5357

Below, we configure CSR2 as a TWAMP session-reflector ("responder" command) and a server ("server" command). For TWAMP v1.0, which is the version currently supported, these roles must be on the same device. The following show command shows that you've properly enabled TWAMP as no sessions currently exist. I have not tested TWAMP with a client yet, and since XE and XR don't support that role, it's probably not in scope for the exam.

```

! CSR2
ip sla responder twamp
  timeout 60

ip sla server twamp
  port 60000
  timer inactivity 60

```

```
R2#show ip sla twamp session
IP SLAs Responder TWAMP is: Enabled
```

47.8 Enhanced Object Tracking (EOT)

Enhanced Object Tracking (EOT) is a way for IP SLA results (among other things) to ultimately interact with several unrelated features. For example, a policy-based routing (PB) use-case would include testing reachability to a next-hop. If the next-hop is reachable and fast, send DSCP EF traffic over the link along with DSCP AF11 data flows. If it's reachable but slow, send only data DSCP AF11 over the link. If it's completely down, don't use it at all. IP SLAs can be in one of three non-error states as we discussed earlier: OK, timeout, or over threshold. However, objects can only be up or down. This implies that we need at least two different tracking mechanisms to determine whether or not "over threshold" qualifies as down or not.

We will implement the aforementioned PBR policy as an example. Engineering a test for this in the data-plane is a little tricky. I apply an 8 kbps shaper outbound on CSR1 on a new link to CSR2 (no IGP).

```
! CSR1
policy-map SHAPE_8K
  class class-default
    shape average 8000

interface GigabitEthernet2.521
  encapsulation dot1Q 3521
  ip address 12.0.0.1 255.255.255.0
  service-policy output SHAPE_8K

! CSR2
interface GigabitEthernet3.521
  encapsulation dot1Q 3521
  ip address 12.0.0.2 255.255.255.0
```

I create a probe [ID 170] to test this link using a large packet. I then create a congestion source [IP 171] to send a large packet every second, which is about 12 kbps of traffic. We only care about IP SLA ID 170 and will use that for our EOT testing; IP SLA ID 171 exists purely to create congestion. While this congestion probe is running, the primary probe will alternate between over threshold and timeout, depending.

```
! CSR1
ip sla 170
  icmp-echo 12.0.0.2 source-interface GigabitEthernet2.521
  request-data-size 1450
  threshold 1000
  frequency 5

ip sla 171
  icmp-echo 12.0.0.2 source-interface GigabitEthernet2.521
```

```
request-data-size 1450
threshold 1000
timeout 1000
frequency 1
```

```
ip sla schedule 170 life forever start-time now
ip sla schedule 171 life forever start-time now
```

We create two tracked objects, 10 and 11, that track the state and reachability of our primary probe, respectively. The “state” option is more strict and requires an OK return code (and is the default option), whereas “reachability” is up even if the probe is over threshold. If your probe threshold and timeout are the same value, state and reachability do exactly the same thing from a practical perspective. We can tie these objects into the PBR policy so we can achieve the desired forwarding result.

```
! CSR1
track 10 ip sla 170 state
track 11 ip sla 170 reachability

ip access-list extended ACL_DSCP_EF
 permit ip any any dscp ef

ip access-list extended ACL_DSCP_AF11
 permit ip any any dscp af11

route-map RM_PBR permit 10
 match ip address ACL_DSCP_EF
 set ip next-hop verify-availability 12.0.0.2 10 track 10

route-map RM_PBR permit 20
 match ip address ACL_DSCP_AF11
 set ip next-hop verify-availability 12.0.0.2 10 track 11
```

We apply the route-map globally to CSR1. This affects only locally-generated traffic (be sure to define the ACLs or else the first clause will match everything, breaking many control plane mechanisms).

```
! CSR1
ip local policy route-map RM_PBR
```

First, we will test an uncongested link where both objects should be up and IP SLA ID 171 is not running.

```
R1#show track brief
```

Track	Type	Instance	Parameter	State	Last Change
10	ip sla	170	state	Up	00:01:09
11	ip sla	170	reachability	Up	00:01:09

We will a “voice” probe to generate UDP echoes on port 16384 to CSR2 [ID 172] marked DSCP EF (TOS

184). We also create a “data” probe using TCP with marking DSCP AF11 (TOS 40) [ID 173]. We start both probes together, and we expect both to be policy-routed.

```
! CSR1
ip sla 172
  udp-echo 153.2.2.2 16384
  tos 184
  frequency 5

ip sla 173
  tcp-connect 153.2.2.2 8080
  tos 40
  timeout 5000
  frequency 5

ip sla schedule 172 life forever start-time now
ip sla schedule 173 life forever start-time now
```

Notice that CSR1 marks both the control and data probes (2 packets each) with the specified TOS, and PBR matches both; this is highly desirable as the signaling and bearer traffic will follow the same path. The voice traffic matches “item 10” which is the first route-map entry while the data traffic matches “item 20”. So far, this is indicative of a working configuration.

```
CSR1#debug ip policy
IP: s=153.1.2.1 (local), d=153.2.2.2, len 80, policy match
IP: route map RM_PBR, item 10, permit
IP: s=153.1.2.1 (local), d=153.2.2.2 (GigabitEthernet2.521), len 80, policy
routed
IP: local to GigabitEthernet2.521 12.0.0.2
IP: s=153.1.2.1 (local), d=153.2.2.2, len 44, policy match
IP: route map RM_PBR, item 10, permit
IP: s=153.1.2.1 (local), d=153.2.2.2 (GigabitEthernet2.521), len 44, policy
routed
IP: local to GigabitEthernet2.521 12.0.0.2

IP: s=153.1.2.1 (local), d=153.2.2.2, len 40, policy match
IP: route map RM_PBR, item 20, permit
IP: s=153.1.2.1 (local), d=153.2.2.2 (GigabitEthernet2.521), len 40, policy
routed
IP: local to GigabitEthernet2.521 12.0.0.2
IP: s=153.1.2.1 (local), d=153.2.2.2, len 40, policy match
IP: route map RM_PBR, item 20, permit
IP: s=153.1.2.1 (local), d=153.2.2.2 (GigabitEthernet2.521), len 40, policy
routed
IP: local to GigabitEthernet2.521 12.0.0.2
```

Now we will enable congestion on that link by starting IP SLA ID 171 (not shown). We expect the voice

traffic to not be policy-routed but the data traffic still should be. Track 10 should go down but track 11 should generally stay up. We can see this by looking at the tracked objects, at the route-map directly (in the case of PBR), or implying it from the IP SLA statistics showing failed probes. The “state” is down because the probe is over threshold, but “reachability” is still achievable; the link is “good enough” for data transport.

```
R1#show track brief
```

Track	Type	Instance	Parameter	State	Last Change
10	ip sla	170	state	Down	00:00:09
11	ip sla	170	reachability	Up	00:02:54

```
R1#show route-map RM_PBR
```

```
route-map RM_PBR, permit, sequence 10
```

```
Match clauses:
```

```
ip address (access-lists): ACL_DSCP_EF
```

```
Set clauses:
```

```
ip next-hop verify-availability 12.0.0.2 10 track 10 [down]
```

```
Policy routing matches: 556 packets, 26296 bytes
```

```
route-map RM_PBR, permit, sequence 20
```

```
Match clauses:
```

```
ip address (access-lists): ACL_DSCP_AF11
```

```
Set clauses:
```

```
ip next-hop verify-availability 12.0.0.2 10 track 11 [up]
```

```
Policy routing matches: 978 packets, 45398 bytes
```

```
R1#show ip sla statistics 170
```

```
IPSLAs Latest Operation Statistics
```

```
IPSLA operation id: 170
```

```
Latest RTT: 2630 milliseconds
```

```
Latest operation start time: 02:19:21 [snip]
```

```
Latest operation return code: Over threshold
```

```
Number of successes: 388
```

```
Number of failures: 13
```

```
Operation time to live: Forever
```

The voice traffic has the control packet and data packet, both of which were not policy routed. However, we still have policy-routing for item 20 containing the AF11 flow. Notice the debug reveals there was a match in item 10 but the traffic was forwarded normally.

```
! CSR1
```

```
IP: s=153.1.2.1 (local), d=153.2.2.2, len 80, policy match
```

```
IP: route map RM_PBR, item 10, permit
```

```
IP: s=153.1.2.1 (local), d=153.2.2.2, len 80, policy rejected -- normal forwarding
```

```
IP: s=153.1.2.1 (local), d=153.2.2.2, len 80, policy match
```

```
IP: route map RM_PBR, item 20, permit
```

```
IP: s=153.1.2.1 (local), d=153.2.2.2 (GigabitEthernet2.521), len 80, policy
routed
IP: local to GigabitEthernet2.521 12.0.0.2
```

Eliminating reachability altogether is simple; just break the link completely. I will mismatch the VLAN encapsulations to cause this failure (not shown), and expect no traffic to be policy routed. You should not rely on the packet counters of the route-map to determine whether PBR occurs. The counters increase when a match is made, whether routing happens in the route-map clause or not. I prove this by clearing the counters, enabling debug, and watching them increase despite both objects being down. The debug shows a policy match but does not say policy-routed; the two terms are different. Like earlier, we have policy matches but normal forwarding.

```
! CSR1
IP: s=153.1.2.1 (local), d=153.2.2.2, len 80, policy match
IP: route map RM_PBR, item 10, permit
IP: s=153.1.2.1 (local), d=153.2.2.2, len 80, policy rejected -- normal
forwarding
```

```
IP: s=153.1.2.1 (local), d=153.2.2.2, len 40, policy match
IP: route map RM_PBR, item 20, permit
IP: s=153.1.2.1 (local), d=153.2.2.2, len 40, policy rejected -- normal
forwarding
```

```
R1#show route-map RM_PBR | include route-map|matches|avail
route-map RM_PBR, permit, sequence 10
    ip next-hop verify-availability 12.0.0.2 10 track 10 [down]
    Policy routing matches: 168 packets, 5712 bytes
route-map RM_PBR, permit, sequence 20
    ip next-hop verify-availability 12.0.0.2 10 track 11 [down]
    Policy routing matches: 420 packets, 18144 bytes
```

Although not directly related to IP SLA, I will discuss a several other handy features of object tracking. Objects can be grouped together in lists where the state of the list-object is the logical combination of the component objects. The Boolean list is used to combine objects with logical AND, OR, and NOT operations. The list itself is an object which can be used in other places. As an example, we create an object to be up only if objects 10 and 11 are both down. Since both objects 10 and 11 are still down, object 12 should be up.

```
! CSR1
track 12 list boolean and
    object 10 not
    object 11 not
```

```
R1#show track brief
```

Track	Type	Instance	Parameter	State	Last Change
10	ip sla	170	state	Down	00:10:17
11	ip sla	170	reachability	Down	00:10:17

```
12      list                                boolean      Up      00:00:24
```

If I fix the encapsulation error on the link, we expect at least “reachability” to return, which will bring down the new object list. Whether we achieve “state” or not is irrelevant since both objects must be down in order to bring the list up.

```
R1#show track brief
Track Type      Instance      Parameter      State Last Change
10   ip sla     170           state         Up    00:00:14
11   ip sla     170           reachability   Up    00:00:14
12   list                               boolean       Down   00:00:14
```

Interface tracking is one of the simplest forms of tracking. You can track either the interface line-protocol (which is only really valuable for certain link-types, such as serial interfaces) or the IP routing capability. I’ve added a new Loopback1 to CSR1 to test, along with a tracked object to test line-protocol [ID 20] tracking first.

```
! CSR1
interface Loopback1
 ip address 100.1.1.1 255.255.255.255

track 20 interface Loopback1 line-protocol
```

Line-protocol is self-explanatory. When the router sees the interface line-protocol go down, it brings the object down. A unique characteristic of this object is the “carrier-delay” option; this option is configured under an interface which tells the interface how long to delay, after loss of carrier sense, to tell the router the link is down. If the router doesn’t about a failure, it can’t tell EOT to change state, so we have another knob for delay. I’ve set the carrier-delay to 5 seconds on the loopback. Carrier-delay is discussed in greater detail in the layer 1 failure detection section.

```
! CSR1
track 20 interface Loopback1 line-protocol
 carrier-delay

interface Loopback1
 carrier-delay 5
```

To test it, I issue the “show clock” and “shutdown” commands as quickly as I can; we see the difference in time is about 5 seconds between shutting down the Loopback to the time it reports the event.

```
R1(config-if)#do show clock
23:33:14.039 UTC [snip]

R1(config-if)#shutdown

23:33:19: %LINEPROTO-5-UPDOWN: Line protocol on Interface Loopback1, changed
```

state to down

23:33:19: %LINK-5-CHANGED: Interface Loopback1, changed state to administratively down

23:33:19: %TRACK-6-STATE: 20 interface Lo1 line-protocol Up -> Down

Tracking IP routing on a loopback interface is worthless because it will always be down. Cisco states this feature “tracks an IP route in the routing table and the ability of an interface to route IP packets.” Let’s track CSR1’s link to XRv3 instead. The interface is currently up because the interface is up/up and has a valid IP address.

! CSR1

track 21 interface GigabitEthernet2.513 ip routing

This feature is useful for dynamically negotiated addresses (DHCP, IPCP) because it will not come up until the link has a valid IP address. Removing the IP address from the link causes the interface to fail immediately, but we could use various delay mechanisms to tune that (discussed later). Adding the IP interface back brings the object back up.

R1#show track 21 brief

Track	Type	Instance	Parameter	State	Last Change
21	interface	GigabitEthernet2.513	ip routing	Up	00:00:13

! CSR1

interface GigabitEthernet2.513

no ip address

23:38:07: %TRACK-6-STATE: 21 interface Gi2.513 ip routing Up -> Down

! CSR1

interface GigabitEthernet2.513

ip address 153.1.13.1 255.255.255.0

23:38:16: %TRACK-6-STATE: 21 interface Gi2.513 ip routing Down -> Up

Another way to combine items into a list is using thresholds. We will use our knowledge of lists from earlier combined with interface tracking for a simple example [ID 15-18]. We will track three interfaces on CSR1: link to CSR2, link to XRv13, and Loopback1. Tracking line-protocol on Ethernet interfaces isn’t a great indication of link status, but I use it in this demonstration for simplicity. The transit links have a weight of 5, and the loopback has a weight of 10 (the default). We need a total “score” of 15 out of a maximum of 20 for the object to be up, and falling to 10 or less brings the object down.

! CSR1

track 15 list threshold weight

object 16

object 17 weight 5

object 18 weight 5


```
threshold weight down 10 up 15
```

```
track 16 interface Loopback1 line-protocol
track 17 interface GigabitEthernet2.512 line-protocol
track 18 interface GigabitEthernet2.513 line-protocol
```

Given these thresholds, the loopback must be up, and we can afford to lose up to one transit link. The output below first shows the objects all up.

```
R1#show track 15
Track 15
  List threshold weight
  Threshold Weight is Up (20/20)
    2 changes, last change 00:00:23
    object 16 Up (10/20)
    object 17 weight 5 Up (5/20)
    object 18 weight 5 Up (5/20)
  Threshold weight down 10 up 15
```

Next, we shutdown Loopback1. The individual tracker for that loopback goes down along with the overall threshold list.

```
! CSR1
interface Loopback1
  shutdown

23:54:44: %TRACK-6-STATE: 16 interface Lo1 line-protocol Up -> Down
23:54:44: %TRACK-6-STATE: 15 list threshold weight Up -> Down
```

```
R1#show track 15
Track 15
  List threshold weight
  Threshold Weight is Down (10/20)
    3 changes, last change 00:02:47
    object 16 Down (0/20)
    object 17 weight 5 Up (5/20)
    object 18 weight 5 Up (5/20)
  Threshold weight down 10 up 15
```

If we bring the loopback up but shut down a single transit link, the object stays up. In this example, only the link to CSR2 is shutdown.

```
! CSR1
interface Loopback1
  no shutdown

interface GigabitEthernet2.512
  shutdown
```

```

R1#show track 15
Track 15
  List threshold weight
  Threshold Weight is Up (15/20)
    4 changes, last change 00:00:09
    object 16 Up (10/20)
    object 17 weight 5 Down (0/20)
    object 18 weight 5 Up (5/20)
  Threshold weight down 10 up 15

```

We can also do a percentage-based threshold list [ID 19]. All objects have equal weight and you are effectively saying “if some of my objects are up, the list is up”. With three objects, each one is worth 33% and this is not adjustable. I want to ensure at least 2/3 of my tracked interfaces are up. Having 1 or less is a failure condition; this is defined by the up/down threshold percentages. This is a simpler way to do threshold lists without the arithmetic required for weighted lists.

```

! CSR1
track 19 list threshold percentage
  object 16
  object 17
  object 18
  threshold percentage down 33 up 66

```

```

R1#show track 19
Track 19
  List threshold percentage
  Threshold Percentage is Up (100%)
    4 changes, last change 00:01:10
    object 16 Up (33%)
    object 17 Up (33%)
    object 18 Up (33%)
  Threshold percentage down 33% up 66%

```

Because they are weighted equal, I can shut down the loopback1 and have the list still up, assuming I bring up the link to CSR2.

```

! CSR1
interface Loopback1
  shutdown

interface GigabitEthernet2.512
  no shutdown

```

```

R1#show track 19
Track 19
  List threshold percentage
  Threshold Percentage is Up (66%)

```

```
4 changes, last change 00:00:03
object 16 Down (0%)
object 17 Up (33%)
object 18 Up (33%)
Threshold percentage down 33% up 66%
```

The list cannot tolerate the loss of 2 of more links, so the list fails when the loopback1 is also shut down.

```
! CSR1
interface Loopback1
shutdown
```

```
R1#show track 19
Track 19
List threshold percentage
Threshold Percentage is Down (33%)
3 changes, last change 00:00:04
object 16 Down (0%)
object 17 Down (0%)
object 18 Up (33%)
Threshold percentage down 33% up 66%
```

The track resolution feature measures the scaled metric and compares it to a threshold. Let's evaluate the EIGRP default value of 2560. The EIGRP route metric is scaled (divided) by 2560 to account for larger metrics. That is, every change in 2560 counts as 1 in the scaled metric. Example: For a given route, if the scaled metric exceeds 20 (or a true EIGRP metric value of 51200), then the tracked object goes down. When the metric crosses back below 10 (or a true EIGRP metric value of 25600), the object comes back up. Each protocol has a different default scaling value; tuning these values is independent for IPv4 and IPv6 and this feature does not exist in XR. The default scale values for IPv4 and IPv6 are shown below (only IPv4 is shown as the default values are equal for IPv6).

```
R1#show track resolution ip
```

```
IP Route Resolution

Route type      Metric Resolution
static          10
EIGRP           2560
OSPF             1
ISIS            10
BGP             2560
```

Personally, I've used this feature with FHRPs to ensure that if a potential LAN gateway has poor reachability to some key routes (like server farms) that it should relinquish its active role. IP SLA could be used similarly to actively test link quality to those prefixes, but that means loading the network with synthetic traffic. This feature is passive and doesn't require additional network traffic as it just looks at

IGP/BGP route metrics. You could even write an EEM script to generate a syslog or SNMP trap just to notify someone that the event happened, assuming the metric change was significant enough.

We will create a tracked object [ID 25] to track the route to CSR2. Over the direct path, the metric to CSR2's loopback from CSR1 is 130816. Dividing by 2560 and ignoring the remainder, this should result in a scaled metric of 51 (the scaled metric is not known by the RIB and not shown in this output). The object isn't created yet since we need to determine the threshold values.

```
R1#show ip route 153.2.2.2
Routing entry for 153.2.2.2/32
  Known via "eigrp 1", distance 90, metric 130816, type internal
  Redistributing via eigrp 1
  Last update from 153.1.2.2 on GigabitEthernet2.512, 00:00:05 ago
  Routing Descriptor Blocks:
  * 153.1.2.2, from 153.1.2.2, 00:00:05 ago, via GigabitEthernet2.512
    Route metric is 130816, traffic share count is 1
    Total delay is 5010 microseconds, minimum bandwidth is 1000000 Kbit
    Reliability 255/255, minimum MTU 1500 bytes
```

Let's shut down the interface to CSR2. The EIGRP metric via XRv13 is 131328 which is too close (within 2560) to the metric directly via CSR2, which means we either have to reduce the scaling factor or increase the metric to XRv3. We will increase the delay on the link to XRv3 from 10 to 500. Below is the output before the change in delay is applied.

```
R1#show ip route 153.2.2.2
Routing entry for 153.2.2.2/32
  Known via "eigrp 1", distance 90, metric 131328, type internal
  Redistributing via eigrp 1
  Last update from 153.1.13.13 on GigabitEthernet2.513, 20:10:00 ago
  Routing Descriptor Blocks:
  * 153.1.13.13, from 153.1.13.13, 20:10:00 ago, via GigabitEthernet2.513
    Route metric is 131328, traffic share count is 1
    Total delay is 5030 microseconds, minimum bandwidth is 1000000 Kbit
    Reliability 255/255, minimum MTU 1500 bytes
    Loading 1/255, Hops 3
```

The metric via XRv13 is now 259072 with a scaled metric of 101 after the delay change.

```
! CSR1
interface GigabitEthernet2.513
  delay 500
```

```
R1#show ip route 153.2.2.2
Routing entry for 153.2.2.2/32
  Known via "eigrp 1", distance 90, metric 259072, type internal
  Redistributing via eigrp 1
  Last update from 153.1.13.13 on GigabitEthernet2.513, 00:00:05 ago
```

Routing Descriptor Blocks:

```
* 153.1.13.13, from 153.1.13.13, 00:00:05 ago, via GigabitEthernet2.513
  Route metric is 259072, traffic share count is 1
  Total delay is 10020 microseconds, minimum bandwidth is 1000000 Kbit
  Reliability 255/255, minimum MTU 1500 bytes
  Loading 1/255, Hops 3
```

The object can tell us whether we are routing directly via CSR2 or not by tracking the scaled metric as follows. If the value is less than the “up” threshold, the object is up. If the value is greater than the “down” threshold, the object is down. Using thresholds of $x < 55$ and $x > 95$ will work well given the current EIGRP metric configuration.

```
! CSR1
track 25 ip route 153.2.2.2 255.255.255.255 metric threshold
threshold metric up 55 down 95
```

The object is currently down since we are routing via XRv13. The router shows you the RIB and scaled metrics also. This is a good way to double-check your math. Recall that the scaled metric is the RIB metric divided by the scaling factor (2560 for EIGRP by default).

```
R1#show track 25
Track 25
  IP route 153.2.2.2 255.255.255.255 metric threshold
  Metric threshold is Down (EIGRP/259072/101)
    2 changes, last change 00:00:17
  Metric threshold down 95 up 55
  First-hop interface is GigabitEthernet2.513
```

If we bring up the interface to CSR2, the object comes up. When EIGRP converges, the cost to CSR2 is much lower, resulting in a lower scaled metric of 51.

```
R1#show track 25
Track 25
  IP route 153.2.2.2 255.255.255.255 metric threshold
  Metric threshold is Up (EIGRP/130816/51)
    3 changes, last change 00:00:00
  Metric threshold down 95 up 55
  First-hop interface is GigabitEthernet2.512
```

If desired, you can change the scaled metric defaults. Different protocols have different metric ranges. EIGRP and BGP can have very large metrics/MEDs, while OSPF and ISIS tend to use smaller numbers. RIP, for example, is never scaled, and its metrics are translated directly since they only count to 15. Notice that the minimum and maximum limits in the parser are determined by the protocol types.

```
R1(config)#track resolution ip route ospf ?
<1-1562> Resolution value
```

```
R1(config)#track resolution ip route eigrp ?
<256-40000000> Resolution value
```

The “stub-object” is an interesting feature. There are no fancy options involved; you simply create the object, assign a default-state (if desired), and it’s always up or always down. The exception to this, and one of its best use cases, is to have it be controlled by EEM. EEM is discussed in detail in a dedicated section, but a quick example is shown below. A stub-object [ID 30] is created with default state “down” (which is the default, so the configuration does not display it). A manual EEM script can be run which changes it to “up”. You could also use it in conjunction with other tracked objects within track lists.

```
! CSR1
track 30 stub-object

event manager applet SET_30_UP
event none
action 100 track set 30 state up
```

```
R1#show track 30 brief
```

Track	Type	Instance	Parameter	State	Last
Change					
30	stub-object	Undefined		Down	00:01:18

To demonstrate using EEM to adjust the value, we write a small script and manually execute it from the CLI. Assuming some other process relied on this object, we can use EEM to influence network behavior.

```
! CSR1
event manager applet SET_30_UP
event none
action 100 track set 30 state up
```

```
R1#event manager run SET_30_UP
```

```
R1#show track 30 brief
```

Track	Type	Instance	Parameter	State	Last
Change					
30	stub-object	EEM		Up	00:00:02

The track feature also has timers regarding how often it re-evaluates its tracked objects. There are two sets of timers: a global timer which determines how often the router reassesses the state of an object, and a per-object timer which determines how long to delay going up or going down. We will re-use route tracking object 25 from earlier for this demonstration. The default timers are shown below. Notice that some fields say “expired”; this is because there are zero objects of that specified type. We have not configured any IPv6 route objects, which explains this output. The poll interval is measured in seconds and represents the global timer for that object type. The right-most column just counts down to zero, then restarts. Notice that the stub-object, by default, has a fast timer. The route objects have slower timers to allow protocols to converge without flapping object states constantly. If you are using scaled

metric threshold tracking in your environment, the poll interval for IPv4/IPv6 routes should be slightly longer than your IGP convergence time.

```
R1#show track timers
```

```
Tracking Timers
```

Object type	Poll Interval	Time to next poll
interface	1	0.887
ip route	15	12.607
ip sla	5	0.292
ipv6 route	15	expired
list	1	0.67
stub-object	1	0.230

We will enable timestamps on CSR1 and modify the timer to be 30 seconds. You can also set millisecond granularity if desired.

```
! CSR1
service timestamps debug datetime
service timestamps log datetime
track timer ip route 30
```

```
R1#show track timers | include ip_route
ip route          30          12.718
```

The network is fully operational which means the tracked object is up. When the timer restarts at 30 seconds, we will shut down the link to CSR2 on CSR1 immediately thereafter. We expect the tracked object to stay down for about 26-28 seconds or so, depending on how fast I am. The sequence of events is shown below with timestamps and key actions highlighted. The EIGRP neighbor fails immediately as expected, but the tracked object remains in the up state for some time. Notice that the object took 28 seconds to go down, which is what we expected.

```
R1#show track timers | include ip_route
ip route          30          29.887
```

```
! CSR1
interface GigabitEthernet2.512
shutdown
```

```
23:17:28: %DUAL-5-NBRCHANGE: EIGRP-IPv4 1: Neighbor 153.1.2.2
(GigabitEthernet2.512) is down: interface down
```

```
R1#show track timers | include ip_route
ip route          30          0.608
```

```
23:17:56: %TRACK-6-STATE: 25 ip route 153.2.2.2/32 metric threshold Up ->
```

Down

We can also tell the object itself to delay going up or down on a per-object basis. We configure this probe to delay for 10 additional seconds when coming up. We will repeat the same test except we will bring the interface back up, and expect 35-38 seconds of delay (~30 seconds from the track timer plus ~10 seconds from the up-delay). First, we configure the up-delay and verify that the 10 second delay was properly applied to the object.

```
! CSR1
track 25 ip route 153.2.2.2 255.255.255.255 metric threshold
  threshold metric up 55 down 95
  delay up 10

R1#show track 25
Track 25
  IP route 153.2.2.2 255.255.255.255 metric threshold
  Metric threshold is Up (EIGRP/130816/51)
    9 changes, last change 00:13:52
  Metric threshold down 95 up 55
  Delay up 10 secs
  First-hop interface is GigabitEthernet2.512
```

We follow the same test procedure as above except “no shutdown” the link between CSR1 and CSR2. In this case, it was about 36 seconds, since it took EIGRP a few seconds to discover CSR2 and converge. Notice that when the countdown timer hits 0.593 seconds, we still have to wait another 10 seconds because of the per-object delay-up time. In the last example, as soon as the global timer expired, the router re-evaluated the object and changed its state. This time, the router still performed the re-evaluation, directed the object to change state, but the object delayed the “up” action.

```
R1#show track timers | include ip_route
ip route          30          29.626

! CSR1
interface GigabitEthernet2.512
  no shutdown

23:21:00: %DUAL-5-NBRCHANGE: EIGRP-IPv4 1: Neighbor 153.1.2.2
(GigabitEthernet2.512) is up: new adjacency

! About 30 seconds later
R1#show track timers | include ip_route
ip route          30          0.593

! Another 10 seconds later
R1#show track timers | include ip_route
ip route          30          21.441
```



```
23:21:36: %TRACK-6-STATE: 25 ip route 153.2.2.2/32 metric threshold Down ->
Up
```

47.9 IPv6 SLA

A number of probes are supported for IPv6, but not all. Cisco documentation specifies that only ICMP-echo, UDP-echo, UDP-jitter, and TCP-connect are supported, but in reality, the path-echo and path-jitter are also supported (and possibly more). IPv6 SLA operation is identical to that of its IPv4 counterpart. I've configured OSPFv3 using IPv6 AF only (XR doesn't support OSPFv3 AF IPv4) for this demonstration. The configuration is basic and is not shown. The probe numbers used earlier will be repeated but added to 60. For example, IP SLA ID 3 from the IPv4 testing will be ID 63 here.

The first probe to examine is the ICMP-echo probe [ID 61]. The probe has been discussed in detail already so the basic operation is not revisited. Two new options exist under the probe; traffic-class and flow-label. Traffic-class is functionally equivalent to IP TOS and is still an unsigned 8 bit integer. We will use DSCP AF21 for this probe; DSCP AF21 in decimal is 18. Bit-wise left-shifting twice results in a traffic-class value of 72 or 0x48.

```
R1(config-ip-sla-echo)#traffic-class ?
<0-255> Traffic Class Value
```

The flow "label" has nothing to do with MPLS, despite being 20 bits (the exact same size as the label field in the MPLS shim header). This is a component of the IPv6 header with a few key uses. First, it allows routers to avoid having to look deeper into the packet (could be encrypted, etc) to identify the flow. Loosely analogous to the GRE tunnel key with "entropy" enabled, it could be unique per flow to allow for better load-sharing in the core. Packets in the same flow should have the same flow label and should be forwarded along the same path. Intermediate network devices will generally keep packets with the same flow-label in the same path if they have multiple paths to a destination (ECMP, UCMP, etc). In this case, we set a value of 19, but it doesn't do anything significant in the network. A value of zero means the packet has no flow specifications.

```
R1(config-ip-sla-echo)#flow-label ?
<0-1048575> Flow Label Value
```

All other probe configurations (timers, history, scheduling, object tracking) are the same. The probe configuration is shown below for completeness.

```
! CSR1
ip sla 61
 icmp-echo ::153:2:2:2 source-ip ::153:1:1:1
 traffic-class 72
 flow-label 19
 frequency 5
ip sla schedule 61 life forever start-time now
```

We see the probe succeeds without issue. We can also see that our QoS and flow-label settings were properly applied as CSR2's debug reveals those values. The traffic class is shown as a decimal number from 0 to 255 while the flow label is shown in hexadecimal.

```
R1#show ip sla statistics 61 | include Number
Number of successes: 4
Number of failures: 0
```

```
R2#debug ipv6 icmp
IPV6: source ::153:1:1:1 (GigabitEthernet2.512)
      dest ::153:2:2:2 (Loopback0)
      traffic class 72, flow 0x13, len 76+18, prot 58, hops 64, forward to ulp
```

Next, we test the TCP connect probe [ID 62]. Again, the threshold, timeout, and frequency settings work identically for IPv6 as they do for IPv4.

```
! CSR1
ip sla 62
  tcp-connect ::153:2:2:2 55555 source-ip ::153:1:1:1
  threshold 10000
  timeout 10000
  frequency 10
ip sla schedule 62 life forever start-time now
```

We also enable debugging on CSR2, which is a dynamic responder, to ensure the IPv6 TCP operation is successful. Note that "ip sla responder" applies to IPv6 probes without any additional configuration. On an XE responder, the operation 0 must be specified for the debug. Notice that the responder performs the same actions it did for IPv4 and also tracks the IPv6 addressing in its "recent source" list. CSR1 reports no failures.

```
R1#show ip sla statistics 62 | include Number
Number of successes: 5
Number of failures: 0
```

```
R2#debug ip sla trace 0
IPSLA-RESP_TRACE: Receive Control msg: len =68
IPSLA-RESP_TRACE: Ctrl-Msg Ver: 1 ID: 168 Len: 68
IPSLA-RESP_TRACE: Ctrl-Msg: command: SLA_CMD_TCPV6_CONN_ENABLE, ip:
::153:2:2:2, port: 55555, duration: 10000
[snip]
IPSLA-RESP_TRACE: %TRACE: TCP socket accepted
IPSLA-RESP_TRACE: Cleaning up port (::153:2:2:2,55555)
```

```
R2#show ip sla responder
  General IP SLA Responder on Control port 1967
  General IP SLA Responder on Control V2 port 1167
General IP SLA Responder is: Enabled
```

```
Number of control message received: 2 Number of errors: 0
Recent sources:
```

```
::153:1:1:1 [13:53:16.508 UTC DAY MON 11 2015]
::153:1:1:1 [13:53:06.508 UTC DAY MON 11 2015]
```

We also test using a static responder. We configure a probe on CSR2 to send UDP-echoes to CSR1 without any control messaging [ID 63]. This means we need a static responder configured on CSR1 for the target IPv6 address and port pair. The command syntax is the same as it is for IPv4; this is somewhat odd since there is no “ipv6 sla” tree, yet the IPv6 functionality is nested within the “ip sla” tree.

```
! CSR1
ip sla responder udp-echo ipaddress ::153:1:1:1 port 55556
```

```
R1#show ip sla responder | section udp
udpEcho Responder:
```

IP Address	Port
153.1.1.1	55556
153.1.1.1	55557
::153:1:1:1	55556

Next, we configure the probe on CSR2. As expected, no failures are observed since CSR1 is responding properly to these probes.

```
! CSR2
ip sla 63
  udp-echo ::153:1:1:1 55556 control disable
  frequency 5
ip sla schedule 63 life forever start-time now
```

```
R2#show ip sla statistics 63 | include Number
Number of successes: 3
Number of failures: 0
```

Recall that UDP-jitter was a complicated operation because the information it is capable of resolving is dependent upon a few things. First, NTP synchronization determines whether the probe resolves one-way latency or not. Second, a dynamic responder is required for jitter/MOS resolution. We quickly verify these behaviors with IPv6 SLA. We create a UDP-jitter probe on CSR2 with control-messages disabled to probe CSR1 [ID 64]. With NTP configured, we expect to resolve one-way latency only. For consistency, we will use the same parameters as IP SLA ID 4, to include the port number, which means we need to quickly add another static responder entry to CSR1 (not shown).

```
! CSR2
ip sla 64
  udp-jitter ::153:1:1:1 55557 control disable num-packets 200 interval 25
  request-data-size 72
  frequency 10
```

```
ip sla schedule 64 life forever start-time now
```

```
R2#show ntp status | include ^sync
```

```
Clock is synchronized, stratum 9, reference is 153.1.1.1
```

```
R2#show ip sla statistics 64 | section Latency|Jitter|Number
```

```
Number Of RTT: 200 RTT Min/Avg/Max: 1/1/2 milliseconds
```

```
Latency one-way time:
```

```
Number of Latency one-way Samples: 1
```

```
Source to Destination Latency one way Min/Avg/Max: 9/9/9 milliseconds
```

```
Destination to Source Latency one way Min/Avg/Max: 3/3/3 milliseconds
```

```
Jitter Time:
```

```
Number of SD Jitter Samples: 0
```

```
Number of DS Jitter Samples: 0
```

```
Source to Destination Jitter Min/Avg/Max: 0/0/0 milliseconds
```

```
Destination to Source Jitter Min/Avg/Max: 0/0/0 milliseconds
```

```
Number Of RTT Over Threshold: 0 (0%)
```

```
Source to Destination Loss Periods Number: 0
```

```
Destination to Source Loss Periods Number: 0
```

```
Number of successes: 7
```

```
Number of failures: 0
```

Trying a similar probe in the opposite direction from CSR1 to CSR2 [ID 65], we get statistics for jitter and latency. This is because NTP is synchronized and CSR2 is already a dynamic IP SLA responder.

```
! CSR1
```

```
ip sla 65
```

```
udp-jitter ::153:2:2:2 55558 num-packets 200 interval 25
```

```
request-data-size 72
```

```
frequency 10
```

Note that if the latency is very low, the router may just display all zeroes, even for the number of samples. To increase latency a little, I shut down the direct link between CSR1 and CSR2. The reason we know that this was not occurring for IP SLA ID 64 is because the number of jitter samples was still zero.

```
R1#show ip sla statistics 65 | section Jitter|Latency|Number
```

```
Number Of RTT: 200 RTT Min/Avg/Max: 3/4/66 milliseconds
```

```
Latency one-way time:
```

```
Number of Latency one-way Samples: 16
```

```
Source to Destination Latency one way Min/Avg/Max: 4/11/65 milliseconds
```

```
Destination to Source Latency one way Min/Avg/Max: 0/0/2 milliseconds
```

```
Jitter Time:
```

```
Number of SD Jitter Samples: 199
```

```
Number of DS Jitter Samples: 199
```

```
Source to Destination Jitter Min/Avg/Max: 0/2/61 milliseconds
```

```
Destination to Source Jitter Min/Avg/Max: 0/1/3 milliseconds
```

```
Number Of RTT Over Threshold: 0 (0%)
```

```
Source to Destination Loss Periods Number: 0
```

```
Destination to Source Loss Periods Number: 0
Number of successes: 33
Number of failures: 0
```

Advanced ICMP probes, such as path-echo and path-jitter, also appear supported. The path-echo probe configured on CSR1 to test IPv6 [ID 70] mirrors the configuration of its IPv4 counterpart.

```
! CSR1
ip sla 70
  path-echo ::153:2:2:2
  threshold 50
  timeout 100
  frequency 10
  hops-of-statistics-kept 3
  samples-of-history-kept 10
ip sla schedule 70 life forever start-time now
```

Remember that in order to see valuable data from this probe, we need to look at the aggregated statistics. This particular probe discovers each of the hops in each of the paths (2 dimensional array, or matrix of values) and displays them within the aggregated statistics.

```
R1#show ip sla statistics aggregated 70
IPSLAs aggregated statistics

IPSLA operation id: 70
Start Time Index: 14:25:41.428 UTC [snip]
Path Index: 1
Hop in Path Index: 1
Type of operation: path-echo
Number of successes: 2
Number of failures: 0
Target Address ::153:13:13:13

Start Time Index: 14:25:41.428 UTC [snip]
Path Index: 1
Hop in Path Index: 2
Type of operation: path-echo
Number of successes: 2
Number of failures: 0
Target Address ::153:14:14:14

Start Time Index: 14:25:41.428 UTC [snip]
Path Index: 1
Hop in Path Index: 3
Type of operation: path-echo
Number of successes: 2
Number of failures: 0
Target Address ::153:2:2:2
```

Last, we examine path-jitter for IPv6. Recall that this probe discovers all hops as returned by “traceroute”, which runs first. Then, individual prober operations are spawned to measure metrics for each hop. As such, we must have reachability to the transit links (unless IPv6 LL addressing is used), so OSPFv3 prefix-suppression should not be enabled. In this case, the loopback on each device is the only routable IPv6 address, and as such, is used as the source address for all ICMP unreachables returned to CSR1. If the transit links had any other addressing, the probing device would need reachability to whatever “traceroute” returns. The path-jitter probe configured on CSR1 [ID 71] will trace the path to CSR2, and the direct link to CSR2 is still down. We can see the output is very similar to IPv4.

```
! CSR1
ip sla 71
  path-jitter ::153:2:2:2 source-ip ::153:1:1:1
  frequency 5
ip sla schedule 71 life forever start-time now

R1#show ip sla statistics 71
IPSLAs Latest Operation Statistics
IPSLA operation id: 71
    Latest RTT: 4 milliseconds
Latest operation start time: 14:21:58 UTC [snip]
Latest operation return code: OK
```

---- Path Jitter Statistics ----

```
Hop IP ::153:13:13:13:
Round Trip Time milliseconds:
    Latest RTT: 1 ms
    Number of RTT: 10
    RTT Min/Avg/Max: 1/1/2 ms
Jitter time milliseconds:
    Number of jitter: 5
    Jitter Min/Avg/Max: 1/1/1 ms
Packet Values:
    Packet Loss (Timeouts): 0
    Out of Sequence: 0
    Discarded Samples: 0
```

```
Hop IP ::153:14:14:14:
Round Trip Time milliseconds:
    Latest RTT: 2 ms
    Number of RTT: 10
    RTT Min/Avg/Max: 2/2/3 ms
[snip]
```

```
Hop IP ::153:2:2:2:
Round Trip Time milliseconds:
```

```
Latest RTT: 4 ms
Number of RTT: 10
RTT Min/Avg/Max: 3/4/7 ms
[snip]
```

It is likely that the DNS, HTTP, and FTP probes work for IPv6 also, but they are not tested here for brevity. The MPLS ones only work for IPv4 FECs current. DHCP does not appear to support IPv6, which makes sense since the DHCPv4 and DHCPv6 operations are fundamentally different (address vs. prefix).

47.10 IOS-XR IP SLA and EOT

IOS XR IP SLA is not very different from IOS and XE. A few quick notes:

1. TCP-connect and fancy operations (HTTP, DNS, HTTP, DHCP, etc) are not supported.
2. ICMP operations supported include echo, path-echo, and path-jitter. No ICMP-jitter.
3. UDP operations supported include echo and jitter. VOIP-style codec probes are not supported.
4. MPLS operations supported include ping and traceroute for both IPv4 and TE targets.
5. In general, XR is less capable than XE with regards to IP SLA.

Because TCP operations are not supported, all static IP SLA responders only give the option for UDP, and only IPv4 responders appear to be supported also. We will quick iterate over the list of probes already discussed. XRv4 is configured as a dynamic IP SLA responder. The IDs of each re-tested probe will have an ID of 300 plus whatever number the original probe was on CSR1 or CSR2.

```
! XRv4
ipsla
  responder
```

The first test is the basic ICMP-echo probe. We will configure it from XRv3 to XRv4 [ID 301].

```
! XRv3
ipsla
  operation 301
  type icmp echo
  source address 153.13.13.13
  destination address 153.14.14.14
  frequency 5

  schedule operation 301
  start-time now
  life forever
```

We can see the probe is successful and it provides a detailed RTT breakdown also, unlike IOS and XE. Quick tip: You can un-schedule all probes at once with “no ipsla schedule”.

```
RP/0/0/CPU0:XRv3#show ipsla statistics 301
Entry number: 301
```

```

Modification time: 19:57:04.964 UTC [date]
Start time       : 19:57:04.984 UTC [date]
Number of operations attempted: 10
Number of operations skipped  : 0
Current seconds left in Life  : Forever
Operational state of entry    : Active
Operational frequency(seconds): 5
Connection loss occurred      : FALSE
Timeout occurred              : FALSE
Latest RTT (milliseconds)     : 1
Latest operation start time    : 19:57:45.222 UTC [date]
Next operation start time     : 19:57:50.222 UTC [date]
Latest operation return code   : OK
RTT Values:
    RTTAvg   : 1           RTTMin: 1           RTTMax : 1
    NumOfRTT : 1           RTTSum: 1           RTTSum2: 1

```

IOS XR has a different way of indicating failures. The output below clearly shows the number of attempts, but then has some Boolean variables representing whether the operation timed out or connection was lost. Since ICMP is connectionless, this field doesn't apply, but once XRv4's loopback is shutdown, the IP SLA process marks the timeout flag as TRUE and increments the "skipped" counter. This isn't a great description because the operation wasn't skipped; it did run, but it failed. The XE mechanism of reporting successes/failures was more straightforward.

```

RP/0/0/CPU0:XRv3#show ipsla statistics 301
Entry number: 301
[snip]
Number of operations attempted: 10
Number of operations skipped  : 1
Current seconds left in Life  : Forever
Operational state of entry    : Active
Operational frequency(seconds): 5
Connection loss occurred      : FALSE
Timeout occurred              : TRUE
[snip]

```

Next, we will configure a path-echo probe in the same direction [ID 310]. I have shutdown XRv3's link to XRv4 to force traffic through CSR1 and CSR2.

```

! XRv3
ipsla
 operation 310
  type icmp path-echo
  source address 153.13.13.13
  destination address 153.14.14.14
  frequency 10

 schedule operation 301

```



```
start-time now
life forever
```

Traceroute quickly reveals the effect of this minor routing change.

```
RP/0/0/CPU0:XRv3#traceroute 153.14.14.14 source 153.13.13.13
```

```
Type escape sequence to abort.
Tracing the route to 153.14.14.14
```

```
 1  153.1.13.1 [MPLS: Label 1004 Exp 0] 0 msec  0 msec  0 msec
 2  153.1.2.2 [MPLS: Label 2004 Exp 0] 0 msec  0 msec  0 msec
 3  153.2.14.14 0 msec  *  0 msec
```

Next, I disable ICMP unreachable on XRv4's interface facing CSR2, causing traceroute to fail.

```
! XRv4
interface GigabitEthernet0/0/0/0.524
  ipv4 unreachable disable
```

```
RP/0/0/CPU0:XRv3#traceroute 153.14.14.14 source 153.13.13.13
```

```
Type escape sequence to abort.
Tracing the route to 153.14.14.14
```

```
 1  153.1.13.1 [MPLS: Label 1004 Exp 0] 0 msec  0 msec  0 msec
 2  153.1.2.2 [MPLS: Label 2004 Exp 0] 0 msec  0 msec  0 msec
 3  * * *
[snip]
```

Unfortunately, we can't see much in XR because the debugs are not helpful; they are all marked with "cisco-support", so Cisco did not intend for the general user to rely on them. The good news is that the show command actually does report failures and timeouts at least for this probe type, unlike IOS and XE.

```
RP/0/0/CPU0:XRv3#debug ipsla ?
```

```
common      Debug the IPSLA common modules(cisco-support)
error        Output IP SLAs Error Messages(cisco-support)
ippmserver   Debug the IPSLA IPPM Server process(cisco-support)
master-agent Debug the IPSLA master agent process(cisco-support)
responder    Debug the IPSLA responder process(cisco-support)
sub-agent    Debug the IPSLA sub agent process(cisco-support)
trace        Output IP SLAs Trace Messages(cisco-support)
```

```
RP/0/0/CPU0:XRv3#show ipsla statistics 310
```

```
Entry number: 310
```

```
[snip]
```

```
Number of operations attempted: 8
```

```
Number of operations skipped : 24
```

```

Current seconds left in Life : Forever
Operational state of entry   : Active
Operational frequency(seconds): 10
Connection loss occurred     : FALSE
Timeout occurred             : TRUE
[snip]

```

Once we rollback the change on XRv4 to allow it to send unreachable again (not shown), we see the timeout value change to false and the “attempted” operations begins to increase again.

```

RP/0/0/CPU0:XRv3#show ipsla statistics 310
Entry number: 310
[snip]
Number of operations attempted: 12
Number of operations skipped   : 32
Current seconds left in Life   : Forever
Operational state of entry     : Active
Operational frequency(seconds): 10
Connection loss occurred       : FALSE
Timeout occurred               : FALSE
[snip]

```

Next is the path-jitter which is configured on XRv3 [ID 312]. This operation issues a traceroute first, follow by ICMP-echo probes to each hop returned within the traceroute list. We still have the same variables from before: N, S, T, and F. I’ve included their default values where known.

```

RP/0/0/CPU0:XRv3(config-ipsla-op)#type icmp path-jitter ?
count      Number of packets to be transmitted during a probe ! (N, 10 pkts)
interval    Inter packet interval ! (T, 20 msec)

```

```

RP/0/0/CPU0:XRv3(config-ipsla-op)#type icmp path-jitter ?
datasize    Protocol data size in payload of probe packets ! (S)
frequency    Frequency of the probing ! (F, 60 sec)

```

```

! XRv3
ipsla
operation 312
type icmp path-jitter
source address 153.13.13.13
destination address 153.14.14.14
frequency 10

schedule operation 312
start-time now
life forever

```

We start the probe and then check the statistics. We expect to see three hops in the path (CSR1, CSR2,

and XRv4). The output from XR is very extensive but provides similar information as IOS and XE. I have omitted the statistical output from subsequent hops but it is present for the first hop. There does not appear to be way to specify "targetOnly" as we did in IOS and XE, but this is a minor feature anyway.

```
RP/0/0/CPU0:XRv3#show ipsla statistics 312
```

```
Entry number: 312
```

```
Modification time: 20:22:23.390 UTC [date]
Start time       : 20:22:23.410 UTC [date]
Number of operations attempted: 1
Number of operations skipped : 0
Current seconds left in Life : Forever
Operational state of entry   : Active
Operational frequency(seconds): 10
Connection loss occurred    : FALSE
Timeout occurred           : FALSE
Latest RTT (milliseconds)   : 8
Latest operation start time  : 20:22:26.650 UTC [date]
Next operation start time    : 20:22:36.650 UTC [date]
Latest operation return code : OK
```

```
---- ICMP Path Jitter Statistics ----
```

```
Source IP          - 153.13.13.13
Destination IP     - 153.14.14.14
Number of Echos    - 10
Interval between Echos - 20 ms
```

```
Hop Index: 0
```

```
Hop IP 153.1.13.1:
```

```
RTT Values:
```

```
RTTAvg   : 1          RTTMin: 1          RTTMax : 1
NumOfRTT: 10          RTTSum: 10         RTTSum2: 10
```

```
Packet Loss Values:
```

```
PacketCount : 10          PacketLoss      : 0
OutOfSequence: 0          DiscardedSamples: 0
VerifyErrors : 0          DroppedErrors   : 0
```

```
Jitter Values :
```

```
Jitter      : 0          PosCount      : 0          NegCount:0
JitterAve   : 0
MinPosJitter: 0          MaxPosJitter: 0          SumPos:0
Sum2Pos     : 0          AvePosJitter: 0
MinNegJitter: 0          MaxNegJitter: 0          SumNeg: 0
Sum2Neg     : 0          AveNegJitter: 0
```

```
Hop Index: 1
```

```
Hop IP 153.1.2.2:
```

```
[snip]
```

```
Hop Index: 2
```

```
Hop IP 153.14.14.14:
```

[snip]

Next, we will configure a UDP-echo on XRv4 to probe XRv3's loopback [ID 303]. We will make XRv3 a static responder for UDP 55556 and the IP address of its loopback (which is the address XRv4 is targeting). Control-messages should be disabled on XRv4 as the UDP data should be sent immediately.

```
! XRv3
ipsla
  responder
    type udp ipv4 address 153.13.13.13 port 55556

! XRv4
ipsla
  operation 303
    type udp echo
    source address 153.14.14.14
    destination address 153.13.13.13
    control disable
    destination port 55556

schedule operation 303
  start-time now
  life forever
```

XRv4 reports success and XRv3, as a static responder, also counts the received probes on that specific port/address pair. Notice there are no control probes on a static (aka permanent) responder.

```
RP/0/0/CPU0:XRv4#show ipsla statistics 303
```

```
Entry number: 303
```

[snip]

```
Number of operations attempted: 2
Number of operations skipped   : 0
Current seconds left in Life   : Forever
Operational state of entry     : Active
Operational frequency(seconds): 60
Connection loss occurred      : FALSE
Timeout occurred               : FALSE
Latest RTT (milliseconds)     : 1
[snip]
```

```
RP/0/0/CPU0:XRv3#show ipsla responder statistics all ports
```

```
Port Statistics
```

Local Address	Port	Port Type	Probes	Drops	CtrlProbes	Discard
153.13.13.13	55556	Permanent	2	0	0	

Next we will examine the UDP-jitter probe on XR. This was an interesting probe with many contributing

factors beyond the scope of the probe itself. We will configure a probe on XRv4 [ID 304] to probe XRv3's loopback for latency/jitter information. Notice we use the same UDP destination port of 55556 which already has a corresponding static responder configured on XRv3 from the last example.

```
! XRv4
ipsla
  operation 304
    type udp jitter
    source address 153.14.14.14
    destination address 153.13.13.13
    control disable
    destination port 55556
    frequency 10

  schedule operation 304
    start-time now
    life forever
```

The two routers are not synchronized with NTP and the responder is static. As expected, we cannot resolve jitter or one-way delay (latency). So far, this is consistent with IOS and XE behaviors.

```
RP/0/0/CPU0:XRv4#show ipsla statistics 304
Entry number: 304
  [snip]
  Number of operations attempted: 5
  Number of operations skipped   : 0
  Current seconds left in Life   : Forever
  Operational state of entry     : Active
  Operational frequency(seconds): 10
  Connection loss occurred       : FALSE
  Timeout occurred               : FALSE
  [snip]
  Latest operation return code   : OK
  RTT Values:
    RTTAvg   : 1           RTTMin: 1           RTTMax : 1
    NumOfRTT: 10          RTTSum: 10          RTTSum2: 10
  Packet Loss Values:
    [snip]
  Jitter Values :
    MinOfPositivesSD: 0           MaxOfPositivesSD: 0
    NumOfPositivesSD: 0           SumOfPositivesSD: 0
    Sum2PositivesSD : 0
    MinOfNegativesSD: 0           MaxOfNegativesSD: 0
    [snip]
  One Way Values :
    NumOfOW: 0
    OWMinSD : 0           OWMaxSD: 0           OWSumSD: 0
    OWSum2SD: 0           OWAveSD: 0
```

```
OWMinDS : 0          OWMaxDS: 0          OWSumDS: 0
OWSum2DS: 0         OWAveDS: 0
```

Configuring another UDP jitter probe on XRv3 to probe XRv4 will use XRv4's dynamic responder and should give us jitter, but not latency [ID 305].

```
! XRv3
ipsla
  operation 305
    type udp jitter
    source address 153.13.13.13
    destination address 153.14.14.14
    destination port 55558

schedule operation 305
  start-time now
  life forever
```

As expected, IP SLA ID 305 can resolve jitter only as a result of XRv4 being a dynamic responder. The lack of NTP synchronization implies that one-way latency cannot be collected.

```
RP/0/0/CPU0:XRv3#show ipsla statistics 305
Entry number: 305
[snip]
Number of operations attempted: 2
Number of operations skipped : 0
Current seconds left in Life : Forever
Operational state of entry : Active
Operational frequency(seconds): 60
Connection loss occurred : FALSE
Timeout occurred : FALSE
[snip]
Latest operation return code : OK
[snip]
Jitter Values :
  MinOfPositivesSD: 40          MaxOfPositivesSD: 40
  NumOfPositivesSD: 1          SumOfPositivesSD: 40
  Sum2PositivesSD : 1600
[snip]
JitterAve: 40          JitterSDAve: 40          JitterDSAve: 0
Interarrival jitterout: 0          Interarrival jitterin: 0
One Way Values :
  NumOfOW: 0
  OWMinSD : 0          OWMaxSD: 0          OWSumSD: 0
  OWSum2SD: 0          OWAveSD: 0
  OWMinDS : 0          OWMaxDS: 0          OWSumDS: 0
  OWSum2DS: 0          OWAveDS: 0
```

Notice how the responder on XRv4 records this as a dynamically-learned probe and tracks its progress. The responder also shows that control probes were received, which makes sense for a dynamic responder.

```
RP/0/0/CPU0:XRv4#show ipsla responder statistics all ports
Port Statistics
-----
Local Address   Port  Port Type  Probes    Drops    CtrlProbes Discard
153.14.14.14    55558 Dynamic    20         0         2         ON
```

Let's quickly enable NTP on the XRv routers, using CSR1 as the server because it's already configured for it. We wait several minutes for NTP to synchronize.

```
! XRv3 and XRv4
ntp
server 153.1.1.1 source Loopback0
```

```
RP/0/0/CPU0:XRv3#show ntp status | include sync
Clock is synchronized, stratum 9, reference is 153.1.1.1
```

```
RP/0/0/CPU0:XRv4#show ntp status | include sync
Clock is synchronized, stratum 9, reference is 153.1.1.1
```

IP SLA ID 305 on XRv3 should start reporting both latency and jitter since it is targeting a dynamic responder. IP SLA ID 304 on XRv4 should start reporting latency only since it is targeting a static responder. The only change we made in the network was synchronizing time; this proves that the XR behavior is identical to IOS and XE with respect to latency and jitter resolution/collection.

```
RP/0/0/CPU0:XRv3# show ipsla statistics 305
Entry number: 305
[snip]
Jitter Values :
  MinOfPositivesSD: 10      MaxOfPositivesSD: 10
  NumOfPositivesSD: 1       SumOfPositivesSD: 10
  Sum2PositivesSD : 100
  MinOfNegativesSD: 10     MaxOfNegativesSD: 10
  NumOfNegativesSD: 1       SumOfNegativesSD: 10
  Sum2NegativesSD : 100
  MinOfPositivesDS: 20     MaxOfPositivesDS: 20
  NumOfPositivesDS: 1       SumOfPositivesDS: 20
  Sum2PositivesDS : 400
  MinOfNegativesDS: 20     MaxOfNegativesDS: 20
  NumOfNegativesDS: 1       SumOfNegativesDS: 20
  Sum2NegativesDS : 400
  JitterAve: 15           JitterSDAve: 10      JitterDSAve: 20
  Interarrival jitterout: 0      Interarrival jitterin: 0
One Way Values :
```

```

NumOfOW: 2
OWMinSD : 10      OWMaxSD: 10      OWSumSD: 20
OWSum2SD: 200     OWAVESD: 10
OWMinDS : 0       OWMaxDS: 20      OWSumDS: 20
OWSum2DS: 400     OWAveDS: 10

```

```
RP/0/0/CPU0:XRv4#show ipsla statistics 304
```

```
[snip]
```

```
Jitter Values :
```

```

MinOfPositivesSD: 0      MaxOfPositivesSD: 0
NumOfPositivesSD: 0      SumOfPositivesSD: 0
Sum2PositivesSD : 0

```

```
[snip]
```

```
One Way Values :
```

```

NumOfOW: 1
OWMinSD : 19      OWMaxSD: 19      OWSumSD: 19
OWSum2SD: 361     OWAVESD: 19
OWMinDS : 1       OWMaxDS: 1      OWSumDS: 1
OWSum2DS: 1       OWAveDS: 1

```

Next, we will examine the MPLS ping and traceroute operations. Remember that “mpls oam” must be explicitly enabled in order to use these commands; this is not enabled by default in XR. We will configure an MPLS LSP ping [ID 313] and LSP traceroute [ID 314] on XRv3 targeting XRv4’s loopback address. Before configuring the probes, we will manually test MPLS ping/traceroute from the CLI. Everything looks correct so far.

```
RP/0/0/CPU0:XRv3#ping mpls ipv4 153.14.14.14/32 source 153.13.13.13
```

```
[snip]
```

```
!!!!!
```

```
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/10 ms
```

```
RP/0/0/CPU0:XRv3#trace mpls ipv4 153.14.14.14/32 source 153.13.13.13
```

```
[snip]
```

```
Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
       'L' - labeled output interface, 'B' - unlabeled output interface,
```

```
[snip]
```

```
Type escape sequence to abort.
```

```

0 153.1.13.13 MRU 1500 [Labels: 1004 Exp: 0]
L 1 153.1.13.1 MRU 1500 [Labels: 2004 Exp: 0] 0 ms
L 2 153.1.2.2 MRU 1500 [Labels: implicit-null Exp: 0] 10 ms
! 3 153.2.14.14 1 ms

```

Next, we configure the probes. It is interesting to note that sometimes exp-null is required to get XR to respond to MPLS OAM traceroute probes. This is true when XR is the target of an LSP traceroute, but isn’t terribly significant since XR IP SLA supports it. Notice that the configuration between the LSP ping and traceroute operations is identical since the features share many common parameters and

behaviors.

```
! XRv3
ipsla
  operation 313
    type mpls lsp ping
    source address 153.13.13.13
    target ipv4 153.14.14.14/32
    frequency 10
    force explicit-null

  operation 314
    type mpls lsp trace
    source address 153.13.13.13
    target ipv4 153.14.14.14/32
    frequency 10
    force explicit-null

  schedule operation 313
    start-time now
    life forever

  schedule operation 314
    start-time now
    life forever
```

Checking the probe statistics, everything appears to be working. This makes sense since we manually tested the LSP ping/traceroute operations before configuring the probes.

```
RP/0/0/CPU0:XRv3#show ipsla statistics 313 | utility egrep
'Number|Timeout|loss'
  Number of operations attempted: 6
  Number of operations skipped   : 0
  Connection loss occurred       : FALSE
  Timeout occurred               : FALSE

RP/0/0/CPU0:XRv3#show ipsla statistics 314 | utility egrep
'Number|Timeout|loss'
  Number of operations attempted: 4
  Number of operations skipped   : 0
  Connection loss occurred       : FALSE
  Timeout occurred               : FALSE
```

We will temporarily break the LSP by telling CSR2 to stop advertising LDP labels completely. We will rerun our manual verifications and expect the probes to report failures also. XRv did not increase the “skipped” counter for the MPLS ping failures, but instead said connection loss occurred. The MPLS traceroute doesn’t change the timeout/connection flags nor does it increment the “skipped” counter; it only reports the RTT as unknown, so you can use that as a measurement. This can be difficult to identify

when looking at the outputs quickly.

```
! CSR2
no mpls ldp advertise-labels

RP/0/0/CPU0:XRv3#ping mpls ipv4 153.14.14.14/32 source 153.13.13.13
[snip]
Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
       'L' - labeled output interface, 'B' - unlabeled output interface,
[snip]
BBBBB
Success rate is 0 percent (0/5)

RP/0/0/CPU0:XRv3#traceroute mpls ipv4 153.14.14.14/32 source 153.13.13.13
[snip]
Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
       'L' - labeled output interface, 'B' - unlabeled output interface,
[snip]

  0 153.1.13.13 MRU 1500 [Labels: 1004 Exp: 0]
B 1 153.1.13.1 MRU 1500 [No Label] 0 ms

RP/0/0/CPU0:XRv3#show ipsla statistics 313 | utility egrep
'Number|Timeout|loss'
    Number of operations attempted: 50
    Number of operations skipped   : 0
    Connection loss occurred       : TRUE
    Timeout occurred               : FALSE

RP/0/0/CPU0:XRv3#show ipsla statistics 314 | utility egrep
'Number|Timeout|loss|Latest RTT'
    Number of operations attempted: 55
    Number of operations skipped   : 0
    Connection loss occurred       : FALSE
    Timeout occurred               : FALSE
    Latest RTT (milliseconds)     : Unknown
```

To see some extra valuable information on the MPLS traceroute, be sure to use the aggregated statistics show command. Unfortunately, unlike XE, this does not show the LSP hop-by-hop. The detail modifier prints the output in a compact tabular format shown below.

```
RP/0/0/CPU0:XRv3#show ipsla statistics aggregated detail 314 | begin ^Entry S
Entry StartT      Pth Hop Dst Comps   SumCmp  SumCmp2H  SumCmp2L  TMax  TMin
314   1436637790389 0   0  0   22    22      0      22      1    1
314   1436638019453 0   0  0   17    17      0      17      1    1
```

The MPLS ping and traceroute features can also test a TE tunnel. Be careful if you are using XRv 5.3.0 with verbatim TE; bug CSCus55972 states that the next hop in the ERO path must be a /32, so XRv3 has a static route to 153.1.13.1/32. The connected /24 is not good enough for TE tunnel construction as a result. Using a static route is Cisco's suggested workaround. This is totally unrelated to IP SLA but I mention it here to explain why the static route is configured. The only reason we use verbatim path-options is to build TE tunnels over EIGRP as the IGP as it is incapable of carrying TE topology information.

```
! XRv3
explicit-path name R1_R2_XRv4
  index 1 next-address strict ipv4 unicast 153.1.13.1
  index 2 next-address strict ipv4 unicast 153.1.2.2
  index 3 next-address strict ipv4 unicast 153.2.14.14
  index 4 next-address strict ipv4 unicast 153.14.14.14

interface tunnel-te34
  ipv4 unnumbered Loopback0
  destination 153.14.14.14
  path-option 10 explicit name R1_R2_XRv4 verbatim

router static
  address-family ipv4 unicast
    153.1.13.1/32 GigabitEthernet0/0/0/0.513
```

Let's ensure the tunnel is up and do a manual MPLS ping/traceroute to ensure MPLS data plane is functional, too.

```
RP/0/0/CPU0:XRv3#show mpls traffic-eng tunnels brief
          TUNNEL NAME          DESTINATION    STATUS  STATE
          tunnel-te34          153.14.14.14    up      up
Displayed 1 (of 1) heads, 0 (of 0) midpoints, 0 (of 0) tails
Displayed 1 up, 0 down, 0 recovering, 0 recovered heads

RP/0/0/CPU0:XRv3#ping mpls traffic-eng tunnel-te 34
[snip]
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/10 ms

RP/0/0/CPU0:XRv3#traceroute mpls traffic-eng tunnel-te 34
[snip]

  0 153.1.13.13 MRU 1500 [Labels: 1005 Exp: 0]
  L 1 153.1.13.1 MRU 1500 [Labels: 2005 Exp: 0] 10 ms
  L 2 153.1.2.2 MRU 1500 [Labels: implicit-null Exp: 0] 0 ms
  ! 3 153.2.14.14 1 ms
```

We will configure a ping probe [ID 370] and a traceroute probe [ID 371] on XRv3 to test the state of this tunnel at regular intervals (every 10 seconds).

```

! XRv3
ipsla
  operation 370
    type mpls lsp ping
    target traffic-eng tunnel 34
    frequency 10

  operation 371
    type mpls lsp trace
    target traffic-eng tunnel 34
    frequency 10

schedule operation 370
  start-time now
  life forever

schedule operation 371
  start-time now
  life forever

```

Below are the outputs from probes; both are successful.

```
RP/0/0/CPU0:XRv3#show ipsla statistics 370 | utility egrep
```

```

'Number|loss|return code|Latest RTT'
  Number of operations attempted: 2
  Number of operations skipped   : 0
  Connection loss occurred       : FALSE
  Latest RTT (milliseconds)      : 1
  Latest operation return code   : OK

```

```
RP/0/0/CPU0:XRv3#show ipsla statistics 371 | utility egrep
```

```

'Number|loss|return code'
  Number of operations attempted: 3
  Number of operations skipped   : 0
  Connection loss occurred       : FALSE
  Latest operation return code   : OK

```

To simulate a failure, we will disable MPLS TE tunnels globally on CSR2. This rips out the RSVP-based LSP and sends a PATHERR message back to the headend. We would expect the tunnel to be dysfunctional now. In fact, the MPLS OAM process doesn't even send the pings because the tunnel is down.

```

! CSR2
no mpls traffic-eng tunnels

```

```
RP/0/0/CPU0:XRv3#show mpls traffic-eng tunnels brief
```

TUNNEL NAME	DESTINATION	STATUS	STATE

```

tunnel-te34          153.14.14.14      down  down
Displayed 1 (of 1) heads, 0 (of 0) midpoints, 0 (of 0) tails
Displayed 0 up, 1 down, 0 recovering, 0 recovered heads

RP/0/0/CPU0:XRv3#ping mpls traffic-eng tunnel-te 34
[snip]
Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
QQQQQ
Success rate is 0 percent (0/5)

RP/0/0/CPU0:XRv3#traceroute mpls traffic-eng tunnel-te 34
[snip]

Q 1 *
```

Below are the failed probes. These outputs are more consistent than the IPv4 MPLS ping/traceroute manual tests shown above. They both continue to increment the “attempted” counter, but both indicate connection loss and tell us there was a TX error along the LSP. The output above indicates the requests were not even sent because the LSP is down entirely. This output is very clear and concise, even more so than the XE equivalent.

```

RP/0/0/CPU0:XRv3#show ipsla statistics 370 | utility egrep
'Number|loss|return code'
  Number of operations attempted: 9
  Number of operations skipped   : 0
  Connection loss occurred       : TRUE
  Latest operation return code   : MplsLspEchoTxError

RP/0/0/CPU0:XRv3#show ipsla statistics 371 | utility egrep
'Number|loss|return_code'
  Number of operations attempted: 14
  Number of operations skipped   : 0
  Connection loss occurred       : TRUE
  Latest operation return code   : MplsLspEchoTxError
```

XR simplifies the configuration logic for aggregation statics and history. There are two different syntax trees now: history and statistics. We will reuse IP SLA ID 301 to test these features and attempt to mirror the settings from IP SLA ID 1 on CSR1. The original goal was to collect aggregate statistics for RTT in “buckets” of width 25 ms between 0 and 100 (so we need 4 buckets). We also want to collect history for all events (not just failures) in a rotating list with 15 buckets. XR does not give us an option on how often to sample for history, probably because it writes history in step with the probe frequency (which makes sense). The configuration is shown below. Be sure to modify the “statistics hourly” category since the statistics “interval” is measured only in seconds. The hourly aggregated statistics is where we achieve the granularity we expect. The “statistics interval” is used for collecting enhanced statistics which is shown later.

```

! XRv3
ipsla
operation 301
type icmp echo
  history
  lives 1
  filter all
  buckets 15

source address 153.13.13.13
destination address 153.14.14.14
statistics hourly
  buckets 1
  distribution count 4
  distribution interval 25

frequency 5
statistics interval 5
  buckets 4

```

The output doesn't show us the bucket breakdowns like XE did, but we know each row is a bucket of width 25ms. All of our probes are landing in the first bucket since the network is small and RTT is low.

```

RP/0/0/CPU0:XRv3#show ipsla statistics aggregated detail 301 | begin Entry S
Entry StartT      Pth Hop Dst Comps  SumCmp  SumCmp2H  SumCmp2L  TMax  TMin
301  1436638963658 1   1   0   10     10      0       10      1     1
301  1436638963658 1   1   1   0      0      0       0       0     0
301  1436638963658 1   1   2   0      0      0       0       0     0
301  1436638963658 1   1   3   0      0      0       0       0     0

```

We can also view enhanced statistics which is based on the interval we configured earlier. Every 5 seconds, the router computes these statistics for display. This shows enhanced failure statistics over time.

```

RP/0/0/CPU0:XRv3#show ipsla statistics enhanced aggregated 301
Entry number: 301
Interval : 5 seconds
Bucket : 1
  Start Time Index: 18:27:24.469 UTC [snip]
  Number of Failed Operations due to a Disconnect      : 0
  Number of Failed Operations due to a Timeout         : 0
  Number of Failed Operations due to a Busy            : 0
  Number of Failed Operations due to a No Connection   : 0
  Number of Failed Operations due to an Internal Error : 0
  Number of Failed Operations due to a Sequence Error  : 0
  Number of Failed Operations due to a Verify Error    : 0
  RTT Values:
    RTTAvg   : 1          RTTMin: 1          RTTMax : 1

```

```
      NumOfRTT: 1          RTTSum: 1          RTTSum2: 1
      Bucket : 2
[snip]
```

Without examining every case, we will briefly examine IP SLA reaction configurations. In this example, we will configure XRv3 to issue a syslog message and start a path-echo operation [ID 312] if XRv4's loopback becomes unreachable after 3 consecutive failures of the ICMP-echo operation [ID 301]. The path-echo should only run for 5 minutes (300 seconds) when invoked. The configuration is rather involved, just like XE, and is shown below. The key command is "reaction trigger" which effectively says "if probe 301 experiences the trigger condition, start probe 312".

```
! XRv3
ipsla
  reaction operation 301
    react timeout
    action logging
    action trigger
    threshold type consecutive 3

  reaction trigger 301 312

  schedule operation 301
    start-time now
    life forever

  schedule operation 312
    start-time pending
    life 300
```

Initially, the ICMP-echo probe succeeds and the path-echo is not yet running, as expected.

```
RP/0/0/CPU0:XRv3#show ipsla statistics 301 | utility egrep 'Number|occurred'
  Number of operations attempted: 6
  Number of operations skipped   : 0
  Connection loss occurred       : FALSE
  Timeout occurred               : FALSE

RP/0/0/CPU0:XRv3#show ipsla statistics 312
Entry number: 312
  Modification time: 18:58:55.476 UTC [snip]
  Start time       : Not started yet
[snip]
```

If we temporarily apply an ACL to CSR1 that denies all ICMP echoes, we expect it see XRv3 log the event and start the path-echo probe after 3 consecutive failures. I will temporarily remove LDP from CSR1's interface to XRv3 so that traffic is raw IP, otherwise the ACL cannot classify it. Below is the syslog

message, failures from the original probe, and the starting of the second probe. The second probe fails too because it is ICMP-based but it doesn't matter for this test. I was just illustrating the reaction configuration to ensure that second probe was started at the proper time.

```
! CSR1
ip access-list extended ACL_DENY_ICMP_ECHO
  deny  icmp any any echo
  permit ip any any

interface GigabitEthernet2.513
  ip access-group ACL_DENY_ICMP_ECHO in
  no mpls ip

! XRv3
ipsla_sa[260]: %MGBL-IPSLA-5-THRESHOLD_SET : Monitor element has exceeded the
threshold condition. Op:301, TargetAddr:153.14.14.14,
MonElem:Timeout (Type:consecutive)

RP/0/0/CPU0:XRv3#show ipsla statistics 301 | utility egrep 'Number|occurred'
  Number of operations attempted: 9
  Number of operations skipped   : 9
  Connection loss occurred      : FALSE
  Timeout occurred              : TRUE

RP/0/0/CPU0:XRv3#show ipsla statistics 312 | utility egrep 'Number|occurred'
  Number of operations attempted: 11
  Number of operations skipped   : 0
  Connection loss occurred      : FALSE
  Timeout occurred              : TRUE
```

When I re-enabled MPLS on the link, the traffic succeeds as it tunnels through CSR1 so the ACL does nothing. Notice this log message says “threshold clear” and “reset”, while the first one said “threshold set” and “exceeded”. I also remove the ACL for cleanup purposes.

```
! CSR1
interface GigabitEthernet2.513
  no ip access-group ACL_DENY_ICMP_ECHO in
  mpls ip

! XRv3
ipsla_sa[260]: %MGBL-IPSLA-5-THRESHOLD_CLEAR : Monitor element has reset the
threshold reaction. Op:301, TargetAddr:153.14.14.14,
MonElem:Timeout (Type:consecutive)

RP/0/0/CPU0:XRv3#show ipsla statistics 301 | utility egrep 'Number|occurred'
  Number of operations attempted: 76
  Number of operations skipped   : 17
```



```
Connection loss occurred      : FALSE
Timeout occurred             : FALSE
```

TWAMP was discussed earlier but in the context of XR the feature set is identical; only the server and session-reflector are supported in XR. Like XE, “server” enables TWAMP server and “responder” enables TWAMP session-reflector, which for v1.0 must be on the same router. Also like XE, we can see the TWAMP standards for reference; this is useful when determining interoperability between different platforms or vendors.

```
! XRv4
ipsla
  server twamp
  port 60000
```

```
  responder
  twamp
```

```
RP/0/0/CPU0:XRv4#show ipsla twamp status
```

```
TWAMP Server is enabled
```

```
TWAMP Server port : 60000
```

```
TWAMP Reflector is enabled
```

```
RP/0/0/CPU0:XRv4#show ipsla twamp standards
```

Feature	Organization	Standard
TWAMP Server	IETF	RFC5357
TWAMP Reflector	IETF	RFC5357

Now we will examine the IPv6 SLA probes supported in XR. Their behavior is generally identical to their IPv4 counterparts, as we saw in XE also. We begin with the IPv6 ICMP-echo probe [ID 361] on XRv3. XR still calls the IPv6 “TOS” as opposed to the more correct “traffic-class”. There does not appear to be an option for assigning flow-labels at this time, either. This is probably a result of sharing the same syntax tree as IPv4 rather than being subcomponents of a probe-type as seen in XE.

```
! XRv3
operation 361
  type icmp echo
  source address ::153:13:13:13
  destination address ::153:14:14:14
  frequency 5

schedule operation 301
  start-time now
  life forever
```

We schedule the probe to ping XRv4’s loopback and see no issues.

```
RP/0/0/CPU0:XRv3#show ipsla statistics 361 | utility egrep 'Number|occurred'
```

```
Number of operations attempted: 7
Number of operations skipped   : 0
Connection loss occurred      : FALSE
Timeout occurred              : FALSE
```

None of the UDP probes appear to support IPv6 in XR v5.3.0. As such, they are not discussed further.

```
RP/0/0/CPU0:XRv4(config-ipsla-op)#type udp echo destination address ?
A.B.C.D Enter IPv4 address of the target device
```

```
RP/0/0/CPU0:XRv4(config-ipsla-op)#type udp jitter destination address ?
A.B.C.D Enter IPv4 address of the target device
```

Enhanced Object Tracking (EOT) is similar in XR as in IOS/XE. One enhancement is the ability to use named objects, not just numbers. However, XR only supports a limited number of objects: rtr (ip sla), ipv4 route (not IPv6), list, and line-protocol. We will quickly test these options using some dummy objects as their functions are identical to what was already tested and documented. There does not appear a way to set an object's default state. My workaround for this is to use the line-protocol object; to have an always-up object, track a loopback interface. To have an always-down object, track a non-existent interface. Below is the verification of my basic TRUE/FALSE objects; these are like stub-objects in a sense but cannot have their states changed by EEM.

```
! XRv3
track ALWAYS_TRUE
type line-protocol state
interface Loopback0
```

```
track ALWAYS_FALSE
type line-protocol state
interface Loopback999
```

```
RP/0/0/CPU0:XRv3#show track brief
```

Track	Object	Parameter	Value
ALWAYS_FALSE	interface Loopback999	line protocol	Down
ALWAYS_TRUE	interface Loopback0	line protocol	Up

Next, we will test the "route" object to test reachability to a specific prefix. I made two objects; one specific for the loopback /32 address, and one for a generic /24 for which we do not have an exact match. The former is up and the latter is down.

```
! XRv3
track XRV14_LOOPBACK_ROUTE
type route reachability
route ipv4 153.14.14.14/32
```

```
track XRV14_LOOPBACK_LESS_SPECIFIC
type route reachability
route ipv4 153.14.14.0/24
```

```
RP/0/0/CPU0:XRv3#show track XRV14_LOOPBACK_ROUTE
Track XRV14_LOOPBACK_ROUTE
  Ip route 153.14.14.14 255.255.255.255 reachability ip vrf default
  Reachability is UP
  First Hop interface GigabitEthernet0_0_0_0.534
  1 change, last change 15:53:12 UTC [snip]
```

```
RP/0/0/CPU0:XRv3#show track XRV14_LOOPBACK_LESS_SPECIFIC
Track XRV14_LOOPBACK_LESS_SPECIFIC
  Ip route 153.14.14.0 255.255.255.0 reachability ip vrf default
  Reachability is DOWN
  1 change, last change 15:54:13 UTC [snip]
```

Next, we will tie EOT to an IP SLA operation. EOT in XR does not support the “state” option, so measuring the threshold values inside EOT is not possible; only reachability appears is supported. The object below tracks the reachability of IP SLA ID 301 and when it times out, the object goes down. **This feature appears to be dysfunctional in XR v5.3.0 (it may be XRv-specific).** I have the IP SLA running successfully but EOT does not acknowledge it.

```
! XRv3
track XRV14_REACH
type rtr 301 reachability
```

```
RP/0/0/CPU0:XRv3#show ipsla statistics 301 | utility egrep 'Number|occurred'
  Number of operations attempted: 35
  Number of operations skipped   : 0
  Connection loss occurred       : FALSE
  Timeout occurred               : FALSE
```

```
RP/0/0/CPU0:XRv3#show track XRV14_REACH
Track XRV14_REACH
  Response Time Reporter 301 reachability
  ipsla operation not in progress
```

To test the object-list features, I will add three objects to track XRv3 loopback0 and its two transit interfaces to ensure the line-protocol is up. XR does not let me track the loopback using a route object, for some reason, but it the parser accepts it for the transit links.

```
! XRv3
track XRV3_TO_CSR1
type line-protocol state
interface GigabitEthernet0/0/0/0.513

track XRV3_TO_XRV4
```

```

type line-protocol state
interface GigabitEthernet0/0/0/0.534

track XRV3_LOOPBACK
type route reachability
route ipv4 153.13.13.13/32

```

The track process does not appear to actually track connected routes though, so using line-protocol is better for this test. I leave it in the configuration for Loopback0 as a demonstration.

```

! XRV3
track XRV3_LOOPBACK
type route reachability
route ipv4 153.13.13.13/32
!!% 'FIB' detected the 'warning' condition 'Invalid argument'

```

```

RP/0/0/CPU0:XRv3#show track brief | include XRV3
XRV3_LOOPBACK0      interface Loopback0                line protocol      Up
XRV3_TO_XRV4        interface GigabitEthernet0/0/0/0.534    reachability       Up
XRV3_TO_CSR1        interface GigabitEthernet0/0/0/0.513    reachability       Up

```

The logic of the weighted threshold list is very odd in XR. In XE, if the sum of the weights was higher than the up-value, the object was up. If the sum of the weights was lower than the down-value, the object was down. In XR, the sum of the weights must lie in the range of the two values to be up. The interesting thing about XR's implementation is that you can specify multiple threshold ranges, but the logical comparison between those ranges is Boolean AND. In this example, the list can suffer the loss of a single transit link only. The current sum of the weights is 20, which is within the range $11 \leq x \leq 20$.

```

! XRV3
track LIST_WEIGHT
type list threshold weight
object XRV3_TO_CSR1 weight 5
object XRV3_TO_XRV4 weight 5
object XRV3_LOOPBACK0 weight 10
threshold weight up 20 down 11

RP/0/0/CPU0:XRv3#show track LIST_WEIGHT
Track LIST_WEIGHT
List threshold weight is UP
4 changes, last change 16:18:24 UTC [snip]
object XRV3_LOOPBACK0 UP
object XRV3_TO_XRV4 UP
object XRV3_TO_CSR1 UP

```

Adding a bogus threshold brings the object down, because it's impossible that a number is $11 \leq x \leq 20$ and $400 \leq x \leq 500$ concurrently. This proves that Boolean AND logic is used between ranges.

```

! XRV3
track LIST_WEIGHT
  type list threshold weight
    object XRV3_TO_CSR1 weight 5
    object XRV3_TO_XRV4 weight 5
    object XRV3_LOOPBACK0 weight 10
    threshold weight up 20 down 11
    threshold weight up 500 down 400

RP/0/0/CPU0:XRv3#show track LIST_WEIGHT
Track LIST_WEIGHT
  List threshold weight is DOWN
  11 changes, last change 16:27:51 UTC [snip]
    object XRV3_LOOPBACK0 UP
    object XRV3_TO_XRV4 UP
    object XRV3_TO_CSR1 UP

```

Let's remove the bogus threshold (not shown). If we shut down the link to CSR1, the object stays up. This is because the sum of the weights becomes 15, which is valid ($11 \leq 15 \leq 20$).

```

RP/0/0/CPU0:XRv3#show track LIST_WEIGHT
Track LIST_WEIGHT
  List threshold weight is UP
  6 changes, last change 16:23:45 UTC [snip]
    object XRV3_LOOPBACK0 UP
    object XRV3_TO_XRV4 UP
    object XRV3_TO_CSR1 DOWN

```

If we also shut down the link to XRV4 while the link to CSR1 is still down, the object goes down since the weight becomes 10; this crosses the "down" threshold.

```

RP/0/0/CPU0:XRv3#show track LIST_WEIGHT
Track LIST_WEIGHT
  List threshold weight is DOWN
  7 changes, last change 16:25:24 UTC [snip]
    object XRV3_LOOPBACK0 UP
    object XRV3_TO_XRV4 DOWN
    object XRV3_TO_CSR1 DOWN

```

Even with both transit links up, shutting down the loopback brings the whole list down. The sum of the weights is 10, like the previous example, and the list is down for the same reason.

```

RP/0/0/CPU0:XRv3#show track LIST_WEIGHT
Track LIST_WEIGHT
  List threshold weight is DOWN
  9 changes, last change 16:25:51 UTC [snip]
    object XRV3_LOOPBACK0 DOWN
    object XRV3_TO_XRV4 UP

```

```
object XRV3_TO_CSR1 UP
```

The percentage threshold list is similar to the weighted one except all objects contribute equally to the list's status. A list with three objects allows each to contribute 33% to the list, and the threshold determines at what point is the list considered up. This list can tolerate the loss of up to one interface.

```
! XRV3
track LIST_PERCENT
type list threshold percentage
object XRV3_TO_CSR1
object XRV3_TO_XRV4
object XRV3_LOOPBACK0
threshold percentage up 100 down 66

! XRV3
Track LIST_PERCENT
List threshold percentage is UP
1 change, last change 16:33:06 UTC [snip]
object XRV3_LOOPBACK0 UP
object XRV3_TO_XRV4 UP
object XRV3_TO_CSR1 UP
```

First, we shutdown the loopback and notice the object stays up, unlike the weighted list.

```
RP/0/0/CPU0:XRv3#show track LIST_PERCENT
Track LIST_PERCENT
List threshold percentage is UP
3 changes, last change 16:35:10 UTC [snip]
object XRV3_LOOPBACK0 DOWN
object XRV3_TO_XRV4 UP
object XRV3_TO_CSR1 UP
```

Additionally, we shutdown the link to CSR1 and the object goes down as expected.

```
RP/0/0/CPU0:XRv3#show track LIST_PERCENT
Track LIST_PERCENT
List threshold percentage is DOWN
4 changes, last change 16:35:37 UTC [snip]
object XRV3_LOOPBACK0 DOWN
object XRV3_TO_XRV4 UP
object XRV3_TO_CSR1 DOWN
```

Access-list based forwarding (ABF) is the XR equivalent to policy-based routing (PBR) in XE. We use it here primarily to demonstrate object tracking by reusing some objects from earlier. The ACL prefers to route HTTP traffic to CSR1 if the interface is up, and prefers to route voice to XRV4 if that interface is up. Finally, it prefers to route telnet to CSR1 only if both interfaces are up. I created a Boolean AND list to accomplish this. I also used the ALWAYS_FALSE object and negated it to demonstrate the NOT feature

(the object is always true now). The output from the show command for tracked objects is interesting because it doesn't actually say ALWAYS_FALSE is "up", it just says "not" plus whatever the state is. I did not apply the list anywhere but left it in the configuration for reference.

```
! XRv1
track LIST_BOOLEAN_AND
type list boolean and
  object ALWAYS_FALSE not
  object XRV3_TO_CSR1
  object XRV3_TO_XRV4

RP/0/0/CPU0:XRv3#show track LIST_BOOLEAN_AND
Track LIST_BOOLEAN_AND
  List boolean and is UP
    1 change, last change 16:49:43 UTC [snip]
      object XRV3_TO_XRV4 UP
      object XRV3_TO_CSR1 UP
      object ALWAYS_FALSE not DOWN

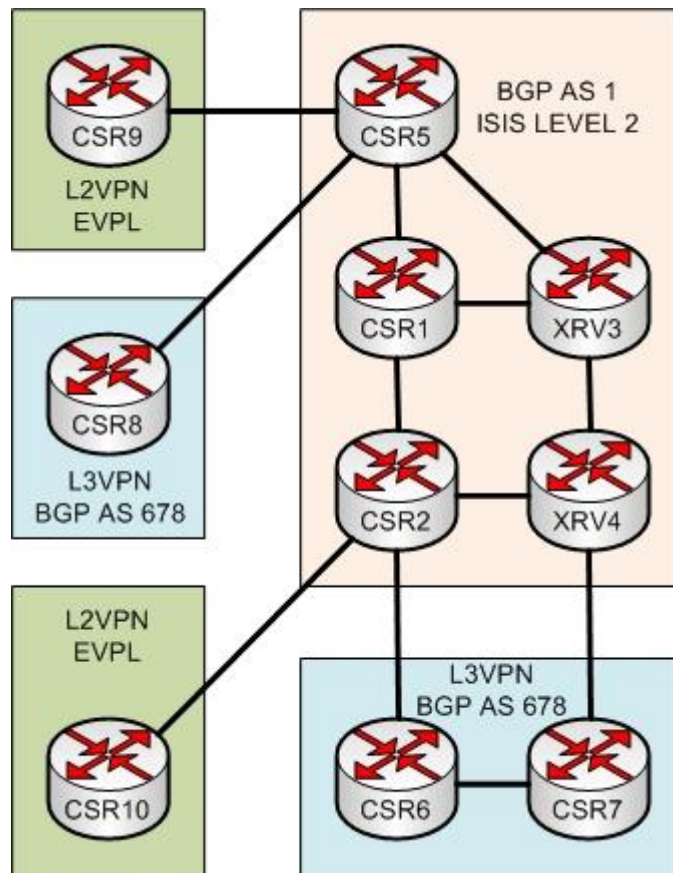
! XRv3
ipv4 access-list ACL_POLICY_ROUTE
10 permit tcp any any eq www nexthop1 track XRV3_TO_CSR1 ipv4 153.1.13.1
20 permit udp any any range 16384 32767 nexthop1 track XRV3_TO_XRV4 ipv4 153.13.14.14
30 permit tcp any any eq telnet nexthop1 track LIST_BOOLEAN_AND ipv4 153.1.13.1
```

48. Describe, implement, and troubleshoot MPLS OAM and Ethernet OAM

As a general comment, all MVPN OAM related commands are tested under their respective MVPN sections. There is too much variation between the MVPN profiles to integrate them intelligently below. The unicast OAM mechanisms are discussed in this section and used throughout many other labs.

48.1 MPLS ping, MPLS traceroute, and VCCV

MPLS ping and traceroute are valuable tools to capture more information about MPLS LSPs. They integrate well with IP SLA and are basic troubleshooting tools for MPLS network administrators. The network below uses basic L3VPN and L2VPN to test OAM functionality. CSR6, CSR7, and CSR8 are L3VPN customers while CSR9 and CSR10 are L2VPN customers. The routers in AS 1 make up the MPLS core.



The MPLS ping syntax is similar to an ordinary IPv4/v6 ping except instead of specifying a destination IP address, you specify a destination prefix, which represents a FEC. This is because MPLS label allocation mechanisms (LDP, BGP, etc) will bind labels to prefixes and the routing table might have multiple routes. For example, if CSR1 has an IGP route to 153.14.14.14/32 with an LDP label and another IGP route to 153.14.14.0/24 with a different LDP label, the MPLS ping allows us to test both FECs independently. Normally, pinging or tracerouting 153.14.14.14 by itself would always follow the /32 path unless it was not available (basic longest-match routing). The MPLS ping requires that the FEC being probed exists in the routing table with an appropriate label binding based on the supported FEC types (varies between platforms and versions). I've temporarily added 153.14.14.14/24 as a loopback on CSR5 and advertised it into IS-IS to demonstrate this (configuration not shown). The verbose option lets us see the source of the reply. We can clearly see different routers replying to each probe.

```
R2#ping mpls ipv4 153.14.14.14/32 source 153.2.2.2 verbose repeat 1
Sending 1, 72-byte MPLS Echos to 153.14.14.14/32,
    timeout is 2 seconds, send interval is 0 msec:
```

```
Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
[snip]
```

```
Type escape sequence to abort.
```

```
!    size 72, reply addr 153.13.14.14, return code 3
```



```
Success rate is 100 percent (1/1), round-trip min/avg/max = 9/9/9 ms
Total Time Elapsed 9 ms
```

```
R2#ping mpls ipv4 153.14.14.0/24 source 153.2.2.2 verbose repeat 1
Sending 1, 72-byte MPLS Echos to 153.14.14.0/24,
    timeout is 2 seconds, send interval is 0 msec:
```

```
Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
[snip]
```

```
Type escape sequence to abort.
```

```
!      size 72, reply addr 153.1.5.5, return code 3
```

```
Success rate is 100 percent (1/1), round-trip min/avg/max = 8/8/8 ms
Total Time Elapsed 8 ms
```

An interesting characteristic of the MPLS ping is that the destination of the IP packet inside of the MPLS ping is set to a local address (127.0.0.0/8). This ensures that if the LSP is broken, that packet will not simply be forwarded via IP routing. Using EPC on CSR1, we send the ping again via CSR2. The 4 bytes after the MPLS ethertype (0x8847) represent the top-most (and only) label in the stack currently. The value is 0x003E9 which is 1004 and highlighted in yellow; this is CSR1's label for 153.14.14.0/24. The S-bit is set and the TOS is zero since the next hex digit is 0x1 (0001). The TTL is 0xFF or 255 since the router has not decremented it yet. I also highlight the IP protocol, which is UDP (0x11 is 17 in decimal and highlighted green); we may have expected to see ICMP here. I highlight the source IP 0x99020202 which is CSR2's loopback in hex (cyan). Following that is the destination IP address 0x7f000001, which is 127.0.0.1, shown in pink. This is not a routable IP address. I also highlight the source and destination UDP ports 0x0DAF in grey, which is 3503, the well-known MPLS LDP ping port.

```
R1#show monitor capture CAP buffer detailed
  1 118      0.642017 00:50:56:A9:BE:8A -> 00:50:56:A9:1A:AA MPLS unicast
0000: 005056A9 1AAA0050 56A9BE8A 81000DB8 .PV....PV.....
0010: 8847003E C1FF4600 00602256 40000111 .G.>..F..`"V@...
0020: A82D9902 02027F00 00019404 00000DAF .-.....
0030: 0DAF0048 10650001 00000102 000066C4 ...H.e.....f.
```

```
R1#show mpls forwarding-table labels 1004
```

Local	Outgoing	Prefix	Bytes	Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched		interface	
1004	Pop Label	153.14.14.0/24	5884		Gi2.515	153.1.5.5

We know that due to PHP on CSR1, this is sent as raw IP to CSR5, which won't route the packet further. We theorize that the MPLS ping we just tested may not ensure that the link between the penultimate hop and the ultimate hop is MPLS enabled. To test this, we disable LDP on the link between CSR1 and CSR5. The LFIB used to "pop" the topmost label; now it removes all labels from the stack.

```
! CSR1
interface GigabitEthernet2.515
 no mpls ldp igp autoconfig
```

```
R1#show mpls forwarding-table 153.14.14.0 24
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Label	Outgoing interface	Next Hop
1004	No Label	153.14.14.0/24	7024		Gi2.515	153.1.5.5

Fortunately, the LSP ping still tells us there is a fault as it reports “unlabeled interface”, but the ping still completes unsuccessfully. It does not timeout when you run it from exec, but IP SLA does report it as a timeout. This shows that we do have IP connectivity, but not MPLS connectivity.

```
R2#ping mpls ipv4 153.14.14.0/24 source 153.2.2.2
Sending 5, 72-byte MPLS Echos to 153.14.14.0/24,
 timeout is 2 seconds, send interval is 0 msec:
```

```
Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
 'L' - labeled output interface, 'B' - unlabeled output interface,
```

Type escape sequence to abort.

```
BBBBB
```

```
Success rate is 0 percent (0/5)
```

```
Total Time Elapsed 11 ms
```

We can ensure the ping follows an LSP everywhere using “force-explicit-null”, which adds an explicit-null at the bottom of the stack. This way, there is no possible way a path could not be enabled for MPLS anywhere in the path. Now the manual test times out. This additional shim could also be valuable for testing MPLS fragmentation when coupled with large LSP pings.

```
R2#ping mpls ipv4 153.14.14.0/24 source 153.2.2.2 force-explicit-null
Sending 5, 72-byte MPLS Echos to 153.14.14.0/24,
 timeout is 2 seconds, send interval is 0 msec:
```

```
Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
```

Type escape sequence to abort.

```
.....
```

```
Success rate is 0 percent (0/5)
```

```
Total Time Elapsed 9636 ms
```

We can introduce MPLS traceroute to troubleshoot this timeout problem. The command syntax is almost identical to the LSP ping, except we use the “traceroute” keyword. I’ve added TTL=2 so the failed traceroute dies quickly instead of trying to trace 30 hops. Notice that on hop index 1, there is no label stack at all. Traceroute indicates that the LSP suddenly ended along a FEC unexpectedly. The output

limits the scope of the problem to the link between CSR1 and CSR2, which is valuable for fault isolation and troubleshooting in large networks.

```
R2#traceroute mpls ipv4 153.14.14.0/24 source 153.2.2.2 force-explicit-null
ttl 2
```

```
Tracing MPLS Label Switched Path to 153.14.14.0/24, timeout is 2 seconds
```

```
Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
       'P' - no rx intf label prot, 'p' - premature termination of LSP,
```

```
Type escape sequence to abort.
```

```
0 153.1.2.2 MRU 1500 [Labels: 1004/explicit-null Exp: 0/0]
```

```
p 1 153.1.2.1 9 ms
```

```
. 2 *
```

We will now shutdown this temporary /24 interface on CSR5 and use real FECs from now on. CSR1 enables LDP towards CSR5 again for cleanup (not shown). Another common return code is “Q” for source-quench. This means that the source didn’t even send the probe for some reason. On CSR1, I have filtered advertisement of the local label for 153.14.14.14/32 (temporarily).

```
! CSR1
ip access-list standard ACL_NO_XRV4_LABEL
deny 153.14.14.14
permit any
```

```
no mpls ldp advertise-labels
mpls ldp advertise-labels for ACL_NO_XRV4_LABEL
```

```
R1#show mpls ldp bindings 153.14.14.14 32 advertisement-acls
Advertisement spec:
Prefix acl = ACL_NO_XRV4_LABEL
```

```
R2#show mpls forwarding-table 153.14.14.14 32
```

Local	Outgoing	Prefix	Bytes Label	Outgoing	Next Hop
Label	Label	or Tunnel Id	Switched	interface	
2002	No Label	153.14.14.14/32	0	Gi2.512	153.1.2.1

Running the probe returns the “Q” flag as expected. Because the originating router doesn’t have a valid FEC there is nothing more to do. This is a way to see if the beginning of the LSP has MPLS/label issues.

```
R2#ping mpls ipv4 153.14.14.14/32 source 153.2.2.2
Sending 5, 72-byte MPLS Echos to 153.14.14.14/32,
timeout is 2 seconds, send interval is 0 msec:
```

```
Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
```

```
Type escape sequence to abort.
```

00000

Success rate is 0 percent (0/5)

Total Time Elapsed 0 ms

The LSP ping tests LSPs; remember that LSPs are unidirectional. This means that the return path could be non-MPLS and the MPLS ping would still work. On CSR5, we create a new ACL to disallow CSR5 from advertising a label for CSR2's loopback. Return traffic cannot be MPLS encapsulated since XRv3 won't have a label for 153.2.2.2/32.

```
! CSR5
ip access-list standard ACL_NO_CSR2_LABEL
deny 153.2.2.2
permit any

no mpls ldp advertise-labels
mpls ldp advertise-labels for ACL_NO_CSR2_LABEL
```

RP/0/0/CPU0:XRv3#show mpls forwarding-table prefix 153.2.2.2/32

Local Label	Outgoing Label	Prefix or ID	Outgoing Interface	Next Hop	Bytes Switched
93003	Unlabelled	153.2.2.2/32	Gi0/0/0/0.535	153.5.13.5	0

The ping from CSR2 to XRv4 works but the opposite is not true. CSR2 has a valid LSP to 153.14.14.14/32 but XRv4 does not have a valid LSP to 153.2.2.2/32.

```
R2#ping mpls ipv4 153.14.14.14/32 source 153.2.2.2
Sending 5, 72-byte MPLS Echos to 153.14.14.14/32,
    timeout is 2 seconds, send interval is 0 msec:
[snip]
Type escape sequence to abort.
```

!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 2/3/8 ms

Total Time Elapsed 20 ms

```
RP/0/0/CPU0:XRv4#ping mpls ipv4 153.2.2.2/32 source 153.14.14.14 verbose
repeat 1
```

```
Sending 1, 100-byte MPLS Echos to 153.2.2.2/32,
    timeout is 2 seconds, send interval is 0 msec:
```

```
Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
       'L' - labeled output interface, 'B' - unlabeled output interface,
```

Type escape sequence to abort.

B size 100, reply addr 153.13.14.13, return code 9

Of note, XR routers must force exp-null if PHP occurs when using LSP ping. To test the LSP from XRv4 to CSR2, we use the following ping. I've set the EXP just to demonstrate the feature. The XE routers do not appear to be as strict when testing LSPs.

```
RP/0/0/CPU0:XRv4#ping mpls ipv4 153.2.2.2/32 source 153.14.14.14 force-  
explicit-null exp 4
```

```
Sending 5, 100-byte MPLS Echos to 153.2.2.2/32,  
  timeout is 2 seconds, send interval is 0 msec:
```

Type escape sequence to abort.

```
!!!!
```

```
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/6/20 ms
```

Both XE and XR can specify the FEC type for the LSP also by selecting LDP or BGP. Since we do not have BGP labeled-unicast configured, selecting that FEC-type results in a lack of FEC mapping for the LSP. I show both the XE and XR outputs. For XR, notice I include exp-null and the verbose option for variety.

```
R1#ping mpls ipv4 153.5.5.5/32 fec-type bgp source 153.1.1.1  
Sending 5, 72-byte MPLS Echos to BGP + Label IP FEC 153.5.5.5/32,  
  timeout is 2 seconds, send interval is 0 msec:
```

```
Codes: '!' - success, 'Q' - request not sent, '.' - timeout,  
  'D' - DS Map mismatch, 'F' - no FEC mapping, 'f' - FEC mismatch,
```

Type escape sequence to abort.

```
FFFFF
```

```
Success rate is 0 percent (0/5)
```

```
Total Time Elapsed 11 ms
```

```
RP/0/0/CPU0:XRv4#ping mpls ipv4 153.2.2.2/32 source 153.14.14.14 force-  
explicit-null fec-type bgp verbose repeat 1
```

```
Sending 1, 100-byte MPLS Echos to 153.2.2.2/32,  
  timeout is 2 seconds, send interval is 0 msec:
```

Type escape sequence to abort.

```
F      size 100, reply addr 153.1.2.2, return code 4
```

The MPLS ping has some other interesting characteristics and options. All LSP probes use the IP option “router-alert” so that that IP packet is automatically punted to the CPU, bypassing any hardware dependencies on the probe target. This adds another 4 bytes of overhead to the IP header and cannot be disabled. The originator (head) of the probe can determine the “reply mode” for the receiver (tail). The default is an ordinary IPv4 UDP packet which may or may not be MPLS encapsulated; it depends entirely on how the tail end reaches the headend. The other option is to add the “router-alert” option to

the reply packet also to bypass hardware dependencies on the return path. This does not guarantee the LSP works in both directions, since they are unidirectional, but might be useful for troubleshooting. We will use some debugs to help us identify the presence or absence of the “router-alert” option. Using Wireshark, we can see what this option looks like because we have never seen it before. Once we see the bits, we can determine its hex string: 0x94040000. This will make it easily identifiable in EPC captures.

```

Options: (4 bytes), Router Alert
  Router Alert (4 bytes): Router shall examine packet (0)
    Type: 148
      1... .... = Copy on fragmentation: Yes
      .00. .... = Class: Control (0)
      ...1 0100 = Number: Router Alert (20)
    Length: 4
    Router Alert: Router shall examine packet (0)

```

Next, I will enable some MPLS LSP verification (lspv) debugging so we can see the key events and data packets. I will send a single LSP ping with reply-mode router-alert so I don't overwhelm the console. Notice the reply-mode is “3”; this is an identifier for the router-alert reply-mode. The outgoing packet shows the packet encapsulation for layer 3 and above (MPLS not included). I have highlighted the router-alert IP option in the request. Highlighted in green is the “number of 4-byte words” in the IP header, which is now 6. This implies that the IP header length is 24 bytes, not the standard 20. Notice that just before the router-alert option is the destination IP address, 127.0.0.1, which is shown in cyan. When the reply packet returns, the debug shows us the encapsulation for layer 2 and above. After sorting through the Ethernet and 802.1Q headers, we see the IP packet with the router-alert option set in the reply. Notice that the reply packet was 98 bytes in length.

```

R1#debug mpls lspv packet data
R1#debug mpls lspv event

```

```

R1#ping mpls ipv4 153.14.14.14/32 source 153.2.2.2 reply mode router-alert
repeat 1

```

```

LSPV: label_shim_mode 0, label_ra_shim_mode 0, reply_mode 3, source_addr
153.2.2.2, dest_addr 127.0.0.1
LSPV: Echo Request to be sent on IPv4 LSP, pathindex 0, size 72, datagramsize
96
46 00 00 60 B8 C1 40 00 01 11 11 C2 99 02 02 02
7F 00 00 01 94 04 00 00 0D AF 0D AF 00 48 32 04
00 01 00 00 01 03 00 00 66 C4 C5 C4 00 00 00 01
D9 4C DF B7 E3 D7 0A 3D 00 00 00 00 00 00 00
FC 00 00 0C 00 00 00 09 00 01 00 04 00 00 00 04
00 01 00 0C 00 01 00 05 99 0E 0E 0E 20 00 00 00
LSPV: waiting for 2000 msec
LSPV: sender_handle: 0x66C4C5C4, [->Waiting for Echo Reply]
LSPV: Echo packet received: rx interface Gi2.512, ip tableid 0, src
153.13.14.14, dst 153.2.2.2, datagramsize 98

```

```

00 50 56 A9 BE 8A 00 50 56 A9 1A AA 81 00 0D B8
08 00 46 C0 00 50 09 0C 00 00 FB 11 DE AC 99 0D
0E 0E 99 02 02 02 94 04 00 00 0D AF 0D AF 00 38
DA 63 00 01 00 00 02 03 03 01 66 C4 C5 C4 00 00
00 01 D9 4C DF B7 E3 D7 0A 3D D9 4C DF B7 69 E6
CF E9 FC 00 00 0C 00 00 00 09 00 01 00 04 00 00
00 04

```

LSPV: Packet received with 0 labels

LSPV: sender_handle: 0x66C4C5C4, Event Echo Reply Received, [Waiting for Echo Reply->]

Using the default reply-mode of IPv4, we see that the router-alert is still present in the request, but not in the reply. The response is 4 bytes shorter (94 versus 98) and we do not see the router-alert option in the IPv4 header of the echo-reply. This uses reply-mode “2” as the code; this is actually buried in the LSP payload and is not examined. I’ve highlighted the place the router-alert would have been, but in this case, it’s the first half of the UDP header which are the port numbers for LSP ping. Just for extra testing, I set reply DSCP EF to make sure the feature works (DSCP EF = 46 in decimal, 184 as TOS = 0xB8). I’ve also highlighted 0xB8 in green which is DSCP EF, present only on the reply, not the on request. The length of the reply IP header is 20 bytes again (5 words, no 6) since the router-alert option is absent. Lastly, I’ve highlighted the router-alert option in the request; it is always there.

```
R1#debug mpls lspv packet data
```

```
R1#debug mpls lspv event
```

```
R1#ping mpls ipv4 153.14.14.14/32 source 153.2.2.2 reply dscp ef repeat 1
```

```
LSPV: label_shim_mode 0, label_ra_shim_mode 0, reply_mode 2, source_addr
153.2.2.2, dest_addr 127.0.0.1
```

```
LSPV: fec_type 0, stlvtype 1
```

```
LSPV: Echo Request to be sent on IPv4 LSP, pathindex 0, size 72, datagramsize
112
```

```

46 00 00 70 BA 1C 40 00 01 11 10 57 99 02 02 02
7F 00 00 01 94 04 00 00 0D AF 0D AF 00 58 C3 23
00 01 00 00 01 02 00 00 66 C4 C5 C5 00 00 00 01
D9 4C E1 F0 44 9B A5 E3 00 00 00 00 00 00 00
FC 00 00 14 00 00 00 09 00 01 00 04 00 00 00 04
00 02 00 04 B8 00 00 00 00 01 00 0C 00 01 00 05
99 0E 0E 0E 20 00 00 00 00 0A 00 04 B8 00 00 00

```

```
LSPV: waiting for 2000 msec
```

```
LSPV: sender_handle: 0x66C4C5C5, [->Waiting for Echo Reply]
```

```
LSPV: Echo packet received: rx interface Gi2.512, ip tableid 0, src
153.13.14.14, dst 153.2.2.2, datagramsize 94
```

```

00 50 56 A9 BE 8A 00 50 56 A9 1A AA 81 00 0D B8
08 00 45 B8 00 4C 09 0D 00 00 FB 11 73 BC 99 0D
0E 0E 99 02 02 02 0D AF 0D AF 00 38 E2 40 00 01
00 00 02 02 03 01 66 C4 C5 C5 00 00 00 01 D9 4C
E1 F0 44 9B A5 E3 D9 4C E1 EF C8 DD 68 3A FC 00
00 0C 00 00 00 09 00 01 00 04 00 00 00 00 04

```

```
LSPV: Packet received with 0 labels
```

```
LSPV: sender_handle: 0x66C4C5C5, Event Echo Reply Received, [Waiting for Echo Reply->]
```

We can also use the MPLS ping to probe routers for their downstream mappings (dsmap) for the FEC we are trying to reach. You must set the TTL accordingly to target specific routers in the path; in this case, I set TTL=1 from XRv4 so we can see what XRv3's label stack is. Not surprisingly, there is no label.

```
RP/0/0/CPU0:XRv4#ping mpls ipv4 153.2.2.2/32 source 153.14.14.14 dsmap ttl 1
repeat 1
Sending 1, 100-byte MPLS Echos to 153.2.2.2/32,
    timeout is 2 seconds, send interval is 0 msec:
```

```
Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
       'L' - labeled output interface, 'B' - unlabeled output interface,
```

Type escape sequence to abort.

```
B Echo Reply received from 153.13.14.13
  DSMAP 0, DS Router Addr 153.5.13.5, DS Intf Addr 153.5.13.5
    Depth Limit 0, MRU 1500 [No Label]
[snip]
```

After removing the LDP label filter from CSR5, we repeat the command to see what XRv3's label binding should be. The query returns label value 5001. We could trace the entire LSP this way instead of using traceroute if we wanted to.

```
RP/0/0/CPU0:XRv4#ping mpls ipv4 153.2.2.2/32 source 153.14.14.14 dsmap ttl 1
repeat 1
Sending 1, 100-byte MPLS Echos to 153.2.2.2/32,
    timeout is 2 seconds, send interval is 0 msec:
```

```
Codes: '!' - success, 'Q' - request not sent, '.' - timeout,
       'L' - labeled output interface, 'B' - unlabeled output interface,
```

Type escape sequence to abort.

```
L Echo Reply received from 153.13.14.13
  DSMAP 0, DS Router Addr 153.5.13.5, DS Intf Addr 153.5.13.5
    Depth Limit 0, MRU 1500 [Labels: 5001 Exp: 0]
[snip]
```

One of the most powerful uses of the DSMAP is to determine load-sharing. Recall that MPLS performs load-sharing three possible ways: if the tunneled packet is IPv4 or IPv6, load-sharing is based on a hash of the source and destination addresses. If its non-IP traffic, load-sharing is based on the bottom label. We will specifically look at IPv4 here. But first, we need to change IGP to give us multiple paths. We change all of CSR5's transit links to metric 5 so CSR1 also has two equal-cost paths to XRv4 (not shown).

The total path cost will be 30 regardless of which path is selected. We verify load sharing on CSR1 via the LFIB show command.

```
R1#show mpls forwarding-table 153.14.14.14 32
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Outgoing interface	Next Hop
1002	5004	153.14.14.14/32	146	Gi2.515	153.1.5.5
	93000	153.14.14.14/32	103394	Gi2.513	153.1.13.13

Next, we query CSR1 for its downstream label mappings from CSR2. Based on CSR1's LFIB above, we can see those labels are correct, but we see there are two "hash" buckets, 0 and 1 (highlighted), each corresponding to a different downstream path.

```
R2#ping mpls ipv4 153.14.14.14/32 source 153.2.2.2 ttl 1 dsmap repeat 1
Sending 1, 72-byte MPLS Echos to 153.14.14.14/32,
    timeout is 2 seconds, send interval is 0 msec:
```

Type escape sequence to abort.

```
L Echo Reply received from 153.1.2.1
```

```
DSMAP 0, DS Router Addr 153.1.5.5, DS Intf Addr 153.1.5.5
    Depth Limit 0, MRU 1500 [Labels: 5004 Exp: 0]
    Multipath Addresses:
```

```
DSMAP 1, DS Router Addr 153.1.13.13, DS Intf Addr 153.1.13.13
    Depth Limit 0, MRU 1500 [Labels: 93000 Exp: 0]
    Multipath Addresses:
```

```
Success rate is 0 percent (0/1)
Total Time Elapsed 8 ms
```

We can also ask the router what its multipath distribution looks like. Remember, ECMP with MPLS will be done based on the source/destination hash for IPv4 and IPv6 packets. Even though the destination is a 127.0.0.0/8 local-host address, this value does matter for load-sharing. MPLS ping takes advantage of this and the DSMAP can tell you, based on the destination (combined with a user-specified source), where the packet goes. We need to include a hashkey for IPv4 and, in a value called "bitmap", specify the number of addresses we want to see from that node for load-sharing. For example, given 8 different destination addresses, we reveal which addresses map to which paths. By doing this ahead of time, we can construct MPLS pings with the proper destinations to test all ECMP paths reliably and concurrently.

```
R2#ping mpls ipv4 153.14.14.14/32 source 153.2.2.2 ttl 1 dsmap hashkey ipv4
bitmap 8 repeat 1
[snip]
```

```
L Echo Reply received from 153.1.2.1
```

```
DSMAP 0, DS Router Addr 153.1.5.5, DS Intf Addr 153.1.5.5
    Depth Limit 0, MRU 1500 [Labels: 5004 Exp: 0]
```

Multipath Addresses:

127.0.0.1 127.0.0.4 127.0.0.5 127.0.0.8

DSMAP 1, DS Router Addr 153.1.13.13, DS Intf Addr 153.1.13.13

Depth Limit 0, MRU 1500 [Labels: 93000 Exp: 0]

Multipath Addresses:

127.0.0.2 127.0.0.3 127.0.0.6 127.0.0.7

[snip]

We can send two MPLS pings from CSR2 to XRv4's loopback as the FEC target, but specify IPv4 destinations according to CSR1's DSMAP. We will clear CSR1's LFIB counters, then send a pair of pings. The first will not specify a destination since we know the default is 127.0.0.1, so this should use CSR5 for transport according to the DSMAP. Then, we will specify a destination of 127.0.0.6 to test the path via XRv3, because CSR2 said that address would be routed towards XRv3 in the DSMAP. We see almost exactly the same amount of bytes label switched (minor differences due to LDP/BGP session traffic, etc). Tying this destination address into IP SLA probes is an automated way to test all ECMP paths and is discussed later. This destination address is often referred to as the "LSP selector".

```
R2#ping mpls ipv4 153.14.14.14/32 source 153.2.2.2 repeat 100 size 1450
```

```
R1#show mpls forwarding-table 153.14.14.14 32
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
1002	5004	153.14.14.14/32	147488	Gi2.515	153.1.5.5
	93000	153.14.14.14/32	0	Gi2.513	153.1.13.13

```
R2#ping mpls ipv4 153.14.14.14/32 source 153.2.2.2 repeat 100 size 1450
```

```
destination 127.0.0.6
```

```
R1#show mpls forwarding-table 153.14.14.14 32
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop
1002	5004	153.14.14.14/32	147838	Gi2.515	153.1.5.5
	93000	153.14.14.14/32	147492	Gi2.513	153.1.13.13

Aside from MPLS OAM, when we do a normal ping from CSR2 to XRv4 using the loopbacks, we can query CSR1 to see which way it would route the traffic by checking the LFIB. This command is similar to what we can do with CEF to determine which load-sharing paths are used. We need to specify the local (incoming) label, followed by the IPv4 or IPv6 source/destination pair. We see that CSR5 is the preferred transit path. I follow this verification with a quick ICMP ping test (assume counters have been cleared).

```
R1#show mpls forwarding-table exact-route label 1002 ipv4 source 153.2.2.2 destination 153.14.14.14
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Label Switched	Outgoing interface	Next Hop

```
1002          5004          153.14.14.14/32  0          Gi2.515      153.1.5.5
```

```
R2#ping 153.14.14.14 source 153.2.2.2 size 1450 repeat 100
```

```
R1#show mpls forwarding-table 153.14.14.14 32
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Label	Outgoing interface	Next Hop
1002	5004	153.14.14.14/32	154768		Gi2.515	153.1.5.5
	93000	153.14.14.14/32	146		Gi2.513	153.1.13.13

Initially, this may sound suboptimal, because all of the MPLS traffic between this set of LSP endpoints will never use the link to XRv3. This is not true, since the VPN traffic plays a role also. For non-IP flows, the bottom-most VPN label influences the choice, and for IPv4/IPv6, the VPN source/destination influence it. Below is a coincidental and convenient example showing CSR6 sending traffic to CSR7 over MPLS. CSR1 it uses the link to XRv3 as shown below.

```
R1#show mpls forwarding-table exact-route label 1002 ipv4 source 6.6.6.6 destination 7.7.7.7
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Label	Outgoing interface	Next Hop
1002	93000	153.14.14.14/32	0		Gi2.513	153.1.13.13

```
R6#ping 7.7.7.7 source 6.6.6.6 repeat 100 size 1450
```

```
R1#show mpls forwarding-table 153.14.14.14 32
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Label	Outgoing interface	Next Hop
1002	5004	153.14.14.14/32	350		Gi2.515	153.1.5.5
	93000	153.14.14.14/32	147746		Gi2.513	153.1.13.13

Verifying layer 2 circuits is a bit different. A pseudowire exists from CSR2 to CSR5 to test MPLS ping over that link. We will test its health using Virtual Circuit Connectivity Verification (VCCV). There are two key mechanisms of VCCV that are negotiated between the peers when tLDP is in use.

CV: Connectivity Verification. This is a 1-byte field in the VCCV parameter and describes the method by which the PW can be verified. There are two types:

ICMP Ping (0001 or 0x1): Uses ICMP for IPv4 or IPv6 depending on how the PW was established. Not supported by Cisco, at least on the CSR1000v and XRv.

LSP Ping (0010 or 0x2): Uses the usual UDP echo we saw earlier to test the PW. Inside the payload, it includes the FEC, reply-mode, and all the other details. This is what we have tested thus far.

CC: Control Channel. This is a 1-byte field in the VCCV parameter and determines the mechanics of performing the verification. There are three types:

Type 1: In-Band VCCV (0001 or 0x1): Using the PW associated channel header (PW-ACH), the signaling for VCCV is carried in the control-word (CW). This is the preferred VCCV mechanism and is what Cisco routers prefer. It is only used if CW capabilities are negotiated. The CW is discussed in great detail in the L2VPN section.

Type 2: Out-of-Band VCCV (0010 or 0x2): Uses the MPLS router-alert (RA) label as the second-to-bottom label, right above the PW label. When the topmost label is PHP'ed, the remote PE sees this router-alert and punts it to the CPU for processing. This is the next most preferable way to do VCCV if the control-word isn't present. The RA label has a value of 1 and can technically appear anywhere in the label stack except the bottom. Its purpose, in general, is identical to that of the router-alert IP option seen earlier.

Type 3: TTL Expiry VCCV (0100 or 0x4): Using TTL=1 on the PW label forces the packet to be examined by the remote PE's control-plane. This is the least preferable VCCV option but it does work as a last resort, and is what CSR1000v routers use when the PW-ACH option (type 1) is not available.

A router sets the appropriate bit to acknowledge supporting a given option during PW negotiation when tLDP is used for signaling. Looking at the VC bindings, we see that our CSR PEs support all three options. The debug output shows VCCV CC 0x7 (0111) which is the result of setting all three flags. We also see VCCV CV 0x2 (0010) which claims support for the LSP ping, or LSPV, but not ICMP ping. The output below shows what we advertised (local) and what we received (remote), including labels and VCCV capabilities. Based on this, our VCCV tests should be using PW-ACH control world signaling inside the LSP ping.

```
R2#debug mpls l2transport vc vccv event
R2#debug mpls l2transport signaling event
R2#debug mpls l2transport signaling message
```

```
AToM LDP[153.5.5.5]: Sending label mapping msg vc type 5, cbit 1,
AToM LDP[153.5.5.5]:   , vc id 90,
AToM LDP[153.5.5.5]:   group id 0, vc label 2011,
AToM LDP[153.5.5.5]:   status 0x7/LDP 0x1/ADJ 0x0,
AToM LDP[153.5.5.5]:   mtu 1500, vc handle 0x80000001
AToM LDP[153.5.5.5]:   VCCV CC 0x7, CV 0x2
AToM LDP[153.5.5.5]: Received label mapping msg, id 77, GR instance 4 vc type
5, cbit 1,
AToM LDP[153.5.5.5]:   , vc id 90,
AToM LDP[153.5.5.5]:   group id 0, vc label 5007,
AToM LDP[153.5.5.5]:   status 0x7/LDP 0x1/ADJ 0x0,
AToM LDP[153.5.5.5]:   mtu 1500, vc handle 0x80000001
AToM LDP[153.5.5.5]:   VCCV CC 0x7, CV 0x2
```

We can also verify this with a show command. The legacy command is shown below; the newer alternative is “show l2vpn atom binding 90”. This book makes use for both methods for variety. In either case, the supported CC and CV capabilities are clearly shown so the administrator need not memorize the hexadecimal values.

```
R2#show mpls l2transport binding 90
Destination Address: 153.5.5.5, VC ID: 90
Local Label: 2011
  Cbit: 1,    VC Type: Ethernet,    GroupID: 0
  MTU: 1500,  Interface Desc: n/a
  VCCV: CC Type: CW [1], RA [2], TTL [3]
  CV Type: LSPV [2]
Remote Label: 5008
  Cbit: 1,    VC Type: Ethernet,    GroupID: 0
  MTU: 1500,  Interface Desc: n/a
  VCCV: CC Type: CW [1], RA [2], TTL [3]
  CV Type: LSPV [2]
```

Next, we examine an LSP ping sent over a pseudowire. From CSR2, I initiate the probe while using EPC on CSR1, and I use EXP1 so we can more easily identify the traffic in the EPC buffer. The output from CSR1 contains many components: a two label stack, the PW-ACH, and the IP packet. The MPLS encapsulation is 8 bytes and is highlighted in yellow. The label stack is as follows: { 1001/EXP1/TTL255 }, { 5008/EXP1/S-bit/TTL1 }. Grey represents the PW-ACH information; the first nibble is always 0001 in binary while the last 2 bytes of 0x0021 represents IPv4 (IPv6 would be 0x0057). In green, we see the familiar IP source/destination addresses of 153.2.2.2 and 127.0.0.1, just like with the L3VPN verifications. In pink is the familiar 4-byte router-alert which is always present on the echo-request as an IP option. Last, in cyan, is the first half of the UDP header showing the well-known LSP ping port number 3503. It is important to understand that while the CW is an MPLS L2VPN component and is 4 bytes long, it is not an MPLS shim header and never carries a label value.

```
R2#ping mpls pseudowire 153.5.5.5 90 exp 1

R1#show monitor capture CAP buffer detail
16 134    1.076992  00:50:56:A9:BE:8A -> 00:50:56:A9:1A:AA MPLS unicast
0000:  005056A9 1AAA0050 56A9BE8A 81000DB8  .PV....PV.....
0010:  8847003E 92FF0139 03011000 00214620  .G.>...9.....!F
0020:  00680570 40000111 C4EB9902 02027F00  .h.p@.....
0030:  00019404 00000DAF 0DAF0050 81900001  ....P....
```

For completeness, here is the response from CSR5. It’s nothing special, just a regular IP packet with a single MPLS label for transport (which technically doesn’t need to be there). The label 0x3E8 is 1000, which is what CSR1 advertised to CSR5 to reach 153.2.2.2/32. It appears that the default reply DSCP is CS6 (0xC0) in the IP header (green), which means EXP6 (0xC + 1 for S-bit = 0xD) in the shim header. Note that the default reply mode is IPv4, which means it not part of the PW LSP (no PW label, no CW, etc).

```
R1#show monitor capture CAP buffer detail
  5 118      0.165992 00:50:56:A9:DC:63 -> 00:50:56:A9:1A:AA MPLS unicast
0000: 005056A9 1AAA0050 56A9DC63 81000DBB .PV....PV..c....
0010: 8847003E 8DFF45C0 006011CB 0000FF11 .G.>...E..`.....
0020: 6FF79901 05059902 02020DAF 0DAF004C o.....L
0030: 3FFE0001 00000202 03015DF4 74B20000 ?.....].t...
```

```
R5#show ip cef 153.2.2.2
153.2.2.2/32
  nexthop 153.1.5.1 GigabitEthernet2.515 label 1000
```

Let's instruct CSR2 not to use control word (CW) on its end of the PW. The PW flaps, and when it comes back up, we see different bindings, using both debug and show commands on CSR2.

```
! CSR2
interface pseudowire90
  control-word exclude
```

Even though we disabled CW on CSR2, this means that CSR5 is not willing to use that method to interact with CSR2. The debug shows CSR2 advertising support for the CW while CSR5 is not, which seems backwards. It's odd how both CSR2 and CSR5 both say CW is supported locally, but not remotely, even though only CSR2 withdrew the CW support.

```
R2#debug mpls l2transport vc vccv event
R2#debug mpls l2transport signaling event
R2#debug mpls l2transport signaling message
```

```
AToM LDP[153.5.5.5]: Sending label mapping msg vc type 5, cbit 0,
AToM LDP[153.5.5.5]:   , vc id 90,
AToM LDP[153.5.5.5]:   group id 0, vc label 2008,
AToM LDP[153.5.5.5]:   status 0x7/LDP 0x1/ADJ 0x0,
AToM LDP[153.5.5.5]:   mtu 1500, vc handle 0x80000001
AToM LDP[153.5.5.5]:   VCCV CC 0x7, CV 0x2
```

```
AToM LDP[153.5.5.5]: Received label mapping msg, id 331, GR instance 3 vc
type 5, cbit 0,
AToM LDP[153.5.5.5]:   , vc id 90,
AToM LDP[153.5.5.5]:   group id 0, vc label 5010,
AToM LDP[153.5.5.5]:   status 0x7/LDP 0x1/ADJ 0x0,
AToM LDP[153.5.5.5]:   mtu 1500, vc handle 0x80000001
AToM LDP[153.5.5.5]:   VCCV CC 0x6, CV 0x2
```

```
R2#show l2vpn atom binding 90
  Destination Address: 153.5.5.5, VC ID: 90
    Local Label: 2008
      Cbit: 0,      VC Type: Ethernet,      GroupID: 0
      MTU: 1500,    Interface Desc: n/a
      VCCV: CC Type: CW [1], RA [2], TTL [3]
```

```

CV Type: LSPV [2]
Remote Label: 5010
  Cbit: 0, VC Type: Ethernet, GroupID: 0
  MTU: 1500, Interface Desc: n/a
  VCCV: CC Type: RA [2], TTL [3]
  CV Type: LSPV [2]

```

```

R5#show l2vpn atom binding 90
Destination Address: 153.2.2.2, VC ID: 90
Local Label: 5010
  Cbit: 0, VC Type: Ethernet, GroupID: 0
  MTU: 1500, Interface Desc: n/a
  VCCV: CC Type: CW [1], RA [2], TTL [3]
  CV Type: LSPV [2]
Remote Label: 2008
  Cbit: 0, VC Type: Ethernet, GroupID: 0
  MTU: 1500, Interface Desc: n/a
  VCCV: CC Type: RA [2], TTL [3]
  CV Type: LSPV [2]

```

We retest our LSP PW ping from CSR2 to CSR5, which succeeds. CSR2 uses CC type 3 (TTL expiry), as I do not see a router-alert label in the stack. It's the same packet as before except without the PW-ACH. The PW label has TTL=1 which forces CSR5 to examine it. Notice the PW label changed from 5008 to 5010 because when we removed CW support, the session flapped and a new label was allocated.

```
R2#ping mpls pseudowire 153.5.5.5 90 exp 1
```

```

R1#show monitor capture CAP buffer detail
 4 130 0.011001 00:50:56:A9:BE:8A -> 00:50:56:A9:1A:AA MPLS unicast
0000: 005056A9 1AAA0050 56A9BE8A 81000DB8 .PV....PV.....
0010: 8847003E 92FF0139 23014620 00680CAE .G.>...9#.F .h..
0020: 40000111 BDAD9902 02027F00 00019404 @.....
0030: 00000DAF 0DAF0050 1D780001 00000102 .....P.x.....

```

Again, for completeness, here is the reply from CSR5, using label 1000 to reach CSR2's loopback. This is a regular IPv4 packet that happened to be MPLS encapsulated along the return path due to IGP/LDP, but is totally unrelated to the PW being tested (no PW label). Before continuing, we add the CW to this PW.

```

R1#show monitor capture CAP buffer detail
 1 118 0.502964 00:50:56:A9:DC:63 -> 00:50:56:A9:1A:AA MPLS unicast
0000: 005056A9 1AAA0050 56A9DC63 81000DBB .PV....PV..c....
0010: 8847003E 8DFF45C0 006012D9 0000FF11 .G.>...E..`.....
0020: 6EE99901 05059902 02020DAF 0DAF004C n.....L
0030: 65B10001 00000202 03015DF4 74B70000 e.....].t...

```

```

R5#show mpls forwarding-table 153.2.2.2 32
Local      Outgoing  Prefix          Bytes Label  Outgoing  Next Hop

```

Label	Label	or Tunnel Id	Switched	interface	
5005	1000	153.2.2.2/32	16886	Gi2.515	153.1.5.1

Another useful debug command is shown below. For every LSP ping sent and received, a message is generated which clarifies the local and remote CW information. This is less destructive than flapping the PW and is a good addition to the show commands. The output below shows that all 3 VCCV CCs are supported; this makes sense since we added the CW back to this PW.

```
R2#debug l2vpn atom vc event
```

```
R2#ping mpls pseudowire 153.5.5.5 90 repeat 1
```

```
AToM[153.5.5.5, 90]: MSPW: LocCC CW [1], RA [2], TTL [3] Rem CW [1], RA [2], TTL [3]
AToM[153.5.5.5, 90]: MSPW: LocCC CW [1], RA [2], TTL [3] Rem CW [1], RA [2], TTL [3]
```

Pseudowire-based L2VPN does not appear to work in XRv at all, not even in the control plane.

```
! CSR5
interface pseudowire45
 encapsulation mpls
 neighbor 153.14.14.14 450
 vc type ethernet
 control-word exclude

! XRv4
l2vpn
 pw-class VPLS
 encapsulation mpls

xconnect group PW
 p2p CSR5
 interface GigabitEthernet0/0/0/0.998
 neighbor 153.5.5.5 pw-id 450
 pw-class VPLS
```

```
R5#show mpls l2transport vc 45 detail | include error
Last error: Remote peer access circuit is down
```

```
RP/0/0/CPU0:XRv4#show l2vpn xc pw-id 45 detail | include Error
Error: AToM Dynamic with mismatched AC config
```

However, static VPLS does appear to work in the control-plane on XRv, so we built a P2P VPLS connection from CSR5 to XRv4 to test XR VCCV. The PW never comes up because XRv always thinks there is a problem with its AC. I left the broken PW configuration there for reference in case it starts to work with newer code.

```
! CSR5
```



```

l2vpn vfi context XR_VCCV
  vpn id 45
  member pseudowire45

bridge-domain 450
  member GigabitEthernet2 service-instance 450
  member vfi XR_VCCV

! XRv4
l2vpn
  pw-class VPLS
  encapsulation mpls

bridge group XR_VCCV
  bridge-domain XR_VCCV
  interface GigabitEthernet0/0/0/0.998

  vfi XR_VCCV
    vpn-id 45
    neighbor 153.5.5.5 pw-id 450
    pw-class VPLS

```

VPLS makes a pseudowire behind the scenes, which basically the same as the P2P xconnect that failed, with the addition of a bridge-domain and MAC address learning.

```
R5#show l2vpn atom vc vcid 450
```

Interface	Peer ID	VC ID	Service Type	Service Name	Status
pw100003	153.14.14.14	450	vfi	XR_VCCV	UP

```
RP/0/0/CPU0:XRv4#show l2vpn bridge-domain pw-id 450 brief
```

Legend: pp = Partially Programmed.

Bridge Group:Bridge-Domain Name	ID	State	Num ACs/up	Num PWs/up
XR_VCCV:XR_VCCV	0	up	1/1	1/1

Next, we check XRv4 for its supporting VCCV capabilities. We can see the MPLS PW labels exchanged also. For completeness, we will examine CSR5's PW bindings also to see the MPLS label and VCCV information. We can see that the control-word VCCV capability was not negotiated; we will come back later and manually add the control word.

```
RP/0/0/CPU0:XRv4#show l2vpn bridge-domain pw-id 450 detail | begin List of VFI
```

List of VFIs:

VFI XR_VCCV (up)

PW: neighbor 153.5.5.5, PW ID 450, state is up (established)

PW class not set, XC ID 0xff000002

Encapsulation MPLS, protocol LDP
Source address 153.14.14.14
PW type Ethernet, control word disabled, interworking none
Sequencing not set

PW Status TLV in use		
MPLS	Local	Remote

Label	94007	5013
Group ID	0x0	0x0
Interface	XR_VCCV	unknown
MTU	1500	1500
Control word	disabled	disabled
PW type	Ethernet	Ethernet
VCCV CV type	0x2	0x2
	(LSP ping verification)	(LSP ping verification)
VCCV CC type	0x6	0x6
	(router alert label)	(router alert label)
	(TTL expiry)	(TTL expiry)

```
R5#show mpls l2transport binding 450
Destination Address: 153.14.14.14, VC ID: 450
Local Label: 5013
  Cbit: 0,    VC Type: Ethernet,    GroupID: 0
  MTU: 1500,  Interface Desc: n/a
  VCCV: CC Type: RA [2], TTL [3]
        CV Type: LSPV [2]
Remote Label: 94007
  Cbit: 0,    VC Type: Ethernet,    GroupID: 0
  MTU: 1500,  Interface Desc: XR_VCCV
  VCCV: CC Type: RA [2], TTL [3]
        CV Type: LSPV [2]
```

Let's test the PW and make sure it's working bidirectionally. No issues here.

```
RP/0/0/CPU0:XRv4#ping mpls pseudowire 153.5.5.5 450
[snip]
Type escape sequence to abort.
!!!!!!
Success rate is 100 percent [snip]
```

```
R5#ping mpls pseudowire 153.14.14.14 450
[snip]
Type escape sequence to abort.
!!!!!!
Success rate is 100 percent [snip]
```

We inspect the EPC dump of these LSPV packets to determine whether they are using the router-alert label or not, since it's the only VCCV CC mechanism we have not observed. I will shutdown the link between XRv3 and CSR5 temporarily so CSR1 is always in the transit path. Again, I use EXP1 to help differentiate this traffic among everything else collected by CSR1. We see the CSR5 using the CC type-3 TTL-expiry method because the TTL on the PW label is 1 with no router-alert label or PW-ACH present.

```
R5#ping mpls pseudowire 153.14.14.14 450 exp 1
```

```
R1#show monitor capture CAP5 buffer detail
  2  130      0.225986  00:50:56:A9:DC:63 -> 00:50:56:A9:1A:AA MPLS unicast
0000:  005056A9 1AAA0050 56A9DC63 81000DBB  .PV....PV..c....
0010:  8847003E D2FF16F3 73014620 00682CC0  .G.>....s.F .h,.
0020:  40000111 9A959905 05057F00 00019404  @.....
0030:  00000DAF 0DAF0050 4AE40001 00000102  .....PJ.....
```

We will test the LSP ping from XRv4 to see if it follows the RFC guidance and uses the router-alert over the TTL-expiry option. The output below proves that it does, in fact, use CC type-2 when the PW-ACH isn't available. The original label stack is shown in yellow while the router-alert label is shown in cyan. When the router obeys the RA label and looks at the underlying IP packet, it finds the familiar IP router-alert option and UDP destination port 3503, which indicates an LSPV message. Note that the TTL of the PW label is 255 which would never trigger type-3 VCCV.

```
RP/0/0/CPU0:XRv4#ping mpls pseudowire 153.5.5.5 450 exp 1
```

```
R1#show monitor capture CAP13 buffer detail
  2  134      0.169989  00:50:56:A9:DB:37 -> 00:50:56:A9:1A:AA MPLS unicast
0000:  005056A9 1AAA0050 56A9DB37 81000DB9  .PV....PV..7....
0010:  8847003E 92FE0000 12FF0139 53FF4620  .G.>.....9S.F
0020:  00680369 40000111 BADA990E 0E0E7F00  .h.i@.....
0030:  00019404 00000DAF 0DAF0050 BF9D0001  .....P....
```

I cannot find a way to allow XE to offer type-1 CC switching to XRv. I enable the control word successfully on both platforms as shown below.

```
! CSR5
interface pseudowire45
 control-word include

! XRv4
l2vpn
 pw-class VPLS
 encapsulation mpls
 control-word
```

The result is the same. The CW was successfully negotiated and could be used for non-VCCV functions such as sequencing. Both routers claim they have CW enabled, but only XR agrees to use PW-ACH for

VCCV. I assume this is a data-plane-related limitation of establishing a PW between the CSR1000v and XRv. We do not investigate it further.

```
RP/0/0/CPU0:XRv4#show l2vpn bridge-domain pw-id 450 detail | begin List of VFI
```

List of VFIs:

[snip]

PW Status TLV in use		
MPLS	Local	Remote
-----	-----	-----
Label	94007	5008
Group ID	0x0	0x0
Interface	XR_VCCV	unknown
MTU	1500	1500
Control word	enabled	enabled
PW type	Ethernet	Ethernet
VCCV CV type	0x2	0x2
	(LSP ping verification)	(LSP ping verification)
VCCV CC type	0x7	0x6
	(control word)	
	(router alert label)	(router alert label)
	(TTL expiry)	(TTL expiry)
-----	-----	-----

```
R5#show mpls l2transport binding 450
Destination Address: 153.14.14.14,VC ID: 450
Local Label: 5008
  Cbit: 1,    VC Type: Ethernet,    GroupID: 0
  MTU: 1500,  Interface Desc: n/a
  VCCV: CC Type: RA [2], TTL [3]
    CV Type: LSPV [2]
Remote Label: 94007
  Cbit: 1,    VC Type: Ethernet,    GroupID: 0
  MTU: 1500,  Interface Desc: XR_VCCV
  VCCV: CC Type: CW [1], RA [2], TTL [3]
    CV Type: LSPV [2]
```

```
R5#show mpls l2transport vc 450 detail | include Control
Control Word: On (configured: autosense)
```

Let's quickly retest in both directions to see what kind of VCCV CC mechanisms the routers use now that there is a disagreement about the best method. First, we will ping from CSR5 to XRv4. CSR5 used the PW-ACH control word (type-1 method) because XRv4 said it was capable of supporting it. The MPLS label stack is highlighted in yellow with the CW in cyan. In this way, it appears like the VCCV CC methods can be asymmetric; provided the CW is negotiated, CSR5 can use the PW-ACH for OAM but only in the transmitting direction.

```
R5#ping mpls pseudowire 153.14.14.14 450 exp 1
```

```

R1#show monitor capture CAP5 buffer detail
  1  134      0.713012  00:50:56:A9:DC:63 -> 00:50:56:A9:1A:AA MPLS unicast
0000:  005056A9 1AAA0050 56A9DC63 81000DBB  .PV....PV..c....
0010:  8847003E D2FF16F3 73011000 00214620  .G.>....s....!F
0020:  006831F7 40000111 955E9905 05057F00  .h1.@....^.....
0030:  00019404 00000DAF 0DAF0050 8FC40001  .....P....

```

The next test involves testing the LSP in the reverse direction from XRv4 to CSR5. The ping totally fails. I theorize that XR is trying to play “by the rules” and send packets with a router-alert label, but CSR5 is not accepting it because it secretly doesn’t support it.

```

RP/0/0/CPU0:XRv4#ping mpls pseudowire 153.5.5.5 450 source 153.14.14.14 exp 1
[snip]
.....

```

XR has a great feature where you can actually pick which VCCV CC method you want to use. It’s tagged as “cisco-support” but I believe it’s useful for ordinary troubleshooting. By default, the router picks the “best” CC method available, which in this case is type-2 RA label method. If we quickly iterate over the other options, we see that type-1 CW does work, despite the CSR not advertising that capability in VCCV correctly. We also see that type-3 TTL-Expiry works as expected, but type-2 RA label does not. I am assuming the CSR simply does not support RA when it expects PW-ACH or TTL-Expiry to be used. We know the RA label worked before when CW was totally disabled and not negotiated on either end, assuming XRv4 initiates the LSP ping. It seems CSR5 did not clearly communicate its support for PW-ACH during the PW setup or else we wouldn’t have this problem.

```

RP/0/0/CPU0:XRv4#ping mpls pseudowire 153.5.5.5 450 source 153.14.14.14 exp 1
force-control-channel control-word
[snip]
!!!!!!

```

```

RP/0/0/CPU0:XRv4#ping mpls pseudowire 153.5.5.5 450 source 153.14.14.14 exp 1
force-control-channel ttl-expiry
[snip]
!!!!!!

```

```

RP/0/0/CPU0:XRv4#ping mpls pseudowire 153.5.5.5 450 source 153.14.14.14 exp 1
force-control-channel ra-label
[snip]
.....

```

As a final test, we disable the CW from both sides to retest CC type-1 CW from XRv4 to see how CSR5 reacts. CW removal is not shown for brevity. A quick verification on CSR5 indicates CW is not a supported VCCV CC at either end. As expected, this fails, but both RA-label and TTL-expiry work.

```

R5#show mpls l2transport binding 450 | include VCCV:

```

```

VCCV: CC Type: RA [2], TTL [3]
VCCV: CC Type: RA [2], TTL [3]

RP/0/0/CPU0:XRv4#ping mpls pseudowire 153.5.5.5 450 source 153.14.14.14 exp 1
force-control-channel control-word
[snip]
.....

RP/0/0/CPU0:XRv4#ping mpls pseudowire 153.5.5.5 450 source 153.14.14.14 exp 1
force-control-channel ttl-expiry
[snip]
!!!!!!

RP/0/0/CPU0:XRv4#ping mpls pseudowire 153.5.5.5 450 source 153.14.14.14 exp 1
force-control-channel ra-label
[snip]
!!!!!!

```

Because this section continues into the next one using the same topology, the configurations are included only at the end of the next section.

48.2 MPLS LSP Monitor (MPLSLM) / LSP Health Monitor

Earlier we discussed IP SLA probes as tools to actively monitor network devices using specific ports, protocols, encapsulations, etc. Based on the last test with MPLS LSP load-sharing and testing each path, you can manually define some IP SLA probes to test the LSP health between PEs. Since IP SLA supports several MPLS probes, this is a good way to test LSPs to ensure traffic doesn't get blackholed or looped. The downside is that it is configuration intensive, prone to human error, and scales poorly. **MPLSLM is a feature used to automatically discover MPLS L3VPN PE devices and probe them via IP SLA.** This is tricky on XE because the manner in which things are configured (auto ip sla) is different than how they are verified (show ip sla auto). Older versions had more straightforward configuration syntax (ip sla auto).

The first step in configuring the "LSP Health Monitor", as it is sometimes called, is enabling MPLS L3VPN PE discovery. This is step simple; simply turn on the feature and tell the MPLS process how frequently to refresh its auto-discovered PEs using the BGP next-hop values. The default timer for scanning these new PEs is 300 (5 minutes) seconds, but we reduce it to 15 for faster discovery.

```

! CSR2
mpls discovery vpn interval 15
mpls discovery vpn next-hop

```

We can verify with some simple show commands. The command specifically shows the refresh interval and a list of BGP next-hops. Each next-hop specifies which VRF's are locally configured on that PE. In this case, only VRF OAM is in play. The router successfully discovered both CSR5 and XRv4 as remote PEs without adding any discovery traffic to the network or reconfiguring other MPLS LSRs.

```

R2#show mpls discovery vpn
Refresh interval set to 15 seconds.
Next refresh in 8 seconds

Next hop 153.5.5.5 (Prefix: 153.5.5.5/32)
      in use by: OAM

Next hop 153.14.14.14 (Prefix: 153.14.14.14/32)
      in use by: OAM

```

Having learned these remote VPN next-hops, the router can automatically spawn IP SLA probes to test LSP reachability to them. The probes can be LSP ping (echo) or traceroute (pathEcho) probes. I've configured an automatic probe [ID 10] which is effectively a template for all the BGP next-hops discovered thus far. The template has an ACL applied which specifies a list of PEs that we are allowed to probe. This might be useful to ensure you don't probe too many (or rogue) PEs in the network. A numbered, standard ACL must be used, and you must define the list before applying it.

```

! CSR2
access-list 45 permit 153.14.14.14
access-list 45 permit 153.5.5.5

auto ip sla mpls-lsp-monitor 10
  type echo ipsla-vrf-all
  access-list 45

```

If you apply the ACL before defining it, you are presented with the following error.

```

R2(config-auto-ip-sla-mpls-params)#access-list 45
%Access List 45 doesn't exist. There may be inconsistency in the filtering
due to this.

```

I've also defined the "secondary frequency" value. This is used only when a failure condition occurs (timeout or connection-loss) to more aggressively probe an LSP. When we schedule the probe later, we will use a frequency of 120 seconds. We specify secondary frequencies of 45 for connection-loss and timeout conditions, and this value should always be less (faster) than or equal to the schedule frequency. The default frequency and secondary frequency time values are 60 seconds. There are 3 options for configuring the secondary frequency and are listed below. There is no ability to set different secondary frequency values for connection-loss and timeout concurrently. We use the first method.

1. Specify secondary frequency for both timeout and connection-loss ("both") using a single value
2. Specify secondary frequency for timeout only
3. Specify secondary frequency for connection-loss only

```

! CSR2
auto ip sla mpls-lsp-monitor 10
  secondary-frequency both 45

```

When we manually originated MPLS pings, we specified the “destination” as a mechanism for load-sharing by selecting different local-host addresses (127.0.0.0/8). Within the context of IP SLA, this is the LSP-selector attribute. By specifying a value, we can determine which path our upstream LSRs take to reach the target FEC (the target is determined automatically). Earlier, we saw that using destinations 127.0.0.[2367] would send traffic through XRv3 directly rather than through CSR5, which was CSR2’s default choice. We will force traffic through XRv3 for this health monitor using LSP-selector 127.0.0.3.

```
! CSR2
auto ip sla mpls-lsp-monitor 10
  lsp-selector 127.0.0.3
```

Next, we will schedule the MPLSLM probe just like an ordinary IP SLA probe, except we prefix the command with “auto” as shown below. In the IP SLA discussion, we discussed group scheduling and its associated timers. Because this automatic template is essentially a group of probes, we must use group scheduling timers. The schedule-period is the timer by which probes are evenly started. With a schedule period of 30 seconds, one LSP probe will happen at t=0, and another at t=15, assuming there are 2 probes (implying 2 remote PEs). If I introduced a third PE and added it to ACL 45, the probes would start at t=0, t=10, and t=20. They are always spaced evenly when not using LSP path discovery (LPD) which is discussed later. We specified the path we wanted to take with the LSP-selector for this example. The probe process runs every 120 seconds (2 minutes).

```
! CSR2
auto ip sla mpls-lsp-monitor schedule 10 schedule-period 30 frequency 120
start-time pending
```

First, we will verify the configuration. We can check our “neighbors” which is effectively a list of PEs after being processed by the ACL. We also see the automatically assigned IP SLA probe IDs, along with the original template probe of 10. This command is different than the MPLS VPN discovery command since it shows that probes were actually spawned.

```
R2#show ip sla mpls-lsp-monitor neighbors
IP SLA MPLS LSP Monitor Database : 10
BGP Next hop 153.5.5.5 (Prefix: 153.5.5.5/32)
  ProbeID: 100007 (OAM)
BGP Next hop 153.14.14.14 (Prefix: 153.14.14.14/32)
  ProbeID: 100008 (OAM)
```

We notice that the probes are not working when we look at the basic IP SLA statistics for each probe. When we enable LSPV debugging on CSR2 as we did earlier when examining the router-alert, we notice an interesting IPv4 source address for the probe. The probe is using the transit link between CSR2 and CSR1 as its source address (153.1.2.2), shown in yellow. The destination LSP-selector is shown in green for completeness (127.0.0.3). There does not appear to be a way to set the source-address within the

auto IP SLA template for MPLSLM. I consider this a significant issue since OSPF and IS-IS prefix-suppression techniques for IPv4 are quite common in MPLS core networks.

```
R2#show ip sla statistics 100007 | include Number
Number of successes: 0
Number of failures: 8
```

```
R2#show ip sla statistics 100008 | include Number
Number of successes: 0
Number of failures: 8
```

```
R2#debug mpls lspv packet data
LSPV: Echo Request to be sent on IPv4 LSP, pathindex 0, size 100,
datagramsize 112
46 00 00 70 C0 89 40 00 01 11 09 E9 99 01 02 02
7F 00 00 03 94 04 00 00 0D AF 0D AF 00 58 C0 00
[snip]
```

For the purposes of testing this feature, I allow CSR2 to advertise its connected interfaces as IP prefixes inside its IS-IS link-state packet (not shown). XRv4 will have a route for 153.1.2.0/24 via XRv3 now. Restarting the auto IP SLA scheduler, we receive a new set of probe IDs and will check the statistics specifically for those. The reason one probe has 3 successes and the other has 2 is because of the schedule-period; they are spaced 15 seconds apart.

```
R2#show ip sla mpls-lsp-monitor neighbors
IP SLA MPLS LSP Monitor Database : 10
BGP Next hop 153.5.5.5 (Prefix: 153.5.5.5/32)
  ProbeID: 100009 (OAM)
BGP Next hop 153.14.14.14 (Prefix: 153.14.14.14/32)
  ProbeID: 100010 (OAM)
```

```
R2#show ip sla statistics 100009 | include Number
Number of successes: 3
Number of failures: 0
```

```
R2#show ip sla statistics 100010 | include Number
Number of successes: 2
Number of failures: 0
```

In Cisco's terms, this technique is considered "MPLSLM without LSP Discovery". The reason is because although we discovered the BGP next-hops, we manually selected the LSP inside the probe template using the LSP-selector option. In other words, we are not testing all possible ECMP paths and thus do not have a full picture of the MPLS forwarding health between CSR1 and XRv4. This new discovery process is detailed in Cisco's documentation. Below is a summary of the process:

1. We first discover the PEs via MPLS general discovery, which is a common step.

2. We build a discovery group per PE, which contains a list of LSPs to that PE. The number of paths will be 1, at worst, if there is no load-sharing along any path to that PE.
3. According to some algorithm, the process begins sending MPLS pings with varying destination addresses (127.0.0.1, 127.0.0.5, etc) to discover the ECMPs. Once the algorithm completes, the number of unique LSPs per PE will be known, along with their LSP-selectors. Assume that this number is N.
4. The IP SLA process will send N LSP pings with the recently computed LSP-selectors/outgoing interfaces as part of a single IP SLA superoperation. This means that the number of IP SLA instances grows proportionally with the number of PEs, not with the number of ECMPs. The single operation sends multiple probes and tracks them within the discovery group.

The LSP discovery process only works for L3VPN and only supports MPLS ping. MPLS traceroute and VCCV operations are not supported when using LSP discovery.

We will create a new auto IP SLA [ID 20] that can discover all the LSPs. Notice that we have no option to set the LSP-selector under this sub-configuration as it contravenes the purpose of LSP path discovery. Because the discovery group is an aggregated collection of sorts, we have some familiar IP SLA history options available, such as hours-of-statistics-kept. Since an MPLS network could be very large, we can constrain the “tree traces” (discovery process). The “scan-period” is an upper-bound on how long this process should take, measured in minutes. Tree traces are performed regularly when this interval expires. “Maximum-sessions” is a number that constrains how many tree trace requests (there is one per discovered LSP) can happen concurrently, and “session-timeout” specifies how long each request should take. You can also start the discovery process from any arbitrary local-host address using “lsp-selector-base”, with the default being 127.0.0.0. Of greatest significance is the scan-period which is set to 3 minutes, and we will see this in the debugs. “Timeout” and “interval” are not part of the discovery process and represent ordinary MPLS ping attributes already discussed.

```
! CSR2
auto ip sla mpls-lsp-monitor 20
  type echo ipsla-vrf-all
  path-discover
  maximum-sessions 4
  scan-period 3
  session-timeout 1
  interval 250

auto ip sla mpls-lsp-monitor schedule 20 schedule-period 30 frequency 120
start-time pending
```

When we schedule the probe, it is valuable to enable debugs to watch the discovery process. The first chunk of output is generalized information for this LSP monitor and is not specific to the LSP discovery process or LPD groups. It simply tells us that two PEs were discovered, probes were spawned, LPD groups were created around those probes (one group per PE), and tree traces were scheduled.

```
R2#debug ip sla trace mpls-lsp-monitor 20
```

```
IPSLA-AUTOMPLS_TRACE:OPER:20 Next hop 153.5.5.5 added (1)
IPSLA-AUTOMPLS_TRACE:OPER:20 Adding ProbeID 100011 to tree entry 153.5.5.5
(2)
IPSLA-AUTOMPLS_TRACE:OPER:20 Next hop 153.14.14.14 added (1)
IPSLA-AUTOMPLS_TRACE:OPER:20 Adding ProbeID 100012 to tree entry 153.14.14.14
(2)
IPSLA-AUTOMPLS_TRACE:OPER:20 probe list 100011-100012
IPSLA-AUTOMPLS_TRACE:OPER:20 Lpd Group List 100011-100012
```

The tree trace process begins for 153.5.5.5 (CSR5). We know there is only one path to CSR5 using label 1001 (I verify this by checking the LDP bindings also). We also see some of our limits set, such as maximum-sessions set to 4. After the algorithm finds the path (the output is unintelligible), the path is displayed along with the LSP-selector discovered by the path. The path is stored, the “found-paths” field is incremented, and the tree trace completes.

```
! CSR2
IPSLA-AUTOMPLS_TRACE:OPER:20 Starting LPD Tree Trace for (153.5.5.5)
IPSLA-AUTOMPLS_TRACE:OPER:20 Running TT is 1 Limit is 4, Last BGP Hop is
153.5.5.5
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Group(100011): Starting Path Discovery
Session
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Group(100011): Target (153.5.5.5/32)
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Group(100011): Client Handle 140553756147256
sent
IPSLA-AUTOMPLS_TRACE:OPER:20 Process Status Event
```

[snip, detailed debugs are beyond the scope, algorithm finds the only path]

```
IPSLA-AUTOMPLS_TRACE:OPER:20 Path 0 is OK
IPSLA-AUTOMPLS_TRACE:OPER:20 output-interface Gi2.512 source 153.1.2.2
destination 127.0.0.0
IPSLA-AUTOMPLS_TRACE:OPER:20 [Labels: 1001 Exp: 0 ]
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Group(100011): Store this path
IPSLA-AUTOMPLS_TRACE:OPER:20 Process Status Event
IPSLA-AUTOMPLS_TRACE:OPER:20 Paths (found/broken/unexplored) (1/0/0)
IPSLA-AUTOMPLS_TRACE:OPER:20 Echo Request (sent/fail) (2/0)
IPSLA-AUTOMPLS_TRACE:OPER:20 Echo Reply (received/timeout) (2/0)
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Group(100011): Path Discovery Status - 1
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Group(100011): Path Discovery Completed
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Group(100011): Updating LPD Stats
```

```
R2#show mpls ldp bindings 153.5.5.5 32 neighbor 153.1.1.1
  lib entry: 153.5.5.5/32, rev 10
    remote binding: lsr: 153.1.1.1:0, label: 1001
```

Next, the tree-trace (TT) begins for XRv4. The initial output is similar to the TT for CSR5, except now that “Running TT” is set to 2; these two traces ran concurrently but I sorted the debugs by TT for readability. We can only have a maximum of 4 traces at once per the configuration; this is a way to throttle the TT process so it does not hog CPU/memory resources. The traces may take longer with these controls but it ensures the router doesn’t bog down. The algorithm runs twice to find both paths, since CSR1 can ECMP via CSR5 or XRv3 to reach XRv4. The downstream label from CSR2’s perspective is 1002 for both paths since CSR1 is the next-hop for both. I verify this as I did earlier using the LDP LIB. Since the next-hop uses the same label, the LSP-selector uses two different destination IPv4 addresses to force traffic down both paths. We see 127.0.0.0 and 127.0.0.2 used for paths 0 and 1, respectively.

```
! CSR2
IPSLA-AUTOMPLS_TRACE:OPER:20 Starting LPD Tree Trace for (153.14.14.14)
IPSLA-AUTOMPLS_TRACE:OPER:20 Running TT is 2 Limit is 4, Last BGP Hop is
153.14.14.14
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Group(100012): Starting Path Discovery
Session
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Group(100012): Target (153.14.14.14/32)
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Group(100012): Client Handle 140553783514944
sent
IPSLA-AUTOMPLS_TRACE:OPER:20 Process Status Event
```

[snip, detailed debugs are beyond the scope, algorithm finds the first path]

```
IPSLA-AUTOMPLS_TRACE:OPER:20 Path 0 is OK
IPSLA-AUTOMPLS_TRACE:OPER:20 output-interface Gi2.512 source 153.1.2.2
destination 127.0.0.0
IPSLA-AUTOMPLS_TRACE:OPER:20 [Labels: 1002 Exp: 0 ]
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Group(100012): Store this path
```

[snip, detailed debugs are beyond the scope, algorithm finds the next path]

```
IPSLA-AUTOMPLS_TRACE:OPER:20 Path 1 is OK
IPSLA-AUTOMPLS_TRACE:OPER:20 output-interface Gi2.512 source 153.1.2.2
destination 127.0.0.2
IPSLA-AUTOMPLS_TRACE:OPER:20 [Labels: 1002 Exp: 0 ]
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Group(100012): Store this path
IPSLA-AUTOMPLS_TRACE:OPER:20 Process Status Event
IPSLA-AUTOMPLS_TRACE:OPER:20 Paths (found/broken/unexplored) (2/0/0)
IPSLA-AUTOMPLS_TRACE:OPER:20 Echo Request (sent/fail) (6/0)
IPSLA-AUTOMPLS_TRACE:OPER:20 Echo Reply (received/timeout) (6/0)
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Group(100012): Path Discovery Status - 1
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Group(100012): Path Discovery Completed
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Group(100012): Updating LPD Stats
```

```
R2#show mpls ldp bindings 153.14.14.14 32 neighbor 153.1.1.1
  lib entry: 153.14.14.14/32, rev 14
    remote binding: lsr: 153.1.1.1:0, label: 1002
```

Last, the LPD discovery process terminates and reschedules itself for ~180 seconds (3 minutes) from now. This is the scan-period time minus the time it took for LPD discovery to run (70 ms or so).

```
! CSR2
IPSLA-AUTOMPLS_TRACE:OPER:20 Completed the LPD Discovery
IPSLA-AUTOMPLS_TRACE:OPER:20 Starting the rediscovery, timer expiry after
179931 (ms)
```

Many new show commands are available to view the status of this feature and its discovered LSPs. I will display them in sequence. The first command isn't terribly useful as it only reports the IP SLA superoperation ID. There is one per PE, not per LSP, and if the probe needs to test multiple LSPs, it sends multiple echo requests out under the umbrella of the IP SLA superoperation. Also, the ordinary IP SLA show commands no longer report a counter of success and failure, but only the most recent return code.

```
R2#show ip sla mpls-lsp-monitor collection-statistics
Entry number: 100011
Entry number: 100012
```

```
R2#show ip sla statistics 100011 | include return_code
Latest operation return code: OK
```

```
R2#show ip sla statistics 100012 | include return_code
Latest operation return code: OK
```

Examining the operational state of these probes is more valuable. Knowing that ID 100012 (to XRv4) has two paths and 100011 (to CSR5) only has one, we will examine the most interesting 100012 probe. The preamble is general information about the probe, along with the parent MPLSLM template number and statistical information. Of greater interest is the path list. We see two paths that appear identical at a glance; the only differentiator is the LSP-selector. The debugs indicated one path used a destination of 127.0.0.0 while the other used 127.0.0.2. The output is confusing and requires explanation. To see what the local-host destination is, convert the decimal number to hex, add in the dots, and convert it back as 4 separate bytes. For example 2130706434 is our second LSP and converts to 0x7F000002 in hex. Adding dots yields 7F.00.00.02, and converting to decimal per-byte yields 127.0.0.2. The value 2130706432 is 0x7F000000 in hex, which is 127.0.0.0 without the dots. Just displaying the local-host IPv4 address in the first place would have been much more convenient.

```
R2#show ip sla mpls-lsp-monitor lpd operational-state 100012
Entry number: 100012
MPLSLM Entry Number: 20
Target FEC Type: LDP IPv4 prefix
Target Address: 153.14.14.14
Number of Statistic Hours Kept: 2
Last time LPD Stats were reset: .17:55:08.634 UTC [snip]
Traps Type: 1
```

```

Latest Path Discovery Mode: rediscovery complete
Latest Path Discovery Start Time: .14:55:05.002 UTC [snip]
Latest Path Discovery Return Code: OK
Latest Path Discovery Completion Time(ms): 81
Number of Paths Discovered: 2

```

Path Information :

Path Index	Outgoing Interface	Lsp Selector	Link Type	Conn Id	Adj Addr	NextHop Addr	Downstream Label Stack	Status
1	Gi2.512	2130706432	90	0	153.1.2.1	153.1.2.1	1002	OK
2	Gi2.512	2130706434	90	0	153.1.2.1	153.1.2.1	1002	OK

Earlier, we used the neighbors command when testing MPLSLM without LSP discovery. Here, we see some extra information, including the “OK” path count. This is a quick verification to see if LSP discovery worked. It also shows the probe IDs so you can dig deeper if needed. LPD is smart in that it differentiates between “OK” paths and broken paths.

```

R2#show ip sla mpls-lsp-monitor neighbors
IP SLA MPLS LSP Monitor Database : 20
BGP Next hop 153.5.5.5 (Prefix: 153.5.5.5/32) OK Paths: 1
  ProbeID: 100011 (OAM)
BGP Next hop 153.14.14.14 (Prefix: 153.14.14.14/32) OK Paths: 2
  ProbeID: 100012 (OAM)

```

Another good command for quick verification is shown below. It includes a well-defined legend at the top as well. It will show one row per LPD group, specify its ID, and specify its status. Individual path-level detail is not revealed here.

```

R2#show ip sla mpls-lsp-monitor summary
Index          - MPLS LSP Monitor probe index
Destination    - Target IP address of the BGP next hop
Status         - LPD group status
LPD Group ID   - Unique index to identify the LPD group
Last Operation Time - Last time an operation was attempted by
                  a particular probe in the LPD Group

```

Index	Destination	Status	LPD Group ID	Last Operation Time
20	153.5.5.5	up	100011	.15:03:30.002 UTC [snip]
20	153.14.14.14	up	100012	.15:03:15.001 UTC [snip]

Next, we simulate a failure on the link between CSR1 and CSR5. I will configure CSR5 not to advertise a label for 153.14.14.14/32, which means the path will be unlabeled from CSR1’s perspective. It is still an ECMP according to IGP, though. The LSP to 153.14.14.14/32 still has a path through XRv13, so the LPD group is considered “partially” up if at least one component LSP is operational. We expect no change to LPD group 100011 (CSR5) since we did not perform any filtering against the LDP binding for 153.5.5.5/32. Before applying the filter to CSR5, I first enable IP SLA debugs on CSR2 again so we can see the failures. After we apply the filter, let’s verify it worked by checking CSR1’s LFIB. The tree trace process only happens every 3 minutes, so we might have to wait a bit.

```
! CSR5
no mpls ldp advertise-labels
mpls ldp advertise-labels for ACL_NO_XRV4_LABEL
```

```
R1#show mpls forwarding-table 153.14.14.14 32
```

Local Label	Outgoing Label	Prefix or Tunnel Id	Bytes Switched	Outgoing interface	Next Hop
1002	93000	153.14.14.14/32	27140	Gi2.513	153.1.13.13
	No Label	153.14.14.14/32	0	Gi2.515	153.1.5.5

The debug information is below. First, we quickly acknowledge that the path to 153.5.5.5/32 is unchanged. There was only every 1 path to CSR5 and this is still functional.

```
! CSR2
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Rediscovery interval timer expired

IPSLA-AUTOMPLS_TRACE:OPER:20 Starting LPD Tree Trace for (153.5.5.5)
IPSLA-AUTOMPLS_TRACE:OPER:20 Running TT is 1 Limit is 4, Last BGP Hop is
153.5.5.5
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Group(100011): Starting Path Discovery
Session
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Group(100011): Target (153.5.5.5/32)
```

[snip]

```
IPSLA-AUTOMPLS_TRACE:OPER:20 Path 0 is OK
IPSLA-AUTOMPLS_TRACE:OPER:20 output-interface Gi2.512 source 153.1.2.2
destination 127.0.0.0
IPSLA-AUTOMPLS_TRACE:OPER:20 Path already exists, status is OK
```

Of greater interest is the path to XRv4 via CSR5. We can see that the TT algorithm detects the breakage and updates the MPLSLM process. The downstream label binding is still the same because CSR1 still allocates a label for 153.14.14.14/32. Just because a neighbor gives you a label binding for the target FEC does not mean the LSP is operational end-to-end. The first path via XRv3 is operational but the second path via CSR5 is not.

```
! CSR2
IPSLA-AUTOMPLS_TRACE:OPER:20 Starting LPD Tree Trace for (153.14.14.14)
IPSLA-AUTOMPLS_TRACE:OPER:20 Running TT is 2 Limit is 4, Last BGP Hop is
153.14.14.14
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Group(100012): Starting Path Discovery
Session
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Group(100012): Target (153.14.14.14/32)
```

[snip]

```
IPSLA-AUTOMPLS_TRACE:OPER:20 Path 0 is OK
IPSLA-AUTOMPLS_TRACE:OPER:20 output-interface Gi2.512 source 153.1.2.2
destination 127.0.0.2
IPSLA-AUTOMPLS_TRACE:OPER:20 Path already exists, status is OK
IPSLA-AUTOMPLS_TRACE:OPER:20 [Labels: 1002 Exp: 0 ]
```

[snip]

```
IPSLA-AUTOMPLS_TRACE:OPER:20 Path 1 is Broken
IPSLA-AUTOMPLS_TRACE:OPER:20 output-interface Gi2.512 source 153.1.2.2
destination 127.0.0.0
IPSLA-AUTOMPLS_TRACE:OPER:20 Path already exists, status is Broken
IPSLA-AUTOMPLS_TRACE:OPER:20 [Labels: 1002 Exp: 0 ]
IPSLA-AUTOMPLS_TRACE:OPER:20 Process Status Event
IPSLA-AUTOMPLS_TRACE:OPER:20 Paths (found/broken/unexplored) (1/1/0)
```

The following show commands can reveal the path breakage also. The first one is tricky because the path ID's returned in the debug output (0 and 1) do not correlate well with the indices here (1 and 2). Nonetheless, we can see an issue with one of the LSPs. The second command shows the status of the LPD group as "partial" because it still has at least one valid LSP to the destination in question. Last, the basic IP SLA show command still indicates the probe is OK overall. This is a dangerous condition because 50% of traffic could be blackholed as IGP still thinks using that path is acceptable. This is a good use-case for LDP/IGP synchronization, which was discussed in detail in the LDP section.

```
R2#show ip sla mpls-lsp-monitor lpd operational-state 100012 | begin Path_In
Path Information :
```

Path Index	Outgoing Interface	Lsp Selector	Link Type	Conn Id	Adj Addr	NextHop Addr	Downstream Label	Status
1	Gi2.512	2130706432	90	0	153.1.2.1	153.1.2.1	1002	Broken
2	Gi2.512	2130706434	90	0	153.1.2.1	153.1.2.1	1002	OK

```
R2#show ip sla mpls-lsp-monitor summary | begin Index.*Dest
Index Destination Status LPD Group ID Last Operation Time
20 153.5.5.5 up 100011 .15:27:30.002 UTC [snip]
20 153.14.14.14 partial 100012 .15:27:45.001 UTC [snip]
```

```
R2#show ip sla statistics 100012 | include return_code
Latest operation return code: OK
```

Let's completely disable LDP on CSR1 on the link facing CSR5 (not shown). This will break the LSP for 153.5.5.5/32 since IGP does not have any ECMPs for it. In other words, the LPD group for 153.5.5.5 only ever had one path. After disabling LDP, let's ensure the LFIB shows an unlabeled path to 153.5.5.5/32. Before that, we enable debugs on CSR2 and look for the breakage.

```
R1#show mpls forwarding-table 153.5.5.5 32
Local Outgoing Prefix Bytes Label Outgoing Next Hop
Label Label or Tunnel Id Switched interface
```



```
1001      No Label      153.5.5.5/32      49254      Gi2.515      153.1.5.5
```

```
! CSR2
```

```
IPSLA-AUTOMPLS_TRACE:OPER:20 LPD Rediscovery interval timer expired
```

```
IPSLA-AUTOMPLS_TRACE:OPER:20 Starting LPD Tree Trace for (153.5.5.5)
```

```
IPSLA-AUTOMPLS_TRACE:OPER:20 Running TT is 1 Limit is 4, Last BGP Hop is
```

```
153.5.5.5
```

```
[snip]
```

```
IPSLA-AUTOMPLS_TRACE:OPER:20 Path 0 is Broken
```

```
IPSLA-AUTOMPLS_TRACE:OPER:20 output-interface Gi2.512 source 153.1.2.2
```

```
destination 127.0.0.0
```

```
IPSLA-AUTOMPLS_TRACE:OPER:20 Path already exists, status is Broken
```

```
IPSLA-AUTOMPLS_TRACE:OPER:20 [Labels: 1001 Exp: 0 ]
```

```
IPSLA-AUTOMPLS_TRACE:OPER:20 Process Status Event
```

```
IPSLA-AUTOMPLS_TRACE:OPER:20 Paths (found/broken/unexplored) (0/1/0)
```

Using the same three show commands, we see that the LPD group contains one broken path, the overall group is considered down (not partial), and the general IP SLA probe is failing. Before continuing, we remove all filters and enable LDP on all core interfaces to restore baseline network connectivity.

```
R2#show ip sla mpls-lsp-monitor lpd operational-state 100011 | begin Path_In
```

```
Path Information :
```

Path Index	Outgoing Interface	Lsp Selector	Link Type	Conn Id	Adj Addr	NextHop Addr	Downstream Label Stack	Status
1	Gi2.512	2130706432	90	0	153.1.2.1	153.1.2.1	1001	Broken

```
R2#show ip sla mpls-lsp-monitor summary | begin Index.*Dest
```

Index	Destination	Status	LPD Group ID	Last Operation Time
20	153.5.5.5	down	100011	.15:39:00.003 UTC [snip]
20	153.14.14.14	partial	100012	.15:38:45.001 UTC [snip]

```
R2#show ip sla statistics 100011 | include return_code
```

```
Latest operation return code: No connection
```

The feature is similar in XR. We will attempt to demonstrate it on XRv4 to discover CSR2 and CSR5. First, we need to enable PE discovery based on BGP next hops. This is part of the “ipsla” tree unlike in XE, which is more consistent with the configuration logic. The remaining configuration is very similar to XE. One key note is that we need to identify the LSP-selector on XRv4. To do this, we query CSR5 for its DSMAP so we can force traffic a given way. In this example, we will force it through CSR1 directly. We can use the IPv4 destination address of 127.0.0.1 or 127.0.0.2 to accomplish this. We will use the latter.

```
RP/0/0/CPU0:XRv4# ping mpls ipv4 153.2.2.2/32 dsmap hash-key ipv4 bitmap 4  
ttl 1 repeat 1
```

```
L Echo Reply received from 153.13.14.13
```

```
DSMAP 0, DS Router Addr 153.1.13.1, DS Intf Addr 153.1.13.1
```

```

    Depth Limit 0, MRU 1500 [Labels: 1000 Exp: 0]
    Multipath Addresses:
        127.0.0.1          127.0.0.2
    DSMAP 1, DS Router Addr 153.5.13.5, DS Intf Addr 153.5.13.5
    Depth Limit 0, MRU 1500 [Labels: 5005 Exp: 0]
    Multipath Addresses:
        127.0.0.3          127.0.0.4

```

I've added some other options like exp-null and EXP3 for variety. Otherwise, the configuration is as close to CSR2 as it can reasonably be.

```

! XRv4
ipsla
mpls lsp-monitor
monitor 310
type mpls lsp ping
exp 3
force explicit-null
lsp selector ipv4 127.0.0.2

schedule monitor 310
frequency 30
schedule period 10
start-time now

mpls discovery vpn
interval 60

```

XRv MPLSLM does not work at all. The discovery process never finds a remote PE, no matter what.
 Ordinary MPLS L3VPN is working perfectly, and clearly XRv4 has the VPN routes with PE loopbacks.

```

RP/0/0/CPU0:XRv4#show ipsla mpls discovery vpn
Next refresh after: 35 seconds
BGP next hop      Prefix          VRF          PfxCount
[no output]

```

```

RP/0/0/CPU0:XRv4#show bgp vpnv4 unicast vrf OAM
[snip]
Route Distinguisher: 1:14 (default for vrf OAM)
*>i6.6.6.6/32      153.2.2.2          0      100      0 678 i
*> 7.7.7.7/32      153.7.14.7         0      100      0 678 i
*>i8.8.8.8/32      153.5.5.5          0      100      0 678 i

```

All of the IP SLA debugs are marked as “cisco-support” on IOS XR. Enabling the seemingly relevant debug prints a message every time the discovery interval expires, but that’s all it does. The router provides no indication of a fault nor any discovered PEs.

```

RP/0/0/CPU0:XRv4#debug ipsla sub-agent mpls discovery

```

```
ipsla_sa[260]: DISC: Refresh interval expired
```

For completeness, I configure a second monitor [ID 320] to mirror the LSP discovery monitor configured on CSR2. As expected, it does not work either, since the PE discovery process is faulty. I even specify the VRF OAM for this second probe, in case the documentation was mistaken. The documentation states that not configuring an explicit VRF means all VRFs are monitored.

```
! XRv4
monitor 320
  type mpls lsp ping
  vrf OAM
  path discover
  session limit 4
  session timeout 1
  scan period 3

schedule monitor 320
  frequency 120
  schedule period 30
  start-time now
```

Additional Reading – Reference configurations "mpls-oam"

48.3 Ethernet Management Tools (CFM, OAM, and E-LMI)

The advent of carrier Ethernet (Ethernet as a WAN technology) has brought forth the need for many new protocols to manage these networks. Other WAN technologies, like frame-relay, ATM, and SONET/SDH all had built-in support for link-monitoring. Ethernet was designed as a LAN protocol, so these new tools are add-ons to the existing Ethernet technology to make it suitable for WAN use. Ethernet OAM in general is a sizable topic and some of the features are limited to switches for obvious reasons. The core technology labs use three Cisco ME-3400-24TS-A switches on version 12.2-60.EZ8.

48.3.1 Connectivity Fault Management (CFM) (802.3ag)

CFM is a service-level OAM tool used in carrier Ethernet networks for proactive connectivity monitoring, fault detection, and fault isolation. The general idea is that by defining certain Ethernet interfaces as endpoints, we can create different hierarchical domains for maintenance. The definition of these specific points enables ping/traceroute tools, as well as other alarm reporting which is commonly not available on Ethernet LANs. CFM introduces many new terms, which are defined below.

Maintenance End Point (MEP): These are administratively-defined interfaces within an Ethernet network which define maintenance boundaries. These can be thought of like IP addresses where pings/traceroutes can be sourced from, along with other CFM messaging. There are two flavors of MEPs.

- a. **Inward or “up”:** These are endpoints configured on a switch to bound the internal domain but are configured at the “other end” of the switch. For example, if a switch has port 1 connected to the internal domain and port 2 connected to an external domain, an “UP MEP” would be

configured on port 2, because that port faces inwards “towards the switch” inside the same domain. This uses the “relay function” to send CFM messaging into the domain; it would not make sense to send internal CFM messaging out of port 2, since that goes to an external node.

- b. **Outward or “down”:** These are MEPs configured on port 2, but actually send CFM messages out of that wire-side interface, rather than internally using the “relay function”. This might be used on a CE switch which terminates the highest level CFM domain (levels are discussed later). These can also be used inside a carrier’s network at the port level for hop-by-hop maintenance.

Maintenance Intermediate Point (MIP): These are administratively-defined or auto-configured on intermediate switches. They exist primarily to forward CFM messaging back and forth, and to respond to traceroute probes from MEPs. MIPs allow the Ethernet network to be segmented so that faults can be isolated quickly, though MIPs are not tied to service-instances.

MEP and MIPs treat CFM messages that arrive from the wire-side based on the level in which those packets exist:

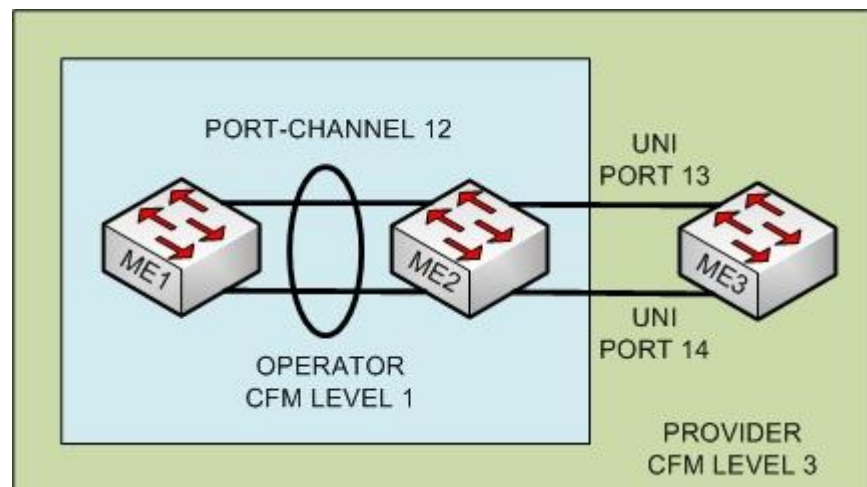
- a. Lower levels: CFM drops these packets
- b. Same level: CFM processes these packets
- c. Higher level: CFM forwards these packets without processing them (transparent). This is also true from packets arriving from the relay function side (inside).

There are 8 possible levels (0-7) that can be configured with CFM. The lower levels imply smaller networks. An example of a common CFM design would include operators, service providers, and customers. The operators are providing transit service over certain geographical areas, but perhaps not end-to-end connectivity. They are responsible for their own OAM and device management. An SP may subcontract multiple adjacent operators to enable connectivity for a customer, and would be interested to see faults between (but not within) those operator networks, since the sum of those operator networks is the SP network. The customer is the highest layer which would want to see alarms from the PE-CE links; although unable to see intra-SP or intra-operator errors, having connectivity between CE devices means the circuit is functional. Alternatively, issues in the customer network or SP network need not be revealed to operators, and adjacent operators need not see errors from one another.

There are 5 types of CFM messages. All of them are described in detail later, but the summaries are below.

- 1. **Continuity check message (CCM):** This is primarily used by CFM to discover remote MEPs and is essentially a hello packet. They can also carry alarm information as required.
- 2. **Loopback message (LBM):** The equivalent of an ICMP echo.
- 3. **Loopback reply (LBR):** The equivalent of an ICMP echo-reply.
- 4. **Link trace message (LTM):** The equivalent of a traceroute probe.
- 5. **Link trace reply (LTR):** The equivalent of a traceroute reply (TTL expiry or port unreachable).

The initial configurations are done on three hardware ME-3400-24TS-A switches arrayed in a line. ME1 and ME2 are operator bridges and ME3 is an SP-owned bridge. The entire domain is owned by the SP, and due to a lack of hardware, the notional customers are not CFM-aware (dummy ports used).



The basic NNI, LACP, and trunking configurations are not shown. Ports 15 and 16 are bonded on ME1 and ME2 using LACP, which CFM supports. The first thing to do is getting the operator network set up. This will include basic CFM configuration, such as enabling the feature, creating domains, creating service instances, and identifying MIPs/MEPs to support each service instance. On both switches, we enable the feature globally (can also do it per-interface) using IEEE 802.1ag versus the pre-standard Cisco version. I also enable logging to assist with troubleshooting. We define a CFM domain at level 1, which is a low level, and then create a service instance for VLAN 10. It is in this VLAN that the CCMs will be sent; since CFM is service-oriented, this session is specific to VLAN 10, which might be a customer transit VLAN. We enable the CCM messaging on this interface which has a default interval of 10 seconds and a default hold-time of 3 times that (30 seconds). Since the LACP bond faces inward, that is the MIP, which operates at level 1 and services a handful of test VLANs. Port 9 connects to an external entity, like another operator or customer, but it is an inward-facing (up) MEP so it sends its messaging through the switch, out towards the LACP bond. This MEP is specific to the OPERATOR domain and OP1 service instance.

```
! ME1
ethernet cfm ieee
ethernet cfm global
ethernet cfm logging
ethernet cfm domain OPERATOR level 1
  service OP1 vlan 10
  continuity-check

interface Port-channel12
  ethernet cfm mip level 1 vlan 10-19

interface FastEthernet0/9
```

```
ethernet cfm mep domain OPERATOR mpid 101 service OP1
```

The configuration on ME2 is essentially identical, except it has two MEPs towards ME3. I used different MPIDs for these so they are treated separately for CFM. The LACP bond is still a MIP, and the MEPs are still inward-facing (up). Note that both MEPs are part of the same service instance (OP1).

```
! ME2
ethernet cfm ieee
ethernet cfm global
ethernet cfm logging
ethernet cfm domain OPERATOR level 1
    service OP1 vlan 10
    continuity-check

interface Port-channel12
    ethernet cfm mip level 1 vlan 10-19

interface FastEthernet0/13
    ethernet cfm mep domain OPERATOR mpid 102 service OP1

interface FastEthernet0/14
    ethernet cfm mep domain OPERATOR mpid 112 service OP1
```

Next, we can verify the basic configuration on each switch locally, starting with ME1. It has one domain named OPERATOR at level 1 with one service configured. The service supports VLAN 10 as well as the continuity check (CC) messages at 10 second intervals. The maintenance association (MA) is OP1, which was our custom name. The direction is UP; don't be fooled, this isn't like line-protocol or state, this is just the "direction" of the CFM MEP. This doesn't mean things are working yet.

```
ME1#show ethernet cfm domain OPERATOR
Domain Name: OPERATOR
Level: 1
Total Services: 1
  Services:
  Type Id   Dir CC CC-int Static-rmep Crosscheck MaxMEP Source MA-Name
  Vlan 10  Up   Y  10s      Disabled   Disabled   100    Static  OP1
```

Next, we will verify the MPs we configured. This will show both MEPs and MIPs, but you can filter it down with variations of the command. The MEP is associated with the OP1 service, is configured on port 9 statically, supports CCMs, uses VLAN 10, and has an internal MAC of 001d.4692.f700 (discussed later). Notice the level is "Up", but this does not mean the MEPs are communicating yet. The MIP is less involved as it is not tied to a service instance, but shows the port, internal MAC, and VLAN supported.

```
ME1#show ethernet cfm maintenance-points local domain OPERATOR
Local MEPs:
```

```
-----
```

MPID	Domain Name	Lvl	MacAddress	Type	CC
Ofld	Domain Id	Dir	Port	Id	
	MA Name		SrvcInst	Source	
	EVC name				

```

101 OPERATOR 1 001d.4692.f700 Vlan Y
No OPERATOR Up Fa0/9 10
OP1 N/A Static
N/A

```

Total Local MEPs: 1

Local MIPs:
 * = MIP Manually Configured

Level	Port	MacAddress	SrvcInst	Type	Id	Source

```

*1 Po12 001d.4692.f700 N/A Vlan 10

```

Total Local MIPs: 1

The brain MAC is used for UP MEPs to communicate with the internal CFM level via the bridge relay function. We can reveal it by looking at the CFM statistics command.

```

ME1#show ethernet cfm statistics | include BRAIN
BRAIN MAC: 001d.4692.f700

```

Next, we can check the remote MEPs from ME1's perspective. This is similar to checking IGP neighbors to see if the protocol actually converged. On ME1, we see two remote MEPs. One of them represents ME2 port 13 (MEP 102) and one represents ME3 port 14 (MEP 112). Notice that both of the interface states are "Up", but one of the port states is "blocking". This is due to STP; ME3 is the root for all VLANs, and port 13 is ME2's root port. CFM can still monitor this link even though production traffic won't use it; in this way, CFM can still monitor all links in the network regardless of their STP state. The MAC address for both entries is the same because like ME1, ME2's MEP is inward facing (UP) so it uses the bridge-relay function with the brain MAC address. In summary, this output shows that the MEPs have "discovered" one another, and is one of the most valuable CFM show commands.

```

ME1#show ethernet cfm maintenance-points remote domain OPERATOR

```

MPID	Domain Name	MacAddress	IfSt	PtSt
Lvl	Domain ID	Ingress		
RDI	MA Name	Type Id	SrvcInst	
	EVC Name		Age	
	Local MEP Info			

```

112 OPERATOR 04c5.a453.d800 Up Blck
1 OPERATOR Po12
- OP1 Vlan 10 N/A
N/A 0s
MPID: 101 Domain: OPERATOR MA: OP1

```

102	OPERATOR	04c5.a453.d800	Up	Up
1	OPERATOR	Po12		
-	OP1	Vlan 10	N/A	
	N/A		3s	

MPID: 101 Domain: OPERATOR MA: OP1

We didn't configure the MEPs to find another statically, so we should explore the discovery mechanism. Starting with a basic packet debug on ME1, we can see the CCMs being sent to/received from ME2. ME1 only sends one CCM since it has only one MEP, yet ME2 sends two (one per MEP). They are spaced 10 seconds apart as expected. I use three different colors to show the different CCMs on the network. Since CCMs are critical to the maintenance of an Ethernet network, they are automatically marked with CoS 7.

```
ME1#debug ethernet cfm packets domain OPERATOR
Ethernet CFM level 1 domain OPERATOR service OP1 packet debugging is on

! ME1
08:24:17: CFM-PKT: Received a CC packet with MPID 102, level 1, vlan 10 from
interface Po12
08:24:21: CFM-PKT: Received a CC packet with MPID 112, level 1, vlan 10 from
interface Po12
08:24:27: CFM-PKT: Sending Up direction MEP 101 CC message, level 1, cos 7,
vlan 10
08:24:28: CFM-PKT: Received a CC packet with MPID 102, level 1, vlan 10 from
interface Po12
08:24:31: CFM-PKT: Received a CC packet with MPID 112, level 1, vlan 10 from
interface Po12
08:24:37: CFM-PKT: Sending Up direction MEP 101 CC message, level 1, cos 7,
vlan 10
```

We will configure local SPAN on ME2 so we can see traffic entering and exiting the LACP bond. This will give us more insight as to the CFM CCM frame format.

```
! ME2
monitor session 1 source interface Po12
monitor session 1 destination interface Fa0/23 encapsulation replicate
```

Unlike Cisco's REP, Wireshark recognizes 802.1ag as a standard and gives us some additional information. We can see that the source MACs are the brain MACs we saw earlier, and the destination MAC is a multicast MAC of 0180.2c00.0031. Being multicast, this is flooded to all ports in the VLAN, and this is how MEPs discover one another. Later, we will see that the last digit of the MAC address corresponds to the CFM level; in this case, the operator is using level 1. The ethertype is 0x8902 so we know this is a non-IP protocol custom-built for CFM. Looking inside the PDU, we can see the domain name, service name, MPID, and other information revealed from the debug.

No.	Time	Source	Destination	Protocol	Length	Info
	3.160611000	00:1d:46:92:f7:00	01:80:c2:00:00:31	CFM	105	Type Continuity Check Message (CCM)
	3.917815000	04:c5:a4:53:d8:00	01:80:c2:00:00:31	CFM	105	Type Continuity Check Message (CCM)
<						
+ Frame 31: 105 bytes on wire (840 bits), 105 bytes captured (840 bits) on interface 0						
- Ethernet II, Src: 00:1d:46:92:f7:00 (00:1d:46:92:f7:00), Dst: 01:80:c2:00:00:31 (01:80:c2:00:00:31)						
+ Destination: 01:80:c2:00:00:31 (01:80:c2:00:00:31)						
+ Source: 00:1d:46:92:f7:00 (00:1d:46:92:f7:00)						
Type: IEEE 802.1ag Connectivity Fault Management (CFM) protocol (0x8902)						
+ CFM EOAM 802.1ag/ITU Protocol, Type Continuity Check Message (CCM)						
- CFM CCM PDU						
+ Flags: 0x05						
First TLV offset: 70						
Sequence Number: 1299524336						
...0 0000 0110 0101 = Maintenance Association End Point Identifier: 101						
- Maintenance Association Identifier (MEG ID)						
MD Name Format: Character String (4)						
MD Name Length: 8						
MD Name (String): OPERATOR						
Short MA Name (MEG ID) Format: Character String (2)						
Short MA Name (MEG ID) Length: 3						
Short MA Name: OP1						
Zero-Padding						
+ Defined by ITU-T Y.1731						

More debugging reveals what actually happens when these CCMs are received. The remote MEPs are “found”, and then the remote MEP (rmep) is added to the CC database (CCDB) with the appropriate information.

```
ME2#debug ethernet cfm events domain OPERATOR
Ethernet CFM level 1 domain OPERATOR service OP1 event debugging is on
```

```
! ME2
CFM-EVT: Found remote mep for domain OPERATOR, level 1 vlan 10, mpid 101 mac
001d.4692.f700
CFM-EVT: Updated rmep in MIP CCDB, domain OPERATOR level 1, vlan 10 mac
001d.4692.f700 intf Po12
```

Leaving this event-level debugging enabled, I will unplug both LACP member ports from ME2. The link should fail, and CFM should notify us in many ways. Since logging is enabled, we should see log messages as well as CCDB changes. The alarm indication signal (AIS) is raised since there is a defect condition on the LACP bond. This is a “down” failure since it is towards the remote MEP. The “U” failures on the next debug messages show that the local MEPs no longer have a MIP from which they can send CCMs.

```
! ME2
%E_CFM-6-ENTER_AIS_INT: Interface Port-channel12 enters AIS defect condition
for Down direction
CFM-EVT: CC error event 0x200 cause an errdb entry for level 1, vlan 10, mac
04c5.a453.d800, si 0x0, domain OPERATOR, csi OP1, mpid 8190, archive_time 100
lifetime 210000 idb 0x0 idb name NULL

CFM-AIS: AIS Defect condition for mep at Interface Fa0/14 vlan 10 level 1
period 60
```

```
%E_CFM-6-ENTER_AIS: local mep with mpid 112 level 1 VLAN 10 dir U Interface
Fa0/14 enters AIS defect condition
CFM-AIS: AIS Defect condition for mep at Interface Fa0/13 vlan 10 level 1
period 60
%E_CFM-6-ENTER_AIS: local mep with mpid 102 level 1 VLAN 10 dir U Interface
Fa0/13 enters AIS defect condition

CFM-SAA-EVT: MEP_DELETE for mpid=101 mac=001d.4692.f700 in domain OPERATOR
OP1 vlan 10
```

We can confirm the remote MEPs are no longer available by checking the CCDB on ME2.

```
ME2#show ethernet cfm maintenance-points remote domain OPERATOR
[no output]
```

In case we were not using syslog or debug to troubleshoot, the current CFM errors are also stored in memory. We see two errors at present. The first shows ME1's brain MAC and that its lifetime expired, meaning it is unreachable. It happened 122 seconds ago. The other indicates that the local switch received the AIS signal, which refreshes every 60 seconds by default as long as there is an error.

```
ME2#show ethernet cfm errors domain-id OPERATOR
-----
MPID Domain Id                      Mac Address      Type Id
   MA Name                          Reason           Lvl  Age
   Local MEP Info
-----
101  OPERATOR                      001d.4692.f700   Vlan 10
     OP1                          Lifetime Expired 1    122s
     MPID: 112 Domain: OPERATOR MA: OP1
-   OPERATOR                      04c5.a453.d800   Vlan 10
     OP1                          Receive AIS      1    29s
     MPID: 112 Domain: OPERATOR MA: OP1
```

The debug reveals the reason for the AIS signaling. Although it points to ports 13 and 14, the real issue is the LACP bond being down. Since these MEPs are useless without the relay function (and the MIP), the AIS alarm is against those MEPs.

```
! ME2
CFM-AIS: AIS Defect condition for mep at Interface Fa0/14 vlan 10 level 1
period 60
CFM-AIS: AIS Defect condition for mep at Interface Fa0/13 vlan 10 level 1
period 60
```

Restoring the link, we can see that CFM recovers fully. Several syslog messages are generated, along with debugs, to indicate that AIS conditions were cleared. The MEP discovery process happens again

```
! ME2
```

```
%E_CFM-6-EXIT_AIS_INT: Interface Port-channel12 exited AIS defect condition
for Down direction
CFM-EVT: Found remote mep for domain OPERATOR, level 1 vlan 10, mpid 101 mac
001d.4692.f700
[snip]
%E_CFM-6-EXIT_AIS: local mep with mpid 112 level 1 VLAN 10 dir U Interface
Fa0/14 exited AIS defect condition
%E_CFM-6-EXIT_AIS: local mep with mpid 102 level 1 VLAN 10 dir U Interface
Fa0/13 exited AIS defect condition
[snip]
CFM-EVT: Updated rmep in MIP CCDB, domain OPERATOR level 1, vlan 10 mac
001d.4692.f700 intf Po12
%E_CFM-6-REMOTE_MEP_UP: Continuity Check message is received from a remote
MEP with mpid 101 vlan 10 MA name OP1 domain OPERATOR interface status Up
event code Returning.
```

A quick check of the errors and MEPs shows no error conditions and one remote MEP, as expected.

```
ME2#show ethernet cfm errors domain-id OPERATOR
[no output]
```

```
ME2#show ethernet cfm maintenance-points remote domain OPERATOR
```

```
-----
MPID   Domain Name           MacAddress           IfSt  PtSt
  Lvl   Domain ID              Ingress
  RDI   MA Name              Type Id              SrvcInst
      EVC Name              Age
      Local MEP Info
-----
101     OPERATOR              001d.4692.f700      Up    Up
  1     OPERATOR              Po12
  -     OP1                  Vlan 10              N/A
      N/A                  7s
      MPID: 112 Domain: OPERATOR MA: OP1
```

```
Total Remote MEPs: 1
```

Now that we have built an operator network, we will nest this inside of a larger service provider CFM domain at level 3. ME3 will be an auxiliary switch that the SP contributes to the CFM domain, and this will have MEPs on both ME1 and ME3. This implies that ME2's ports would all be MIPs for this SP domain. Since ME2 LACP bond to ME1 is already a MIP at level 1, this means any higher levels can automatically pass through it, so we don't need to do anything there. However, ports 13 and 14 will suddenly become MIPs for level 3 while remaining MEPs at level 1. This is supported, and makes sense; level 1 should not leak beyond the operator, but the operator is encompassed (nested) within the SP level 3 CFM domain. ME2 does not need any other configuration regarding services that the SP devices have on their MEPs. This greatly enhances CFM scalability because lower levels don't have to touch every single transit device when new CFM services are added by higher levels at the network edges.

```

! ME2
interface range FastEthernet0/13 - 14
 ethernet cfm mip level 3 vlan 10-19

```

Checking the local MIPs, we can see those two new ports as level 3 MIPs, which means traffic at level 3 or above can pass through. The LACP bond is still at level 1, which is fine, because 3 is greater than 1.

```

ME2#show ethernet cfm maintenance-points local mip
Local MIPs:
* = MIP Manually Configured

```

Level	Port	MacAddress	SrvcInst	Type	Id	Source
*3	Fa0/13	04c5.a453.d800	N/A	Vlan	10-19	
*3	Fa0/14	04c5.a453.d800	N/A	Vlan	10-19	
*1	Po12	04c5.a453.d800	N/A	Vlan	10-19	

Next, we will configure the services/MEPs on the endpoints. ME1 and ME3 have the same domain-level configurations, creating a new domain at level 3 and a single service instance to start. The only difference is the MPIDs they have. ME3 also needs to add ports 13 and 14 as MIPs at level 3, just like ME2 did, so that the bridge-relay functionality can be used to send CCMs from the MEPs towards ME2.

```

! ME1 and ME3
ethernet cfm domain PROVIDER level 3
 service SP vlan 10
 continuity-check

```

```

! ME1 only
interface FastEthernet0/9
 ethernet cfm mep domain PROVIDER mpid 201 service SP

```

```

! ME3 only
interface FastEthernet0/9
 ethernet cfm mep domain PROVIDER mpid 203 service SP

```

```

interface range FastEthernet0/13 - 14
 ethernet cfm mip level 3 vlan 10-19

```

From ME3, which has no level 1 operator configuration at all, we will verify the MPs in the CFM domain. This will show the MIPs (2 of them) and MEPs (1 of them).

```

ME3#show ethernet cfm maintenance-points local domain PROVIDER service SP
Local MEPs:

```

MPID	Domain Name	Lvl	MacAddress	Type	CC
Ofld	Domain Id	Dir	Port	Id	
	MA Name		SrvcInst	Source	
	EVC name				

```

-----
203 PROVIDER 3 001f.9d0b.1680 Vlan Y
No PROVIDER Up Fa0/9 10
SP N/A Static
N/A

```

Total Local MEPS: 1

Local MIPs:

* = MIP Manually Configured

```

-----
Level Port      MacAddress      SrvcInst  Type  Id    Source
-----
*3 Fa0/13 001f.9d0b.1680 N/A      Vlan   10
*3 Fa0/14 001f.9d0b.1680 N/A      Vlan   10

```

Total Local MIPs: 2

Next, we check the remote MEPS. We can see MEP 201 from brain MAC 001d.4692.f700 (ME1). The interface and port states are both up, which implies this peer is reachable via port 13 on ME3.

```
ME3#show ethernet cfm maintenance-points remote domain PROVIDER service SP
```

```

-----
MPID  Domain Name      MacAddress      IfSt  PtSt
Lvl  Domain ID          Ingress
RDI  MA Name           Type Id        SrvcInst
     EVC Name          Age
     Local MEP Info
-----
201  PROVIDER          001d.4692.f700  Up    Up
3    PROVIDER          Fa0/13
-    SP                Vlan 10        N/A
     N/A                1s
MPID: 203 Domain: PROVIDER MA: SP

```

We quickly debug CCMs to show the packets being exchanged every 10 seconds. Because the local SPAN session is still up on ME2, we can sniff the wire and see the CCMs there also. Notice the destination is now 0180.c200.0033, which has 3 as the last digit instead of 1. This shows the CFM level and is used for automatic and efficient filtering at the CFM boundaries since it can (probably) be hardware-offloaded.

```
ME3#debug ethernet cfm packets domain PROVIDER service SP
```

```
Ethernet CFM level 3 domain PROVIDER service SP packet debugging is on
```

```
! ME3
```

```
09:43:46: CFM-PKT: Received a CC packet with MPID 201, level 3, vlan 10 from
interface Fa0/13
```

```
09:43:51: CFM-PKT: Sending Up direction MEP 203 CC message, level 3, cos 7,
vlan 10
```

```
09:43:56: CFM-PKT: Received a CC packet with MPID 201, level 3, vlan 10 from
interface Fa0/13
```

09:44:01: CFM-PKT: Sending Up direction MEP 203 CC message, level 3, cos 7, vlan 10

No.	Time	Source	Destination	Protocol	Length	Info
	3.479762000	00:1f:9d:0b:16:80	01:80:c2:00:00:33	CFM	105	Type Continuity Check Message (CCM)
<						
Frame 31: 105 bytes on wire (840 bits), 105 bytes captured (840 bits) on interface 0						
Ethernet II, Src: 00:1f:9d:0b:16:80 (00:1f:9d:0b:16:80), Dst: 01:80:c2:00:00:33 (01:80:c2:00:00:33)						
Destination: 01:80:c2:00:00:33 (01:80:c2:00:00:33)						
Source: 00:1f:9d:0b:16:80 (00:1f:9d:0b:16:80)						
Type: IEEE 802.1ag Connectivity Fault Management (CFM) protocol (0x8902)						
CFM EOAM 802.1ag/ITU Protocol, Type Continuity Check Message (CCM)						
CFM CCM PDU						
Flags: 0x05						
First TLV Offset: 70						
Sequence Number: 3454364120						
...0 0000 1100 1011 = Maintenance Association End Point Identifier: 203						
Maintenance Association Identifier (MEG ID)						
MD Name Format: Character String (4)						
MD Name Length: 8						
MD Name (String): PROVIDER						
Short MA Name (MEG ID) Format: Character String (2)						
Short MA Name (MEG ID) Length: 2						
Short MA Name: SP						
Zero-Padding						

We quickly check for errors and find none. We can assume that the multi-level CFM is functional.

```
ME3#show ethernet cfm errors domain-id PROVIDER service SP
[no output]
```

We can test these MEPs using ping and traceroute. First, we will test the intra-operator MEPs using the CFM loopback messages, which are basically Ethernet pings. The syntax is straightforward where we specify the remote MPID, domain, and service instance. Based on this, the switch looks in the CCDB for a matching remote MEP. This determines the destination MAC address (remote brain MAC) along with the contents of the message so that the proper MEP is probed. From ME1, we will probe ME2's port 13 MEP.

```
ME1#ping ethernet mpid 102 domain OPERATOR service OP1
```

Type escape sequence to abort.

Sending 5 Ethernet CFM loopback messages to 04c5.a453.d800, timeout is 5 seconds:!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 1/3/8 ms

Wireshark identifies these packets as loopback messages (LBM) and loopback replies (LBR), which are probes and responses, respectively. There isn't much detail in the packet other than identifying the type (LBM vs. LBR) with the source/destination MAC addresses, along with the CFM level. Notice the unicast destination of the LBM and LBR messages.

No.	Time	Source	Destination	Protocol	Length	Info
4.936308000		00:1d:46:92:f7:00	04:c5:a4:53:d8:00	CFM	114	Type Loopback Message (LBM)
4.937805000		04:c5:a4:53:d8:00	00:1d:46:92:f7:00	CFM	114	Type Loopback Reply (LBR)

<

+

 Frame 63: 114 bytes on wire (912 bits), 114 bytes captured (912 bits) on interface 0

+

 Ethernet II, Src: 00:1d:46:92:f7:00 (00:1d:46:92:f7:00), Dst: 04:c5:a4:53:d8:00 (04:c5:a4:53:d8:00)

+

 Destination: 04:c5:a4:53:d8:00 (04:c5:a4:53:d8:00)

+

 Source: 00:1d:46:92:f7:00 (00:1d:46:92:f7:00)

+

 Type: IEEE 802.1ag Connectivity Fault Management (CFM) protocol (0x8902)

+

 CFM EOAM 802.1ag/ITU Protocol, Type Loopback Message (LBM)

+

 001. = CFM MD Level: 1

+

 ...0 0000 = CFM Version: 0

+

 CFM OpCode: Loopback Message (LBM) (3)

+

 CFM LBM PDU

+

 CFM TLVs

We can also ping across the provider domain to the MEP on ME3. This is allowed because CFM messages of a higher level can transparently pass through MEPs/MIPs at lower levels.

```
ME1#ping ethernet mpid 203 domain PROVIDER service SP
```

Type escape sequence to abort.

Sending 5 Ethernet CFM loopback messages to 001f.9d0b.1680, timeout is 5 seconds:!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 1/4/9 ms

Traceroute is very powerful as well. Before actually testing it, we can enable the traceroute cache on all switches. Unlike IPv4/v6 traceroutes, CFM traceroutes can be stored in memory for a certain amount of time and up to a certain quantity. These records can assist with troubleshooting over time. I slightly adjust the default size of 100 entries and default time of 100 minutes for demonstration. We can clear the cache (thought it is currently empty) using the command below.

```
! All ME switches
ethernet cfm traceroute cache
ethernet cfm traceroute cache size 512
ethernet cfm traceroute cache hold-time 120
```

```
ME1#clear ethernet cfm traceroute-cache
```

Now, we will perform some traceroutes. First, we will traceroute from ME1 to ME2 inside of the operator level. This is pretty boring since there is only one hop between MEPs, but we learn valuable link information. The MEP on ME1 (source) counts as an intermediate bridge as well, which is why there are “2 hops”. We can clearly see the ingress and egress interfaces, forwarding actions, brain MACs, and relay activity. Compared to IPv4/v6 traceroutes, CFM provides much greater detail.

```
ME1#traceroute ethernet mpid 102 domain OPERATOR service OP1
Type escape sequence to abort. TTL 64. Linktrace Timeout is 5 seconds
Tracing the route to 04c5.a453.d800 on Domain OPERATOR, Level 1, service OP1, vlan 10
Traceroute sent via Po12, path found via FDB
```

B = Intermediary Bridge

! = Target Destination

* = Per hop Timeout

Hops	Host	MAC Forwarded	Ingress Egress	Ingr Action Egr Action	Relay Action Previous Hop
B 1		001d.4692.f700	Fa0/9	IngOk	RlyFDB
		Forwarded	Pol2	EgrOK	
! 2		04c5.a453.d800	Pol2	IngOk	RlyHit:MEP
		Not Forwarded			001d.4692.f700

Looking at Wireshark, we can see a new pair of messages, link trace message (LTM) and link trace reply (LTR). The LTR shows some new interesting fields, such as a TTL value, original/target addresses, and a transaction ID. The multicast destination MAC addresses changes based on the level plus 8. For example, at level 1, the CCMs used a last digit of 1, and the LTRs use a last digit of 9. This helps support my argument that the internal CFM filters are hardware-assisted because, in binary, 1 and 9 are identical with the exception of the first bit in the nibble (0001 versus 1001). The CFM domain can ignore the first bit and only look at the last 3 for filtering. An LTR at level 7, for example, would use a value of F, while the CCMs would use 7 (1111 vs. 0111). The original/target unicast MAC addresses are contained within the payload of the LTM frame as seen below.

No.	Time	Source	Destination	Protocol	Length	Info
	7.093757000	00:1d:46:92:f7:00	01:80:c2:00:00:39	CFM	60	Type Linktrace Message (LTM)
	7.101550000	04:c5:a4:53:d8:00	00:1d:46:92:f7:00	CFM	78	Type Linktrace Reply (LTR)
<						
+ Frame 77: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0						
- Ethernet II, Src: 00:1d:46:92:f7:00 (00:1d:46:92:f7:00), Dst: 01:80:c2:00:00:39 (01:80:c2:00:00:39)						
+ Destination: 01:80:c2:00:00:39 (01:80:c2:00:00:39)						
+ Source: 00:1d:46:92:f7:00 (00:1d:46:92:f7:00)						
Type: IEEE 802.1ag Connectivity Fault Management (CFM) protocol (0x8902)						
+ CFM EOAM 802.1ag/ITU Protocol, Type Linktrace Message (LTM)						
- CFM LTM PDU						
+ Flags: 0x00						
First TLV Offset: 17						
Linktrace Transaction Identifier: 3335634746						
Linktrace TTL: 63						
Linktrace Message: Original Address: 00:1d:46:92:f7:00 (00:1d:46:92:f7:00)						
Linktrace Message: Target Address: 04:c5:a4:53:d8:00 (04:c5:a4:53:d8:00)						
- CFM TLVs						
- TLV: LTM Egress Identifier TLV (t=7,l=8)						
TLV Type: LTM Egress Identifier TLV (7)						
TLV Length: 8						
Egress Identifier - Unique Identifier: 0000						
Egress Identifier - MAC of LT Initiator/Responder: 00:1d:46:92:f7:00 (00:1d:46:92:f7:00)						
+ TLV: End TLV (t=0,l=0)						

The LTR is much more interesting since it contains all of the important information shown in the traceroute reply. The ingress and egress details are carried in their own TLVs along with the relevant ingress/egress brain MACs.

No.	Time	Source	Destination	Protocol	Length	Info
	7.101550000	04:c5:a4:53:d8:00	00:1d:46:92:f7:00	CFM	78	Type Linktrace Reply (LTR)
+ Frame 81: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface 0 - Ethernet II, Src: 04:c5:a4:53:d8:00 (04:c5:a4:53:d8:00), Dst: 00:1d:46:92:f7:00 (00:1d:46:92:f7:00) + Destination: 00:1d:46:92:f7:00 (00:1d:46:92:f7:00) + Source: 04:c5:a4:53:d8:00 (04:c5:a4:53:d8:00) Type: IEEE 802.1ag Connectivity Fault Management (CFM) protocol (0x8902) + CFM EOAM 802.1ag/ITU Protocol, Type Linktrace Reply (LTR) + CFM LTR PDU + CFM TLVs + TLV: LTR Egress Identifier TLV (t=8,l=16) - TLV: Reply Ingress TLV (t=5,l=13) TLV Type: Reply Ingress TLV (5) TLV Length: 13 Ingress Action: IngOK (1) Ingress MAC address: 04:c5:a4:53:d8:00 (04:c5:a4:53:d8:00) Chassis ID Length: 4 Chassis ID Sub-type: 5 Chassis ID: 506f3132 - TLV: Reply Egress TLV (t=6,l=15) TLV Type: Reply Egress TLV (6) TLV Length: 15 Egress Action: EgrOK (1) Egress MAC address: 04:c5:a4:53:d8:00 (04:c5:a4:53:d8:00) Chassis ID Length: 6 Chassis ID Sub-type: 5 Chassis ID: 4661302f3134						

A more interesting example involves tracing across the provider CFM domain from ME1 to ME3. Here, we can see both ME1 and ME2 are intermediary bridges, with ME3 as the final termination point. Notice that the details of the operator network (LACP bond, etc) are not included in this traceroute. Only the MIPs/MEPs relevant for level 3 are shown. This gives the operators some level of “privacy” since the larger levels only care about having service and testing the “seams”. The LACP bond does not support processing any CFM traffic above level 1, which is why the LTM is transparently passed through. No egress record is added by ME1 and no ingress record is added by ME2, which hides the operator domain from the provider. Instead, the level-3 MIP on ME2’s port 13 responds with a successful egress action as expected. The brain MACs of all switches that have level-3 MPs are revealed in the proper sequence.

```
ME1#traceroute ethernet mpid 203 domain PROVIDER service SP
Type escape sequence to abort. TTL 64. Linktrace Timeout is 5 seconds
Tracing the route to 001f.9d0b.1680 on Domain PROVIDER, Level 3, service SP, vlan 10
Traceroute sent via Po12, path found via FDB
```

B = Intermediary Bridge
 ! = Target Destination
 * = Per hop Timeout

Hops	Host	MAC Forwarded	Ingress Egress	Ingr Action Egr Action	Relay Action Previous Hop
B 1		001d.4692.f700	Fa0/9	IngOk	RlyFDB
		Forwarded			
B 2		04c5.a453.d800			RlyFDB
		Forwarded	Fa0/13	EgrOK	001d.4692.f700
! 3		001f.9d0b.1680	Fa0/13	IngOk	RlyHit:MEP

The LTR is shown below. There is only one LTM that is sent, unlike IP traceroute where subsequent TTL expirations cause more probes to be sent. The probe is relayed by the forwarding database (RlyFDB) which means it was successfully passed along. The reason the first LTR is only 62 bytes is because it didn't contain an ingress record TLV, as shown above. The final LTR is shown below where both the ingress and egress statuses are displayed from the remote MEP's response. The middle LTR was generated by the level-3 MIP, which is why MIPs are important to the CFM architecture.

No.	Time	Source	Destination	Protocol	Length	Info
	16.0806/2000	00:1d:46:92:f7:00	01:80:c2:00:00:3b	CFM	60	Type Linktrace Message (LTM)
	16.083382000	04:c5:a4:53:d8:00	00:1d:46:92:f7:00	CFM	62	Type Linktrace Reply (LTR)
	16.085707000	00:1f:9d:0b:16:80	00:1d:46:92:f7:00	CFM	79	Type Linktrace Reply (LTR)

Frame 261: 79 bytes on wire (632 bits), 79 bytes captured (632 bits) on interface 0

Ethernet II, Src: 00:1f:9d:0b:16:80 (00:1f:9d:0b:16:80), Dst: 00:1d:46:92:f7:00 (00:1d:46:92:f7:00)

CFM EOAM 802.1ag/ITU Protocol, Type Linktrace Reply (LTR)

CFM LTR PDU

CFM TLVs

- TLV: LTR Egress Identifier TLV (t=8,l=16)
- TLV: Reply Ingress TLV (t=5,l=15)
 - TLV Type: Reply Ingress TLV (5)
 - TLV Length: 15
 - Ingress Action: IngOK (1)
 - Ingress MAC address: 00:1f:9d:0b:16:80 (00:1f:9d:0b:16:80)
 - Chassis ID Length: 6
 - Chassis ID Sub-type: 5
 - Chassis ID: 4661302f3133
- TLV: Reply Egress TLV (t=6,l=14)
 - TLV Type: Reply Egress TLV (6)
 - TLV Length: 14
 - Egress Action: EgrOK (1)
 - Egress MAC address: 00:1f:9d:0b:16:80 (00:1f:9d:0b:16:80)
 - Chassis ID Length: 5
 - Chassis ID Sub-type: 5
 - Chassis ID: 4661302f39

We can check the traceroute-cache to see this traceroute recorded for future reference. We can specifically look for a traceroute to a target MPID within a domain/service. The entry is also timestamped for easy troubleshooting.

```
ME1#show ethernet cfm traceroute-cache mpid 203 domain PROVIDER service SP
Current Cache-size: 9 Hops
Max Cache-size: 512 Hops
Hold-time: 120 Minutes
```

```
Traceroute to 001f.9d0b.1680 on Domain PROVIDER, Level 3, service SP, vlan 10 issued
at *10:16:45.693 UTC Mon Mar 1 1993
path found via FDB
```

```
B = Intermediary Bridge
! = Target Destination
* = Per hop Timeout
```

```
-----
Hops      Host      MAC          Ingress      Ingr Action  Relay Action
          Forwarded Egress       Egr Action   Previous Hop
```

```

-----
B 1          001d.4692.f700 Fa0/9          IngOk      RlyFDB
              Forwarded
B 2          04c5.a453.d800
              Forwarded      Fa0/13          EgrOK      001d.4692.f700
! 3          001f.9d0b.1680 Fa0/13          IngOk      RlyHit:MEP
              Not Forwarded              04c5.a453.d800

```

```

1 matched linktrace found for dest_mac 001f.9d0b.1680, domain PROVIDER service SP,
vlan 10.

```

After performing valuable diagnostic tests like ping and traceroute, we can check the statistics of them. The LTR (traceroute reply) statistics are limited to unexpected LTR receptions, which should always be 0. The LBRs (ping reply) are counted even when they pass, and given two sets of pings (I only showed one for the PROVIDER domain), a total of 10 good LBRs were received. This also counts CCMs transmitted, which is good way to tell if the MEP is even sending CCMs at all. I also show ME3's statistics for the PROVIDER domain because it shows 10 LBR's transmitted, since it was the device hosting the MEP replying to the pings.

```

ME1#show ethernet cfm statistics domain PROVIDER service SP
BRAIN MAC: 001d.4692.f700

```

```

DomainName: PROVIDER
MA Name: SP
MPID: 201
  Last clearing of counters:  never
    CCMs:
      Transmitted:                1212    Rcvd Seq Errors:            0
    LTRs:
      Unexpected Rcvd:            0
    LBRs:
      Transmitted:                0      Rcvd Seq Errors:            0
      Rcvd in order:             10      Rcvd Bad MSDU:             0

```

```

ME3#show ethernet cfm statistics domain PROVIDER service SP
BRAIN MAC: 001f.9d0b.1680

```

```

DomainName: PROVIDER
MA Name: SP
MPID: 203
  Last clearing of counters:  never
    CCMs:
      Transmitted:                1263    Rcvd Seq Errors:            0
    LTRs:
      Unexpected Rcvd:            0
    LBRs:
      Transmitted:                10      Rcvd Seq Errors:            0
      Rcvd in order:             0      Rcvd Bad MSDU:             0

```

To support multiple customers, a single domain can have multiple services. On ME1 and ME3, we will create another service called SP_STATIC using VLAN 11. We will apply several minor options to this service. First, we statically identify the remote MEPs we expect to see within the domain, which can be used for additional strictness on neighbor formation. We enable CCMs for this new service and we tune their interval to be much faster at 1 per second. We permit the loss of up to 5 CCMs versus the default of 3, so MEP failure detection should be about 5 seconds. We can apply the MEPs to the same port as the other MEPs provided we have a different MPID and service tag. Under the service details, I adjust the CoS of the packets to 4 (default is 7 as seen earlier) and enable all possible alarms.

```
! ME3
ethernet cfm domain PROVIDER level 3
  service SP_STATIC vlan 11
    mep mpid 213
    continuity-check
    continuity-check interval 1s
    continuity-check loss-threshold 5
    continuity-check static rmep

interface FastEthernet0/9
  ethernet cfm mep domain PROVIDER mpid 213 service SP_STATIC
  cos 4
  alarm notification all

! ME1
ethernet cfm domain PROVIDER level 3
  service SP_STATIC vlan 11
    mep mpid 213
    continuity-check
    continuity-check interval 1s
    continuity-check loss-threshold 5
    continuity-check static rmep

interface FastEthernet0/9
  ethernet cfm mep domain PROVIDER mpid 211 service SP_STATIC
  cos 4
  alarm notification all
```

We will perform our verifications from the perspective of ME1. Using a long show command, we will print out the details of the specific MEP we just added. Although there are no defects at present, I highlight this field because it is useful for checking defects associated with MEPs when problems do arise. The most significant information is at the top, and we confirm that it is correct.

```
ME1#show ethernet cfm maintenance-points local detail mep domain PROVIDER
service SP_STATIC
Local MEPs:
-----
MPID: 211
```

```
DomainName: PROVIDER
MA Name: SP_STATIC
Level: 3
Direction: Up
Vlan: 11
Interface: Fa0/9
CC Offload: No
CC-Status: Enabled
CC Loss Threshold: 5
MAC: 001d.4692.f700
LCK-Status: Enabled
LCK Period: 60000(ms)
LCK Expiry Threshold: 3.5
Level to transmit LCK: Default
Defect Condition: No Defect
presentRDI: FALSE
AIS-Status: Enabled
AIS Period: 60000(ms)
AIS Expiry Threshold: 3.5
Level to transmit AIS: Default
Suppress Alarm configuration: Enabled
Suppressing Alarms: No
Source: Static
```

We can likewise verify the remote MPID using a detailed show command. We see that the interface/port states are up for this peer and that the remote defect indication (RDI) is not set. No CCMs have been received in error, either.

```
ME1#show ethernet cfm maintenance-points remote detail mpid 213
Version: IEEE-CFM
MAC Address: 001f.9d0b.1680
Domain Name: PROVIDER
MA Name: SP_STATIC
Level: 3
VLAN: 11
MPID: 213
Incoming Port(s): Port-channel12
CC Lifetime(sec): 5.500
Age of Last CC Message(sec): 0
CC Packet Statistics: 1416/0 (Received/Error)
MEP interface status: Up
MEP port status: Up
Receive RDI: FALSE
```

Debugging packets on ME1 reveals the CCMs for this new service. They arrive very quickly, one per second, and are sent at the same rate. We verify the MPID, VLAN, and CoS as well of the packets we originate.

```
ME1#debug ethernet cfm packets domain PROVIDER service SP_STATIC
Ethernet CFM level 3 domain PROVIDER service SP_STATIC packet debugging is on
```

```
00:23:04: CFM-PKT: Received a CC packet with MPID 213, level 3, vlan 11 from
interface Po12
```

```
00:23:04: CFM-PKT: Sending Up direction MEP 211 CC message, level 3, cos 4,
vlan 11
```

```
00:23:05: CFM-PKT: Received a CC packet with MPID 213, level 3, vlan 11 from
interface Po12
```

```
00:23:05: CFM-PKT: Sending Up direction MEP 211 CC message, level 3, cos 4,
vlan 11
```

We can verify connectivity using the ping utility again to generate LBMs. Rather than perform another unicast Ethernet ping, we can perform a multicast Ethernet ping. This is a handy way to discovery other MEPs on the segment using the same destination MAC address used for CCMs at the domain's CFM level. Since we are pinging within CFM level 3, the destination MAC should be 0180.c200.0033. We see one response from 001f.9d0b.1680, which is the brain MAC of ME3.

```
ME1#ping ethernet multicast domain PROVIDER service SP_STATIC
Type escape sequence to abort.
Sending 5 Ethernet CFM loopback messages to 0180.c200.0033, timeout is 5
seconds:
Reply to Multicast request via interface Port-channel12, from 001f.9d0b.1680,
0 ms
Total Loopback Responses received: 1
```

For variety, I follow up with a unicast ping to this destination after having just discovered it. We can also specify the brain MAC as the destination, along with minor options like size and CoS adjustments.

```
ME1#ping ethernet 001f.9d0b.1680 domain PROVIDER service SP_STATIC cos 3 size
555
Type escape sequence to abort.
Sending 5 Ethernet CFM loopback messages to 001f.9d0b.1680, timeout is 5
seconds:!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms
```

Next, we will examine CFM errors in greater detail. This is a very powerful feature of CFM since it tells you exactly what the problem is, and where. There are 5 main error codes which are described clearly in "IEEE8021-CFM-MIB-200811180000Z.txt":

Defect	Priority	Description
DefXconCCM	1 (highest)	Received CCM from another maintenance domain or a lower CFM level. These are fundamental CFM violations; cross-connect error.
DefErrorCCM	2	Received invalid CCM; often due to unknown MEP or loop error.
DefRemoteCCM	3	Not receiving valid CCMs from a valid RMEP; missing MEP.
DefMACStatus	4	Failure in the port-status TLV; MEP is down.

DefRDICCM	5 (lowest)	Received a valid CCM from the peer, but RDI is set to indicate a remote fault; check peer for error condition.

Just like with syslog, we can adjust what the is least important event that is still important enough to generate error conditions. In the previous example, we enabled all alarm notifications, which is loosely analogous to syslog level 7 (debug). Rather than describe each one, below is the context-sensitive help. I've rearranged the output to sort them in order from least restrictive (all) to most restrictive (none) which makes it easier to read. By default, RDI CCM events are not reported, which I've highlighted.

```
ME1(config)#ethernet cfm alarm notification ?
  all                      Report all defects: DefRDI, DefMACStatus, DefRemote,
                          DefError, DefXcon
  mac-remote-error-xcon    Report only: DefMACStatus, DefRemote, DefError and
                          DefXcon (default)
  remote-error-xcon        Report only: DefRemote, DefError and DefXcon
  error-xcon               Report only: DefError and DefXcon
  xcon                     Report only: DefXcon
  none                     No defects to be reported
```

Next, we will examine the particular errors in more detail. Since the CCMs are being verified against a static configuration with the SP_STATIC service instance, we can change one of the RMEP lists on either switch to break it. On ME1, I remove RMEP 213 and replace it with an invalid RMEP 99. This means ME1 is going to review CCMs from MPID 213 despite it not being configured. It will also fail to receive CCMs from MPID 99 which is configured. Both of these are faults.

```
! ME1
ethernet cfm domain PROVIDER level 3
  service SP_STATIC vlan 11
  no mep mpid 213
  mep mpid 99
```

Immediately upon entering this configuration, ME1 generates a log message indicating a fault alert. Based on the table above, we know that this is due to an invalid RMEP.

```
! ME1
%E_CFM-3-FAULT_ALARM: A fault has occurred in the network for the local MEP
having mpid 211 vlan 11 for service MA name SP_STATIC with the event code
DefErrorCCM.
```

We have two different errors as described above. First, we see CCMs from RMEP 213 which is not configured, and is thus considered invalid. Second, we have RMEP 99 configured but don't see it anywhere in the CCDB, so there is no brain MAC for it. The switch doesn't know what the "right" answer is, but this is a likely indication of a misconfiguration.

```
ME1#show ethernet cfm errors domain-id PROVIDER service SP_STATIC
```

MPID	Domain Id	Mac Address	Type	Id
	MA Name	Reason	Lvl	Age
	Local MEP Info			
99	PROVIDER	0000.0000.0000	Vlan 11	
	SP_STATIC	RMEP missing	3	0s
	MPID: 211 Domain: PROVIDER MA: SP_STATIC			
213	PROVIDER	001f.9d0b.1680	Vlan 11	
	SP_STATIC	Unknown MEP	3	0s
	MPID: 211 Domain: PROVIDER MA: SP_STATIC			

Shortly thereafter (as soon as ME1 sends the message), ME3 is aware of some issue. From ME3's perspective, there is no problem with the network, since the RMEP it has configured is still correct. The error message indicates that a CCM was received from MPID 213 with RDI set within the CCM. This indicates some fault with the RMEP; it is ME1's way of saying "I have a problem and need help". When we look at the error table, we don't get much detail other than having received a CCM with RDI set. The value of this cursory check is that it tells us the MPID and brain MAC of the defective device. From ME3's perspective, this doesn't help with fault identification, but it does help with fault isolation.

! ME3

```
%E_CFM-3-FAULT_ALARM: A fault has occurred in the network for the local MEP
having mpid 213 vlan 11 for service MA name SP_STATIC with the event code
DefRDICCM.
```

```
ME3#show ethernet cfm errors domain-id PROVIDER service SP_STATIC
```

MPID	Domain Id	Mac Address	Type	Id
	MA Name	Reason	Lvl	Age
	Local MEP Info			
211	PROVIDER	001d.4692.f700	Vlan 11	
	SP_STATIC	Receive RDI	3	0s
	MPID: 213 Domain: PROVIDER MA: SP_STATIC			

Unlike ME1, ME3 actually has MPID 211 in its database but sees RDI is enabled. There isn't an error with the CCMs per se, but this is ME1's way of signaling an issue as stated earlier.

```
ME3#show ethernet cfm maintenance-points remote detail mpid 211
```

```
Version: IEEE-CFM
MAC Address: 001d.4692.f700
Domain Name: PROVIDER
MA Name: SP_STATIC
Level: 3
VLAN: 11
MPID: 211
Incoming Port(s): FastEthernet0/13
CC Lifetime(sec): 5.500
Age of Last CC Message(sec): 0
```



```
CC Packet Statistics: 2654/0 (Received/Error)
MEP interface status: Up
MEP port status: Up
Receive RDI: TRUE
```

Checking ME1's local MEP detailed status, we can see that AIS is identified as the defect condition and RDI is presented to any remote peers of this MPID.

```
ME1#show ethernet cfm maintenance-points local detail mep domain PROVIDER
service SP_STATIC | begin AIS
Defect Condition: AIS
presentRDI: TRUE
AIS-Status: Enabled
AIS Period: 60000(ms)
AIS Expiry Threshold: 3.5
Level to transmit AIS: Default
Suppress Alarm configuration: Enabled
Suppressing Alarms: No
Source: Static
```

To continue detailed verification, we enable debugs on ME1. Because an invalid CCM has been received from MPID 211 (the unspecified RMEP), the CCMs are discarded. The error codes are included in the debugs also. The same debugging on ME3 reveals similar output and shows the reception of RDI CCMs. The defects are ignored because those are already the "highest" defects in the error tables. On ME1, we see a DefRemoteCCM which relates to MPID 99 being configured but receiving no CCMs. DefErrorCCM relates to the RMEP 213 CCMs being received but invalid due to not having MPID 213 configured. The DefErrorCCM issue is a "bigger deal" as it is a higher priority fault.

```
ME1#debug ethernet cfm events domain PROVIDER service SP_STATIC
Ethernet CFM level 3 domain PROVIDER service SP_STATIC event debugging is on

! ME1
00:52:28: CFM-EVT: Ignoring defect DefRemoteCCM, highest defect is
DefErrorCCM for MEP mpid 211
00:52:28: CFM-EVT: Unknown remote mep 213 at domain PROVIDER level 3, vlan
11, addr 001f.9d0b.1680
00:52:28: CFM-EVT: Ignoring defect DefErrorCCM, highest defect is DefErrorCCM
for MEP mpid 211
00:52:28: CFM-EVT: Ignoring defect DefRemoteCCM, highest defect is
DefErrorCCM for MEP mpid 211

! ME3
00:54:46: CFM-EVT: Found remote mep for domain PROVIDER, level 3 vlan 11,
mpid 211 mac 001d.4692.f700
00:54:46: CFM-EVT: Service up at domain PROVIDER level 3, vlan 11
00:54:46: CFM-EVT: Ignoring defect DefRDICCM, highest defect is DefRDICCM for
MEP mpid 213
```

```
00:54:46: CFM-EVT: Updated rmep in MIP CCDB, domain PROVIDER level 3, vlan 11
mac 001d.4692.f700 intf Fa0/13
```

To begin fixing the problem, I add MPID 213 back to the service SP_STATIC. ME1 quickly prints a log message to indicate a MEP UP condition, which means valid CCMs were received from MPID 213.

```
! ME1
ethernet cfm domain PROVIDER level 3
  service SP_STATIC vlan 11
    mep mpid 213
```

```
%E_CFM-6-REMOTE_MEP_UP: Continuity Check message is received from a remote
MEP with mpid 213 vlan 11 MA name SP_STATIC domain PROVIDER interface status
Up event code New.
```

However, we don't see RDI clearing on ME3, which indicates something else is still wrong. ME1 is still expecting CCMs for MPID 99 per the configuration, which triggers a DefRemoteCCM alarm.

```
ME1#show ethernet cfm errors domain-id PROVIDER service SP_STATIC
-----
MPID Domain Id                      Mac Address      Type Id
  MA Name                          Reason           Lvl  Age
  Local MEP Info
-----
99  PROVIDER                        0000.0000.0000   Vlan 11
    SP_STATIC                      RMEP missing     3    0s
    MPID: 211 Domain: PROVIDER MA: SP_STATIC
```

To fully fix this problem, we have to remove MPID 99 from ME1's configuration again. This clears the AIS defect for MPID 211 which only occurs once all alarms are clear. We verify by checking the error tables on both ME1 and ME3 to verify, and for good measure, use LBMs to triple check.

```
! ME1
ethernet cfm domain PROVIDER level 3
  service SP_STATIC vlan 11
    no mep mpid 99
```

```
%E_CFM-6-EXIT_AIS: local mep with mpid 211 level 3 VLAN 11 dir U Interface
Fa0/9 exited AIS defect condition
```

```
ME1#show ethernet cfm errors domain-id PROVIDER service SP_STATIC
[no output]
```

```
ME3#show ethernet cfm errors domain-id PROVIDER service SP_STATIC
[no output]
```

```
ME1#ping ethernet mpid 213 domain PROVIDER service SP_STATIC
Type escape sequence to abort.
```

Sending 5 Ethernet CFM loopback messages to 001f.9d0b.1680, timeout is 5 seconds:!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/8 ms

To test another common fault, I will unplug port 9 on ME3. This will affect both the SP and SP_STATIC services bound to that port. ME1 receives CCMs from ME3 that indicate the port has failed, and in return, sets the DefMACstatus alarm. This generally represents an RMEP being down, so seeing that particular error messages makes sense in this case. From CFM's perspective, the service cannot possibly be operational when the UNI port is down.

```
! ME1
%E_CFM-6-REMOTE_MEP_UP: Continuity Check message is received from a remote
MEP with mpid 213 vlan 11 MA name SP_STATIC domain PROVIDER interface status
Down event code PortState.
%E_CFM-6-REMOTE_MEP_UP: Continuity Check message is received from a remote
MEP with mpid 203 vlan 10 MA name SP domain PROVIDER interface status Down
event code PortState.
%E_CFM-3-FAULT_ALARM: A fault has occurred in the network for the local MEP
having mpid 211 vlan 11 for service MA name SP_STATIC with the event code
DefMACstatus.
%E_CFM-3-FAULT_ALARM: A fault has occurred in the network for the local MEP
having mpid 201 vlan 10 for service MA name SP with the event code
DefMACstatus.
```

Examining debugs on ME1, we can see similar output as before. It shows that the current defect is DefMACstatus for ME1, which in turn results in RDI CCMs towards ME3. The RDI CCMs are sent in both SP and SP_STATIC services in ME3 port 9 supported both services.

```
! ME1
01:31:03: CFM-EVT: Found remote mep for domain PROVIDER, level 3 vlan 11,
mpid 213 mac 001f.9d0b.1680
01:31:03: CFM-EVT: Service up at domain PROVIDER level 3, vlan 11
01:31:03: CFM-EVT: Updated rmep in MIP CCDB, domain PROVIDER level 3, vlan 11
mac 001f.9d0b.1680 intf Po12
01:31:03: CFM-EVT: Ignoring defect DefMACstatus, highest defect is
DefMACstatus for MEP mpid 211
```

ME3#show ethernet cfm errors domain-id PROVIDER

MPID Domain Id		Mac Address	Type Id	
MA Name		Reason	Lvl	Age
Local MEP Info				
201	PROVIDER	001d.4692.f700	Vlan 10	
	SP	Receive RDI	3	1s
	MPID: 203 Domain: PROVIDER MA: SP			
211	PROVIDER	001d.4692.f700	Vlan 11	
	SP_STATIC	Receive RDI	3	0s

MPID: 213 Domain: PROVIDER MA: SP_STATIC

Because ME3 saw RDI CCMs from ME1 (identified by brain MAC), we know the problem is on ME1. Again, RDI helps with fault isolation, not necessarily fault identification. Oddly, the ME1 error table doesn't reveal this fault, but the remote MPID details give us a hint. We can see the RMEP is down, which explains why CFM raises the alarm.

```
ME1#show ethernet cfm maintenance-points remote detail mpid 213
Version: IEEE-CFM
MAC Address: 001f.9d0b.1680
Domain Name: PROVIDER
MA Name: SP_STATIC
Level: 3
VLAN: 11
MPID: 213
Incoming Port(s): Port-channel12
CC Lifetime(sec): 5.500
Age of Last CC Message(sec): 0
CC Packet Statistics: 821/0 (Received/Error)
MEP interface status: Down
MEP port status: Up
Receive RDI: FALSE
```

Plugging the cable back in, as expected, resolves the issue quickly. The RDI CCM errors are cleared on ME3, which allows the port to exit AIS status. ME1 displays log message that indicate valid CCMs are received from RMEPs 213 and 203, which cover both services in the PROVIDER domain.

```
! ME3
%E_CFM-6-EXIT_AIS_INT: Interface FastEthernet0/9 exited AIS defect condition
for Down direction
```

```
! ME1
%E_CFM-6-REMOTE_MEP_UP: Continuity Check message is received from a remote
MEP with mpid 213 vlan 11 MA name SP_STATIC domain PROVIDER interface status
Up event code PortState.
%E_CFM-6-REMOTE_MEP_UP: Continuity Check message is received from a remote
MEP with mpid 203 vlan 10 MA name SP domain PROVIDER interface status Up
event code PortState.
```

We can also define MEPs based on ports rather than VLANs. These are considered outward-facing (down) MEPs which means that the CCMs are sent via the wire-side and not the bridge-relay function. Rather than ruin our existing network to demonstrate this, I use a pair of dummy ports on ME1 and ME3 to configure it. Because those ports don't actually communicate, the MEPs won't see one another, but we can view the port-based MEP operation with show and debug commands. For variety, I used a number-based service (rather than a custom string) although this not specific to the port-MEP configuration. I also specify the maximum number of MEPs within a service instance; the default is 100.

```

! ME1 and ME3
ethernet cfm domain PROVIDER level 3
  service number 19 port
  continuity-check
  maximum meps 2

! ME1 only
interface FastEthernet0/7
  ethernet cfm mep domain PROVIDER mpid 291 service number 19

! ME3 only
interface FastEthernet0/7
  ethernet cfm mep domain PROVIDER mpid 293 service number 19

```

Looking at ME3, we check the MEPs configured on port 7. The port-MEP is port-based, not VLAN-based, so I use a dedicated port for this. We can see the direction is “Down”, type is “Port”, CCMs are enabled, and the CFM level is 3. To identify this as a port-MEP, we see the “Id” is “N/A” which means no VLAN.

```

ME3#show ethernet cfm maintenance-points local mep interface fast0/7
Local MEPs:

```

MPID Domain Name		Lvl	MacAddress	Type	CC
Ofld Domain Id		Dir	Port	Id	
MA Name			SrvcInst	Source	
EVC name					
293	PROVIDER	3	001f.9d0b.1689	Port	Y
No	PROVIDER	Down	Fa0/7	none	
	number 19		N/A	Static	
	N/A				

Using debugs for service 19, we can see the CCMs being sent out of this physical interface. This is different from the “UP MEP” where the bridge-relay function used the brain MAC as the source and send CFM frames in towards the CFM level. In this case, packets are being sent out of port 7 every 10 seconds in the “down” direction.

```

ME3#debug ethernet cfm packets domain PROVIDER service number 19
Ethernet CFM level 3 domain PROVIDER service number 19 packet debugging is on

```

```

! ME3
00:14:40: CFM-PKT: Sending Down direction MEP 293 CC message, level 3, cos 7,
port on Fa0/7
00:14:50: CFM-PKT: Sending Down direction MEP 293 CC message, level 3, cos 7,
port on Fa0/7

```

We will create a local SPAN session to capture this in Wireshark to verify the source MAC address. I also reveal both the brain MAC and the port MAC. We can see the source MAC matches that of port 7. I

highlight the maintenance association (MA) short name in the capture since it shows 0x0013, which is 19 in decimal. This is the service number, which could be a text string as we saw earlier.

```
ME3#show interfaces fastEthernet 0/7 | include bia
Hardware is Fast Ethernet, address is 001f.9d0b.1689 (bia 001f.9d0b.1689)
```

```
ME3#show ethernet cfm statistics | include BRAIN
BRAIN MAC: 001f.9d0b.1680
```

```
! ME3
interface FastEthernet0/23
description SPAN PORT
```

```
monitor session 1 source interface Fa0/7 tx
monitor session 1 destination interface Fa0/23 encapsulation replicate
```

No.	Time	Source	Destination	Protocol	Length	Info
	71.720278000	00:1f:9d:0b:16:89	01:80:c2:00:00:33	CFM	101	Type Continuity Check Message (CCM)
<						
Frame 26: 101 bytes on wire (808 bits), 101 bytes captured (808 bits) on interface 0						
Ethernet II, Src: 00:1f:9d:0b:16:89 (00:1f:9d:0b:16:89), Dst: 01:80:c2:00:00:33 (01:80:c2:00:00:33)						
CFM EOAM 802.1ag/ITU Protocol, Type Continuity Check Message (CCM)						
CFM CCM PDU						
Flags: 0x05						
First TLV Offset: 70						
Sequence Number: 2468526776						
...0 0001 0010 0101 = Maintenance Association End Point Identifier: 293						
Maintenance Association Identifier (MEG ID)						
MD Name Format: Character String (4)						
MD Name Length: 8						
MD Name (String): PROVIDER						
Short MA Name (MEG ID) Format: 2-octet integer (3)						
Short MA Name (MEG ID) Length: 2						
Short MA Name: 0013						

Since the port-MEPs don't have reachability to one another, the RMEPs are never discovered. This technique makes sense on a CE node, potentially, where an entire port is classified uniformly into a single CFM domain and level.

```
ME3#show ethernet cfm maintenance-points remote domain PROVIDER service
number 19
[no output]
```

The last CFM feature we will test is the MEP crosscheck function. This is an exec command that checks the dynamically discovered RMEPs (via CCMs) against the statically configured RMEPs. In the SP_STATIC service, we configured static RMEPs and used CCMs to verify them. This is a recurring/dynamic way to ensure rogue MEPs do not appear in the network and that static MEPs always are present in the network. Cross-check is a one-time, user-initiated method of accomplishing the same thing. Looking at ME3, we can check the cross-check history for all services. Only SP_STATIC shows up since the feature requires static RMEPs configured, otherwise cross-check doesn't know what to test against. The current

status is N/A since the feature has not been used yet. Notice that the show command was very generic and the other services, like number 19 and SP, did not show up; neither have static RMEPs configured.

```
ME3#show ethernet cfm maintenance-points remote crosscheck
```

MPID	Domain Name	Lvl	Type	Id	Mep-Up
	MA Name				
	Local MEP Info				
211	PROVIDER	3	Vlan	11	n/a
	SP_STATIC				
	N/A				

Considering CCMs are already verifying the reachability of MPID 211 from ME3's perspective, the manual cross-check isn't as useful as the automatic CCM method. We will initiate a cross-check on ME3 within the PROVIDER domain and SP_STATIC service. Since the cross-check just uses CCMs anyway, there is nothing to investigate with debugging/Wireshark on the devices. After cross-check completes, a syslog is generated to show that all RMEPs were found with no rogue RMEPs found. The show command also shows "Yes" for each RMEP that was cross-checked.

```
ME3#ethernet cfm mep crosscheck enable domain PROVIDER service SP_STATIC
```

```
00:37:19: %E_CFM-6-CROSSCHECK_SERVICE UP: The device received CC messages from all remote MEPs within a given MA name SP_STATIC in the maintenance domain PROVIDER.
```

```
ME3#show ethernet cfm maintenance-points remote crosscheck
```

MPID	Domain Name	Lvl	Type	Id	Mep-Up
	MA Name				
	Local MEP Info				
211	PROVIDER	3	Vlan	11	Yes
	SP_STATIC				
	N/A				

To temporarily demonstrate the value of cross-check, I will remove the CCM static RMEP checking from ME3 only, while adding a bogus remote MPID. Because ME3 isn't using CCMs to check for this anymore, it won't see an error. ME1 sees no issue with the configuration since it has one RMEP configured and one RMEP discovered. ME3 sees no issue since it isn't actively verifying the presence/absence of MPIDs 99 and 211.

```
! ME3
ethernet cfm domain PROVIDER level 3
service SP_STATIC vlan 11
no continuity-check static rmep
```

```
mep mpid 99
```

```
ME1#show ethernet cfm errors
```

```
[no output]
```

```
ME3#show ethernet cfm errors
```

```
[no output]
```

We can re-run the manual cross-check on ME3 to discover this fault. The cross-check process probes for MPIDs 99 and 211 but only finds 211. MPID 99 was expected, but not seen, within the specific service instance. This raises the AIS condition for the local MPID, and the cross-check show command shows the reachability of each remote MEP.

```
ME3#ethernet cfm mep crosscheck enable domain PROVIDER service SP_STATIC
```

```
00:47:57: %E_CFM-6-ENTER_AIS: local mep with mpid 213 level 3 VLAN 11 dir U  
Interface Fa0/9 enters AIS defect condition
```

```
00:47:57: %E_CFM-3-CROSSCHECK_MEP_MISSING: The device does not receive any CC  
messages from a remote MEP with mpid 99 MA name SP_STATIC that it was  
expecting to be part of the vlan 11 in the domain PROVIDER.
```

```
ME3#show ethernet cfm maintenance-points remote crosscheck
```

MPID	Domain Name	Lvl	Type Id	Mep-Up
	MA Name			
	Local MEP Info			

211	PROVIDER	3	Vlan 11	Yes
	SP_STATIC			
	N/A			
99	PROVIDER	3	Vlan 11	No
	SP_STATIC			
	N/A			

Checking the errors on ME3, we can see there is a remote MEP missing. As seen earlier, we know this is as DefRemoteCCM condition behind the scenes. We also see that since AIS was raised on ME3, the RDI bit is set on CCMs going to ME1, who logs this as a DefRDICCM alarm. ME1's reception of RDI CCMs from ME3 isolates the fault as being local to ME3.

```
ME3#show ethernet cfm errors domain-id PROVIDER service SP_STATIC
```

MPID	Domain Id	Mac Address	Type	Id
	MA Name	Reason	Lvl	Age
	Local MEP Info			

99	PROVIDER	0000.0000.0000	Vlan 11	
	SP_STATIC	RMEP missing	3	182s
MPID: 213 Domain: PROVIDER MA: SP_STATIC				


```
ME1#show ethernet cfm errors domain-id PROVIDER service SP_STATIC
```

```
-----
MPID Domain Id                               Mac Address      Type Id
MA Name                                       Reason           Lvl  Age
Local MEP Info
-----
213  PROVIDER                                001f.9d0b.1680   Vlan 11
     SP_STATIC                               Receive RDI      3    0s
     MPID: 211 Domain: PROVIDER MA: SP_STATIC
-----
```

In this way, cross-check can be useful for a user-initiated check of the network if a platform doesn't support the automatic CCM version. Before continuing, I remove MPID 99 from ME3 and re-enable the static RMEP checks for CCM under the SP_STATIC service instance. This clears all errors.

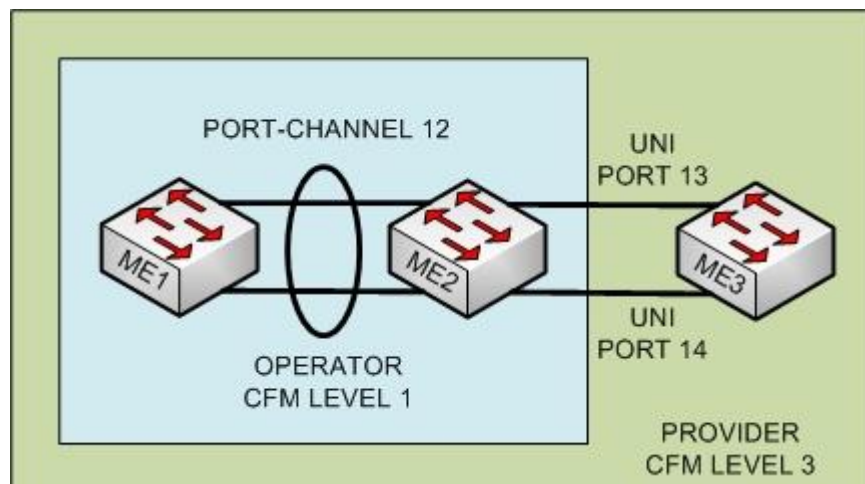
```
ME3#show ethernet cfm errors domain-id PROVIDER service SP_STATIC
[no output]
```

```
ME3#show ethernet cfm errors domain-id PROVIDER service SP_STATIC
[no output]
```

The configurations for CFM are rolled into the next section since the technologies are complementary.

48.3.2 Ethernet OAM (IEEE 802.3ah)

Although "OAM" is a general term that encompasses many technologies, there actually is a feature called "Ethernet OAM". This is defined in IEEE 802.3ah and is meant mostly for link-monitoring and fault detection. Unlike CFM, which is end-to-end fault management (service-oriented), this is enabled on a per-link basis so there is no flooding of OAM PDUs past a local link. The configuration and verification is much less involved than CFM. The feature is not supported on Etherchannels, so in continuing from the CFM lab, we will enable it on ME2 and ME3 ports 13 and 14. OAM and CFM are not mutually exclusive and both can exist together. The CFM network diagram is shown again for reference. No adjustments to CFM are made.



Enabling the feature is a single command at the link level shown below.

```
! ME2 and ME3
interface range FastEthernet0/13 - 14
  ethernet oam
```

One nice behavior of this feature is that it notifies you when peers are discovered. After configuring it, two syslog messages appear notifying us that the OAM session was created as highlighted in yellow. If we remove the configuration (not shown), we get other syslog messages notifying us about the “down” event as well, shown in green. The “enter session” logs only show up when an OAM neighbor is established, which is similar to most other protocols (IGP, BGP, CFM, etc).

```
! ME2 and ME3
%ETHERNET_OAM-6-ENTER_SESSION: The client on interface Fa0/13 has entered the
OAM session.
%ETHERNET_OAM-6-ENTER_SESSION: The client on interface Fa0/14 has entered the
OAM session.

%ETHERNET_OAM-6-EXIT_SESSION: The client on interface Fa0/13 has left the OAM
session.
%ETHERNET_OAM-6-EXIT_SESSION: The client on interface Fa0/14 has left the OAM
session.
```

We can begin with basic verifications. On ME2, we can see OAM peers on ports 13 and 14 which use their port MAC addresses. This is basically the OAM neighbor table. OUI “00000C” represents Cisco as a vendor, and the list of supported capabilities is shown as well.

```
ME2#show ethernet oam summary
Symbols:          * - Master Loopback State,  # - Slave Loopback State
                  & - Error Block State
Capability codes: L - Link Monitor,  R - Remote Loopback
                  U - Unidirection,  V - Variable Retrieval
```

Local Interface	MAC Address	Remote OUI	Mode	Capability
Fa0/13	001f.9d0b.168f	00000C	active	L
Fa0/14	001f.9d0b.1690	00000C	active	L

To understand how OAM discovery works, we will configure another SPAN session on ME2 to capture traffic on these ports. Once we find the OAM packets, we can see that the source MAC is ME2’s MAC on each of the ports (shown below). The destination MAC is 0180.c200.0002 which, unlike CFM, is processed by the MAC sublayer and cannot be forwarded past a switch that can understand the frame. This is why OAM is link-level; the filtering is done at a very low level in the OSI model. The OAM PDU itself contains many flags the show the link status, each represented as a bit shown in the capture

below. The only thing of interest, at present, is that the link is stable at the both local and remote ends. Because there is only a concept of local/remote, OAM is not supported on half-duplex or shared links; only full-duplex P2P Ethernet links are supported.

```
! ME2
monitor session 2 source interface Fa0/13 - 14
monitor session 2 destination interface Fa0/21 encapsulation replicate
```

No.	Time	Source	Destination	Protocol	Length	Info
	2.160127000	04:c5:a4:53:d8:0f	01:80:c2:00:00:02	OAM	60	OAMPDU: Information
	2.160166000	04:c5:a4:53:d8:10	01:80:c2:00:00:02	OAM	60	OAMPDU: Information

Frame 41: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0						
Ethernet II, Src: 04:c5:a4:53:d8:0f (04:c5:a4:53:d8:0f), Dst: 01:80:c2:00:00:02 (01:80:c2:00:00:02)						
OAM Protocol						
Slow Protocols subtype: OAM (0x03)						
Flags: 0x0050 (local: Discovery complete, remote: Discovery complete)						
.... ...0 = Link Fault: False						
.... ..0. = Dying Gasp: False						
.... .0.. = Critical Event: False						
.... 0... = Local Evaluating: False						
...1 = Local Stable: True						
..0. = Remote Evaluating: False						
.1.. = Remote Stable: True						
OAMPDU code: Information (0x00)						
+ Type: Local Information TLV (0x01)						
+ Type: Remote Information TLV (0x02)						

Debugging received OAM PDUs on ME2 reveals the rate at which packets are sent. This uses the slow protocols ethertype (0x8809, not shown) which is different than other Ethernet protocols with customer ethertypes. The default, the rate is 1 packet per second, and can be reduced to 10 packets per second (100ms interval). We will adjust these timers later. We can prove this with a show command revealing the interface status.

```
ME2#debug ethernet oam packet rx
Ingress packets debugging is on
```

```
! ME2
01:44:26: EOAM RX PAK(Fa0/13): infotlv w/ same revision
01:44:26: EOAM RX PAK(Fa0/14): infotlv w/ same revision
01:44:27: EOAM RX PAK(Fa0/13): infotlv w/ same revision
01:44:27: EOAM RX PAK(Fa0/14): infotlv w/ same revision
01:44:28: EOAM RX PAK(Fa0/13): infotlv w/ same revision
01:44:28: EOAM RX PAK(Fa0/14): infotlv w/ same revision
```

```
ME2#show ethernet oam status interface fast0/13
FastEthernet0/13
General
-----
Admin state:          enabled
Mode:                 active
PDU max rate:         10 packets per second
```

```
PDU min rate:      1 packet per 1 second
Link timeout:      5 seconds
[snip]
```

I will quickly adjust the OAM PDU rate on port 13 on both switches. I will set the min-rate to 2, which means 1 packet per 2 seconds, or a 2 second interval. Debugging received OAM PDUs, we should see two PDUs received on port 14 for every one received on port 13. I also adjust the max-rate which would be used for fast PDU flooding during specific events, halved from 10 pps to 5 pps. The link-timeout is set to 10 seconds as well. In cases the configuration was confusing, the show command clearly details what the numerical inputs accomplish.

```
! ME2 and ME3
interface FastEthernet0/13
  ethernet oam min-rate 2
  ethernet oam max-rate 5
  ethernet oam timeout 10

ME2#show ethernet oam status interface fast0/13
FastEthernet0/13
General
-----
Admin state:      enabled
Mode:             active
PDU max rate:     5 packets per second
PDU min rate:     1 packet per 2 seconds
Link timeout:     10 seconds
```

```
ME2#debug ethernet oam packet rx
Ingress packets debugging is on
ME2#
01:54:16: EOAM RX PAK(Fa0/13): infotlv w/ same revision
01:54:17: EOAM RX PAK(Fa0/14): infotlv w/ same revision
01:54:18: EOAM RX PAK(Fa0/14): infotlv w/ same revision
01:54:18: EOAM RX PAK(Fa0/13): infotlv w/ same revision
01:54:19: EOAM RX PAK(Fa0/14): infotlv w/ same revision
01:54:20: EOAM RX PAK(Fa0/14): infotlv w/ same revision
```

For more details on the OAM neighbors, we can check the discovery details. Somewhat similar to the MPLS LDP discovery show commands, this shows both local and remote information on a specific interface. From ME2, we see more verbose descriptions of the supported capabilities and OUIs, along with MTU settings and revision information. The PDU revision increments every time a change is made to the discovery PDU, such as active/passive mode (seen later), MTU adjustments, etc. The MTU is 1518 to account for Ethernet (14 bytes) and dot1q (4 bytes) encapsulation. There is no reason to change it in this setup, although the MTU can be mismatched between peers.

```
ME2#show ethernet oam discovery interface fast0/13
```

FastEthernet0/13

Local client

Administrative configurations:

Mode: active
Unidirection: not supported
Link monitor: supported (on)
Remote loopback: not supported
MIB retrieval: not supported
Mtu size: 1518

Operational status:

Port status: operational
Loopback status: no loopback
PDU revision: 0

Remote client

MAC address: 001f.9d0b.168f

Vendor(oui): 00000C(cisco)

Administrative configurations:

PDU revision: 0
Mode: active
Unidirection: not supported
Link monitor: supported
Remote loopback: not supported
MIB retrieval: not supported
Mtu size: 1518

The runtime show command is somewhat cryptic but useful. On a particular interface, we can see low-level details regarding some of the bits seen in the OAM PDU above. For example, local stable and remote stable are both “yes”, which corresponds with the Wireshark capture. The state machine at the bottom reveals very low-level details, so I don’t discuss it.

ME2#show ethernet oam runtime interface fast0/13

Runtime Settings:

local_pdu: any
local_mux: fwd
local_par: fwd
local_link_status: OK
local_satisfied: Yes
local_stable: Yes
enter_loopback: No
pdu_cnt: 10
pdu_timer: Running
lost_link_timer: Running

```
loopback_timer: stopped(timeout=2)
remote_state_valid: Yes
remote_stable: Yes
remote_evaluating: 0
remote_rfi: 0
peer version: 3
```

State Machine:

```
-----
sm(ether_oam_port Fa0/13), running yes, state SEND_ANY
Last transition recorded: (remote_state_valid)-> SEND_LOCAL_REMOTE
(local_satisfied)-> SEND_LOCAL_REMOTE_OK (remote_stable)-> SEND_ANY (end)->
INACTIVE (begin)-> FAULT (begin)-> FAULT2 (mode_active)-> ACTIVE_SEND_LOCAL
(remote_state_valid)-> SEND_LOCAL_REMOTE (local_satisfied)->
SEND_LOCAL_REMOTE_OK (remote_stable)-> SEND_ANY
```

Ethernet OAM maintains very detailed statistics as well. The output is very long so I truncate it, but we can see that it tracks many PDU types and events. We will use pieces of this output later during our advanced testing.

```
ME2#show ethernet oam statistics interface fast0/13
```

```
FastEthernet0/13
```

```
Counters:
```

```
-----
Information OAMPDU Tx           : 1418
Information OAMPDU Rx           : 1419
Unique Event Notification OAMPDU Tx      : 0
Unique Event Notification OAMPDU Rx      : 0
Duplicate Event Notification OAMPDU TX    : 0
Duplicate Event Notification OAMPDU RX    : 0
Loopback Control OAMPDU Tx             : 0
Loopback Control OAMPDU Rx             : 0
[snip]
```

Another minor option involves active and passive modes. Just like LACP, active mode sends OAM PDUs to try and establish a session, while passive mode only responds. At least one of the OAM peers on a link must be active, which is the default mode. I configure ME3 port 13 as passive just for variety. We can verify this from either ME3 or ME2 since the operating mode is carried in the OAM discovery PDUs. I demonstrate both verifications methods below.

```
! ME3
```

```
interface FastEthernet0/13
```

```
    ethernet oam mode passive
```

```
ME3#show ethernet oam discovery interface fast0/13 | begin ^Local
```

```
Local client
```

```
-----
```

Administrative configurations:

```
Mode: passive
Unidirection: not supported
Link monitor: supported (on)
Remote loopback: not supported
MIB retrieval: not supported
Mtu size: 1518
```

```
ME2#show ethernet oam discovery interface fast0/13 | begin ^Remote
Remote client
```

```
MAC address: 001f.9d0b.168f
Vendor(oui): 00000C(cisco)
```

Administrative configurations:

```
PDU revision: 1
Mode: passive
Unidirection: not supported
Link monitor: supported
Remote loopback: not supported
MIB retrieval: not supported
Mtu size: 1518
```

Next, we will configure some advanced OAM features. The OAM link-monitor is self-explanatory and is used to check the quality of a given link. It is enabled by default with several tunable parameters. This is also shown under the “status” for a link after the general information (timers, etc) seen earlier. There are low-level details for tuning error thresholds with respect to link quality. When the low-threshold is crossed, an event notification PDU is sent to the OAM peer to notify them of the fault. When the high-threshold is crossed, more extreme action (like disabling an interface) can occur. The defaults are shown below; the low-thresholds are very aggressive so all faults are reported, but all high thresholds are blank so no extreme measures are taken.

```
ME2#show ethernet oam status interface fast0/13 | begin Link_Mon
Link Monitoring
```

```
Status: supported (on)
```

Symbol Period Error

```
Window: 100 x 1048576 symbols
Low threshold: 1 error symbol(s)
High threshold: none
```

Frame Error

```
Window: 10 x 100 milliseconds
Low threshold: 1 error frame(s)
High threshold: none
```

Frame Period Error

Window: 1000 x 10000 frames
Low threshold: 1 error frame(s)
High threshold: none

Frame Seconds Error

Window: 100 x 100 milliseconds
Low threshold: 1 error second(s)
High threshold: none

Receive-Frame CRC Error

Window: 10 x 100 milliseconds
Low threshold: 10 error frame(s)
High threshold: none

Transmit-Frame CRC Error: Not Supported

Simulating these link failures will be very difficult without advanced test equipment, so we will limit our verifications to show commands. Although unnecessary, I configured every possible option just for the sake of doing it once. This does not have to match between OAM peers, so it only configure it on ME3. I also show the context-sensitive help which describes what each fault condition means. The most significant command below is the ability to err-disable an interface that crosses a high threshold, and is highlighted.

```
ME3(config-if)#ethernet oam link-monitor ?
frame           Errored frames in a period measured by time
frame-period    Errored frames in a period measured by number of frames
frame-seconds   Errored frame seconds in a period
high-threshold  Additional high threshold configuration
on              Enable link monitoring on the interface
receive-crc     Received frames with CRC errors in a period
supported       Support link monitoring on the interface
symbol-period   Errored symbols in a period
```

```
! ME3
interface FastEthernet0/13
 ethernet oam link-monitor symbol-period window 32
 ethernet oam link-monitor symbol-period threshold low 12
 ethernet oam link-monitor symbol-period threshold high 24
 ethernet oam link-monitor frame window 200
 ethernet oam link-monitor frame threshold low 2
 ethernet oam link-monitor frame threshold high 50
 ethernet oam link-monitor frame-period window 40000
 ethernet oam link-monitor frame-period threshold low 50
 ethernet oam link-monitor frame-period threshold high 750
 ethernet oam link-monitor frame-seconds window 500
 ethernet oam link-monitor frame-seconds threshold low 35
 ethernet oam link-monitor frame-seconds threshold high 40
 ethernet oam link-monitor receive-crc window 300
```



```
ethernet oam link-monitor receive-crc threshold low 2
ethernet oam link-monitor receive-crc threshold high 10
ethernet oam link-monitor high-threshold action error-disable-interface
```

A brief verification of these minor options is shown below. I don't show the full output, but enough is revealed to see the changes were accepted.

```
ME3#show ethernet oam status interface fast0/13 | begin Link_Mon
Link Monitoring
-----
Status: supported (on)

Symbol Period Error
Window:                32 x 1048576 symbols
Low threshold:         12 error symbol(s)
High threshold:        24 error symbol(s)

Frame Error
Window:                200 x 100 milliseconds
Low threshold:         2 error frame(s)
High threshold:        50 error frame(s)

Frame Period Error
Window:                40000 x 10000 frames
Low threshold:         50 error frame(s)
High threshold:        750 error frame(s)
[snip]
```

Another useful feature is the remote-failure indication (RFI) mechanism. When an OAM speaker discovers a loss of signal (link-fault), dying gasp (unrecoverable condition, such as the peer disabling OAM), or critical-event (vendor specific event), this is communicated to the OAM peer. This particular platform (ME3400) cannot generate link-fault or critical-event PDUs, but can process them upon receipt. It can, however, generate dying-gasp PDUs to show when OAM is removed from an interface, an interface is shutdown, a switch reloads, or some other catastrophic condition that will undoubtedly break OAM. We will configure ME3 port 14 to process all of these events and errdisable the interface as needed.

```
! ME3
interface FastEthernet0/14
 ethernet oam remote-failure link-fault action error-disable-interface
 ethernet oam remote-failure dying-gasp action error-disable-interface
 ethernet oam remote-failure critical-event action error-disable-interface
```

Because the RFI action is errdisable, this implies we could potentially configure auto-recovery for it. The err-disable recovery feature isn't specific to OAM (it supports many applications) but I enable it on ME3

for completeness. If a high-threshold was crossed, the link remains down for 600 seconds (10 minutes) and then the OAM process attempts to recover it.

```
! ME3
errdisable recovery cause oam-remote-failure
errdisable recovery interval 600

ME3#show errdisable detect | include oam
oam-remote-failure      Enabled      port

ME3#show errdisable recovery | exclude Disabled
ErrDisable Reason      Timer Status
-----
oam-remote-failure      Enabled

Timer interval: 600 seconds
```

We can check the statistics for port 14 on ME3 and see that there have been no RFIs so far. This is an indication that the remote peer (ME2 port 14) has not suffered/seen any issues.

```
ME3#show ethernet oam statistics interface fast0/14 | begin Remote_Fault
Remote Faults:
-----
  0 Link Fault records
  0 Dying Gasp records
  0 Critical Event records
```

We can simulate this by removing the OAM configuration from port 14 on ME2 (or shut the interface down, or reload the switch, etc). Before we do that, I enabled state machine (sm) debugging on ME3. We still have Wireshark capturing traffic on ports 13 and 14 as well. ME3 also prints well-written syslog messages which clearly show the problem. The errdisable process shuts down the interface and the line protocol follows.

```
! ME2
interface FastEthernet0/14
 ethernet oam
```

```
! ME3
02:31:45: %ETHERNET_OAM-6-RFI: The client on interface Fa0/14 has received a
remote failure indication from its remote peer(failure reason = remote client
administratively turned off action = error-disable)
02:31:45: %PM-4-ERR_DISABLE: oam-remote-failure error detected on Fa0/14,
putting Fa0/14 in err-disable state
02:31:46: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/14,
changed state to down
02:31:46: %ETHERNET_OAM-6-EXIT_SESSION: The client on interface Fa0/14 has
left the OAM session.
```

In about 5 more minutes (I waited 5 already), the port will attempt to come back online. We can see the cause is due to an OAM RFI, which was the “dying-gasp” message sent from ME2.

```
ME3#show interfaces fast0/14 status
Port      Name      Status      Vlan      Duplex  Speed Type
Fa0/14    TO ME2    err-disabled trunk      auto    auto  10/100BaseTX
```

```
ME3#show errdisable recovery | begin next
Interfaces that will be enabled at the next timeout:
```

Interface	Errdisable reason	Time left(sec)
-----	-----	-----
Fa0/14	oam-remote-failure	319

The debugs reveal nothing of value, other than the port state changing from SEND_ANY to INACTIVE, which indicates a failure of some kind. The syslog messages were more informative, though.

```
ME3#debug ethernet oam sm
State machine changes debugging is on

! ME3
02:31:46:      ether_oam_port Fa0/14: during state SEND_ANY, got event 1(end)
02:31:46: @@@ ether_oam_port Fa0/14: SEND_ANY -> INACTIVE
```

Checking the interface statistics, we can see there was a single dying-gasp received on this interface, along with the timestamp. This is useful in case logging is disabled or for large-scale failures. Notice that there were not any link-fault or critical-event records; we will not be able to generate those in our ME3400 environment.

```
ME3#show ethernet oam statistics interface fast0/14 | begin Remote_Fault
Remote Faults:
-----
 0 Link Fault records
 1 Dying Gasp records
   Total dying gasps      : 1
   Time stamp              : 02:19:41

 0 Critical Event records
```

The capture below shows the final OAM PDU sourced from ME2 port 14 before the port stops running OAM. This is similar to how many IGPs work; when the configuration is removed or the port is gracefully shut down, they may notify their neighbors immediately. The OAM PDU below shows the dying-gasp flag set to 1, which ME3 interprets as an RFI.

No.	Time	Source	Destination	Protocol	Length	Info
	2.715417000	04:c5:a4:53:d8:10	01:80:c2:00:00:02	OAM	60	OAMPDU: Organization Specific
<						
Frame 67: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0						
Ethernet II, Src: 04:c5:a4:53:d8:10 (04:c5:a4:53:d8:10), Dst: 01:80:c2:00:00:02 (01:80:c2:00:00:02)						
OAM Protocol						
Slow Protocols subtype: OAM (0x03)						
Flags: 0x0052 (Dying Gasp, local: Discovery complete, remote: Discovery complete)						
....0 = Link Fault: False						
....1 = Dying Gasp: True						
....0 = Critical Event: False						
...0... = Local Evaluating: False						
...1... = Local Stable: True						
..0. = Remote Evaluating: False						
.1. = Remote Stable: True						
OAMPDU code: Organization Specific (0xfe)						
organizationally unique identifier: 00000c (00:00:0c)						

Even by adding “ethernet oam” back onto ME2 port 14, ME3 remains in errdisable until the recovery timer expires. This prevents against constant interface flaps, which can negatively affect STP, REP, G.8032, or any other layer-2 loop prevention mechanism. We could manually shutdown/no shutdown the interface on ME3, or wait until the timer expires and the OAM neighbor forms again. Although not shown here, CFM recovers as well, but I do not show the logs for clarity. I quickly check the OAM status to ensure I see both ports operational.

```
! ME3
02:41:45: %PM-4-ERR_RECOVER: Attempting to recover from oam-remote-failure
err-disable state on Fa0/14
02:41:49: %LINK-3-UPDOWN: Interface FastEthernet0/14, changed state to up
02:41:50: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/14,
changed state to up
02:41:51: %ETHERNET_OAM-6-ENTER_SESSION: The client on interface Fa0/14 has
entered the OAM session.
```

```
ME3#show ethernet oam summary | begin ^Interface
Interface      MAC Address      OUI      Mode      Capability

Fa0/13          04c5.a453.d80f  00000C  active    L
Fa0/14          04c5.a453.d810  00000C  active    L
```

Assuming these RFI and link-monitoring parameters were standard for all OAM links, this would be challenging to configure on many interfaces, especially if the threshold values are tuned regularly. We can define an OAM template to define it once, and then apply it to interfaces. I create a short dummy template on ME3, and then apply it to a dummy interface as an example. This is just a configuration optimization and has the same effect as the things we tested above from an operational perspective. The templates work just like MPLS L2VPN templates, BGP templates, or any other kind of inheritance-based structure in any Cisco CLI.

```
! ME3
template TMP_OAM
ethernet oam link-monitor frame threshold high 1000
```

```
ethernet oam link-monitor high-threshold action error-disable-interface
ethernet oam remote-failure dying-gasp action error-disable-interface
```

```
interface FastEthernet0/3
description DUMMY FOR OAM TEMPLATES
port-type nni
source template TMP_OAM
ethernet oam
```

We know from earlier that checking the interface status will reveal these detailed parameters as shown again below. We can also verify it by looking at the template and seeing that OAM parameters are configured. It also lists the interfaces to which the template is bound.

```
ME3#show ethernet oam status interface fast0/3
FastEthernet0/3
General
-----
Admin state:          enabled
Mode:                 active
PDU max rate:         10 packets per second
PDU min rate:         1 packet per 1 second
Link timeout:         5 seconds
High threshold action: error disable interface
Link fault action:    no action
Dying gasp action:    error disable interface
Critical event action: no action
[snip]
Frame Error
Window:               10 x 100 milliseconds
Low threshold:        1 error frame(s)
High threshold:       1000 error frame(s)
[snip]
```

```
ME3#show template TMP_OAM
Template      class/type      Component(s)
TMP_OAM      owner          Ethernet OAM
BOUND: Fa0/3
```

The last OAM feature is remote loopback. Like link-monitoring, this is a capability that is exchanged in OAM PDUs. Unlike link-monitoring, it is disabled by default, so I enable it on ME2 and ME3 port 13 with a timeout of 5 seconds, which means that inactivity after 5 seconds will disable the loopback session. Both switches see their peers on port 13 with the “R” capability, which is remote loopback. The feature is disabled by default because it normally makes sense to configure on an as-needed basis for security purposes. The feature must be configured on both ends of a link to operate correctly.

```
! ME2 and ME3
interface FastEthernet0/13
```

```
ethernet oam remote-loopback supported
ethernet oam remote-loopback timeout 5
```

```
ME3#show ethernet oam summary
```

```
Symbols:      * - Master Loopback State, # - Slave Loopback State
              & - Error Block State
Capability codes: L - Link Monitor, R - Remote Loopback
              U - Unidirection, V - Variable Retrieval
```

Local Interface	MAC Address	Remote OUI	Mode	Capability
Fa0/13	04c5.a453.d80f	00000C	active	L R
Fa0/14	04c5.a453.d810	00000C	active	L

```
ME2#show ethernet oam summary
```

```
Symbols:      * - Master Loopback State, # - Slave Loopback State
              & - Error Block State
Capability codes: L - Link Monitor, R - Remote Loopback
              U - Unidirection, V - Variable Retrieval
```

Local Interface	MAC Address	Remote OUI	Mode	Capability
Fa0/13	001f.9d0b.168f	00000C	passive	L R
Fa0/14	001f.9d0b.1690	00000C	active	L

The command also warns us that the command will break traffic flow, as traditional link-level loopbacks would. Notice that only active clients (not passive) can start remote-loopback sessions, so this is a drawback to using OAM passing mode. Since we configured ME3 port 13 as a passive interface, it cannot initiate this session.

```
ME3#ethernet oam remote-loopback start interface fast0/13
```

```
This is a intrusive loopback.
```

```
Therefore, while you test Ethernet OAM MAC connectivity,
you will be unable to pass traffic across that link.
```

```
Proceed with Remote Loopback? [confirm]y
```

```
%Remote loopback must be started from an active Ethernet OAM client.
```

When we issue the same command on ME2, the remote loopback succeeds. The loopback process starts and a syslog message is generated to inform us that ME2 is the “master” of the loopback session.

```
ME2#ethernet oam remote-loopback start interface fast0/13
```

```
This is a intrusive loopback.
```

```
Therefore, while you test Ethernet OAM MAC connectivity,
you will be unable to pass traffic across that link.
```

```
Proceed with Remote Loopback? [confirm]y
```

```
%ETHERNET_OAM-6-LOOPBACK: Interface Fa0/13 has entered the master loopback mode.
```

On ME3, we see a syslog message saying that it entered remote-loopback in slave mode, which means the loopback process is controlled by ME2.

```
! ME3
```

```
%ETHERNET_OAM-6-LOOPBACK: Interface Fa0/13 has entered the slave loopback mode.
```

Ignoring all of the angry syslog messages from CFM telling us the link failed, we can see that OAM loopback is working. Checking the OAM summary on both switches, we can see ME2 identifies its relationship to the OAM peer on port 13 (ME3) as “master loopback”. ME3 does the same except uses “slave loopback”.

```
ME2#show ethernet oam summary
```

```
Symbols:      * - Master Loopback State, # - Slave Loopback State
              & - Error Block State
Capability codes: L - Link Monitor, R - Remote Loopback
                  U - Unidirection, V - Variable Retrieval
```

Local Interface	MAC Address	Remote OUI	Mode	Capability
* Fa0/13	001f.9d0b.168f	00000C	passive	L R
Fa0/14	001f.9d0b.1690	00000C	active	L

```
ME3#show ethernet oam summary
```

```
Symbols:      * - Master Loopback State, # - Slave Loopback State
              & - Error Block State
Capability codes: L - Link Monitor, R - Remote Loopback
                  U - Unidirection, V - Variable Retrieval
```

Local Interface	MAC Address	Remote OUI	Mode	Capability
# Fa0/13	04c5.a453.d80f	00000C	active	L R
Fa0/14	04c5.a453.d810	00000C	active	L

We can verify this further by checking the discovery details on both interfaces. ME2, as the master, shows the port in “remote loopback” mode while ME3, as the slave, shows it in “local loopback” mode.

```
ME2#show ethernet oam discovery interface f0/13
```

```
FastEthernet0/13
```

```
Local client
```

```
-----
```

```
Administrative configurations:
```

```
Mode:                active
Unidirection:        not supported
Link monitor:         supported (on)
Remote loopback:      supported
MIB retrieval:        not supported
Mtu size:             1518
```

Operational status:

```
Port status:         operational
Loopback status:      remote loopback
PDU revision:         6
```

```
ME3#show ethernet oam discovery interface f0/13
```

```
FastEthernet0/13
```

```
Local client
```

```
-----
```

Administrative configurations:

```
Mode:                passive
Unidirection:        not supported
Link monitor:         supported (on)
Remote loopback:      supported
MIB retrieval:        not supported
Mtu size:             1518
```

Operational status:

```
Port status:         operational
Loopback status:      local loopback
PDU revision:         7
```

At this point, frames sent from ME2 will be looped back to ME2 via ME3. This is true for all frames that are not OAM PDUs or pause frames. At this point, the looped back frames can be used for measuring latency, jitter, throughput, loss, etc. We can disable the loopback feature using the command below. Syslog messages are generated on both ends to show the loopback being deactivated.

```
ME2#ethernet oam remote-loopback stop interface fast0/13
```

```
%ETHERNET_OAM-6-LOOPBACK: Interface Fa0/13 has exited the master loopback mode.
```

```
! ME3
```

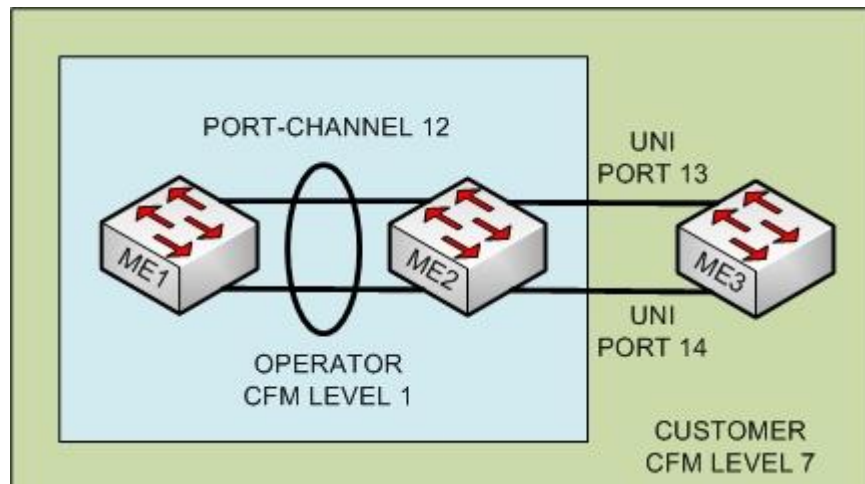
```
%ETHERNET_OAM-6-LOOPBACK: Interface Fa0/13 has exited the slave loopback mode.
```

Additional Reading – Reference configurations "cfm-oam"

48.3.3 Ethernet Local Management Interface (E-LMI) (MEF.16)

Ethernet LMI (E-LMI) is used exclusively between UPE and CE devices within the carrier Ethernet network. This offers the CE access to configuration parameters on the UPE, to include status messages

and error reporting. It interworks seamlessly with CFM and the two are generally configured together. Because CFM is end-to-end, we will configure that as well, and then overlay E-LMI on the UPE-CE link. The OAM network is similar to the CFM/OAM test from earlier, except with some key modifications. ME3 is now a CE device with ME2 as the UPE as shown in the diagram below.



ME2's links to ME3 are the UPE-CE links, each one providing services for a different VLAN. Port 13 uses VLAN 13 only and port 14 uses VLAN 14 only for simplicity. The operator CFM network is largely unchanged, but the all-encompassing CFM domain is a level 7 "customer" network. I could have used the level 3 provider network again, but I wanted it to be somewhat realistic in terms of configuration and naming. The UPE ports are MEPs for the operator's network which offer service for VLAN 10; this is used within the operator's network only. ME1 simulates the remote UPE and has two MEPs within the customer domain, one per customer VLAN (13 and 14). The vast majority of the Ethernet OAM configuration was removed for simplicity and a basic OAM configuration was added back to port 13 only. Although not required, I raise the MIP level on the UPE ports to 7 so that there is no chance the customer uses a level less than 7. This helps guard against cross-connect issues deeper in the network. The MIPs on the UPEs only operate at that level for the specific C-VLANs being serviced. VLAN 10 is allowed on the trunk so that CFM works correctly; if VLAN 10 wasn't available on the UPE, ME2 would not send CCMs since the service cannot work. The CFM configuration is shown below but not discussed.

```
! ME1 and ME3
ethernet cfm domain CUSTOMER level 7
  service number 13 vlan 13
    continuity-check
  service number 14 vlan 14
    continuity-check

! ME1 only
interface FastEthernet0/9
  ethernet cfm mep domain CUSTOMER mpid 113 service number 13
    alarm notification all
  ethernet cfm mep domain CUSTOMER mpid 114 service number 14
```

```

    alarm notification all
    ethernet cfm mep domain OPERATOR mpid 101 service OP1

! ME3 only
interface FastEthernet0/9
    ethernet cfm mep domain CUSTOMER mpid 313 service number 13
    alarm notification all
    ethernet cfm mep domain CUSTOMER mpid 314 service number 14
    alarm notification all
    ethernet cfm mep domain OPERATOR mpid 101 service OP1

interface FastEthernet0/13
    switchport trunk allowed vlan 13
    ethernet cfm mip level 7 vlan 13
    ethernet oam

interface FastEthernet0/14
    switchport trunk allowed vlan 14
    ethernet cfm mip level 7 vlan 14

! ME2
interface FastEthernet0/13
    description TO ME3 (CUSTOMER VLAN 13)
    switchport trunk allowed vlan 10,13
    switchport mode trunk
    ethernet cfm mip level 7 vlan 13
    ethernet cfm mep domain OPERATOR mpid 102 service OP1
    ethernet oam

interface FastEthernet0/14
    description TO ME3 (CUSTOMER VLAN 14)
    switchport trunk allowed vlan 10,14
    switchport mode trunk
    ethernet cfm mip level 7 vlan 14
    ethernet cfm mep domain OPERATOR mpid 112 service OP1

```

So far, none of this is new, so I quickly verify the CFM and OAM components. Within the operator network on ME1, we can see 2 remote MEPs. Each one represents a UPE, and there are no errors associated with these MEPs. A pair of Ethernet pings confirms that both are reachable.

```
ME1#show ethernet cfm maintenance-points remote domain OPERATOR
```

MPID	Domain Name	MacAddress	IfSt	PtSt
Lvl	Domain ID	Ingress		
RDI	MA Name	Type Id	SrvcInst	
	EVC Name		Age	
	Local MEP Info			
102	OPERATOR	04c5.a453.d800	Up	Up

```

1    OPERATOR                                Po12
-    OP1                                    Vlan 10          N/A
      N/A                                    5s
      MPID: 101 Domain: OPERATOR MA: OP1
112 OPERATOR                                04c5.a453.d800   Up    Up
1    OPERATOR                                Po12
-    OP1                                    Vlan 10          N/A
      N/A                                    4s
      MPID: 101 Domain: OPERATOR MA: OP1

```

ME1#show ethernet cfm errors domain-id OPERATOR

[no output]

ME1#ping ethernet mpid 102 domain OPERATOR service OP1

Type escape sequence to abort.

Sending 5 Ethernet CFM loopback messages to 04c5.a453.d800, timeout is 5 seconds:!!!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

ME1#ping ethernet mpid 112 domain OPERATOR service OP1

Type escape sequence to abort.

Sending 5 Ethernet CFM loopback messages to 04c5.a453.d800, timeout is 5 seconds:!!!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/8 ms

Next, we will verify that the customer CFM domain is functional following the same steps. ME1 sees two MEPs within the customer domain, each with a different service number. We check for errors in the customer domain, then ping each MEP to ensure reachability. Again, this is just basic CFM verification.

ME1#show ethernet cfm maintenance-points remote domain CUSTOMER

```

-----
MPID  Domain Name          MacAddress      IfSt  PtSt
Lvl   Domain ID            Ingress
RDI   MA Name              Type Id
      EVC Name              SrvcInst
      Local MEP Info        Age
-----
313   CUSTOMER              001f.9d0b.1680  Up    Up
7     CUSTOMER              Po12
-     number 13             Vlan 13          N/A
      N/A                    8s
      MPID: 113 Domain: CUSTOMER MA: number 13
314   CUSTOMER              001f.9d0b.1680  Up    Up
7     CUSTOMER              Po12
-     number 14             Vlan 14          N/A
      N/A                    6s
      MPID: 114 Domain: CUSTOMER MA: number 14

```

ME1#show ethernet cfm errors domain-id CUSTOMER

[no output]

```
ME1#ping ethernet mpid 313 domain CUSTOMER service number 13
Type escape sequence to abort.
Sending 5 Ethernet CFM loopback messages to 001f.9d0b.1680, timeout is 5
seconds:!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms
```

```
ME1#ping ethernet mpid 314 domain CUSTOMER service number 14
Type escape sequence to abort.
Sending 5 Ethernet CFM loopback messages to 001f.9d0b.1680, timeout is 5
seconds:!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms
```

Next, we will verify the OAM configuration on port 13. OAM PDUs are being exchanged as ME2 sees ME3 as a peer without issue.

```
ME2#show ethernet oam summary
Symbols:          * - Master Loopback State,  # - Slave Loopback State
                  & - Error Block State
Capability codes: L - Link Monitor,    R - Remote Loopback
                  U - Unidirection,    V - Variable Retrieval
```

Local Interface	MAC Address	Remote OUI	Mode	Capability
Fa0/13	001f.9d0b.168f	00000C	active	L

With a functional CFM architecture and OAM between the UPE-CE on a subset of the available services, we will look at E-LMI. For a feature that has limited scope, the configuration is surprisingly involved. The documentation suggests that CFM must be configured (OAM doesn't have to be, though) and the interworking between E-LMI and CFM is required, too. Following that guidance, we configured and verified both CFM and OAM so that the network is prepared for E-LMI. The first step in configuring E-LMI is defining an EVC. This isn't a circuit for carrying traffic per se, but is used to interwork CFM with E-LMI. In this EVC, we specify the OAM protocol of CFM, along with the S-VLAN that has service across the provider network. If the network were larger, this would normally be in a "provider" domain, not an operator one, but the logic is the same. Since VLAN 10 is the service terminated by the operator MEPs on the UPEs (ME1 dummy port and ME2 ports 13 and 14), that is the S-VLAN we specify. The VLAN has transport within the operator domain, and E-LMI needs to know this for CFM interworking. Since we are going to recycle this EVC on two UNI ports, there will be three total UNIs (2 local ports and one remote port). This is a sloppy way to create EVCs as it should be one per "virtual circuit", but I demonstrate this method to show that the UNI count can be changed under EVCs to support multipoint operations.

```
! ME2
ethernet evc EVC_ELMI
uni count 3
oam protocol cfm svlan 10 domain OPERATOR
```

Next, we will enable E-LMI on our UPE interfaces and bind the EVC to a service instance. The service instance maps the C-VLANs to the EVC with the S-VLAN; this is not QinQ or anything like that. This is for signaling only, since the E-LMI between UPE-CE on one end of the network doesn't have visibility to the other side. With CFM, they do. The UNI ID is a required field (string) that is meant to identify this UNI uniquely in the network. I use simple strings "VLAN13" and "VLAN14" for this.

```
! ME3
interface FastEthernet0/13
  ethernet lmi interface
  ethernet uni id VLAN13
  service instance 13 ethernet EVC_ELMI
  ethernet lmi ce-vlan map 13

interface FastEthernet0/14
  ethernet lmi interface
  ethernet uni id VLAN14
  service instance 14 ethernet EVC_ELMI
  ethernet lmi ce-vlan map 14
```

The configuration on the CE side is much simpler. In global configuration, we enable E-LMI for "CE" mode, where "PE" mode is the default. We can also enable E-LMI globally rather than per port, which we do here. We should take care to disable E-LMI explicitly on all other ports that don't need it. The global configuration is probably more appropriate on a UPE with many UNI interfaces, but we can configure it anyway we like.

```
! ME3
ethernet lmi global
ethernet lmi ce

interface range FastEthernet0/7 , FastEthernet0/9, FastEthernet0/11
  no ethernet lmi interface
```

With the configuration complete, we will check the E-LMI status. This is done either in summary form or on a per-EVC basis (detailed). Both outputs are shown below; the "A" stands for active and represents a correct configuration. The detailed version shows additional information like the UNI ID, EVC type, and VLAN ID.

```
ME2#show ethernet lmi evc
```

St	EVC Id	Port
A	EVC_ELMI	Fa0/13,
A		Fa0/14

Key: St=Status, A=Active, P=Partially Active, I=Inactive, ?=Link Down

```
ME2#show ethernet lmi evc detail EVC_ELMI
```

```

EVC Id: EVC_ELMI
interface FastEthernet0/13
  Time since Last Full Report: 00:29:38
  Ether LMI Link Status: Up
  UNI Status: Up
  UNI Id: VLAN13
  CE-VLAN/EVC Map Type: Bundling
  VLAN: 13

EVC Status: Active
EVC Type: Point-to-Point
Remote UNI Count: Configured = 1, Active = 2

UNI Id                UNI Status            Port
-----              -
VLAN14                Up                    Fa0/14

interface FastEthernet0/14
  Time since Last Full Report: 00:29:38
  Ether LMI Link Status: Up
  UNI Status: Up
  UNI Id: VLAN14
  CE-VLAN/EVC Map Type: Bundling
  VLAN: 14

EVC Status: Active
EVC Type: Point-to-Point
Remote UNI Count: Configured = 1, Active = 2

UNI Id                UNI Status            Port
-----              -
VLAN13                Up                    Fa0/13

```

We can also verify the EVC status. Notice the EVC-type is MP2MP but we never specified it. This is the result of poor planning; if the UNI count is greater than 2, IOS assumes this is a multipoint service, when in reality we just reused an EVC configuration on multiple ports. There really isn't any connectivity between the two UPE ports based on the C-VLAN mappings, but we accounted for all three UNIs. Checking the UNI count configured/active is a good way to ensure your configuration is correct.

```

ME2#show ethernet service evc id EVC_ELMI detail
EVC ID: EVC_ELMI
EVC Type: MP-MP
UNI Count: Configured = 3, Active = 3
EVC Status: Active
Associated Local Interfaces:
  Interface            CE-Vlans
  FastEthernet0/13     13
  FastEthernet0/14     14

```

Associated UNIs:

UNI-Identifier	Status	Location
VLAN14	Up	Fa0/14
VLAN13	Up	Fa0/13

With the configuration verified, we will explore the discovery process using debugs and Wireshark. On ME2, we enable E-LMI packet debugging; the debugging is verbose but easily read. We can see “status enquiry” messages being received in ports 13 (yellow) and 14 (green) from the CE. The UPE sends “status” messages in reply which is a different message type. Neither message carries any significant information since these are “check” messages. This is the CE’s way of probing the UPE to ensure it is reachable.

```
ME2#debug ethernet lmi packets
Ethernet LMI packets debugging is on
```

```
! ME2
```

```
06:58:33: ELMI Fa0/13 PACKET: Incoming
```

```
    Protocol Version      : 0x1
    Message                : STATUS ENQ (0x75)
    Report Type            : Check
    Sequence Number        : Snd(0x7D), Rcv(0x7C)
    Data Instance          : Value(0x7)
```

```
06:58:33: ELMI Fa0/13 PACKET: Outgoing
```

```
    Protocol Version      : 0x1
    Message                : STATUS (0x7D)
    Report Type            : Check
    Sequence Number        : Snd(0x7D), Rcv(0x7D)
    Data Instance          : Value(0x7)
```

```
06:58:33: ELMI Fa0/14 PACKET: Incoming
```

```
    Protocol Version      : 0x1
    Message                : STATUS ENQ (0x75)
    Report Type            : Check
    Sequence Number        : Snd(0x80), Rcv(0x7E)
    Data Instance          : Value(0xA)
```

```
06:58:33: ELMI Fa0/14 PACKET: Outgoing
```

```
    Protocol Version      : 0x1
    Message                : STATUS (0x7D)
    Report Type            : Check
    Sequence Number        : Snd(0x7F), Rcv(0x80)
    Data Instance          : Value(0xA)
```

Given the existing SPAN session on ME2, we can see the packets in Wireshark as well. Since Wireshark reveals the MAC addresses, we will confirm them on all 4 ports involved. This helps prove that the E-LMI messages exchanged on a link don’t use the concept of a brain MAC like CFM; the actual link addresses

are used. Additionally, the destination MAC is always 0180.c200.0007, which is unlike CFM where the last 4 bits change depending on the level. The ethertype is always 0x88EE which is specific for E-LMI. Wireshark can understand E-LMI as it parses the frame types and payloads nicely.

```
ME3#show interfaces fastEthernet 0/13 | include bia
Hardware is Fast Ethernet, address is 001f.9d0b.168f (bia 001f.9d0b.168f)
```

```
ME3#show interfaces fastEthernet 0/14 | include bia
Hardware is Fast Ethernet, address is 001f.9d0b.1690 (bia 001f.9d0b.1690)
```

```
ME2#show interfaces fastEthernet 0/13 | include bia
Hardware is Fast Ethernet, address is 04c5.a453.d80f (bia 04c5.a453.d80f)
```

```
ME2#show interfaces fastEthernet 0/14 | include bia
Hardware is Fast Ethernet, address is 04c5.a453.d810 (bia 04c5.a453.d810)
```

No.	Time	Source	Destination	Protocol	Length	Info
	14.982521000	00:1f:9d:0b:16:8f	01:80:c2:00:00:07	E-LMI	60	Status enquiry
	14.982553000	00:1f:9d:0b:16:90	01:80:c2:00:00:07	E-LMI	60	Status enquiry
	14.983551000	04:c5:a4:53:d8:0f	01:80:c2:00:00:07	E-LMI	60	Status
	14.983799000	04:c5:a4:53:d8:10	01:80:c2:00:00:07	E-LMI	60	Status

<

+

Frame 44: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0

+

Ethernet II, Src: 00:1f:9d:0b:16:8f (00:1f:9d:0b:16:8f), Dst: 01:80:c2:00:00:07 (01:80:c2:00:00:07)

+

Destination: 01:80:c2:00:00:07 (01:80:c2:00:00:07)

+

Source: 00:1f:9d:0b:16:8f (00:1f:9d:0b:16:8f)

+

Type: Ethernet Local Management Interface (MEF16) (0x88ee)

+

Ethernet Local Management Interface (E-LMI)

+

Version: 1

+

Message type: Status enquiry (0x75)

+

Information element: Report type

+

Tag: Report type (0x01)

+

Length: 1

+

Report type: E-LMI check (1)

+

Information element: Sequence numbers

+

Information element: Data instance

The status message reply (as a “check”) isn’t interesting, but I show it for completeness. This is sent by the UPE (ME2) back to the CE (ME3) on all ports they share.

No.	Time	Source	Destination	Protocol	Length	Info
	14.982521000	00:1f:9d:0b:16:8f	01:80:c2:00:00:07	E-LMI	60	Status enquiry
	14.982553000	00:1f:9d:0b:16:90	01:80:c2:00:00:07	E-LMI	60	Status enquiry
	14.983551000	04:c5:a4:53:d8:0f	01:80:c2:00:00:07	E-LMI	60	Status
	14.983799000	04:c5:a4:53:d8:10	01:80:c2:00:00:07	E-LMI	60	Status

<

Frame 46: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0

- Ethernet II, Src: 04:c5:a4:53:d8:0f (04:c5:a4:53:d8:0f), Dst: 01:80:c2:00:00:07 (01:80:c2:00:00:07)
 - Destination: 01:80:c2:00:00:07 (01:80:c2:00:00:07)
 - Source: 04:c5:a4:53:d8:0f (04:c5:a4:53:d8:0f)
 - Type: Ethernet Local Management Interface (MEF16) (0x88ee)
- Ethernet Local Management Interface (E-LMI)
 - Version: 1
 - Message type: Status (0x7d)
 - Information element: Report type
 - Tag: Report type (0x01)
 - Length: 1
 - Report type: E-LMI check (1)
 - Information element: Sequence numbers
 - Information element: Data instance

Next, we will debug the discovery process on both routers. Unlike CFM and OAM, E-LMI isn't really meant for failure detection. It's more about communicating circuit information between UPE and CE, so when the link fails, nothing interesting happens to E-LMI (no logs, no good debug, etc). The most valuable indication we get from E-LMI that a link is down is a question mark in the summary output. I unplug port 13 on ME3 to break the link temporarily with debug enabled on both ME2 and ME3.

```
ME2#show ethernet lmi evc
St   EVC Id                                     Port
---  ---
?A   EVC_ELM1                                Fa0/13,
A                                         Fa0/14
```

Debugging E-LMI events on both switches for port 13 (only ME2 command is shown), the only significant message we see is shown below. The UPE and CE have slightly different messages with the UPE message being more straightforward.

```
ME2#debug ethernet lmi events interface fast0/13
Ethernet LMI events debugging is on for FastEthernet0/13

! ME2
ELMI Fa0/13 EVENT: Link Down, Async Ignored on this interface

! ME3
ELMI Fa0/13 EVENT: ce_event: State 0x0, Event 0x4
ELMI Fa0/13 EVENT: Old State=0x0, Event=0x4, New State=0x2
```

Plugging the cable back in, we receive a lot of debug from both devices. In summary, the devices exchange E-LMI messages to establish their peering relationship again, and we can see some of the update details as well. It shows the number of UNIs configured, number of VLANs, and other information. Wireshark doesn't interpret this information well, so we don't examine the captures for it. Notice that the concept of "full" messages are introduced here. Unlike "check" messages, these actually

contain the EVC status information that a CE would normally require. These are advertised from UPE to CE early in the session formation and sustained via “check” messages.

```
! ME2
ELMI Fa0/13 EVENT: process_pe: State 0x0, Event 0x0
ELMI Fa0/13 EVENT: Update seq: current send 0x2 rcv 0x2
ELMI Fa0/13 EVENT: Local di 0x2 Remote 0x1
ELMI Fa0/13 EVENT: Updated Stat Type: ETHER_LMI_ST_LFULL_MSG_RCVD
ELMI Fa0/13 EVENT: Update evc_sts: ref_id: 0x1
ELMI Fa0/13 EVENT: Update evc_param: type 0x0
ELMI Fa0/13 EVENT: Update evc_id
ELMI Fa0/13 EVENT: Update remote_uni_sum cfgd 1 up 2
ELMI Fa0/13 EVENT: Update cevlan_evc_map: ref_id: 0x1 seq# 0x1
ELMI Fa0/13 EVENT: Update evc_map: num_vlans 1
ELMI Fa0/13 EVENT: Update remote_uni_det: evc ref_id: 0x1 uni ref_id 0x70
ELMI Fa0/13 EVENT: Updated Stat Type: ETHER_LMI_ST_LFULL_MSG_SENT
```

```
! ME3
07:20:31: ELMI Fa0/13 EVENT: Old State=0x2, Event=0x5, New State=0x0
07:20:31: ELMI Fa0/13 EVENT: Old State=0x0, Event=0x5, New State=0x1
07:20:31: ELMI Fa0/13 EVENT: Updated Stat Type: ETHER_LMI_ST_LFULL_MSG_SENT
07:20:31: ELMI Fa0/13 EVENT: ce_event: State 0x1, Event 0x0
07:20:31: ELMI Fa0/13 EVENT: Update seq: current send 0x1 rcv 0x0
07:20:31: ELMI Fa0/13 EVENT: Update uni:
07:20:31: ELMI Fa0/13 EVENT: Update evc_sts: ref_id: 0x1
07:20:31: ELMI Fa0/13 EVENT: Update evc_param: type 0x0
07:20:31: ELMI Fa0/13 EVENT: Update evc_id
07:20:31: ELMI Fa0/13 EVENT: Update remote_uni_sum cfgd 1 up 2
07:20:31: ELMI Fa0/13 EVENT: Update cevlan_evc_map: ref_id: 0x1 seq# 0x1
07:20:31: ELMI Fa0/13 EVENT: Update evc_map: num_vlans 1
07:20:31: ELMI Fa0/13 EVENT: Update remote_uni_det: evc ref_id: 0x1 uni
ref_id 0x70
07:20:31: ELMI Fa0/13 EVENT: Update di: current 0x0 rcvd 0x1
07:20:31: ELMI Fa0/13 EVENT: Old State=0x1, Event=0x0, New State=0x0
07:20:31: ELMI Fa0/13 EVENT: Updated Stat Type: ETHER_LMI_ST_LFULL_MSG_RCVD
```

To see the value of E-LMI, we can introduce some misconfigurations. On the UPE, we will add a bogus VLAN 99 to the C-VLAN mappings associated with the EVC. This doesn't break connectivity, but E-LMI prints an error message on the CE about the mismatch. Essentially, when the PE receives the C-VLAN mappings from the UPE, it compares them against its trunk allowed list. Since only 13 is allowed, the message notes the discrepancy. If you miss the log message, it is stored in the EVC detailed show command as part of the available C-VLANs. This isn't very dangerous because even if VLAN 99 service is offered, the CE isn't using it, so it isn't a big problem.

```
! ME2
interface FastEthernet0/13
 service instance 13 ethernet EVC_ELMI
```

```
ethernet lmi ce-vlan map 13,99
```

```
! ME3
```

```
%ETHER_LMI-6-MISMATCHED_VLAN_NOT_CONFIGURED: VLAN 99 not configured but in  
VLAN mapping for UNI FastEthernet0/13
```

```
ME3#show ethernet lmi evc detail EVC_ELMI interface fast0/13
```

```
EVC Id: EVC_ELMI
```

```
interface FastEthernet0/13
```

```
Time since Last Full Report: 00:01:57
```

```
Ether LMI Link Status: Up
```

```
UNI Status: Up
```

```
UNI Id: VLAN13
```

```
CE-VLAN/EVC Map Type: Bundling
```

```
VLAN: 13,99
```

```
[snip]
```

To test C-VLAN mapping comparison to the local trunk allowed list, we can create the opposite problem also. On ME3, we add VLAN 99 to the trunk allowed list, then remove VLAN 99 from the C-VLAN mappings on the UPE (ME2).

```
! ME3
```

```
interface FastEthernet0/13
```

```
switchport trunk allowed vlan 13,99
```

```
! ME2
```

```
interface FastEthernet0/13
```

```
service instance 13 ethernet EVC_ELMI
```

```
no ethernet lmi ce-vlan map 99
```

The error message is now reversed as VLAN 99 is locally configured on the CE but is not offered as a service by the UPE. This is the more dangerous case because the CE might think VLAN 99 has connectivity but it doesn't. E-LMI is useful since it clearly communicates that VLAN 99 isn't available. Before continuing, I remove VLAN 99 from all switches to restore the network to its baseline.

```
! ME3
```

```
%ETHER_LMI-6-MISMATCHED_VLAN_CONFIGURED: VLAN 99 configured but not in VLAN  
mapping for UNI FastEthernet0/13
```

Another misconfiguration that is probably common (since Cisco doesn't show it in all of their configuration examples) is forgetting to identify a UNI ID to the UPE. I will remove the UNI IDs "VLAN13" and "VLAN14" from the UPE to demonstrate this.

```
! ME2
```

```
interface FastEthernet0/13
```

```
no ethernet uni id VLAN13
```

```
interface FastEthernet0/14
no ethernet uni id VLAN14
```

We will pretend like we never configured the UNI IDs correctly in the first place and begin troubleshooting. Since the CE initiates communication with status enquiry messages, the UPE is expected to respond. However, this cannot occur if a UNI ID is not configured. We can see packets arriving but no responses being sent, and the E-LMI error debugs reveal the issue. A quick look at the sequence numbers in the packet helps confirm the one-way communications since only the “send” value is incrementing from the peer. The CE doesn’t show any errors since it is not aware of a problem; it just sends status enquiry messages over and over.

```
ME2#debug ethernet lmi errors
Ethernet LMI errors debugging is on
```

```
ME2#debug ethernet lmi packets
Ethernet LMI packets debugging is on
```

```
! ME2
```

```
08:07:18: ELMI Fa0/13 PACKET: Incoming
          Protocol Version      : 0x1
          Message                : STATUS ENQ (0x75)
          Report Type            : Full
          Sequence Number        : Snd(0xB), Rcv(0x0)
          Data Instance          : Value(0x0)
```

```
08:07:18: ELMI Fa0/13 ERROR: Ether LMI PE config incomplete: missing UNI Id
08:07:18: ELMI Fa0/13 ERROR: pe_handler: State 0x0, Event 0x0, Rep_type 0x0
```

```
! ME3
```

```
08:14:20: ELMI Fa0/13 PACKET: Outgoing
          Protocol Version      : 0x1
          Message                : STATUS ENQ (0x75)
          Report Type            : Full
          Sequence Number        : Snd(0xD), Rcv(0x0)
          Data Instance          : Value(0x0)
```

It is interesting to note that the UPE considers the E-LMI sessions “active” in this broken state. This output could be technically true, but it complicates troubleshooting. Even worse, the CE gives no output at all. The lesson here is to rely on debugs when the show commands don’t help.

```
ME2#show ethernet lmi evc
```

St	EVC Id	Port
A	EVC_ELMI	Fa0/13,
A		Fa0/14

Key: St=Status, A=Active, P=Partially Active, I=Inactive, ?=Link Down

```
ME3#show ethernet lmi evc
[no output]
```

Before continuing, I add the UNI IDs back to the UPE ports so E-LMI works properly. Some misconfigurations simply are not allowed by the parser. For example, we used S-VLAN 10 in the operator network within the EVC. If we try to configure a bogus VLAN, the parser rejects it. It checks the CFM domain for the VLAN ID you try to configure to ensure there is a valid CFM service for it within the specified domain. This is very helpful as a protection mechanism.

```
ME2(config-evc)#oam protocol cfm svlan 18 domain OPERATOR
svlan 18 not configured under domain name OPERATOR. First configure it.
```

Two key benefits of E-LMI are that it notifies the CE when EVCs are added and deleted. To test this, I add another pair of operator MEPs between ME1 and ME2 (port 13 UPE only). This will use the operator VLAN 11, and we must add it to the UPE trunk allowed list in order for ME2 to send CCMs for this service instance. This is effectively serving the same purpose of VLAN 10.

```
! ME1 and ME2
ethernet cfm domain OPERATOR level 1
  service OP2 vlan 11
  continuity-check

! ME1 only
interface FastEthernet0/9
  ethernet cfm mep domain OPERATOR mpid 111 service OP2

! ME2 only
interface FastEthernet0/13
  switchport trunk allowed vlan 10,11,13
  ethernet cfm mep domain OPERATOR mpid 111 service OP2
```

We quickly verify this new pair of MEPs has bidirectional reachability. First, we check the MEP listing to see MPID 111 available and servicing VLAN 11 on ME2. Then, we use Ethernet ping to test reachability.

```
ME2#show ethernet cfm maintenance-points remote domain OPERATOR service OP2
```

MPID	Domain Name	MacAddress	IfSt	PtSt
Lvl	Domain ID	Ingress		
RDI	MA Name	Type Id	SrvcInst	
	EVC Name		Age	
	Local MEP Info			
111	OPERATOR	001d.4692.f700	Up	Up
1	OPERATOR	Pol2		
-	OP2	Vlan 11	N/A	
	N/A		3s	

```
MPID: 211 Domain: OPERATOR MA: OP2
```

```
ME2#ping ethernet mpid 111 domain OPERATOR service OP2
Type escape sequence to abort.
Sending 5 Ethernet CFM loopback messages to 001d.4692.f700, timeout is 5
seconds:!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/9 ms
```

Next, I will create a dummy EVC that uses this new S-VLAN for signaling. CFM must be operational for this service if we specify CFM as the OAM protocol. The “OAM manager” process is what provides the interworking between CFM and E-LMI, so a functional CFM domain is a prerequisite. We don’t need to change the UNI count as we did before since there are only 2 ports using this EVC (UNI count is 2 by default). Re-using EVCs as I did for VLANs 13 and 14 is bad practice, but technically works. Notice that the EVC below isn’t associated with interfaces yet as we have not applied it.

```
! ME2
ethernet evc EVC_DUMMY
  oam protocol cfm svlan 11 domain OPERATOR

ME2#show ethernet service evc id EVC_DUMMY detail
EVC ID: EVC_DUMMY
EVC Type: P-P
UNI Count: Configured = 2, Active = 2
EVC Status: Active
Associated UNIs:
  UNI-Identifier      Status      Location
  VLAN13              Up          Fa0/13
```

Once we configure it on the UPE port, we can see the EVC bindings are updated. Again we use the bogus VLAN 99 as an offered service for the CE.

```
! ME2
interface FastEthernet0/13
  service instance 99 ethernet EVC_DUMMY
  ethernet lmi ce-vlan map 99

ME2#show ethernet service evc id EVC_DUMMY detail
EVC ID: EVC_DUMMY
EVC Type: P-P
UNI Count: Configured = 2, Active = 2
EVC Status: Active
Associated Local Interfaces:
  Interface      CE-Vlans
  FastEthernet0/13  99
Associated UNIs:
  UNI-Identifier      Status      Location
  VLAN13              Up          Fa0/13
```

Just like last time, VLAN 99 is not configured on ME3's trunk allowed list, so we see a familiar error message. There is a slight difference in the way this information is stored on ME3. This is technically a new EVC, so the EVC table on ME3 is updated to reflect this. The C-VLAN mappings offered by this EVC contain VLANs that are different than the existing EVC, which explains the new table entry. Both EVCs share a UNI ID since that is port-specific, but the actual "virtual circuit" VLAN mappings may differ.

```
! ME3
%ETHER_LMI-6-MISMATCHED_VLAN_NOT_CONFIGURED: VLAN 99 not configured but in
VLAN mapping for UNI FastEthernet0/13
```

```
ME3#show ethernet lmi evc
St  EVC Id                                     Port
---
A   EVC_DUMMY                                Fa0/13
A   EVC_ELMi                                Fa0/13,
A                                     Fa0/14
```

```
ME3#show ethernet lmi evc detail EVC_DUMMY
EVC Id: EVC_DUMMY
interface FastEthernet0/13
  Time since Last Full Report: 00:01:48
  Ether LMI Link Status: Up
  UNI Status: Up
  UNI Id: VLAN13
  CE-VLAN/EVC Map Type: Bundling
  VLAN: 99
  [snip]
```

Debugging E-LMI packets on ME3, we can see this update. Like the E-LMI "hellos", this is a status message, but is the "full" type, not the "check" type. EVC_ELMi is referenced as 0x1 while EVC_DUMMY is referenced as 0x3. Later in the debug, we can see the EVC-to-C-VLAN mappings where the EVC number (0x1 or 0x3) precedes the stanza showing the VLANs. Because this was an addition, we can see that EVC_DUMMY is marked as "new" (green) while EVC_ELMi is "old" (yellow). Different EVCs may have different activity states based on their CFM S-VLAN states. Again, this illustrates why E-LMI is useful for communicating network state to customers, especially if they don't support CFM.

```
ME3#debug ethernet lmi packets interface fast0/13
Ethernet LMI packets debugging is on for FastEthernet0/13
```

```
! ME3
8:45:21: ELMi Fa0/13 PACKET: Incoming
      Protocol Version      : 0x1
      Message                : STATUS (0x7D)
      Report Type           : Full
      Sequence Number       : Snd(0xB5), Rcv(0xCB)
      Data Instance         : Value(0x5)
      UNI                   : Bundle
```

```

UNI Id          : 'VLAN13'
EVC Status      : Evc Ref(0x1), Old, Active
EVC Parameters  : MultiPoint-to-MultiPoint
EVC Id          : 'EVC_ELM1'
Remote UNI Sum  : Cfgd(2), Up(2)
EVC Status      : Evc Ref(0x3), New, Active
EVC Parameters  : Point-to-Point
EVC Id          : 'EVC_DUMMY'
Remote UNI Sum  : Cfgd(1), Up(1)
CEVLAN EVC Map  : Evc Ref(0x1), Seq(0x1)
EVC Map         : Num Vlans(1), 13
CEVLAN EVC Map  : Evc Ref(0x3), Seq(0x1)
EVC Map         : Num Vlans(1), 99
Remote UNI Status : Evc Ref(0x1), Uni Ref(0x70), Up

```

To clean up the network, I remove the dummy EVC (not shown). There is no explicit log message, but the fact that the EVC table is stored on the CE qualifies as “informing” the CE of changes in on the UPE. In this way, E-LMI can inform the customers of EVC changes, VLAN mappings, etc. The removal of a deleted EVC is implicit since the full update simply does not include that EVC at all. When a CE receives a full status update, it removes EVCs not listed in the update. This is loosely analogous to LDP implicit label withdrawals, where a new label binding simply overwrites the old one without an explicit withdraw message being issued.

```
ME3#show ethernet lmi evc detail EVC_DUMMY
```

```
Cannot find EVC ID: EVC_DUMMY
```

```
ME3#debug ethernet lmi packets interface fast0/13
```

```
Ethernet LMI packets debugging is on for FastEthernet0/13
```

```
! ME3
```

```
08:50:11: ELM1 Fa0/13 PACKET: Incoming
```

```

Protocol Version      : 0x1
Message               : STATUS (0x7D)
Report Type           : Full
Sequence Number       : Snd(0xD3), Rcv(0xE9)
Data Instance         : Value(0x6)
UNI                   : Bundle
  UNI Id              : 'VLAN13'
EVC Status            : Evc Ref(0x1), Old, Active
EVC Parameters        : MultiPoint-to-MultiPoint
EVC Id                : 'EVC_ELM1'
Remote UNI Sum        : Cfgd(2), Up(2)
CEVLAN EVC Map        : Evc Ref(0x1), Seq(0x1)
EVC Map               : Num Vlans(1), 13
Remote UNI Status     : Evc Ref(0x1), Uni Ref(0x70), Up

```


Last, we will examine E-LMI minor optimizations, which include the N393 error counter and T392 polling timer. These timer names are unintuitive and warrant additional explanation/testing. Fortunately, the parser gives us some valuable information.

```
ME2(config-if)#ethernet lmi ?
  interface  Enable Ethernet LMI on this interface
  n393       Error counter
  t392       Polling verification timer
```

The MEF strictly defines these fields in MEF.16-0106 (the output even tells you the specification), their defaults, and their ranges. T392 is called the polling verification timer (PVT) which is 15 seconds by default. This number should be greater than T391, which is the polling timer applicable to the CE; this cannot be adjusted on the UPE. N393 counts the number of consecutive errors, and if the threshold is crossed, E-LMI is considered to be dysfunctional. The default value is 4. N391 counts the number of “check” cycles before requesting a “full” status with a default of 360 (on CE only). I slightly modify both values on ME2 port 13 just for demonstration purposes. To verify it, we can look at the E-LMI interface parameters on both port 13 (customized) and 14 (default) to compare the values. Notice that some other timers are not applicable on PE devices.

```
! ME2
interface FastEthernet0/13
  ethernet lmi t392 20
  ethernet lmi n393 6
```

```
ME2#show ethernet lmi parameters interface fast0/13
E-LMI Parameters for interface FastEthernet0/13
Version : MEF.16-0106
Mode : PE
T391 : NA
T392 : 20
N391 : NA
N393 : 6
```

```
ME2#show ethernet lmi parameters interface fast0/14
E-LMI Parameters for interface FastEthernet0/14
Version : MEF.16-0106
Mode : PE
T391 : NA
T392 : 15
N391 : NA
N393 : 4
```

On the CE, we have a different set of options. N391 (how many consecutive polls before a full status) and T391 (how often to poll) are revealed. We can still count errors, too, as the polling verification timer (T392) is no longer available. I made minor adjustments here as well, then verify it. I also show the defaults from port 14 for comparison.

```
ME3(config-if)#ethernet lmi ?
  interface  Enable Ethernet LMI on this interface
  n391       Polling counter
  n393       Error counter
  t391       Polling timer
```

```
! ME3
interface FastEthernet0/13
 ethernet lmi n391 90
 ethernet lmi t391 12
```

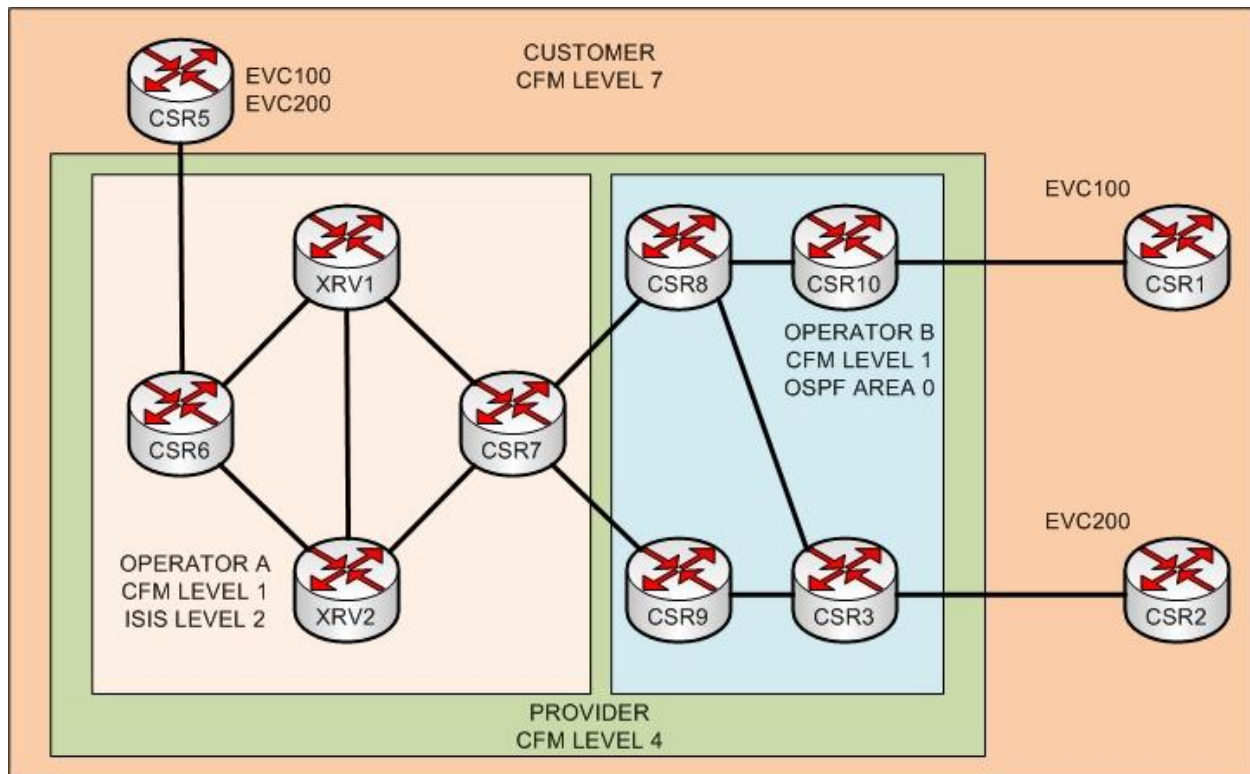
```
ME3#show ethernet lmi parameters interface fast0/13
E-LMI Parameters for interface FastEthernet0/13
Version : MEF.16-0106
Mode : CE
T391 : 12
T392 : NA
N391 : 90
N393 : 4
```

```
ME3#show ethernet lmi parameters interface fast0/14
E-LMI Parameters for interface FastEthernet0/14
Version : MEF.16-0106
Mode : CE
T391 : 10
T392 : NA
N391 : 360
N393 : 4
```

Additional Reading – Reference configurations "cfm-elmi"

48.3.4 Ethernet CFM, OAM, E-LMI, and Y.1731 on CSR1000v (Comprehensive)

This lab tests all of the Ethernet management tools from the previous chapters on the CSR1000v. The main focus of this lab will be a realistic, hierarchical CFM deployment with multiple operators. The service provider maintenance domain spans both operators and provides E-LINE service (EVPL) between three customer sites. CSR5 has a P2P EVPL to CSR1 and CSR2 with EIGRP running inside of it. Each operator, for the sake of not turning every router into a switch, runs a pair of PWs between their edge routers to connect the E-LINE service end-to-end. Ethernet OAM and E-LMI are used between CSR3 (UPE) and CSR2 (CE) for demonstration but are not the focus of the lab. Y.1731 PM and ordinary IP SLA probes are also tested. The diagram is below which shows the operator domains, provider domain, and customer domain. Operator A uses IS-IS and operator B uses OSPF; both of them use LDP and static PWs for their MPLS service delivery. XRv1 and XRv2 are just P routers and their configurations are not shown as they are very basic.



First, we will configure operator A. Since CSR6 and CSR7 are the edge routers in this operator's network, they will have inward-facing (upward) MEPs on their service-instances. This will allow the operator to run CFM within its own network. Since the provider has subcontracted these operators to provide last-mile transport, it will also establish an upward MEP on the service instance that faces the customer only; the provider has no reason to monitor the inter-operator links directly. Since the customer will run CFM as well, the customer-facing service-instance is a level 7 MIP to allow level 7 CFM frames to transit that interface. The inter-operator service-instances are level 4 MIPs to confine lower level traffic but still allow higher level traffic to pass though. Notice that CSR7 does not have the provider domain configured as it does not terminate any MEPs for the provider; the level 4 MIP will allow the provider CCMs to pass. The EVCs also invoke CFM as their OAM protocol. This is mostly relevant for E-LMI but for completeness I add it everywhere. Since the customer (CSR5) has two physical ports connected, each EVPL uses a different C-VLAN tag, which are 3556 (EVC 100) and 3565 (EVC 200) shown below.

```
! CSR6
ethernet cfm ieee
ethernet cfm global
ethernet cfm logging

ethernet cfm domain PROVIDER level 4
  service number 100 evc EVC100
    continuity-check
  service number 200 evc EVC200
    continuity-check
```

```

ethernet cfm domain OPERATOR level 1
  service number 100 evc EVC100
    continuity-check
  service number 200 evc EVC200
    continuity-check

ethernet evc EVC100
  oam protocol cfm domain PROVIDER
ethernet evc EVC200
  oam protocol cfm domain PROVIDER

interface GigabitEthernet2
  service instance 100 ethernet EVC100
  encapsulation dot1q 3556
  rewrite ingress tag pop 1 symmetric
  cfm mep domain PROVIDER mpid 106
    alarm notification all
  cfm mep domain OPERATOR mpid 106
    alarm notification all
  cfm mip level 7

service instance 200 ethernet EVC200
  encapsulation dot1q 3565
  rewrite ingress tag pop 1 symmetric
  cfm mep domain PROVIDER mpid 160
    alarm notification all
  cfm mep domain OPERATOR mpid 160
    alarm notification all
  cfm mip level 7

```

The configuration on CSR7 is similar but lacks the provider domain as there are no MEPs for it. MEP level 4 is used to confine the operator messages (level 1), process the provider (level 4) messages, and pass the customer (level 7) messages. This is what allows CFM to be hierarchical and offer multiple layers of maintenance for different managing organizations.

```

! CSR7
ethernet cfm ieee
ethernet cfm global
ethernet cfm logging
ethernet cfm domain OPERATOR level 1
  service number 100 evc EVC100
    continuity-check
  service number 200 evc EVC200
    continuity-check

ethernet evc EVC100
  oam protocol cfm domain OPERATOR

```

```

ethernet evc EVC200
  oam protocol cfm domain OPERATOR

interface GigabitEthernet2
  service instance 100 ethernet EVC100
  encapsulation dot1q 3578
  rewrite ingress tag pop 1 symmetric
  cfm mep domain OPERATOR mpid 107
  alarm notification all
  cfm mip level 4

  service instance 200 ethernet EVC200
  encapsulation dot1q 3579
  rewrite ingress tag pop 1 symmetric
  cfm mep domain OPERATOR mpid 170
  alarm notification all
  cfm mip level 4

```

The routed interfaces below are the MPLS transit links. This is a very basic IS-IS configuration and is not examined in detail. CSR7 is configured similarly (different NET, different IP addresses, etc) but it is not shown for brevity.

```

! CSR6
interface GigabitEthernet2.561
  encapsulation dot1Q 3561
  ip address 131.6.11.6 255.255.255.0
  ip router isis 131
  isis network point-to-point

interface GigabitEthernet2.562
  encapsulation dot1Q 3562
  ip address 131.6.12.6 255.255.255.0
  ip router isis 131
  isis network point-to-point

router isis 131
  net 00.0000.0000.0006.00
  is-type level-2-only
  advertise passive-only
  metric-style wide
  log-adjacency-changes all
  passive-interface Loopback0
  mpls ldp autoconfig

```

Last, we configure the basic L2VPN. Because I used generic naming for the VPN configurations, this syntax is identical on CSR6 and CSR7. I enable L2VPN logging and connect the service-instance to the appropriate PW.

```

! CSR6 and CSR7
l2vpn
  logging pseudowire status

l2vpn xconnect context XC100
  member GigabitEthernet2 service-instance 100
  member pseudowire100

l2vpn xconnect context XC200
  member GigabitEthernet2 service-instance 200
  member pseudowire200

```

Assuming XRv1 and XRv2 have been configured with basic IS-IS and LDP as discussed earlier, the operator's network should be functional. We see several log messages on CSR6 and CSR7 that indicate the PW has come up and the remote operator MEPs have been discovered. This is specific to service number 100 within the operator domain, but identical messages are generated for service number 200 as well.

```

! CSR6
%XCONNECT-5-PW_STATUS: MPLS peer 131.0.0.7 vcid 100, VC state UP
%E_CFM-6-REMOTE_MEP_UP: Continuity Check message is received from a remote
MEP with mpid 107 evc EVC100 MA name number 100 domain OPERATOR interface
status Up event code New.

```

We quickly check the status of the PWs and remote MEPs in case we missed the log messages, beginning with service number 100. The remote MEP also references the PW information, to include the tLDP peer address and VCID. We complete our verification with an Ethernet ping.

```
R6#show l2vpn atom vc vcid 100
```

Interface	Peer ID	VC ID	Service		Status
			Type	Name	
pw100	131.0.0.7	100	p2p	XC100	UP

```
R6#show ethernet cfm maintenance-points remote domain OPERATOR service number 100
```

MPID	Domain Name	MacAddress	IfSt	PtSt
Lvl	Domain ID	Ingress		
RDI	MA Name	Type Id	SrvcInst	
	EVC Name		Age	
	Local MEP Info			
107	OPERATOR	001e.49ca.a4bf	Up	Up
1	OPERATOR	Gi2:(131.0.0.7, 100)		
-	number 100	XCON N/A	100	
	EVC100		2s	
	MPID: 106 Domain: OPERATOR MA: number 100			

Total Remote MEPs: 1

```
R6#ping ethernet mpid 107 domain OPERATOR service number 100
Type escape sequence to abort.
Sending 5 Ethernet CFM loopback messages to 001e.49ca.a4bf, timeout is 5
seconds:!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/5/6 ms
```

Next, we perform the same verifications for service number 200 inside of the operator domain. This shows that both PWs and both MEPs are functional.

```
R6#show l2vpn atom vc vcid 200
```

Interface	Peer ID	VC ID	Service		Status
			Type	Name	
pw200	131.0.0.7	200	p2p	XC200	UP

```
R6#show ethernet cfm maintenance-points remote domain OPERATOR service number
200
```

MPID	Domain Name	MacAddress	IfSt	PtSt
Lvl	Domain ID	Ingress		
RDI	MA Name	Type Id	SrvcInst	
	EVC Name		Age	
	Local MEP Info			
170	OPERATOR	001e.49ca.a4bf	Up	Up
1	OPERATOR	Gi2:(131.0.0.7, 200)		
-	number 200	XCON N/A	200	
	EVC200		5s	
	MPID: 160 Domain: OPERATOR MA: number 200			

Total Remote MEPs: 1

```
R6#ping ethernet mpid 170 domain OPERATOR service number 200
Type escape sequence to abort.
Sending 5 Ethernet CFM loopback messages to 001e.49ca.a4bf, timeout is 5
seconds:!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/5/6 ms
```

Operator B is different as there are 2 inter-operator routers and 2 UPEs, where in operator A there was only 1 of each. The logic is identical, except that each router only has a single service instance rather than two. I re-use the domain “OPERATOR” at level 1 as well as EVC100 and PW100. Since this information is local to each operator, I was able to quickly build operator B since the configuration is very similar to operator A. The MIP is at level 4 which is identical to operator A, which confines the operator-level CCMs. The OSPF routed interfaces are shown for completeness; like IS-IS in operator A,

there is no IGP complexity in this lab. The PW configuration is identical to what was seen on operator A and two PWs are configured. One PW is between CSR8-CSR10 to service EVC100 while the other is between CSR9-CSR3 to service EVC200.

```
! CSR8
ethernet cfm ieee
ethernet cfm global
ethernet cfm logging
ethernet cfm domain OPERATOR level 1
    service number 100 evc EVC100
    continuity-check

ethernet evc EVC100
    oam protocol cfm domain OPERATOR

interface GigabitEthernet2
    service instance 100 ethernet EVC100
    encapsulation dot1q 3578
    rewrite ingress tag pop 1 symmetric
    cfm mep domain OPERATOR mpid 108
    alarm notification all
    cfm mip level 4

interface GigabitEthernet2.538
    encapsulation dot1Q 3538
    ip address 132.3.8.8 255.255.255.0
    ip ospf network point-to-point
    ip ospf 132 area 0

interface GigabitEthernet2.580
    encapsulation dot1Q 3580
    ip address 132.8.10.8 255.255.255.0
    ip ospf network point-to-point
    ip ospf 132 area 0

l2vpn
    logging pseudowire status

l2vpn xconnect context XC100
    member GigabitEthernet2 service-instance 100
    member pseudowire100
```

CSR9 is very similar to CSR8 except it services only EVC200. I do not show the IGP details on CSR9 for brevity.

```
! CSR9
ethernet cfm ieee
ethernet cfm global
```



```

ethernet cfm logging

ethernet cfm domain OPERATOR level 1
  service number 200 evc EVC200
  continuity-check

ethernet evc EVC200
  oam protocol cfm domain OPERATOR

interface GigabitEthernet2
  service instance 200 ethernet EVC200
  encapsulation dot1q 3579
  rewrite ingress tag pop 1 symmetric
  cfm mep domain OPERATOR mpid 209
  alarm notification all
  cfm mip level 4

l2vpn
  logging pseudowire status

l2vpn xconnect context XC200
  member GigabitEthernet2 service-instance 200
  member pseudowire200

```

CSR10 is the UPE for EVC100. It terminates the EVC100 PW and the operator MA. It also terminates the provider MA which should discover the MEPs configured earlier on CSR6. Thus, CSR10 will see MEP 108 in the operator domain (CSR8) and MEP 106 in the provider domain (CSR6). Like CSR6, CSR10 also defines a MIP on this service-instance to keep the provider/operator CCMs away from the customer, yet permit the customer CCMs on ingress.

```

! CSR10
ethernet cfm ieee
ethernet cfm global
ethernet cfm logging
ethernet cfm domain PROVIDER level 4
  service number 100 evc EVC100
  continuity-check
ethernet cfm domain OPERATOR level 1
  service number 100 evc EVC100
  continuity-check

ethernet evc EVC100
  oam protocol cfm domain PROVIDER

interface GigabitEthernet2
  service instance 100 ethernet EVC100
  encapsulation dot1q 3510
  rewrite ingress tag pop 1 symmetric

```

```

cfm mep domain PROVIDER mpid 110
  alarm notification all
cfm mep domain OPERATOR mpid 110
  alarm notification all
cfm mip level 7

```

```

l2vpn
logging pseudowire status

```

```

l2vpn xconnect context XC100
member GigabitEthernet2 service-instance 100
member pseudowire100

```

We can verify the PW is operational and that we see remote MEPs in both the operator and provider MAs. Because the remote MEP in the provider domain came up, we can automatically assume that we have connectivity between operators.

```

R10#show l2vpn atom vc vcid 100

```

Interface	Peer ID	VC ID	Service		Status
			Type	Name	
pw100	132.0.0.8	100	p2p	XC100	UP

```

R10#show ethernet cfm maintenance-points remote

```

MPID	Domain Name	MacAddress	IfSt	PtSt
Lvl	Domain ID	Ingress		
RDI	MA Name	Type Id	SrvcInst	
	EVC Name		Age	
	Local MEP Info			
108	OPERATOR	001e.e64d.4dbf	Up	Up
1	OPERATOR	Gi2: (132.0.0.8, 100)		
-	number 100	XCON N/A	100	
	EVC100			5s
	MPID: 110 Domain: OPERATOR MA: number 100			
106	PROVIDER	001e.bd69.62bf	Up	Up
4	PROVIDER	Gi2: (132.0.0.8, 100)		
-	number 100	XCON N/A	100	
	EVC100			8s
	MPID: 110 Domain: PROVIDER MA: number 100			

```

Total Remote MEPs: 2

```

We can prove this using Ethernet traceroute; recall that this is the only time MIPs use loopback messaging with MEPs. The MIPs at the inter-operator boundary reply to the LTM messages to show the hops between the seams inside of the provider domain. If there were lower-level MIPs within the operator networks, the provider-level traceroute would not have revealed those hops. Before running

the traceroute, I reveal all of the “brain” MAC addresses. Since all of the MEPs have been “UP MEPs” so far, they rely on the bridge-relay function of each CFM bridge and thus do not use the wire-side MAC addresses. I color-code them below and map these colors to the traceroute replies at each hop.

```
R10#show ethernet cfm statistics | include BRAIN
```

```
BRAIN MAC: 001e.146f.83bf
```

```
R8#show ethernet cfm statistics | include BRAIN
```

```
BRAIN MAC: 001e.e64d.4dbf
```

```
R7#show ethernet cfm statistics | include BRAIN
```

```
BRAIN MAC: 001e.49ca.a4bf
```

```
R6#show ethernet cfm statistics | include BRAIN
```

```
BRAIN MAC: 001e.bd69.62bf
```

```
R10#traceroute ethernet mpid 106 domain PROVIDER service number 100
Type escape sequence to abort. TTL 64. Linktrace Timeout is 5 seconds
Tracing the route to 001e.bd69.62bf on Domain PROVIDER, Level 4, service
number 100, evc EVC100
Traceroute sent via Gi2:(132.0.0.8, 100), path found via MPDB
```

B = Intermediary Bridge

! = Target Destination

* = Per hop Timeout

Hops	Host	MAC Forwarded	Ingress Egress	Ingr Action Egr Action	Relay Action Previous Hop
B 1		001e.146f.83bf Forwarded	Gi2	IngOk	RlyMPDB
B 2		001e.e64d.4dbf Forwarded	Gi2	EgrOK	RlyMPDB 001e.146f.83bf
B 3		001e.49ca.a4bf Forwarded	Gi2	IngOk	RlyMPDB 001e.e64d.4dbf
! 4		001e.bd69.62bf Not Forwarded			RlyHit:MEP 001e.49ca.a4bf

Next, we will complete the configuration for EVC200 by configuring CSR3. This is almost identical to CSR10 except services EVC200. I also use a different physical port for the UPE-CE connection so I can support Ethernet OAM as well. The traffic is untagged as this port is connected to the same VLAN (without trunking) as the customer. CSR3 also has a MEP in the operator B and provider domains, so it should behave identically to CSR10 with respect to CFM. Like CSR10, we verify that the PW is up, along with the remote MEPs in both the operator and provider MEGs. We won't verify Ethernet OAM/E-LMI yet as we have not configured the customer devices.

```
! CSR3
```

```

ethernet cfm ieee
ethernet cfm global
ethernet cfm logging
ethernet cfm domain PROVIDER level 4
    service number 200 evc EVC200
    continuity-check
ethernet cfm domain OPERATOR level 1
    service number 200 evc EVC200
    continuity-check

ethernet evc EVC200
    oam protocol cfm domain PROVIDER

interface GigabitEthernet3
    ethernet lmi interface
    ethernet oam
    ethernet uni id UNI200
    service instance 200 ethernet EVC200
    encapsulation untagged
    ethernet lmi ce-vlan map untagged
    cfm mep domain PROVIDER mpid 203
        alarm notification all
    cfm mep domain OPERATOR mpid 203
        alarm notification all
    cfm mip level 7

l2vpn
    logging pseudowire status

l2vpn xconnect context XC200
    member GigabitEthernet3 service-instance 200
    member pseudowire200

```

R3#show l2vpn atom vc vcid 200

Interface	Peer ID	VC ID	Service		Status
			Type	Name	
pw200	132.0.0.9	200	p2p	XC200	UP

R3#show ethernet cfm maintenance-points remote

MPID	Domain Name	MacAddress	IfSt	PtSt
Lvl	Domain ID	Ingress		
RDI	MA Name	Type Id	SrvcInst	
	EVC Name		Age	
	Local MEP Info			
209	OPERATOR	001e.e5a2.57bf	Up	Up
1	OPERATOR	Gi3:(132.0.0.9, 200)		

```

-      number 200                                XCON N/A                200
      EVC200                                      8s
      MPID: 203 Domain: OPERATOR MA: number 200
160  PROVIDER                                001e.bd69.62bf          Up      Up
      4 PROVIDER                                Gi3:(132.0.0.9, 200)
-      number 200                                XCON N/A                200
      EVC200                                      7s
      MPID: 203 Domain: PROVIDER MA: number 200

```

Total Remote MEPs: 2

For completeness, I reveal all of the brain MACs within the provider network so that our traceroute verification makes sense. We can see the output is very similar to CSR10 as the last 2 hops are identical, since operator A's network has not changed. The first two hops now represent CSR3 and CSR9, which do not participate in EVC100 at all.

```
R3#show ethernet cfm statistics | include BRAIN
```

```
BRAIN MAC: 001e.e5a8.ffbf
```

```
R9#show ethernet cfm statistics | include BRAIN
```

```
BRAIN MAC: 001e.e5a2.57bf
```

```
R7#show ethernet cfm statistics | include BRAIN
```

```
BRAIN MAC: 001e.49ca.a4bf
```

```
R6#show ethernet cfm statistics | include BRAIN
```

```
BRAIN MAC: 001e.bd69.62bf
```

```

R3#traceroute ethernet mpid 160 domain PROVIDER service number 200
Type escape sequence to abort. TTL 64. Linktrace Timeout is 5 seconds
Tracing the route to 001e.bd69.62bf on Domain PROVIDER, Level 4, service
number 200, evc EVC200
Traceroute sent via Gi3:(132.0.0.9, 200), path found via MPDB

```

B = Intermediary Bridge

! = Target Destination

* = Per hop Timeout

Hops	Host	MAC Forwarded	Ingress Egress	Ingr Action Egr Action	Relay Action Previous Hop

B 1		001e.e5a8.ffbf	Gi3	IngOk	RlyMPDB
		Forwarded			
B 2		001e.e5a2.57bf			RlyMPDB
		Forwarded	Gi2	EgrOK	001e.e5a8.ffbf
B 3		001e.49ca.a4bf	Gi2	IngOk	RlyMPDB
		Forwarded			001e.e5a2.57bf
! 4		001e.bd69.62bf			RlyHit:MEP

Now that the E-LINE services are fully functional and CFM-managed, we can begin configuring the customers. In the dedicated CFM chapter, I discussed “port MEPs” and configured dummy ports as an example. Here, we will configure and test them on the customer devices. Port MEPs actually make sense here since the customer typically wouldn’t want to configure “UP MEPs” to run CFM with the provider. Port MEPs are always in the down direction, which means the bridge-relay function is not used, and the wire-side MAC address is the source MAC. The CCMs are sent out of the wire-side address as well, so port MEPs are outward-facing (down) MEPs by definition. Port MEPs are also supported on routed ports, and we enable IPv4 processing to prove it. CSR5 uses two different physical interfaces to connect to the UPE which allows us to use port MEPs in the first place. The customer only configures a CFM domain for itself and at the highest level; this guarantees that its CFM frames will be able to transit all MIPs in the provider/operator networks. There are no EVCs configured on the customer sites, and basic EIGRP is enabled to continuously test IP transport over E-LINE.

```
! CSR5
ethernet cfm ieee
ethernet cfm global
ethernet cfm logging
ethernet cfm domain CUSTOMER level 7
  service number 100 port
  continuity-check
  service number 200 port
  continuity-check

interface GigabitEthernet2
  description SERVICE 100 TO CSR1
  ip address 10.1.5.5 255.255.255.0
  ethernet cfm mep domain CUSTOMER mpid 105 service number 100
  alarm notification all

interface GigabitEthernet3
  description SERVICE 200 TO CSR2
  ip address 10.2.5.5 255.255.255.0
  ethernet cfm mep domain CUSTOMER mpid 205 service number 200
  alarm notification all

router eigrp CUSTOMER
  address-family ipv4 unicast autonomous-system 125
  network 10.0.0.0
```

In order to see the customer remote MEPs and EIGRP neighbors, we must configure CSR1 and CSR2 as well. CSR2 also runs OAM/E-LMI with the provider which is verified later.

```
! CSR1
ethernet cfm ieee
```

```

ethernet cfm global
ethernet cfm logging
ethernet cfm domain CUSTOMER level 7
    service number 100 port
    continuity-check

interface GigabitEthernet2
    description SERVICE 100 TO CSR5
    ip address 10.1.5.1 255.255.255.0
    ethernet cfm mep domain CUSTOMER mpid 101 service number 100
    alarm notification all

router eigrp CUSTOMER
    address-family ipv4 unicast autonomous-system 125
    network 10.0.0.0

! CSR2
ethernet lmi ce
ethernet cfm ieee
ethernet cfm global
ethernet cfm logging
ethernet cfm domain CUSTOMER level 7
    service number 200 port
    continuity-check

interface GigabitEthernet2
    description SERVICE 200 TO CSR5
    ip address 10.2.5.2 255.255.255.0
    ethernet lmi interface
    ethernet oam
    ethernet cfm mep domain CUSTOMER mpid 202 service number 200
    alarm notification all

router eigrp CUSTOMER
    address-family ipv4 unicast autonomous-system 125
    network 10.0.0.0

```

After applying this configuration, we will verify that the customer MEPs came up. We will perform this on CSR5 since it should see both CSR1 and CSR2 as remote MEPs. We also verify the port-level MAC addresses on CSR1 and CSR2 on Gig2; port MEPs use the wire-side MAC addresses for their CCMs, unlike upward-facing MEPs. Again, I use color-coding to make identification easy.

```

R1#show interfaces gigabitEthernet 2 | include bia
    Hardware is CSR vNIC, address is 0050.56a9.1aaa (bia 0050.56a9.1aaa)

R2#show interfaces gigabitEthernet 2 | include bia
    Hardware is CSR vNIC, address is 0050.56a9.be8a (bia 0050.56a9.be8a)

```

```
R5#show ethernet cfm maintenance-points remote domain CUSTOMER
```

MPID	Domain Name	MacAddress	IfSt	PtSt
Lvl	Domain ID	Ingress		
RDI	MA Name	Type Id	SrvcInst	
	EVC Name		Age	
	Local MEP Info			
202	CUSTOMER	0050.56a9.be8a	Up	N/A
7	CUSTOMER	Gi3		
-	number 200	Port none	N/A	
	N/A		9s	
	MPID: 205 Domain: CUSTOMER MA: number 200			
101	CUSTOMER	0050.56a9.1aaa	Up	N/A
7	CUSTOMER	Gi2		
-	number 100	Port none	N/A	
	N/A		8s	
	MPID: 105 Domain: CUSTOMER MA: number 100			

```
Total Remote MEPs: 2
```

We can also verify IP connectivity by checking EIGRP neighbors. This ensures that the ESXi virtual networking, PWs, IPv4 ARP, and IP packet transport are all functioning properly. I highlight the “Q-Cnt” column because if EIGRP messages are building in the message queue, this is often a sign of a unicast connectivity issue. Since the queue is empty, we assume EIGRP has converged properly.

```
R5#show eigrp address-family ipv4 neighbors
```

```
EIGRP-IPv4 VR(CUSTOMER) Address-Family Neighbors for AS(125)
```

H	Address	Interface	Hold Uptime	SRTT	RTO	Q	Seq
			(sec)	(ms)		Cnt	Num
0	10.1.5.1	Gi2	11 01:07:04	12	100	0	9
1	10.2.5.2	Gi3	10 02:22:02	649	3894	0	8

For additional verification, we will enable CFM packet debugging on CSR6. All CCMs that are for MEPs that terminate on CSR6 will be revealed here; notice there are no CCMs for the customer domain at level 7 as CSR6 is just a MIP for that level.

```
R6#debug ethernet cfm packets
```

```
Ethernet CFM packet debugging is on for all
```

```
18:47:05.193: CFM-PKT: Received a CC packet with MPID 110, level 4, evc
EVC100 from interface Gi2:(131.0.0.7, 100)
18:47:06.032: CFM-PKT: Sending Up direction MEP 160 CC message, level 4, cos
7, evc EVC200
18:47:06.032: CFM-PKT: Sending Up direction MEP 106 CC message, level 4, cos
7, evc EVC100
```



```

18:47:07.494: CFM-PKT: Received a CC packet with MPID 170, level 1, evc
EVC200 from interface Gi2:(131.0.0.7, 200)
18:47:09.830: CFM-PKT: Received a CC packet with MPID 203, level 4, evc
EVC200 from interface Gi2:(131.0.0.7, 200)
18:47:13.127: CFM-PKT: Received a CC packet with MPID 107, level 1, evc
EVC100 from interface Gi2:(131.0.0.7, 100)
18:47:20.079: CFM-PKT: Received a CC packet with MPID 203, level 4, evc
EVC200 from interface Gi2:(131.0.0.7, 200)
18:47:13.203: CFM-PKT: Sending Up direction MEP 160 CC message, level 1, cos
7, evc EVC200
18:47:13.460: CFM-PKT: Sending Up direction MEP 106 CC message, level 1, cos
7, evc EVC100

```

Checking CSR7, we only see CCMs for the operator domain at level 1 since that is the only CFM domain on CSR7. The provider and customer domains are too high in the CFM hierarchy and their CCMs are forwarded across CSR7's MIPs without being processed.

```

! CSR7
18:50:17.498: CFM-PKT: Sending Up direction MEP 107 CC message, level 1, cos
7, evc EVC100
18:50:17.602: CFM-PKT: Received a CC packet with MPID 160, level 1, evc
EVC200 from interface Gi2:(131.0.0.6, 200)
18:50:17.858: CFM-PKT: Received a CC packet with MPID 106, level 1, evc
EVC100 from interface Gi2:(131.0.0.6, 100)
18:50:22.106: CFM-PKT: Sending Up direction MEP 170 CC message, level 1, cos
7, evc EVC200
18:50:27.738: CFM-PKT: Sending Up direction MEP 107 CC message, level 1, cos
7, evc EVC100

```

Since the customer isn't even aware of the operator or provider network architectures, it does not see CCMs from either one. They are bounded by level 7 MIPs at the UPEs as a good design practice. We confirm this by debugging on CSR2; notice the CCMs have the "down" direction set, which is different than all other CCM debugging seen so far. This is not adjustable since all port MEPs are down MEPs. Another use for port MEPs in an operator network is on the back-to-back inter-switch links. These can be at level 0 and are like "link-local" MEPs that can be used to test reachability across a very small MEG. The operator level 1 domain, as we have configured it, can span the entire operator's network, and typically uses UP MEPs.

```

! CSR2
18:53:53.533: CFM-PKT: Received a CC packet with MPID 205, level 7, port from
interface Gi2
18:53:54.708: CFM-PKT: Sending Down direction MEP 202 CC message, level 7,
cos 7, port on Gi2
18:54:03.778: CFM-PKT: Received a CC packet with MPID 205, level 7, port from
interface Gi2
18:54:04.951: CFM-PKT: Sending Down direction MEP 202 CC message, level 7,
cos 7, port on Gi2

```

Now that CFM has been fully verified, we will examine Ethernet OAM between CSR2 (CE) and CSR3 (UPE). The two peers discover one another using OAMPDUs to enter the OAM session. The interface MAC addresses are used and are color-coded below. We see that the “link monitor” capability has been established, which was discussed in detail earlier. This allows the routers to report frame loss, slips, and other important link quality characteristics.

```
R2#show interfaces gigabitEthernet 2 | include bia
Hardware is CSR vNIC, address is 0050.56a9.be8a (bia 0050.56a9.be8a)
```

```
R3#show interfaces gigabitEthernet 2 | include bia
Hardware is CSR vNIC, address is 0050.56a9.8ccf (bia 0050.56a9.8ccf)
```

```
R2#show ethernet oam summary
Symbols:          * - Master Loopback State,  # - Slave Loopback State
                  & - Error Block State
Capability codes: L - Link Monitor,   R - Remote Loopback
                  U - Unidirection,   V - Variable Retrieval
```

Local Interface	MAC Address	Remote OUI	Mode	Capability
Gi2	0050.56a9.86c3	00000C	active	L

```
R3#show ethernet oam summary | begin ^Interface
Interface      MAC Address    OUI      Mode    Capability

Gi3            0050.56a9.be8a  00000C   active  L
```

We will quickly enable “dying-gasp” on CSR3, the UPE. This will err-disable the interface if OAM is removed from the configuration or fails to negotiate a peer. After enabling this feature, we can verify the status of the interface to ensure it is enabled.

```
! CSR3
interface GigabitEthernet3
 ethernet oam remote-failure dying-gasp action error-disable-interface
```

```
R3#show ethernet oam status interface gig 3
GigabitEthernet3
General
-----
Admin state:      enabled
Mode:             active
PDU max rate:     10 packets per second
PDU min rate:     1 packet per 1000 ms
Link timeout:     5000 ms
High threshold action: no action
```

```

Link fault action:      no action
Dying gasp action:     error disable interface
Critical event action: no action

```

I remove Ethernet OAM from CSR2 entirely to test this (not shown). This breaks connectivity to CSR2 entirely even though nothing changed with CFM or MPLS. Using OAM in this way, we can guard against misconfigurations and effectively deny service to customers that are not compliant with OAM policies.

```

! CSR2
%ETHERNET_OAM-6-EXIT_SESSION: The client on interface Gi2 has left the OAM
session.

```

```

! CSR3
%ETHERNET_OAM-6-RFI: The client on interface Gi3 has received a remote
failure indication from its remote peer(failure reason = remote client
administratively turned off action = error-disable)

```

```

R3#show interfaces gig3 | include line_proto
GigabitEthernet3 is administratively down, line protocol is down (err-
disabled)

```

This triggers non-root-cause errors with CFM as well. There are no remote maintenance points since the UNI is down and cannot send/receive CCMs.

```

R3#show ethernet cfm errors

```

MPID	Domain Id	Mac Address	Type	Id
	MA Name	Reason	Lvl	Age

209	OPERATOR	001e.e5a2.57bf	XCON	N/A
	number 200	Lifetime Expired	1	224s
	MPID: 203 Domain: OPERATOR MA: number 200			
-	PROVIDER	001e.e5a8.ffbf	XCON	N/A
	number 200	Receive AIS	4	44s
	MPID: 203 Domain: PROVIDER MA: number 200			
160	PROVIDER	001e.bd69.62bf	XCON	N/A
	number 200	Lifetime Expired	4	223s
	MPID: 203 Domain: PROVIDER MA: number 200			

Adding Ethernet OAM back to the interface (not shown) does not resolve the problem since the interface is stuck in the err-disabled state. When testing on the ME3400s, we used err-discovery recovery, but this time we will shutdown/no shutdown CSR3's UNI port Gig3. This will clear the err-disable state, allowing Ethernet OAM to form again. Ultimately, this clears the CFM alarms since the MEPs associated with that UNI send/receive CCMs as they normally would.

```
R3#show ethernet cfm errors
```

```
[no output]
```

```
R3#show ethernet oam summary | begin ^Interface
```

Interface	MAC Address	OUI	Mode	Capability

Gi3	0050.56a9.be8a	00000C	active	L

```
R3#show ethernet cfm maintenance-points remote | include Total
```

```
Total Remote MEPs: 2
```

E-LMI doesn't appear to work well in this setup. Although it worked on the ME-3400s, the CSR1000v (UPE) seems to tunnel the E-LMI status enquiries inside the PW rather than process it locally. We know that the E-LMI ethertype is 0x88EE (yellow), so by using EPC outbound on CSR3 towards the MPLS core, we can confirm this. The E-LMI destination MAC is shown in cyan with CSR2's wire-side MAC address in green, and this is tunneled inside MPLS (grey) which isn't correct. CSR3 does not see the E-LMI packets at all so it can never send a status message; this causes CSR2 to simply time out.

```
R2#debug ethernet lmi packet
```

```
Ethernet LMI packets debugging is on
```

```
19:23:19.792: ELMi Gi2 PACKET: Outgoing
```

```
Protocol Version      : 0x1
```

```
Message               : STATUS ENQ (0x75)
```

```
Report Type           : Full
```

```
Sequence Number       : Snd(0x59), Rcv(0x0)
```

```
Data Instance         : Value(0x0)
```

```
R3#show monitor capture CAP buffer detailed
```

```
28  86  12.948000  00:50:56:A9:8C:CF -> 00:50:56:A9:D6:72 MPLS unicast
0000:  005056A9 D6720050 56A98CCF 8100DD3  .PV..r.PV.....
0010:  88470233 21FF0000 00000180 C2000007  .G.3!.....
0020:  005056A9 BE8A88EE 01750101 00020268  .PV.....u.....h
0030:  00030500 00000000 00000000 00000000  .....
```

Cisco's documentation clearly states that, on the ASR1000, E-LMI is not supported on "EoMPLS ports". I assume this includes the EoMPLS access circuit port, which is precisely what we tried to do here. It also states E-LMI is not supported on router ports, which is what we tried to do on CSR2 as well. It seems to me like this feature is best used on actual switches or routers using bridge-domains without any kind of PWs. I will leave the configuration for reference but will not troubleshoot E-LMI further.

Next, we will configure IP SLA probes and Y.1731 PM. First, we configure basic echo and jitter operations on CSR1 and CSR2 towards CSR5, respectively. These operations occur within the customer CFM domain, and as long as NTP is synchronized, the delay probe will yield one-way delay information as well. CSR5 will be the NTP master with CSR1 and CSR2 as NTP clients. Before continuing, we wait for NTP to synchronize. Note: these loopback addresses were advertised via EIGRP but not shown earlier.

```
! CSR5
ntp master
```

```
! CSR1 and CSR2
ntp server 10.0.0.5 minpoll 4 maxpoll 4
ntp source Loopback0
```

```
R5#show ntp status | include sync
Clock is synchronized, stratum 8, reference is 127.127.1.1
```

```
R1#show ntp status | include sync
Clock is synchronized, stratum 9, reference is 10.0.0.5
```

```
R2#show ntp status | include sync
Clock is synchronized, stratum 9, reference is 10.0.0.5
```

First, we will configure the simple Ethernet echo (loopback message) to CSR5. The IP SLA syntax only lets us specify an EVC, VLAN, or port MEP. Since the concept of EVC and VLAN aren't applicable to CSR1, we must select port, which implies there can only be one port within the customer domain. It is for this reason that we cannot use these basic IP SLA probes on CSR5 due to the artificiality of the design.

```
! CSR1
ip sla 15
  ethernet echo mpid 105 domain CUSTOMER port
  cos 3
  threshold 3000
  frequency 5
ip sla schedule 15 life forever start-time now
```

Once we start the probe, we verify that it is successful across the entire service provider network.

```
R1#show ip sla statistics 15
IPSLAs Latest Operation Statistics
```

```
IPSLA operation id: 15
Type of operation: ethernet-echo
  Latest RTT: 7 milliseconds
Latest operation start time: 19:58:35 UTC
Latest operation return code: OK
RTT Values:
  Number Of RTT: 1 RTT Min/Avg/Max: 7/7/7 milliseconds
Number of successes: 701
Number of failures: 0
Operation time to live: Forever
```

For additional debugging, CFM packet debugs also reveal these LBMs. They are sent 5 seconds apart; note that the LBR messages are not revealed in this debug.

```

! CSR1
20:02:25.317: CFM-PKT: Send LM Msg level 7, cos 3, port mac 0050.56a9.dc63 to
interface Gi2
20:02:30.318: CFM-PKT: Send LM Msg level 7, cos 3, port mac 0050.56a9.dc63 to
interface Gi2

```

Next, we configure a basic Ethernet jitter probe on CSR2. This probe sends 10 frames 100 ms apart for a total of 1 second transmission time.

```

! CSR2
ip sla 25
 ethernet jitter mpid 205 domain CUSTOMER port interval 100 num-frames 10
 cos 4
 threshold 3000
 frequency 5
ip sla schedule 25 life forever start-time now

```

Unfortunately, this does not appear to work. I also tried it within the provider/operator networks to no avail. While Y.1731 is not supported on port MEPs or untagged EFPs, this probe is not specific to Y.1731 and should work. I leave it in the configuration for completeness though the router is not even sending probes.

```

R2#show ip sla statistics 25 | include ^Number
Number of successes: 0
Number of failures: 4

```

Since the Ethernet echo operation appears to work, we will use the ethernet-monitor feature to spawn echo probes on CSR3. This will occur inside of the provider CFM domain and use EVC200. For each RMEP inside of this EVC, an Ethernet echo probe is spawned to test reachability. This runs once per minute and is targeting MPID 160, which was gleaned from the remote MEP database. The probe number is 100001 which we learned from the summary show command. We can also see the statistics for this specific probe by using the show command below. This feature appears to work identically in the CSR1000v as it did in the ME3400.

```

! CSR3
ip sla ethernet-monitor 36
 type echo domain PROVIDER evc EVC200
ip sla ethernet-monitor schedule 36 schedule-period 60 start-time now

```

```

R3#show ip sla summary
IPSLAs Latest Operation Summary
Codes: * active, ^ inactive, ~ pending

```

ID	Type	Destination	Stats (ms)	Return Code	Last Run

```
-----
*100001      ethernet-ec Domain:PROVIDER M RTT=13      OK      8 seconds
ago
      ho      pid:160 Evc:EVC20
```

```
R3#show ip sla statistics 100001 | include ^Number
Number of successes: 2
Number of failures: 0
```

Next, we will configure Y.1731 delay and loss measurements. Since 1DM was not supported at all on the ME3400, we attempt to test that here. This only measures one-way delay between two points and the receiving end must be configured explicitly to support it. It also requires NTP synchronization, which we quickly configure first.

```
! CSR6
ntp master
```

```
! CSR7
ntp source Loopback0
ntp server 131.0.0.6 minpoll 4 maxpoll 4
```

```
R6#show ntp status | include sync
Clock is synchronized, stratum 8, reference is 127.127.1.1
```

```
R7#show ntp status | include sync
Clock is synchronized, stratum 9, reference is 131.0.0.6
```

Next, we will configure CSR7 as receive-only for these one-way delay measurement (1DM) frames. We can adjust the distribution buckets just like we did on the ME-3400 switches, as well as adjust some other minor options. Notice that this probe is in “receive” mode and only specified the local (target) MPID. This is loosely analogous to an IP SLA responder, except specific to Y.1731 1DM operations.

```
! CSR7
ip sla 100
  ethernet y1731 delay receive 1DM domain OPERATOR evc EVC100 cos 0 mpid 107
  max-delay 500
  aggregate interval 60
  distribution delay one-way 4 2500,5000,7500,4294967295
  distribution delay-variation one-way 3 2000,4000,4294967295
ip sla schedule 100 life forever start-time now
```

CSR6’s probe looks more standard as it probes MPID 107 from MPID 106. The configuration parameters are the same for consistency.

```
!CSR6
ip sla 100
```

```
ethernet y1731 delay 1DM domain OPERATOR evc EVC100 mpid 107 cos 0 source
mpid 106
max-delay 500
aggregate interval 60
distribution delay one-way 4 2500,5000,7500,4294967295
distribution delay-variation one-way 3 2000,4000,4294967295
ip sla schedule 100 life forever start-time now
```

A quick check of the probe statistics shows that it is successful. Since the frames are one-way, CSR6 initiates while CSR7 completes.

```
R6#show ip sla statistics 100
IPSLAs Latest Operation Statistics

IPSLA operation id: 100
Delay Statistics for Y1731 Operation 100
Type of operation: Y1731 Delay Measurement
Latest operation start time: 20:54:02.183 UTC
Latest operation return code: OK
Distribution Statistics:

Interval
Start time: 20:54:02.183 UTC
End time: 20:54:39.708 UTC
Number of measurements initiated: 38
Number of measurements completed: 0
Flag: OK
```

```
R7#show ip sla statistics 100
IPSLAs Latest Operation Statistics

IPSLA operation id: 100
Delay Statistics for Y1731 Operation 100
Type of operation: Y1731 Delay Measurement
Latest operation start time: 20:50:02.186 UTC
Latest operation return code: OK
Distribution Statistics:

Interval
Start time: 20:50:02.186 UTC
Elapsed time: 533 seconds
Number of measurements initiated: 0
Number of measurements completed: 52
Flag: OK
```

The detailed statistics for Y.1731 do not seem to ever populate despite the probe succeeding. The date is locked in the year 1900 and none of the statistics ever increment, even with NTP synchronized.


```
R7#show ip sla statistics 100 details | begin ^Delay:
```

```
Delay:
```

```
Number of forward observations: 0
```

```
Min/Avg/Max forward: 0/0/0 (microsec)
```

```
Time of occurrence forward:
```

```
Min - 00:00:00.000 UTC Mon Jan 1 1900
```

```
Max - 00:00:00.000 UTC Mon Jan 1 1900
```

```
Bin forward:
```

Bin Range (microsec)	Total observations
0 - < 2500	0
2500 - < 5000	0
5000 - < 7500	0
7500 - < 4294967295	0

```
[snip]
```

To be absolutely sure the Y.1731 delay frames are being transmitted, we can enable debugging on CSR6 (transmitter) and CSR7 (receiver). CSR7's brain MAC is highlighted in yellow and CSR6's brain MAC is in green. We can clearly see the frames flowing in one direction as expected. Ethertype 0x8902 is shown in cyan which is used for CFM and Y.1731 as seen many times earlier.

```
R7#show ethernet cfm statistics | include BRAIN
```

```
BRAIN MAC: 001e.49ca.a4bf
```

```
R6#show ethernet cfm statistics | include BRAIN
```

```
BRAIN MAC: 001e.bd69.62bf
```

```
R6#debug ethernet cfm pm packets
```

```
Ethernet PM Packet debugging is on
```

```
20:55:27.246: EPM-PKT(session ID: 0) sec 56522BBF nsec EA944CA
20:55:27.246: 00 1E 49 CA A4 BF 00 1E BD 69 62 BF 89 02
20:55:27.246: 20 2D 00 10 56 52 2B BF 0E A9 44 CA 00 00
20:55:27.246: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20:55:27.246: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20:55:27.246: 00 00 00 00 00 00 00 00 00
20:55:27.246: EPM-PKT(session ID: 0) Number of frames xmit 26
20:55:28.246: EPM-PKT(session ID: 0) sec 56522BC0 nsec EA944CA
20:55:28.246: 00 1E 49 CA A4 BF 00 1E BD 69 62 BF 89 02
20:55:28.246: 20 2D 00 10 56 52 2B C0 0E A9 44 CA 00 00
20:55:28.246: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20:55:28.246: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20:55:28.246: 00 00 00 00 00 00 00 00 00
20:55:28.246: EPM-PKT(session ID: 0) Number of frames xmit 27
```

Debugging on CSR7 reveals similar output. These are not the exact same packets as above, but upon cursory review, appears to indicate receiving the Y.1731 1DM frames and processing them correctly. We

never saw these issues on the ME-3400, and I suspect it has something to do with Y.1731 limitations over EoMPLS/Xconnect.

```
! CSR7
20:57:13.267: EPM-PKT(session ID: 0) curr nano 263978658 0 0 0, prev nano
264985686 0 0 0
20:57:13.267: EPM-PKT(session ID: 0) T[c] 31 T[p] 30
20:57:14.267: EPM-PKT(session ID: 0) Total 1DM rx 433
20:57:14.267: 00 1E 49 CA A4 BF 00 1E BD 69 62 BF 81 00
20:57:14.267: 00 00 81 00 00 00 89 02 20 2D 00 10 56 52
20:57:14.267: 2C 2A 0F BB FE A2 00 00 00 00 00 00 00 00
20:57:14.267: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20:57:14.267: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20:57:14.267: 00 00 45 C0 00 28 98 4B 00 00 FD 06 1E B7
20:57:14.267: 83 00
20:57:14.267: EPM-PKT(session ID: 0) curr nano 263978658 0 0 0, prev nano
263978658 0 0 0
20:57:14.267: EPM-PKT(session ID: 0) T[c] 32 T[p] 31
20:57:15.267: EPM-PKT(session ID: 0) Total 1DM rx 434
20:57:15.267: 00 1E 49 CA A4 BF 00 1E BD 69 62 BF 81 00
20:57:15.267: 00 00 81 00 00 00 89 02 20 2D 00 10 56 52
[snip]
```

For testing loss, only LMM is supported in XR version 3.13.2S, which means loss is measured by examining data traffic rather than synthesizing it. For this test, I will probe across both operator networks using the provider MEPs. There is no need to configure anything on CSR10, the target, as that receiver-oriented configuration is specific to 1DM operations. As a result, CSR6 sees both the probe initiations and completions, which makes the probe appear functional.

```
! CSR6
ip sla 200
  ethernet y1731 loss LMM domain PROVIDER evc EVC200 mpid 203 cos 0 source
mpid 160
  aggregate interval 60
ip sla schedule 200 life forever start-time now

R6#show ip sla statistics 200
IPSLAs Latest Operation Statistics

IPSLA operation id: 200
Loss Statistics for Y1731 Operation 200
Type of operation: Y1731 Loss Measurement
Latest operation start time: 21:12:09.475 UTC
Latest operation return code: OK
Distribution Statistics:

Interval
Start time: 21:12:09.475 UTC
```

```
Elapsed time: 36 seconds
Number of measurements initiated: 31
Number of measurements completed: 31
Flag: OK
```

We start a continuous ping from CSR5 to CSR2 to generate some extra user traffic (not shown). However, we see the same negative result from Y.1731. No detailed statistics are being captured, even though we are certain traffic is flowing across the monitored EVC. The time is still frozen in the year 1900.

```
R6#show ip sla statistics 200 details | begin ^Forward
Forward
  Number of Observations 0
  Available indicators: 0
  Unavailable indicators: 0
  Tx frame count: 0
  Rx frame count: 0
  Min/Avg/Max - (FLR % ): 0:0/000.00%/0:0
  Cumulative - (FLR % ): 000.0000%
  Timestamps forward:
    Min - 00:00:00.000 UTC Mon Jan 1 1900
    Max - 00:00:00.000 UTC Mon Jan 1 1900
[snip]
```

Debugging on CSR6 reveals the LMM/LMR exchange for this probe as well. When an LMM is transmitted an LMR is received almost immediately. CSR3's brain MAC is the target of the probe and is in yellow, while CSR6's brain MAC is the source in green.

```
R3#show ethernet cfm statistics | include BRAIN
BRAIN MAC: 001e.e5a8.ffbf
```

```
R6#show ethernet cfm statistics | include BRAIN
BRAIN MAC: 001e.bd69.62bf
```

```
R6#debug ethernet cfm pm packets session 1
Ethernet PM Packet debugging for session 1 is on
```

```
21:25:48.653: EPM-PKT(session ID: 1) Tx_f 39 TxFcf 0 RxFcf 0 TxFcb 0 RxFcb 0
Tx_p 38 Prev TxFcf 0 RxFcf 0 TxFcb 0 RxFcb 0 rpol 0x7F3FE1DA30B8 stat
0x7F3FE0F4AF48
21:25:49.616: 00 1E E5 A8 FF BF 00 1E BD 69 62 BF 89 02
21:25:49.616: 80 2B 00 0C 00 00 00 00 00 00 00 00 00 00
21:25:49.616: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
21:25:49.616: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
21:25:49.616: 00 00 00 00 00 00 00 00
21:25:49.616: EPM-PKT(session ID: 1) Number of frames xmit 41
21:25:49.637: EPM-PKT(session ID: 1) Received LMR. Total LMR rx for this
session 41
```

```

21:25:49.637: 00 1E BD 69 62 BF 00 1E E5 A8 FF BF 89 02
21:25:49.637: 80 2A 00 0C 00 00 00 00 00 00 00 00 00 00
[snip]

```

To troubleshoot this further, we will look at the raw Y.1731 PM information within the CFM syntax tree. This shows both IP SLA operations in summary form. Both of them appear to be successfully registered as Y.1731 probes.

```
R6#show ethernet cfm pm
```

EPM-ID	SLA-ID	Lvl/Type/ID/Cos/Dir	Src-Mac-address	Dst-Mac-address
0	100	1/XCON/N/A/0/Up	001e.bd69.62bf	001e.49ca.a4bf
1	200	4/XCON/N/A/0/Up	001e.bd69.62bf	001e.e5a8.ffbf

First, we will look at session 0 which is the 1DM probe. We can see that the probe is actively transmitting but not receiving, which appears correct. 1DM frames are being received in the statistics, which is odd because the frames are only flowing in one direction. This may be a cosmetic decision or it may represent some 1DM control-specific response. We also see VSM/VSR messages, which were not discussed earlier. These are vendor-specific messages and replies which vary between vendors. I assume that Cisco is using these messages to report one-way delay measurements from target to source, which is why there are many more VSRs than VSMs. There does not appear to be anything terribly wrong here other than the date stuck in the year 1900.

```

R6#show ethernet cfm pm session detail 0
Session ID: 0
Sla Session ID: 100
Level: 1
Service Type: XCON
Service Id: N/A
Direction: Up
Source Mac: 001e.bd69.62bf
Destination Mac: 001e.49ca.a4bf
Session Version: 0
Session Operation: Proactive
Session Status: Active
MPID: 106
Tx active: yes
Rx active: no
RP monitor Tx active: yes
RP monitor Rx active: no
Timeout timer: stopped
Last clearing of counters: 00:00:00.000 UTC Mon Jan 1 1900
DMMs:
    Transmitted: 0
DMRs:
    Rcvd: 0

```

```
1DMs:
    Transmitted: 666
    Rcvd: 668
LMs:
    Transmitted: 0
LMRs:
    Rcvd: 0
VSMs:
    Transmitted: 14
VSRs:
    Rcvd: 79
SLMs:
    Transmitted: 0
SLRs:
    Rcvd: 0
Test ID 0
```

We also verify the LMM probe across the provider CFM domain to CSR3. There are no vendor-specific exchanges here, and we see LMM/LMR messages being exchanged as expected. The debug revealed this as well, but we can see the counters increment here also. This probe is both TX/RX active as expected.

```
R6#show ethernet cfm pm session detail 1
Session ID: 1
Sla Session ID: 200
Level: 4
Service Type: XCON
Service Id: N/A
Direction: Up
Source Mac: 001e.bd69.62bf
Destination Mac: 001e.e5a8.ffbf
Session Version: 0
Session Operation: On-demand
Session Status: Active
MPID: 160
Tx active: yes
Rx active: yes
RP monitor Tx active: yes
RP monitor Rx active: yes
Timeout timer: stopped
Last clearing of counters: 00:00:00.000 UTC Mon Jan 1 1900
DMMs:
    Transmitted: 0
DMRs:
    Rcvd: 0
1DMs:
    Transmitted: 0
    Rcvd: 0
LMMs:
```

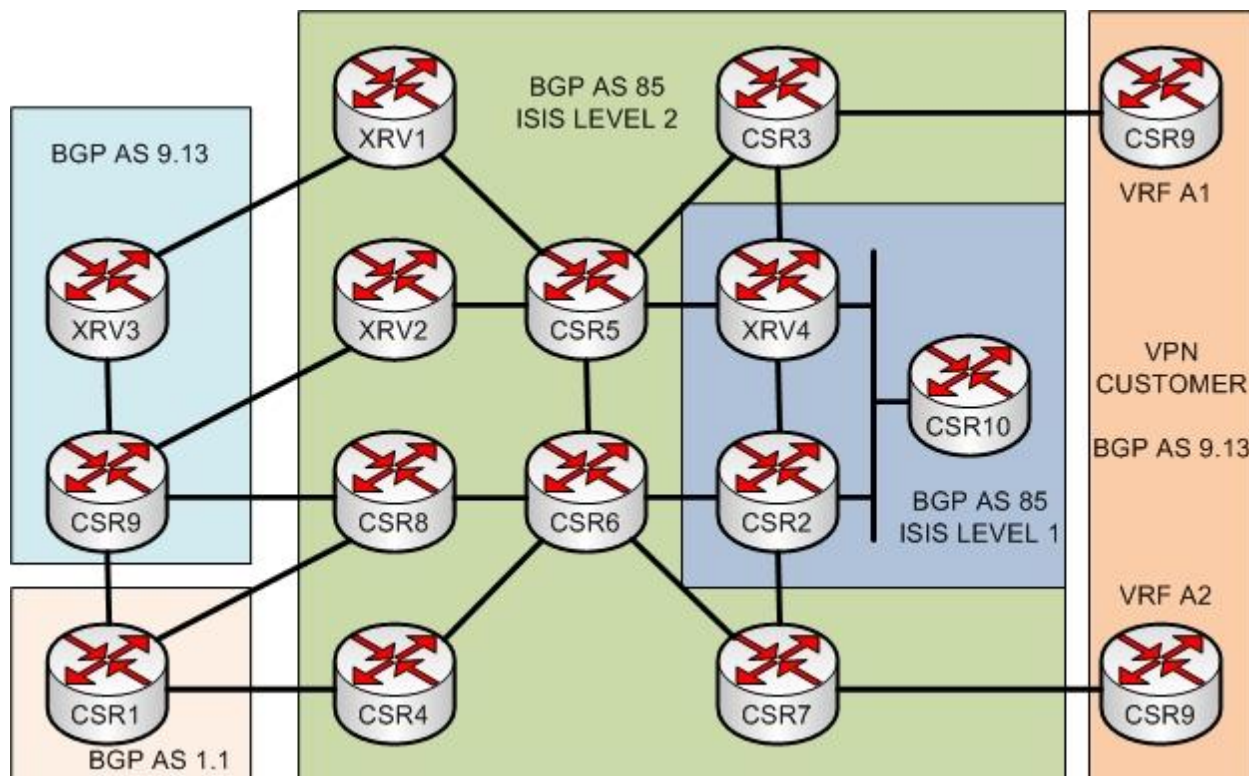
```
Transmitted: 760
LMRs:
Rcvd: 760
VSMs:
  Transmitted: 0
VSRs:
  Rcvd: 0
SLMs:
  Transmitted: 0
SLRs:
  Rcvd: 0
Test ID 0
```

In summary, Ethernet management functionality works similarly on the CSR1000v as it does on the ME3400. As documented, the router has additional limitations when interworking with EoMPLS, port MEPs, untagged EFPs, and a slew of other non-obvious design constraints. CFM and OAM are the simplest features and “just work”. E-LMI and Y.1731 PM are less forgiving and have additional restrictions; always consult the documentation when configuring complex layer-2 features on routing platforms.

Additional Reading – Reference configurations "eth-oam-csr"

49. Service Provider security best practices (Comprehensive)

Many of the SP security topics listed in the blueprint are independent security topics that are most effective when combined together. I use a large security lab to demonstrate many of these explicitly-defined blueprint topics, along with Cisco’s SP security best practices document. Cisco breaks down their security recommendations into three sections: control plane, management plane, and data plane. Some of the very basic techniques, such as banners, usernames, passwords, and session-timeouts, may not be fully demonstrated since I don’t want to deal with those “timer wasters” every time I access a device for testing. The topology is very large and complex intentionally as there are more than 50 techniques discussed, configured, and tested here. The diagram is shown below, along with all configurations. AS 85 encompasses the routers in “our” network. The ASes on the left are for testing external connectivity. CSR9 has a few VRFs to test L3VPN between CSR3 and CSR7, but this is used only briefly.



Additional Reading – Reference configurations "security"

49.1 Control plane security best practices

1. **Ingress BGP prefix filtering from L3VPN customers:** This refers to basic BGP filtering from L3VPN customers, such as removing bogons, checking for bogus AS paths, etc. The network only has one customer, which includes CSR9 running a pair of VRFs for a basic L3VPN between CSR3 and CSR7 (PEs). Filtering bogons is simple; create a prefix-list based on the current bogon listing and apply it inbound facing the peer. I used two different logical methods for this. The IPv4 bogon list permits the bogons, and the route-map clause denies them. The IPv6 bogon list specifies only the permitted routes (non-bogons), and the route-map clause permits them. This is because the IPv6 bogon list is enormous compared to the IPv4 one, so the opposite logical implementations make sense. The significant difference between inter-SP and customer filtering is that customers often use RFC1918 addressing, so our IPv4 bogon lists at our PEs should not filter this. The show commands for the lists are revealed as the configurations are very long; both lists are configured on both CSR3 and CSR7.

```
R7#show ip prefix-list PL_BOGON_LIST
ip prefix-list PL_BOGON_LIST: 10 entries
  seq 5 permit 0.0.0.0/8 le 32
  seq 15 permit 100.64.0.0/10 le 32
  seq 20 permit 127.0.0.0/8 le 32
  seq 25 permit 169.254.0.0/16 le 32
  seq 35 permit 192.0.0.0/24 le 32
  seq 40 permit 192.0.2.0/24 le 32
```

```

seq 50 permit 198.18.0.0/15 le 32
seq 55 permit 198.51.100.0/24 le 32
seq 60 permit 203.0.113.0/24 le 32
seq 65 permit 224.0.0.0/3 le 32

```

```

R7#show ipv6 prefix-list PL_IPV6_ALLOWED_ROUTES
ipv6 prefix-list PL_IPV6_ALLOWED_ROUTES: 38 entries
  seq 5 deny 2001:DB8::/32 le 128
  seq 10 permit 2001::/32
  [snip]
  seq 190 permit 2C00::/12 le 64

```

We wrap them in a route-map and apply to the peers within the VRF. The configuration is shown from the perspective of CSR7, but CSR3 has an identical configuration, addresses notwithstanding.

```

! CSR7
route-map RM_CUSTOMER_IN deny 10
  match ip address prefix-list PL_BOGON_LIST
route-map RM_CUSTOMER_IN permit 1000

route-map RM_CUSTOMER_IPV6_IN permit 10
  match ipv6 address prefix-list PL_IPV6_ALLOWED_ROUTES

router bgp 85
  address-family ipv4 vrf A
    neighbor 172.16.79.9 route-map RM_CUSTOMER_IN in
  address-family ipv6 vrf A
    neighbor FD00:172:16:79::9 route-map RM_CUSTOMER_IPV6_IN in

```

To verify this configuration, we have some bogon addressing configured within CSR9's test VRFs (192.0.0.91/32 and 192.0.0.92/32, 2FFF::91/128 and 2FFF::92/128) we can use for testing. These are advertised into BGP along with all of the legitimate addresses within the L3VPN (not shown).

```

! CSR9
interface Loopback91
  vrf forwarding A1
  ip address 192.0.0.91 255.255.255.255 secondary
  ipv6 address 2FFF::91/128

interface Loopback92
  vrf forwarding A2
  ip address 192.0.0.92 255.255.255.255 secondary
  ipv6 address 2FFF::92/128

```

Real customer addresses use the 172.16.0.0/16 range. We see that on CSR9, only the 172.16.0.0/16 routes were distributed across the SP. CSR3 and CSR7 specifically say they filtered out those bogon routes. We have full reachability between the proper VPN routes though, but not the bogons.


```

R9#show bgp vpnv4 unicast all | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 9:1 (default for vrf A1)
*> 172.16.9.1/32      0.0.0.0              0          32768 i
*> 172.16.9.2/32      172.16.93.3          0          0 85 85 i
*> 192.0.0.91/32      0.0.0.0              0          32768 i
Route Distinguisher: 9:2 (default for vrf A2)
*> 172.16.9.1/32      172.16.79.7          0          0 85 85 i
*> 172.16.9.2/32      0.0.0.0              0          32768 i
*> 192.0.0.92/32      0.0.0.0              0          32768 i

R3#show ip prefix-list detail PL_BOGON_LIST | include hit count: [1-9]+
seq 35 permit 192.0.0.0/24 le 32 (hit count: 1, refcount: 1)

R7#show ip prefix-list detail PL_BOGON_LIST | include hit count: [1-9]+
seq 35 permit 192.0.0.0/24 le 32 (hit count: 1, refcount: 1)

R9#ping vrf A1 172.16.9.2 source 172.16.9.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.9.2, timeout is 2 seconds:
Packet sent with a source address of 172.16.9.1
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/6/20 ms

R9#ping vrf A1 192.0.0.92 source 192.0.0.91
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.0.0.92, timeout is 2 seconds:
Packet sent with a source address of 192.0.0.91
.....
Success rate is 0 percent (0/5)

```

The same series of verifications and data plane tests are quickly repeated here for IPv6. The PEs do not transport IPv6 bogons and show hit counts under the permitted route ACL entries.

```

R9#show bgp vpnv6 unicast all | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 9:1 (default for vrf A1)
*> 2003:0:0:91::/64   ::              0          32768 ?
*> 2003:0:0:92::/64   FD00:172:16:93::3
                                0 85 85 ?
*> 2FFF::91/128       ::              0          32768 ?
Route Distinguisher: 9:2 (default for vrf A2)
*> 2003:0:0:91::/64   FD00:172:16:79::7
                                0 85 85 ?
*> 2003:0:0:92::/64   ::              0          32768 ?
*> 2FFF::92/128       ::              0          32768 ?

```

```
R3# show ipv6 prefix-list detail PL_IPV6_ALLOWED_ROUTES | include hit count:
[1-9]+
```

```
seq 155 permit 2003::/18 le 64 (hit count: 3, refcount: 3)
```

```
R7# show ipv6 prefix-list detail PL_IPV6_ALLOWED_ROUTES | include hit count:
[1-9]+
```

```
seq 155 permit 2003::/18 le 64 (hit count: 2, refcount: 3)
```

```
R9#ping vrf A1 2003:0:0:92::92 source 2003:0:0:91::91
```

```
Type escape sequence to abort.
```

```
Sending 5, 100-byte ICMP Echos to 2003:0:0:92::92, timeout is 2 seconds:
```

```
Packet sent with a source address of 2003:0:0:91::91%A1
```

```
!!!!!!
```

```
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/6/20 ms
```

```
R9#ping vrf A1 2FFF::92 source 2FFF::91
```

```
Type escape sequence to abort.
```

```
Sending 5, 100-byte ICMP Echos to 2FFF::92, timeout is 2 seconds:
```

```
Packet sent with a source address of 2FFF::91%A1
```

```
.....
```

```
Success rate is 0 percent (0/5)
```

A more difficult problem to solve is ensuring that customers don't add bogus AS paths to their AS path lists. Our prefix-based bogon filter cannot prevent this. Customers should be allowed to prepend their own AS for traffic engineering, but prepending other ASes is not permitted. Using advanced regular expressions, we can solve this problem dynamically. Regex parenthesis () means evaluate the expression, then take the value and put it into memory slot 1. The next set of () goes into memory slot 2, etc. If you escape a digit (\1), this references the contents of memory slot 1, and the router can store up to 9 memory slots. On the SP side, use `^([0-9]+)(\1)*$` to match a single AS repeated one or more times. We will define this in an AS-path ACL and integrate it with our route-maps.

```
! CSR3 and CSR7
```

```
ip as-path access-list 85 per ^([0-9]+)(\1)*$
```

How we integrate this new filter with our existing bogon filter depends on the logic of the route-map. With IPv4, we add this to the default permit clause at the end of the route-map so we could add more specific deny clauses later, or possibly exceptions to this prepend security. For IPv6, we include this with our bogon filter to form a Boolean AND. This concludes the SP configuration; it is easy, precise, and dynamic. The AS-path ACL can be reused for any customer regardless of AS, which allows the configuration to scale (no need to make a custom regex per customer).

```
! CSR3 and CSR7
```

```
route-map RM_CUSTOMER_IN permit 1000
```

```
match as-path 85
```

```
route-map RM_CUSTOMER_IPV6_IN permit 10
```

```
match as-path 85
```

I also add some networks to CSR9, some of which are properly prepend, others are not. For IPv4, if the third octet is “97” then the customer prepends incorrectly and the PEs reject it. If the third octet is “99”, the customer prepends properly, and the PE accepts it. This basic configuration is very long but is critical to test the AS-path filter in both directions.

```
! CSR9
interface Loopback91
  vrf forwarding A1
  ip address 192.168.97.91 255.255.255.255 secondary
  ip address 192.168.99.91 255.255.255.255 secondary

interface Loopback92
  vrf forwarding A2
  ip address 192.168.99.92 255.255.255.255 secondary
  ip address 192.168.97.92 255.255.255.255 secondary

ip prefix-list PREPEND_RIGHT seq 5 permit 192.168.99.0/24 le 32
ip prefix-list PREPEND_WRONG seq 5 permit 192.168.97.0/24 le 32

route-map RM_IPV6_OUT permit 10
  match ipv6 address prefix-list IPV6_PREPEND_WRONG
  set as-path prepend 589837 7 589837
route-map RM_IPV6_OUT permit 20
  match ipv6 address prefix-list IPV6_PREPEND_RIGHT
  set as-path prepend 589837 589837 589837
route-map RM_IPV6_OUT permit 1000

route-map RM_IPV4_OUT permit 10
  match ip address prefix-list PREPEND_WRONG
  set as-path prepend 589837 7 589837
route-map RM_IPV4_OUT permit 20
  match ip address prefix-list PREPEND_RIGHT
  set as-path prepend 589837 589837 589837
route-map RM_IPV4_OUT permit 1000

router bgp 589837
  address-family ipv4 vrf A1
    neighbor 172.16.93.3 route-map RM_IPV4_OUT out
  address-family ipv6 vrf A1
    neighbor FD00:172:16:93::3 route-map RM_IPV6_OUT out
  address-family ipv4 vrf A2
    neighbor 172.16.79.7 route-map RM_IPV4_OUT out
  address-family ipv6 vrf A2
    neighbor FD00:172:16:79::7 route-map RM_IPV6_OUT out
```

We can verify this by checking VPN routes on CSR9. The “97” routes never make it across the PE network and are filtered due to their AS paths. The “99” networks are advertised across the VPN since they appear in opposite VRFs. We also see the provider’s AS path has overwritten all instances of the local AS thanks to the “as-override” feature on CSR3 and CSR7.

```
R9#show bgp vpnv4 unicast all 192.168.64.0/18 longer-prefixes | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 9:1 (default for vrf A1)
*> 192.168.97.91/32 0.0.0.0              0          32768 ?
*> 192.168.99.91/32 0.0.0.0              0          32768 ?
*> 192.168.99.92/32 172.16.93.3              0 85 85 85 85 85 ?
Route Distinguisher: 9:2 (default for vrf A2)
*> 192.168.97.92/32 0.0.0.0              0          32768 ?
*> 192.168.99.91/32 172.16.79.7              0 85 85 85 85 85 ?
*> 192.168.99.92/32 0.0.0.0              0          32768 ?
```

```
R9#ping vrf A1 192.168.99.92 source 192.168.99.91
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.99.92, timeout is 2 seconds:
Packet sent with a source address of 192.168.99.91
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/6/20 ms
```

```
R9#ping vrf A1 192.168.97.92 source 192.168.97.91
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.97.92, timeout is 2 seconds:
Packet sent with a source address of 192.168.97.91
.....
Success rate is 0 percent (0/5)
```

Performing the same verification for IPv6, we see that the IPv6 prefixes contained “979” have improper prepending applied and are rejected by the PEs. Prefixes that have “999” are properly prepended and are accepted.

```
R9#show bgp vpnv6 unicast all 2003:0:0:9000::/52 longer-prefixes | begin
Network
      Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 9:1 (default for vrf A1)
*> 2003:0:0:9791::/64
      ::              0          32768 ?
*> 2003:0:0:9991::/64
      ::              0          32768 ?
*> 2003:0:0:9992::/64
      FD00:172:16:93::3
      0 85 85 85 85 85 ?
Route Distinguisher: 9:2 (default for vrf A2)
*> 2003:0:0:9792::/64
      ::              0          32768 ?
```

```
*> 2003:0:0:9991::/64
                                FD00:172:16:79::7
                                0 85 85 85 85 85 ?
*> 2003:0:0:9992::/64
                                ::
                                0 32768 ?
```

```
R9#ping vrf A1 2003:0:0:9992::92 source 2003:0:0:9991::91
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2003:0:0:9992::92, timeout is 2 seconds:
Packet sent with a source address of 2003:0:0:9991::91%A1
!!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/6/20 ms
```

```
R9#ping vrf A1 2003:0:0:9792::92 source 2003:0:0:9791::91
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2003:0:0:9792::92, timeout is 2 seconds:
Packet sent with a source address of 2003:0:0:9791::91%A1
.....
Success rate is 0 percent (0/5)
```

2. **Intra-AS filtering:** Control unnecessarily advertisement of iBGP prefixes via RRs if needed. This is less of a technical feature as it is a policy control mechanism. The intent is to use standard BGP filtering techniques within an AS, likely on the RRs, to not distribute all information to all routers, unless necessary. This may include using the XE BGP RR-group or XR RT-retention features on RRs to conserve memory or limit route propagation. There is no demonstration for this task as no new technologies or methods are introduced. The general logic of this recommendation is to only advertise routes across the network to devices that actually require reachability.

3. **Baseline the routing table:** This technique helps detect routing anomalies over time. This is more of a policy decision than a technical one, with the exception of the “route profile” utility. This is not available in XR and appears to only support IPv4 at this time in XE. I enable this feature on CSR2 as a demonstration. The intent is to monitor changes in the routing table over 5 second intervals. The first column (change/interval) says “I saw this many changes in the last 5 seconds”. The remaining columns are tallies for the number of times an event occurred. In this example, there were 6 intervals whereby zero new prefixes were added, for example. This output shows a stable routing table.

```
! CSR2
ip route profile
```

```
R2#show ip route profile
IP routing table change statistics:
Frequency of changes in a 5 second sampling interval
```

```
-----
Change/   Fwd-path  Prefix   Nexthop  Pathcount  Prefix
interval change    add      change   change     refresh
-----
```

```

0          6          6          6          6          6
1          0          0          0          0          0
2          0          0          0          0          0
3          0          0          0          0          0
4          0          0          0          0          0
5          0          0          0          0          0
10         0          0          0          0          0
[snip]

```

If I clear IS-IS and BGP several times (not shown), we can see several routing changes in this profile. This doesn't tell us which routes changed, but it gives a rough estimate. It helps diagnose instability when dealing with thousands of routes. In this output, we see that within a single 5 second interval, we had between 10 and 14 fwd-path changes. We also had several prefix additions and path count changes at one time as well. Towards the end of the convergence cycle, we had a single route fwd-path change in its own 5 second interval. Low numbers in these columns, like 1 or 2, indicate normal, occasional reconvergence events. If the numbers are growing very quickly, or the row index (change/interval) is very large, chances are there is a routing issue or extreme churn. Notice that the "0" row has the highest values because most of the time, the routing table remains stable.

```

R2#show ip route profile
IP routing table change statistics:
Frequency of changes in a 5 second sampling interval

```

```

-----
Change/  Fwd-path  Prefix  Nexthop  Pathcount  Prefix
interval change   add    change   change     refresh
-----
0         82        83     85       83         85
1         1         0      0         1          0
2         0         0      0         0          0
3         0         0      0         0          0
4         0         1      0         0          0
5         1         1      0         1          0
10        1         0      0         0          0
15        0         0      0         0          0
20        0         0      0         0          0
[snip]

```

4. **Authenticate routing updates for BGP:** This prevents attacker from inserting information. On XE boxes, only a single MD5 password is allowed, but XR has support for key-chains. In order for the key-chains to work, you must specify the crypto method of HMAC-MD5, otherwise BGP will not send the hash. I made a rotating key-chain to change keys each month (with a 2 minute overlap) for the rest of calendar year 2015. For brevity, I only show the months of July and August below; this shows the rollover configuration logic. Note that this method requires the devices to have time synchronization.

```

! XRv1, XRv2, and XRv4
key chain KC_BGP_AUTH

```

```

key 7
  accept-lifetime 23:58:00 june 30 2015 23:59:59 july 31 2015
  key-string password 080B796230262E322B
  send-lifetime 23:58:00 june 30 2015 23:59:59 july 31 2015
  cryptographic-algorithm HMAC-MD5
key 8
  accept-lifetime 23:58:00 july 31 2015 23:59:59 august 31 2015
  key-string password 106F3C3E30242634272113
  send-lifetime 23:58:00 july 31 2015 23:59:59 august 31 2015
  cryptographic-algorithm HMAC-MD5

```

Using the key-chain is considered enhanced authentication (EA) and is not supported by XE, so I only enabled it between XR routers for extra security. On XR, I have a separate session-group for the XE routers that only uses the password method. Using inheritance of session-groups, we use the same basic iBGP session parameters for all iBGP peers, but use different authentication methods between platforms based on their capabilities. The application of the session-group to all peers is not shown. Other peers have been configured with the proper key-chains and passwords as required.

```

! XRv4
session-group IBGP_XE
  use session-group IBGP_BASIC
  password encrypted 072D067C71282C313F

session-group IBGP_XR
  use session-group IBGP_BASIC
  keychain KC_BGP_AUTH

```

We can verify MD5 authentication by looking at the TCP information. XR has a succinct way of displaying this; we can see a brief of all BGP peers along with their authentication type. All XE routers use MD5 while the XR routers use EA (which is MD5-based as well).

```

RP/0/0/CPU0:XRv4#show tcp authentication database checkpoint | include BGP
0x40002fd8  8.8.8.8          BGP  0x60000000  MD5  0x00004cb8
0x40002e38  4.4.4.4          BGP  0x60000000  MD5  0x00004cb8
0x40002dd8  12.12.12.12     BGP  0x60000000  EA   0x0002680d
0x40002e98  11.11.11.11     BGP  0x60000000  EA   0x0002680d
0x40002e58  7.7.7.7         BGP  0x60000000  MD5  0x00004cb8
0x40002e18  3.3.3.3         BGP  0x60000000  MD5  0x00004cb8
0x40002df8  10.10.10.10     BGP  0x60000000  MD5  0x00004cb8

```

There are some restrictions on EA. Specifically, **you must specify a time range**, or else you get this error. Even if that time-range is infinite, it must be configured, which is different than some other protocols that rely on key-chains for authentication.

```
! XRv4
```

```
IP-TCP-3-EAOPT_SEND_NOKEY : No active send-key for key-chain 'KC_BGP_CUST'
for the Enhanced Authentication option to send to peer 10.11.14.14, error -
'Key Chain Server Process' detected the 'fatal' condition 'No key available'
```

Additionally, the key ID must fit into 6 bits (63 or less), or else you get this error when committing the candidate configuration. If using monthly keys, I recommend using the month number (1 through 12). If using weekly keys, use the week number (1 through 52).

```
! XRv4
Send key-id 100 for keychain 'KC_BGP_CUST' (peer 10.11.14.11) greater than
Enhanced Authentication max(63)
```

On XE, we can check the TCP database for the TCP control block (TCB) for a specific BGP session, then check the negotiated option flags. MD5 is option 19 for TCP (this number isn't revealed here), so ensure any firewalls/data-plane filters in the network do not drop packets carrying this option.

```
R2#show tcp brief | include 11\11
7F5318233800 2.2.2.2.179 11.11.11.11.25413 ESTAB
```

```
R2#show tcp tcb 7F5318233800 | include ^Option
Option Flags: nagle, path mtu capable, md5, Retrans timeout
```

5. **Limit BGP prefix counts:** This feature prevents a flood of BGP routes from an external peer. This is enabled using the “maximum-prefix” command in XE and XR and is AFI-specific. It sets a limit on how many prefixes can be received from a BGP peer, along with a warning threshold and restart timer. If the restart timer is not set, the neighbor will never advertise any routes until it is hard-cleared. Below are the configurations and verifications for IPv4 and IPv6 across XR and XE. We will apply these changes inside the eBGP AFI templates so that it automatically applies to all eBGP peers.

```
! XRv1
router bgp 85
  af-group AF_IPV4_EBGP address-family ipv4 unicast
    maximum-prefix 100 50
  af-group AF_IPV6_EBGP address-family ipv6 unicast
    maximum-prefix 100 50
```

```
RP/0/0/CPU0:XRv1#show bgp ipv6 unicast neighbors 10.1.3.13 | utility egrep
'For Address|Maximum|warn'
```

```
For Address Family: IPv4 Unicast
  Maximum prefixes allowed 100
  Threshold for warning message 50%, restart interval 0 min
For Address Family: IPv6 Unicast
  Maximum prefixes allowed 100
  Threshold for warning message 50%, restart interval 0 min
```

```
! CSR8
router bgp 85
```



```

template peer-policy IPV4_EBGP
  maximum-prefix 100 50
template peer-policy IPV6_EBGP
  maximum-prefix 100 50

R8#show bgp ipv4 unicast neighbors 10.1.8.1 | include For_add|Maximum|warn
For address family: IPv4 Unicast
  Maximum prefixes allowed 100
  Threshold for warning message 50%
For address family: IPv6 Unicast
  Maximum prefixes allowed 100
  Threshold for warning message 50%

```

If we temporarily reduce CSR8's maximum to 4 towards CSR1, the prefix limit will be exceeded. CSR8 makes it very clear with several BGP logs, first stating the prefix limit (this number 4 is actually encoded in the message sent to the offending peer, CSR1), then notifying the peer of its transgression. We can see that this applies to both IPv4 and IPv6 AFIs in this example.

```

! CSR1
router bgp 85
  address-family ipv4
    neighbor 10.1.8.1 maximum-prefix 4
  address-family ipv6
    neighbor 10.1.8.1 maximum-prefix 4

%BGP-3-MAXPFEXCEED: Number of prefixes received from 10.1.8.1 (afi 0): 17
exceeds limit 4
%BGP-3-NOTIFICATION: sent to neighbor 10.1.8.1 6/1 (Maximum Number of
Prefixes Reached) 7 bytes 00010100 000004
%BGP-5-ADJCHANGE: neighbor 10.1.8.1 Down Peer over prefix limit
%BGP_SESSION-5-ADJCHANGE: neighbor 10.1.8.1 IPv6 Unicast topology base
removed from session Peer over prefix limit
%BGP_SESSION-5-ADJCHANGE: neighbor 10.1.8.1 IPv4 Unicast topology base
removed from session Peer over prefix limit

```

BGP also makes it very clear in other show commands in case the log messages are not seen. As an added benefit, the command tells us how to remedy the situation. This could be automatic if we configured a restart timer, which is not the default.

```

R8#show bgp ipv4 unicast neighbors 10.1.8.1 | begin : IPv4 Unicast
For address family: IPv4 Unicast
[snip]
Peer had exceeded the max. no. of prefixes configured.
Maximum prefixes allowed 4
Threshold for warning message 75%
Reduce the no. of prefix and clear ip bgp 10.1.8.1 to restore peering

```

From CSR1's perspective, we see some logs that indicate a capability change. This is CSR8 signaling to CSR1 that it no longer supports IPv4 unicast, which means CSR1 will never advertise any IPv4 unicast prefixes, either. That is why the hard-clear is required to fix this issue, since the session itself has been poisoned as a protection mechanism. CSR8's BGP OPEN message no longer contains the IPv4 unicast capability towards CSR1 at all, which guarantees that IPv4 unicast prefixes cannot be received. CSR1 also shows this when we examine the negotiated capabilities with CSR8. CSR8 is no longer willing to run IPv4 unicast with CSR1 after having been told it was advertising too many prefixes.

```
! CSR1
%BGP_SESSION-5-ADJCHANGE: neighbor 10.1.8.8 IPv4 Unicast topology base
removed from session BGP Notification received
%BGP_SESSION-5-ADJCHANGE: neighbor 10.1.8.8 IPv4 Unicast topology base
removed from session Capability changed
```

```
R1#show bgp ipv4 unicast neighbors 10.1.8.8 | section capabilities
Neighbor capabilities:
  Route refresh: advertised and received(new)
  Four-octets ASN Capability: advertised and received
  Address family IPv4 Unicast: advertised
  Address family IPv6 Unicast: advertised and received
  Enhanced Refresh Capability: advertised and received
```

Before continuing, I fixed the issue by setting maximum back to 100 and clearing the session (not shown). A quick verification shows everything is working again as expected.

```
R1#show bgp ipv4 unicast neighbors 10.1.8.8 | section capabilities
Neighbor capabilities:
  Route refresh: advertised and received(new)
  Four-octets ASN Capability: advertised and received
  Address family IPv4 Unicast: advertised and received
  Address family IPv6 Unicast: advertised and received
  Enhanced Refresh Capability: advertised and received
```

```
R1#show bgp ipv4 unicast neighbors 10.1.8.8 advertised-routes | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
*>  1.10.0.0/16      0.0.0.0          0         32768 ?
*>  1.11.0.0/16      0.0.0.0          0         32768 ?
*>  1.12.0.0/16      0.0.0.0          0         32768 ?
[snip]
```

6. **Ingress BGP prefix filtering from peer ISPs:** This is used for removal of bogons, RFC 1918 addressing, etc. This was covered in detail when we examined ingress L3VPN customer BGP filtering. The only difference is that the bogon prefix-lists will contain RFC1918 addressing this time as these should not be leaked between SPs (exceptions may include architectures like Inter-AS MPLS Option C). The application is still the same; inbound route-map or route-policy facing the peer while wrapping a prefix-list or prefix-set to deny bogons. Because this is a general best practice, I prefer using peer-policy

templates or AF-groups to modularize the configuration. Although not mentioned in this document, I suggest only advertising locally-assigned addresses out, assuming the local AS is not a transit AS. The AS 85 outbound policies simply permit 85.0.0.0/8 for simplicity; using the regex “^\$” would also be effective as it would permit local routes only. The XE configuration can be trickier because when running IPv6 over an IPv4 BGP session, next-hops must be manually modified, which means a different outbound route-map per peer. The commands to enable peer-policy or AF-group inheritance are not shown. Use the normal BGP show commands to see if the proper routes are entering and exiting the network, as we saw earlier. The verification process is the same and is not repeated for brevity. The RPL/route-map names are self-explanatory so their contents are not displayed.

```
! Basic XR policy
router bgp 85
  af-group AF_IPV4_EBGP address-family ipv4 unicast
    route-policy RPL_EBGP_IN in
    route-policy RPL_85_ONLY out

  af-group AF_IPV6_EBGP address-family ipv6 unicast
    route-policy RPL_EBGP_IPV6_IN in
    route-policy RPL_EBGP_IPV6_OUT out

  neighbor 10.1.3.13
    address-family ipv4 unicast
      use af-group AF_IPV4_EBGP
    address-family ipv6 unicast
      use af-group AF_IPV6_EBGP

! Basic XE policy
template peer-policy IPV4_EBGP
  route-map RM_EBGP_IN in
  route-map RM_EBGP_OUT out

template peer-policy IPV6_EBGP
  route-map RM_EBGP_IPV6_IN in

router bgp 85
  address-family ipv4
    neighbor 10.1.8.1 inherit peer-policy IPV4_EBGP
    neighbor 10.8.9.9 inherit peer-policy IPV4_EBGP
  address-family ipv6
    neighbor 10.1.8.1 inherit peer-policy IPV6_EBGP
    neighbor 10.8.9.9 inherit peer-policy IPV6_EBGP
    neighbor 10.1.8.1 route-map RM_EBGP_OUT_R1 out ! Needed to adjust NHOP
    neighbor 10.8.9.9 route-map RM_EBGP_OUT_R9 out ! Needed to adjust NHOP
```

7. **BGP dampening:** This is used to penalize unstable routes which prevents non-stop flapping and BGP instability. Unlike throttling mechanisms, this feature will actually break reachability to prefixes that

are deemed sufficiently unstable. Several parameters can be configured to tune how aggressively BGP dampens routes, but if no parameters are specified, default values are used. The formula is as follows:

$$\text{max-penalty} = \text{reuse-limit} * 2^{(\text{max-suppress-time}/\text{half-life})}$$

For variety, XRv1 and CSR8 have BGP dampening enabled for IPv4 and IPv6 AFs with default parameters. I do not believe there is a way to see the default parameters in XR, but XE shows it. These parameters are the same for all AFs.

```
! XRv1
router bgp 85
  address-family ipv4 unicast
    bgp dampening
  address-family ipv6 unicast
    bgp dampening
```

```
! CSR8
router bgp 85
  address-family ipv4
    bgp dampening
  address-family ipv6
    bgp dampening
```

```
R8#show bgp ipv4 unicast dampening parameters
```

```
dampening 15 750 2000 60 (DEFAULT)
Half-life time       : 15 mins      Decay Time          : 2320 secs
Max suppress penalty: 12000         Max suppress time: 60 mins
Suppress penalty    : 2000         Reuse penalty       : 750
```

Two important values are not listed here nor are they configurable. When a route flaps, the router imposes a penalty value of 1,000 to that route. If the route only has an attribute change, the penalty is 500. If the route has enough of these “changes” to surpass the “suppress penalty”, the route is dampened. This value is a threshold that, once crossed, means BGP considers the route unusable and won’t consider it for best-path selection. For example, if a route flaps twice and has its MED changed once within a short time, the penalty will be around 2500, enough to trip the threshold.

The reason I say “around” 2500 is because of the half-life. This value determines how quickly the penalty fades away (like radioactive decay). A 15 minute half-life means that after 15 minutes, the penalty will become half of its current value. Therefore, a penalty of 2500 will take 15 minutes to reach 1250. The max-penalty and max-suppress-time are upper bounds on how “bad” a route can be. After 12,000 max-penalty points, the counter stops incrementing, but the route still has to stay dampened until the penalty goes back below the suppress reuse penalty threshold. The max-suppress-time is input to the formula that ultimately determines the max-penalty value. Being the numerator in the fraction, a larger max-suppress-time implies a potentially higher max-penalty. A shorter half-life means the route gets “forgiven” more rapidly, but be careful to ensure the math resolves to sensible values. Once the penalty

falls below the re-use penalty (750), the route can be re-used, like a falling-threshold. Once it falls below half of the re-use penalty (375), all penalty points are removed.

On CSR4, we will configure custom dampening parameters in a route-map. We could also specify specific prefixes, communities, AS-paths, etc. to apply dampening only to certain routes. In this case, we apply dampening only to routes originating inside AS 1.1 using an AS-path ACL. Let's say we want to use the default parameters except we want the half life to be very slow, making "bad" routes take longer to re-enter the BGP table as best-path candidates. In this case, we select a half-life that is very large (slow), so much so that the math is invalid. It does not make sense for the max-penalty to be less than the suppress-penalty; if it were, then routes would never be suppressed, since they could only be penalized up to 1880 points when 2000 is required for dampening.

```
! CSR4
ip as-path access-list 1 permit _65537$

route-map RM_DAMP_FROM_AS_1DOT1 permit 10
  match as-path 1
  set dampening 45 750 2000 60

router bgp 85
  address-family ipv4
    bgp dampening route-map RM_DAMP_FROM_AS_1DOT1
  address-family ipv6
    bgp dampening route-map RM_DAMP_FROM_AS_1DOT1
```

We evaluate the formula: $\text{max-penalty} = 750 * 2^{(60/45)} = 750 * 2^{(2.5)} = 750 * \sim 2.51 \sim 1880$

The router notifies us of our folly with a log message, and also removes our invalid BGP dampening configuration. These logs are important because once the configuration is removed, it may not be obvious why the router is rejecting the configuration.

```
! CSR4
%BGP-5-DAMPENING LOW MAX PENALTY: Maximum penalty (1880) is less than
suppress penalty (2000), Dampening is OFF
```

There is also a hard-limit on the max-penalty of 20,000. It is possible to exceed this value using a different set of invalid parameters. This would occur if we were trying to have a very fast turnover using a small half-life. If the half-life is only 5 minutes with all other parameters set to their defaults, we get another message telling us our math is still wrong but for the opposite reason.

```
! CSR4
route-map RM_DAMP_FROM_AS_1DOT1 permit 10
  set dampening 5 750 2000 60

router bgp 85
```

```

address-family ipv4
  bgp dampening route-map RM_DAMP_FROM_AS_1DOT1
address-family ipv6
  bgp dampening route-map RM_DAMP_FROM_AS_1DOT1

```

We evaluate the formula as follows: $\text{max-penalty} = 750 * 2^{(60/5)} = 750 * 2^{(12)} = 750 * 4096 = 3072000$

```
! CSR4
```

```
%BGP-5-DAMPENING_HIGH_MAX_PENALTY: Maximum penalty (3072000) is more than
allowed maximum (20000). Dampening is OFF
```

Clearly, BGP dampening is a tricky topic given the math involved. Let's say our design goal is to have a more reactive policy. We want a faster half life (more forgiving), but we also want routes suppressed more quickly. We should reduce the half-life, max-suppress-timer, and suppress-penalty, and potentially increase the re-use penalty. Now, the routes are penalized sooner at 1,500 penalty points, but are also allowed back into the RIB sooner when their penalty drops under 1,000. The half life will be very fast at 5 minutes, with no route being suppressed longer than 15 minutes. This gives us an 8,000 point max penalty, which is greater than the suppress penalty of 1500 and less than the max-penalty hard-limit of 20,000. Again, this upper-bound on the max-penalty does not appear configurable.

```
! CSR4
```

```
route-map RM_DAMP_FROM_AS_1DOT1 permit 10
  set dampening 5 1000 1500 15
```

```
router bgp 85
  address-family ipv4
    bgp dampening route-map RM_DAMP_FROM_AS_1DOT1
  address-family ipv6
    bgp dampening route-map RM_DAMP_FROM_AS_1DOT1

```

We evaluate the formula as follows: $\text{max-penalty} = 1000 * 2^{(15/5)} = 1000 * 2^{(3)} = 1000 * 8 = 8000$

We don't see a log message because we did not cause an error. The parameters for dampening also include the route-map entry from which the settings were derived. We can see the max penalty of 8,000 also to check our math. The benefit of not using prefix-lists in these route-maps means they can be recycled between AFs. Communities and AS-paths are typically not AF-dependent and are good options for filtering due to their flexibility and reusability.

```

R4#show bgp ipv4 unicast dampening parameters
dampening 5 1000 1500 15 (route-map RM_DAMP_FROM_AS_1DOT1 10)
Half-life time       : 5 mins      Decay Time          : 720 secs
Max suppress penalty: 8000         Max suppress time: 15 mins
Suppress penalty    : 1500        Reuse penalty       : 1000

```

Next, we configure XRv2 with the same dampening parameters to ensure XR rejects the invalid options and accepts the valid options. XR is smart in that it will not let you commit an invalid configuration. XR also doesn't tell you the result of the math, unlike XE, so you must know the formula. The verbiage hints that our max-suppress is too low, which is true. These hints can be helpful for fixing the errors.

```
! XRv2
route-policy RPL_BGP_DAMPENING_FROM_AS_9DOT13
  if as-path originates-from '589837' then
    set dampening halflife 45 suppress 2000 reuse 750 max-suppress 60
  endif
end-policy

router bgp 85
  address-family ipv4 unicast
    bgp dampening route-policy RPL_BGP_DAMPENING_FROM_AS_9DOT13

% Failed to commit [snip]

!!% Verify failed for policy: Policy [RPL_BGP_DAMPENING_FROM_AS_9DOT13] uses
an invalid argument in the statement 'set dampening halflife 45 suppress 2000
reuse 750 max-suppress 60' at the bgp dampening-dflt attach point. Policy
attempted to set invalid values for dampening parameters. Inconsistent
dampening parameters. Decrease half-life, or increase suppress, max-
suppress, or reuse-penalty.
```

Let's use another invalid example as a quick test by causing the max-penalty to be greater than 20,000. The verbiage is a little different because its telling us the max-suppress is too high.

```
! XRv2
route-policy RPL_BGP_DAMPENING_FROM_AS_9DOT13
  if as-path originates-from '589837' then
    set dampening halflife 5 suppress 2000 reuse 750 max-suppress 60
  endif
end-policy

router bgp 85
  address-family ipv4 unicast
    bgp dampening route-policy RPL_BGP_DAMPENING_FROM_AS_9DOT13

% Failed to commit [snip]

!!% Verify failed for policy: Policy [RPL_BGP_DAMPENING_FROM_AS_9DOT13] uses
an invalid argument in the statement 'set dampening halflife 5 suppress 2000
reuse 750 max-suppress 60' at the bgp dampening-dflt attach point. Policy
attempted to set invalid values for dampening parameters. Inconsistent
dampening parameters. Decrease max-suppress or reuse-penalty, or increase
half-life.
```

We add a corrected example at the end and ensure XR accepts the change.

```
! XRv2
route-policy RPL_BGP_DAMPENING_FROM_AS_9DOT13
  if as-path originates-from '589837' then
    set dampening halflife 5 suppress 1500 reuse 1000 max-suppress 15
  endif
end-policy

router bgp 85
  address-family ipv4 unicast
    bgp dampening route-policy RPL_BGP_DAMPENING_FROM_AS_9DOT13
```

Next, we should test the feature to make sure it works. We will use the route 9.14.0.0/16 for testing since both XRv1 and XRv2 prefer the eBGP paths via AS 9.13 (always verify this first). The two routers have varying dampening parameters, though, so flapping this route will have different effects. Recall that XRv1 uses the default parameters while XRv2 has a custom policy. We will examine both together.

```
RP/0/0/CPU0:XRv1#show bgp ipv4 unicast 9.14.0.0/16 brief | begin Network
  Network      Next Hop      Metric LocPrf Weight Path
* i9.14.0.0/16  8.8.8.8          0      100      0 589837 ?
*>              10.1.3.13          0      100      0 589837 ?
```

```
RP/0/0/CPU0:XRv2#show bgp ipv4 unicast 9.14.0.0/16 brief | begin Network
  Network      Next Hop      Metric LocPrf Weight Path
* i9.14.0.0/16  8.8.8.8          0      100      0 589837 ?
*>              10.2.9.9          0      100      0 589837 ?
* i            11.11.11.11      100      0 589837 ?
```

I delete the IP prefix from CSR9, which causes BGP to withdraw it (not shown). Both XRv1 and XRv2 mark it with “history” to note that this prefix is currently not reachable, but we are remembering it so we can keep a tally of its penalty points. BGP cannot simply purge this prefix from memory or else it would have no mechanism to track the incurring penalties.

```
RP/0/0/CPU0:XRv1#show bgp ipv4 unicast 9.14.0.0/16 | begin Paths
Paths: (1 available, no best path)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  589837, (history entry)
  10.1.3.13 from 10.1.3.13 (13.13.13.13)
    Origin incomplete, localpref 100, external
    Received Path ID 0, Local Path ID 0, version 0
    Origin-AS validity: not-found
```

```
RP/0/0/CPU0:XRv2#show bgp ipv4 unicast 9.14.0.0/16 | begin Paths
Paths: (1 available, no best path)
```



```

Not advertised to any peer
Path #1: Received by speaker 0
Not advertised to any peer
589837, (history entry)
10.2.9.9 from 10.2.9.9 (10.8.9.9)
Origin incomplete, metric 0, localpref 100, external
Received Path ID 0, Local Path ID 0, version 0
Origin-AS validity: not-found

```

Both routers record the flap in their flap-statistics for quick viewing. Both routers have seen the route flap one time. They also show the duration of time required for the route to be useable again.

```

RP/0/0/CPU0:XRv1#show bgp ipv4 unicast flap-statistics 9.14.0.0/16 | begin
Network

```

Network	From	Flaps	Duration	Reuse	Path
*> 9.14.0.0/16	10.1.3.13	1	00:01:42		589837 ?

```

RP/0/0/CPU0:XRv2#show bgp ipv4 unicast flap-statistics 9.14.0.0/16 | begin
Network

```

Network	From	Flaps	Duration	Reuse	Path
*> 9.14.0.0/16	10.2.9.9	1	00:01:50		589837 ?

When the route comes back, there is no issue because the suppress penalties of 2000 (XRv1) and 1500 (XRv2) have not been breached (not shown). The route has penalty points but the suppress penalty has not been breached, so both XRv1 and XRv2 can consider the route for best-path selection. If we flap the route again (also not shown), XRv1 still accepts the route, because a few seconds of half life means that 1000 plus a little less than 1000 is a little less than 2000. XRv1 is not currently dampening this prefix despite flapping twice. XRv1 needs to wait about 4 minutes before reusing this prefix.

```

RP/0/0/CPU0:XRv1#show bgp ipv4 unicast flap-statistics 9.14.0.0/16 | begin
Network

```

Network	From	Flaps	Duration	Reuse	Path
*> 9.14.0.0/16	10.1.3.13	2	00:04:01		589837 ?

```

RP/0/0/CPU0:XRv1#show bgp ipv4 unicast dampened-paths

```

[no output]

XRv2, however, dampens the route as it exceeded the 1500 penalty ($\sim 2000 > 1500$). The BGP entry now contains additional dampening information to include timers as to when the route will be usable again (that is to say, its penalty drops beneath 1000). XRv2's dampening statistics also show a "d" for dampening path, while earlier we saw a ">" symbol indicating XRv2 selected it as a best-path. Notice that XRv2 selects the iBGP path via XRv1 as its best path. This is the expected behavior since dampened paths are not candidates in the best-path selection algorithm.

```

RP/0/0/CPU0:XRv2#show bgp ipv4 unicast 9.14.0.0/16 | begin dampen

```

```

589837, (suppressed due to dampening)
10.2.9.9 from 10.2.9.9 (10.8.9.9)

```

```

Origin incomplete, metric 0, localpref 100, valid, external
Received Path ID 0, Local Path ID 0, version 0
Origin-AS validity: not-found
Dampinfo: penalty 1445, flapped 2 times in 00:04:15, reuse in 00:02:40
          half life 00:05:00, suppress value 1500, reuse value 1000
          Maximum suppress time 00:15:00

```

```

Path #3: Received by speaker 0
Not advertised to any peer
589837

```

```

11.11.11.11 (metric 20) from 14.14.14.14 (11.11.11.11)
Origin incomplete, localpref 100, valid, internal, best, group-best
Received Path ID 0, Local Path ID 1, version 237
Originator: 11.11.11.11, Cluster list: 14.14.14.14

```

```

RP/0/0/CPU0:XRv2#show bgp ipv4 unicast dampened-paths | begin Network
Network          From          Reuse      Path
*d 9.14.0.0/16    10.2.9.9      00:02:20  589837 ?

```

For variety, we will quickly repeat the same test on CSR4 and CSR8 for IPv6 AF using the prefix 2001:1C00:B::/48 on CSR1. Recall that CSR4 has the customized parameter set while CSR8 uses the defaults. CSR8 should behave like XRv1 and CSR4 should behave like XRv2 from the previous example. Before starting, we quickly verify the best paths to the route and ensure it is via CSR1 on both routers.

```

R8#show bgp ipv6 unicast 2001:1C00:B::/48
[snip]
65537
FD00:10:1:8::1 from 10.1.8.1 (10.1.8.1)
Origin incomplete, metric 0, localpref 100, valid, external, best
mpls labels in/out 8023/nolabel
rx pathid: 0, tx pathid: 0x0
[snip]

```

```

R4#show bgp ipv6 unicast 2001:1C00:B::/48
[snip]
65537
FD00:10:1:4::1 from 10.1.4.1 (10.1.8.1)
Origin incomplete, metric 0, localpref 100, valid, external, best
mpls labels in/out 4012/nolabel
rx pathid: 0, tx pathid: 0x0

```

We delete the prefix from CSR1 (not shown) and observe the history entries on the CSRs. Unlike XR, we can see the “dampinfo” here in the history entry, which is useful.

```

R4#show bgp ipv6 unicast 2001:1C00:B::/48
BGP routing table entry for 2001:1C00:B::/48, version 464
Paths: (1 available, no best path)
Not advertised to any peer
Refresh Epoch 7

```

65537 (history entry)

FD00:10:1:4::1 from 10.1.4.1 (10.1.8.1)
Origin incomplete, metric 0, localpref 100, external
Dampinfo: penalty 1000, flapped 1 times in 00:00:02
mpls labels in/out 4012/nolabel
rx pathid: 0, tx pathid: 0

R8#show bgp ipv6 unicast 2001:1C00:B::/48
BGP routing table entry for 2001:1C00:B::/48, version 351
Paths: (2 available, no best path)
Flag: 0x820

Not advertised to any peer

Refresh Epoch 2

65537 (history entry)

FD00:10:1:8::1 from 10.1.8.1 (10.1.8.1)
Origin incomplete, metric 0, localpref 100, external
Dampinfo: penalty 996, flapped 1 times in 00:00:07
mpls labels in/out 8023/nolabel
rx pathid: 0, tx pathid: 0

Refresh Epoch 1

589837 65537 (history entry)

FD00:10:8:9::9 from 10.8.9.9 (10.8.9.9)
Origin incomplete, localpref 100, external
Dampinfo: penalty 996, flapped 1 times in 00:00:07
mpls labels in/out 8023/nolabel
rx pathid: 0, tx pathid: 0

Bringing the route back on CSR1 (also not shown) causes both routers to re-install it as a best path. Both routers record the flap statistics within the prefix, so that even as a best-path, the administrator is aware of the potentially instability.

R4#show bgp ipv6 unicast 2001:1C00:B::/48
BGP routing table entry for 2001:1C00:B::/48, version 465
Paths: (1 available, best #1, table default)

Advertised to update-groups:

9

Refresh Epoch 7

65537

FD00:10:1:4::1 from 10.1.4.1 (10.1.8.1)
Origin incomplete, metric 0, localpref 100, valid, external, best
Dampinfo: penalty 870, flapped 1 times in 00:01:11
mpls labels in/out 4012/nolabel
rx pathid: 0, tx pathid: 0x0

R8#show bgp ipv6 unicast 2001:1C00:B::/48
BGP routing table entry for 2001:1C00:B::/48, version 352
Paths: (3 available, best #2, table default)
Advertised to update-groups:

```

15
Refresh Epoch 1
65537
::FFFF:4.4.4.4 (metric 20) from 14.14.14.14 (14.14.14.14)
  Origin incomplete, metric 0, localpref 100, valid, internal
  Originator: 4.4.4.4, Cluster list: 14.14.14.14
  mpls labels in/out 8023/4012
  rx pathid: 0, tx pathid: 0
Refresh Epoch 2
65537
FD00:10:1:8::1 from 10.1.8.1 (10.1.8.1)
  Origin incomplete, metric 0, localpref 100, valid, external, best
  Dampinfo: penalty 942, flapped 1 times in 00:01:19
  mpls labels in/out 8023/nolabel
  rx pathid: 0, tx pathid: 0x0
Refresh Epoch 1
589837 65537
FD00:10:8:9::9 from 10.8.9.9 (10.8.9.9)
  Origin incomplete, localpref 100, valid, external
  Dampinfo: penalty 942, flapped 1 times in 00:01:19
  mpls labels in/out 8023/nolabel
  rx pathid: 0, tx pathid: 0

```

Flapping the route again on CSR1 (not shown) causes CSR4 to suppress the route while CSR8 is still accepting it. In this situation, CSR4 picks an iBGP best path via CSR8 to reach this prefix. **Dampening never applies to internal prefixes, but only external and confed-external.** Because CSR8 has not dampened the iBGP prefix, this is a valid alternative path. This is the same thing we observed on XRv2 when it used the iBGP path via XRv1 as a backup. In about 3 minutes, the eBGP path will be usable again. This is based on the half-time and how long it takes for the penalty to fall below the reuse threshold.

```

R4#show bgp ipv6 unicast 2001:1C00:B::/48
BGP routing table entry for 2001:1C00:B::/48, version 467
Paths: (2 available, best #1, table default)
  Not advertised to any peer
Refresh Epoch 1
65537
::FFFF:8.8.8.8 (metric 20) from 14.14.14.14 (14.14.14.14)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  Originator: 8.8.8.8, Cluster list: 14.14.14.14
  mpls labels in/out 4012/8023
  rx pathid: 0, tx pathid: 0x0
Refresh Epoch 7
65537, (suppressed due to dampening)
FD00:10:1:4::1 from 10.1.4.1 (10.1.8.1)
  Origin incomplete, metric 0, localpref 100, valid, external
  Dampinfo: penalty 1544, flapped 2 times in 00:03:05, reuse in 00:03:10
  mpls labels in/out 4012/nolabel
  rx pathid: 0, tx pathid: 0

```

Other show commands and quickly show us flap statistics and dampened paths. In CSR4's case, we have statistics on one route, and it also happens to be dampened. Notice the "d" for dampened.

```
R4#show bgp ipv6 unicast dampening flap-statistics | begin Network
      Network          From          Flaps Duration Reuse      Path
*d  2001:1C00:B::/48  10.1.4.1            2      00:03:29 00:02:50 65537
```

```
R4# show bgp ipv6 unicast dampening dampened-paths | begin Network
      Network          From          Reuse      Path
*d  2001:1C00:B::/48  10.1.4.1            00:02:50 65537 ?
```

On CSR8, we have collected history statistics from multiple eBGP peers on the prefix but it is not dampened, which is different than CSR1's current state.

```
R8#show bgp ipv6 unicast dampening flap-statistics | begin Network
      Network          From          Flaps Duration Reuse      Path
*>  2001:1C00:B::/48  10.1.8.1            2      00:03:17      65537
*      10.8.9.9            2      00:03:17 589837 65537
```

```
R8#show bgp ipv6 unicast dampening dampened-paths
[no output]
```

8. **BGP neighbor logging:** This allows the SP to track unstable BGP peers. Neighbor logging is very basic feature and is enabled by default on XE and XR BGP implementations. We have seen several examples of BGP logging thus far and it should be obvious as to why this is desirable. In a production environment, sending them to a syslog server for analysis is most effective. Below are snippets to disable this feature; I do not apply this configuration anywhere since BGP neighbor logging is valuable.

```
! XE
router bgp 85
  no bgp log-neighbor-changes

! XR
router bgp 85
  bgp log neighbor changes disable
```

9. **TTL-security:** This feature is applied on external and confed-external peers to prevent remote attacks against the BGP peering process. This is a simple but highly intelligent mechanism. BGP differentiates between "connected" and "process-learned" (OSPF, EIGRP, RIP, static, etc) routes when forming eBGP neighbors. By default, only connected routes can be used for eBGP peers. Some documentation suggests that if you configure eBGP peerings between loopbacks you need to increase the session TTL using "ebgp-multihop". This is not true; TTL=1 is sufficient to reach the remote neighbor (assuming the two routers are connected), but we need to tell BGP to specifically use process-learned

routes for it. This is what “disable-connected-check” does in XE and “ignore-connected-check” does in XR. Here is a quick summary of TTL-security behavior when configured in concert with other features.

eBGP peers with no fancy options: TTL=1 and will only look at connected routes

eBGP peers + disable-connected-check: TTL=1 and will also look at connected or process

eBGP peers + ebgp-multihop x: TTL=x and also look at connected or process

eBGP peers + TTL-security hops 1: TTL=255 and only looking at connected routes

eBGP peers + TTL-security hops x: TTL=255 and also looking at process routes

eBGP peers + TTL-security hops 1 + disable-connected-check: TTL=255 and also looking at connected or process routes. This is the most secure way to configure eBGP across two connected router loopbacks.

The expected received TTL must be at least 255-x with TTL security, since x is the max hop count. If “ttl-security hops 1” is configured on one side, at a minimum, the remote peer must send eBGP packets with TTL=255. This can be accomplished by also enabling TTL-security or using eBGP multihop; the remote peer must use one of these two mechanisms or else the session will not form. In XR, the TTL-security hop count is always 1 and cannot be changed. I have enabled TTL security on all eBGP peers. A sample of router configurations is shown below.

```
! XRv1
router bgp 85
  neighbor 10.1.3.13
    ttl-security

! CSR8
router bgp 85
  neighbor 10.8.9.9 ttl-security hops 1
```

We can verify the feature is functional by checking the neighbor status. XR doesn’t give as much information but it’s still a valid verification. Also notice that XE says the eBGP peer is configured for connected checks, which implies “disable-connected-check” was not configured. Since there is no need for process-learned routes to form this eBGP session, we omit this feature; this adds security.

```
RP/0/0/CPU0:XRv1#show bgp ipv4 unicast neighbors 10.1.3.13 | include TTL
TTL security is configured
```

```
R8#show bgp ipv4 unicast neighbors 10.8.9.9 | include hop|TTL
  External BGP neighbor may be up to 1 hop away.
  External BGP neighbor configured for connected checks (single-hop no-
  disable-connected-check)
Connection is ECN Disabled, Minimum incoming TTL 254, Outgoing TTL 255
```

10. BGP policy accounting (PA): This feature is used to account for traffic using communities or other matching mechanisms. The configuration is very similar to QPPB, which I use on XRv1 and CSR4 later to show other security features. We will configure BGP PA on XRv2 and CSR8 for this

demonstration. CSR10 will set the community value on specific prefixes when it injects them into BGP; I use a route-map and network statement for this and the configuration is shown later. On ingress, the ASBRs will check their BGP table-policies for specific communities, and then set the traffic-index marker, of which there are 64 total. CEF will account for packets/bytes based on the traffic-index and we can get a quick summary of those statistics using BGP PA. My policy is as follows:

- a. Routes with community 85:1024 are placed in traffic-index 1
- b. Routes with community 85:2048 are placed in traffic-index 2
- c. Routes with community 85:4096 are placed in traffic-index 4

First, we will configure CSR10 to originate IPv4 and IPv6 routes into BGP with some communities. The same prefixes will be used for QPPB later, and we are using 85.19.8.0/21 and 2001:85:19::/63 with communities set to 85:1024 and 85:2048, respectively.

```
! CSR10
route-map RM_SET_COMM permit 10
  set community 85:1024

route-map RM_SET_COMM_IPV6 permit 10
  set community 85:2048

router bgp 85
  address-family ipv4
    network 85.19.8.0 mask 255.255.248.0 route-map RM_SET_COMM
  address-family ipv6
    network 2001:85:19::/63 route-map RM_SET_COMM_IPV6
```

A quick check of the BGP tables on the ASBRs to ensure the communities were properly communicated is warranted. Rather than checking the prefix, I will query based on communities. This ensures that communities are being exchanged between iBGP peers (default behavior for XR but not XE).

```
R8#show bgp ipv4 unicast community 85:1024 | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
* i  85.19.8.0/21  10.10.10.10      0      100      0 i
*>i
      10.10.10.10      0      100      0 i

R8#show bgp ipv6 unicast community 85:2048 | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
* i  2001:85:19::/63  ::FFFF:10.10.10.10
                                     0      100      0 i
*>i
      ::FFFF:10.10.10.10
                                     0      100      0 i

RP/0/0/CPU0:XRv2#show bgp ipv4 unicast community 85:1024 | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
*>i 85.19.8.0/21  10.10.10.10      0      100      0 i
* i  10.10.10.10      0      100      0 i
```

```

Processed 1 prefixes, 2 paths
RP/0/0/CPU0:XRv2#show bgp ipv6 unicast community 85:2048 | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
*>i2001:85:19::/63  10.10.10.10      0      100      0 i
* i                10.10.10.10      0      100      0 i

```

Next, we configure the policies used by BGP. These are XE route-maps or XR RPLs that are applied under the BGP table-map, just like QPPB. We will start with CSR8 first.

```

! CSR8
ip community-list standard COML_TRAF_IDX1 permit 85:1024
ip community-list standard COML_TRAF_IDX2 permit 85:2048
ip community-list standard COML_TRAF_IDX4 permit 85:4096

route-map RM_BGP_PA per 10
  match community COML_TRAF_IDX1
  set traffic-index 1
route-map RM_BGP_PA per 20
  match community COML_TRAF_IDX2
  set traffic-index 2
route-map RM_BGP_PA per 30
  match community COML_TRAF_IDX4
  set traffic-index 4

router bgp 85
  address-family ipv4
    table-map RM_BGP_PA
  address-family ipv6
    table-map RM_BGP_PA

interface GigabitEthernet2.518
  bgp-policy accounting input

interface GigabitEthernet2.589
  bgp-policy accounting input

```

We should verify that the interface-level commands were accepted. The output is strange in that it says BGP policy mapping is disabled, then in the next line, says it is an input feature. I will assume that the feature was enabled correctly and continue despite the odd output.

```

R8#show ip interface gig2.518 | include BGP
  BGP Policy Mapping is disabled
  Input features: BGP Policy Map, Access List, uRPF, MCI Check

R8#show ip interface gig2.589 | include BGP
  BGP Policy Mapping is disabled
  Input features: BGP Policy Map, uRPF, MCI Check

```


The tricky part is the order of operations: if the routes were installed in the FIB before the PA policy is applied, you must “clear ip route” so they are re-added. Below is a before-and-after example of this behavior. For simplicity, I refresh all IPv4 routes rather than target the specific one; both methods work.

```
R8#show ip cef 85.19.8.0 detail
85.19.8.0/21, epoch 2, flags [rib only nolabel, rib defined all labels]
  recursive via 10.10.10.10
    nexthop 85.6.8.6 GigabitEthernet2.568 label 6006
```

```
R8#clear ip route *
```

```
R8#show ip cef 85.19.8.0 detail
85.19.8.0/21, epoch 2, flags [rib only nolabel, rib defined all labels]
  BGP: traffic_index 1
  recursive via 10.10.10.10
    nexthop 85.6.8.6 GigabitEthernet2.568 label 6006
```

Like QPPB, XE does not appear to support BGP PA for IPv6. Notice there is no traffic-index applied for the IPv6 traffic, even after a RIB/FIB refresh. We will test this in the data-plane later just to make sure it isn't a cosmetic issue with the show command.

```
R8#show ipv6 cef 2001:85:19::/63 detail
2001:85:19::/63, epoch 0, flags [rib defined all labels]
  recursive via 10.10.10.10 label 10007
    nexthop 85.6.8.6 GigabitEthernet2.568 label 6006
```

```
R8#clear ipv6 route *
```

```
R8#show ipv6 cef 2001:85:19::/63 detail
2001:85:19::/63, epoch 0, flags [rib defined all labels]
  recursive via 10.10.10.10 label 10007
    nexthop 85.6.8.6 GigabitEthernet2.568 label 6006
```

We can check the counters by looking at CEF policy statistics. Right now we see no statistics on either external interface.

```
R8#show cef interface gig2.518 policy-statistics
GigabitEthernet2.518 is up (if_number 14)
  Corresponding hwidb fast_if_number 13
  Corresponding hwidb firstsw->if_number 8
  BGP based Policy accounting on input is enabled
```

Index	Packets	Bytes
1	0	0
2	0	0
3	0	0
4	0	0

```

R8#show cef interface gig2.589 policy-statistics
GigabitEthernet2.589 is up (if_number 15)
  Corresponding hwidb fast_if_number 13
  Corresponding hwidb firstsw->if_number 8
  BGP based Policy accounting on input is enabled
Index      Packets      Bytes
  1         0         0
  2         0         0
  3         0         0
  4         0         0

```

We will test from CSR9 to start, but we must ensure CSR9 is routing through CSR8 to reach 85.19.8.0.

```

R9#show ip route 85.19.8.0
Routing entry for 85.19.8.0/21
  Known via "bgp 589837", distance 20, metric 0
  Tag 85, type external
  Last update from 10.8.9.8 13:54:00 ago
  Routing Descriptor Blocks:
  * 10.8.9.8, from 10.8.9.8, 13:54:00 ago
    Route metric is 0, traffic share count is 1
    AS Hops 1
    Route tag 85
    MPLS label: none

```

```

R9#ping 85.19.8.1 source 9.15.9.9 repeat 50
[snip]
Success rate is 100 percent (50/50), round-trip min/avg/max = 1/18/27 ms

```

CSR8's CEF process has accounted for 50 packets and 5000 total bytes (the pings were 100 bytes each) within traffic-index 1 as expected. Note that CEF does not account for layer 2 encapsulation with BGP PA. Because this is per-interface, the link between CSR8 and CSR1 has not logged any packets yet.

```

R8#show cef interface gig2.589 policy-statistics
GigabitEthernet2.589 is up (if_number 15)
  Corresponding hwidb fast_if_number 13
  Corresponding hwidb firstsw->if_number 8
  BGP based Policy accounting on input is enabled
Index      Packets      Bytes
  1        50       5000
  2         0         0
  3         0         0
  4         0         0

```

```

R8#show cef interface gig2.518 policy-statistics
GigabitEthernet2.518 is up (if_number 14)
  Corresponding hwidb fast_if_number 13

```

```

Corresponding hwidb firstsw->if_number 8
BGP based Policy accounting on input is enabled
Index          Packets          Bytes
1              0              0
2              0              0
3              0              0
4              0              0

```

A quick test from CSR1 (ensuring it routes through CSR8) shows that the traffic is also classified but kept separate. This time we send twice as many packets of double the size, so the total byte count should be 4 times what it was in the previous test.

```

R1#show ip route 85.19.8.0
Routing entry for 85.19.8.0/21
  Known via "bgp 1.1", distance 20, metric 0
  Tag 85, type external
  Last update from 10.1.8.8 00:00:03 ago
  Routing Descriptor Blocks:
  * 10.1.8.8, from 10.1.8.8, 00:00:03 ago
    Route metric is 0, traffic share count is 1
    AS Hops 1
    Route tag 85
    MPLS label: none

```

```

R1#ping 85.19.8.1 source 1.14.1.1 repeat 100 size 200
[snip]
Success rate is 100 percent (100/100), round-trip min/avg/max = 1/38/157 ms

```

We verify the statistics again. Nothing new came from CSR9, but the link to CSR1 now accounts for those new packets.

```

R8#show cef interface gig2.589 policy-statistics
GigabitEthernet2.589 is up (if_number 15)
  Corresponding hwidb fast_if_number 13
  Corresponding hwidb firstsw->if_number 8
  BGP based Policy accounting on input is enabled
Index          Packets          Bytes
1              50             5000
2              0              0
3              0              0
4              0              0

```

```

R8#show cef interface gig2.518 policy-statistics
GigabitEthernet2.518 is up (if_number 14)
  Corresponding hwidb fast_if_number 13
  Corresponding hwidb firstsw->if_number 8
  BGP based Policy accounting on input is enabled
Index          Packets          Bytes

```

1	100	20000
2	0	0
3	0	0
4	0	0

We also want to test IPv6 quickly to see if it's actually supported. A quick test from CSR9 shows that it is not, so I won't belabor it. CSR9 is routing through CSR8 as expected, but when traffic is sent, nothing is captured by BGP PA.

```
R9#show ipv6 route 2001:85:19::1
Routing entry for 2001:85:19::/63
  Known via "bgp 589837", distance 20, metric 0, type external
  Route count is 1/1, share count 0
  Routing paths:
    FD00:10:8:9::8
      MPLS label: nolabel
      Last updated 14:01:59 ago
```

```
R9#ping 2001:85:19::1 source 2001:9000:A::9 repeat 20 size 300
[snip]
Success rate is 100 percent (20/20), round-trip min/avg/max = 15/30/93 ms
```

```
R8#show cef interface gig2.589 policy-statistics
GigabitEthernet2.589 is up (if_number 15)
  Corresponding hwidb fast_if_number 13
  Corresponding hwidb firstsw->if_number 8
  BGP based Policy accounting on input is enabled
```

Index	Packets	Bytes
1	50	5000
2	0	0
3	0	0
4	0	0

Next, we examine the BGP PA feature on XRv2. The configuration is similar to XE except we use RPL instead with inline community sets for brevity/variety. We also need to be more specific at the interface level to determine whether we are doing source or destination accounting. If you don't specify in XE, it assumes destination, while source is another option. For XR, you must specify source or destination, and you can select both concurrently. XRv supports IPv6 and explicitly uses separate PA IPv4/v6 commands.

```
! XRv2
route-policy RPL_BGP_PA
  if community matches-every (85:1024) then
    set traffic-index 1
  elseif community matches-every (85:2048) then
    set traffic-index 2
  elseif community matches-every (85:4096) then
    set traffic-index 4
```

```

    else
        pass
    endif
end-policy

router bgp 85
  address-family ipv4 unicast
    table-policy RPL_BGP_PA
  address-family ipv6 unicast
    table-policy RPL_BGP_PA

interface GigabitEthernet0/0/0/0.529
  ipv4 bgp policy accounting input destination-accounting
  ipv6 bgp policy accounting input destination-accounting

```

We will verify that it has been enabled for both IPv4 and IPv6 on XRv2's external interface. We also verify the current BGP policy statistics for this interface, and we expect no matches. XR simply doesn't show anything if nothing appears, versus XE that shows a list of zeroes per bucket. Our final initial verification is ensuring the traffic-indices were applied properly. XR shows the traffic-index mappings in the RIB where XE shows this in the FIB. This validates our BGP PA RPL configuration; if we did not see these traffic-index markings in the RIB, we should clear the routing table so everything gets refreshed, depending on the order of operations.

```
RP/0/0/CPU0:XRv2#show ipv4 interface gig0/0/0/0.529 | utility egrep
'BGP|Desti'
```

```
IP Input BGP policy accounting is configured
Input-Source is Disabled, Input-Destination is Enabled
```

```
RP/0/0/CPU0:XRv2#show ipv6 interface gig0/0/0/0.529 | utility egrep
'BGP|Desti'
```

```
IP Input BGP policy accounting is configured
Input-Source is Disabled, Input-Destination is Enabled
```

```
RP/0/0/CPU0:XRv2#show cef ipv6 interface gig0/0/0/0.529 bgp-policy-statistics
GigabitEthernet0/0/0/0.529 is UP
```

```
Input BGP policy accounting on dst IP address enabled
  buckets      packets      bytes
```

```
RP/0/0/CPU0:XRv2#show cef interface gig0/0/0/0.529 bgp-policy-statistics
GigabitEthernet0/0/0/0.529 is UP
```

```
Input BGP policy accounting on dst IP address enabled
  buckets      packets      bytes
```

```
RP/0/0/CPU0:XRv2#show route ipv6 2001:85:19::0
```

```
Routing entry for 2001:85:19::/63
```

```
Known via "bgp 85", distance 200, metric 0, type internal, Traffic Index 2
```

```
Routing Descriptor Blocks
```

```
::ffff:10.10.10.10, from ::ffff:2.2.2.2
```

[snip]

```
RP/0/0/CPU0:XRv2#show route ipv4 85.19.8.0
Routing entry for 85.19.8.0/21
  Known via "bgp 85", distance 200, metric 0, type internal, Traffic Index 1
  Routing Descriptor Blocks
    10.10.10.10, from 2.2.2.2
    [snip]
```

Next, we ensure CSR9 is routing through XRv2 to reach 2001:85:19::0/63. We may need to clear BGP with CSR8 so that the oldest route is learned from XRv2. CSR9 sends traffic to this destination successfully.

```
R9#show ipv6 route 2001:85:19::0
Routing entry for 2001:85:19::/63
  Known via "bgp 589837", distance 20, metric 0, type external
  Route count is 1/1, share count 0
  Routing paths:
    FE80::12, GigabitEthernet2.529
      MPLS label: nolabel
      Last updated 00:00:08 ago
```

```
R9#ping 2001:85:19::1 source 2001:9000:C::9 repeat 20 size 400
[snip]
Success rate is 100 percent (20/20), round-trip min/avg/max = 5/75/252 ms
```

When we check the CEF counters on XRv2, we see the packets properly measured. In XE, we saw nothing, likely because BGP policy features like PA and QPPB are not supported for IPv6 at all. The IPv6 traffic was supposed to end up in bucket 2, and it did.

```
RP/0/0/CPU0:XRv2#show cef ipv6 interface gig0/0/0/0.529 bgp-policy-statistics
GigabitEthernet0/0/0/0.529 is UP
Input BGP policy accounting on dst IP address enabled
  buckets      packets      bytes
  2           20           8000
```

We will quickly test IPv4 also, which also appears to work. Once we validate CSR9 is routing through XRv2, we send some traffic from CSR9 and measure it on XRv2. The traffic is in bucket 1 as expected.

```
R9#show ip route 85.19.8.0
Routing entry for 85.19.8.0/21
  Known via "bgp 589837", distance 20, metric 0
  Tag 85, type external
  Last update from 10.2.9.12 00:08:37 ago
  Routing Descriptor Blocks:
    * 10.2.9.12, from 10.2.9.12, 00:08:37 ago
      Route metric is 0, traffic share count is 1
```

```
AS Hops 1
Route tag 85
MPLS label: none
```

```
R9#ping 85.19.8.1 source 9.14.9.9 repeat 15 size 150
[snip]
Success rate is 100 percent (15/15), round-trip min/avg/max = 2/13/146 ms
```

```
RP/0/0/CPU0:XRv2#show cef interface gig0/0/0/0.529 bgp-policy-statistics
GigabitEthernet0/0/0/0.529 is UP
Input BGP policy accounting on dst IP address enabled
  buckets      packets      bytes
  1           15          2250
```

11. **LDP and IGP security:** LDP and IGP peers should be authenticated for the same reasons described in the BGP authentication section. Interestingly, XR does not support key-chains for LDP authentication but does for BGP. XE is the opposite, so LDP peers between CSRs use key chains where exceptions are made (single passwords) for XR peers. CSR2 is a good example. The key chain is not shown for brevity. LDP security is discussed in detail in the LDP section.

```
! CSR2
mpls ldp password required
mpls ldp password fallback KC_LDP_AUTH
mpls ldp neighbor 14.14.14.14 password XR_LDP_AUTH
```

A quick way to check MD5 authentication on LDP peers is to look at the detailed LDP neighbor output, and specifically at the TCP connection information. We can also use the manual TCP method shown earlier in the BGP section. I show it again for reference.

```
R2#show mpls ldp neighbor detail | include TCP
TCP connection: 14.14.14.14.37949 - 2.2.2.2.646; MD5 on
TCP connection: 6.6.6.6.47235 - 2.2.2.2.646; MD5 on
TCP connection: 7.7.7.7.35620 - 2.2.2.2.646; MD5 on
TCP connection: 10.10.10.10.21470 - 2.2.2.2.646; MD5 on
```

```
R2#show tcp brief | include 6\.6
TCB          Local Address      Foreign Address      (state)
7F5323C2CE68 2.2.2.2.646             6.6.6.6.47235       ESTAB
```

```
R2#show tcp tcb 7F5323C2CE68 | include ^Option|MD5
Option Flags: non-blocking reads, non-blocking writes,
MD5 lossless password switchover, Retrans timeout
```

XR configuration is very straightforward. Under the general neighbor stanza, just enable a password and it applies to all neighbors. Again, additional LDP authentication testing is done in the LDP section.

```
mpls ldp
neighbor
password encrypted 033C69342A2B11736F3C2D2D
```

XR verification is similar exception we don't need to use the "detail" keyword.

```
RP/0/0/CPU0:XRv4#show mpls ldp neighbor | include MD5
TCP connection: 2.2.2.2:646 - 14.14.14.14:37949; MD5 on
TCP connection: 3.3.3.3:646 - 14.14.14.14:38725; MD5 on
TCP connection: 10.10.10.10:646 - 14.14.14.14:15188; MD5 on
TCP connection: 5.5.5.5:646 - 14.14.14.14:29837; MD5 on
```

12. **ICMP unreachable:** These packets can be disabled so router CPUs don't have to create tons of replies in response to a DDoS attack. Disabling unreachables can also hide network topology when using traceroute. An alternative is to leave them on with policers to implement backscatter traceback techniques. This advanced example is demonstrated later, but for now, we will disable ICMP unreachables on the external-facing links of XRv1 and CSR4. This is a very basic feature and is not discussed further after brief verification. Not surprisingly, the configuration is nearly identical for IPv6 as only the verbiage changes. XR is shown first.

```
RP/0/0/CPU0:XRv1#show ipv4 interface gig0/0/0/0.513 | include unreach
ICMP unreachables are always sent
RP/0/0/CPU0:XRv1#show ipv6 interface gig0/0/0/0.513 | include unreach
ICMP unreachables are enabled
```

```
! XRv1
interface GigabitEthernet0/0/0/0.513
ipv4 unreachables disable
interface GigabitEthernet0/0/0/0.513
ipv6 unreachables disable
```

```
RP/0/0/CPU0:XRv1#show ipv4 interface gig0/0/0/0.513 | include unreach
ICMP unreachables are never sent
RP/0/0/CPU0:XRv1#show ipv6 interface gig0/0/0/0.513 | include unreach
ICMP unreachables are disabled
```

The configuration on XE is just as easy and is shown below.

```
R4#show ip interface gig2.514 | include unreach
ICMP unreachables are always sent
R4#show ipv6 interface gig2.514 | include unreach
ICMP unreachables are sent
```

```
! CSR4
interface GigabitEthernet2.514
no ip unreachables
no ipv6 unreachables
```



```
R4#show ip interface gig2.514 | include unreach
  ICMP unreachables are never sent
R4#show ipv6 interface gig2.514 | include unreach
  ICMP unreachables are disabled
```

Both XE and XR have mechanisms to automatically rate-limit the local generation of ICMP unreachables for IPv4 only. First, we look at CSR4. I typically disable this feature in a lab environment but will re-enable it only on CSR4 for this demonstration. The default setting is to allow the router to issue ICMP unreachables once every 500 ms. There is additional granularity with respect to DF-bit (packet-too-big) unreachables since those are critical for PMTUD to function properly.

```
! CSR4 (command is enabled by default)
ip icmp rate-limit unreachable 500
```

```
R4#show ip icmp rate-limit
```

	DF bit unreachables	All other unreachables
Interval (millisecond)	500	500

Interface	# DF bit unreachables	# All other unreachables
-----	-----	-----
GigabitEthernet2.514	0	0
GigabitEthernet2.546	0	0
Loopback0	0	0

We will adjust the default policer on CSR4. I will allow DF unreachables to be sent at 100 ms intervals (more liberal) and will also log excessive unreachable generation if more than 10 unreachables are supposed to be generated in a single interval. Logs will be placed 60 seconds apart. This may be appropriate in a network where PMTUD is commonly used by hosts; it also enables extra security by logging excessive unreachables. All other unreachables still follow the 500 ms interval.

```
! CSR4
ip icmp rate-limit unreachable DF 100 log 10 60000
```

```
R4#show ip icmp rate-limit
```

	DF bit unreachables	All other unreachables
Interval (millisecond)	100	500
Log threshold (packet)	10	10
Log interval (millisecond)	60000	60000

Interface	# DF bit unreachables	# All other unreachables
-----	-----	-----
GigabitEthernet2.514	0	0
GigabitEthernet2.546	0	0
Loopback0	0	0

In XR, the configuration is similar. We will examine XRv1. I have not found a show command for the feature but it does appear to work. Logging is also not supported. In this example, I set the DF interval to 100 ms like before, and also reduce the other-unreachables limit down to 300 ms.

```
! XRv1
icmp ipv4 rate-limit unreachable DF 100
icmp ipv4 rate-limit unreachable 300
```

13. **Input hold-queue:** This represents the input FIFO queue per interface. The default depth is 75 packets, but Cisco recommends using 1500. This modification requires more memory, but if the value is too low, it could negatively influence selective packet discard (SPD), discussed later. We configure all XE routers to meet this specification, then verify it. Since the XR hardware platforms often handle this sort of thing at the line-card level (in a traffic manager process), we would apply MQC to the interface and adjust the queue depth in class default. The QoS section showed us that XRv won't accept this configuration, so we will skip XR to focus on CSR4 and CSR8. I only show the configuration from CSR8 since it is identical on CSR4. Notice that the command is per physical interface; my current design uses a single physical interface for testing, but in real life, dedicated physical interfaces should be used to face internally and externally.

```
! CSR8
interface GigabitEthernet2
  hold-queue 1500 in
```

```
R8#show interfaces gig2 | include Input
Input queue: 0/1500/0/0 (size/max/drops/flushes); Total output drops: 0
```

14. **Selective packet discard (SPD):** SPD is used to create a global pool of buffers to add "headroom" to the input queues for important packets (IGP, BGP, keepalives etc). Cisco advises increasing this headroom from 100 to 1000 packets, memory permitting (this is the new default on XE). This feature is not supported for XR or IPv6, and is a hidden command on XE. We only evaluate it because it is explicitly listed in Cisco's best-practices document. After the input queue is full, SPD offers a mechanism for important, router-destined traffic to not be dropped. The admission criteria for this headroom include things like BGP, IGP, and IPP6 traffic. Even beyond this headroom is the "extended headroom" which is even more stringent on entry criteria and is used when the headroom is full. This includes layer 2 keepalives, LDP (in some versions), and ARP, as examples. Packets of a low priority, such as an IPP0 ping, can never enter the headroom or extended headroom. A BGP packet could never enter the extended headroom, but is allowed in the regular headroom. The SPD "mode" is deprecated and used to mean the difference between accepting any packet (normal) versus rejected all malformed packets (aggressive). Routers nowadays always reject malformed packets and the "ip spd mode aggressive" command does nothing. We can see the default settings with "show ip spd".

```
R8#show ip spd
Current mode: normal.
```

```
Queue min/max thresholds: 73/74, Headroom: 1000, Extended Headroom: 10
IP normal queue: 0, priority queue: 0.
SPD special drop mode: none
```

Notice the queue thresholds. These are based on the default input-queue value of 75. When the packets in the queue are between min and max, random dropping occurs. In this case, that virtually never happens, because the two are adjacent numbers. In our case, we need to tune these values so they better reflect our new input queue size. When the input queue is full and the traffic is not a candidate for SPD headroom, all packets are tail dropped.

```
! CSR8
ip spd queue max 1499
ip spd queue min 1000
```

We will also increase the headroom and extended headroom as Cisco recommends. We will set both standard headroom and extended headroom values to 1500 for good measure. Notice these commands are not prefixed with “ip” unlike the previous commands, for some reason.

```
! CSR8
spd headroom 1500
spd extended 1500
```

We can see the final result of our configuration to ensure all our new changes were successful. All of these commands are hidden in the parser, as previously mentioned.

```
R8#show ip spd
Current mode: normal.
Queue min/max thresholds: 1000/1499, Headroom: 1500, Extended Headroom: 1500
IP normal queue: 0, priority queue: 0.
SPD special drop mode: none
```

15. Control-plane policing and Local Packet Transport Service (CPP and LPTS): On XE and XR respectively, these features police packets destined to the router itself. It is often used to prevent against DDoS attacks targeting network devices that try to hog resources. This can cause critical router processes, such as BGP/IGP, to fail. These features are examined in detail in a dedicated section.

49.2 Management plane security best practices

1. Disable unnecessary services: Disabling features not needed will reduce the number of attack vectors. In our environment, I’ve selected a handful of uncommonly used services to demonstrate the process. Many of these are off by default, but this may change between versions. Note that disabling “service dhcp” disables server, client, and relay functionality.

```
! IOS and XE “unnecessary” services
no service tcp-small-servers
```

```
no service udp-small-servers
no service dhcp
no service pad
no service config
no service finger
no service call-home
```

All four of the following services are off by default in XR.

```
! XR "unnecessary" services
no service ipv4 tcp-small-servers
no service ipv4 udp-small-servers
no service ipv6 tcp-small-servers
no service ipv6 udp-small-servers
```

2. **Appropriate banners:** Banners often serve to detail the legal consequences of attacking the system. I just made a quick message-of-the-day (motd) banner to demonstrate this, but normally there would be login, exec, and incoming banners also. The configuration is identical in XE and XR.

```
! All BGP AS 85 XE and XR routers
banner motd %
*** Warning: Protected system ***
%
```

3. **Enable secret:** The word “secret” in this context refers to a password used to protect privileged access to the router. It does not apply to XR since there is always a username/password.

```
! CSR6
enable secret TEST_PASS
```

4. **Exec-timeout on CON/VTY lines:** This feature prevent an exec shell from staying open forever. I’ve set these to never timeout for lab use, but they are tunable. More realistic settings could use a 10 minute VTY timeout with a 30 minute console timeout. The first digit is minutes and the last is seconds.

```
! CSR8
line con 0
exec-timeout 0 0
```

```
! XRv1
line console
exec-timeout 0 0
```

5. **TCP-keepalives:** This feature enforces a timeout inactive or dysfunctional TCP sessions by issuing keepalives. Automatically enabled in XR and not configurable. Not enabled in XE by default. Enabling them bidirectionally on both routers makes the most sense, so either side can tear down the session if connectivity is lost.

```
! CSR6 and CSR8
service tcp-keepalives-in
service tcp-keepalives-out
```

As a quick test, we will telnet from CSR6 to CSR8, then check the session details. We can see that CSR8 shows the keepalive feature enabled for this telnet session. If there is a network outage and no activity is seen from the TCP client, the session is torn down after about 5 minutes.

```
R6# telnet 85.6.8.8
Trying 85.6.8.8 ... Open
```

```
*** Warning: Protected system ***
```

```
R8#show tcp brief | include 85\.6
7F0DD04C0178 85.6.8.8.23 85.6.8.6.38383 ESTAB
```

```
R8#show tcp tcb 7F0DD04C0178 | include ^Option
Option Flags: keepalive running, Retrans timeout
```

6. VTY access-class (ACL) security: This can prevent certain networks/hosts from accessing the device. I've added a very liberal ACL to demonstrate this on CSR8 and XRv1. It doesn't allow loopbacks in my network to telnet in, but allows sessions to or from any transit link.

```
! CSR8
ip access-list standard ACL_VTY_ACCESS
 permit 85.0.0.0 0.255.255.255

line vty 0 4
 access-class ACL_VTY_ACCESS in
 access-class ACL_VTY_ACCESS out
```

XR VTY line configuration is rather complex. First, we need to say how many telnet server instances (concurrent connections) we allow. We then create a line template that we can apply to lines later on. Under this template, we apply our security controls; we use the same ACL logic shown above for CSR8. Last, we create a VTY pool to identify which VTY lines can be used (0 through 4) and specify the template that defines the line parameters.

```
! XRv1
ipv4 access-list ACL_VTY_ACCESS
 10 permit ipv4 85.0.0.0 0.255.255.255 any

telnet vrf default ipv4 server max-servers 3

line template VTY
 access-class ingress ACL_VTY_ACCESS
```

```
access-class egress ACL_VTY_ACCESS

vty-pool default 0 4 line-template VTY
```

7. **Usage of SSHv2:** SSHv2 is more secure than SSHv1 and telnet for device management. We will extend the security of the previous configuration to disable telnet and enable SSHv2 on XRv1 and CSR8. SSH requires a username/pass login. This is unlike telnet which, in the most lenient case, requires no authentication. We must define a domain-name before we can generate our RSA keys used for SSH. We also will disable telnet as an option in our incoming VTY line stanza, which ensures telnet is never allowed for managing CSR8. Then, we configure some extra SSH options, including logging and version 2.

```
! CSR8
username ssh privilege 15 password ssh
ip domain-name lab.local
crypto key generate rsa modulus 1024

line vty 0 4
  transport input none
  transport input ssh
  login local

ip ssh ver 2
ip ssh authentication-retries 2
ip ssh logging events
```

A successful SSH session from an SSH client (CSR6) creates a detailed log message. After logging in, CSR6 logs out, which is also a logged event.

```
! CSR8, when CSR6 logs in
%SSH-5-SSH2_SESSION: SSH2 Session request from 85.6.8.6 (tty = 0) using
crypto cipher 'aes128-ctr', hmac 'hmac-sha1' Succeeded
%SSH-5-SSH2_USERAUTH: User 'ssh' authentication for SSH2 Session from
85.6.8.6 (tty = 0) using crypto cipher 'aes128-ctr', hmac 'hmac-sha1'
Succeeded

! CSR8, when CSR6 logs out
%SSH-5-SSH2_CLOSE: SSH2 Session from 85.6.8.6 (tty = 0) for user 'ssh' using
crypto cipher 'aes128-ctr', hmac 'hmac-sha1' closed
```

A failed session also creates a log message. Don't be fooled by the first message that says the session opened; this is the low-level TCP negotiation process. The user failed to enter the proper password which means authentication ultimately failed.

```
! CSR8, when CSR6 uses the incorrect password
```

```
%SSH-5-SSH2_SESSION: SSH2 Session request from 85.6.8.6 (tty = 0) using
crypto cipher 'aes128-ctr', hmac 'hmac-sha1' Succeeded
R8(config)#
%SSH-5-SSH2_USERAUTH: User 'ssh' authentication for SSH2 Session from
85.6.8.6 (tty = 0) using crypto cipher 'aes128-ctr', hmac 'hmac-sha1' Failed
%SSH-5-SSH2_CLOSE: SSH2 Session from 85.6.8.6 (tty = 0) for user 'sh' using
crypto cipher 'aes128-ctr', hmac 'hmac-sha1' closed
```

In XR, having already defined the VTY pool, we only need to enable SSH and our RSA keys. Note that the RSA keys are generated in exec mode, unlike XE. XR also does not appear to require an explicitly-defined domain-name before trying to generate RSA keys for SSH.

```
! XRv1
ssh server v2
ssh server logging
ssh server session-limit 3

! XRv1 exec mode
crypto key generate rsa general-keys
```

Below is a successful and failed login, with messages similar to those seen in XE.

```
! XRv1, successful login from CSR5
RP/0/0/CPU0:20:43:46.455 : SSHD_[65713]: %SECURITY-SSHD-6-INFO_SUCCESS :
Successfully authenticated user 'nrusso' from '85.1.5.5' on 'vty0' (cipher
'aes128-cbc', mac 'hmac-sha1')

! XRv1, failed login from CSR5
RP/0/0/CPU0:20:43:58.474 : SSHD_[65713]: %SECURITY-SSHD-4-INFO_FAILURE :
Failed authentication attempt by user 'nrusso' from '85.1.5.5' on 'vty0'
RP/0/0/CPU0:20:43:59.664 : SSHD_[65713]: %SECURITY-SSHD-3-ERR_GENERAL :
Failed to receive User authentication request
```

8. **One-time passwords:** Useful for when SSH is not available on a device or when you know a user will only need to access a device one time. We will demonstrate this on CSR6. There are two caveats with this feature that may not be obvious. First, if a user logs in using some other username, the one-time password is not deleted. Second, once the one-time user logs in, the username is removed from the running-config, so if the startup-config still has it, a reboot will restore the one-time user/password.

```
! CSR6
username ONCE one-time password ONCE

R6#show run | include ONCE
username ONCE one-time password 0 ONCE
```

We log into CSR6 once from CSR5 successfully.

```
R5#telnet 6.6.6.6
Trying 6.6.6.6 ... Open
```

```
Username: ONCE
Password: ONCE
```

```
R6>
```

After exiting, we cannot log in again. CSR6 has removed the “ONCE” username from its running configuration. Reloading CSR6 would technically work to generate another one-time login, assuming the command was written to the startup configuration.

```
R5#telnet 6.6.6.6
Trying 6.6.6.6 ... Open
```

```
Username: ONCE
Password: ONCE
```

```
% Login invalid
```

```
R6#show run | include ONCE
[no output]
```

9. **AAA:** Authentication, Authorization, and Accounting (AAA) is a subsystem that centralizes these 3 features in a RADIUS or TACACS server. RADIUS and TACACS can be sizable topics if one looks at all the options. The general idea is to centralize certain network security functions so individual logins need not exist on every device. It can also be used for 802.1x, which is a port-based authentication mechanism on switches capable of assigning VLANs based on authentication level.

AAA server configuration on IOS-like platforms (XE) has changed in recent releases to be more modular, much like XR. On CSR3, I configure a pair of RADIUS servers, one using the new ports (authentication port 1812, accounting port 1813) and one using the old ports (auth 1645, acct 1646). Both point to a notional RADIUS server; one is IPv4 and one is IPv6. I’ve added some extra variation; the retransmit option is simply a retry count for accessing the server, and the timeout is measured in seconds for a response to be considered inconclusive. The automated tester works like IP SLA where it just tests reachability to the RADIUS server. Normally this can be done with the “test” exec-level command, but an automated mechanism is useful also.

```
! CSR3
radius server RADIUS_NEW_IPV6
  address ipv6 ::1:2:3:4 auth-port 1812 acct-port 1813
  timeout 1
  retransmit 3
  key RAD_KEY

radius server RADIUS_OLD_IPV4
```



```
address ipv4 1.2.3.4 auth-port 1645 acct-port 1646
automate-tester username RAD_TEST probe-on ignore-acct-port
key RAD_KEY
```

Below, we examine the server statistics. Because the first server isn't actively doing anything, I removed the long list statistics full of zeroes. The second server is trying to authenticate the user RAD_TEST and failing due to timeout. We see a different statistic stanza for authentication, authorization, and accounting. Interestingly, the router thinks the first server is up, probably because nothing has failed yet. The second server is considered "dead" as a result of the automated test failures.

```
R3#show aaa servers
```

```
RADIUS: id 1, priority 1, host ::1:2:3:4, auth-port 1812, acct-port 1813
```

```
State: current UP, duration 177s, previous duration 0s
```

```
Dead: total time 0s, count 0
```

```
[snip]
```

```
RADIUS: id 2, priority 2, host 1.2.3.4, auth-port 1645, acct-port 1646
```

```
State: current DEAD, duration 122s, previous duration 0s
```

```
Dead: total time 122s, count 0
```

```
Quarantined: No
```

```
Authen: request 4, timeouts 4, failover 0, retransmission 3
```

```
Response: accept 0, reject 0, challenge 0
```

```
Response: unexpected 0, server error 0, incorrect 0, time 0ms
```

```
Transaction: success 0, failure 1
```

```
Throttled: transaction 0, timeout 0, failure 0
```

```
Author: request 0, timeouts 0, failover 0, retransmission 0
```

```
Response: accept 0, reject 0, challenge 0
```

```
Response: unexpected 0, server error 0, incorrect 0, time 0ms
```

```
Transaction: success 0, failure 0
```

```
Throttled: transaction 0, timeout 0, failure 0
```

```
Account: request 0, timeouts 0, failover 0, retransmission 0
```

```
Request: start 0, interim 0, stop 0
```

```
Response: start 0, interim 0, stop 0
```

```
Response: unexpected 0, server error 0, incorrect 0, time 0ms
```

```
Transaction: success 0, failure 0
```

```
Throttled: transaction 0, timeout 0, failure 0
```

```
Elapsed time since counters last cleared: 2m
```

```
Estimated Outstanding Access Transactions: 0
```

```
Estimated Outstanding Accounting Transactions: 0
```

```
Estimated Throttled Access Transactions: 0
```

```
Estimated Throttled Accounting Transactions: 0
```

```
Maximum Throttled Transactions: access 0, accounting 0
```

```
Requests per minute past 24 hours:
```

```
high - 0 hours, 1 minutes ago: 4
```

```
low - 0 hours, 2 minutes ago: 0
```

```
average: 2
```

We can manually test a AAA server from exec mode as mentioned above. Even with the failed test of trying to reach the IPv6 server, the output still says up. We see exactly 4 failed requests, because there were 3 retries after the first failure, as we configured.

```
R3#test aaa group radius server name RADIUS_NEW_IPV6 RAD_USER rad_pass legacy
Attempting authentication test to server-group radius using radius
No authoritative response from any server.
```

```
R3#show aaa servers
RADIUS: id 1, priority 1, host ::1:2:3:4, auth-port 1812, acct-port 1813
  State: current UP, duration 438s, previous duration 0s
  Dead: total time 0s, count 0
  Quarantined: No
  Authen: request 4, timeouts 4, failover 0, retransmission 3
    Response: accept 0, reject 0, challenge 0
    Response: unexpected 0, server error 0, incorrect 0, time 0ms
    Transaction: success 0, failure 1
[snip]
```

We can also group servers together. For example, we might want to use both servers as potential candidates for login authentication. I also demonstrate adding a “private” RADIUS server which can only be accessed by referencing the group. So, the group contains three servers, two public and one private. The show command referencing the group reveals all servers encompassed by the group, seen below.

```
! CSR3
aaa group server radius RAD_GROUP
  server name RADIUS_NEW_IPV6
  server name RADIUS_OLD_IPV4
  server-private 4.3.2.1 key RAD_KEY

R3#show radius server-group RAD_GROUP
Server group RAD_GROUP
  Sharecount = 1  sg_unconfigured = FALSE
  Type = standard  Memlocks = 1
  Server(::1:2:3:4:1812,1813) Transactions:
    Authen: 0  Author: 0  Acct: 0
  Server_auto_test_enabled: FALSE
  Keywrap enabled: FALSE
  Server(1.2.3.4:1645,1646) Transactions:
    Authen: 0  Author: 0  Acct: 0
  Server_auto_test_enabled: TRUE
  Keywrap enabled: FALSE
  Server(4.3.2.1:1645,1646) Transactions:
    Authen: 0  Author: 0  Acct: 0
  Server_auto_test_enabled: FALSE
  Keywrap enabled: FALSE
```

This server group can be added a “method-list” for authentication. For PPP authentication, such as PAP, CHAP, and EAP, we use this server group by default. If it fails, then AAA will fall back to the local database. Be careful when enabling AAA globally; it automatically relies on local logins for VTY lines unless manually overridden.

```
! CSR3
aaa new-model
aaa authentication ppp default group RAD_GROUP local

R3#show aaa method-lists authentication | include PPP|RAD_GROUP
  authen queue=AAA_ML_AUTHEN_PPP
    name=default valid=TRUE id=0 :state=ALIVE : SERVER_GROUP RAD_GROUP LOCAL
```

TACACS+ configuration is almost identical, with only small changes. Here is a mirrored configuration from the RADIUS example, with two public TACACS servers and one private TACACS server in a server group. I also show the verification quickly; despite being a AAA server, it doesn’t show under the show commands for AAA in all categories. The TCP port for TACACS is 49; I change this to 94 for the IPv6 server for variety. Notice that the output annotates which servers are “public” and which are “private”; the AAA show commands used above for RADIUS servers did not reveal this.

```
! CSR3
aaa group server tacacs+ TAC_GROUP
  server name TAC_OLD_IPV4
  server name TAC_NEW_IPV6
  server-private ::4:3:2:1 key TAC_KEY

tacacs server TAC_NEW_IPV6
  address ipv6 ::1:2:3:4
  key TAC_KEY
  port 94
  single-connection

tacacs server TAC_OLD_IPV4
  address ipv4 1.2.3.4
  key TAC_KEY

R3#show tacacs
Tacacs+ Server - public :
    Server name: TAC_NEW_IPV6
    Server address: ::1:2:3:4
    Server port: 94
    Socket opens: 0
    Socket closes: 0
    Socket aborts: 0
    Socket errors: 0
    Socket Timeouts: 0
    Failed Connect Attempts: 0
```

```
Total Packets Sent:      0
Total Packets Recv:      0
Expected Replies:        0
```

```
Tacacs+ Server - public :
    Server name: TAC_OLD_IPV4
    Server address: 1.2.3.4
    Server port: 49
[snip]
```

```
Tacacs+ Server - private :
    Server address: ::4:3:2:1
    Server port: 49
[snip]
```

Let's assume this group should also be used for PPP authentication but for a custom link. We define a non-default method-list and bind the TACACS server group to it. Under the PPP interface, we would reference this method-list for authentication (not configured) as needed.

```
! CSR3
aaa authentication ppp CUSTOM group TAC_GROUP none

R3#show aaa method-lists authentication | include PPP|CUSTOM
authen queue=AAA_ML_AUTHEN_PPP
    name=CUSTOM valid=TRUE id=D1000003 :state=ALIVE : SERVER_GROUP TAC_GROUP
NONE
```

XR's configuration is very similar to XE, except there does not appear to be IPv6 host support in v5.3.0. Rather than define a server name, we just specify the host address and ports in the initial server definition. I define two public servers and one private server, wrapped in a server group. One of the public servers has automated testing enabled, so I created a fake user for that. Within the group there are various load sharing options so all servers can be used "equally". I also specify the RADIUS source interface, which is a common practice to ensure consistent reachability, especially in an SP core with IGP prefix-suppression enabled.

```
! XRv2
radius source-interface Loopback0 vrf default
radius-server host 1.2.3.4 auth-port 1645 acct-port 1646
    key 7 1425332F332F0F12
    timeout 1
    test username RAD_TEST
    retransmit 3
    ignore-acct-port

radius-server host 4.3.2.1 auth-port 1812 acct-port 1813
    key 7 1425332F332F0F12
```

```

username RAD_TEST
  group operator
  password 7 122B24332D20293D

aaa group server radius RAD_GROUP
  server 4.3.2.1 auth-port 1812 acct-port 1813
  server 1.2.3.4 auth-port 1645 acct-port 1646
  load-balance method least-outstanding batch-size 25
  server-private 6.7.8.9 auth-port 1645 acct-port 1646
  key 7 0121272064202336

```

A few basic show commands can verify our configuration. The outputs are not discussed in detail because they are very similar to XE outputs. Notice that the first server has automated testing enabled, which is failing because the RADIUS server is imaginary. The last server is private and is only accessible by invoking the server-group RAD_GROUP. Like the XE show commands for TACACS, this output shows us which servers are “public” and which are “private”.

```

RP/0/0/CPU0:XRv2#show radius authentication
Server: 1.2.3.4, port: 1645/
  0 requests, 0 pending, 0 retransmits
  0 accepts, 0 rejects, 0 challenges
  0 timeouts, 0 bad responses, 0 bad authenticators
  0 unknown types, 0 dropped, 0 ms latest rtt
  Throttled: 0 transactions, 0 timeout, 0 failures
  Estimated Throttled Access Transactions: 0
  Maximum Throttled Access Transactions: 0
  Automated TEST Stats:
    1141 requests, 1141 timeouts, 0 response, 0 pending

Server: 4.3.2.1, port: 1812/
  [snip]

Server: 6.7.8.9, port: 1645/ [RAD_GROUP / private]
  [snip]

```

```

RP/0/0/CPU0:XRv2#show radius source-interfaces
Interface          IP-Address          VRF Id
Loopback0          12.12.12.12         0x60000000

```

```

RP/0/0/CPU0:XRv2#show radius server-groups
Global list of servers
  Contains 2 server(s)

```

```

Server 1.2.3.4/1645/1646  is DEAD

```

```

Server 4.3.2.1/1812/1813  is UP

```

```

Server group 'RAD_GROUP' has 3 server(s)

```

```
Dead time: 0 minute(s) (inherited from global)
Contains 3 server(s)
```

```
Server 4.3.2.1/1812/1813 is UP
```

```
Server 1.2.3.4/1645/1646 is DEAD
```

```
Server 6.7.8.9/1645/1646 [private] is UP
```

The RADIUS group is bound identically to XE, where it can be applied to a method-list for any type of authentication/authorization that supports RADIUS server groups. We use the PPP example again.

```
! XRv2
aaa authentication ppp RAD_PPP group RAD_GROUP local
```

TACACS+ configuration in XR is almost identical except introduces some TACACS-specific features and terms. We set the TACACS DSCP to CS6 globally for IPv6 packet; this will increase the probability of TACACS traffic reaching the server when originated. The configuration is shown below.

```
! XRv2
tacacs-server host ::1:2:3:4 port 49
  key 7 06322E0273652C20
  single-connection

tacacs-server host ::5:4:2:1 port 94
  key 7 1331363134272113

aaa group server tacacs+ TAC_GROUP
  server ::5:4:2:1
  server ::1:2:3:4
  server-private 1.2.3.4 port 49
  key 7 073B006F71223C3C

tacacs-server ipv6 dscp cs6
```

The verification is almost similar to RADIUS. We can see the private server marked as such, but otherwise there is no TACACS activity at present.

```
RP/0/0/CPU0:XRv2#show tacacs
Server: ::1:2:3:4/49 opens=0 closes=0 aborts=0 errors=0
  packets in=0 packets out=0
  status=up single-connect=true family=IPv6

Server: ::5:4:2:1/94 opens=0 closes=0 aborts=0 errors=0
  packets in=0 packets out=0
  status=up single-connect=false family=IPv6
```

```
Server: 1.2.3.4/49 vrf=default [private]
      opens=0 closes=0 aborts=0 errors=0
      packets in=0 packets out=0
      status=up single-connect=false family=IPv4
```

10. **Local passwords:** Local passwords are irreversibly encrypted and used as last resort to manage a device. Newer versions of IOS support SHA256 and SCRYPT mechanisms, in addition to MD5, for securing the password. PBKDF2 and SCRYPT are both key derivation functions (KDFs), and are more secure than MD5. The technical differences between the two are beyond the scope of CCIE SP, but Cisco seems to hint that SCRYPT is more secure.

```
R6(config)#username SECRET_USER algorithm-type ?
  md5      Encode the password using the MD5 algorithm
  scrypt   Encode the password using the SCRYPT hashing algorithm
  sha256   Encode the password using the PBKDF2 hashing algorithm
```

```
! CSR6
username SECRET_USER algorithm-type scrypt secret password
```

The enter result is an enormous hashed password string, which is very hard to crack. MD5 was considered level 5. In terms of new features, PBKDF2 is level 8 and SCRYPT is level 9. Looking at the configuration, we can see the level 9 password encryption.

```
R6#show running-config | include SECRET_USER
username SECRET_USER secret 9
$9$8dXXbV3TNlYym8$fsIu5emmXjFRyJjba.ps8QUSKy5Iz3HlhDzm3ai82Vw
```

XR only appears to support the MD5 method, as no options are available for anything newer.

```
! XRv1
username SECRET_USER
  secret 5 $1$JeSw$qf3PnPhDVZiElr/Icj3H5/
```

11. **Privilege level tuning:** This technique is based on the premise of only granting users “just enough” privilege to do their jobs. On CSR7, we create a JUNIOR_PRIV account for a junior engineer. At privilege level 5, we grant access to all show and debug commands, but no configuration or test commands, for example.

```
! CSR7
username JUNIOR_PRIV privilege 5 nopassword
privilege exec all level 5 show
privilege exec all level 5 debug
privilege exec all level 5 undebug
```

We log into CSR7 using this new username and quickly verify that our user changes to reflect this new user. Any activities done by this user will be logged as such (if logging is configured).

```
R7#login
```

```
Username: JUNIOR_PRIV
```

```
R7#show users
```

Line	User	Host(s)	Idle	Location
* 0 con 0	JUNIOR_PRI	idle	00:00:00	

The user has access only to the commands at level 5 and below, which include only show, debug, and undebug commands. This is effectively a “read-only” view of the router.

```
R7#debug ?
```

aaa	AAA Authentication, Authorization and Accounting
access-expression	Boolean access expression
[snip]	

```
R7#show ?
```

aaa	Show AAA values
access-expression	List access expression
[snip]	

```
R7#undebug ?
```

aaa	AAA Authentication, Authorization and Accounting
access-expression	Boolean access expression
[snip]	

```
R7#test ?
```

```
% Unrecognized command
```

```
R7#configure ?
```

```
% Unrecognized command
```

XR does not appear to support this legacy feature and instead implements a flavor of RBAC.

12. **RBAC:** Role-Based Access Control is a more modular way of adjusting user privileges. In this way, we create “views” which define what a user can do. Users are then assigned views which determine their privileges upon the creation of an exec shell. This is more flexible and intuitive than the privilege method. On CSR7, I create a few roles with various requirements:

- SHOW_ISIS: can only execute “show” commands relating to IS-IS
- DEBUG_ISIS: can only execute “debug” commands relating to IS-IS
- ALL_ISIS: able to execute any “show” or “debug” commands relating to IS-IS

We will define three separate roles (known as “views”) to describe the capabilities of each category, then optionally assign them to usernames. There is also a hidden “root” view that gives full access.

```
! CSR7
parser view SHOW_ISIS
  secret 5 $1$A/Az$3x8cHb/Gq.asVZZRjuPa00
  commands exec include all show isis
  commands exec include all show ip
  commands exec include show

parser view DEBUG_ISIS
  secret 5 $1$F/vA$qFQWdvXO/tSvluHjkQWl41
  commands exec include all undebug isis
  commands exec include all undebug
  commands exec include all debug isis
  commands exec include debug

parser view ALL_ISIS superview
  secret 5 $1$r1m8$9CLQrKv2IjLW9pKZzQPgm.
  view SHOW_ISIS
  view DEBUG_ISIS
```

From the shell, you can either log in as a user with a view or manually select the view using the “enable” command. Below we see an error that AAA must be configured. Once that is done, we can see the view works properly.

```
R7#enable view ALL_ISIS
% AAA must be configured.
```

```
! CSR7
aaa new-model
aaa authentication login default local
```

After enabling a basic AAA for local login, we can change the view. We verify that we have all the show and debug commands for IS-IS with this view. The commands don’t include other features, as expected, since those were not included in the view.

```
R7#enable view ALL_ISIS
Password: [the view password goes here]
```

```
R7#show parser view
Current view is 'ALL_ISIS'
```

```
R7#show ?
  banner      Display banner information
  bootflash:  display information about bootflash: file system
  flash:      display information about flash: file system
```

ip	IP information
isis	IS-IS routing information
parser	Display parser information

R7#show isis ?

*	All IS-IS address families
WORD	IS-IS Process tag (use "null" for no tag)
database	IS-IS link state database

R7#debug ?

isis	IS-IS Information

R7#debug isis ?

*	All IS-IS address families
WORD	IS-IS Process tag (use "null" for no tag)
adj-packets	IS-IS Adjacency related packets

On XR, there are several pre-made groups. The groups are pretty self-explanatory and the documentation spells out the specifics of each. The XR constructs require administrators to determine a set of tasks (taskgroup) and then assign those sets of tasks to sets of users (usergroup). Users themselves inherit from usergroups, creating a 3-level hierarchy. I have defined a few custom task groups to manage IGP/LDP and MPLS functionality, and then referenced them from an MPLS management user group. MPLS administrators would be placed into this user group, ideally. The task options are read, write, execute, and debug.

```
! XRv4
taskgroup MPLS
  task read l2vpn
  task read mpls-te
  task write l2vpn
  task write mpls-te
  task execute l2vpn
  task execute mpls-te
  task debug mpls-te

taskgroup IGP_LDP
  task read isis
  task read mpls-ldp
  task execute isis
  task execute mpls-ldp
  task debug isis
  task debug mpls-ldp

usergroup MPLS_MGR
  taskgroup MPLS
  taskgroup IGP_LDP
  description MANY MPLS RELATED TASKS
```

```
username MPLS_GUY
group MPLS_MGR
password 7 15020A1F173D24362C
```

If we log in as MPLS_GUY, we have no mechanism to change/modify IS-IS or LDP since the “write” permission was not granted in the union of all permissions between all inherited task-groups. We also cannot debug L2VPN, but we can configure it. We can verify permissions and our own identity with some basic commands. Once we verify this, we test out our capabilities to ensure the restricted permissions are in effect. Rather than suggest the command is “unrecognized” as XE does, XR is more honest by saying a user is not authorized to perform a specific action.

```
RP/0/0/CPU0:XRv4#show user
```

```
MPLS_GUY
```

```
RP/0/0/CPU0:XRv4#show user tasks
```

```
Task:          isis      : READ          EXECUTE      DEBUG
Task:          l2vpn     : READ          WRITE        EXECUTE
Task:          mpls-ldp   : READ          EXECUTE      DEBUG
Task:          mpls-te    : READ          WRITE        DEBUG
```

```
RP/0/0/CPU0:XRv4(config)#mpls ldp
```

```
% This command is not authorized
```

```
RP/0/0/CPU0:XRv4(config)#router isis 1
```

```
% This command is not authorized
```

```
RP/0/0/CPU0:XRv4#debug l2vpn ?
```

```
% This command is not authorized
```

This user is authorized to look at IS-IS/LDP information, as well as configure global MPLS-TE parameters.

```
RP/0/0/CPU0:XRv4#show mpls ldp neighbor brief
```

Peer	GR	NSR	Up Time	Discovery		Addresses		Labels	
				ipv4	ipv6	ipv4	ipv6	ipv4	ipv6
2.2.2.2:0	N	N	21:13:22	2	0	5	0	11	0
3.3.3.3:0	N	N	21:13:16	1	0	3	0	13	0
10.10.10.10:0	N	N	21:13:12	1	0	13	0	8	0
5.5.5.5:0	N	N	21:13:12	1	0	6	0	16	0

```
P/0/0/CPU0:XRv4#show isis neighbors summary
```

```
IS-IS 1 neighbor summary:
```

State	L1	L2	L1L2
Up	3	2	0
Init	0	0	0
Failed	0	0	0

```
RP/0/0/CPU0:XRv4(config)#mpls traffic-eng ?
```

```
affinity-map          Affinity Map Configuration
[snip]
```

The “execute” option relates to things from the exec shell, such as direct operations on a feature or using the “clear” command. We do not have execute permissions for MPLS-TE, so we cannot reoptimize tunnels for example. We could clear IS-IS neighbors or MPLS LDP neighbors, though.

```
RP/0/0/CPU0:XRv4#clear isis ?
  afi-all      IPv4 and IPv6 commands
  instance      Clear data from a single IS-IS instance
[snip]
```

```
RP/0/0/CPU0:XRv4#clear mpls ?
  ldp  Label Distribution Protocol information
  oam   OAM information
```

```
RP/0/0/CPU0:XRv4#mpls ?
      ^
```

```
% Invalid input detected at '^' marker.
```

13. **Auditing/tracking configurations:** This ensures that there are no unauthorized changes to device configurations over time. This is more of a policy application than configuration. XR handles this automatically with the commit list and rollback feature. XE can use the “archive” feature to accomplish a similar mechanism of backing up configurations, demonstrated on CSR2. I create a folder on the bootflash for the configurations (“configs”), then tell the archive feature to store up to 5 copies of backups. **You must include the filename prefix in the path, not just a folder, or else archive does nothing.** This occurs automatically every time I save the configuration, also. This works with “write memory” and the newer “copy run start”. You can also manually backup a configuration with “archive config”. I demonstrate all three methods below.

```
! CSR2
mkdir bootflash:/configs

archive
  path bootflash:/configs/ARCHIVE_CFG_
  maximum 5
  write-memory
```

We show that the folder was created successfully, then save the configuration on CSR2. This automatically archives the configuration to our “configs” directory as shown below.

```
R2#dir bootflash: | include configs
600289  drwx          4096  Jul 19 2015 21:01:14 +00:00  configs
```

```
R2#write memory
Building configuration...
```

[OK]

```
R2#dir bootflash:/configs
Directory of bootflash:/configs/
```

```
600290  -rw-      8300   Jul 19 2015 21:06:22 +00:00  ARCHIVE_CFG_-Jul-19-21-06-22.321-0
```

Below, I demonstrate the manual method. This creates another archive in the “configs” folder.

```
R2#archive config
```

```
R2#dir bootflash:/configs
Directory of bootflash:/configs/
```

```
600290  -rw-      8300   Jul 19 2015 21:06:22 +00:00  ARCHIVE_CFG_-Jul-19-21-06-22.321-0
600291  -rw-      9594   Jul 19 2015 21:06:39 +00:00  ARCHIVE_CFG_-Jul-19-21-06-39.060-1
```

Rather than use the “write memory” legacy command, the newer “copy run start” also works.

```
R2#copy running-config startup-config
```

```
Destination filename [startup-config]?
Building configuration...
[OK]
```

```
R2#dir bootflash:/configs
Directory of bootflash:/configs/
```

```
600290  -rw-      8300   Jul 19 2015 21:06:22 +00:00  ARCHIVE_CFG_-Jul-19-21-06-22.321-0
600291  -rw-      9594   Jul 19 2015 21:06:39 +00:00  ARCHIVE_CFG_-Jul-19-21-06-39.060-1
600292  -rw-      8300   Jul 19 2015 21:06:54 +00:00  ARCHIVE_CFG_-Jul-19-21-06-54.350-2
```

There is no configuration required in XR, but as a quick example, we can see the past configuration commits easily. I added a loopback999 interface in my last commit and added a brief “label” to it. I also added a longer comment which is revealed using the detail keyword. Other commits just have numeric, sequential labels (the label is set to the commit ID), and no comments. These labels/comments can be used for easy file identification when performing rollbacks.

```
RP/0/0/CPU0:XRv1(config)#commit label ADDED_LO99 comment this is cool
```

```
RP/0/0/CPU0:XRv1#show configuration commit list
```

SNo.	Label/ID	User	Line	Client	Time Stamp
1	ADDED_LO99	nrusso	con0_0_CPU0	CLI	[snip]
2	1000000156	nrusso	con0_0_CPU0	CLI	[snip]

```
RP/0/0/CPU0:XRv1#show configuration commit list detail
```

1) CommitId: 1000000157	Label: ADDED_LO999
UserId: nrusso	Line: con0_0_CPU0
Client: CLI	Time: [snip]

Comment: this is cool

2) CommitId: 1000000156	Label: NONE
UserId: nrusso	Line: con0_0_CPU0
Client: CLI	Time: [snip]

14. **Logging:** Logs should be sent to a few syslog servers for offline storage. Syslog commonly uses UDP 514 or sometimes TCP 601 for transport. I've configured two "syslog servers" using CSR5 as a syslog client to demonstrate the basic configuration. I instruct the router to send messages of severity level 5 or lower (lower numerically but higher priority) to the syslog servers, along with using its hostname as the ID inside the packet. Counters are shown below, and notice that the TCP-based logging is not working, as the session does not exist. UDP based transport is totally asynchronous and the messages are sent whether the target is a syslog server or not.

```
! CSR5
logging trap notifications
logging origin-id hostname
logging host ipv6 ::14:14:14:14 transport tcp
logging host 14.14.14.14
```

```
R5#show logging | section Trap
Trap logging: level notifications, 247 message lines logged
  Logging to ::14:14:14:14 (tcp port 601, audit disabled,
    link down),
    0 message lines logged,
    0 message lines rate-limited,
    0 message lines dropped-by-MD,
    xml disabled, sequence number disabled
    filtering disabled
  Logging to 14.14.14.14 (udp port 514, audit disabled,
    link up),
    6 message lines logged,
    0 message lines rate-limited,
    0 message lines dropped-by-MD,
    xml disabled, sequence number disabled
    filtering disabled
Logging Source-Interface:      VRF Name:
```

The XR configuration is similar, except I specify two different severity levels for the syslog hosts. I did not enable TCP transport, but notice that because no critical messages have occurred, the IPv6 server has not received any packets. I can also set the source interface and originator string inside the log message.

```
! XRv1
logging 14.14.14.14 vrf default severity info
logging ::14:14:14:14 vrf default severity critical
logging source-interface Loopback0
logging hostnameprefix XRv1
```

```
RP/0/0/CPU0:XRv1#show logging
Syslog logging: enabled (0 messages dropped, 0 flushes, 0 overruns)
  Console logging: level debugging, 525 messages logged
  Monitor logging: level debugging, 274 messages logged
  Trap logging: level notice, 2 messages logged
  Logging to 14.14.14.14, 2 message lines logged
  Logging to ::14:14:14:14, 0 message lines logged
  Buffer logging: level debugging, 799 messages logged
[snip]
```

Different severity levels of logs can be sent to different output devices. Below are output devices:

- a. **Console:** writes to the console, which is self explanatory. VTY users will not see it.
- b. **Monitor:** writes to TTY sessions, not enabled by default, users use “terminal monitor” to enable.
- c. **Trap:** writes to the configured syslog servers (not related to SNMP traps).
- d. **Buffered:** writes to the local log, in memory, which is non persistent. File logging is available on Catalyst platforms and possibly a few others. The buffer is circular and it has limited size.
- e. **History:** When syslog messages are generated, “history” will wrap them in SNMP traps and send to SNMP servers. There is a history table size that determines how many logs can be stored. This was the original syslog/SNMP-trap interworking operation which has since changed.
- f. **SNMP-trap:** A new command that details when traps are sent. The history table stores the most recent entries and if this command is set, determines when to send SNMP traps to the syslog servers.

The history and SNMP-trap options require some additional explanation. Enabling logging history created a history table of length 1, which holds only the most recent syslog entry. Every syslog has an entry number, the most recent being 95. When the table is full, this number minus the maximum table size will equal the number of entries flushed ($95 - 1 = 94$). Essentially, the previous 94 entries were flushed because the table only holds 1 entry.

```
! CSR5
logging history
logging history informational
```

```
R5#show logging history
Syslog History Table: 1 maximum table entries,
saving level informational or higher
174 messages ignored, 0 dropped, 0 recursion drops
94 table entries flushed
SNMP notifications not enabled
  entry number 95 : SYS-5-CONFIG_I
    Configured from console by console
    timestamp: 91546549
```

Increasing the table to 5 entries allows 5 logs to be stored. To create 5 syslog messages, I just went into configuration mode and back out several times. Focusing on SNMP briefly, we see that “SNMP notifications are not enabled”. We will rectify this next.

```
! CSR5
logging history size 5

R5#show logging history
Syslog History Table: 5 maximum table entries,
saving level informational or higher
174 messages ignored, 0 dropped, 0 recursion drops
95 table entries flushed
SNMP notifications not enabled
  entry number 96 : SYS-5-CONFIG_I
    Configured from console by console
    timestamp: 91566675
  entry number 97 : SYS-5-CONFIG_I
    Configured from console by console
    timestamp: 91574959
  entry number 98 : SYS-5-CONFIG_I
    Configured from console by console
    timestamp: 91576225
  entry number 99 : SYS-5-CONFIG_I
    Configured from console by console
    timestamp: 91576536
  entry number 100 : SYS-5-CONFIG_I
    Configured from console by console
    timestamp: 91576860
```

To enable SNMP traps to carry syslog messages, these two things must happen. First, logging for SNMP-traps must be enabled, and SNMP traps for syslog must also be enabled in global configuration. Once both of these changes are made, the output indicates that SNMP notifications are now enabled.

```
! CSR5
logging snmp-trap emergencies
logging snmp-trap alerts
logging snmp-trap critical
logging snmp-trap errors
logging snmp-trap warnings
snmp-server enable traps syslog

R5#show logging history | include SNMP
SNMP notifications enabled, 0 notifications sent
```

XR also supports logging history. The configuration is almost identical to XE. I also enable logging at the notification level (5), rather than the informational level (6) just for variety.


```
! XRv1
logging history size 5
logging history notifications
```

```
RP/0/0/CPU0:XRv1#show logging history
Syslog History Table: '5' maximum table entries
saving level 'notice' or higher
0 messages ignored, 0 dropped, 44 table entries flushed
SNMP notifications disabled
```

```
RP/0/0/CPU0:: ntpd[236]: %IP-IP_NTP-5-LP_CONN_RECOVERED : At least a low
priority NTP peer connection was recovered - Stratum 15->8.
[snip, more logs]
```

There is no “logging snmp-trap” command in XR, so to enable SNMP trap logging inside syslog, just enable “snmp-server traps syslog” in global configuration.

```
! XRv1
snmp-server traps syslog
```

```
RP/0/0/CPU0:XRv1#show logging history
Syslog History Table: '5' maximum table entries
saving level 'notice' or higher
0 messages ignored, 0 dropped, 44 table entries flushed
SNMP notifications enabled, 0 notifications sent
[snip]
```

Earlier I mentioned “file” logging was no longer supported, because the feature has been expanded into the “persistent” logging feature. This can write to any URL and has several performance-related parameters that can be tuned. For example, logs can be written immediately or grouped together in batches, the maximum size of each log file can be set, and the total amount of disk space used can be limited. All of these options must be entered at once in one giant command, but it is straightforward to comprehend. On CSR5, I enable this feature; keep in mind the URL destination is a directory designed to contain the log files, not a file itself, so name it accordingly.

```
! CSR5
logging persistent url bootflash:/syslogs size 16384 filesize 16384 immediate
notify
```

```
R5#show logging | include Persist
Persistent logging: enabled, url bootflash:/syslogs, disk space 16384
bytes, file size 16384 bytes, batch size 4096 bytes, threshold capacity 0
, immediate , notify
```

When you use the “notify” keyword as I did, the system prints a syslog telling you that someone issued the “show logging” command. The syslog message below is a result of issuing the “show logging”

command shown above. The “immediate” options writes to the URL immediately, which is nice in a test environment or if syslog messages must be assessed quickly. This comes at a performance penalty since every syslog message will invoke a disk write. Grouping these disk writes is a better option in production environments where syslog messages may be generated regularly.

```
! CSR5
%SYS-6-SHOW_LOGGING: User Unknown has activated the show logging command.
```

We can check the contents of the log directly to see what syslog has added so far.

```
R5#dir bootflash: | include syslogs
275809  drwx                4096  Jul 26 2015 14:16:17 +00:00  syslogs

R5#cd bootflash:syslogs

R5#dir
Directory of bootflash:/syslogs/

275810  -rw-                673   Jul 26 2015 14:21:20 +00:00  log_20150726-142120
275811  -rw-                146   Jul 26 2015 14:21:26 +00:00  log_20150726-142122

7835619328 bytes total (6013329408 bytes free)

R5#more bootflash:log_20150726-142120
%SYS-5-LOG_CONFIG_CHANGE: Persistent logging: enabled, url
bootflash:/syslogs, disk space 16384 bytes, file size 16384 bytes, batch size
4096 bytes
%SYS-5-CONFIG_I: Configured from console by console
%SYS-5-CONFIG_I: Configured from console by console
%SYS-6-SHOW_LOGGING: User Unknown has activated the show logging command.
%SYS-6-SHOW_LOGGING: User Unknown has activated the show logging command.
%SYS-6-SHOW_LOGGING: User Unknown has activated the show logging command.
%SYS-5-CONFIG_I: Configured from console by console

R5#more bootflash:log_20150726-142122
%SYS-5-CONFIG_I: Configured from console by console
%SYS-5-CONFIG_I: Configured from console by console
```

The equivalent of persistent logging in XR is “archive” logging. Many of the parameters are used to tune size/time parameters, and the URL equivalent just points to a disk location. Beware that the boot device cannot be used as a log destination. XRv only comes with one disk by default, so I had to power off the VM, add a new disk, and then enable logging on disk1. If you try to commit the configuration when disk1 is not available, you get an error also. Lastly, if you try to commit with no device specified at all, you get an error. All three are shown below.

```
! XRv1
logging archive
```

```

device disk0
!!% Invalid argument: Boot device /disk0: cannot be used as archive
destination

! XRv1
logging archive
device disk1
!!% No such file or directory: Failed to get status of the archive device.
Error:No such file or directory

! XRv1
logging archive
severity informational
!!% Invalid argument: Attempting to configure other parameters when mandatory
device configuration is absent

```

Below is a valid configuration. Unfortunately, even after adding the new disk to XRv, it did not recognize the second disk and the configuration was rejected. Just specifying “harddisk” seemed to work, even though it picked disk0.

```

! XRv1
logging archive
device harddisk
severity informational
file-size 2
frequency daily
archive-size 4
archive-length 2

```

XR does not appear to have an “immediate” option, so we are left waiting for the logs to be written to disk. We can generate logs using the exec “logmsg” command. The file has been built in the hd_staging subdirectory but is 0 bytes in length.

```

RP/0/0/CPU0:XRv1#dir | include syslog
63529      drwx  1024      Sun Jul 26 14:30:01 2015  syslog

RP/0/0/CPU0:XRv1#logmsg HELLO
RP/0/0/CPU0: logger[65736]: %OS-SYSLOG-6-LOG_INFO : HELLO

RP/0/0/CPU0:XRv1#dir disk0:/syslog/hd_flush
Directory of disk0:/syslog/hd_flush
No files in directory
2377105408 bytes total (1865272320 bytes free)

```

Inside the hd_staging directory, we find the empty file discussed above.

```

RP/0/0/CPU0:XRv1#dir disk0:/syslog/hd_staging
Directory of disk0:/syslog/hd_staging

```

```
61550      -rw-  0      Sun Jul 26 14:32:57 2015  daily.20150726.26
```

```
2377105408 bytes total (1865272320 bytes free)
```

```
RP/0/0/CPU0:XRv1#more disk0:/syslog/hd_staging/daily.20150726.26
```

```
[no output]
```

Logs can also be selectively filtered based on facility, severity, message contents, and other parameters. This feature appears to be broken in XE (perhaps CSR1000v-related) but works on 12.4T. The snippet and verifications are below, but in XE the discriminator doesn't actually work. In this example, I don't want to see the "%SYS-5-CONFIG_I: Configured from console by USER" message on the console line.

```
! CSR5
logging discriminator SYSCFG severity drops 5 facility drops SYS
logging console discriminator SYSCFG
```

```
R5#show logging | section SYSCFG|Console
SYSCFG      severity group drops      5
            facility      drops      SYS
```

```
Console logging: level debugging, 367 messages logged, xml disabled,
                  filtering disabled, discriminator(SYSCFG),
                  0 messages rate-limited, 0 messages dropped-by-MD
```

A quick GNS3 test using 12.4T shows this exact configuration actually works.

```
R1#show logging | section SYSCFG|Console
SYSCFG      severity group drops      5
            facility      drops      SYS
```

```
Console logging: level debugging, 11 messages logged, xml disabled,
                  filtering disabled, discriminator(SYSCFG),
                  0 messages rate-limited, 7 messages dropped-by-MD
```

XR also supports this feature except it is called "logging suppress". You have to memorize a ton of information to know what exactly you are configuring, so I just took an example from Cisco. This will hide the syslog that gets generated when a commit is successful. This doesn't appear to work, though, and Cisco doesn't go into much detail. Cisco also spelled "MGBL" incorrectly in their example. The configuration is shown below and left on XRv1 for reference.

```
! XRv1
logging suppress rule MGBLHIDE
alarm MGBL COMMIT SUCCEDED

logging suppress apply rule MGBLHIDE
all-of-router
```

15. **NTP:** NTP keeps devices time synchronized to help with log analysis. NTP itself must be secured also. There is an entire section dedicated to NTP. This lab takes many of those security techniques, including authentication and access-groups, and applied them to all devices in the SP network as applicable. A quick verification to see if authentication and access-control works is shown here, from CSR6's perspective. Recall that the peer ACL determines who is allowed to update the local clock (should only be authorized servers), and the serve ACL determines to whom you may serve time (clients should never serve time). CSR6 is an NTP client and its configuration is shown below. We can see that packets from CSR10, the NTP master, are permitted by the ACL and authenticated by key 85.

```
! CSR6
ip access-list standard ACL_NTP_PEER
  permit 10.10.10.10
ip access-list standard ACL_NTP_SERVE
  deny any

ntp logging
ntp authentication-key 85 md5 0525323F1E6D7B3D31 7
ntp authenticate
ntp trusted-key 85
ntp source Loopback0
ntp access-group peer ACL_NTP_PEER
ntp access-group serve ACL_NTP_SERVE
ntp server 10.10.10.10 key 85

R6#show ntp associations detail | include master
10.10.10.10 configured, ipv4, authenticated, our_master, sane,valid,stratum 8

R6#show access-list ACL_NTP_PEER
Standard IP access list ACL_NTP_PEER
  10 permit 10.10.10.10 (11 matches)

R6#show access-lists ACL_NTP_SERVE
Standard IP access list ACL_NTP_SERVE
  10 deny any
```

16. **SNMP:** SNMP is used to manage network devices. It should not use default community strings in SNMPv1 and SNMPv2c, and should preferably use SNMPv3 for additional security. ACLs should be put in place to limit read-only and read-write capabilities. In this example, we configure CSR6 with several SNMP features, including version 2c (community based) and version 3 (auth/priv based). The communities are basically passwords used to read or write to a device using SNMP. In this example, I define two communities, one with read-only access, and one with read-write. The access to each is further restricted by ACLs, one for IPv6 and one for IPv4. Generally speaking, the RW ACL should be more restrictive than the RO one.

```
! CSR6
ip access-list standard ACL_SNMP_RO_IPV4
```

```

permit 85.0.0.0 0.255.255.255

ip access-list standard ACL_SNMP_RW_IPV4
permit 85.100.46.57

ipv6 access-list ACL_SNMP_RO_IPV6
permit ipv6 ::85:0:0:0/80 any

ipv6 access-list ACL_SNMP_RW_IPV6
permit ipv6 host ::85:100:F:F any

snmp-server community R6_RO_COMM RO ipv6 ACL_SNMP_RO_IPV6 ACL_SNMP_RO_IPV4
snmp-server community R6_RW_COMM RW ipv6 ACL_SNMP_RW_IPV6 ACL_SNMP_RW_IPV4

```

A more secure way of configuring SNMP is to use version 3. The configuration is much more involved as there is a three-step procedure. First, views should be configured which are essentially permission sets. In my example, I configure two views, one with full permissions and one for a BGP administrator who only needs a handful of rights (similar to RBAC in concept). The “show snmp view” command is very basic and takes no arguments, other than showing you the configured views.

```

! CSR6
snmp-server view ALL iso included
snmp-server view BGP_ADMIN ip included
snmp-server view BGP_ADMIN bgp included
snmp-server view BGP_ADMIN ipForward included

R6#show snmp view | include BGP|ALL
ALL iso - included nonvolatile active
BGP_ADMIN ip - included nonvolatile active
BGP_ADMIN bgp - included nonvolatile active
BGP_ADMIN ipForward - included nonvolatile active

```

The views are bound to groups, which defined the manner in which users authenticate. We create an ADMINS group that requires both authentication and encryption, and has RW permissions. I re-use the ACLs from earlier because, in a migration from v2c to v3, normally the same users would have the same rights they always did. The BGP_GUYS group is less secure, requiring authentication only, but only granting RO rights. The show command takes no arguments and summarizes the group (v3) and community string (v1/v2c) configuration. The “v1default” view is applied automatically to v2c communities and gives full access.

```

! CSR6
snmp-server group ADMINS v3 priv read ALL write ALL access ipv6
ACL_SNMP_RW_IPV6 ACL_SNMP_RW_IPV4

snmp-server group BGP_GUYS v3 auth read BGP_ADMIN access ipv6
ACL_SNMP_RO_IPV6 ACL_SNMP_RO_IPV4

```

```
R6#show snmp group
[snip]
```

```
groupname: ADMINIS                      security model:v3 priv
contextname: <no context specified>      storage-type: nonvolatile
readview : ALL                          writeview: ALL
notifyview: <no notifyview specified>
row status: active                      access-list: ACL_SNMP_RW_IPV4   ipv6 access-list:
ACL_SNMP_RW_IPV6
```

```
groupname: BGP_GUYS                      security model:v3 auth
contextname: <no context specified>      storage-type: nonvolatile
readview : BGP_ADMIN                    writeview: <no writeview
specified>
notifyview: <no notifyview specified>
row status: active                      access-list: ACL_SNMP_RO_IPV4   ipv6 access-list:
ACL_SNMP_RO_IPV6
```

```
groupname: R6_RO_COMM                    security model:v1
contextname: <no context specified>      storage-type: permanent
readview : vldefault                    writeview: <no writeview
specified>
notifyview: <no notifyview specified>
row status: active                      access-list: ACL_SNMP_RO_IPV4   ipv6 access-list:
ACL_SNMP_RO_IPV6
```

```
groupname: R6_RO_COMM                    security model:v2c
[snip]
```

```
groupname: R6_RW_COMM                    security model:v1
contextname: <no context specified>      storage-type: nonvolatile
readview : vldefault                    writeview: <no writeview
specified>
notifyview: <no notifyview specified>
row status: active
```

```
groupname: R6_RW_COMM                    security model:v2c
contextname: <no context specified>      storage-type: permanent
readview : vldefault                    writeview: vldefault
[snip]
```

It's interesting that we see both v1 and v2c groups for our communities above (not related to v3). SNMP configuration in these platforms is somewhat unintuitive; we have to specify the community as a "group" and remove SNMPv1 as a supported. Even worse, these commands do not show up in the configuration, and must be entered manually. Also notice above that v1 permissions under the RW group did not grant write permission. There are only two fundamental differences between v1 and v2c: introduction of the inform message (discussed in the SNMP traps section) and support for 64-bit

counters. SNMPv2c isn't any more secure than SNMPv1, but disabling v1 support at least minimizes the number of servers that could poll the device using that old version.

```
! CSR6, not retained in configuration
no snmp-server group R6_RO_COMM v1
no snmp-server group R6_RW_COMM v1
```

```
R6#show snmp group | include COMM
groupname: R6_RO_COMM                security model: v2c
groupname: R6_RW_COMM                security model: v2c
```

The SNMPv3 users are defined last and are assigned to groups. The users never show up in the running configuration on XE, so I've pasted the commands for reference; they are very long commands. You can add the ACLs here for additional security so if your credentials are compromised, intruders cannot log in from unauthorized IP addresses. Unlike the other show commands, you can specify the user here.

```
! CSR6
snmp-server user NRUSO ADMINS v3 auth sha AUTHPASS priv aes 256 PRIVPASS
access ipv6 ACL_SNMP_RW_IPV6 ACL_SNMP_RW_IPV4

snmp-server user JLI BGP_GUYS v3 auth sha AUTHPASS access ipv6
ACL_SNMP_RO_IPV6 ACL_SNMP_RO_IPV4

R6#show snmp user
User name: JLI
Engine ID: 800000090300005056A96CBE
storage-type: nonvolatile          active access-list: ACL_SNMP_RO_IPV4   IPv6
access-list: ACL_SNMP_RO_IPV6
Authentication Protocol: SHA
Privacy Protocol: None
Group-name: BGP_GUYS

User name: NRUSO
Engine ID: 800000090300005056A96CBE
storage-type: nonvolatile          active access-list: ACL_SNMP_RW_IPV4   IPv6
access-list: ACL_SNMP_RW_IPV6
Authentication Protocol: SHA
Privacy Protocol: AES256
Group-name: ADMINS
```

I've added several other SNMP features, some of which indirectly support security. Setting the source interface to a loopback, and also establishing some QoS value, can be valuable to ensure HA for your SNMP-managed devices. I also configured it to silently discard packets from unknown users. The location, contact, and chassis-ID are just text strings used to identify the device. The last command ensures that the interface ID's assigned persist across a reboot.


```

! CSR6
snmp-server trap-source Loopback0
snmp-server source-interface informs Loopback0
snmp-server drop unknown-user
snmp-server location R6
snmp-server ip dscp 16
snmp-server contact nrusso@r6.net
snmp-server chassis-id R6_CHASSIS
snmp ifmib ifindex persist

```

As expected, the SNMP configuration in XR is similar to XE. XR supports IPv6 DSCP markings, which I set, but does not appear to support both IPv4 and IPv6 ACLs concurrently. You can pick one or the other, so I added IPv4 only. I won't walk through the configuration since it is nearly identical to what was reviewed earlier. Unlike XE, XR displays the users in the configuration with encrypted credentials.

```

! XRv1
ipv4 access-list ACL_SNMP_RO_IPV4
 10 permit ipv4 85.0.0.0/8 any

ipv4 access-list ACL_SNMP_RW_IPV4
 10 permit ipv4 host 85.100.46.57 any

snmp-server ipv4 dscp cs2
snmp-server ipv6 dscp cs2
snmp-server user JLI BGP_GUYS v3 auth sha encrypted 106F3C2D2D2733383F
ACL_SNMP_RO_IPV4
snmp-server user NRUSO ADMINS v3 auth sha encrypted 0800797A2129242421 priv
aes 256 encrypted 097C7C202F35362138 ACL_SNMP_RW_IPV4

snmp-server view ALL iso included
snmp-server view BGP_ADMIN ip included
snmp-server view BGP_ADMIN bgp included

snmp-server community XRV11_RO_COMM RO ACL_SNMP_RO_IPV4
snmp-server community XRV11_RW_COMM RW ACL_SNMP_RW_IPV4

snmp-server group ADMINS v3 priv read ALL write ALL ACL_SNMP_RW_IPV4
snmp-server group BGP_GUYS v3 auth read BGP_ADMIN ACL_SNMP_RO_IPV4
snmp-server chassis-id XRV11_CHASSIS
snmp-server contact nrusso@xrv11.net
snmp-server location XRV11
snmp-server trap-source Loopback0
snmp-server ifindex persist

```

Last, we execute some basic show commands to verify the XRv1 configuration. I haven't found a way to disable SNMPv1 on XR. Security model "usm" implies SNMPv3.

```

RP/0/0/CPU0:XRv1#show snmp view | utility egrep 'BGP|ALL'

```

ALL iso - included nonVolatile active
BGP_ADMIN ip - included nonVolatile active
BGP_ADMIN bgp - included nonVolatile active

RP/0/0/CPU0:XRv1#show snmp group

groupname: ADMINs security model: usm
readview : ALL writeview: ALL
notifyview:
row status: active

groupname: BGP_GUYS security model: usm
readview : BGP_ADMIN writeview:
[snip]

groupname: XRV11_RO_COMM security model: snmpv1
readview : vldefault writeview:
[snip]

groupname: XRV11_RO_COMM security model: snmpv2c
[snip]

groupname: XRV11_RW_COMM security model: snmpv1
readview : vldefault writeview: vldefault
[snip]

groupname: XRV11_RW_COMM security model: snmpv2c
[snip]

RP/0/0/CPU0:XRv1#show snmp user

User Name: JLI
EngineId: localSnmpID
storage-type: nonvolatile active

User Name: NRUSO
EngineId: localSnmpID
storage-type: nonvolatile active

17. **Out-of-band management:** OOBM ensures that if the production network is compromised, you can still manage your devices safely. OOBM implementations can use terminal servers with reverse telnet or an OOBM IP network. In my virtual lab, there is a virtual serial port concentrator (vSPC) running on a Linux VM that connects to each console line of the routers. That is how I manage it, so any data-plane issues should not affect one's ability to manage my device (unless the issue is so serious it hogs all the resources or crashes the router). In real life, even inexpensive layer 2 switches are sufficient for medium-sized OOBM networks since there is very little traffic on it, and it is expected to be stable/redundant. Larger networks may use stacked switches or even routers for OOBM.

49.3 Data plane security best practices

1. **Antispoof ACL:** This is the classic method of ensuring only traffic from legitimate sources can enter a network. Modern networks typically use unicast RPF to handle this automatically. I've coupled this task with the following two tasks since only one ACL can be applied to an interface at one time.
2. **Anti-bogon and RFC1918 ACLs:** These reject traffic that should never be seen on the Internet, which includes private addressing and several other ranges (some are public, but unallocated). This is combined with the antispoof and infrastructure ACL testing.
3. **Infrastructure ACLs (iACL):** These are used to protect the infrastructure networks (links between routers, MPLS LSP loopback endpoints, management LANs, etc). There is no reason why external traffic should ever be destined to (or sourced from) these networks. Normally, unicast RPF (uRPF) is used to protect infrastructure spoofing, which is discussed in detail later. The only exception would be if the transit links are not advertised into IGP, then the ASBRs would never have a route to compare against. I've configured the iACL solution on XRv1 and CSR4 only. After combining the antispoof, RFC1918/bogon, and iACL, the result is shown below. Notice that it closely resembles the bogon route-filter we used earlier, except this is added data-plane protection. I use object-groups to help modularize the configuration, but it makes readability more difficult (more scrolling through the configuration). The idea is that I want to block traffic coming from or going to any combination of these "bad networks". Included in this list, in order to prevent spoofing and protect infrastructure, is the 85.0.0.0/8 network which is used by the SP. Using object-groups this way creates enormous ACLs behind the scenes, which is N^2 entries in this case (N is the number of entries in the object-group). Notice that the ACL must permit critical control-plane traffic; because the BGP session is using an RFC1918 transit link, we need a match higher in the ACL processing sequence to permit that traffic. First, we examine XE configuration on CSR4. These object groups do not support IPv6 at this time.

```
! CSR4
object-group network OBJGRP_BAD_NETS
 0.0.0.0 255.0.0.0
10.0.0.0 255.0.0.0
100.64.0.0 255.192.0.0
127.0.0.0 255.0.0.0
169.254.0.0 255.255.0.0
172.16.0.0 255.240.0.0
192.0.0.0 255.255.255.0
192.0.2.0 255.255.255.0
192.168.0.0 255.255.0.0
198.18.0.0 255.254.0.0
198.51.100.0 255.255.255.0
203.0.113.0 255.255.255.0
224.0.0.0 224.0.0.0
85.0.0.0 255.0.0.0

ip access-list extended ACL_PEER_SP_IN
 permit tcp host 10.1.4.1 host 10.1.4.4 eq bgp
 permit tcp host 10.1.4.1 eq bgp host 10.1.4.4
```

```
deny ip object-group OBJGRP_BAD_NETS object-group OBJGRP_BAD_NETS
permit ip any any
```

```
interface GigabitEthernet2.514
 ip access-group ACL_PEER_SP_IN in
```

A quick test on CSR1 shows that the list is working. We add a route to some RFC1918 addressing and send packets to it. We will also flap the eBGP session with CSR1 to generate legitimate traffic. The BGP traffic is matched by the ACL and passed, while the 50 bogus packets are matched and dropped.

```
! CSR1
ip route 10.0.0.0 255.0.0.0 10.1.4.4
```

```
R1#ping 10.0.0.5 timeout 0 repeat 50
Type escape sequence to abort.
Sending 50, 100-byte ICMP Echos to 10.0.0.5, timeout is 0 seconds:
```

```
.....
Success rate is 0 percent (0/50)
```

```
R4#clear ip bgp 1.1
```

```
R4#show access-l ACL_PEER_SP_IN
Extended IP access list ACL_PEER_SP_IN
 10 permit tcp host 10.1.4.1 host 10.1.4.4 eq bgp
 20 permit tcp host 10.1.4.1 eq bgp host 10.1.4.4 (13 matches)
 30 deny ip object-group OBJGRP_BAD_NETS object-group OBJGRP_BAD_NETS (50
matches)
 40 permit ip any any
```

The configuration on XR is not much different, except that IPv6 is an option also. We configure this on XRv1 to protect the interface facing XRv3.

```
! XRv1
object-group network ipv4 OBJGRP_BAD_NETS
 0.0.0.0/8
 10.0.0.0/8
 85.0.0.0/8
 100.64.0.0/10
 127.0.0.0/8
 172.16.0.0/12
 198.18.0.0/15
 224.0.0.0/3
 169.254.0.0/16
 192.0.0.0/24
 192.0.2.0/24
 192.168.0.0/16
 198.51.100.0/24
 203.0.113.0/24
```

```

ipv4 access-list ACL_PEER_SP_IN
 10 permit tcp host 10.1.3.13 host 10.1.3.11 eq bgp
 20 permit tcp host 10.1.3.13 eq bgp host 10.1.3.11
 30 deny ipv4 net-group OBJGRP_BAD_NETS net-group OBJGRP_BAD_NETS
1000 permit ipv4 any any

interface GigabitEthernet0/0/0/0.513
  ipv4 access-group ACL_PEER_SP_IN ingress

```

On a hardware-based XR platform, this would have worked, but XRv does not support object-groups. I will delete the configuration from XRv1 and leave it here for reference. Instead, I will program a basic ACL just to protect against RFC1918 and infrastructure spoofing.

```

object-group network ipv4 OBJGRP_BAD_NETS
!!% 'obj_group' detected the 'warning' condition 'Object groups not supported
on this platform'

```

```

! XRv1
ipv4 access-list ACL_PEER_SP_IN
 10 permit tcp host 10.1.3.13 host 10.1.3.11 eq bgp
 20 permit tcp host 10.1.3.13 eq bgp host 10.1.3.11
 30 deny ipv4 10.0.0.0/8 any
 40 deny ipv4 172.16.0.0/12 any
 50 deny ipv4 192.168.0.0/16 any
 900 deny ipv4 85.0.0.0/8 any
 910 deny ipv4 any 85.0.0.0/8
1000 permit ipv4 any any

interface GigabitEthernet0/0/0/0.513
  ipv4 access-group ACL_PEER_SP_IN ingress

```

Like earlier, I create a static route on XRv13 to flood RFC1918 traffic in AS 85, and I clear BGP on XRv11 to ensure the BGP session is permitted by the ACL. So far, XE and XR appear to have very similar ACL implementations, which is expected given the fundamental nature of the feature.

```

! XRv3
router static
  address-family ipv4 unicast
    10.0.0.0/8 10.1.3.11

RP/0/0/CPU0:XRv3#ping 10.0.0.5 count 50 timeout 0

RP/0/0/CPU0:XRv1#clear bgp ipv4 unicast as 9.13 soft

RP/0/0/CPU0:XRv1#show access-lists ACL_PEER_SP_IN
ipv4 access-list ACL_PEER_SP_IN
 10 permit tcp host 10.1.3.13 host 10.1.3.11 eq bgp

```

```

20 permit tcp host 10.1.3.13 eq bgp host 10.1.3.11 (7 matches)
30 deny ipv4 10.0.0.0 0.255.255.255 any (50 matches)
40 deny ipv4 172.16.0.0 0.15.255.255 any
50 deny ipv4 192.168.0.0 0.0.255.255 any
900 deny ipv4 85.0.0.0 0.255.255.255 any
910 deny ipv4 any 85.0.0.0 0.255.255.255
1000 permit ipv4 any any

```

4. **Classification ACL:** The ACL ultimately permits all traffic, but by having specific permit statements, you can use the ACE counters to see what traffic is present in your network as a quick troubleshooting technique. External packet capturing tools (Wireshark), Netflow, and EPC would be other options. We will temporarily configure this on CSR8 and XRv2 to measure traffic coming in. I will select some common traffic types to try and narrow down what kind of flows are causing issues.

```

! CSR8
ip access-list extended ACL_PEER_SP_CLASSIFY_IN
 permit icmp any any echo
 permit icmp any any echo-reply
 permit icmp any any unreachable
 permit tcp any any
 permit udp any any
 permit ip any any

interface GigabitEthernet2.518
 ip access-group ACL_PEER_SP_CLASSIFY_IN in

```

We may need to flap BGP so that the routes via CSR8 are oldest from the perspective of AS 1.1. Assuming they are, any eBGP path originating in AS 85 should be reachable via CSR8. We will flood traffic to an arbitrary destination from CSR1. The ACL doesn't block it, so it's possible that the pings may succeed, but the purpose is to measure anomalies. The handful of TCP packets are most likely from the existing eBGP session. The ACL shows 50 new ICMP echo requests. At this point, a more detailed inspection is warranted; the purpose of this feature is to find general anomalies.

```

R1#show ip cef 85.19.16.0
85.19.16.0/20
  nexthop 10.1.8.8 GigabitEthernet2.518

```

```

R1#ping 85.19.16.254 repeat 50 timeout 0

```

```

R8#show access-lists ACL_PEER_SP_CLASSIFY_IN
Extended IP access list ACL_PEER_SP_CLASSIFY_IN
 10 permit icmp any any echo (50 matches)
 20 permit icmp any any echo-reply
 30 permit icmp any any unreachable
 40 permit tcp any any (9 matches)
 50 permit udp any any
1000 permit ip any any

```

The XR configuration is almost identical. We configure the same ACL in XRv2 facing CSR9.

```
! XRv2
ipv4 access-list ACL_PEER_SP_CLASSIFY_IN
 10 permit icmp any any echo
 20 permit icmp any any echo-reply
 30 permit icmp any any unreachable
 40 permit tcp any any
 50 permit udp any any
1000 permit ipv4 any any

interface GigabitEthernet0/0/0/0.529
 ipv4 access-group ACL_PEER_SP_CLASSIFY_IN ingress
```

This time, we will test from CSR9 into the SP network. We may need to flap BGP so that CSR9 prefers paths via XRv2. We see 50 matches under the proper ACL entry, as expected.

```
R9#show ip cef 85.6.144.0/20
85.6.144.0/20
  nexthop 10.2.9.12 GigabitEthernet2.529
```

```
R9#ping 85.6.144.254 repeat 50 timeout 0
```

```
RP/0/0/CPU0:XRv2#show access-lists ACL_PEER_SP_CLASSIFY_IN
ipv4 access-list ACL_PEER_SP_CLASSIFY_IN
 10 permit icmp any any echo (50 matches)
 20 permit icmp any any echo-reply
 30 permit icmp any any unreachable
 40 permit tcp any any (7 matches)
 50 permit udp any any
1000 permit ipv4 any any
```

5. **Unicast RPF (uRPF):** This is a dynamic, intelligent, and CEF-path mechanism to drop spoofed traffic. If the source of a packet is not reachable via the interface on which the packet was received, the packet is dropped. Since CSR8 and XRv2 do not have any “hard” ACLs applied yet, we will use them for uRPF testing. uRPF operates in two modes: strict and loose. Strict means that a packet arriving on an interface must have the source reachable via that interface, just like multicast RPF. Loose means that a packet arriving on an interface is accepted if there is any valid route out of any interface except null0. This is important for remotely-triggered black holes (RTBH) seen later. We will begin by enabling strict mode on XRv2. Immediately after enabling the feature, we verify the configuration. uRPF has more options in XE than XR, along with better outputs, but we examine XRv2 first. Of note, the “debug ip/ipv6 cef drops rpf” do not work in the CSR1000v, possibly due to it being virtual. Many packet-level debugs relying on CEF do not work, so verification will be done with show commands.

```
! XRv2
```

```
interface GigabitEthernet0/0/0/0.529
  ipv4 verify unicast source reachable-via rx
```

```
RP/0/0/CPU0:XRv2#show ipv4 interface gig0/0/0/0.529 | include RPF
IP unicast RPF check is enabled
RPF mode strict
```

We can test XRv2 by creating an asymmetric routing condition. XRv2 is preferring an iBGP path to CSR1's BGP routes due to AS-path length. If packets sourced from CSR1's routes enter AS 85 from AS 9.13 into XRv2, they should be dropped. You may need to clear CSR9's BGP peering with CSR8 so that XRv2's paths are older, and thus preferred. Let's ensure CSR9 routes through XRv2 to reach the 85 networks, and also verify that XRv2 is routing through CSR4 or CSR8 to each AS 1.1

```
RP/0/0/CPU0:XRv2#show bgp ipv4 unicast 1.12.0.0/16 brief | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
*>i1.12.0.0/16      4.4.4.4          0      100      0 65537 ?
*                   10.2.9.9          0      100      0 589837 65537 ?
* i                 4.4.4.4          0      100      0 65537 ?
```

```
R9#show bgp ipv4 unicast 85.19.4.0/22 bestpath
BGP routing table entry for 85.19.4.0/22, version 35
Paths: (4 available, best #3, table default)
  Advertised to update-groups:
    1          2
  Refresh Epoch 1
  85
    10.2.9.12 from 10.2.9.12 (12.12.12.12)
      Origin incomplete, localpref 100, valid, external, best
      rx pathid: 0, tx pathid: 0x0
```

We temporarily direct CSR1 to route all BGP traffic through AS 9.13 using local preference. CSR1 would have normally preferred either CSR4 or CSR8 to reach AS 85 prefixes, but no longer.

```
! CSR1
route-map RM_LPREF_IN per 10
  set local-preference 200

router bgp 1.1
  address-family ipv4 un
  neighbor 10.1.9.9 route-map RM_LPREF_IN in
```

```
R1#show bgp ipv4 unicast 85.19.4.0/22 bestpath
BGP routing table entry for 85.19.4.0/22, version 260
Paths: (3 available, best #1, table default)
  Advertised to update-groups:
    1
  Refresh Epoch 2
```


9.13 85

```
10.1.9.9 from 10.1.9.9 (9.10.9.9)
  Origin incomplete, localpref 200, valid, external, best
  rx pathid: 0, tx pathid: 0x0
```

From CSR1, we send packets from 1.12.1.1 to 85.19.4.1, which arrive into AS 85 via XRv2, and should fail uRPF. Because uRPF requires CEF, the CEF process keeps track of the counters.

```
R1#ping 85.19.4.1 source 1.12.1.1 timeout 1 repeat 100
[snip]
Success rate is 0 percent (0/100)
```

```
RP/0/0/CPU0:XRv2#show cef ipv4 interface gig0/0/0/0.529 rpf-statistics
Unicast RPF drops 100
```

We will clear the statistics from XRv2, configure loose mode, verify the configuration, and re-test. Because XRv2 has a valid route to the previously-tested IPv4 prefix, we expect the packets to be passed.

```
RP/0/0/CPU0:XRv2#clear cef ipv4 interface gig0/0/0/0.529 rpf-statistics

! XRv2
interface GigabitEthernet0/0/0/0.529
  no ipv4 verify unicast source reachable-via rx
  ipv4 verify unicast source reachable-via any

RP/0/0/CPU0:XRv2#show ipv4 interface gig0/0/0/0.529 | include RPF
IP unicast RPF check is enabled
RPF mode loose
```

As expected, CSR1 is able to ping this remote destination despite the asymmetric routing. XRv2 does not drop these packets since it has a valid route to the destination.

```
R1#ping 85.19.4.1 source 1.12.1.1 timeout 1 repeat 100
[snip]
Success rate is 100 percent (100/100)

RP/0/0/CPU0:XRv2#show cef ipv4 interface gig0/0/0/0.529 rpf-statistics
Unicast RPF drops 0
```

We will run the same tests with IPv6 uRPF on XRv2 to validate that its behavior is identical to IPv4. We need to verify three things to ensure asymmetry: XRv2 routing through CSR4 or CSR8, CSR1 routing through CSR9, and CSR9 routing through XRv2 (in a clock-wise triangle). The same local-preference route-map applied inbound on CSR1 facing CSR9 has been re-used for IPv6 (configuration not shown).

```
RP/0/0/CPU0:XRv2#show bgp ipv6 unicast 2001:1c00:c::/48 brief | begin Network
  Network      Next Hop      Metric LocPrf Weight Path
*>i2001:1c00:c::/48 4.4.4.4      0      100      0 65537 ?
```

```
*                               fd00:10:2:9::9                               0 589837 65537 ?
```

```
R1#show bgp ipv6 unicast 2001:85:6:84::/62
```

```
[snip]
```

```
9.13 85
```

```
FD00:10:1:9::9 (FE80::9) from 10.1.9.9 (9.10.9.9)  
Origin incomplete, localpref 200, valid, external, best  
rx pathid: 0, tx pathid: 0x0
```

```
R9#show bgp ipv6 unicast 2001:85:6:84::/62
```

```
[snip]
```

```
85
```

```
FD00:10:2:9::12 (FE80::12) from FD00:10:2:9::12 (12.12.12.12)  
Origin incomplete, localpref 100, valid, external, best  
rx pathid: 0, tx pathid: 0x0
```

With the routing “correct”, we enable strict-mode uRPF on XRv2 for IPv6. The configuration and verification are identical to IPv4 uRPF.

```
! XRv2  
interface GigabitEthernet0/0/0/0.529  
  ipv6 verify unicast source reachable-via rx
```

```
RP/0/0/CPU0:XRv2#show ipv6 interface gig0/0/0/0.529 | include RPF  
IP unicast RPF check is enabled  
RPF mode strict
```

Using strict mode, we expect the packets sourced from CSR1 to fail, since they follow the path through AS 9.13, while XRv2 prefers to route iBGP through CSR4.

```
R1#ping 2001:85:6:84::1 source 2001:1C00:C::1 repeat 100 timeout 1  
[snip]  
Success rate is 0 percent (0/100)
```

```
RP/0/0/CPU0:XRv2#show cef ipv6 interface gig0/0/0/0.529 rpf-statistics  
Unicast RPF drops 100
```

Clearing the counters and changing to loose-mode, we see the traffic flows through without any drops. This is the expected behavior and mirrors the test conducted for IPv4.

```
RP/0/0/CPU0:XRv2#clear cef ipv6 interface gig0/0/0/0.529 rpf-statistics
```

```
! XRv2  
interface GigabitEthernet0/0/0/0.529  
  no ipv6 verify unicast source reachable-via rx  
  ipv6 verify unicast source reachable-via any
```

```
R1#ping 2001:85:6:84::1 source 2001:1C00:C::1 repeat 100 timeout 1
```

```
[snip]
```

```
Success rate is 100 percent (100/100)
```

```
RP/0/0/CPU0:XRv2#show cef ipv6 interface gig0/0/0/0.529 rpf-statistics
```

```
Unicast RPF drops 0
```

Next, we will test the uRPF features on CSR8. We begin with strict mode on both externally-facing interfaces (AS 1.1 and AS 9.13 links). After configuring the feature, as always, we must verify it. Fortunately, the output is more compact and we get more information from a single command, where XR required more work.

```
! CSR8
```

```
interface GigabitEthernet2.589
```

```
ip verify unicast source reachable-via rx
```

```
interface GigabitEthernet2.518
```

```
ip verify unicast source reachable-via rx
```

```
R8#show ip interface gig2.589 | begin IP_verify
```

```
IP verify source reachable-via RX
```

```
0 verification drops
```

```
0 suppressed verification drops
```

```
0 verification drop-rate
```

```
R8#show ip interface gig2.518 | begin IP_verify
```

```
IP verify source reachable-via RX
```

```
0 verification drops
```

```
0 suppressed verification drops
```

```
0 verification drop-rate
```

Leaving the local-preference 200 route-maps in place on CSR1 towards CSR9, CSR1 prefers AS 9.13 to reach all routes from AS 85. We can clear the BGP peer with XRv2 on CSR9 so that it selects CSR8 as the bestpath for its AS 85 prefixes. CSR8 still believes that CSR1 is the bestpath to reach all AS 1.1 originated prefixes (makes sense since it is a direct peer), yet traffic sourced from those routes will arrive from AS 9.13 where uRPF strict-mode is enabled. We will verify the BGP routing before continuing.

```
R1#show bgp ipv4 unicast 85.6.144.0 bestpath
```

```
BGP routing table entry for 85.6.144.0/20, version 431
```

```
Paths: (3 available, best #1, table default)
```

```
Advertised to update-groups:
```

```
2
```

```
Refresh Epoch 2
```

```
9.13 85
```

```
10.1.9.9 from 10.1.9.9 (9.10.9.9)
```

```
Origin incomplete, localpref 200, valid, external, best
```

```
rx pathid: 0, tx pathid: 0x0
```

```

R9#show bgp ipv4 unicast 85.6.144.0 bestpath
BGP routing table entry for 85.6.144.0/20, version 55
Paths: (3 available, best #2, table default)
  Advertised to update-groups:
    1          2
  Refresh Epoch 2
  85
    10.8.9.8 from 10.8.9.8 (8.8.8.8)
      Origin incomplete, localpref 100, valid, external, best
      rx pathid: 0, tx pathid: 0x0

```

```

R8#show bgp ipv4 unicast 1.15.0.0 bestpath
BGP routing table entry for 1.15.0.0/16, version 130
Paths: (4 available, best #4, table default)
  Advertised to update-groups:
    20
  Refresh Epoch 2
  65537
    10.1.8.1 from 10.1.8.1 (10.1.8.1)
      Origin incomplete, metric 0, localpref 100, valid, external, best
      rx pathid: 0, tx pathid: 0x0

```

When CSR1 goes to send traffic to AS 85, it will fail uRPF as it arrives on CSR8 from AS 9.13. This is consistent with XRV's uRPF behavior so far.

```

R1#ping 85.6.144.1 source 1.15.1.1 timeout 1 repeat 10
[snip]
Success rate is 0 percent (0/100)

```

```

R8#show ip interface gig2.589 | begin IP_verify
IP verify source reachable-via RX
100 verification drops
0 suppressed verification drops
0 verification drop-rate

```

Changing the interface facing CSR9 to loose-mode, packets arriving along this same path should be accepted. We will clear the counters for cleanliness. Interestingly, we see packets logged as suppressed verification drops. The documentation states that this counter is used for packets that failed uRPF, but were permitted manually by an ACL. I did not apply a "pardon" ACL (discussed later), so I assume this counter also shows packets permitted by loose-mode uRPF that would have failed under strict-mode uRPF conditions.

```

R8#clear ip interface gig2.589 stats

! CSR8
interface GigabitEthernet2.589
  no ip verify unicast source reachable-via rx

```

```
ip verify unicast source reachable-via any
```

```
R1#ping 85.6.144.1 source 1.15.1.1 timeout 1 repeat 100  
[snip]
```

```
Success rate is 100 percent (100/100)
```

```
R8#show ip interface gig2.589 | begin IP_verify  
IP verify source reachable-via ANY  
0 verification drops  
100 suppressed verification drops  
0 verification drop-rate
```

Like with XRv2, we will quickly test IPv6 uRPF on XE. We first need to configure the feature and verify it. Notice that the IPv6 counters differentiate between process and CEF matched packets.

```
! CSR8  
interface GigabitEthernet2.518  
ipv6 verify unicast source reachable-via rx
```

```
interface GigabitEthernet2.589  
ipv6 verify unicast source reachable-via rx
```

```
R8#show ipv6 interface GigabitEthernet2.518 | include verif  
IPv6 verify source reachable-via rx  
0 verification drop(s) (process), 0 (CEF)  
0 suppressed verification drop(s) (process), 0 (CEF)
```

```
R8#show ipv6 interface GigabitEthernet2.589 | include verif  
IPv6 verify source reachable-via rx  
0 verification drop(s) (process), 0 (CEF)  
0 suppressed verification drop(s) (process), 0 (CEF)
```

We quickly verify our BGP routing still forms an asymmetric triangle. CSR1 routes to AS 85 via CSR9, CSR9 routes to CSR8, and CSR8 routes back to AS 1.1 via CSR1. This asymmetry was introduced by the local-preference adjustment made on CSR1 facing CSR9 earlier in the test.

```
R1#show bgp ipv6 unicast 2001:85:6:88::/61  
[snip]  
9.13 85  
FD00:10:1:9::9 (FE80::9) from 10.1.9.9 (9.10.9.9)  
Origin incomplete, localpref 200, valid, external, best  
rx pathid: 0, tx pathid: 0x0
```

```
R9#show bgp ipv6 unicast 2001:85:6:88::/61  
[snip]  
85  
FD00:10:8:9::8 (FE80::8) from 10.8.9.8 (8.8.8.8)  
Origin incomplete, localpref 100, valid, external, best
```

```
rx pathid: 0, tx pathid: 0x0
```

```
R8#show bgp ipv6 unicast 2001:1C00:E::/48
```

```
[snip]
```

```
65537
```

```
FD00:10:1:8::1 from 10.1.8.1 (10.1.8.1)
```

```
Origin incomplete, metric 0, localpref 100, valid, external, best
```

```
mpls labels in/out 8025/nolabel
```

```
rx pathid: 0, tx pathid: 0x0
```

Now, we ping from CSR1 to CSR10 and record the result. All of the packets were uRPF failures dropped by IPv6 CEF. With uRPF operating in strict-mode, this is the expected behavior.

```
R1#ping 2001:85:6:88::1 source 2001:1C00:E::1 timeout 1 repeat 100
```

```
[snip]
```

```
Success rate is 0 percent (0/100)
```

```
R8#show ipv6 interface GigabitEthernet2.589 | include verif
```

```
IPv6 verify source reachable-via rx
```

```
0 verification drop(s) (process), 100 (CEF)
```

```
0 suppressed verification drop(s) (process), 0 (CEF)
```

Next, we will clear the counters (I didn't find a specific command, so I cleared all interface parameters) and enable loose-mode. Upon re-test, all packets succeed, but like IPv4, they are marked as suppressed verification drops as they would have failed the strict uRPF check, but passed only due to loose-mode.

```
R8#clear counters gig2
```

```
interface GigabitEthernet2.589
```

```
no ipv6 verify unicast source reachable-via rx
```

```
ipv6 verify unicast source reachable-via any
```

```
R1#ping 2001:85:6:88::1 source 2001:1C00:E::1 timeout 1 repeat 100
```

```
[snip]
```

```
Success rate is 100 percent (100/100)
```

```
R8#show ipv6 interface GigabitEthernet2.589 | include verif
```

```
IPv6 verify source reachable-via any
```

```
0 verification drop(s) (process), 0 (CEF)
```

```
0 suppressed verification drop(s) (process), 100 (CEF)
```

uRPF can also accept ACLs in XE only. The ACL logic can be thought of like an executive pardon. Despite commit a crime (spoofing), if the ACL permits the packet, it is allowed through anyway. That is why the XE counter calls this a "suppressed" verification, since the RPF drop-action was suppressed. If the ACL also denies it, the packet is dropped normally. Commonly, a deny-any ACL is used with uRPF with logging enabled as this is the only way to log drops from uRPF. In XR, there is no option to log individual packet drops. Only numbered, extended ACLs can be used with uRPF with the current version of XE.

I will demonstrate this using two methods spread across IPv4 and IPv6. The first and most obvious test is, given our still-asymmetric routing, to change CSR8 to do strict-mode uRPF. CSR1 will send packets to a specific network within AS 85 and normally CSR8 would drop it inbound from CSR9, but I will “pardon” spoofed traffic towards that network. This network might be a sinkhole (discussed more later) where the SP wants to attract potential attack traffic for analysis. When verifying this configuration, the ACL numbered is displayed in the show command for completeness.

```
! CSR8
access-list 108 permit ip any 85.6.144.0 0.0.15.255
access-list 108 deny ip any any log-input

interface GigabitEthernet2.589
 ip verify unicast source reachable-via rx 108

interface GigabitEthernet2.518
 ip verify unicast source reachable-via rx 108

R8#show ip interface GigabitEthernet2.589 | begin IP_verify
IP verify source reachable-via RX, ACL 108
 0 verification drops
 0 suppressed verification drops
 0 verification drop-rate
```

When pinging that specific destination from CSR1, the traffic succeeds, and we see matches against suppressed verification drops as expected. This is consistent with Cisco’s documentation; despite strict mode being enabled, this traffic was “pardoned” by a manual ACL. We also see ACL matches; only packets that fail uRPF are subject to the follow-on ACL matching, which is a performance enhancement compared to a traditional inbound ACL.

```
R1#ping 85.6.144.1 source 1.15.1.1 repeat 100 timeout 1
[snip]
Success rate is 100 percent (100/100), round-trip min/avg/max = 1/24/203 ms
```

```
R8#show ip interface GigabitEthernet2.589 | begin IP_verify
IP verify source reachable-via RX, ACL 108
 0 verification drops
100 suppressed verification drops
 0 verification drop-rate
```

```
R8#show access-lists 108
Extended IP access list 108
 10 permit ip any 85.6.144.0 0.0.15.255 (100 matches)
 20 deny ip any any log-input
```

Pinging any other AS 85 destination network, even from the same source, is rejected. We see a log message on CSR8 also. We check the counters to confirm all packets were dropped. When coupled with syslog exportation, logging is a good tool to identify spoofing attacks from a management perspective.

```
R1#ping 85.6.136.1 source 1.15.1.1 repeat 15 timeout 1
[snip]
Success rate is 0 percent (0/15)

%FMANFP-6-IPACCESSLOGDP: F0: fman_fp_image: list 108 denied icmp 1.15.1.1
GigabitEthernet2.589-> 85.6.136.1 (8/0), 1 packet

R8#show ip interface GigabitEthernet2.589 | begin IP_verify
IP verify source reachable-via RX, ACL 108
15 verification drops
100 suppressed verification drops
0 verification drop-rate

R8#show access-lists 108
Extended IP access list 108
10 permit ip any 85.6.144.0 0.0.15.255 (100 matches)
20 deny ip any any log-input (15 matches)
```

We will test IPv6 on CSR8 also, but will do it using a less obvious method. One caveat of loose-mode is that routes recursing to null0 are considered invalid, so if a source has no route or a null0 route, the packet is dropped. Assuming CSR8 wants to avoid sending traffic to and receiving it from certain networks, this is a good technique. With loose-mode on all external interfaces, it protects against spoofing from sources in the null0 route. Based on ordinary routing rules, it prevents the router from forwarding traffic destined to those networks also. In this example, two of CSR1's local networks have been blackholed to Null0. The IPv6 ACL permits traffic sourced from half of those networks (2001:1C00:A::/48) as long as it's destined to a real AS 85 route.

```
! CSR8
ipv6 route 2001:1C00:A::/48 null0
ipv6 route 2001:1C00:B::/48 null0

ipv6 access-list ACL_URPF_PERMIT_IPV6
permit ipv6 2001:1C00:A::/48 2001:85::/32
deny ipv6 any any log-input

interface GigabitEthernet2.589
ipv6 verify unicast source reachable-via any ACL_URPF_PERMIT_IPV6

interface GigabitEthernet2.518
ipv6 verify unicast source reachable-via any ACL_URPF_PERMIT_IPV6
```

We quickly verify that uRPF was configured properly, to include the static null0 routes.


```

R8#show ipv6 interface GigabitEthernet2.589 | include verif
IPv6 verify source reachable-via any, ACL ACL_URPF_PERMIT_IPV6
  0 verification drop(s) (process), 0 (CEF)
  0 suppressed verification drop(s) (process), 0 (CEF)

R8#show ipv6 route static | begin ^S
S   2001:1C00:A::/48 [1/0]
    via Null0, directly connected
S   2001:1C00:B::/48 [1/0]
    via Null0, directly connected

R8#show access-lists ACL_URPF_PERMIT_IPV6
IPv6 access list ACL_URPF_PERMIT_IPV6
    permit ipv6 2001:1C00:A::/48 2001:85::/32 sequence 10
    deny ipv6 any any log-input sequence 20

```

When CSR1 sources traffic from the “A” subnet, the packets succeed. The suppressed verification drop counter increments since these normally would have failed uRPF due to the null0 route, but have been “pardoned” by the ACL.

```

R1#ping 2001:85:19:8::1 source 2001:1C00:A::1 timeout 1 repeat 20
[snip]
Success rate is 100 percent (20/20),

```

```

R8#show ipv6 interface GigabitEthernet2.589 | include verif
IPv6 verify source reachable-via any, ACL ACL_URPF_PERMIT_IPV6
  0 verification drop(s) (process), 0 (CEF)
  0 suppressed verification drop(s) (process), 20 (CEF)

R8#show access-lists ACL_URPF_PERMIT_IPV6
IPv6 access list ACL_URPF_PERMIT_IPV6
    permit ipv6 2001:1C00:A::/48 2001:85::/32 (20 matches) sequence 10
    deny ipv6 any any log-input sequence 20

```

When CSR1 sources traffic from the “B” subnet, the packets are dropped and logged.

```

R1#ping 2001:85:19:8::1 source 2001:1C00:B::1 timeout 1 repeat 12
[snip]
Success rate is 0 percent (0/12)

```

```

! CSR8
%FMANFP-6-IPV6ACCESSLOGDP: F0: fman_fp_image: list ACL URPF PERMIT IPV6
denied icmpv6 2001:1c00:b::1 GigabitEthernet2.589-> 2001:85:19:8::1
(0/32768), 1 packet

```

```

R8#show ipv6 interface GigabitEthernet2.589 | include verif
IPv6 verify source reachable-via any, ACL ACL_URPF_PERMIT_IPV6
  0 verification drop(s) (process), 12 (CEF)

```

```
0 suppressed verification drop(s) (process), 20 (CEF)
```

```
R8#show access-lists ACL_URPF_PERMIT_IPV6
IPv6 access list ACL_URPF_PERMIT_IPV6
  permit ipv6 2001:1C00:A::/48 2001:85::/32 (20 matches) sequence 10
  deny ipv6 any any log-input (12 matches) sequence 20
```

The static black-hole routes will be removed before continuing (not shown). Next, we analyze the “l2-src” option on XE. This adds an additional check to ensure the source IP and MAC bindings are correct based on the ARP cache. This is only supported for IPv4 currently and only works with strict-mode. In many cases, switched networks will already be running something like DHCP snooping, IP source guard, and/or DAI, but on transit links between service providers, those features may not be appropriate or supported. We will modify CSR8 on the interface towards CSR1 for this test. The configuration verification does not indicate any special for this feature. Looking at CSR1’s IP/MAC binding, we record the proper value. We are not going to test this feature because it will take a lot of effort with a vSwitch that doesn’t support ethertype filtering to block ARP messages, etc.

```
! CSR8
interface GigabitEthernet2.518
 ip verify unicast source reachable-via rx l2-src 108
```

```
R8#show ip interface GigabitEthernet2.518 | begin IP_verify
IP verify source reachable-via RX, ACL 108
  0 verification drops
  0 suppressed verification drops
  0 verification drop-rate
```

```
R8#show ip arp 10.1.8.1
```

Protocol	Address	Age (min)	Hardware Addr	Type	Interface
Internet	10.1.8.1	210	0050.56a9.1aaa	ARPA	GigabitEth2.518

By default, the default-route cannot be used for the uRPF check. If the uRPF-enabled router only has a default route pointing upstream, all traffic will be dropped in both strict and loose modes. There is an option to allow the default route to be used, and it is supported in both XE and XR for both IPv4 and IPv6. CSR9 will originate an IPv4 default to CSR8 and an IPv6 default to XRv2 (configuration not shown). Then, CSR9 will attempt to send traffic into AS 85 from its transit link to XRv3. This is a unique-local IPv6 address but XRv2 and CSR8 do not have security to protect from this, by design. You may need to flap BGP on CSR9 so that it routes to AS 85 via XRv2.

```
R9#show bgp ipv6 unicast 2001:85:19:8::/61
[snip]
85
  FD00:10:2:9::12 (FE80::12) from FD00:10:2:9::12 (12.12.12.12)
    Origin incomplete, localpref 100, valid, external, best
    rx pathid: 0, tx pathid: 0x0
```

When CSR9 sends traffic, they fail uRPF arriving on XRv2, despite XRv2 having a default route.

```
R9#ping 2001:85:19:8::1 source FD00:10:3:9::9 timeout 1
[snip]
Success rate is 0 percent (0/5)
```

```
RP/0/0/CPU0:XRv2#show cef ipv6 interface gig0/0/0/0.529 rpf-statistics
Unicast RPF drops 5
```

```
RP/0/0/CPU0:XRv2#show cef ipv6 FD00:10:3:9::9
::/0, version 1724, proxy default, internal 0x5000011 0x0 (ptr 0xa13b8074)
[1], 0x0 (0xa1384050), 0xa20 (0xa152c4d8)
local adjacency fe80::9
Prefix Len 0, traffic index 0, precedence n/a, priority 4
via fe80::9, GigabitEthernet0/0/0/0.529, 7 dependencies, weight 0, class
0, bgp-ext [flags 0x6020]
```

We can instruct XRv2 to allow traffic so long as it matches the default-route. This makes uRPF less secure, but is required for some environments. We still have protection against reverse-spoofing attacks where an attacker uses a source address within our infrastructure. uRPF still honors the longest-match, so having a more specific IGP/BGP route internally will cause that traffic to fail as expected. When we retest connectivity from CSR1, we see it is successful.

```
! XRv2
interface GigabitEthernet0/0/0/0.529
ipv6 verify unicast source reachable-via any allow-default
```

```
RP/0/0/CPU0:XRv2#show ipv6 interface GigabitEthernet0/0/0/0.529 | include RPF
IP unicast RPF check is enabled
RPF mode loose, allow default route
```

```
RP/0/0/CPU0:XRv2#clear cef ipv6 interface gig0/0/0/0.529 rpf-statistics
```

```
R9#ping 2001:85:19:8::1 source FD00:10:3:9::9 timeout 1
[snip]
Success rate is 100 percent (5/5)
```

```
RP/0/0/CPU0:XRv2#show cef ipv6 interface gig0/0/0/0.529 rpf-statistics
Unicast RPF drops 0
```

We quickly test the same concept on CSR8. Assuming CSR9 routes through CSR8 to reach AS 85, packets matching the default-route for uRPF are dropped initially. We still have the ACL 108 to “pardon” certain packets and uRPF is operating in strict mode presently.

```
R9#show bgp ipv4 unicast 85.19.4.0/22
[snip]
85
```

```
10.8.9.8 from 10.8.9.8 (8.8.8.8)
  Origin incomplete, localpref 100, valid, external, best
  rx pathid: 0, tx pathid: 0x0
```

```
R9#ping 85.19.4.1 source 10.3.9.9 timeout 1
[snip]
Success rate is 0 percent (0/5)
```

As expected, CSR8 creates a log message and records these drops.

```
! CSR8
%FMANFP-6-IPACCESSLOGDP: F0: fman_fp_image: list 108 denied icmp 10.3.9.9
GigabitEthernet2.589-> 85.19.4.1 (8/0), 1 packet
```

```
R8#show ip interface gig2.589 | begin IP_verify
IP verify source reachable-via RX, ACL 108
  5 verification drops
  0 suppressed verification drops
  0 verification drop-rate
```

```
R8#show access-lists 108
Extended IP access list 108
  10 permit ip any 85.6.144.0 0.0.15.255
  20 deny ip any any log-input (5 matches)
```

We instruct CSR8 to allow the default-route to be used for uRPF, clear counters, and re-test. Everything works as expected. Notice that this does not count as a uRPF “failure”. In loose mode, we saw that if a packet arrives on the “incorrect” interface by strict-mode standards, it is allowed through but the suppressed verification drop counter increments. We are using strict-mode here, and the packet did indeed arrive on the proper interface, despite being from the default-route. uRPF does not log an issue.

```
! CSR8
interface gig2.589
  ip verify unicast source reachable-via rx allow-default 108
```

```
R8#clear counters gig2
```

```
R8#show ip interface gig2.589 | begin IP_verify
IP verify source reachable-via RX, allow default, ACL 108
  0 verification drops
  0 suppressed verification drops
  0 verification drop-rate
```

```
R9#ping 85.19.4.1 source 10.3.9.9 timeout 1
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/5/19 ms
```

```
R8#show ip interface gig2.589 | begin IP_verify
```

```
IP verify source reachable-via RX, allow default, ACL 108
0 verification drops
0 suppressed verification drops
0 verification drop-rate
```

The BGP default-route advertisements are removed from the configuration before continuing.

6. **Netflow:** There is an entire section on this topic, so it is discussed briefly here. Netflow captures transit traffic based on fields identified by the administrator. The traffic can then be exported to a collector for further analysis or simply viewed on the CLI as a troubleshooting tool. The feature does not appear to be supported on XRv at all since it is linecard-specific. In this example, both CSR5 and CSR6 have identical Flexible Netflow (FNF) configurations. On their interfaces facing the ASBRs, they capture 100% of the MPLS traffic, 50% of the IPv4 traffic, and 20% of the IPv6 traffic. I use a common NetFlow-v9 collector for all records in this example. Using aliases, an administrator can quickly view this information using commands “nfm”, “nf4”, and “nf6” respectively. I prefer the table format, but when the terminal window is narrow, the record format is best. For demonstration, I enabled packet/byte counter collects on the MPLS record. This can be used like a classification ACL, except more powerful and dynamic.

```
! CSR5 and CSR6
flow record FLOW_RECORD_IPV4
match ipv4 dscp
match ipv4 protocol
match ipv4 source address
match ipv4 destination address
match transport source-port
match transport destination-port
match interface input

flow record FLOW_RECORD_MPLS
match interface input
match mpls label 1 details
match mpls label 2 details
match mpls label 3 details
match mpls label 4 details
collect counter bytes
collect counter packets

flow record FLOW_RECORD_IPV6
match ipv6 dscp
match ipv6 protocol
match ipv6 source address
match ipv6 destination address
match transport source-port
match transport destination-port
match interface input

flow exporter FLOW_EXPORTER_COMMON
```

```

destination 10.10.10.10
source Loopback0
dscp 48
ttl 16

flow monitor FLOW_MONITOR_IPV4
exporter FLOW_EXPORTER_COMMON
record FLOW_RECORD_IPV4

flow monitor FLOW_MONITOR_IPV6
exporter FLOW_EXPORTER_COMMON
record FLOW_RECORD_IPV6

flow monitor FLOW_MONITOR_MPLS
exporter FLOW_EXPORTER_COMMON
record FLOW_RECORD_MPLS

sampler FLOW_SAMPLERS_IPV4
mode random 1 out-of 2

sampler FLOW_SAMPLERS_IPV6
mode random 1 out-of 5

alias exec nfm show flow monitor FLOW_MONITOR_MPLS cache format table
alias exec nf4 show flow monitor FLOW_MONITOR_IPV4 cache format table
alias exec nf6 show flow monitor FLOW_MONITOR_IPV6 cache format table

interface [all core interfaces]
ip flow monitor FLOW_MONITOR_IPV4 sampler FLOW_SAMPLERS_IPV4 input
ipv6 flow monitor FLOW_MONITOR_IPV6 sampler FLOW_SAMPLERS_IPV6 input
mpls flow monitor FLOW_MONITOR_MPLS input

```

FNF can be configuration intensive but the logic is very simple, much like MQC or RPL. Using the exec alias below, we can look at the MPLS cache information currently seen on the network. The other commands are not demonstrated here for brevity.

```

R5#nfm
Cache type:                Normal (Platform cache)
Cache size:                200000
Current entries:           4
High Watermark:           5

Flows added:               30
Flows aged:               26
- Inactive timeout      (   15 secs)  26

```

INTF INPUT	MPLS LABEL 1	MPLS LABEL 2	MPLS LABEL 3	MPLS LABEL 4	bytes	pkts
Gi2.525	5006 /0	0 /0	0 /0	0 /0	610	5
Gi2.515	5007 /6*	0 /0	0 /0	0 /0	101	1
Gi2.515	5001 /6*	0 /0	0 /0	0 /0	183	2
Gi2.515	5006 /0	0 /0	0 /0	0 /0	610	5

7. **Committed Access Rate (CAR):** CAR can be used for on-demand policing of certain traffic using QPPB techniques. BGP can signal (using communities) that traffic for a certain destination should receive some kind of QoS treatment (usually a police action within the context of security) and the edge routers act on it. QPPB is discussed in detail in the QoS section, but we will deploy it here as a security tool. CSR10 will set the community value on specific prefixes when it injects them into BGP; I use a route-map and network statement for this. On ingress, the ASBRs will check their BGP table-policies for specific communities, then set the qos-group marker. Later in the input interface order of operations, MQC occurs, and the qos-group can be matched with policing as an action. In this example, ASBRs XRv1 and CSR4 have a policy that states the following:

- a. Routes with community 85:1024 are policed to 1Mbps
- b. Routes with community 85:2048 are policed to 2Mbps
- c. Routes with community 85:4096 are policed to 4Mbps

The benefit of using prefix-independent QPPB matching policies, such as communities as AS-paths, is that they apply to all AFs. XE does not appear to support IPv6 QPPB at all. The configurations are summarized here; CSR10's community imposition was detailed in the "BGP policy accounting" test since the same communities were used. The remaining QPPB configuration on CSR4 is shown below. This is very similar to the BGP PA test since the same feature set is used for traffic matching. The only significant difference is that we set a qos-group, not a traffic-index.

```
! CSR4
ip community-list standard COML_1MBPS permit 85:1024
ip community-list standard COML_2MBPS permit 85:2048
ip community-list standard COML_4MBPS permit 85:4096

route-map RM_QPPB permit 10
  match community COML_1MBPS
  set ip qos-group 1
route-map RM_QPPB permit 20
  match community COML_2MBPS
  set ip qos-group 2
route-map RM_QPPB permit 30
  match community COML_4MBPS
  set ip qos-group 4

router bgp 85
  address-family ipv4
    table-map RM_QPPB
  address-family ipv6
    table-map RM_QPPB

interface GigabitEthernet2.514
  bgp-policy destination ip-qos-map
```

QPPB is quite involved, so we will verify the key components. First, we verify that the route has the proper community and is being received by the ASBRs from both RRs. Recall that XR automatically distributes communities to iBGP peers, but XE does not.

```
R4#show bgp ipv4 unicast 85.19.8.0/21
BGP routing table entry for 85.19.8.0/21, version 815
Paths: (2 available, best #1, table default)
  Advertised to update-groups:
    13
  Refresh Epoch 1
  Local
    10.10.10.10 (metric 30) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 0, localpref 100, valid, internal, best
      Community: 85:1024
      Originator: 10.10.10.10, Cluster list: 2.2.2.2
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  Local
    10.10.10.10 (metric 30) from 14.14.14.14 (14.14.14.14)
      Origin IGP, metric 0, localpref 100, valid, internal
      Community: 85:1024
      Originator: 10.10.10.10, Cluster list: 14.14.14.14
      rx pathid: 0, tx pathid: 0
```

Next, ensure the QPPB policy is matching the community properly and applying qos-group 1 to the prefix. If CSR4's routing table doesn't show this setting, this likely indicates a problem with the configuration. Unlike BGP PA, clearing the RIB is not required for qos-group modifications to take effect.

```
R4#show ip route 85.19.8.0
Routing entry for 85.19.8.0/21
  Known via "bgp 85", distance 200, metric 0, qos-group 1, type internal
  Last update from 10.10.10.10 00:04:13 ago
  Routing Descriptor Blocks:
  * 10.10.10.10, from 2.2.2.2, 00:04:13 ago
    Route metric is 0, traffic share count is 1
    AS Hops 0
    MPLS label: none
```

Next, ensure QPPB is enabled on the interface. This completes the QPPB verification.

```
R4#show ip interface gig2.514 | include BGP
BGP Policy Mapping is enabled (output ip-qos-map)
Input features: BGP Policy Map, Access List, MCI Check
```

We quickly define our MQC policy to police based on QG markings, and then apply it inbound to the interface in question. This is nothing new and is not discussed in detail.


```

! CSR4
class-map match-all CMAP_QG_1
  match qos-group 1
class-map match-all CMAP_QG_2
  match qos-group 2
class-map match-all CMAP_QG_4
  match qos-group 4

policy-map PMAP_POLICE_IN
  class CMAP_QG_1
    police 1024000
  class CMAP_QG_2
    police 2048000
  class CMAP_QG_4
    police 4096000

interface GigabitEthernet2.514
  service-policy input PMAP_POLICE

```

We verify that the policy was applied successfully. Here, we can see both QPPB and MQC correctly enabled on the interface as input features.

```

R4#show ip interface gig2.514 | include BGP|QoS
  BGP Policy Mapping is enabled (output ip-qos-map)
  Input features: BGP Policy Map, QoS Classification, Access List, MCI Check

R4#show policy-map interface brief
Service-policy input: PMAP_POLICE_IN
  GigabitEthernet2.514

```

To test it from CSR1, we will remove the local-preference 200 modification we used to test uRPF earlier. CSR1 must route through CSR4 to reach AS 85. You may need to flap BGP on CSR8 so that CSR4 advertises older eBGP routes. Notice that CSR1 has no idea about the qos-group or community settings; communities were not leaked outside of AS 85 and qos-group is always a local-router-only setting.

```

R1#show bgp ipv4 unicast 85.19.8.0/21 bestpath
BGP routing table entry for 85.19.8.0/21, version 532
Paths: (3 available, best #3, table default)
  Advertised to update-groups:
    2
  Refresh Epoch 2
  85
    10.1.4.4 from 10.1.4.4 (4.4.4.4)
      Origin IGP, localpref 100, valid, external, best
      rx pathid: 0, tx pathid: 0x0

```

Pinging from CSR1 to this special destination, we see the packets match the MQC policy and are policed accordingly. Pinging from CSR1 (or any source) to any other destination in AS 85 will not be policed since no other routes have been marked with a special community indicating QoS treatment. Of course, any other MQC-supported action is appropriate here (QoS marking, etc) but the CAR technique is specific using QPPB for policing.

```
R1#ping 85.19.8.1 source 1.14.1.1 repeat 50 timeout 1
[snip]
Success rate is 100 percent (50/50), round-trip min/avg/max = 2/18/26 ms

R1#ping 85.19.4.1 source 1.14.1.1 repeat 150 timeout 1
Success rate is 100 percent (150/150), round-trip min/avg/max = 4/22/129 ms
```

```
R4#show policy-map interface gigabitEthernet 2.514 input
GigabitEthernet2.514
```

```
Service-policy input: PMAP_POLICE_IN
```

```
Class-map: CMAP_QG_1 (match-all)
  50 packets, 5900 bytes
  5 minute offered rate 0000 bps, drop rate 0000 bps
  Match: qos-group 1
  police:
    cir 1024000 bps, bc 32000 bytes
    conformed 50 packets, 5900 bytes; actions:
      transmit
    exceeded 0 packets, 0 bytes; actions:
      drop
    conformed 0000 bps, exceeded 0000 bps
```

```
Class-map: CMAP_QG_2 (match-all)
  0 packets, 0 bytes
[snip]
```

```
Class-map: CMAP_QG_4 (match-all)
  0 packets, 0 bytes
[snip]
```

```
Class-map: class-default (match-any)
  157 packets, 18979 bytes
  5 minute offered rate 1000 bps, drop rate 0000 bps
  Match: any
```

We cannot apply MQC policies in XRv at all, so testing the data-plane is impossible. However, I've added the appropriate QPPB configurations on XRv1 that are comparable to CSR4. Like BGP PA on XR, this feature is very similar.

```
! XRv1
```

```

route-policy RPL_QPPB
  if community matches-every (85:1024) then
    set qos-group 1
  elseif community matches-every (85:2048) then
    set qos-group 2
  elseif community matches-every (85:4096) then
    set qos-group 4
  else
    pass
  endif
end-policy

router bgp 85
  address-family ipv4 unicast
    table-policy RPL_QPPB
  address-family ipv6 unicast
    table-policy RPL_QPPB

interface GigabitEthernet0/0/0/0.513
  ipv4 bgp policy propagation input qos-group destination
  ipv6 bgp policy propagation input qos-group destination

```

We will limit the verification to control-plane show commands. First, we verify that the ASBR received the route with the proper communities. Two entries appear, one from each RR.

```

RP/0/0/CPU0:XRv1#show bgp ipv4 unicast community 85:1024 | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
*>i85.19.8.0/21    10.10.10.10          0    100      0 i
* i                10.10.10.10          0    100      0 i

```

Next, we verify the BGP policy is applied to the interface so traffic can have its qos-group set. This information is shown in the FIB, but not in the RIB.

```

RP/0/0/CPU0:XRv1#show cef ipv4 85.19.8.0/21
85.19.8.0/21, version 852, internal 0x5000001 0x0 (ptr 0xa13d30f4) [1], 0x0
(0x0), 0x0 (0x0)
  local adjacency 85.1.5.5
  Prefix Len 21, traffic index 0, precedence n/a, priority 4
  QoS Group: 1
[snip]

```

For completeness, the MQC policy is shown below, but cannot be applied due to XRv limitations.

```

! XRv1
class-map match-all CMAP_QG_1
  match qos-group 1
class-map match-all CMAP_QG_2
  match qos-group 2

```

```
class-map match-all CMAP_QG_4
  match qos-group 4
```

```
policy-map PMAP_POLICE_IN
  class CMAP_QG_1
    police rate 1024000 bps
  class CMAP_QG_2
    police rate 2048000 bps
  class CMAP_QG_4
    police rate 4096000 bps
```

It is safe to assume that if the MQC policy was applied inbound on the outside interface, that traffic destined to this special route would be policed in accordance with the class matching qos-group 1, which specifies a 1 Mbps maximum rate. CSR10 is configured to advertise an IPv6 route with a community requesting a 2 Mbps shaped rate. We will ensure XRv1 sets the proper QoS group. The MQC policy would work automatically without any further modifications to support IPv6. We left an IPv6 QPPB table-map on CSR4 for demonstration purposes, but it doesn't accomplish anything.

```
RP/0/0/CPU0:XRv1#show bgp ipv6 unicast community 85:2048 | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
*>i2001:85:19::/63    10.10.10.10              0      100      0 i
* i                   10.10.10.10              0      100      0 i

RP/0/0/CPU0:XRv1#show cef ipv6 2001:85:19::/63
2001:85:19::/63, version 1113, internal 0x5000001 0x0 (ptr 0xa13b97f4) [1],
0x0 (0x0), 0x208 (0xa152c118)
Prefix Len 63, traffic index 0, precedence n/a, priority 4
QoS Group: 2
[snip]
```

As a sanity check on CSR4, we see the community is properly received but the qos-group is never applied, reinforcing my belief that IPv6 QPPB is not supported on XE currently.

```
R4#show bgp ipv6 unicast community 85:2048 | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
*>i 2001:85:19::/63    ::FFFF:10.10.10.10
                                     0      100      0 i
* i                   ::FFFF:10.10.10.10
                                     0      100      0 i

R4#show ipv6 route 2001:85:19::/63
Routing entry for 2001:85:19::/63
  Known via "bgp 85", distance 200, metric 0, type internal
  Route count is 1/1, share count 0
  Routing paths:
    10.10.10.10%default indirectly connected
    MPLS label: 10007
```

49.4 Advanced security techniques and features

1. **Sinkholes:** A sinkhole is an advanced technique used to attract traffic, like a magnet. Commonly, this is an isolated part of the SP network that advertises routes in an attempt to fool attackers; a sinkhole is to a network like a honey pot is to a server farm. Often times, “dark IP addresses”, or addresses allocated to a provider but not used, along with RFC1918 address, are originated via static routes and distributed across the topology via BGP. Indiscriminate attacks will follow these routes and into a sinkhole where different analysis tools are used to study the attacks. Netflow and classification ACLs are simple mechanisms, but purpose-built sniffer appliances (intrusion detection systems, etc) are more appropriate in real life. Beware that sinkholes can attract a lot of “junk” traffic, so the sinkhole gateway router should have a few high-speed interfaces to the core and should be a powerful router in general. Rather than make static routes to null0, which would just drop the traffic on the sinkhole gateway, a static route points out to the “analysis LAN” interface (where all the sniffer tools reside) with a bogus MAC address. The switched network will receive Ethernet frames destined to this bogus MAC and no switch will have CAM entries for it, implying it will be flooded as an unknown-unicast frame. A dedicated switched network is usually desired in a sinkhole since it is essentially a quarantine for attacks. CSR3 and CSR7 are sinkhole gateways in this design, where CSR3 attracts all of the unused space in AS 85 using a route to 85.0.0.0/8 and 2001:85::/32, while CSR7 attracts all the other traffic with IPv4/IPv6 default routes. Sinkholes can be deployed in anycast fashion, where they both advertise the same routes, and the network selects the closer one based on local BGP (or possibly IGP, but not recommended) best-path calculations. Both sinkhole gateways define basic FNF policies below, along with the sinkhole LANs and static routes.

```
! CSR3 only
interface GigabitEthernet2.3503
  description SINKHOLE
  encapsulation dot1Q 3503
  ip address 10.0.0.2 255.255.255.252
  ipv6 address FE80::3 link-local
  ipv6 address FD00::1/127

ip route 85.0.0.0 255.0.0.0 GigabitEthernet2.3503 10.0.0.1
ipv6 route 2001:85::/32 GigabitEthernet2.3503 FD00::

arp 10.0.0.1 0000.beef.cafe ARPA
ipv6 neighbor FD00:: GigabitEthernet2.3503 0000.beef.cafe
```

There are a few different sinkhole designs, but the preferred design is the one with a single router. Both CSR3 and CSR7 implement this design. On the sinkhole gateways, we verify their static routes pointing out towards their sniffer VLANs with bogus MAC addresses. We trace the entire route recursion for both IPv4 and IPv6, and also verify the ARP/IPv6 ND bindings, which were statically assigned. It is preferable to use very small subnets for the sinkhole from the perspective of the gateway router, which ensures the route-recursion doesn't point some other way. It also prevents the routing from resolving ARP/ND

entries for sniffer devices, which increases security, as those passive listening devices never need to send receive directed traffic from the network.

```
R3#show ip route 85.0.0.0 255.0.0.0
Routing entry for 85.0.0.0/8
  Known via "static", distance 1, metric 0
  Advertised by bgp 85
  Routing Descriptor Blocks:
  * 10.0.0.1, via GigabitEthernet2.3503
    Route metric is 0, traffic share count is 1

R3#show ip route 10.0.0.1
Routing entry for 10.0.0.0/30
  Known via "connected", distance 0, metric 0 (connected, via interface)
  Routing Descriptor Blocks:
  * directly connected, via GigabitEthernet2.3503
    Route metric is 0, traffic share count is 1

R3#show ip arp 10.0.0.1
Protocol Address          Age (min)  Hardware Addr   Type   Interface
Internet 10.0.0.1          -         0000.beef.cafe  ARPA

R3#show ipv6 route 2001:85::/32
Routing entry for 2001:85::/32
  Known via "static", distance 1, metric 0
  Route count is 1/1, share count 0
  Routing paths:
  FD00::, GigabitEthernet2.3503
    Last updated 00:02:42 ago

R3#show ipv6 route FD00::
Routing entry for FD00::/127
  Known via "connected", distance 0, metric 0, type connected
  Route count is 1/1, share count 0
  Routing paths:
  directly connected via GigabitEthernet2.3503
    Last updated 00:03:09 ago

R3#show ipv6 neighbors gig2.3503 FD00::
IPv6 Address                               Age Link-layer Addr State Interface
FD00::                                     - 0000.beef.cafe  REACH Gi2.3503
```

After advertising both the IPv4 and IPv6 sinkhole routes into BGP, let's verify that the ASBRs received them. For brevity we will verify XRv1 and CSR4. Each router has two copies of the path, one from each RR. We can use network statements or redistribution for this; CSR3 will use network statements. We also use "next-hop-self" as needed because no one else in AS 85 will have routes to the bogus static route next-hops.

```

! CSR3
router bgp 85
  address-family ipv4
    network 85.0.0.0
    neighbor 2.2.2.2 next-hop-self
    neighbor 14.14.14.14 next-hop-self
  address-family ipv6
    network 2001:85::/32
    ! next-hop-self inherited from AF_IPV6_LUCAST template

RP/0/0/CPU0:XRv1#show bgp ipv4 unicast 85.0.0.0/8 brief | begin Network
      Network           Next Hop           Metric LocPrf Weight Path
*>i85.0.0.0/8           3.3.3.3              0      100      0 i
* i                     3.3.3.3              0      100      0 i

RP/0/0/CPU0:XRv1#show bgp ipv6 unicast 2001:85::/32 brief | begin Network
      Network           Next Hop           Metric LocPrf Weight Path
*>i2001:85::/32         3.3.3.3              0      100      0 i
* i                     3.3.3.3              0      100      0 i

R4#show bgp ipv4 unicast 85.0.0.0/8
BGP routing table entry for 85.0.0.0/8, version 820
Paths: (2 available, best #2, table default)
  Advertised to update-groups:
    13
  Refresh Epoch 1
  Local
    3.3.3.3 (metric 30) from 14.14.14.14 (14.14.14.14)
      Origin IGP, metric 0, localpref 100, valid, internal
      Originator: 3.3.3.3, Cluster list: 14.14.14.14
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  Local
    3.3.3.3 (metric 30) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 0, localpref 100, valid, internal, best
      Originator: 3.3.3.3, Cluster list: 2.2.2.2
      rx pathid: 0, tx pathid: 0x0

R4#show bgp ipv6 unicast 2001:85::/32
BGP routing table entry for 2001:85::/32, version 729
Paths: (2 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 4
  Local
    ::FFFF:3.3.3.3 (metric 30) from 2.2.2.2 (2.2.2.2)
      Origin IGP, metric 0, localpref 100, valid, internal, best
      Originator: 3.3.3.3, Cluster list: 2.2.2.2
      mpls labels in/out nolabel/3014
      rx pathid: 0, tx pathid: 0x0

```

Refresh Epoch 1

Local

::FFFF:3.3.3.3 (metric 30) from 14.14.14.14 (14.14.14.14)

Origin IGP, metric 0, localpref 100, valid, internal

Originator: 3.3.3.3, Cluster list: 14.14.14.14

mpls labels in/out nolabel/3014

rx pathid: 0, tx pathid: 0

To measure traffic entering the sinkhole, CSR3 will reuse the FNF configuration for IPv4 and IPv6 already configured on CSR5 and CSR6 (records not shown again). This will be applied outbound towards the sinkhole to capture 100% of the traffic. In real life, you would never do this, but since the sinkhole LAN is notional, we confirm the router is forwarding traffic onto that LAN properly. The configuration is shown below without the flow record definitions; just the flow monitors and application of FNF policies.

! CSR3

flow monitor FLOW_MONITOR_IPV4

record FLOW_RECORD_IPV4

flow monitor FLOW_MONITOR_IPV6

record FLOW_RECORD_IPV6

interface GigabitEthernet2.3503

ip flow monitor FLOW_MONITOR_IPV4 output

ipv6 flow monitor FLOW_MONITOR_IPV6 output

From CSR1 and CSR9, we will flood some garbage traffic to fake destinations within AS 85's address block. Regardless of where it enters the AS, it should end up in the sinkhole behind CSR3. Because we are piling many security features on top of each other, I had to remove the ACLs applied to some of the routers so that traffic to infrastructure networks (typically blocked by iACLs) is allowed to enter. As expected, some of the SP security features are mutually exclusive, so having a comprehensive design is important. The FNF output below proves that IPv4 traffic is being sent into the sinkhole LAN.

R1#ping 85.33.44.55 source 1.14.1.1 repeat 100 timeout 0

R9#ping 85.66.33.11 source 9.15.9.9 repeat 44 timeout 0

RP/0/0/CPU0:XRv3#ping 85.77.33.44 source 13.15.13.13 count 77 timeout 0

R3#show flow monitor FLOW_MONITOR_IPV4 cache format record

[snip]

IPv4 SOURCE ADDRESS:	9.15.9.9
IPv4 DESTINATION ADDRESS:	85.66.33.11
TRNS SOURCE PORT:	0
TRNS DESTINATION PORT:	2048
INTERFACE INPUT:	Gi2.535
IP DSCP:	0x00


```

IP PROTOCOL:                1

IPV4 SOURCE ADDRESS:        13.15.13.13
IPV4 DESTINATION ADDRESS:    85.77.33.44
[snip]

IPV4 SOURCE ADDRESS:        1.14.1.1
IPV4 DESTINATION ADDRESS:    85.33.44.55
[snip]

```

Let's also verify that IPv6 traffic ends up in the sinkhole also by repeating the test with IPv6 addressing. Again, FNF proves the traffic is ending up on the LAN destined to our bogus MAC address.

```

R1#ping 2001:85:55:55::44 source 2001:1C00:D::1 repeat 50 timeout 0

R9#ping 2001:85:33:88::22 source 2001:9000:F::9 repeat 20 timeout 0

RP/0/0/CPU0:XRv3#ping 2001:85:22:55::11 source 2001:3000:a::13 count 50
timeout 0

R3#show flow monitor FLOW_MONITOR_IPV6 cache format record
[snip]

```

```

IPV6 SOURCE ADDRESS:        2001:3000:A::13
IPV6 DESTINATION ADDRESS:    2001:85:22:55::11
TRNS SOURCE PORT:           0
TRNS DESTINATION PORT:      32768
INTERFACE INPUT:            Gi2.535
IP DSCP:                     0x00
IP PROTOCOL:                58

IPV6 SOURCE ADDRESS:        2001:1C00:D::1
IPV6 DESTINATION ADDRESS:    2001:85:55:55::44
[snip]

IPV6 SOURCE ADDRESS:        2001:9000:F::9
IPV6 DESTINATION ADDRESS:    2001:85:33:88::22
[snip]

```

CSR7 will be configured similarly except using default routes. First, we should verify the route recursion and ARP/IPv6 ND bindings are correct on CSR7. Notice that the routes are tagged with a value of 10; this will be used for redistribution into BGP later.

```

! CSR7
interface GigabitEthernet2.3507
 description SINKHOLE
 encapsulation dot1Q 3507
 ip address 10.0.0.2 255.255.255.252

```

```

ip flow monitor FLOW_MONITOR_IPV4 output
ipv6 flow monitor FLOW_MONITOR_IPV6 output
ipv6 address FE80::7 link-local
ipv6 address FD00::1/127

ip route 0.0.0.0 0.0.0.0 GigabitEthernet2.3507 10.0.0.1 tag 10
ipv6 route ::/0 GigabitEthernet2.3507 FD00:: tag 10

arp 10.0.0.1 0000.dead.deed ARPA
ipv6 neighbor FD00:: GigabitEthernet2.3507 0000.dead.deed

R7#show ip route 0.0.0.0 0.0.0.0
Routing entry for 0.0.0.0/0, supernet
  Known via "static", distance 1, metric 0, candidate default path
  Redistributing via bgp 85
  Advertised by bgp 85
  Routing Descriptor Blocks:
  * 10.0.0.1, via GigabitEthernet2.3507
    Route metric is 0, traffic share count is 1

R7#show ip route 10.0.0.1
Routing entry for 10.0.0.0/30
  Known via "connected", distance 0, metric 0 (connected, via interface)
  Routing Descriptor Blocks:
  * directly connected, via GigabitEthernet2.3507
    Route metric is 0, traffic share count is 1

R7#show ip arp 10.0.0.1

```

Protocol	Address	Age (min)	Hardware Addr	Type	Interface
Internet	10.0.0.1	-	0000.dead.deed	ARPA	

```

R7#show ipv6 route ::/0
Routing entry for ::/0
  Known via "static", distance 1, metric 0
  Redistributing via bgp 85
  Route count is 1/1, share count 0
  Routing paths:
    FD00::, GigabitEthernet2.3507
      Last updated 00:05:12 ago

R7#show ipv6 route FD00::
Routing entry for FD00::/127
  Known via "connected", distance 0, metric 0, type connected
  Route count is 1/1, share count 0
  Routing paths:
    directly connected via GigabitEthernet2.3507
      Last updated 00:06:02 ago

R7#show ipv6 neighbors gig2.3507 FD00::

```

IPv6 Address	Age	Link-layer Addr	State	Interface
FD00::	-	0000.dead.deed	REACH	Gi2.3507

Once it is complete, we can advertise the default routes into BGP via redistribution. Rather than test this from all routers as we did earlier, we will verify that the ASBR's receive the default route, and then test from CSR1 only. Rather than use "next-hop-self" as CSR3 did, we will manually adjust the next-hops for these redistributed routes to add complexity.

```
! CSR7
route-map RM_STATIC_BGP_IPV4 permit 10
  match tag 10
  set ip next-hop 7.7.7.7

route-map RM_STATIC_BGP_IPV6 permit 10
  match tag 10
  set ipv6 next-hop ::FFFF:7.7.7.7

router bgp 85
  address-family ipv4
    redistribute static route-map RM_STATIC_BGP_IPV4
    default-information originate
  address-family ipv6
    redistribute static route-map RM_STATIC_BGP_IPV6
    default-information originate
```

```
RP/0/0/CPU0:XRv1#show bgp ipv4 unicast 0.0.0.0/0 brief | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
*>i0.0.0.0/0      7.7.7.7          0      100      0 ?
* i                7.7.7.7          0      100      0 ?
```

```
RP/0/0/CPU0:XRv1#show bgp ipv6 unicast ::/0 brief | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
*>i::/0          7.7.7.7          0      100      0 ?
* i              7.7.7.7          0      100      0 ?
```

```
R4#show bgp ipv4 unicast 0.0.0.0/0
BGP routing table entry for 0.0.0.0/0, version 823
Paths: (2 available, best #2, table default)
  Advertised to update-groups:
    13
  Refresh Epoch 1
  Local
    7.7.7.7 (metric 20) from 14.14.14.14 (14.14.14.14)
    Origin incomplete, metric 0, localpref 100, valid, internal
    Originator: 7.7.7.7, Cluster list: 14.14.14.14
    rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  Local
```

```

7.7.7.7 (metric 20) from 2.2.2.2 (2.2.2.2)
  Origin incomplete, metric 0, localpref 100, valid, internal, best
  Originator: 7.7.7.7, Cluster list: 2.2.2.2
  rx pathid: 0, tx pathid: 0x0

R4#show bgp ipv6 unicast ::/0
BGP routing table entry for ::/0, version 732
Paths: (2 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 6
  Local
    ::FFFF:7.7.7.7 (metric 20) from 2.2.2.2 (2.2.2.2)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Originator: 7.7.7.7, Cluster list: 2.2.2.2
      mpls labels in/out nolabel/7015
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 1
  Local
    ::FFFF:7.7.7.7 (metric 20) from 14.14.14.14 (14.14.14.14)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Originator: 7.7.7.7, Cluster list: 14.14.14.14
      mpls labels in/out nolabel/7015
      rx pathid: 0, tx pathid: 0

```

CSR1 does not have any default-route initially, as seen below, for either IPv4 or IPv6. This is the result of our outbound BGP filtering policies. We would adjust those policies, but advertising default routes between SPs over the Internet is generally not advisable.

```

R1#show ip cef 86.1.2.3
0.0.0.0/0
  no route

R1#show ipv6 cef 2001:86:1:2::3
::/0
  no route

```

Adding a temporary static default on CSR1 pointing to CSR4 solves this. This ensures CSR1 will send all default traffic towards CSR4, which. Upon receipt of this traffic, CSR4 should direct it into the sinkhole. Notice that the first octet is 86, not 85, so rather than go to the “intra-network” sinkhole at CSR3, this should go to the “everything else” sinkhole at CSR7.

```

R1#ping 86.1.2.3 source 1.14.1.1 repeat 30 timeout 0

R1#ping 86.1.2.5 source 1.14.1.1 repeat 30 timeout 0

R1#ping 86.1.2.6 source 1.14.1.1 repeat 30 timeout 0

```

```
R7#show flow monitor FLOW_MONITOR_IPV4 cache format record
[snip]
```

```
IPV4 SOURCE ADDRESS:      1.14.1.1
IPV4 DESTINATION ADDRESS:  86.1.2.6
TRNS SOURCE PORT:         0
TRNS DESTINATION PORT:    2048
INTERFACE INPUT:          Gi2.567
IP DSCP:                   0x00
IP PROTOCOL:              1
```

```
IPV4 SOURCE ADDRESS:      1.14.1.1
IPV4 DESTINATION ADDRESS:  86.1.2.5
[snip]
```

```
IPV4 SOURCE ADDRESS:      1.14.1.1
IPV4 DESTINATION ADDRESS:  86.1.2.3
[snip]
```

Another quick test ensures that the IPv6 default traffic is also routed towards the sinkhole. Again, notice that 86 in the second IPv6 stanza, which means CSR7 will attract this traffic.

```
R1#ping 2001:86:1:2::3 source 2001:1C00:D::1 repeat 50 timeout 0
```

```
R1#ping 2001:86:1:2::4 source 2001:1C00:D::1 repeat 50 timeout 0
```

```
R1#ping 2001:86:1:2::5 source 2001:1C00:D::1 repeat 50 timeout 0
```

```
R7#show flow monitor FLOW_MONITOR_IPV6 cache format record
[snip]
```

```
IPV6 SOURCE ADDRESS:      2001:1C00:D::1
IPV6 DESTINATION ADDRESS:  2001:86:1:2::4
TRNS SOURCE PORT:         0
TRNS DESTINATION PORT:    32768
INTERFACE INPUT:          Gi2.567
IP DSCP:                   0x00
IP PROTOCOL:              58
```

```
IPV6 SOURCE ADDRESS:      2001:1C00:D::1
IPV6 DESTINATION ADDRESS:  2001:86:1:2::5
[snip]
```

```
IPV6 SOURCE ADDRESS:      2001:1C00:D::1
IPV6 DESTINATION ADDRESS:  2001:86:1:2::3
[snip]
```

The default routes are removed from CSR1 before continuing.

2. Remotely triggered blackholes (RTBH) with next-hop modification: Similar to the remote CAR technique using QPPB, BGP communities, and qos-groups, this feature is used to signal ASBRs to drop traffic towards a specific destination. This variation of RTBH is less desirable and more clunky than the community-based version analyzed later. The logic is as follows: a router in the topology originates a BGP route with a next-hop set to 192.0.2.1, for example. All of the ASBRs have a static route to 192.0.2.0/24 pointing to null0. Because BGP cannot advertise a prefix with a null next-hop, this route-recursion technique essentially lets the originating router (the “trigger” router) immediately blackhole all traffic heading towards that destination. This is typically used when a host/subnet is under DDoS attack and it is less damaging to blackhole all traffic to that host/subnet rather than let the attack continue. The time it takes to blackhole all traffic is limited only by iBGP convergence time; given an advertisement-interval default value of 0, this is normally fast. and null0 drop mechanisms happen in CEF on the ASBRs, so there is little to no performance impact on the ASBRs.

The first step is ensuring that all ASBRs have static routes for 192.0.2.0/24 (or any other dummy prefix, but this is commonly used because it’s the “Test-Network”) pointing to null0. For IPv6, I used FD00:FFFF::/32. We verify this on XRv1 and CSR4 for brevity.

```
! CSR4 and CSR8
ip route 192.0.2.0 255.255.255.0 Null0
ipv6 route FD00:FFFF::/32 Null0

! XRv1 and XRv2
router static
  address-family ipv4 unicast
    192.0.2.0/24 Null0
  address-family ipv6 unicast
    fd00:ffff::/32 Null0

RP/0/0/CPU0:XRv1#show cef ipv4 192.0.2.0/24
192.0.2.0/24, version 883, attached, null0 adjacency, internal 0x1000041 0x0
(ptr 0xa13d3ff4) [1], 0x0 (0xa139e050), 0x0 (0x0)
Prefix Len 24, traffic index 0, precedence n/a, priority 2
  via Null0, 3 dependencies, weight 0, class 0 [flags 0x8]
    path-idx 0 NHID 0x0 [0xa0ce7050 0x0]
    null0 adjacency

RP/0/0/CPU0:XRv1#show cef ipv6 FD00:FFFF::/32
fd00:ffff::/32, version 1184, attached, null0 adjacency, internal 0x1000041
0x0 (ptr 0xa13b8ef4) [1], 0x0 (0xa13846c8), 0x0 (0x0)
Prefix Len 32, traffic index 0, precedence n/a, priority 2
  via Null0, 3 dependencies, weight 0, class 0 [flags 0x8]
    path-idx 0 NHID 0x0 [0xa0ccd050 0x0]
    null0 adjacency

R4#show ip cef 192.0.2.0 255.255.255.0
```

```
192.0.2.0/24
```

```
attached to Null0
```

```
R4#show ipv6 cef FD00:FFFF::/32
```

```
FD00:FFFF::/32
```

```
attached to Null0
```

Now, any routes (BGP or otherwise) with a next-hop in these null ranges will be dropped. We can create a semi-dynamic mechanism on XRv4 and CSR7, our trigger routers, to handle this. When an administrator configures a static route with a certain tag (99 in this case), BGP redistributes it from static and sets some key attributes. Local-preference is set to a high value to defeat other candidate best-paths in the network. The well-known communities “no-export” or “local-as” are set so the blackhole route does not leak out of the AS or sub-AS, respectively. Most importantly, the next-hop is changed to 192.0.2.1 or FD00:FFFF::1 which causes the route recursion on each ASBR to evaluate to null0. Be careful that any existing static into BGP redistribution is not broken; earlier we used redistribution on CSR7 to advertise a default route for a sinkhole. We need to ensure that functionality does not break in addition to configuring RTBH. The route-maps for IPv4 and IPv6 are identical except for the next-hop change. The first clause matches ordinary static routes that need to be injected into BGP (not shown again), and the second clause is for RTBH routes. The route-map was already applied under the IPv6 AF when redistributing static into BGP, and likewise the IPv4 map is applied under the IPv4 AF.

```
! CSR7
```

```
route-map RM_STATIC_BGP_IPV6 permit 20
```

```
match tag 99
```

```
set local-preference 200
```

```
set community no-export
```

```
set ipv6 next-hop FD00:FFFF::1
```

```
route-map RM_STATIC_BGP_IPV4 permit 20
```

```
match tag 99
```

```
set local-preference 200
```

```
set community no-export
```

```
set ip next-hop 192.0.2.1
```

The XR configuration is similar except uses RPL. Parameterization allows us to reuse the same RPL policy by feeding in different next-hop values when we call the RPL. I show the RPL attachment points for clarity since the next-hops are passed inline, not as part of an RPL construct like a prefix-set.

```
! XRv4
```

```
route-policy RPL_STATIC_TO_BGP ($NHOP)
```

```
if tag is 99 then
```

```
set local-preference 200
```

```
set community (no-export)
```

```
set next-hop $NHOP
```

```
endif
```

```
end-policy
```

```

router bgp 85
  address-family ipv4 unicast
    redistribute static route-policy RPL_STATIC_TO_BGP (192.0.2.1)

  address-family ipv6 unicast
    redistribute static route-policy RPL_STATIC_TO_BGP (FD00:FFFF::1)

```

At this point, any static routes created on CSR7 or XRv4 with a tag of 99 will be blackholed in the entire BGP network. Let's assume that XRv4 is our primary trigger router and, as a matter of policy, we only use CSR7 when XRv4 fails. There is currently a DDoS attack against a server 85.19.16.1/32, but other servers in the subnet are not under attack. CSR1 is currently attacking the server and has reachability.

```

R1#ping 85.19.16.1 source 1.14.1.1
[snip]
Success rate is 100 percent (5/5)

```

Adding a static route to 85.19.16.1/32 locally on XRv4 with tag 99 should ensure this traffic gets black holed from all ASBRs. However, a quick test from CSR1 shows RTBH not working.

```

! XRv4
router static
  address-family ipv4 unicast
    85.19.16.1/32 Null0 tag 99

R1#ping 85.19.16.1 source 1.14.1.1
[snip]
Success rate is 100 percent (5/5)

```

At this point, we can see the drawback of this design; intermediate BGP routers must have a route to the bogus next-hop, so be sure to add the null routes for 192.0.2.0/24 and FD00:FFFF::/32 as needed. This is basic BGP behavior and is not specific to RTBH. If you forget this step, the trigger routers and/or RRs will not have a best-path to the black-hole route and will not advertise it further. Always pay attention if BGP is not advertising routes; it is probably related to next-hop inaccessibility when using this technique.

```

! XRv4
router static
  address-family ipv4 unicast
    192.0.2.0/24 Null0

```

Let's check CSR4 to see if it received the route. We see the route arrived with a next-hop of 192.0.2.1 from XRv4. Continuing the route lookups, this resolves to null0, and the CEF entry for the black-holed route evaluates to null0. External entities can no longer reach this server.

```

R4#show bgp ipv4 unicast 85.19.16.1/32
BGP routing table entry for 85.19.16.1/32, version 824

```



```
Paths: (1 available, best #1, table default, not advertised to EBGp peer)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    192.0.2.1 from 14.14.14.14 (14.14.14.14)
      Origin IGP, metric 0, localpref 200, valid, internal, best
      Community: no-export
      rx pathid: 0, tx pathid: 0x0
```

```
R4#show ip route 192.0.2.1
Routing entry for 192.0.2.0/24
  Known via "static", distance 1, metric 0 (connected)
  Routing Descriptor Blocks:
    * directly connected, via Null0
      Route metric is 0, traffic share count is 1
R4#show ip cef 85.19.16.1 255.255.255.255
85.19.16.1/32
  nexthop 192.0.2.1 Null0
```

```
R1#ping 85.19.16.1 source 1.14.1.1
[snip]
Success rate is 0 percent (0/5)
```

We will remove the static route from XRv4 and ensure CSR1 has connectivity back to the network once iBGP converges. This is the procedure to undo the RTBH once the DDoS attack subsides. After connectivity is verified (not shown), add a static route to CSR7 to ensure the backup trigger router works the same way. CSR7 is peered to both XRv4 and CSR2 (RRs), which means CSR2 is going to need a route to 192.0.2.1 in order to select a best-path. Again, this technique makes the RTBH design very unwieldy.

```
! CSR7
ip route 85.19.16.1 255.255.255.255 Null0 tag 99

! CSR2
ip route 192.0.2.0 255.255.255.0 Null0
```

The route arrives from both RRs with a next-hop of 192.0.2.1. This resolves to null0 just like before. It doesn't really matter which route is selected as best since the next-hops are the same.

```
R4#show bgp ipv4 unicast 85.19.16.1/32
BGP routing table entry for 85.19.16.1/32, version 838
Paths: (2 available, best #1, table default, not advertised to EBGp peer)
  Not advertised to any peer
  Refresh Epoch 3
  Local
    192.0.2.1 from 2.2.2.2 (2.2.2.2)
      Origin incomplete, metric 0, localpref 200, valid, internal, best
      Community: no-export
```

```

    Originator: 7.7.7.7, Cluster list: 2.2.2.2
    rx pathid: 0, tx pathid: 0x0
Refresh Epoch 1
Local
  192.0.2.1 from 14.14.14.14 (14.14.14.14)
    Origin incomplete, metric 0, localpref 200, valid, internal
    Community: no-export
    Originator: 7.7.7.7, Cluster list: 14.14.14.14
    rx pathid: 0, tx pathid: 0

```

```

R4#show ip route 192.0.2.1
Routing entry for 192.0.2.0/24
  Known via "static", distance 1, metric 0 (connected)
  Routing Descriptor Blocks:
    * directly connected, via Null0
      Route metric is 0, traffic share count is 1

```

```

R4#show ip cef 85.19.16.1 255.255.255.255
85.19.16.1/32
  nexthop 192.0.2.1 Null0

```

```

R1#ping 85.19.16.1 source 1.14.1.1
[snip]
Success rate is 0 percent (0/5)

```

We remove the IPv4 static route from CSR7 to restore network connectivity to the server. Next, we will test IPv6 starting with XRv4. CSR1 has decided to attack 2001:85:6:84::/62, including several hosts on that network. We will blackhole the entire network this time, which implies the local-preference setting will come into effect. Last time, the /32 route had no competitors (longest-match), so BGP path attributes did not matter. Before beginning, ensure that CSR1 can reach the host in that subnet to rule out any routing issues beforehand. CSR1 is considered the attacker in this scenario once again.

```

R1#ping 2001:85:6:84::1 source 2001:1c00:f::1
[snip]
Success rate is 100 percent (5/5),

```

We add the static route to XRv4. Remember that the router needs to have a route to the BGP next-hop, even if it ultimately recurses to null0, so add that also.

```

! XRv4
router static
  address-family ipv6 unicast
    fd00:ffff::/32 Null0
    2001:85:6:84::/62 Null0 tag99

```

When we check the ASBRs, we see two different routes with two different next-hops (and thus two different MPLS labels). The real route from CSR10 is not the best path due to local-preference. This is

why setting a high local-preference is recommended for RTBH routes. However, the next-hop shows XRv4, not the bogus next-hop we initially applied.

```
R8#show bgp ipv6 unicast 2001:85:6:84::/62
[snip]
Local
  ::FFFF:14.14.14.14 from 14.14.14.14 (14.14.14.14)
    Origin incomplete, metric 0, localpref 200, valid, internal, best
[snip]
```

This is because we are using 6PE to tunnel IPv6 over an IPv4/MPLS core. In a somewhat sloppy attempt to manage this dynamically, I apply an RPL to the IPv6 labeled-unicast AF-group to forcibly change the next-hop outbound.

```
! XRv4
route-policy RPL_FORCE_NHOP_CHANGE_IPV6($NHOP)
  if community matches-any (no-export) then
    set next-hop $NHOP
  endif
end-policy

router bgp 85
  af-group AF_IPV6_LUCAST address-family ipv6 labeled-unicast
    route-policy RPL_FORCE_NHOP_CHANGE_IPV6(FD00:FFFF::1) out
```

This does not work; the router complains I am trying to set a next-hop for the wrong AFI.

```
! XRv4
bgp[1046]: %ROUTING-BGP-4-POLICY_NHOP_WRONGAFI : The policy
'RPL_FORCE_NHOP_CHANGE_IPV6(FD00:FFFF::1)' attempted to set a next-hop on
2001:85:6:84::/62 with an unsuitable address type. The next-hop will not be
modified.
```

I also tried setting the next-hop to the IPv4 compatible address for the discard route, seen below. The router gives me the same message.

```
! XRv4
router bgp 85
  af-group AF_IPV6_LUCAST address-family ipv6 labeled-unicast
    route-policy RPL_FORCE_NHOP_CHANGE_IPV6(0::FFFF:192.0.2.1) out

bgp[1046]: %ROUTING-BGP-4-POLICY_NHOP_WRONGAFI : The policy
'RPL_FORCE_NHOP_CHANGE_IPV6(0::FFFF:192.0.2.1)' attempted to set a next-hop
on 2001:85:6:84::/62 with an unsuitable address type. The next-hop will not
be modified.
```

6PE is a special case where the next-hop is initially identified and set as an IPv4 route, and BGP will convert it to IPv6. That is why the router was complaining about having the administrator setting an IPv6 next-hop manually. We simply need to set the same next-hop we did for IPv4 on XRv4, and we don't need our forcible outbound next-hop change. For completeness, we will leave our null route to FD00:FFFF::/32 in the configuration because it will work for ordinary IPv6 unicast in the future.

```
! XRv4
router bgp 85
  address-family ipv6 unicast
    no redistribute static route-policy RPL_STATIC_TO_BGP(0::FFFF:192.0.2.1)
    redistribute static route-policy RPL_STATIC_TO_BGP(192.0.2.1)

  af-group AF_IPV6_LUCAST address-family ipv6 labeled-unicast
    no route-policy RPL_FORCE_NHOP_CHANGE_IPV6(0::FFFF:192.0.2.1) out
```

This technique also fails, and the same message is printed once again.

```
! XRv4
bgp[1046]: %ROUTING-BGP-4-POLICY_NHOP_WRONGAFI : The policy
'RPL_STATIC_TO_BGP(192.0.2.1)' attempted to set a next-hop on
2001:85:6:84::/62 with an unsuitable address type. The next-hop will not be
modified.
```

The next attempt to get this working uses a static RPL (no parameterization, because it won't let parameters beginning with "::"). This is also rejected.

```
! XRv4
route-policy RPL_FORCE_NHOP_CHANGE_IPV6
  if community matches-any (no-export) then
    set next-hop ::ffff:192.0.2.1
  else
    pass
end-policy

router bgp 85
  af-group AF_IPV6_LUCAST address-family ipv6 labeled-unicast
    route-policy RPL_FORCE_NHOP_CHANGE_IPV6 out
```

```
bgp[1046]: %ROUTING-BGP-4-POLICY_NHOP_WRONGAFI : The policy
'RPL_FORCE_NHOP_CHANGE_IPV6' attempted to set a next-hop on 2001:85:6:84::/62
with an unsuitable address type. The next-hop will not be modified.
```

We make one last-ditch attempt, using the same method above with an IPv4 address in the RPL; this actually works. We continue the basic verifications on CSR4 to see that the path to the destination ultimately recurses to null0. Also note that since the BGP-next-hop changed, XRv4 is required to allocate a new label for the prefix.

```

! XRv4
route-policy RPL_FORCE_NHOP_CHANGE_IPV6
  if community matches-any (no-export) then
    set next-hop 192.0.2.1
  else
    pass
  endif

R4#show bgp ipv6 unicast 2001:85:6:84::/62
BGP routing table entry for 2001:85:6:84::/62, version 826
Paths: (2 available, best #2, table default, not advertised to EBGp peer)
  Not advertised to any peer
  Refresh Epoch 12
  Local
    ::FFFF:10.10.10.10 (metric 30) from 2.2.2.2 (2.2.2.2)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Originator: 10.10.10.10, Cluster list: 2.2.2.2
      mpls labels in/out nolabel/10004
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
  Local
    ::FFFF:192.0.2.1 from 14.14.14.14 (14.14.14.14)
      Origin IGP, metric 0, localpref 200, valid, internal, best
      Community: no-export
      mpls labels in/out nolabel/94010
      rx pathid: 0, tx pathid: 0x0

R4#show ip route 192.0.2.1
Routing entry for 192.0.2.0/24
  Known via "static", distance 1, metric 0 (connected)
  Routing Descriptor Blocks:
  * directly connected, via Null0
    Route metric is 0, traffic share count is 1

R4#show ipv6 cef 2001:85:6:84::/62
2001:85:6:84::/62
  nexthop 192.0.2.1 Null0 label 94010

```

The result on CSR1 is a little different now. Earlier, CSR1 still had a route for the IPv4 destination but our ASBR's had longer-match discard routes. This time, since the best-path on the ASBRs has "no-export" set, it is not advertised to any eBGP peers as the output depicts. CSR1 now relies on the 2001:85::/32 route to reach its target, which is a sinkhole route. Nonetheless, the RTBH will drop the traffic at CSR4 and the sinkhole will never see it. Without the sinkhole route, the peer ASBRs wouldn't even have a route in the first place, which is can be thought of as a control-plane-only RTBH approach.

```

R1#show bgp ipv6 unicast 2001:85::/32
[snip]

```

85

```
FD00:10:1:4::4 (FE80::4) from 10.1.4.4 (4.4.4.4)
  Origin IGP, localpref 100, valid, external, best
  rx pathid: 0, tx pathid: 0x0
[snip]
```

```
R1#show ipv6 cef 2001:85:6:84::1
2001:85::/32
  nexthop FD00:10:1:4::4 GigabitEthernet2.514
```

```
R1#ping 2001:85:6:84::1 source 2001:1c00:f::1
[snip]
Success rate is 0 percent (0/5)
```

Next, we remove the trigger static route from XRv4 and re-test using CSR7. Ensure that CSR1 has connectivity to the server again, which validates the SP routing as RTBH is toggled off and on.

```
! XRv4
router static
  address-family ipv6 unicast
    no 2001:85:6:84::/62 Null0 tag99

! R7
ipv6 route 2001:85:6:84::/62 null0 tag 99
```

CSR7 also needs to modify the next-hop outbound versus doing so at the time of redistribution, again due to the 6PE operation. Rather than change the route-map applied to redistribution (that is still valid for IPv6 unicast), I create a new route-map to mimic the forcible RPL created earlier. **Two critical caveats: you must remember to send MPLS labels in each permit clause since there is no explicit labeled-unicast AF within XE. Also, the next-hop must be the IPv6 version of the IPv4 next-hop in XE; the opposite was true in XR.** These inconsistencies make 6PE RTBH a very complex topic when using next-hop adjustments.

```
! CSR7
ip community-list standard COML_NO_EXPORT permit no-export

route-map RM_FORCE_NHOP_CHANGE_IPV6 permit 10
  match community COML_NO_EXPORT
  set mpls-label
  set ipv6 next-hop ::FFFF:192.0.2.1
route-map RM_FORCE_NHOP_CHANGE_IPV6 permit 1000
  set mpls-label
```

Next, we check CSR4 to ensure it learned the black-hole path. Notice that the MPLS label was allocated by CSR7. Again, notice this route is not advertised to CSR1, so CSR1 will rely on the shorter /32 match.

Both XRv4 and CSR2 reflected this route to CSR4, as the local-preference made it both of the RRs best paths.

```
R4#show bgp ipv6 unicast 2001:85:6:84::/62
BGP routing table entry for 2001:85:6:84::/62, version 861
Paths: (2 available, best #2, table default, not advertised to EBGp peer)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    ::FFFF:192.0.2.1 from 14.14.14.14 (14.14.14.14)
      Origin incomplete, metric 0, localpref 200, valid, internal
      Community: no-export
      Originator: 7.7.7.7, Cluster list: 14.14.14.14
      mpls labels in/out nolabel/7013
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 12
  Local
    ::FFFF:192.0.2.1 from 2.2.2.2 (2.2.2.2)
      Origin incomplete, metric 0, localpref 200, valid, internal, best
      Community: no-export
      Originator: 7.7.7.7, Cluster list: 2.2.2.2
      mpls labels in/out nolabel/7013
      rx pathid: 0, tx pathid: 0x0
```

```
R4#show ip route 192.0.2.1
Routing entry for 192.0.2.0/24
  Known via "static", distance 1, metric 0 (connected)
  Routing Descriptor Blocks:
  * directly connected, via Null0
    Route metric is 0, traffic share count is 1
```

```
R4#show ipv6 cef 2001:85:6:84::1
2001:85:6:84::/62
  nexthop 192.0.2.1 Null0 label 7013
```

Trying to reach the target from CSR1 is unsuccessful, as expected.

```
R1#show ipv6 cef 2001:85:6:84::1
2001:85::/32
  nexthop FD00:10:1:4::4 GigabitEthernet2.514
```

```
R1#ping 2001:85:6:84::1 source 2001:1c00:f::1
[snip]
Success rate is 0 percent (0/5),
```

Remove the trigger route from CSR7 (not shown) and ensure connectivity is restored once iBGP converges. This is what makes RTBH so powerful; it's a simple off/on switch.

```
R1#show ipv6 cef 2001:85:6:84::1
2001:85:6:84::/62
  nexthop FD00:10:1:8::8 GigabitEthernet2.518

R1#ping 2001:85:6:84::1 source 2001:1c00:f::1
[snip]
Success rate is 100 percent (5/5),
```

Another use of RTBH is to drop traffic for specific sources when coupled with uRPF. In the past examples, our trigger routers injected static routes representing AS 85 destinations so that the ASBRs would drop traffic towards those internal destinations. The trigger routers can also inject black-hole routes representing sources (routes in other ASes, Internet routes, etc). Recall that uRPF requires a valid route back to the source, and null0 routes do not qualify. We will test this on XRv2 where blocking sources behind CSR9 is the goal. We will use XRv4 to trigger the block for IPv4 routes and CSR7 to trigger the block for IPv6 routes in this example. The sources to block are 9.10.0.0/16 and 2001:9000:A::/48, which are known network locations for “attackers”. First, we quickly verify that uRPF is still enabled on XRv2 for both IPv4 and IPv6. The current configuration is valid, as this RTBH source-based technique works with strict or loose uRPF.

```
RP/0/0/CPU0:XRv2#show ipv4 interface gig0/0/0/0.529 | include RPF
IP unicast RPF check is enabled
RPF mode loose

RP/0/0/CPU0:XRv2#show ipv6 interface gig0/0/0/0.529 | include RPF
IP unicast RPF check is enabled
RPF mode loose, allow default route
```

Let’s ensure CSR9 can reach some destinations (we aren’t trying to protect destinations this time, just reject sources) within AS 85. If we don’t verify this first, we may get false negatives since CSR9 won’t be able to send traffic into AS 85 at all. Also ensure CSR9 is routing through XRv2 before testing since that is the uRPF-enforcing router. Currently, connectivity exists as expected.

```
R9#show ip cef 85.19.2.1
85.19.2.0/23
  nexthop 10.2.9.12 GigabitEthernet2.529

R9#show ipv6 cef 2001:85:6:84::1
2001:85:6:84::/62
  nexthop FE80::12 GigabitEthernet2.529

R9#ping 85.19.2.1 source 9.10.9.9
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/5/19 ms

R9#ping 2001:85:6:84::1 source 2001:9000:A::9
[snip]
```


Success rate is 100 percent (5/5), round-trip min/avg/max = 4/6/13 ms

Next, we configure the IPv4 trigger route on XRv4, then check XRv2 to see if it was installed as the best-path with the proper next-hop (due to local-preference). We quickly trace the route recursion to verify our configuration.

```
! XRv4
router static
  address-family ipv4 unicast
    9.10.0.0/16 Null0 tag 99

RP/0/0/CPU0:XRv2#show bgp ipv4 unicast 9.10.0.0/16 brief | begin Network
      Network          Next Hop          Metric LocPrf Weight Path
*   9.10.0.0/16        10.2.9.9              0         0 589837 ?
*>i                    192.0.2.1              0         200      0 i

RP/0/0/CPU0:XRv2#show route ipv4 192.0.2.1
Routing entry for 192.0.2.0/24
  Known via "static", distance 1, metric 0 (connected)
  Routing Descriptor Blocks
    directly connected, via Null0
      Route metric is 0
  No advertising protos.

RP/0/0/CPU0:XRv2#show cef ipv4 9.10.9.9
9.10.0.0/16, version 1043, null0 adjacency, internal 0x5000001 0x0 (ptr
0xa13d37f4) [1], 0x0 (0x0), 0x0 (0x0)
  Prefix Len 16, traffic index 0, precedence n/a, priority 4
  via 192.0.2.1, 2 dependencies, recursive [flags 0x6000]
  path-idx 0 NHID 0x0 [0xa13d3cf4 0x0]
  next hop 192.0.2.1 via 192.0.2.0/24
```

Before retesting with the black-hole enabled, we ensure our uRPF counters are cleared out.

```
RP/0/0/CPU0:XRv2#show cef interface gig0/0/0/0.529 rpf-statistics
Unicast RPF drops 0
```

When we test, CSR9 no longer has IPv4 reachability (15 packets sent), and we see the correct number of uRPF failures on XRv2. Because the sources were blocked, CSR9 would try to attack anything inside the AS and be rejected. Once a DoS attacker host is identified, RTBH plus uRPF is an effective technique to prevent all traffic entering the AS from that source.

```
R9#ping 85.19.2.1 source 9.10.9.9 repeat 15 timeout 1
[snip]
Success rate is 0 percent (0/15)
```

```
RP/0/0/CPU0:XRv2#show cef interface gig0/0/0/0.529 rpf-statistics
```

Unicast RPF drops 15

Next, we enable RTBH for the IPv6 source from CSR7. All we have to do is add the static route since the existing RTBH infrastructure has already been built and validated. We then perform our typical verifications on XRv2 to ensure the black-hole route is installed and in use. The 6PE verification on XR is a little awkward for a few reasons. First, the IPv6 next-hop shows as an ordinary IPv4 route, when we know it's really ::FFFF:192.0.2.1. Second, the IPv6 CEF entry shows an unresolved adjacency, but the action is "drop" when looking into the details.

```
! CSR7
ipv6 route 2001:9000:A::/48 null0 tag 99

RP/0/0/CPU0:XRv2#show bgp ipv6 unicast 2001:9000:A::/48 brief | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
*>i2001:9000:a::/48 192.0.2.1      0      200      0 ?
*                  fd00:10:2:9::9      0                  0 589837 ?

RP/0/0/CPU0:XRv2#show route ipv4 192.0.2.1
Routing entry for 192.0.2.0/24
  Known via "static", distance 1, metric 0 (connected)
  Routing Descriptor Blocks
    directly connected, via Null0
      Route metric is 0
  No advertising protos.

RP/0/0/CPU0:XRv2#show cef ipv6 2001:9000:a::9 detail
[snip]
  via ::ffff:192.0.2.1, 0 dependencies, recursive, platform-non_preferred
  path-idx 0 NHID 0x0 [0x0 0x0]
  recursion-via-/128
  next hop VRF - 'default', table - 0xe0000000
  unresolved
  labels imposed {7014}
[snip]
Hash  OK  Interface      Address
0      Y   Unknown        drop
```

Next, we ensure our IPv6 uRPF counters are clear, then test connectivity from CSR9. We should see exactly 12 uRPF failures on XRv2 with the black-hole in place.

```
RP/0/0/CPU0:XRv2#show cef ipv6 interface gig0/0/0/0.529 rpf-statistics
Unicast RPF drops 0

R9#ping 2001:85:6:84::1 source 2001:9000:A::9 repeat 12 timeout 1
[snip]
Success rate is 0 percent (0/12)
```

```
RP/0/0/CPU0:XRv2#show cef ipv6 interface gig0/0/0/0.529 rpf-statistics
Unicast RPF drops 12
```

We remove both the IPv4 and IPv6 trigger routes from XRv4 and CSR7 respectively and validate connectivity.

```
R9#ping 85.19.2.1 source 9.10.9.9
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/2/4 ms

R9#ping 2001:85:6:84::1 source 2001:9000:A::9
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/6/14 ms
```

3. Remotely triggered blackholes (RTBH) with communities: The concept of RTBH is identical as the previous section, except this is an alternative and more intelligent design. As we saw earlier, changing the next-hop at the originator (trigger) caused strange issues in the network. First, we had to modify BGP routers so that their best-path algorithms had a route to a bogus next-hop. In addition, we uncovered several inconsistencies between how XE and XR handle 6PE next-hops, and also some complex/interesting RPL parameterization behaviors. Rather than deal with all that unnecessary complexity, the trigger router can set a special community value that signals the ASBRs to locally change the next-hop to something resolving to null0. This ensures the BGP convergence is not affected within the core network (no changes no RRs, etc) and is more consistent with QPPB-based BGP PA and CAR discussed earlier. The only drawback includes more configuration on the ASBRs, which is an inbound RPL/route-map that matches the special community and sets the next-hop. **Using this community-based signaling method is Cisco's recommended RTBH architecture.**

We tested this feature extensively already, so this next set of tests will be limited. Every ASBR has been configured with an RPL or route-map to match the special community (85:98) and set the next-hop to something resolving to null0 (192.0.2.1). Notice that on XR, setting the 6PE next-hop to an IPv4 address is required, but on XE, you must manually compute the IPv4-compatible IPv6 address and set that in the route-map directly.

```
! CSR4 and CSR8
ip community-list standard COML_RTBH per 85:98

route-map RM_RBTH_NHOP_IN per 10
 match community COML_RTBH
 set ip next-hop 192.0.2.1
 set ipv6 next-hop ::FFFF:192.0.2.1
 set local-preference 200
route-map RM_RBTH_NHOP_IN per 10000

router bgp 85
 template peer-policy AF_IPV6_LUCAST
```

```

    route-map RM_RBTH_NHOP_IN in
template peer-policy AF_IPV4_UCAST
    route-map RM_RBTH_NHOP_IN in

! XRv1 and XRv2
route-policy RPL_RBTH_NHOP_IN($NHOP)
    if community matches-any (85:98) then
        set next-hop $NHOP
        set local-preference 200
    else
        pass
    endif
end-policy

router bgp 85
    af-group AF_IPV4_UCAST address-family ipv4 unicast
        route-policy RPL_RBTH_NHOP_IN(192.0.2.1) in
    af-group AF_IPV6_LUCAST address-family ipv6 labeled-unicast
        route-policy RPL_RBTH_NHOP_IN(192.0.2.1) in

```

The trigger routers already have static into BGP redistribution RPL/route-map constructs, so those have been extended to include tag 98, which sets community 85:98 when the route is redistributed. When ASBRs receive their routes, the next-hop modification RPL/route-map is applied inbound for any routes containing that community.

```

! CSR7
route-map RM_STATIC_BGP_IPV4 per 30
    match tag 98
    set community 85:98 no-export

route-map RM_STATIC_BGP_IPV6 per 30
    match tag 98
    set community 85:98 no-export

! XRv4
route-policy RPL_STATIC_TO_BGP($NHOP)
    if tag is 99 then
        [snip]
    elseif tag is 98 then
        set community (85:98, no-export)
    endif
end-policy

```

At this point, all we have to do is add static routes with tag 98 to either XRv4 or CSR7. In this example, CSR7 will trigger an IPv6 black-hole and XRv4 will trigger an IPv4 black-hole.

```

! XRv4
router static

```

```
address-family ipv4 unicast
  85.6.144.0/20 Null0 tag 98
```

```
! CSR7
```

```
ipv6 route 2001:85:6:84::/62 null0 tag 98
```

We will verify XRv2 and CSR8 to ensure both IPv4 and IPv6 routes are being black-holed. For brevity, I will not trace the entire recursion stack as we have done that several times. I use verbose verifications on XE and succinct ones on XR for variation. Also note that these routes have two communities; no-export keeps it within the confederation and 85:98 signals the RTBH.

```
R8#show bgp ipv4 unicast 85.6.144.0
BGP routing table entry for 85.6.144.0/20, version 452
Paths: (2 available, best #1, table default, not advertised to EBGp peer)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    192.0.2.1 from 14.14.14.14 (14.14.14.14)
      Origin incomplete, metric 0, localpref 200, valid, internal, best
      Community: 85:98 no-export
      rx pathid: 0, tx pathid: 0x0
  Refresh Epoch 5
  Local
    10.10.10.10 (metric 30) from 2.2.2.2 (2.2.2.2)
      Origin incomplete, metric 0, localpref 100, valid, internal
      Originator: 10.10.10.10, Cluster list: 2.2.2.2
      rx pathid: 0, tx pathid: 0
```

```
R8#show bgp ipv6 unicast 2001:85:6:84::/62
BGP routing table entry for 2001:85:6:84::/62, version 408
Paths: (2 available, best #2, table default, not advertised to EBGp peer)
  Not advertised to any peer
  Refresh Epoch 1
  Local
    ::FFFF:192.0.2.1 from 14.14.14.14 (14.14.14.14)
      Origin incomplete, metric 0, localpref 200, valid, internal
      Community: 85:98 no-export
      Originator: 7.7.7.7, Cluster list: 14.14.14.14
      mpls labels in/out nolabel/7013
      rx pathid: 0, tx pathid: 0
  Refresh Epoch 13
  Local
    ::FFFF:192.0.2.1 from 2.2.2.2 (2.2.2.2)
      Origin incomplete, metric 0, localpref 200, valid, internal, best
      Community: 85:98 no-export
      Originator: 7.7.7.7, Cluster list: 2.2.2.2
      mpls labels in/out nolabel/7013
      rx pathid: 0, tx pathid: 0x0
```

```
RP/0/0/CPU0:XRv2#show bgp ipv4 unicast community no-export 85:98 | begin
Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
*>i85.6.144.0/20	192.0.2.1	0	200	0	?

```
Processed 1 prefixes, 1 paths
```

```
RP/0/0/CPU0:XRv2#show bgp ipv6 unicast community no-export 85:98 | begin
Network
```

Network	Next Hop	Metric	LocPrf	Weight	Path
*>i2001:85:6:84::/62	192.0.2.1	0	200	0	?
* i	192.0.2.1	0	200	0	?

A quick data plane test clearly shows the traffic being black-holed at the ASBRs.

```
RP/0/0/CPU0:XRv3#ping 2001:85:6:84::1 source 2001:3000:c::13
[snip]
Success rate is 0 percent (0/5)
```

```
RP/0/0/CPU0:XRv3#ping 85.6.144.1 source 13.14.13.13
[snip]
Success rate is 0 percent (0/5)
```

Finally, we remove the trigger routes before continuing and ensure there is full reachability to the formerly black-holed prefixes.

```
RP/0/0/CPU0:XRv3#ping 2001:85:6:84::1 source 2001:3000:c::13
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/6/9 ms
```

```
RP/0/0/CPU0:XRv3#ping 85.6.144.1 source 13.14.13.13
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/9 ms
```

4. Backscatter traceback: This technique combines many techniques already tested. Specifically, it relies on a combination of sinkholes, RTBH, and ICMP unreachable to determine attack ingress points in the network. It is generally designed to defeat IP spoofing attacks and is another tool for SPs to secure their networks. The feature works as follows: an attacker spoofs traffic into the network, the ASBR or some other core router goes to reply, but since the source is not a routable address, it matches a default route advertised from a sinkhole. Thus, the SP's sinkholes will absorb these unreachable and by examining them, we determine which ASBR sent it (the source) and what the spoofed address was (the destination). The traceback allows an SP to quickly narrow down the ingress point of the spoofed traffic and take action (block it via ACL/RTBH, coordinate with the peer SP, etc).

Because uRPF is used to drop packets in CEF, unreachable are not generated regardless of whether they are enabled on the incoming interface (or the null0 interface). As a result, it appears that this

technique is not compatible with uRPF. For this reason, we will focus the next tests on XRv1 and CSR4 where uRPF is not configured. These routers have ICMP unreachables disabled for IPv4 and IPv6 from an earlier task, so we will enable those again (not shown, since it is a basic task). It is important to police these ICMP unreachables once they are enabled on external interfaces, which was discussed earlier.

CSR1 will be used to spoof traffic into the service provider network. CSR1 tries to guess some of AS 85's infrastructure addresses so it uses destinations within 85.0.0.0/8. These unreachables originated by the ASBR as a result of the ACL deny entry will end up in CSR3's sinkhole as it attracts all unknown traffic within 85.0.0.0/8. We rely on the inbound ACL to send the unreachables. CSR1 uses a new Loopback85 to spoof traffic from 85.111.111.111. This will be denied by the ACL with an unreachable sent towards CSR3.

```
R4#show ip cef 85.111.111.111
85.0.0.0/8
  nexthop 85.4.6.6 GigabitEthernet2.546 label 6003

R4#show ip route 85.0.0.0 255.0.0.0
Routing entry for 85.0.0.0/8
  Known via "bgp 85", distance 200, metric 0, type internal
  Last update from 3.3.3.3 05:38:11 ago
  Routing Descriptor Blocks:
  * 3.3.3.3, from 2.2.2.2, 05:38:11 ago
    Route metric is 0, traffic share count is 1
    AS Hops 0
    MPLS label: none
```

When CSR1 starts sending traffic, CSR4 denies those packets and sends unreachables into the sinkhole. Normally, a single unreachable is all that is required, even if CSR1 sent many thousands of spoofed packets within a flow. We check CSR4 for ACL matches and also check the sinkhole. The result of the FNF cache on CSR3 shows that the unreachable came from CSR4 destined to a spoofed source, which helps narrow down ingress points during a DDoS attack when there are several ASBRs in the network. Notice the source address of the unreachable generated by CSR4; it's the transit link between CSR4 and CSR6. Unfortunately, this doesn't tell us from which external interface the spoofed packets arrived. It only narrows down the ASBR. Once logged into the ASBR, however, checking the ACL counters or other classification tools can identify the proper external interface.

```
! CSR1
interface Loopback85
 ip address 85.111.111.111 255.255.255.255

R1#ping 85.19.0.1 source 85.111.111.111
[snip]
Success rate is 0 percent (0/5)

R4#show access-lists ACL_PEER_SP_IN
```

```
Extended IP access list ACL_PEER_SP_IN
 10 permit tcp host 10.1.4.1 host 10.1.4.4 eq bgp (1 match)
 20 permit tcp host 10.1.4.1 eq bgp host 10.1.4.4
 30 deny ip object-group OBJGRP_BAD_NETS object-group OBJGRP_BAD_NETS (5
matches)
 40 permit ip any any
```

```
R3#show flow monitor FLOW_MONITOR_IPV4 cache format record
[snip]
```

```
IPV4 SOURCE ADDRESS:      85.4.6.4
IPV4 DESTINATION ADDRESS: 85.111.111.111
TRNS SOURCE PORT:         0
TRNS DESTINATION PORT:    781
INTERFACE INPUT:          Gi2.535
IP DSCP:                  0x00
IP PROTOCOL:              1
```

CSR1 tries to spoof traffic again, this time from RFC1918 addressing using 172.20.1.1. Our ACL still blocks these packets as expected, and the unreachable flow into CSR7's sinkhole this time.

```
R4#clear access-list counters
```

```
R4#show ip cef 172.20.1.1
0.0.0.0/0
  nexthop 85.4.6.6 GigabitEthernet2.546 label 6002
```

```
R4#show ip route 0.0.0.0 0.0.0.0
Routing entry for 0.0.0.0/0, supernet
  Known via "bgp 85", distance 200, metric 0, candidate default path, type
internal
  Last update from 7.7.7.7 03:39:57 ago
  Routing Descriptor Blocks:
  * 7.7.7.7, from 2.2.2.2, 03:39:57 ago
    Route metric is 0, traffic share count is 1
    AS Hops 0
    MPLS label: none
```

The pings still fail from CSR1, and we check our ACL counters and sinkhole FNF cache to identify the attack.

```
R1#ping 85.19.0.1 source 172.20.1.1
[snip]
Success rate is 0 percent (0/5)
```

```
R4#show access-lists ACL_PEER_SP_IN
Extended IP access list ACL_PEER_SP_IN
 10 permit tcp host 10.1.4.1 host 10.1.4.4 eq bgp (2 matches)
```



```

20 permit tcp host 10.1.4.1 eq bgp host 10.1.4.4
30 deny ip object-group OBJGRP_BAD_NETS object-group OBJGRP_BAD_NETS (5
matches)
40 permit ip any any

```

```

R7#show flow monitor FLOW_MONITOR_IPV4 cache format record
[snip]
IPV4 SOURCE ADDRESS:      85.4.6.4
IPV4 DESTINATION ADDRESS: 172.20.1.1
TRNS SOURCE PORT:        0
TRNS DESTINATION PORT:    781
INTERFACE INPUT:         Gi2.567
IP DSCP:                  0x00
IP PROTOCOL:              1

```

The same technique can be used for IPv6 and on XR, again assuming ICMPv6 unreachables are enabled. This test will be conducted on XRv1 with XRv3 as the attacker. Because ACLs are a simple topic and I did not test this earlier, I created a very weak ACL on XRv1 just to deny some traffic so the router will generate ICMP unreachables. I am blocking traffic sourced from AS 85's address space (spoofed) and unique local addressing.

```

! XRv1
ipv6 access-list ACL_PEER_SP_IN_IPV6
 10 deny ipv6 2001:85::/32 any
 20 deny ipv6 fc00::/7 any
 30 permit ipv6 any any

```

XRv3 will be the attacker in this scenario. First, ensure XRv3 can reach an AS 85 destination from valid sources to check for routing issues.

```

RP/0/0/CPU0:XRv3#ping 2001:85:19:2::1 source 2001:3000:b::13
[snip]
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/6/19 ms

```

XRv3 uses Loopback100 which has two usable IPv6 addresses for spoofing, FC00::13 and 2001:85::13. A quick CEF check on XRv1 indicates the unreachables sent back to these bogus sources will go to the CSR7 and CSR3 sinkholes, respectively. We also check the ACL to see that nothing has been blocked yet.

```

RP/0/0/CPU0:XRv1#show cef ipv6 FC00::13
::/0, version 1314, proxy default, internal 0x5000011 0x0 (ptr 0xa13b8074)
[1], 0x0 (0x0), 0x208 (0xa152c410)
Prefix Len 0, traffic index 0, precedence n/a, priority 4
via ::ffff:7.7.7.7, 3 dependencies, recursive [flags 0x6000]
[snip]

```

```

RP/0/0/CPU0:XRv1#show cef ipv6 2001:85::13

```

```
2001:85::/32, version 1149, internal 0x5000001 0x0 (ptr 0xa13b9bf4) [1], 0x0
(0x0), 0x208 (0xa152c460)
Prefix Len 32, traffic index 0, precedence n/a, priority 4
via ::ffff:3.3.3.3, 3 dependencies, recursive [flags 0x6000]
[snip]
```

```
RP/0/0/CPU0:XRv1#show access-lists ipv6 ACL_PEER_SP_IN_IPV6
ipv6 access-list ACL_PEER_SP_IN_IPV6
 10 deny ipv6 2001:85::/32 any
 20 deny ipv6 fc00::/7 any
 30 permit ipv6 any any (3 matches)
```

XRv3 starts sending traffic into the network using spoofed addresses. Because the source originates from an AS 85 network, it will end up in CSR3's sinkhole. XRv1 also has ACL matches for it. The nice thing about XR is that the source address of the unreachable is the interface upon which it was generated (the link facing XRv3). This is more valuable than XE where the IP address source was the outgoing interface (in the core). If XRv1 had 100 eBGP peers, this narrows it down significantly. IP protocol number 58 refers to IPv6-ICMP and is also revealed in the FNF cache.

```
RP/0/0/CPU0:XRv3#ping 2001:85:19:2::1 source 2001:85::13
[snip]
Success rate is 0 percent (0/5)
```

```
RP/0/0/CPU0:XRv1#show access-lists ipv6 ACL_PEER_SP_IN_IPV6
ipv6 access-list ACL_PEER_SP_IN_IPV6
 10 deny ipv6 2001:85::/32 any (5 matches)
 20 deny ipv6 fc00::/7 any
 30 permit ipv6 any any (4 matches)
```

```
R3#show flow monitor FLOW_MONITOR_IPV6 cache format record
[snip]
```

```
IPV6 SOURCE ADDRESS:      FD00:10:1:3::11
IPV6 DESTINATION ADDRESS: 2001:85::13
TRNS SOURCE PORT:         0
TRNS DESTINATION PORT:    257
INTERFACE INPUT:          Gi2.535
IP DSCP:                   0x00
IP PROTOCOL:               58
```

The last example involves testing XRv3's other spoofed address. When XRv1 blocks it, the unreachable messages that it generates will follow the default route into CSR7's sinkhole.

```
RP/0/0/CPU0:XRv3#ping 2001:85:19:2::1 source fc00::13
[snip]
Success rate is 0 percent (0/5)
```

```
RP/0/0/CPU0:XRv1#show access-lists ipv6 ACL_PEER_SP_IN_IPV6
```

```

ipv6 access-list ACL_PEER_SP_IN_IPV6
 10 deny ipv6 2001:85::/32 any (5 matches)
 20 deny ipv6 fc00::/7 any (5 matches)
 30 permit ipv6 any any (5 matches)

R7#show flow monitor FLOW_MONITOR_IPV6 cache format record
[snip]
IPV6 SOURCE ADDRESS:      FD00:10:1:3::11
IPV6 DESTINATION ADDRESS: FC00::13
TRNS SOURCE PORT:        0
TRNS DESTINATION PORT:    257
INTERFACE INPUT:         Gi2.567
IP DSCP:                  0x00
IP PROTOCOL:              58

```

5. Lawful intercept: The LI feature is used to comply with Government regulations and laws that require a carrier to tap into customer/transit traffic. All it really entails is enabling the feature via AAA, then allowing the routing to be controlled via SNMPv3. An authorized user will use SNMPv3 to read/write LI information using an external device as needed. I configured this on CSR7 because AAA was already enabled there. SNMP was discussed in detail earlier in this section, so the security settings for this are low for demonstration purposes. Normally one would want LI permissions to be very high since misuse of LI permissions is highly undesirable. One caveat of this feature is that SNMP traps normally sent to port 162 must be sent to port 161. The list of traps specified are in accordance with the Cisco configuration guide and RFC 1157. The SNMP-server target address for traps is the LI mediation device. The LI mediation device is provided by a third-party vendor that handles most of the processing and does several things: provides interface to setup and provision LI, generates requests to other network devices to use LI, and converts traffic intercept by LI into the format required by a specific law enforcement agency (LEA). Traffic is sent from the mediation device to the LEA without the target's knowledge.

```

! CSR7
aaa new-model
aaa intercept
snmp-server group LI_GROUP v3noauth read LI_GROUP write LI_GROUP
snmp-server view LI_VIEW ciscoIpTapMIB included
snmp-server view LI_VIEW cisco802TapMIB included
snmp-server view LI_VIEW ciscoTap2MIB included
snmp-server user LI_USER LI_GROUP v3 auth md5 LI_PASSWORD
snmp-server enable traps snmp authentication linkdown linkup coldstart
warmstart
snmp-server host 10.10.10.10 version 3 noauth LI_USER udp-port 161

```

LI does not appear to be supported on XRv, but we can configure the SNMPv3 parameters anyway.

```

RP/0/0/CPU0:XRv2 (config) #law?
^

```

```
% Invalid input detected at '^' marker.
```

```
! XRv2
snmp-server host 10.10.10.10 traps version 3 noauth LI_USER udp-port 161
snmp-server user LI_USER LI_GROUP v3 auth md5 encrypted
072308737E282A36203D3928
snmp-server view LI_VIEW ciscoTap2MIB included
snmp-server view LI_VIEW ciscoUserConnectionMIB included
snmp-server group LI_GROUP v3 noauth read LI_GROUP write LI_GROUP
snmp-server traps snmp linkup
snmp-server traps snmp linkdown
snmp-server traps snmp coldstart
snmp-server traps snmp warmstart
snmp-server traps snmp authentication
```

The configuration won't commit since the XRv does not appear to have the proper MIBs handy, but the configuration above is valid. The remainder of the configuration, minus the two MIB commands below, is committed for reference.

```
! XRv2
snmp-server view LI_VIEW ciscoTap2MIB included
!!% A verifier or EDM callback function returned: 'invalid argument (NULL
pointer?): SNMP oid subtree ciscoTap2MIB is not part of a known MIB module

snmp-server view LI_VIEW ciscoUserConnectionMIB included
!!% A verifier or EDM callback function returned: 'invalid argument (NULL
pointer?): SNMP oid subtree ciscoUserConnectionMIB is not part of a known MIB
module
end
```

6. BGP Flowspec: BGP Flowspec was defined in its own chapter where I discussed the basic capabilities. It is generally used to provide finer granularity to RTBH and QPPB allowing us to police, remark, redirect to VRF, or drop traffic. We can also look at details such as fragments, layer 4 information, and ICMP type/code on some platforms (CRS, but not ASR9K). I will quickly enable BGP flowspec for the IPv4 and IPv6 address families throughout the topology. Note that XE 3.13.2S only provides RR support for BGP flowspec, so the only XE router being configured with BGP flowspec AF will be CSR2. XE 3.15S introduced support for flowspec in the data plane but is not examined here.

```
! CSR2
router bgp 85
address-family ipv4 flowspec
  neighbor 11.11.11.11 activate
  neighbor 11.11.11.11 route-reflector-client
  neighbor 12.12.12.12 activate
  neighbor 12.12.12.12 route-reflector-client
address-family ipv6 flowspec
  neighbor 11.11.11.11 activate
```

```

neighbor 11.11.11.11 route-reflector-client
neighbor 12.12.12.12 activate
neighbor 12.12.12.12 route-reflector-client

! XRv14
router bgp 85
address-family ipv4 flowspec
address-family ipv6 flowspec

af-group AF_IPV4_FLOW address-family ipv4 flowspec
route-reflector-client
af-group AF_IPV6_FLOW address-family ipv6 flowspec
route-reflector-client

neighbor 11.11.11.11
address-family ipv4 flowspec
use af-group AF_IPV4_FLOW
address-family ipv6 flowspec
use af-group AF_IPV6_FLOW

neighbor 12.12.12.12
address-family ipv4 flowspec
use af-group AF_IPV4_FLOW
address-family ipv6 flowspec
use af-group AF_IPV6_FLOW

! XRv1 and XRv2
router bgp 85
address-family ipv4 flowspec
address-family ipv6 flowspec

neighbor 2.2.2.2
address-family ipv4 flowspec
address-family ipv6 flowspec

neighbor 14.14.14.14
address-family ipv4 flowspec
address-family ipv6 flowspec

```

Configuration-wise, the “client”, or receiver of a BGP flowspec policy, does not need any special BGP configuration other than a flowspec-enabled peer. In most designs, this would be the ASBRs and RRs. Up to this point, that is all we have configured, and since flowspec is a negotiated AF, we should verify successful negotiation. Doing this on the route-reflectors makes the most sense because they can tell us if the negotiation was bidirectionally successful (advertised and received).

```

R2#show bgp neighbors 11.11.11.11 | include Flowspec:
    Address family IPv4 Flowspec: advertised and received
    Address family IPv6 Flowspec: advertised and received

```

```

R2#show bgp neighbors 12.12.12.12 | include Flowspec:
    Address family IPv4 Flowspec: advertised and received
    Address family IPv6 Flowspec: advertised and received

RP/0/0/CPU0:XRv4#show bgp neighbor 11.11.11.11 | include Flowspec:
    Address family IPv4 Flowspec: advertised and received
    Address family IPv6 Flowspec: advertised and received

RP/0/0/CPU0:XRv4#show bgp neighbor 12.12.12.12 | include Flowspec:
    Address family IPv4 Flowspec: advertised and received
    Address family IPv6 Flowspec: advertised and received

```

We will use XRv4 as our flowspec policy “server” for IPv4 and IPv6 since it is central to the network. CSR2 won’t really be doing anything in this capacity but we at least show that its flowspec RR capability works. The flowspec policy configuration is very similar to MQC except the class-maps are of type “traffic”. Additionally, you cannot nest class maps, so the policies are somewhat limited in their hierarchy at this time. Below are some sample class maps with a policy-map to define their actions. I used variety in my matching statements as a demonstration. We use a combination of network address, TCP port, and fragment-type matching techniques. The policy performs a combination of QoS marking, policing, dropping, and VRF redirecting actions.

```

! XRv4
class-map type traffic match-all CMAP_TRAF_AS_9_13_TO_US
    match destination-address ipv4 85.0.0.0 255.0.0.0
    match source-address ipv4 13.0.0.0 255.0.0.0

class-map type traffic match-all CMAP_TRAF_POLICE_TELNET
    match protocol tcp
    match destination-port 23

class-map type traffic match-all CMAP_TRAF_DROP_FRAGMENTS
    match fragment-type is-fragment

class-map type traffic match-all CMAP_TRAF_AS_9_13_TO_US_IPV6
    match destination-address ipv6 2001:85::/32
    match source-address ipv6 2001:9000::/32

policy-map type pbr PMAP_PBR_FLOWSPEC
    class type traffic CMAP_TRAF_AS_9_13_TO_US
        set dscp cs1

    class type traffic CMAP_TRAF_AS_9_13_TO_US_IPV6
        redirect ipv6 nexthop vrf ANALYZE

    class type traffic CMAP_TRAF_DROP_FRAGMENTS
        drop

```

```
class type traffic CMAP_TRAF_POLICE_TELNET
  police rate 1024000 bps
```

XRv does not support data-plane flowspec operations since binding this flowspec policy to BGP requires the flowspec process to exist. It doesn't, and neither do its show commands; this is because XRv represents an RSP with no linecards. The idea is that this policy could be distributed to clients, like the ASBRs XRv1 and XRv2, and ultimately programmed into their hardware linecards as a remote-PBR type of operation. The feature is also VRF aware, as seen in the policy-map actions, assuming it works.

```
RP/0/0/CPU0:XRv4(config)#flo?
      ^
% Invalid input detected at '^' marker.

RP/0/0/CPU0:XRv4#show flo?
      ^
% Invalid input detected at '^' marker.
```

7. BGPsec: Most of the configuration relating to BGPsec revolves around the Resource Public Key Infrastructure (RPKI) server configuration. Cisco routers are generally clients in this architecture. Having this RPKI information allows the routers to perform Origin AS validation to determine what routes are legitimate and which are not. Bestpath-wise, valid routes are always preferred over invalid routes and those without any AS-validation attributes. By default, invalid routes are not candidate for bestpath at all, but this can be disabled. Routes with no attributes are still valid but cannot be considered for bestpath or multipath if there is a valid AS-origin route. Rather than have every single router connect directly to the third-party RPKI server, Cisco routers can distribute this information inside iBGP using extended communities. **This can never be exchanged over eBGP, similar to the cost-community, as it is non-transitive.** In this example, CSR2 and XRv4 are logical choices (RRs) to connect directly to our notional RPKI server, and then distribute that information to other peers via extended-communities. XE devices need to explicitly enable extended-community exchange between each other, while XR does this automatically for iBGP peers (configuration not shown). The RPKI server configuration is shown for CSR2 and XRv4, assuming we have IPv4 and IPv6 RPKI servers in our network. The routers also distribute the AS-origin information to their iBGP peers inside extended-communities for both IPv4 unicast and IPv6 address-families. XR also supports using SSH to communicate with the RPKI server for extra security, but XE does not.

```
! CSR2
router bgp 85
  bgp rpki server tcp ::10:10:99:99 port 5006 refresh 3600
  bgp rpki server tcp 10.10.99.99 port 5000 refresh 3600

  template peer-policy AF_IPV4_UCAST
    announce rpki state
  template peer-policy AF_IPV6_LUCAST
    announce rpki state
```

```

!XRv4
router bgp 85
  rpki server 10.10.99.99
    transport tcp port 5000
    refresh-time 3600

  rpki server ::10:10:99:99
    username RPKI_XRV4
    password RPKI_XRV4
    transport ssh port 5022
    refresh-time off

  bgp origin-as validation signal ibgp
  address-family ipv4 unicast
    bgp origin-as validation signal ibgp

  address-family ipv6 unicast
    bgp origin-as validation signal ibgp

```

I do not show the basic RPKI configuration on all the other iBGP routers within AS 85, but I have enabled transmission of both standard and extended communities, as well as RPKI state propagation. This is somewhat similar to the basic BGP flowspec configuration where all routers must be configured to support it, at a minimum. We will perform some basic verification on both routers to ensure the configuration was successful. Since the RPKI servers do not exist, we won't see any session statistics or any prefixes from them.

```

R2#show bgp ipv4 unicast rpki servers
BGP SOVC neighbor is ::10:10:99:99/5006 connected to port 5006
Flags 0, Refresh time is 3600, Serial number is 0, Session ID is 0
InQ has 0 messages, OutQ has 0 messages, formatted msg 0
Session IO flags 0, Session flags 4000
Neighbor Statistics:
  Prefixes 0
  Connection attempts: 0
  Connection failures: 0
  Errors sent: 0
  Errors received: 0
  No active TCP connection

BGP SOVC neighbor is 10.10.99.99/5000 connected to port 5000
[snip]
  No active TCP connection

```

```

RP/0/0/CPU0:XRv4#show bgp rpki server summary

```

Hostname/Address (IPv4/IPv6)	Transport	State	Time	ROAs
10.10.99.99	TCP:5000	CONNECT	00:00:01	N/A
::10:10:99:99	SSH:5022	CONNECT	00:00:01	N/A


```
RP/0/0/CPU0:XRv4#show bgp rpki summary
RPKI cache-servers configured: 2
RPKI global knobs
  Origin-AS validation is ENABLED globally
  Origin-AS validity WILL NOT affect bestpath selection globally
  Origin-AS validity signaling towards iBGP is ENABLED globally
RPKI database
  Total IPv4 net/path: 0/0
  Total IPv6 net/path: 0/0
```

Looking at CSR7, we see that all of the local routes (originating from our own AS) are automatically considered valid. The “V” in the leftmost column means valid; using a regex to only look for local routes helps prove this.

```
R7#show bgp ipv4 unicast quote-regexp "^$" | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
V*> 0.0.0.0        7.7.7.7          0         32768 ?
V* i 10.10.10.85/32 10.10.10.10       0        100      0 ?
V*>i              10.10.10.10       0        100      0 ?
V* i 85.0.0.0      3.3.3.3          0        100      0 i
V*>i              3.3.3.3          0        100      0 i
V* i 85.6.128.0/22 10.10.10.10       0        100      0 ?
V*>i              10.10.10.10       0        100      0 ?
V* i 85.6.132.0/22 10.10.10.10       0        100      0 ?
V*>i              10.10.10.10       0        100      0 ?
```

Examining one of the paths, we see some additional information. Each path has a new extended community of 0x4300, which represents the RPKI state. The last digit represents the validity: 0 means valid, 1 means not-found, and 2 means invalid. In this case, the digit is zero because it is valid, and in case you don’t remember the codes, the output tells you the RPKI state in plain English also.

```
R7#show bgp ipv4 unicast 85.6.128.0/22 bestpath
BGP routing table entry for 85.6.128.0/22, version 174
Paths: (2 available, best #2, table default)
  Not advertised to any peer
  Refresh Epoch 11
  Local
    10.10.10.10 (metric 20) from 2.2.2.2 (2.2.2.2)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: 0x4300:0:0
      Originator: 10.10.10.10, Cluster list: 2.2.2.2
      path 7F8D84CAEE58 RPKI State valid
      rx pathid: 0, tx pathid: 0x0
```

If we look at any path from another AS, we see the path is marked with an “N” for not-found. Because the path is external, we cannot guarantee the AS path is accurate. Peer ASes could have easily injected false AS path information, and this is the problem BGPsec/RPKI attempt to solve.

```
R7#show bgp ipv4 unicast quote-regexp "^[0-9]+$" | begin Network
      Network      Next Hop      Metric LocPrf Weight Path
N* i 1.10.0.0/16    4.4.4.4          0      100      0 65537 ?
N*>i              4.4.4.4          0      100      0 65537 ?
N* i 1.11.0.0/16    4.4.4.4          0      100      0 65537 ?
N*>i              4.4.4.4          0      100      0 65537 ?
N* i 1.12.0.0/16    4.4.4.4          0      100      0 65537 ?
N*>i              4.4.4.4          0      100      0 65537 ?
```

Looking deeper, we see the same extended community with the last digit as a 1, meaning “not-found”. The RPKI state also makes this clear.

```
R7#show bgp ipv4 unicast 1.11.0.0/16 bestpath
BGP routing table entry for 1.11.0.0/16, version 230
Paths: (2 available, best #2, table default)
  Not advertised to any peer
  Refresh Epoch 11
  65537
    4.4.4.4 (metric 20) from 2.2.2.2 (2.2.2.2)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: 0x4300:0:1
      Originator: 4.4.4.4, Cluster list: 2.2.2.2
      path 7F8D84CAEC58 RPKI State not found
      rx pathid: 0, tx pathid: 0x0
```

Recalling the rules about best-path selection, valid paths are always preferred over not-found or invalid ones. We do not have a mechanism to test this since all paths with any non-local AS-path will be “not-found”, and any local paths will be valid. However, we can ignore this behavior and let BGP do its normal calculation to ignore these values. As a compromise, we can match the RPKI states in a route-map, and then adjust BGP parameters. A common approach would be to increase local-preference for valid routes and reduce it for invalid ones, which allows invalid routes to be considered for best-path as a last resort. I illustrate this technique on CSR10; the path-selection here is less relevant and I don’t want to break the network testing a feature half-heartedly. In this example, I will adjust local-preference as follows:

- a. Valid routes will all get local-preference 150
- b. Not-found routes will get local-preference 95
- c. Invalid routes will get local-preference 50

The reason I adjust “not-found” routes is because I want to demonstrate that the route-map actually works. I will test this on 6PE routes specifically. A quick check shows the default local-preference of 100 for this route from AS 1.1. Nothing has been adjusted yet.

```

R10#show bgp ipv6 unicast 2001:1C00:A::/48
BGP routing table entry for 2001:1C00:A::/48, version 3815
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  65537
    ::FFFF:4.4.4.4 (metric 30) from 14.14.14.14 (14.14.14.14)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: 0x4300:0:1
      Originator: 4.4.4.4, Cluster list: 14.14.14.14
      mpls labels in/out nolabel/4028
      path 7F760E6DB020 RPKI State not found
      rx pathid: 0, tx pathid: 0x0

```

The configuration on CSR10 is below and is very straightforward.

```

! CSR10
route-map RM_SET_LPREF_RPKI permit 10
  match rpki valid
  set local-preference 150
route-map RM_SET_LPREF_RPKI permit 20
  match rpki not-found
  set local-preference 95
route-map RM_SET_LPREF_RPKI permit 30
  match rpki invalid
  set local-preference 50

router bgp 85
  template peer-policy AF_IPV6_LUCAST
  route-map RM_SET_LPREF_RPKI in

```

Checking the BGP RIB again for 6PE routes, we see the not-found routes have a new local-preference. This proves that the route-map “match rpki” command works.

```

R10#show bgp ipv6 unicast 2001:1C00:A::/48
BGP routing table entry for 2001:1C00:A::/48, version 3867
Paths: (1 available, best #1, table default)
  Not advertised to any peer
  Refresh Epoch 1
  65537
    ::FFFF:4.4.4.4 (metric 30) from 14.14.14.14 (14.14.14.14)
      Origin incomplete, metric 0, localpref 95, valid, internal, best
      Extended Community: 0x4300:0:1
      Originator: 4.4.4.4, Cluster list: 14.14.14.14
      mpls labels in/out nolabel/4028
      path 7F760E6DB020 RPKI State not found
      rx pathid: 0, tx pathid: 0x0

```

Additionally, we can instruct CSR10 to allow invalid-prefixes as a last resort, given the route-map designed to make them less preferable. This is applied to IPv6 AFI given the configuration below. For completeness, CSR10 will completely ignore the BGPsec feature for the IPv4 AF from the perspective of the best-path selection process; it still retains the “announcement” configurations so that it can support other peers using BGPsec. Manipulating the best-path algorithm in this way can give variable levels of importance to the RPKI state for a prefix.

```
! CSR10
router bgp 85
  address-family ipv6 unicast
    bgp bestpath prefix-validate allow-invalid
  address-family ipv4 unicast
    bgp bestpath prefix-validate disable
```

For completeness, I created an RPL on XRv4 to do the same thing as the route-map did on CSR10, but I did not apply it anywhere. This is for reference only to compare the XE/XR configurations. For variety, rather than match invalid at the end, I used an ordinary else statement. It has the same effect in this case, since there are only three possible states currently.

```
! XRv4
route-policy RPL_SET_LPREF_RPKI
  if validation-state is valid then
    set local-preference 150
  elseif validation-state is not-found then
    set local-preference 95
  else
    set local-preference 50
  endif
end-policy
```

XR allows you to create Route Origin Authorizations (ROAs) that associate a prefix with an AS number. Normally, the authoritative RPKI server would do this and advertise it to routers via BGP. Here, you specify the aggregate prefix, and then the longer prefix length allowed (sort of like a prefix-list), along with the AS. These fields comprise the “resources” with an ROA. XRv4 introduces 85.0.0.0/8 with a max prefix length of 24, originated in AS 85. This means the AS can originate 85.0.128.0/23, 85.0.130.0/24, but not 85.0.131.0/25, since shorter prefixes are generally undesirable on the Internet. XRv4 also validates 2001:85::/32 in the same manner with a maximum prefix length of 96.

```
! XRv4
router bgp 85
  rpk route 85.0.0.0/8 max 24 origin 85
  rpk route 2001:85::/32 max 96 origin 85
```

```
RP/0/0/CPU0:XRv4#show bgp rpk table ipv4
```

Network	Maxlen	Origin-AS	Server
85.0.0.0/8	24	85	0.0.0.0

```
RP/0/0/CPU0:XRv4#show bgp rpki table ipv6
```

Network	Maxlen	Origin-AS	Server
2001:85::/32	96	85	0.0.0.0

I also add some eBGP prefixes to XRv1, specifically encompassing routes from CSR1 and XRv3. The entry for CSR1 is invalid intentionally (notice the incorrect ASN highlighted) while the entry for XRv3 is correct. We can check the RPKI table for these static entries. We would expect that this would validate prefixes from XRv3 and invalidate them from CSR1. This table typically represents a list of prefixes learned from a server. That is why using extended-communities to distribute RPKI state is valuable from a scalability perspective; there is no new AFI/SAFI or negotiated capabilities for RPKI specifically.

```
! XRv1
router bgp 85
  rpki route 1.0.0.0/8 max 20 origin 589837
  rpki route 13.0.0.0/8 max 20 origin 589837
```

```
RP/0/0/CPU0:XRv1#show bgp rpki table ipv4
```

Network	Maxlen	Origin-AS	Server
1.0.0.0/8	20	589837	0.0.0.0
13.0.0.0/8	20	589837	0.0.0.0

We see that the path from CSR1 was invalid because the originating AS is 65537 (1.1) and not 589837 (9.13) like we configured. This is the first time we have seen “valid” or “invalid” states for eBGP prefixes as they would normally be “not-found”.

```
RP/0/0/CPU0:XRv1#show bgp ipv4 unicast 1.12.0.0/16
[snip]
Not advertised to any peer
589837 65537
  10.1.3.13 from 10.1.3.13 (13.13.13.13)
    Origin incomplete, localpref 100, valid, external, group-best
    Received Path ID 0, Local Path ID 0, version 0
    Origin-AS validity: invalid
[snip]
```

Alternatively, the paths from XRv3 are now marked as valid.

```
RP/0/0/CPU0:XRv1#show bgp ipv4 unicast 13.12.0.0/16
[snip]
589837
  10.1.3.13 from 10.1.3.13 (13.13.13.13)
    Origin incomplete, metric 0, localpref 100, valid, external, best,
group-best
    Received Path ID 0, Local Path ID 1, version 226
    Origin-AS validity: valid
```

A quick verification on CSR7 shows that this validity was properly signaled throughout the iBGP network. The invalid path on XRv1 was not even considered for best-path, so everyone else will see this network as not-found from other ASBRs that advertised it. Adjusting the RPKI validity of a route can be used to manipulate the BGP best-path selection process if other attributes cannot be used, although this is highly undesirable.

```
R7#show bgp ipv4 unicast 13.12.0.0/16 bestpath
BGP routing table entry for 13.12.0.0/16, version 270
Paths: (2 available, best #2, table default)
  Not advertised to any peer
  Refresh Epoch 11
  589837
    11.11.11.11 (metric 30) from 2.2.2.2 (2.2.2.2)
      Origin incomplete, metric 0, localpref 100, valid, internal, best
      Extended Community: 0x4300:0:0
      Originator: 11.11.11.11, Cluster list: 2.2.2.2
      path 7F8D84CAF6D8 RPKI State valid
      rx pathid: 0, tx pathid: 0x0
```