BBN Report 5925

TOPS-20 Internetworking

John DelSignore, Jr.
Charles W. Lynn, Jr.

Bolt Beranek and Newman, Inc.
50 Moulton Street
Cambridge MA 02238

May 1985

□

This document describes the TOPS-20 Internetworking software, both that developed by BBN under contract to the Defense Advanced Research Projects Agency (DARPA) and that developed by DEC, as it existed at BBN in May 1985. This code has enhancements in algorithms, numerous bug fixes, better error reporting between protocol layers, additional flow control mechanisms, and increased monitoring and diagnostic capabilities. In addition, the multinet layer has been parameterized to simplify the inclusion of support for additional protocol suites, network technologies, and device drivers; only the site-dependent configuration file (STG.MAC) needs to be changed.

Specific improvements include:

TCP:

- All TCP options are supported.
- Applications may specify the transmission timeout action.
- Automatic support for IP options.
- Better error messages are available from lower protocol layers.
- Connection tracing may be requested when a connection is opened.
- Flow control information from lower protocol layers is utilized.
- Half-open connections are detected; associated jobs are detached.
- ICMP Destination Unreachable, Redirect, and Source Quench messages are processed.
- More graceful connection closing may be specified.
- Packet generation is smoother.
- Packets are repacketized before retransmission.
- The JFN interface is more fully integrated (parsing, persist, PSI, etc.).
- There is a limit on number of outstanding and queued packets.
- Unnecessary retransmissions by other end are discouraged.

TELNET:

- Negotiations are available to application programs.
- Subnegotiations may be sent and received.

IP:

- All IP options are supported.

- Application access to the checksum function has been provided.
- Datagram protocols may send and receive ICMP error messages.

Multinet:

- A loopback device has been included (replacing the IP-specific bypass).
- Forwarding of packets between interfaces can be restricted.
- Multiple interfaces are supported and in regular use.
- Multiple protocol suites can be used over a single interface.
- The domain code from ISI is being integrated to GTHST%.

1822:

- Multiple connections are supported.
- Network error messages are passed to higher protocol layers.
- The RFNM counting algorithms have been made more robust.

Diagnostics and Monitoring:

- Access to ICMP diagnostic datagrams is available at the IP level.
- Diagnostic and testing utilities are available.
- Extensive packet tracing and listing facilities are included.
- Monitoring data can easily be collected and reports written.

For completeness, portions of DEC's Release 6 functional specification, TCPIP-SPEC.DOC [1], have been included with notations identifying those features which DEC has not yet fully implemented. This software, developed under DARPA sponsorship, has been given to DEC so that it might be included in the standard TOPS-20 release if DEC chooses. DEC is in no way committed to accept, distribute, or support the functionality described herein.

We would like to thank the users for their patience and for being unknowing participants in our experiments. Special recognition is also due to Bob Basch, Barbara Dumas, Lisa Flynn, Janet LeBlond, Linda Nidle, and Dan Tappan for their help in making this all possible.

Table of Contents

Introduction

This manual is a collection of much of the documentation about the internetworking facilities available in the TOPS-20 operating system, as developed by BBN for DARPA. It contains sections intended for several different classes of readers.

Part 1, Network Application Writer's Guide, will be of interest to those developing network applications. It contains a general overview of the DARPA protocol suite, including IP, ICMP, TCP, TELNET, and descriptions of the JSYSi used to access them. Also of interest may be the descriptions of the specific algorithms used by those protocols.

Part 2, Installation Guide, is intended for the site administrator. It describes the steps necessary to configure the internetworking software on a TOPS-20.

Part 3, Maintenance Guide, is intended for system development or maintenance personnel. It contains descriptions of the multinet layer, which provides a standard interface for both protocol suites on one side and different physical network technologies on the other.

Conventions used in this manual follow those in the TOPS-20 Monitor Calls Reference Manual [2].

Since the interfaces are still evolving, application writers should use the symbols defined in the standard MONSYM library and used in this document, instead of the symbols' numeric value. Use of symbols should reduce program maintenance if a value changes in a subsequent software release.

1. DARPA Protocol Suite

This section gives a brief overview of the DARPA-developed internetworking protocols included in the TOPS-20 monitor:

- 1822 Host to IMP Protocol [3],
- Internet Protocol [4],
- Internet Control Message Protocol [5],
- Transmission Control Protocol [6],
- Telnet [7],

and related network functions. (Other protocols within the suite which are implemented as applications, such as

- File Transfer Protocol (FTP) [8] and
- Simple Mail Transfer Protocol (SMTP) [9]

are not described.) These protocols are available on several computer systems: HP3000, IBM, MULTICS, PDP-11, TOPS-20, UNIX, VAX, and several minicomputers.

Figure 1 shows the relationship and layering of the DARPA protocols and the other network layers within TOPS-20 below them.

The 1822 Protocol is used for communication between the TOPS-20 Monitor and an Internet Message Processor (IMP) attached to an "1822 Network" (e.g., ARPANet and MilNet). 1822 is present in a system only if the system is connected to such a network.

The Internet Protocol (IP) is used in interconnected systems of packet-switched computer communications networks. The IP allows the transmission and routing of variable length blocks of data (datagrams) from sources to destinations. Sources and destinations are hosts identified by fixed-length addresses. The Internet Protocol is not designed to be absolutely reliable: some packets may not be delivered and some packets may be duplicated. The packets may arrive in a different order from that in which they were sent.

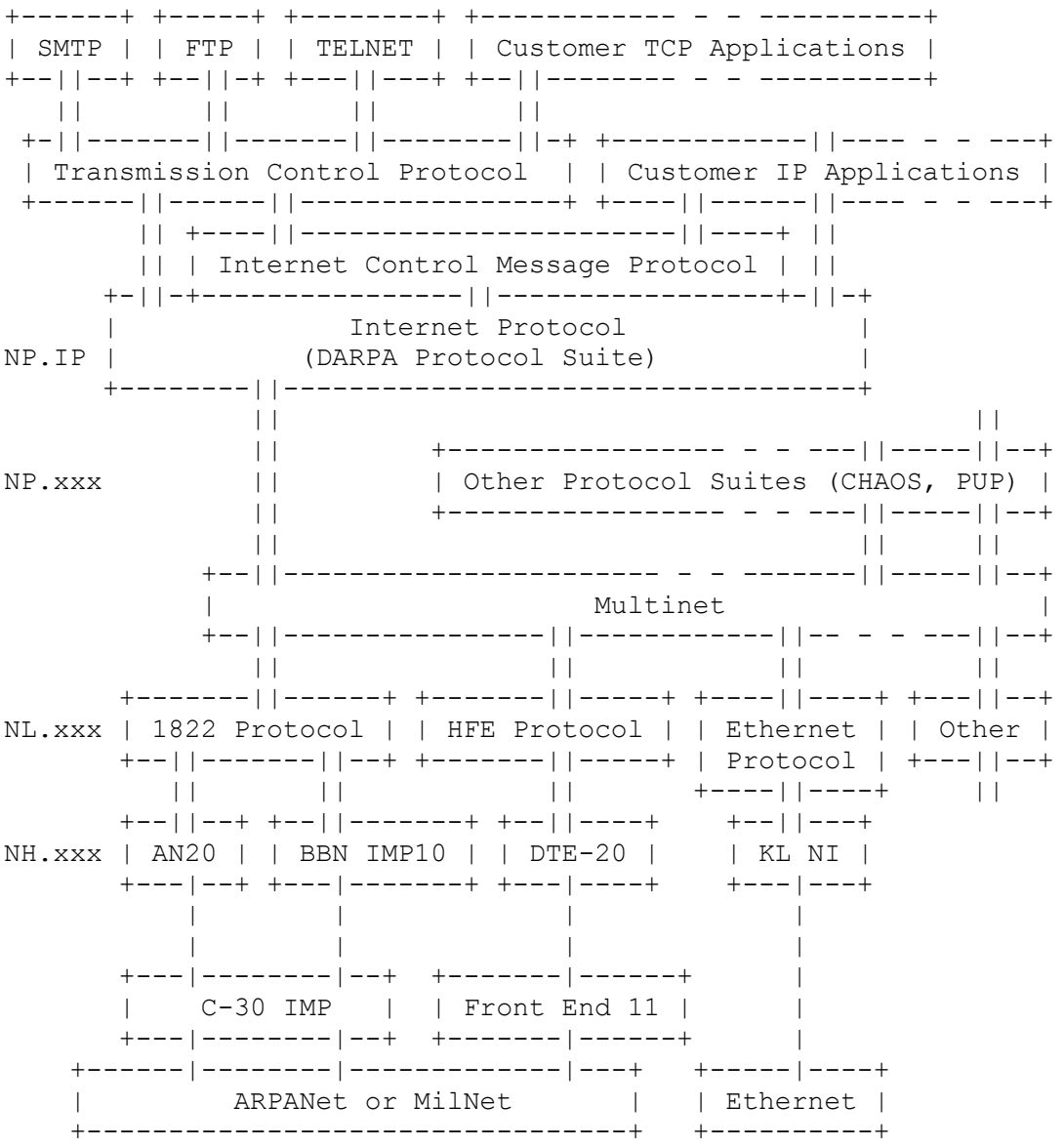


Figure 1. Protocol Layers

The Internet Control Message Protocol (ICMP) is used to:

- o report problems in the network communications environment,
- o report errors in datagram processing,
- o update routing information,
- o provide flow control information, and
- o provide a few diagnostic functions.

An application's interaction with ICMP is limited.

The Transmission Control Protocol (TCP) uses the Internet Protocol to provide application with a highly reliable, multiplexed, full-duplex, connection-based, host-to-host communications service. TCP is not intended for applications where time is critical, such as digitized speech, or broadcast or multicast applications that require communication between more than two processes.

The Telnet Protocol uses TCP to implement Network Virtual Terminals (NVTs). They provide a common basis for supporting different terminals over the internet.

Application programs may interface to the protocol suite at either the IP or ICMP level through the User Queue facility, or at the TCP or Telnet level. Three common applications which interface at the TCP level are User Telnet, a remote terminal access facility, FTP, a file transfer facility, and SMTP, the ARPANet mail delivery protocol. Name and time servers are examples of applications which interface at the IP level.

These protocols and their interfaces evolved over several years. Subsequent sections describe the DEC developed TCP: device JSYS interface and the User Queue JSYSi. The goal of the DEC developed interface is to be consistent with other TOPS-20 file system interfaces. (The old BBN TCP JSYS interface, which is being phased out is also described; it supports functionality which is not yet available in the TCP: interface.)

2. TCP

TCP allows two processes to create a connection to exchange zero or more octets of data (an octet consists of 8 data bits.) A connection is uniquely specified by the 48-bit identifiers of its local and foreign ends. The 48-bit identifier is composed of a 32-bit host address, which identifies the host, and a 16-bit port number, which identifies a process at that host. Since connections are full-duplex, each end may send and receive data.

TCP data transfer has high reliability: the headers and data octets are checksummed. If a packet is received and its checksum is not correct, it is discarded.

Each octet of data is conceptually assigned a 32-bit sequence number. The sequence numbers are used by the receiver to discard duplicated packets and to detect lost or out-of-sequence packets. Receipt of data is acknowledged by returning to the sender the sequence number of the next octet needed. This acknowledgement implies that all lower numbered octets have been correctly received and delivered (placed into the JFN interface's internal buffer, into a TVT's TTY buffer, or into a BBN JSYS interface application's buffer).

The sender continues to retransmit data until it is acknowledged. The retransmission rate is an application specified parameter. If data is unacknowledged for an application specified period of time, a program interrupt notifies the application of a Transmission Timeout error.

TCP automatically provides flow control for each connection. The data receiving TCP periodically notifies the sending TCP of the number of additional data octets it will accept (the "window"). This number is based on the amount of buffer space available at the receiving TCP, packet sizes, and other factors. The data sending TCP may not exceed this window.

The data sending TCP is permitted to internally buffer data octets from the application so that packets may be formed which will use the network resources efficiently. If the application requires that all of the octets it has given to the TCP be delivered, e.g., the application must wait for a reply, it must notify TCP to PUSH the data through to the receiving application, either by using the SOUTR% JSYS, or the .TCPSH function of the TCOPR% JSYS.

TCP also provides an asynchronous signal mechanism called URGENT. Urgent implies that there is "significant" information in the data stream. An urgent signal is associated with a data octet using the .TCSUD function of the TCOPR% JSYS. Its sequence number is communicated to the receiving TCP. The receiving application should look through the data stream to find the urgent information, possibly discarding the intervening data.

The receiving application may enable an interrupt for the "Urgent data has arrived" event. If the sender signals another URGENT, the new sequence number is used. The receiver may only get one URGENT interrupt if previous URGENT data has not yet been received (i.e., data is being processed slowly or the system is heavily loaded).

2.1. TCP Retransmission Algorithms and Parameters

This section describes the data retransmission algorithms and parameters used by the TOPS-20 TCP. Unless an application has a reason to override the default algorithm and parameters, it should not specify any ;PERSIST attribute. The application writer may skip to the section on TCP Options.

TCP provides reliable communication over an unreliable network by conceptually assigning a sequence number to each octet of data which is to be sent. It keeps retransmitting the data until it receives an acknowledgement from the foreign end. The purpose of the retransmission algorithms described below is to estimate the time interval which will be required for a packet to be acknowledged, using measurements from prior packets on the connection and a smoothing function.

If the algorithms compute an interval which is too low, unnecessary retransmissions will be sent; they increase charges to the user, waste network bandwidth, and can cause other packets to be lost (and require retransmissions).

If the algorithms compute an interval which is too long, the throughput will be reduced; interactive users will experience frustrating delays.

The actions taken when a retransmission interval has passed vary from implementation to implementation. The "best" action depends on several factors, some of which the TCP cannot determine. Factors which could be relevant include:

- how much data has not been acknowledged,
- how large a packet can be sent,
- how much data/how many packets will the other end hold before throwing packets away,
- has the packet been lost or just delayed,
- what is the round trip time to the other end,
- what is the (minimum) bandwidth of the networks along the route,
- is cpu load a factor in the round trip delay,
- what criteria is the other end using to determine when data should be acknowledged,
- has the other end exhausted the application buffers due to PUSHes,
- is the demand for network access in the host delaying the transmission of packets,
- have overall network delays increased due to congestion or equipment failure,
- have the network delays decreased,

and several others. The actions taken by the TOPS-20 TCP to deliver a packet are:

1. Just before queuing a packet for transmission, compute an initial retransmission interval, RTI, based on the smoothed round-trip-time, SRTT.
2. Schedule a timer for RTI and queue the packet for transmission.
3. If the data in the packet has not been acknowledged before the time interval has passed, put as much additional data into the packet as it will hold (repacketize) and compute a next retransmission interval, NRTI, using the equation for RTI.

4 Schedule a timer for NRTI and queue the packet for retransmission.

5 If the data in the packet has not been acknowledged before the time interval has passed, check whether the transmission timeout interval has passed. If it has, notify the application of a Transmission Timeout error (and abort the connection unless the application has specified otherwise).

6 If the timeout interval has not passed, compute another NRTI (based on the last value) and go to step 4.

If a packet is still in the host's output queue when the timer goes off (it does happen, the network might be off, blocking, or congested), the timer is reset.

Whenever a packet is acknowledged, update the smoothed round-trip-time, SRTT, using the appropriate equations.

The TOPS-20 TCP does not retransmit more than a single packet of unacknowledged data; observations reveal that single packets are lost much more often than several.

> Some implementations retransmit several packets at once when all but the first had been successfully delivered (but there is no sanctioned way to know that). Such action not only wastes (n-1)*100/n percent of the bandwidth but the flood of packets increases the probability that some (gateway's) queue will overflow and cause additional packet loss.

The algorithms presented below have evolved over several years as the deficiencies of prior algorithms have been identified and as the internet has changed. (Most algorithms worked when the internet consisted primarily of the ARPANet - it enforced flow control and rarely lost a packet.)

Two retransmission algorithms are available, one is "dynamic" (the default) and the other "static". The dynamic algorithm is used until the application specifies that it wants to use the static algorithm.

2.1.1. Dynamic Retransmission Algorithm

The dynamic algorithm is similar to the one described in the TCP Protocol specification. A smoothed round trip time, SRTT, is computed by filtering measurements of the time required for an octet to be acknowledged, RTT, using the equation:

SRTT <- (Alpha * SRTT) + ((1 - Alpha) * RTT)

Mills [10] conducted experiments and concluded that two Alphas would improve the estimate; Alpha-i is used when RTT is greater than or equal to SRTT, and Alpha-d is used when RTT is less than SRTT.

The initial retransmission interval, RTI, is then computed from:

RTI <- max (LBND, min (UBND, Beta * SRTT))

where LBND is a lower bound, UBND is an upper bound, and Beta is a variance factor.

The standard values of the parameters are:

Variable	Symbol	Value
Alpha-d	TPRXD	15 (/16=0.9375)
Alpha-i	TPRXI	12 (/16=0.75)
Beta	TPRXV	32 (/16=2.0)
LBND	TPRXN	1000 millisecond
UBND	TPRXX	60000 milliseconds
Initial SRTT	TPDXI	3 seconds
Scale factor	TPRXF	777774

The values of these parameters may be changed by using the SET command (in the SITE-ADDRESS file) and specifying the symbol listed in the above table. Note that TPRXF is a scale factor of the form <0,,-n> used to normalize the

integers TPRXD, TPRXI, and TPRXV, e.g., they are expressed in 16 ths for a TPRXF value of 777774. The bounds, TPRXN and TPRXX, are defined in milliseconds; TPDXI is defined in seconds.

2.1.2. Static Retransmission Algorithm

The static algorithm allows the application to specify its factors. This algorithm updates a minimum, MNRTT, and maximum, MXRTT, round trip time variable as each packet is acknowledged.

```
MNRTT <- min( MNRTT, RTT )

MXRTT <- max( MXRTT, RTT )
```

Every 30 (TCPBGT) seconds the estimated round trip time is computed from:

```
RTI   <- max ( LBND, min ( UBND, (MNRTT+MXRTT)/2 + 1/2 ))
```

and the limits are time weighted by:

```
MNRTT <- MNRTT + (MXRTT-MNRTT)/10

MXRTT <- MXRTT - (MXRTT-MNRTT)/2
```

The next retransmission interval, NRTI, is computed from the last by:

```
NRXI  <- ( NRXI * TPDXN ) / TPDXD
```

where TPDXN and TPDXD are the backoff numerator and denominator, respectively. They may be specified by the application on a per connection basis.

The initial value for both limits is TPDXI, described above; the default values of the backoff parameters are:

Variable	Symbol	Value
TPDXN	TPDXN	3
TPDXD	TPDXD	2

2.1.3. Measuring the Round Trip Time

The most difficult part of implementing the retransmission algorithms is deciding when and how the time required for an octet to be acknowledged can be computed. Ideally, the time is simply the difference between the time a packet is sent and the time that the packet is acknowledged; unfortunately, TCP acknowledges octets, not packets.

One case is easy: if an octet has only been sent in one packet, and the octet is acknowledged, then the round trip time is the difference between the time the packet was sent and the time that the acknowledgement is received.

The problem is thus reduced to deciding if the easy case is sufficient (retransmissions can be ignored), or, if not, how to measure the round trip time for retransmitted packets.

Ignoring retransmissions has been demonstrated to not work. Consider the case where network delays have suddenly increased beyond the current RTI estimate (but packets are still getting through without loss). Since every packet will be retransmitted, they would not be used to estimate a new SRTT, and the host would be retransmitting every packet needlessly. (However note that things would work if the delay were to build up slowly enough that SRTT could track it.) Some method which includes information for retransmitted packets is thus required (unless the costs associated with excessive retransmissions are insignificant).

One way to measure the round trip time for retransmitted packets is to measure the time since the last retransmission of the packet. (This is equivalent to assuming the reason a packet is retransmitted is packet loss, not increased delay.) The case considered in the last paragraph shows that this method will also fail. In reality, it is even worse since the measured time can be very small if the acknowledgement for the first transmission arrives just a little after the packet is retransmitted. The smoothed round trip time would then decrease, causing even more unnecessary retransmissions.

Another way to measure the round trip time for retransmitted packets is to measure the time since the first transmission of the packet. (This is equivalent to assuming the reason a packet is retransmitted is increased delay, not packet loss.) This method has been demonstrated to not work - packets do get lost, particularly when heavily used gateways are involved. This

method also has a feedback mechanism in it: the measured time contains the sum of all of the retransmission intervals. When it is included in the SRTT computation, SRTT quickly grows to very large values (which cause very long pauses when a packet is lost) and consequently low throughput.

The method now being explored keeps two SRTT values. A "reliable SRTT" is based on packets which have never been retransmitted. An "unreliable SRTT" is based only on packets which have been retransmitted; the RTT which it uses is the average time that the packet has been unacknowledged (less the "reliable SRTT"). SRTT is the sum of the two components. Whenever a packet is acknowledged without retransmission, the "unreliable SRTT" value is set to zero.

2.2. Options

Both TCP and IP provide the means to include a selection of standard options in packets. Options are only required in special situations. Most TCP or IP options which are received are automatically processed by TCP. In particular, the routing options are automatically inverted without intervention by the application process. Options to be sent are derived by merging those specified by the application with those which have been received (and processed). If an option is both specified by the application and received in a packet, the received version is ignored. Once specified, the options are sent in each packet generated.

2.2.1. TCP Options

A TCP option has the form of a string of 1 to 40 octets. Except for NOP and END which are each 1 octet long, an individual option is a string of 2 or more octets.

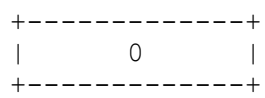
> Note: This is only one interpretation of the formal specification. It allows new (experimental) options of 2 or more octets to be defined and used without requiring all implementations to be simultaneously changed; only defining new 1 octet options will require such a change. Another interpretation which some implementations use is that only the options defined in the specification can appear; anything else causes the options to be unparsable and the packet is discarded.

The first octet is the option code, the second the length in octets of the option string (counting the option code and length octets), followed by additional octets relevant to the particular option.

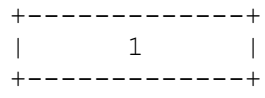
The currently defined TCP options are:

Option Code	Option (Length)	Meaning

0 END (1 octet). Terminates the option string. It may be omitted if the last option ends on a 32-bit boundary.

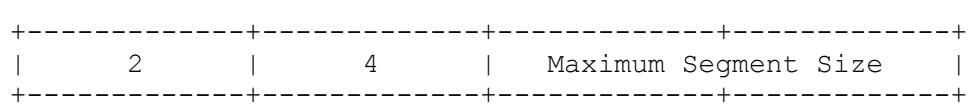


1 NOP (1 octet). Ignored, but may be used for "padding".



2 Maximum Segment Size (4 octets). The receiver of this option should not generate packets whose TCP data length exceeds the number of octets specified by the 16-bit number contained in the third and fourth octets. It also implies a maximum IP packet length of the maximum segment size plus 40 octets (i.e., a standard IP and TCP header, neither containing any options). This option must be specified before OPENF% (or OPEN%). It is processed whenever it is received. If the option is not specified, 536 is assumed.

The option is automatically sent in SYN packets (unless otherwise specified by the application) if the maximum packet size on the network (interface) being used is either less than the 536 default or is significantly greater than 536.



2.2.2. Automatic TCP Processing of IP Options

The format of the IP options can be found in the section on IP Options. The processing which TCP automatically provides for each IP option is listed in the following table.

Option Code	Action automatically taken by TCP for IP Options
0	End. None.
1	NOP. None.
7	Record Route. If there is no application specified routing option, the recorded route is inverted, changed into a Strict Source Route option, and included in subsequently generated packets.
68	Internet Timestamp. If there is no application specified timestamp option and the space reserved for timestamps is not full, the current time is entered into the option and it is returned to the originator in subsequently generated packets.
130	Security. None.
131	Loose Source Routing. If there is no application specified routing option, the source route is automatically inverted for inclusion in subsequently generated packets.
136	Stream Identifier. If there is no application specified stream identifier option, the received identifier is copied for inclusion in subsequently generated packets.
137	Strict Source Routing. If there is no application specified routing option, the source route is automatically inverted for inclusion in subsequently generated packets.

2.3. Creating TCP Connections

Each TCP connection must be unique. Uniqueness is guaranteed by the selection of a 16-bit local port number. There are three ranges of local port numbers:

Range	Use
1 to 255	Special ports for "well-known" services (see Assigned Numbers [11]); their use requires special privileges.
256 to 32767	Ports reserved for normal use.
32768 to 65535	Default port numbers; assigned if no local port number is specified.

The three ways to specify a connection are:

- fully specified active open,
- fully specified passive open, or
- partially wild passive open.

A fully specified active open is used to initiate a connection to a specific foreign end. Opening the connection causes a SYN packet to be sent to the specified foreign host/port. If the SYN packet is acknowledged and a SYN is received from the foreign end, the connection becomes synchronized and may be used to transfer data. If the foreign end of the connection does not exist, a RESET is returned and the open fails. The open will also fail if an ICMP message is received indicating that the foreign host is not reachable, or if the SYN is not acknowledged within the specified Transmission Timeout.

A fully specified passive open is used to wait for a connection request from a specific foreign end. The foreign host and foreign port must be specified. No SYN packet is sent. TCP waits until a SYN (or RESET) packet is received from the foreign end. When a SYN is received, an acknowledgement packet containing a SYN is returned to the foreign end; the connection becomes synchronized when the SYN is acknowledged.

A partially wild passive open is used to wait for a connection request from one of a set of possible foreign hosts and/or foreign ports. If the foreign port is not specified, any foreign port is accepted. If the foreign host is not specified, a connection request by any foreign host is accepted. If the "class and network" field is specified but the "local address" field is zero, only connection requests from hosts on that network will be accepted (see section 4.2, Internet Addresses and Network Numbers). An example is the host's Login responder which accepts connection requests to local port 23 from any port by any host. In this example, a local port of 23 is specified and the foreign host and port entries are omitted (zero).

> Note: TCP does not have fields to identify the name of the hosts which form the ends of a connection; it uses the IP addresses. In order to get around this "problem" and provide more robust service (given the limited capabilities or the ICMP Redirect messages), TCP considers all valid local host addresses as equivalent. Consequently specifying a local host address when opening a listening connection cannot be used to restrict connections. This feature should not cause any problems as there is no way for the local host to restrict the addresses which other systems choose to use for it.

A connection is specified by passing a file specification for the TCP: device to the standard GTJFN% JSYS. The "file name" portion of the file specification describes the local end of the connection and the "file extension" portion describes the foreign end. Several file attributes are available to further specify parameters for the connection. If GTJFN% successfully parses the file specification, it returns a standard JFN which may then be used with the OPENF% JSYS to create, and open, the connection.

After the connection is opened, it may be used where a JFN is acceptable for either input, (e.g., BIN% and SIN%) or output (e.g., BOUT%, NOUT%, and SOUT%). The TCP: device interface uses the standard TOPS-20 JSYSi for most functions. TCP: device specific functions are accessible through the new TCOPR% JSYS, which has been specifically designed for TCP. Connection status information may be obtained from the TCOPR%, SOBE%, SOBF%, and GDSTS% JSYSi.

2.3.1. GTJFN% JSYS 20

The GTJFN% JSYS is used to obtain a file handle for a TCP connection. The format of the GTJFN% JSYS call for the TCP: device is the same as the format for other GTJFN% calls (see the TOPS-20 Monitor Calls Reference Manual [2]). The file specification string provides information about the desired connection.

```
Accepts In AC1: Short: Flags ! Generation.
                Long:  Address of argument table.

           AC2: Short: Source designator.
                Long:  Byte pointer, or 0 if none.

Returns     +1: Failure, TOPS-20 error code in AC1.

            +2: Success.  JFN in AC1.
```

2.3.1.1. File Specification String

The format for the file specification is:

```
TCP:[LOCAL_HOST-][LOCAL_PORT[#]] (continued...)
.[FOREIGN_HOST-][FOREIGN_PORT][;A1..][;A2..]
```

Bracket pairs indicate optional parameters. This does not indicate the parameters are always optional. It does indicate that not all parameters must be supplied at all times.

The components of the file specification are:

Component	Meaning
TCP:	Associates this JFN with TCP (similar to DCN: and SRV: for DNA).
LOCAL_HOST-	Specifies the local host for this connection. Specifying the local host address may be required for some applications running on hosts that have multiple local addresses (multi-homing). The local host may be specified by any alphanumeric

host name acceptable to GTHST% (e.g., DEC-MARLBORO), or the octal host address (e.g., 1200200117). The "-" delimiter after the host name is required to separate the host from the port number; the "-" is required even if the port number is omitted. The default is the highest priority functioning address of the local host. (See the Note in section 2.3.)

LOCAL_PORT — Specifies the local port number for this connection; the field is optional. The port number is specified in decimal. Port numbers must be in the range 1 to 65535. If a local port is not specified a default in the range 32768 to 65535 is assigned.

— The "#" (in reality a "^V#") must be appended to a port in the range of 1 to 255 and in the range of 32768 to 65535 to prevent accidental assignment. A local port in the range 1 to 255 requires permission from the Access Control Job, or WHEEL, OPERATOR, ABSOLUTE-ARPANET-SOCKETS, or ARPANET-WIZARD privileges enabled.

. — Delimits the local host connection values from the foreign host connection values.

FOREIGN_HOST- — ACTIVE connections: specifies the foreign host for this connection; it must be specified.

PASSIVE connections: may be specified to restrict which foreign hosts will be permitted to connect to this connection. The ability to accept a connection from any foreign host is useful for server applications. If the FOREIGN_HOST is not specified, a connection request by any host will be accepted. If just the network field is specified, only requests from hosts using an address on that network will be accepted.

The "-" after the host name delimits it from the port number; the "-" is required even if the port number is omitted.

FOREIGN_PORT ACTIVE connections: specifies the foreign port to be used for this connection; it must be specified.

PASSIVE connections: May be specified to restrict the foreign processes which will be permitted to connect to this connection. If this field is omitted, any port will be accepted.

;A1.. & ;A2.. Specify attributes for the connection. The valid attributes are described below.

Note that a GTJFN% for any valid host name/port pair is allowed. Checking for conflicts with other connections does not occur until the OPENF% is performed. A GTJFN% for the same local parameters with wild foreign parameters is legal.

2.3.1.2. File Specification Attributes

;CONNECTION:ACTIVE (default)
;CONNECTION:PASSIVE
Indicates whether or not TCP should attempt to connect to the foreign host. The default is ACTIVE; both the FOREIGN_HOST- and FOREIGN_PORT must be specified.

;PERSIST:n
;PERSIST:n,m (^V is needed before the comma)
The ;PERSIST attribute is used to specify the retransmission algorithm and its parameters. If transmitted data has not been acknowledged within m seconds, it should be retransmitted. If after n seconds of retransmissions the data has not been acknowledged, the application should be notified of a Transmission Timeout; the default action taken on a Transmission Timeout is to abort the connection.

The default value of n is 30 seconds; the default value of m is 5 seconds. If the value of n is zero, data will be retransmitted until acknowledged; no Transmission Timeout will occur.

If the ;PERSIST attribute is omitted, the standard dynamic retransmission algorithm will be used (see section 2.1, TCP Retransmission Algorithms and Parameters, for a complete description).

;TIMEOUT:n (default: 30 seconds)
The amount of time allowed to pass while waiting for data to arrive from the foreign system before an error is returned to the application. If not given, the default is 30 seconds. If the value of n is 0, no timeout occurs. N may be in the range of 0 to 2^18-1. Note: not fully implemented, there is no timeout on BIN%, SIN%, etc..

;TYPE-OF-SERVICE:n (default: 0)
Indicates what tradeoffs are desired in providing data transmission services. N may be in the range of 0 to 2^18-1. The TCP implementation will only use the low order 8 bits. This attribute is used by TCP and the IP layer underneath. The default value is zero. See section 4.1, IP Type-of-Service, for a description of the possible values.

;SECURITY:n (default: system default level)
The security field is a 16-bit number. The values are described in the Internet Protocol (IP) specification [4]; see also section 4.13, IP Options. If this value is omitted the system default level is used. The system default security level is normally zero. This attribute is not used by TCP but by the IP layer underneath. (Note: The security options have not yet been fully implemented for the JFN interface.) The system default level is set via the .TCSDSL function of the TCOPR% JSYS (Note: The ".TCSDSL" function has not been defined.)

;COMPARTMENTS:n (default: 0)
COMPARTMENTS is a 16-bit number. If not specified, a value of zero is the default. This field is described in the Internet Protocol specification [4]; see also section 4.13, IP Options. This attribute is not used by TCP but by the IP layer underneath. (Note: The security options have not yet been fully implemented for the JFN interface.)

;HANDLING-RESTRICTIONS:n

HANDLING-RESTRICTIONS is a 16-bit number. This field is described in the Internet Protocol specification [4]; see also section 4.13, IP Options. This attribute is not used by TCP but by the IP layer underneath. (Note: The security options have not yet been fully implemented for the JFN interface.)

;TRANSMISSION-CONTROL:n

TRANSMISSION-CONTROL is a 24-bit number. This field is described in the Internet Protocol specification [4]; see also section 4.13, IP Options. This attribute is not used by TCP but by the IP layer underneath. (Note: The security options have not yet been fully implemented for the JFN interface.)

;LOCAL-HOST:a.b.c.d

The LOCAL-HOST attribute is an alternate way to specify the local host. "A", "b", "c", and "d" are decimal octets that form the 32-bit host address. The "." is a required delimiter. A field of zero must be represented by the digit "0".

;FOREIGN-HOST:a.b.c.d

The FOREIGN-HOST attribute is an alternative way to specify the foreign host. "A", "b", "c", and "d" are decimal octets that form the 32-bit host address. The "." is a required delimiter. A field of zero must be represented by the digit "0".

2.3.1.3. File Specification Examples

The following are examples of GTJFN% file specifications for the TCP: device.

File Specification	Description
TCP:1200200117-12345.1200000117-56789	Specifies an active connection from DEC-MARLBORO port 12345 to DEC-TOPS20 port 56789.
TCP:DEC-TOPS20-32145.DEC-MARLBORO-21211	is the same as
TCP:32145.DEC-MARLBORO-21211	is the same as
TCP:32145.1200200117-21211	(assuming the local host is DEC-TOPS20).
TCP:23^V#;CONNECTION:PASSIVE	Specifies a listening connection on port 23. This specification will accept a connection request on local port 23 from any foreign host using any port.
TCP:23^V#.DEC-TOPS20-;CONNECTION:PASSIVE	Specifies a listening connection on port 23; only requests from any port at DEC-TOPS20 will be accepted.
TCP:23^V#.2345;CONNECTION:PASSIVE	specifies a listening connection on port 23 from any host using port 2345.

GTJFN% Error Mnemonics:

In addition to the errors described in the TOPS-20 Monitor Calls Reference Manual [2], the following TCP: specific errors may occur.

TCPXX1 No IP free space for TCB.

TCPXX2 Unable to decode local side TCP of specification.

TCPXX3 Unable to decode foreign side TCP of specification.

TCPXX4 Generation found in TCP specification.

TCPXX5 TCP specification attribute not known to TCP, or no TCB.

TCPXX7 Unable to decode FOREIGN-HOST attribute in TCP specification.

TCPXX8 Unable to decode LOCAL-HOST attribute in TCP specification, or a foreign host was specified.

TCPXX9 Unable to decode PERSIST attribute in TCP specification.

TCPX10 Unable to decode TIMEOUT attribute in TCP specification.

TCPX11 Unable to decode TYPE-OF-SERVICE attribute in TCP specification.

TCPX12 Unable to decode SECURITY attribute in TCP specification.

TCPX13 Unable to decode COMPARTMENTS attribute in TCP specification.

TCPX14 Unable to decode HANDLING-RESTRICTIONS attribute in TCP specification.

TCPX15 Unable to decode TRANSMISSION-CONTROL attribute in TCP specification.

TCPX16 TCP not initialized and available.

2.3.2. OPENF% JSYS 21

The OPENF% JSYS activates the connection, checking for any conflicts. The OPENF% JSYS call is issued after a GTJFN% call. The format of the OPENF% JSYS call for the TCP: device is the same as for other devices. Many parameters pertaining to this connection may be set via the TCOPR% JSYS; some parameters (e.g., routing options or security levels) must be set before the OPENF% JSYS is issued.

Accepts In AC1: JFN.

AC2: OF%BSZ Byte size; 8 or 32.

OF%MOD IO Mode; .TCMWI, .TCMWH, .TCMII, or .TCMIH.

Returns +1: Failure, TOPS-20 error code in AC1.

+2: Success.

2.3.2.1. Byte Sizes

All TCP data transfers are made with 8-bit bytes. The OPENF% byte size, OF%BSZ (AC2 B0:B5), is currently limited to (32 or) 8-bit bytes.

> WARNING: 32-bit mode has been allowed to internally reduce the number of operations required to pass data through the JSYS interface. However, it is possible for a BIN% to hang if it is trying to read 32-bit bytes. The other end might have sent, for example, two 8-bit bytes and is waiting for a reply before it sends more data. In this case, a 32-bit byte cannot be formed.

2.3.2.2. IO Modes

Since all TCP connections are full duplex OF%RD and OF%WR are ignored. Several data modes, OF%MOD (AC2 B6:9), are supported. If no mode is specified, i.e., OF%MOD contains zero, .TCMWI is used.

Mode	Description
.TCMWI (1)	Interactive mode. Wait for connection to be synchronized before returning from the OPENF% JSYS. Wait for the connection to be fully closed before returning from the CLOSF% JSYS. Send all bytes promptly (i.e., send data after each SOUT% or BOUT%). This mode attempts to give "interactive" response by sending many small messages.
.TCMWH (2)	High throughput mode. Same action as mode .TCMWI for OPENF% and CLOSF%. Hold data in buffers until accumulated bytes are sufficient for efficient transmission, or until transmission is requested with the TCOPR% function .TCPSH or SOUTR% JSYSi. This mode attempts to give high throughput at low overhead by sending large messages.
.TCMII (3)	Immediate return mode. Return to application process immediately without waiting on an OPENF% or CLOSF% JSYS. Send all bytes as soon as possible (interactive mode).
.TCMIH (4)	Buffered Immediate return mode. Same as mode .TCMWH except that OPENF% and CLOSF% return immediately.

2.3.2.3. OPENF% Connection Rules

A TCP connection is uniquely specified by the foreign identifier and the local identifier. Each 48-bit identifier consists of the host's 32-bit internet address and the 16-bit port number. Two connections from system A to system B can have the same local ports or the same foreign ports, but not the same local and foreign ports. A connection using a default local port will always be different from any other connection using a default local port (because of the way default local port numbers are assigned). Wild connections (i.e., a connection that allows any foreign host and/or foreign port) may be duplicated in multiple JFNs (in other jobs); in the absence of other distinguishing factors (such as a partially specified connection, precedence, or security levels, etc.), an incoming SYN would be bound to the "oldest" available (listening) connection.

OPENF% Error Mnemonics:

In addition to the errors described in the TOPS-20 Monitor Calls Reference Manual [2], the following TCP: specific errors may occur.

NTWZX1 NET WIZARD capability required for passive OPENF% using local ports less than 256, or access denied by the Access Control Job.

TCPXX2 Unable to decode local side TCP of specification - invalid local port was specified.

TCPXX8 Unable to decode LOCAL-HOST attribute in TCP specification, e.g., wild local port.

TCPX16 TCP not initialized and available.

TCPX17 Illegal IO mode for TCP device, e.g., execute or random access, not readable or writable.

TCPX18 Illegal byte size for TCP device.

TCPX19 TCP connection already exists.

TCPX20 Maximum number of simultaneous TCP connections exceeded, or insufficient space.

TCPX25 Open failure; illegal control bit or connection aborted.

TCPX30 Illegal TCP IO mode, e.g., not .TCMWI, .TCMWH, .TCMII, or .TCMIH.

TCPX31 Connection error or connection rejected by foreign host.

TCPX32 Transmission timeout.

TCPX33 Connection closed.

TCPX35 Illegal to reopen a TCP JFN.

TCPX47 Error in option.

2.4. Using TCP Connections

This section describes the monitor calls that perform I/O for the TCP: device.

After performing a GTJFN%, the application may execute a TCOPR% JSYS to set various fields within the transmission control block (TCB). Some of these functions are allowed only before the OPENF%.

If the application enables state change interrupts, an interrupt is given to the application each time the state of the connection changes:

- connection synchronized (SYNs exchanged and acknowledged),
- connection closing (receipt of a FIN),
- connection closed (FINs exchanged and acknowledged),

and, if the connection is passive,

- beginning synchronization (SYN received).

If the application enables URGENT data interrupts, an interrupt is given each time the urgent data pointer is increased. (That is, notification has been received of additional urgent data beyond that which caused the previous interrupt.) If the application has not read all of the previous urgent data, the interrupt will still be sent but only after DEBRK%ing from the previous interrupt.

If the application enables error interrupts, an interrupt is given under the following conditions:

- A timeout has occurred, either in communications, in opening (synchronizing) the connection, or in closing it.
- An ABORT or RESET message was received.
- A "significant" event has occurred; examine the TCERR word to get the TOPS-20 error code identifying the event. Such events include ICMP Redirect, Destination Unreachable, or Source Quench messages, loss of a network interface, etc.

If an application executes a JSYS that causes input or output; the application process will block until the connection is synchronized and the data specified by the JSYS is moved from (or into) the monitor, or until an event causes the application to be notified of an error.

2.4.1. BIN% and BOUT% JSYSi

The BIN% and BOUT% JSYSi have the same functionality and calling sequences for the TCP: device as for other devices. BIN% reads a single byte and BOUT% sends a single byte.

2.4.2. SIN% and SOUT% JSYSi

The SIN% and SOUT% JSYSi have the same functionality and calling sequences for the TCP: device as for other devices. SIN% will read a stream of bytes and SOUT% will send a stream of bytes.

> WARNING: the TCP PUSH is NOT a "record delimiter"; using it to read "command lines", for example, will not work reliably.

2.4.3. SINR% JSYS

The SINR% JSYS has the same functionality as the SIN% JSYS with one addition. The SINR% JSYS returns when a TCP message with the PUSH flag is received. The SINR% also returns when the byte count is exhausted.

2.4.4. SOUTR% JSYS

The SOUTR% JSYS has the same functionality as the SOUT% JSYS with one addition. The SOUTR% JSYS sets the TCP PUSH flag for the last message generated by this call. This also forces all data, currently held in buffers, to be sent promptly.

IO Error Mnemonics:

In addition to the errors described in the TOPS-20 Monitor Calls Reference Manual [2], the following TCP: specific errors may occur.

TCPXX1 No IP free space for buffer header.

TCPX16 TCP not initialized and available.

TCPX25 Illegal control bit, bad buffer argument, or connection aborted.

TCPX31 Connection error or connection rejected.

TCPX32 Transmission timeout.

TCPX33 Connection closing or closed.

TCPX35 Illegal to reopen a TCP JFN (the connection has been aborted).

TCPX47 Error in option processing.

2.4.5. SIBE% JSYS

At any time, a SIBE% will return the number of unread bytes available to the application, i.e., those bytes which have been acknowledged. This is not the receive window size. To determine the size of the receive window, the program must use the TCOPR% JSYS.

2.4.6. SOBE% and SOBF% JSYSi

The SOBE% and SOBF% JSYSi have the same functionality and calling sequence for the TCP: device as for other devices. (See the TOPS-20 Monitor Calls Reference Manual [2].) A SOBE% or SOBF% will return the number of bytes which are in the output buffer, i.e., the number of bytes which are have not been acknowledged by the foreign host.

2.4.7. GDSTS% JSYS

The GDSTS% JSYS obtains the status of a given connection. The GDSTS% JSYS is implemented for completeness.

Accepts In AC1: JFN of connection.

Returns +1: Always. May cause an Illegal Instruction Trap.

AC2: Current status of the specified connection. The send side status is returned in the left half, and the receive side status is returned in the right half. The states are documented in the .TCTCS word of the .TCRCS function of the TCOPR% JSYS (which may be used instead of GDSTS%).

AC3: The foreign host address.

AC4: The foreign port number.

Note: the JFN must be open; a return from OPENF% is sufficient even though the connection may not yet be synchronized. The DESX5 error (ITRAP) is not given if the JFN is not open and the ACs returned are invalid (monitor bug??).

2.4.8. TCOPR% JSYS 761

Transmission Control Protocol Operations

Special functions related to the TCP: device are performed using the TCOPR% JSYS (which is similar to the MTOPR% JSYS). Note: many of the symbols used below have not been defined in MONSYM.MAC (DEC has not yet implemented the corresponding functions).

Accepts In AC1: JFN of connection.

AC2: Function code (see below).

AC3: Function argument or address of argument block. See each function for details.

Returns +1: Always. May cause an Illegal Instruction Trap.

Function	Argument Block Offset	Field	Meaning
.TCRCS			Read connection status. AC3 points to a block which is at least .TCRCL words long. (Not yet implemented; note that the symbols associated with this function, except for the connection state codes listed under .TCTCS, have not been defined in MONSYM. The use of the .TCRTC function is recommended.)
		.TCLEN	Length of block, including this word.
		.TCTFP	Foreign port. TC%TFP 16 bits, right justified.
		.TCTFH	Foreign host. TC%TFH 32 bits, right justified.
		.TCTLP	Local port. TC%TLP 16 bits, right justified.
		.TCTLH	Local host. TC%TLH 32 bits, right justified.

.TCTRW Receive window.
TC%TRW 16 bits, right justified.

.TCTSW Send window.
TC%TSW 16 bits, right justified.

.TCTCS Connection state.
TC%TCS Send state in left half, receive state in right half.

.TCNOT Connection not open.
.TCTIM Connection in time-wait.
.TCFIN Connection closed.
.TCSYA Connection waiting for foreign connection request.
.TCSYS Connection opening.
.TCSYN Connection open (synchronized).

.TCTBW Buffers waiting for acknowledgement (ACK).
TC%TBW 8 bits, right justified.

.TCTBP Buffers pending receipt.
TC%TBP 8 bits, right justified.

.TCTBS Buffer size.
TC%TBS 16 bits, right justified.

.TCTTS Type-of-Service and security fields. (Note: the security option requires a 16-bit COMPARTMENTS field, a 16-bit HANDLING-RESTRICTIONS field, and a 24-bit TRANSMISSION-CONTROL field; how they can be packed into TC%TSF and TC%TCF is not known.)

TC%TTS 18 bits, 1st 18-bit byte.
TC%TSF 2 bits, 3rd 9-bit byte.
TC%TCF 8 bits, 4th 9-bit byte.

The bits used in the TCP implementation are:

TC%TPR	3 bits	Precedence
TC%TST	1 bit	Stream / Datagram
TC%TRE	2 bits	Reliability
TC%TSR	1 bit	Speed / Reliability
TC%TSP	1 bit	Speed

Note: The format of the Type-of-Service field has changed since these fields were

defined. See section 4.1, IP Type-of-Service.

.TCTTT Transmission timeout.
TC%TTT 9 bits, right justified.

.TCTUD Urgent data information (to be defined).

.TCTRA Retransmission parameters - Alpha.
TC%TRA Rising Alpha, a floating point number.

.TCTDA Retransmission parameters - Alpha.
TC%TDA Decaying Alpha, a floating point number.

.TCTRB Retransmission parameters - Beta.
TC%TRB Beta, a floating point number.

See section 2.1, TCP Retransmission Algorithms and Parameters, for a description of the .TCTRA, .TCRDA, and .TCTRB fields, their uses, and limitations.

.TCTPI PSI channel assignment. See function .TCSPC for a definition of this field.

.TCSUD Send urgent data. AC3 points to a block of the form:

0 Byte pointer to data.
1 Count of bytes or 0.
2 Byte on which to terminate output.

(Note that these are the same as AC2 to AC4 of the SOUT% and SOUTR% JSYSi).

WARNING: It has not been verified that this function will actually send any of the data.

.TCPSH Send all buffered data promptly. Also set the TCP PUSH flag for the last message of the data being sent.

.TCSPA Set passive/active flag.
TC%APF is set in AC3 to indicate an active connection, and cleared to indicate a passive connection. (Note: AC3 equal to zero is passive, non-zero is active.)

.TCSPP Set persistence parameters. AC3 is the number of seconds to wait for connections to complete the synchronization process.

AC3	Meaning
0	do not timeout the connection.
0,,n	an attempt to connect is made for n seconds; TCP determines the retransmission interval.
m,,n	an attempt to connect is made for n seconds; TCP retransmits every m seconds. M must be less than n.

.TCSTP Set timeout parameters. AC3 contains the number of seconds to allow for data to be delivered before a returning a Transmission Timeout error. The value given must be in the range of 0 to 2^18-1. If AC3 contains a 0, no timeout will occur. (Note: the bounds are not checked. This sounds like the n in the ;PERSIST:n,m attribute (without the ability to specify the m), and not the ;TIMEOUT:n attribute.)

.TCSRP Set retransmission parameters. AC3 points to an argument block of three words; each word contains a floating point number. For a description of these three fields, see section 2.1, TCP Retransmission Algorithms and Parameters. (Not yet implemented. These parameters are currently system wide, not per connection; the system wide parameters can be specified at system load time in the SITE-ADDRESS file.)

0 Rising Alpha.
1 Decaying Alpha.
2 Beta.

.TCSTS Set Type-of-Service. AC3 contains the type of service desired. The value must be in the range of 0 to 2^18-1. TCP only uses the low order 8 bits. The values are described in the Internet Protocol specification [4]. See section 4.1, IP Type-of-Service.

.TCSSC Set SECURITY level and COMPARTMENTS. AC3 contains the SECURITY level and the COMPARTMENTS in the form <security,,compartments>. SECURITY and COMPARTMENTS are each 16-bit numbers described in the Internet Protocol specification [4], see section 4.13, IP Options. (These fields are part of the IP security option which is not yet fully implemented for the JFN interface.)

.TCSHT Set HANDLING-RESTRICTIONS and TRANSMISSION-CONTROL fields. HANDLING-RESTRICTIONS is a 16-bit value and is specified in AC3. TRANSMISSION-CONTROL is a 24-bit field and is specified in AC4. The values for both fields are described in the Internet Protocol specification [4], see section 4.13, IP Options. (These fields are part of the IP security option which is not yet fully implemented for the JFN interface.)

.TCSPC Set PSI channels. AC3 contains six 6-bit fields as follows:

TC%TPU 1st byte, Urgent data channel.

TC%TER 2th byte, Error channel.

TC%TSC 3th byte, State change.

TC%TXX 4-6th bytes, unused, must be 77 (octal).

Specify the value 77 (octal) if no change for the event is desired. Specify the value 76 (octal) to clear interrupts for the event.

.TCRTW Read a single entry from the TCB. AC3 contains the word offset into the TCB that is desired. On return, AC3 contains the value of the specified word. (Use of the .TCRTC function is recommended, at least to translate a symbolic name into a word offset. Using .TCRTC will also

isolate the applications software from changes to the internal structure of the TCB.)

.TCLSR Set the loose source route used to transmit messages. AC3 points to an argument block. The first word of that block is the total length of the block, including the length word. The following words are the right justified internet addresses of IP entities (usually gateways) in the route. Note that the last word must be the internet address of the destination host. See section 4.13, IP Options. (Not yet implemented for the JFN interface.)

.TCSSR Set the strict source route used to transmit messages. AC3 points to an argument block. The first word of that block is the total length of the block, including the length word. The following words are the right justified internet addresses of EVERY IP entity (usually gateways) in the route to the destination. Note that the last word must be the internet address of the destination host. See section 4.13, IP Options. (Not yet implemented for the JFN interface.)

.TCRLB Read lower bound for retransmission. The number of seconds is returned in AC3 as a floating point number. (Note: the upper and lower bounds for the retransmission timeout are currently system wide (integer) parameters, not per connection. They may be read using the .IPRIP function of the IPOPR% JSYS.) (Not yet implemented for the JFN interface.) (Is this the timeout or the interval?)

.TCSLB Set lower bound for retransmission. The number of seconds is specified in AC3 as a floating point number. Requires WHEEL, OPERATOR or ARPANET-WIZARD capability enabled. The number must be larger than 0 and less than the current upper bound. (Not yet implemented for the JFN interface. The upper and lower bounds for the retransmission timeout may be set at system load time in the SITE-ADDRESS file.) (Is this the timeout or the interval?)

.TCRUB Read upper bound for retransmission. The number of seconds is returned in AC3 as a floating point number. (Note: the upper and lower bounds for the retransmission timeout are currently system wide (integer) parameters, not per connection. They may be read using the .IPRIP function of the IPOPR% JSYS.) (Not yet implemented for the JFN interface.) (Is this the timeout or the interval?)

.TCSUB Set upper bound for retransmission. The number of seconds is specified in AC3 as a floating point number. Requires WHEEL, OPERATOR or ARPANET-WIZARD capability enabled. The number must be larger than the current lower bound and less than 250. (Not yet implemented for the JFN interface. The upper and lower bounds for the retransmission timeout may be set at system load time in the SITE-ADDRESS file.) (Is this the timeout or the interval?)

.TCSFN Send a FIN without closing the connection. Sending a FIN implies that this end of the connection has no more data to be sent. Data from the other end may still be received, however, until a FIN is received from it (which will cause a closing state change interrupt, if enabled).

.TCRTC Read a block of connection information. The connection must be OPENF%ed. This function can be used to obtain by name information from (or about) the connection control block (TCB). (The .IPRIP function of the IPOPR% JSYS can be used to obtain information which is not related to a particular connection.)

> BEWARE: the order and entries are site dependent and may change in future releases. It is thus recommended that numeric TCB offsets in .TCNMP not be used. Use NT%SD and the ASCII names.

AC3 contains the address of an argument block containing at least four words plus the header word.

.TCLEN Length of block, including this word.

.TCFLG Flags:

NT%SD An LDB pointer should be returned for each of the specified symbols instead of the value. The byte pointers returned assume AC14 (octal) is pointing to a TCB image. Requires that NT%SY be set.

NT%SL .TCLNP points to an area where the length, in LDB bytes, for each datum should be placed. Requires that NT%SY be set.

NT%SY The right-half of .TCNMP contains the address of a list of ASCII TCB variable names for which information should be returned.

.TCNMP Specifies the source. The contents of this word depend on the setting of the NT%SY flag. When NT%SY is set, .TCNMP contains:

-N,,Address of ASCII TCB symbol names

N symbols are given beginning at the specified application address. The symbols must be upper case ASCII containing one to five characters. If NT%SY is not set, .TCNMP contains:

-N,,Offset into TCB

N successive words from the TCB are desired beginning at the specified offset.

.TCDTP Specifies the destination. This word contains:

-M,,Address in application space

The beginning address where data to be returned should be placed. The block is M words long. Note that the returned data are right justified and that some TCB entries contain more than one word of data (e.g., the option fields).

.TCLNP If NT%SL is not set, this word is not used. When NT%SL is set the word contains:

0,,Address in application space

The number of LDB bytes associated with the specified symbols is placed beginning at the specified address. The block is also M words long (see .TCDTP).

The counts and pointers in .TCNMP, .TCDTP, and, if NT%SL was specified, .TCLNP are updated.

Failure due to a bad JFN, offset, or an illegal symbol causes an Illegal Instruction Trap; a TOPS-20 error code is returned in AC1. The pointers in the argument block will point after the last item successfully processed, i.e., at a bad symbol.

The names of some of the variables in a connection's internal control block (TCB) which are recognized by the .TCRTC are listed after the error mnemonics below.

.TCSIL Set the interrupt level for buffers. (Not yet implemented, the intended function is not clear.)

TCOPR% Error Mnemonics:

TCPX10 Illegal TIMEOUT.

TCPX11 Illegal TYPE-OF-SERVICE.

TCPX12 Illegal SECURITY.

TCPX21 Wheel, Operator, or Network Wizard needed for special TCOPR function.

TCPX22 Invalid TCOPR function requested, or .TCPSH or .TCSUD for device other than TCP:.

TCPX26 Illegal Persist parameters.

TCPX27 Illegal TCOPR Function on an OPEN TCP JFN.

TCPX34 TCOPR Argument, e.g., PSI channel, or .TCRTC flag or argument.

TCPX35 Illegal to reopen a TCP JFN (the connection has been aborted).

TCPX36 Illegal TCOPR Function on an UNOPEN TCP JFN.

TCPX40 Function not yet implemented.

TCPX41 TCOPR DEC reserved interrupt channels not off.

TCPX42 TCOPR Invalid TCB offset in .TCRTW.

2.4.8.1. TCB Field Names

The names and contents of some of the variables in a connection's internal control block (TCB) which are recognized by the .TCRTC are listed below. See the NTITEM entries in the TCPTCB macro in ANAUNV.MAC for a complete list. The Number Bytes column is only specified for fields that consist of more than one byte (see NT%SL and NT%SD above).

TCB Field	Number Bytes	Contents
TABTF		(System-wide) ForkX of fork ABORT%ing this connection.
TCBIO	10.	IP options to be sent.
TCBIR	10.	IP options received in last packet.
TCBIU	10.	IP options specified by application process.
TCBPT		TIME% of last inactivity probe.
TCBTO	10.	TCP options to be sent.
TCBTR	10.	TCP options received in last packet.
TCBTU	10.	TCP options specified by application process.
TCERR		Last TOPS-20 error code.
TCLTM		TIME% of last packet received.
TCRTC		# packets received.
TCRXP		# retransmissions.
TCSPC		# packets sent (including retransmissions).
TDEC		TCP: device connection.
TERJN		JFN for advisory messages (TCP%ET) (incomplete).
TERR		Last (BBN) error code.

TERRF	Error wait bit index.
TERRT	Trace events related to this connection (TCP%PT).
TFH	Foreign host internet address.
TFP	Foreign port number.
TIFDF	IP Do-not-Fragment flag.
TIPDO	Send IP Data Offset (words).
TIPOR	Number of received IP option bytes.
TIPOU	Number of application specified IP option bytes.
TJCN	Connection's JCN.
TJFN	Internal JFN number for connection.
TLH	Local host internet address.
TLP	Local port number.
TMNRT	Static retransmission factor "MNRTT".
TMXRT	Static retransmission factor "MXRTT".
TOFRK	(Job-relative) ForkN # of fork which owns this connection.
TOPFH	Initial foreign host internet address.
TOPFP	Initial foreign port number.
TOPLH	Initial local host internet address.
TOPNF	OPEN wait bit index.
TOWNR	Job number of job owning this connection.
TPICA	Channel to receive "reserved" interrupt.
TPICE	Channel to receive error interrupt.
TPICR	Channel to receive receive buffer full interrupt.
TPICS	Channel to receive send buffer empty interrupt.
TPICU	Channel to receive urgent data interrupt.
TPICX	Channel to receive opening/open/closing/closed interrupt.
TPIFA	Fork to receive "reserved" interrupt.
TPIFE	Fork to receive error interrupt.
TPIFR	Fork to receive receive buffer full interrupt.
TPIFS	Fork to receive send buffer empty interrupt.
TPIFU	Fork to receive urgent data interrupt.
TPIFX	Fork to receive opening/open/closing/closed interrupt.
TPRS1	;PERSIST n.
TPRS2	;PERSIST m.
TRBS	Amount of application receive buffer space available.
TRCBY	Byte number of next octet to be reassembled.
TRIS	Initial receive sequence number.
TRLAK	Sequence number of last acknowledgement sent.
TRLFT	Sequence number of next octet to be reassembled.
TRLWN	Sequence number of last receive window sent.
TRPP	Have a partially reassembled packet.
TRSYN	State of receive side of connection.
TRURG	Have additional urgent data to be reassembled.

TRURP	Sequence number of last receive urgent octet.
TRWND	Receive window.
TRXI	Static retransmission factor "RTI".
TRXPD	Static retransmission backoff denominator.
TRXPN	Static retransmission backoff numerator.
TSABT	Connection is being ABORT%ed (RESET%/CLZFF% JSYSi).
TSBFP	# PUSHes sent.
TSBYT	Number of octets available to be sent.
TSCB	Current send (internal) buffer header.
TSCR	Secure connection.
TSEP	Packetizer is being encouraged to generate a packet.
TSESB	Estimated transmission baud rate.
TSFP	Packetizer is being forced to generate a packet.
TSIS	Initial send sequence number.
TSLFT	Sequence number of oldest unacknowledged octet.
TSLVC	Current security level.
TSLVN	Next security level.
TSMRT	Dynamic retransmission factor "reliable SRTT".
TSMXB	Maximum permitted transmission baud rate.
TSMXS	Maximum send segment size.
TSOPN	Application has been notified connection is open.
TSPRB	TIME% of next closed window probe.
TSPRS	TCP%PS was set at OPEN%.
TSRX	Do not abort connection on Transmission Timeout.
TSSEQ	Current send sequence number.
TSSHR	;HANDLING-RESTRICTION n.
TSSCP	;COMPARTMENTS n.
TSSTC	;TRANSMISSION-CONTROL n.
TSSV	TSSEQ is valid.
TSSYN	State of send side of connection.
TSTMW	Need time-wait state.
TSTO	Transmission Timeout in milliseconds.
TSUOP	Application process opened connection.
TSURG	Urgent send mode.
TSURP	Sequence number of current send urgent octet.
TSWND	Current send window.
TTOS	IP Type-of-Service field.
TTPDO	Send TCP Data Offset (words).
TTPOR	Number of received TCP option bytes.
TTPOU	Number of application specified TCP option bytes.
TTTL	Initial IP Time-to-Live.
TTVT	Connection is a TVT.
TVTL	Associated TVT line number.
TUNRT	Dynamic retransmission factor "unreliable SRTT".

3. TELNET

The Telnet Protocol is a general symmetric 8-bit byte oriented communications facility used to create network virtual terminals (NVTs). The protocol is defined in RFC 854 [7]. NVTs which use TCP as the session layer protocol are called TCP virtual terminals (TVTs).

An open TCP connection which has not been used to transfer data may be converted by the ATNVT% JSYS into a TVT. ATNVT% is given a TCP: device JFN. It returns a TOPS-20 TTY designator. The designator can be used to obtain input and output JFNs or assigned to a created job. The connection can be used as any other TOPS-20 TTY.

3.1. Telnet Negotiations

The Telnet Protocol defines several options that specify control information or communicate out-of-band signals between the two processes using the connection. Options are negotiated between the two ends of a connection. One process can request (using .MONEG) that the other process either enable (DO) or disable (DONT) a particular option; the other process may either agree (WILL) or refuse (WONT) the request. Similarly, the second process can offer (WILL) or indicate that it cannot (WONT) perform an option; the first process can then either accept (DO) or reject (DONT) the offer. Note that a process must always accept a request to disable an option (i.e., return to the initial state).

In order to prevent negotiation loops which could occur if both processes initiated an option (for one end) at the same time, replies are not sent if the receiver of an option finds that it is already in the desired state. Since both ends should be keeping track of, and agree upon, the state, a receiver should only find itself in a desired state when the processes simultaneously initiate a negotiation. In this case, each process interprets the other's negotiation as an acknowledgement of its own. There is a timeout associated with a negotiation to allow recovery from an error.

3.2. Telnet Subnegotiations

The state associated with some options involves more than simply enabled or disabled; other options may need to exchange messages. The subnegotiation mechanism is provided for use with these options. Subnegotiations for an option can only be sent (using .MOSSB) after first establishing that the other process implements the option. The format of the subnegotiation data is a string of zero or more 8-bit bytes. The string is interpreted by the application process, not the Telnet protocol. Subnegotiations strings may be read using the .MOGSB function of MTOPR%.

3.3. Telnet State

Each end of the connection must maintain a state table that records the enabled or disabled state of each option, at each end (recall that the Telnet protocol is symmetric). The following table lists the fields in the state byte associated with each Telnet option.

Symbol	Meaning
MO%OND	There is an outstanding DO. A DO has been sent but no reply has been received.
MO%ONW	There is an outstanding WILL. A WILL has been sent but no reply has been received.
MO%CSD	Current state is DO. The other process has enabled the option. Either a DO was sent and a WILL reply was received, or a WILL was received from the other process and this end sent a DO reply. MO%CSD will be zero if a WONT was received from the other process or the option is in the initial disabled state.
MO%CSW	Current state is WILL. This process has enabled the option. Either a DO was received and a WILL reply was sent, or a WILL was sent to the other process and a DO reply was received. MO%CSW will be zero if a DONT was received from the other process or the option is in the initial disabled state.
MO%SBR	Subnegotiation Byte Received and Available. All subnegotiation bytes for the option have been received

and the application may read the bytes using the .MOGSB function of the MTOPR% JSYS.

MO%SBS Subnegotiation Byte Sent (Not used).

3.4. Telnet Option Codes

The following Telnet Option Codes are defined. Only Binary Transmission and Echo are supported by Server Telnet.

Code	Name
0	Binary Transmission
1	Echo
2	Reconnection
3	Suppress Go Ahead
4	Approx Message Size Negotiation
5	Status
6	Timing Mark
7	Remote Controlled Trans and Echo (RCTE)
8	Output Line Width
9	Output Page Size
10	Output Carriage-Return Disposition
11	Output Horizontal Tab Stops
12	Output Horizontal Tab Disposition
13	Output Formfeed Disposition
14	Output Vertical Tab Stops
15	Output Vertical Tab Disposition
16	Output Linefeed Disposition
17	Extended ASCII
18	Logout
19	Byte Macro
20	Data Entry Terminal
21	SUPDUP
22	SUPDUP Output
23	Send Location
24	Terminal Type
25	End of Record
26	Transmit Unique ID
27	Output Mark

3.5. MTOPR% JSYS 77

The MTOPR% JSYS is used to perform special functions related to network virtual terminals and Telnet connections.

Accepts In AC1: JFN of the network virtual terminal.

AC2: Function Code.

AC3: Function arguments.

Returns +1: Always.

3.5.1. TTY Functions

Function	Description
.MOSPD	Set the terminal line speed and, if the terminal is a TVT, set the connection's maximum output baud rate (TSMXB). In the latter case, packets are not sent which would exceed the specified output rate. Setting the limit may be helpful when the remote end of the connection has a low baud rate, or if there is a low bandwidth network link in the path. If set too low, output will appear "choppy". AC3: Left half, input baud rate; Right half, output baud rate.
.MOSND	Promptly send any currently buffered bytes, setting the TCP PUSH flag in the last packet sent. Note: this function implemented the TCP PUSH function for TVTs before TCOPR% existed.

3.5.2. Telnet Functions

Function	Description

.MONEG Negotiate a Telnet option or read the option state byte. This functions causes the NVT to negotiate an option (i.e., send IAC <command> <option>), then return the negotiation state byte. AC3 contains:

MO%RNS Read negotiation state. If MO%RNS is set, no negotiation takes place but the option state byte is returned in AC3. A Telnet negotiation is sent to the remote Telnet if MO%RNS is 0; the process will be blocked until either a reply is received or the negotiation timeout (30-60 seconds, see TVTPR+3) has expired.

MO%NEG Negotiation command. This field must contain one of the following four command codes:

WILL (251.) This end of the connection offers to perform the specified option.

WONT (252.) This end of the connection refuses to perform the specified option.

DO (253.) This end of the connection requests the remote end to perform the specified option.

DONT (254.) This end of the connection requests the remote end to stop or not perform the specified option.

MO%OPT Telnet option. This field must contain a supported Telnet option code. Option codes are defined in RFC 855 [12]. Some of the more common options are specified in RFC 855 through 861.

Upon return, AC3 contains the option state byte, or -1 if the call fails. The application must provide synchronization if multiple processes are using the virtual terminal. The current state of an option should be checked before negotiating it: if the option has

already been negotated the other end will not reply and the process will be blocked until the timeout has expired.

.MOGSB Get Telnet subnegotiation bytes. This function either:

- o returns all currently buffered Telnet subnegotiation bytes to the caller,
- o blocks until subnegotiation bytes are available, or
- o blocks until the timeout period expires.

The subnegotiation byte string may only be read once. AC3 contains the address of a subnegotiation block. The format of the subnegotiation block is:

Offset	Meaning

.MOSBO Telnet option word:

Field	Meaning
MO%TMO	Time out in seconds. This field specifies the maximum number of seconds to wait for a subnegotiation to arrive.
MO%OPT	Telnet option. Specifies the Telnet option whose subnegotiation byte string is to be returned.

.MOSBP Subnegotiation byte pointer. A destination byte pointer to an area into which the received subnegotiation bytes should be copied. The byte pointer is updated by the number of bytes copied.

.MOSBN Number of subnegotiation bytes. The maximum number of bytes available in the destination area. This word is changed to be the actual number of bytes copied.

Upon return words .MOSBP and .MOSBN of the subnegotiation block are updated. .MOSBN is set to 0 if an error occurs.

.MOSSB Send Telnet subnegotiation bytes. This function sends a subnegotiation byte string. The option must be in the WILL state, i.e., MO%CSW must be on in the option state byte for the specified option. IACs (255) found in the subnegotiation byte string are automatically doubled; the receiver will correspondingly convert double IACs to a byte equal to 255 before passing the subnegotiation string to the application. AC3 contains the address of a subnegotiation block. The format of the subnegotiation block is:

Offset	Meaning

.MOSBO Telnet option word:

Field	Meaning
MO%OPT	Telnet option. Specifies the Telnet option for which this is a subnegotiation.

.MOSBP Subnegotiation byte pointer. The source byte pointer to the subnegotiation byte string. Note that the byte pointer should specify 8-bit bytes. The byte pointer is updated.

.MOSBN Number of subnegotiation bytes. The number of bytes in the source string. This word is changed to the actual number of bytes sent.

Upon return words .MOSBP and .MOSBN of the subnegotiation block are updated. .MOSBN is set to 0 if an error occurs.

3.6. ATNVT% JSYS 274

Attach a Network Virtual Terminal using the Telnet Protocol to a TCP connection.

Accepts In AC1: Flags ! connection identifier

AN%TCP This flag is set to indicate that the right-half of AC1 has a BBN TCP interface JCN; if it is not set, the right-half of AC1 has a JFN.

Returns +1: Failure, TOPS-20 error code in AC1, the JFN (or JCN) is still valid.

+2: Success, the terminal designator for the Network Virtual Terminal is returned in AC1, the JFN (or JCN) has been released.

The caller must first open a connection. The connection must be synchronized with no data sent or received before the ATNVT% call. If successful, the JFN (or JCN) is released and the connection is transferred to the Network Virtual Terminal, whose TTY designator is returned in AC1. If unsuccessful, one of the error codes listed below is returned in AC1 and the JFN (or JCN) remains valid. (It should probably be CLOSF%ed (or ABORT%ed).)

ATNVT% Error Mnemonics:

ATNX1 Invalid JFN (JCN).

ATNX2 Receive side not SYNCED.

ATNX3 Connection not usable.

ATNX5 Receive side has been used.

ATNX6 Connection has been closed or has errors.

ATNX8 Send side not SYNCED.

ATNX9 Send JFN not open.

ATNX10 Send JFN is not a NET connection.

ATNX11 Send side has been used.

ATNX13 No free TVTs.

OPNX15 Read/write access required.

TCPX35 Illegal to reopen a TCP JFN, i.e., the connection has closed.

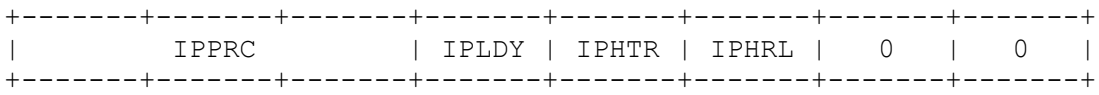
4. IP and User Queues

The main function of the Internet Protocol (IP) is to move the datagrams it receives a step closer to their destination. To perform this function an IP implementation must:

- discard packets whose Time-to-Live has expired,
- be able to recognize all of its addresses,
- be able to determine if a destination address is on a directly connected network,
- maintain a cache of routes to a network,
- be able to update the routing cache from ICMP Redirect messages,
- know the address of at least one gateway (on each of its directly connected networks) to use when the cache does not contain a valid route,
- perform routing based on a packet's Type-of-Service,
- process any IP options found in a packet,
- be able to fragment a packet, if required, prior to transmission over a directly connected network,
- be able to recombine fragments of a packet prior to delivering it to a higher level protocol.

4.1. Type-of-Service

IP allows an application to specify what level of service it requires and what tradeoffs in network capabilities are desired. The IP Type-of-Service field is an 8-bit byte composed of six sub-fields; two fields are undefined.



The left-most 3 bits, IPPRC, form the Precedence field. Its eight possible values are:

7 - Network Control
6 - Internetwork Control
5 - CRITIC/ECP
4 - Flash Override
3 - Flash
2 - Immediate
1 - Priority
0 - Routine

The next bit, IPLDY, is the Low Delay flag. If the flag is set, the packet will be processed in a manner which minimizes "delay" through the internet. If the flag is not set, the packet will be treated in the "standard" manner.

The next bit, IPHTR, is the High Throughput flag. If the flag is set, the packet will be processed in a manner which maximizes "throughput" through the internet. If the flag is not set, the packet will be treated in the "standard" manner.

The next bit, IPHRL, is the High Reliability flag. If the flag is set, the packet will be processed in a manner which maximizes "reliability" through the internet. If the flag is not set, the packet will be treated in the "standard" manner.

The two remaining bits are reserved and should be zero.

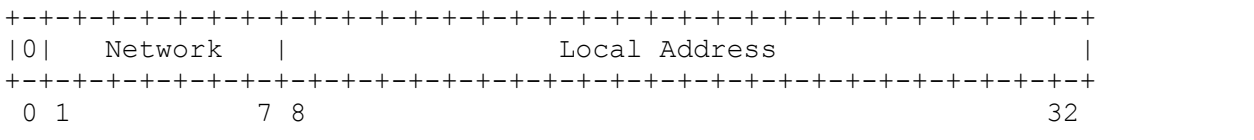
Note that setting any of the Low Delay, High Throughput, or High Reliability flags may increase a user's "costs"; it is illegal to set all three flags.

4.2. Internet Addresses and Network Numbers

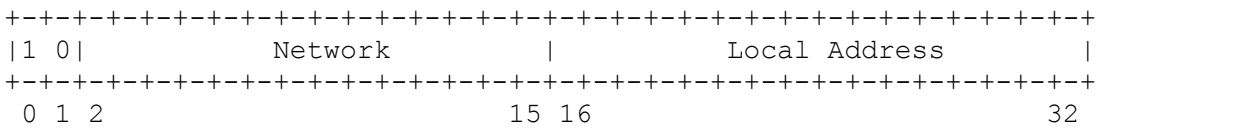
Each host may have several internet addresses on each of its directly connected networks. In addition to the regular address(es), the TOPS-20 IP has partial support for broadcast and multicast addresses (as yet untested).

An internet address is a 32-bit quantity, in which the left-most one, two, or three octets specify the network number and the remaining octets identify a host on that network.

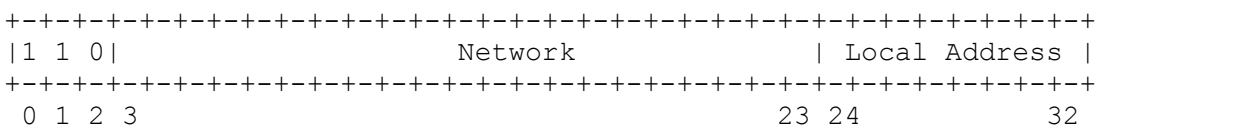
Class A addresses begin with a zero bit, followed by a 7-bit network number and a 24-bit local address.



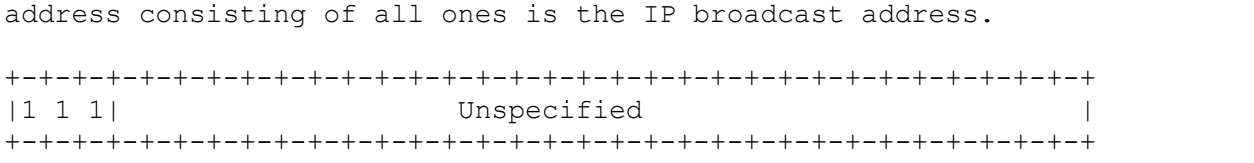
Class B addresses begin with bits 10, followed by a 14-bit network number and a 16-bit local address.



Class C addresses begin with bits 110, followed by a 21-bit network number and an 8-bit local address.



Class D addresses begin with bits 111. The interpretation of the remaining bits is currently unspecified except that the address consisting of all ones is the IP broadcast address.



4.3. Logical Hosts

The TOPS-20 logical host facility allows several hosts to share a single (physical) network interface under its control. This additional demultiplexing function is performed based on the local host address contained in received packets.

To use this facility, some network dependent number of bits from the local address field are used to distinguish the different hosts. The bits, if any, are called the Logical Host field. They are designated by each network administrator. In the case of the ARPANET, which is Internet Class A network 10, the logical host field is the third octet in the Internet Address. IP modules "ignore" the bits in the Logical Host field when making routing decisions but use them when demultiplexing the packets.

4.4. Packet Processing by IP

The processing performed for each packet by IP is illustrated in Figure 2. IP first extracts the destination internet address in the IP header, sets the bits in the Logical Host field to zero, and checks to see if it matches any of the host's addresses. If a datagram is not for the local host a route must be found for it. The network number is extracted from the address. If the network number matches the network number of any operational interface, the datagram is sent directly to the destination over that interface.

If the the destination is on another network, the routing cache is examined for a (network, gateway) pair whose network is the same as that of the destination. If an entry is found, the datagram is sent to the gateway. If no entry is found, a random gateway (unless restricted) is chosen and the (network, gateway) pair is placed into the routing cache, and the datagram is sent to the gateway.

If a gateway which receives a packet has a better route to the desired network, it will send an ICMP Redirect message back to the local host so that the routing cache can be updated.

All of a host's network interfaces must go down before internet communications are completely blocked. Even then, processes within a host can communicate through the loopback interface. Datagrams sent from the host to itself via the loopback interface are delivered quickly.

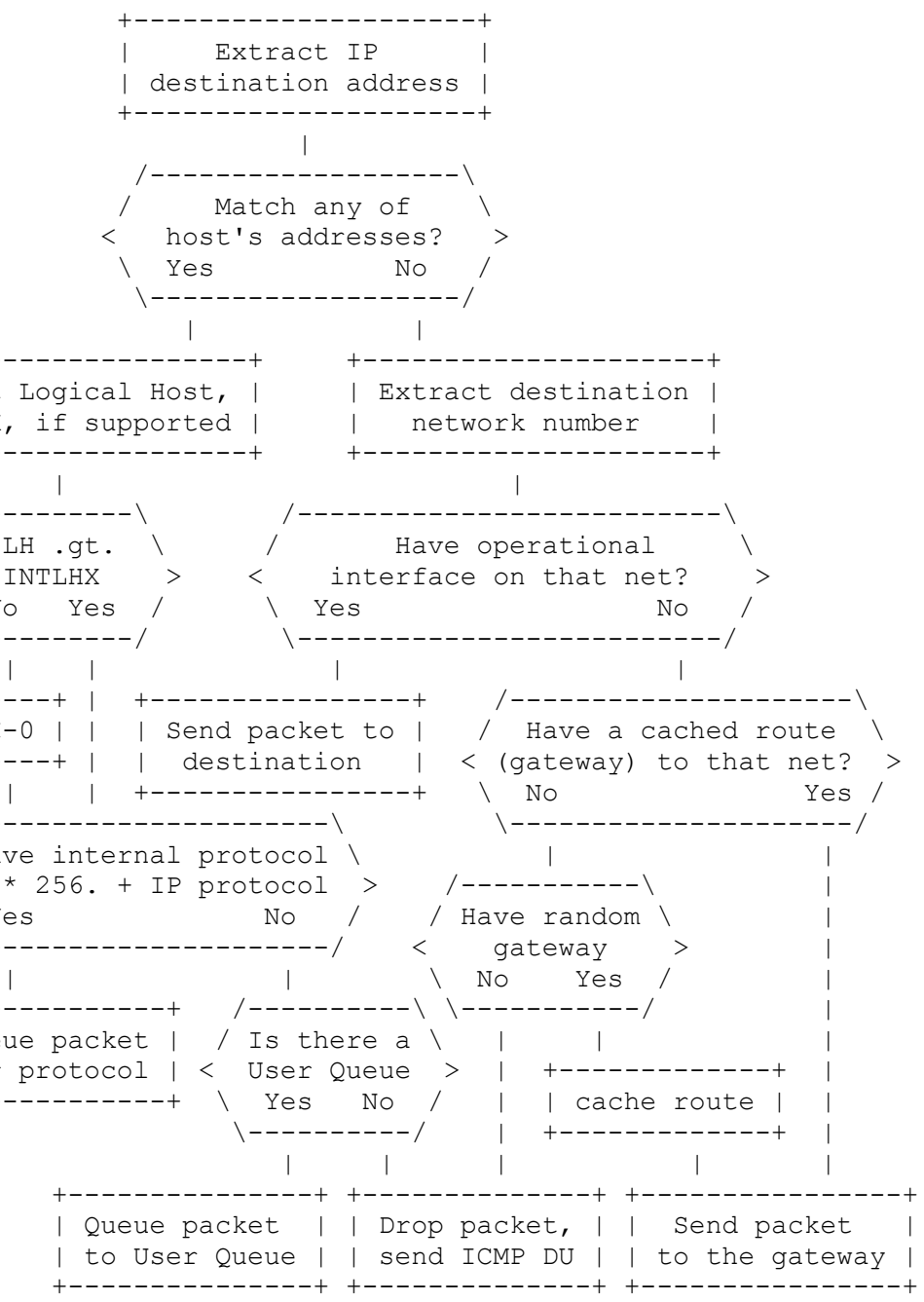


Figure 2. IP Packet Routing and Demultiplexing

When IP receives a datagram destined for (one of the addresses of) the local host, it must pass it to the appropriate protocol

module for further processing. There are two classes of protocols layered above IP: those implemented within the TOPS-20 monitor (such as TCP and ICMP) and those which are not.

Deciding which routines should receive a packet is based on the values contained in three fields. The first is the Internet Version Number; if it is not 4, the packet is discarded. The other two fields are the Internet Protocol field and the Logical Host field in the destination host address.

The TOPS-20 implementation divides the possible Logical Host numbers into two ranges:

- 0 to 3 (the default value of the system constant INTLHX), are all aliases for the TOPS-20, and
- greater than 3, which may be used by other hosts which interface to the network through the TOPS-20, or for alternate protocol implementations within the TOPS-20.

If the value in the Logical Host field is in the first range the packet is placed into the input queue of the internal protocol identified by the Internet Protocol field, if it exists.

If the value in the Logical Host field is in the second range the packet is placed into the input queue of the internal protocol identified by 256 times the Logical Host value plus the Internet Protocol field, if it exists.

In either case, if such a protocol has not been internally implemented the packet is passed to the User Queue dispatcher.

Datagrams for protocols not implemented within the monitor are associated with an application process through a User Queue. Demultiplexing to the correct User Queue is based on six fields from the datagram:

- Protocol code,
- Source and Destination Host addresses,
- Source and Destination Port numbers, and
- Logical Host number.

Each User Queue has a (Value,Mask) pair for each of these six fields. A datagram is associated with a User Queue if each field

from the datagram matches the corresponding Value in each of the bit positions where the Mask contains a one bit.

If no matching User Queue is found IP discards the datagram (possibly sending an ICMP Destination Unreachable message back to the datagram's source).

Thus, a packet for logical host 2, protocol 6 (TCP), would be passed to the standard TOPS-20 TCP module (since protocol 6 is implemented). A packet for logical host 55, protocol 6, would be passed to a User Queue if one has been assigned (which might be an experimental TCP implementation), since protocol 14086 (55*256+6) is not an internally implemented protocol.

This also means that no datagrams for a protocol that is internally implemented will be passed to an application via a User Queue. Assigning such a queue will be possible, but no datagrams will reach the application (unless the internal protocol module has been disabled).

4.5. Specifying a User Queue

An application specifies the six (Value,Mask) pairs using the ASNIQ% JSYS. Before assigning a User Queue, ASNIQ% asks the Access Control Job (function .GOAIQ) if the application can use the requested queue (Value,Mask) pairs.

If there is no Access Control Job, access is allowed if:

1) the user has WHEEL, OPERATOR, ABSOLUTE-ARPANET-SOCKETS, or ARPANET-WIZARD capability enabled, or

2) the user did not have the capabilities but either

2a) the Source Port Mask ANDed with the Source Port Value is greater than 255, or

2b) the Logical Host Mask ANDed with the Logical Host Value is in the range assignable to applications (greater than the value of the configuration variable INTLHX).

Otherwise, access is disallowed.

If access is allowed, the (Value,Mask) pairs are examined to make sure that they differ in some bit position from each of the previously assigned User Queues. If a difference is found then

datagrams can be unambiguously demultiplexed. ASNIQ% assigns a User Queue and returns a Queue Handle (a small system wide number identifying the User Queue, similar in function to a JFN).

4.5.1. ICMP Error Messages

ICMP error messages relating to datagrams are normally discarded when received and may not be sent. The application may, however, specify that ICMP messages should be sent and delivered to it (in the latter case the application must decide if a datagram is an ICMP message). To enable this mode, the AQ%ICM flag must be specified when the queue is assigned. The "included packet" is used when checking to see if the ICMP message is valid for a User Queue.

4.6. Sending Datagrams

An application sends datagrams by passing the Queue Handle and the address of a buffer containing an IP packet to the SNDIN% JSYS. If the local host address has not been specified (i.e., the IP Source Host address is zero), it is filled in with the "best" local host address for the destination (as defined by the BSTADR routine). Then SNDIN% checks various fields of the IP header for legality and to make sure that the datagram corresponds to the User Queue's (Value,Mask) pairs. If an error is detected, SNDIN% will fail. If no error is detected, the IP checksum is computed and placed into the IP header. Then the routing algorithms select a first hop destination, as described above. A local leader to the resulting destination is constructed, and the packet is queued for output on the appropriate interface.

Applications can affect the routing decision in two ways. They may specify a route using one of the IP Source Routing options (see section 4.13, IP Options). They may also specify an explicit first-hop destination internet address as an argument to the SNDIN% JSYS. The first-hop address must be on a directly connected network and have a functioning interface.

4.7. Receiving Datagrams

Datagrams are received by passing a Queue Handle and a buffer address to the RCVIN% JSYS. RCVIN% normally blocks until it receives a datagram from the network for the User Queue. When a

datagram is available, it is placed into the application supplied buffer. Applications have no knowledge of which interface received an incoming packet (although a pretty good heuristic is to examine the IP Destination address). If RCVIN% is called with the RIQ%NW flag set, and there are no available datagrams to be returned, it will return immediately with an error code of -1 in the right-half of AC1.

4.8. Releasing a User Queue

The RELIQ% JSYS is used to release a User Queue when the application has finished with it. Its argument may be:

- a specific Queue Handle,
- -1, to release all User Queues owned by the job, or
- a process handle (job wide or fork relative).

4.9. Implementation Notes

All User Queue JSYSi use only the LEFT 32 bits of each word. This is true of both the ASNIQ% argument block containing the (Value,Mask) pairs, and User Queue Buffers.

A User Queue, which contains a received datagram, is flushed if the datagram is not RCVIN%ed within 30 seconds. The number of messages held for RCVIN% is limited to 8. Thus, flooding a slow application results in dropped messages. (Note that the stated limits are network parameters (INQT0 and INQMX) and can be varied from site to site using the SET command in the SITE-ADDRESS file.)

Applications using User Queues must handle the cases of datagrams not being delivered in the order in which they were sent, of being dropped, or being duplicated. Some may traverse a broadcast network and be clobbered by other packets, lightning, etc. Recovery from these problems usually requires a higher level protocol such as TCP, the User Datagram Protocol (UDP [13]), XNET [14], etc.

4.10. Ports

Not all internet protocols have ports. If an application does not use them, be sure that .IQPTM in the ASNIQ% block contains a zero. If the protocol uses both local and foreign ports, they must be in the first two 16-bit bytes following the IP header, Source Port in left-most 16 bits and Destination Port in the next 16 bits. The location of this word is found by adding the contents of the IP Data Offset field to the address of the zeroth word of the IP header.

Provision has been made for protocols, such as XNET [14], which have only one port. The AQ%SPT flag for ASNIQ% selects single port mode. The first 16-bit byte following the IP header must contain the port. This port is used for demultiplexing incoming packets and checking access when packets are sent.

Note that for a given set of (Value,Mask) pairs, a single port protocol queue will not be assigned if a dual port version already exists, and vice versa. This rule permits an incoming packet to be uniquely associated with a queue. Without the rule, either the first or second 16-bit field following the IP header would have to be used for demultiplexing, but one could not decide which queue without some external information to tell whether one or two ports was present.

4.10.1. ICMP Diagnostic Messages

To send and receive ICMP messages such as Echo, Information Request, Timestamp, etc., which do not contain an "included packet", ICMP is considered to be a "single port" protocol. The AQ%SPT flag should be set for ASNIQ%, and a port (Value,Mask) pair specified. The "port number" occupies the ICMP header's ID field (left-most 16 bits of the SECOND word of the ICMP header, see [5]). To minimize ASNIQ% "queue already assigned" errors, use the job number for left-most 8 bits of the "port number".

4.11. User Queue Buffer

The User Queue Buffer begins with a word containing the length, in words, of the buffer, including the count word, followed immediately by an IP header of five or more words and the contents of the IP datagram (the "data").

The right-half of the length word contains, for SNDIN%, the length, in words, of the IP packet to be sent, plus one for the length word. For RCVIN% the right-half should contain the maximum number of words, including the length word, available for a received IP packet. The left-half will be set to the number of words actually received plus one for the length word.

Field	Contents
B0-17	RCVIN%: Set by RCVIN% to the number of valid words actually used by the IP packet, plus one (this word).
B18-35	RCVIN%: Maximum number of words in the buffer, including this word.
	SNDIN%: Number of words in the IP packet to be sent, plus one (this word).

4.12. IP Header

The IP header and data use the left-most 32 bits of each word. The fields in the IP header are given below. The structure names can be defined by invoking the DEFIP. macro which is defined in the MONSYM library (i.e., DEFIP.).

Symbol	Width	Contents
IPVER	4	Internet Version Number. This field should contain the value .INTVR (defined in MONSYM).
IPDO	4	Data Offset. The length of the IP header, in words (5 to 15).
IPTOS	8	Type-of-Service. The standard value is zero. It is composed of the following sub-fields:

IPPRC 3 Precedence.

IPLDY 1 Set if the user is willing to pay for special Low Delay handling and routing.

IPHTR 1 Set if the user is willing to pay for special High Throughput handling and routing.

IPHRL 1 Set if the user is willing to pay for special High Reliability handling and routing.

Note that IPLDY, IPHTR, and IPHRL should not be set simultaneously. See section 4.1, IP Type-of-Service.

IPPL 16 Datagram Length, in octets, of the IP datagram (header plus any data) (20 to 2**16-1).

IPSID 16 IP Segment ID. This field should be unique for each datagram sent by a protocol; it is used by IP to reconstruct a fragmented datagram.

IPFLG 3 IP Flags, containing:

IPDF 1 Do-not-Fragment flag. If set, a packet may not be fragmented by an internet entity. If the packet is too large to process, it will be discarded; an ICMP error message may be returned.

IPMF 1 More-Fragments flag. Set if this is NOT the last fragment of a packet.

IPFO 13 Fragment-Offset. Byte position of the first data byte of this datagram (fragment) in the original un-fragmented datagram, divided by 8 (all fragments, except possibly the last, are a multiple of 8 octets in data length). Since applications do not normally send fragments, both IPMF and IPFO should be zero. They should be zero in all packets delivered by RCVIN%, but may be set in the included packet of certain ICMP messages.

IPTTL 8 Time-to-Live, in seconds. If the packet cannot be delivered in the time specified, the packet should be discarded. The specified Time-to-Live should be greater than zero. An ICMP error

message may be returned if the packet cannot be delivered within the specified Time-to-Live.

IPPRO 8 IP Protocol number. IP protocol numbers are listed in the current version of "Assigned Numbers" [11].

IPCKS 16 IP header checksum. The contents of this field are automatically computed by SNDIN%. The application process does not have to compute it. Note that the IPOPR% JSYS function .IPCKS is available to compute the 16-bit ones-complement checksum used in the DARPA Protocol Suite.

IPSH 32 Source host internet address.

IPDH 32 Destination host internet address.

B0-31 32 Zero to ten words of IP options, if desired. If necessary, the last word of options is padded with zero octets (END) to fill the word. (See section 4.13, IP Options).

Protocols using port(s) must place them in the word immediately following the IP header. (Except as previously noted for ICMP diagnostic messages.)

B0-15 16 Port number for single-port protocols. Source port number for two-port protocols. Nonexistent for portless protocols.

B16-31 16 Destination port number for two-port protocols, or nonexistent for single-port or portless protocols.

4.13. IP Options

IP provides the means to include a selection of standard options in packets. Options are only required in special situations. An IP option has the form of a string of 1 to 40 octets.

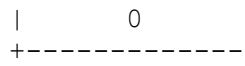
Except for NOP and END which are each 1 octet long, an individual option is a string of 2 or more octets. The first octet contains the option code, the second the length in octets of the option string (counting the code and length octets), followed by additional octets relevant to the particular option.

> Note: This is only one interpretation of the formal specification. It allows new (experimental) options of 2 or more octets to be defined and used without requiring all implementations to be simultaneously changed; only defining new 1 octet options will require such a change. Another interpretation which some implementations use is that only the options defined in the specification can appear; anything else causes the options to be unparsable and the packet discarded.

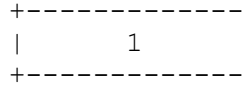
The most significant bit of an option code indicates, if set, that the option must be copied into each fragment of a fragmented datagram.

The currently defined IP options are:

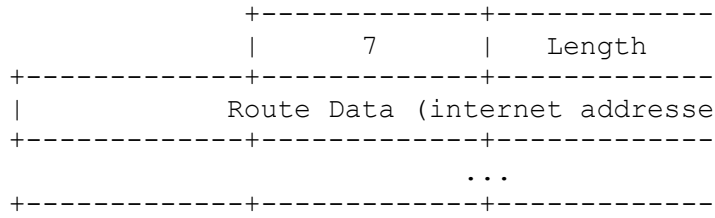
Option Code	Option (Length)	Meaning
0	END (1 octet).	Terminates the option string.



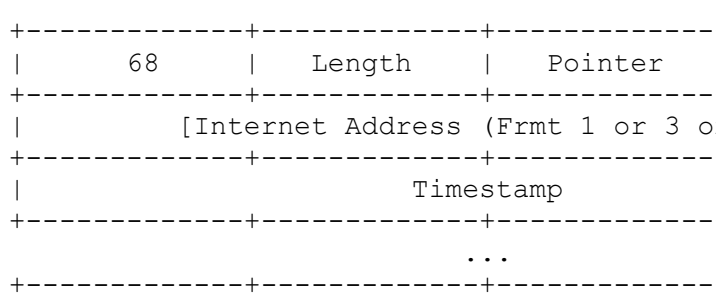
1	NOP (1 octet).	Ignored.



7 Record Route (7 to 39 octets). The originator of this option provides space for each IP entity which processes this datagram to insert the internet address of the interface it will use to send the datagram. The address is inserted into the Route Data area at the position specified by the Pointer (whose minimum value is 4). As each address is added, the value of the pointer is increased by 4. If the value of the pointer exceeds the value of the Length, the Route Data area is full and some IP entities may not have been able to insert their address.



68 Internet Timestamp (8 to 40 octets). The originator of this option provides space for intermediate IP entities to record the time when the packet was processed and, optionally, their internet address. The option begins with the option code and Length octets followed by Pointer and Control octets and a data area. Each entry in the data area contains four octets.



The 4-bit Frmt field specifies the format of the the data portion of the option:

0 Only Timestamps appear,

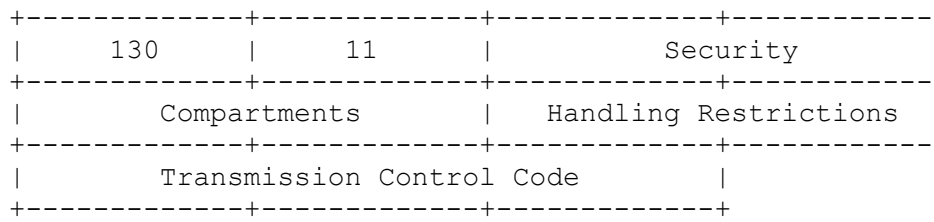
1 Each IP entity inserts its address before its timestamp.

3 The originator has prespecified the internet addresses of those IP entities which are to insert their timestamps. An entity only enters its Timestamp if (one of) its address(es) is next.

The Pointer specifies the current position in the data area for the next (Internet Address and) Timestamp. The minimum value of 5 indicates the first entry (the octet following the Ovfl/Frmt octet). It is incremented by 4 (Frmt 0) or 8 (Frmts 1 and 3). If it points beyond the last data entry (i.e., Pointer exceeds Length), no further entries can be made. The Ovfl field is incremented in this case (Frmts 0 and 1).

The format of the Timestamps is the number of milliseconds since midnight, Universal Time. If the time is not known in that format, any time may be used with the left-most bit set.

130 Security (11 octets). SECURITY, COMPARTMENTS, HANDLING RESTRICTIONS, and TRANSMISSION CONTROL parameters are specified.



The option code and length octets are followed by two octets which specify one of sixteen SECURITY levels. The valid values are:

```
00000000 00000000 - Unclassified
11110001 00110101 - Confidential
01111000 10011010 - EFTO
10111100 01001101 - MMMM
01011110 00100110 - PROG
10101111 00010011 - Restricted
11010111 10001000 - Secret
01101011 11000101 - Top Secret
00110101 11100010 - (Reserved for future use)
10011010 11110001 - (Reserved for future use)
```

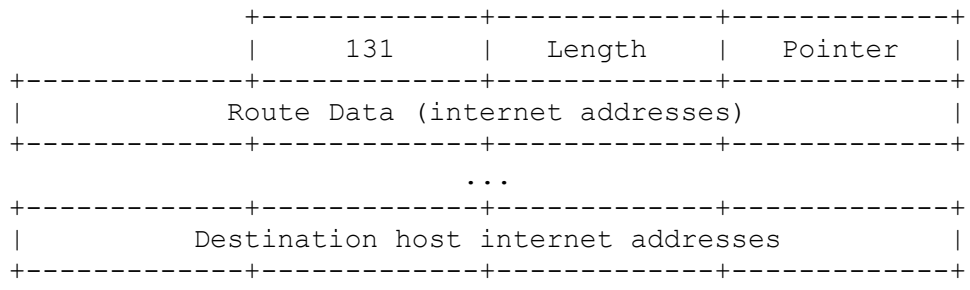
01001101 01111000 - (Reserved for future use)
00100100 10111101 - (Reserved for future use)
00010011 01011110 - (Reserved for future use)
10001001 10101111 - (Reserved for future use)
11000100 11010110 - (Reserved for future use)
11100010 01101011 - (Reserved for future use)

Two COMPARTMENTS field octets follow the Security octets. They may be obtained from the Defense Intelligence Agency.

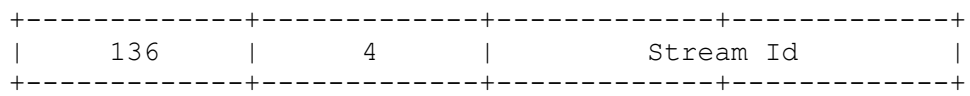
Next are two HANDLING-RESTRICTIONS octets. The control and release markings are given in the "Standard Security Markings" section of Defense Intelligence Agency Manual DIAM 65-19.

The last three octets are used to segregate traffic and define controlled communities of subscribers. TRANSMISSION-CONTROL values may be obtained from HQ DCA Code 530.

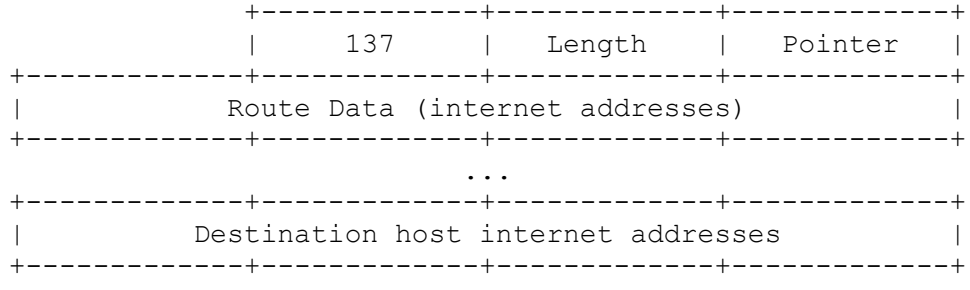
131 Loose Source Route (11 to 39 octets). The originator of this option specifies the addresses of IP entities through which the datagram must pass on its way to the destination. The last entry in the route must be the destination host address. The Pointer indicates the next address in the Route Data area; its initial value is 4. The last address in the Route Data area must be the destination address. As each entry is processed, the IP entity's sending interface internet address replaces the next address, and the Pointer is increased by 4. If the value of Pointer exceeds that of Length, the source route is exhausted and the option is ignored (sent to the destination in the IP header). The route is Loose because any number of intermediate gateways may be used between successive Route Data entries.



136 Stream Identifier (4 octets). The two-octet SATNET stream ID is passed through the internet in this option.



137 Strict Source Route (11 to 39 octets). The originator of this option specifies the addresses of EXACTLY those IP entities through which the datagram must pass on its way to the destination. The last entry must be the destination host address. The Pointer indicates the next address in the Route Data area; its initial value is 4. The last address in the Route Data area must be the destination address. As each entry is processed, the IP entity's sending interface internet address replaces the next address, and the Pointer is increased by 4. If the value of Pointer exceeds the value of Length, the source route is exhausted and the option is ignored (sent directly to the destination in the IP header). The route is Strict because only those gateways specifically listed may be used.



4.14. ASNIQ% JSYS 756

Assign a User Queue; the assignment must be permitted by the Access Control Job, or ARPANET-WIZARD capability must be enabled.

Accepts In AC1: Flags ! Address of User Queue Descriptor block.

AQ%SPT Set for single-port protocols, cleared for two-port or portless protocols.

AQ%ICM Deliver ICMP error messages to this queue and allow them to be sent from it.

AC2: Reserved, must be 0.

AC3: Reserved, must be 0.

Returns +1: Failure, TOPS-20 error code in AC1; conflicting job number in AC2.

+2: Success, User Queue Handle returned in AC1 and maximum IP packet size (before fragmentation will occur) in AC2.

4.14.1. User Queue Descriptor Block

The format of the Queue Descriptor Block, which contains the (Value,Mask) pairs, is given below; the block contains .IQLEN words. The symbols are defined in MONSYM.

Offset	Bits	Usage
.IQPRV	B0-23	Logical Host number (if supported by the directly connected network(s)); zero if no logical host is used.
	B24-31	IP Protocol number.
.IQFHV	B0-31	Foreign host internet address.
.IQSHV	B0-31	Local host internet address.

.IQPTV B0-15 Local port (or the single port if AQ%SPT is set).
B16-31 Foreign port (ignored if AQ%SPT is set).

.IQPRM B0-23 Mask for the Logical Host, if supported and used, or zero.
B24-31 Mask for IP Protocol number.

.IQFHM B0-31 Mask for foreign host internet address. (Zero for any foreign host.)

.IQSHM B0-31 Mask for local host internet address. (Zero for any of this host's addresses; SNDIN% will select the "best" local host address on a packet by packet basis.)

.IQPTM B0-15 Mask for local or single port.
B16-31 Mask for foreign port (zero if AQ%SPT is set). Note B0-31 should be zero for portless protocols.

The mask words specify bit positions that require an exact match. Thus, one can make .IQFHM contain 0 to send or receive datagrams from all internet hosts. For example, by making the low 3 bits of the local port mask word 0 the User Queue owns eight consecutive ports.

ASNIQ% will return an error unless the current Queue Descriptor block differs in the masked bits from all other assigned User Queues.

ASNIQ% Error Mnemonics:

NTWZX1 ARPANET-WIZARD capability, or permission required from Access Control Job, required.

ASNSX1 All User Queues in use or IP not initialized.

ASNSX2 The (Value,Mask) pairs conflict with the job whose job number is in AC2.

4.15. RELIQ% JSYS 757

Release a User Queue.

Accepts In AC1: Queue Handle, or

-1 for all User Queues owned by this job, or

a job or fork-relative process handle.

AC2: Reserved, must be 0.

AC3: Reserved, must be 0.

Returns +1: Failure, TOPS-20 error code in AC1.

+2: Success.

RELIQ% releases ownership of a User Queue so other jobs can assign it. See the TOPS-20 Monitor Calls Reference Manual [2] for a description of the various process handles.

RELIQ% Error Mnemonics:

SQX1 Queue Handle out of range.

SQX2 Queue Handle not assigned.

4.16. SNDIN% JSYS 754

Send an IP datagram.

Accepts In AC1: Queue Handle.

AC2: Extended address of User Queue Buffer containing the IP packet.

AC3: Zero if normal routing is desired, or the first-hop internet address on a directly connected net with a functioning interface, where the datagram should be sent.

Returns +1: Failure, TOPS-20 error code in AC1.

+2: Success.

The User Queue Buffer, described above, must contain:

- a word count (including the count word) in the right-most 18 bits of word-0,
- a valid IP header in the left-most 32 bits of words 1 through 5, and
- optional IP options and or data in the left-most 32 bits of words 6 and following.

If the Source Host field in the IP header has not been specified, the "best" local host address will be used. The IP header checksum will be computed and placed into the packet. The application supplies the rest of the header.

SNDIN% Error Mnemonics:

SQX1 Queue Handle out of range.

SQX2 Queue Handle not assigned.

SNDIX1 Invalid message size. There is a discrepancy between the buffer word count, the IP Packet length field, and the AQ%SPT flag or .IQPTM mask.

SNDIX2 Insufficient resources; no buffers available.

SNDIX4 Invalid header value for this queue. Includes:

- Datagram Length too big,
- Data Offset too small,
- filtering on ports but Datagram Length is too small for packet to contain a port word,
- header does not match the ASNIQ% arguments, or
- attempt to send an ICMP message without having set AQ%ICM.

4.17. RCVIN% JSYS 755

Receive IP datagram.

Accepts In AC1: Flags ! Queue Handle.

RIQ%NW Set to cause fail-return instead of blocking when there is no datagram waiting.

AC2: Extended address of a User Queue Buffer to hold the IP packet.

AC3: Reserved, must be 0.

Returns +1: Failure, TOPS-20 error code in AC1.

+2: Success.

Each RCVIN% returns one datagram from the specified queue. These datagrams match the values in the User Queue Descriptor block when masked by the mask words in the block. The maximum size of the User Queue Buffer (including the count word) when the RCVIN% is executed should be in the right-half of word-0 of the User Queue Buffer.

When a datagram is available, it is placed into the application supplied User Queue Buffer. The size in words of the datagram plus one (for the count word) is set into the left-half of word-0 (the count word). Ordinarily this can be ignored, but if the datagram was too big for the User Queue Buffer, this tells how much space would have been required. If the datagram was too big (determined by the Datagram Length field), as much of the datagram as will fit into the User Queue Buffer is transferred and an error is returned. No retry for the same datagram is possible.

RCVIN% Error Mnemonics:

SQX1 Queue Handle out of range.

SQX2 Queue Handle not assigned.

SNDIX1 Invalid datagram size.

777777 RIQ%NW was on and no datagrams were waiting.

4.18. IPOPR% JSYS 760

Network and Internet Protocol Operations

The IPOPR% JSYS performs general network functions, including those related to the DARPA Protocol Suite which are not specific to a TCP connection (see section 2.4.8, TCOPR%).

Accepts In AC1: Function Code.

AC2: Function Dependent Argument.

AC3: Function Dependent Argument.

Returns +1: Always. An error will cause an Illegal Instruction Trap, with AC1 containing an error code.

Function	Offset	Description
.IPSNT		Change network state. AC2 contains the network number (ARPANET is number 10.). AC3 contains the desired network state (-1 for on, <0,,-1> to cycle, and zero for off). (Note: requires WHEEL, OPERATOR, ARPANET-WIZARD, or MAINTENANCE capability enabled.)
.IPRNT		Determine network state. AC2 contains the network number (ARPANET is number 10.). Network state is returned in AC3 (zero for off, -1 for on, or <0,,-1> for cycling).
.IPINI		Reload host translation tables from SYSTEM:HOSTS.TXT. The host tables are locked (see section 5.2, GTHST% GH%WAT flag), the contents flushed, and the entries from the file installed. This process may take a few minutes to complete. (Note: requires WHEEL, OPERATOR, ARPANET-WIZARD, or MAINTENANCE capability enabled.)

.IPGWY Reload gateway table from SYSTEM:INTERNET.GATEWAYS. (Note: requires WHEEL, OPERATOR, ARPANET-WIZARD, or MAINTENANCE capability enabled.)

.IPRIB Read the state of the loop-back interface. The state is returned in AC2 (zero for off, or -1 for on).

.IPSIB Set the state of the loop-back interface. AC2 contains the desired state (-1 for on, <0,,-1> to cycle, and zero for off). (Note: requires WHEEL, OPERATOR, ARPANET-WIZARD, or MAINTENANCE capability enabled.)

.IPNIP IP on the NI - Enable/Disable NI IP portal, AC2 is zero for off, non-zero for on.

.IPNAP IP on the NI - Enable/Disable NI ARP portal, AC2 is zero for off, non-zero for on.

.IPIGH IP on the NI - Reload NI IP GHT from the SYSTEM:INTERNET-ETHERNET-MAPPINGS.BIN file.

.IPRGH IP on the NI - Return NI IP GHT table. (Not yet implemented.)

.IPRIC IP on the NI - Return NI IP portal counters.

.IPRAC IP on the NI - Return NI ARP portal counters.

.IPNIF Initialize network interfaces as specified in the SYSTEM:SITE-ADDRESS.TXT file (using the ADRINI routine). When invoked, all interfaces are shut off (causing any packets which are being queued for output to be rejected) and the new interfaces are turned on (using the MNETON routine). (Note: requires WHEEL, OPERATOR, ARPANET-WIZARD, or MAINTENANCE capability enabled.)

.IPRIP Read a block of network information. AC2 contains the address of argument block, containing at least 4 words plus the header word.

BEWARE: the order/entries are site dependent. Use of numeric offsets is not recommended. Use NT%SD and the ASCII names.

The format of the argument block is:

.NTLEN Length of block, including this word.

.NTFLG Flags ! Id

NT%IX Return information about the connection specified by the index (1 to I) in the right-half of .NTFLG. The index is a system-wide identifier whose binding may change from call to call. (See NT%NI.)

NT%JS Return information about the connection whose JCN is in the right-half of .NTFLG. (Note: Use the .TCRTC function of TCOPR% for TCP: JFNs.)

NT%NI Return in .NTNMP an AOBJN counter for the currently active connections. (See NT%IX.)

NT%NT Return in .NTNMP an AOBJN counter for the currently active TVTs. (See NT%TV.) (Note that the right-half of .NTNMP contains the terminal line number of the first TVT.)

NT%SD Return an LDB pointer for each of the specified symbols instead of the value. The byte pointers returned assume that AC14 (octal) is pointing to a TCB image and AC13 (octal) is pointing at STAT0 (offset 0 of the statistics area). Requires NT%SY be set.

NT%SL Return in the area pointed to be .NTLNP the length, in LDB bytes, of each datum. Requires NT%SY be set.

NT%ST Return network statistics information instead of information about a TCP connection. The right-half of .NTFLG is ignored. (See NT%SY).

NT%SY The right-half of .NTNMP contains the address of a list of ASCII variable names associated with either a TCP connection (TCB) or network statistic for which information should be returned. See NT%SD and NT%SL.

NT%TV Return information about the TVT whose line number is specified in the right-half of .NTFLG. (See NT%NT.)

.NTNMP Specifies the source. The contents of this word depend on the setting of the NT%SY flag. When NT%SY is set, .NTNMP contains:

-N,,Address of ASCII symbol names

N symbols are given beginning at the specified application address. The symbols must be upper case ASCII containing one to five characters. If NT%SY is not set, .NTNMP contains:

-N,,Offset into TCB

N successive words beginning at the specified offset are to be returned.

.NTDTP Specifies the destination. This word contains:

-M,,Address in application space

The data to be returned should be placed beginning at the specified application address; the block is M words long. Note that the returned data are right justified and that some TCB entries contain more than one word of data (e.g., the option fields).

.NTLNP If NT%SL is not set, this word is not used. When NT%SL is set the word contains

0,,Address in application space

The number of LDB bytes associated with the specified symbols is placed beginning at the specified application address; the block is also M words long.

The counts and pointers in .NTNMP, .NTDTP, and, if NT%SL was specified, .NTLNP are updated.

Failure due to a bad JCN, index, offset, or an illegal symbol causes an Illegal Instruction Trap; a TOPS-20 error code is returned in AC1.

The names of some of the variables in network statistics area which may be used with .IPRIP are listed below after the error mnemonics. See the NTITEM entries in the DEFSTS macro in ANAUNV.MAC for a complete listing.

.IPCKS Compute the 16-bit ones-complement internet checksum.

Accepts In AC1: .IPCKS.

AC2: Zero, or (right justified) sum to be included in the checksum in bits B0-35 (e.g., the sum of TCP's "pseudo header").

AC3: Extended address of first byte to be checksummed. Data is assumed to occupy B0-31.

AC4: Number of 8-bit bytes to be checksummed.

Returns +1: Always.

AC1: 16-bit checksum; right justified.

.IPDOM Reload domain tables from DOMAIN:FLIP.DD and FLOP.DD. (Note: requires WHEEL, OPERATOR, ARPANET-WIZARD, or MAINTENANCE capability enabled.)

IPOPR% Error Mnemonics:

MNTX00 No errors detected.

MNTX01 .IPNIF failed to find SYSTEM:SITE-ADDRESS.TXT file.

MNTX10 .IPINI failed to find SYSTEM:HOSTS.TXT file.

MNTX13 Invalid host or network specified.

MNTX15 Invalid value.

TCPX23 Invalid IPOPR% function requested.

TCPX24 WHEEL, OPERATOR, or NETWORK-WIZARD capabilities needed for special IPOPR% function.

TCPX34 Illegal .IPRIP flag or argument.

TCPX44 Monitor does not support TCP over Ethernet.

4.18.1. Network Statistics Names

The names of some of the variables in network statistics area which may be used with .IPRIP are listed below. See the NTITEM entries in the DEFSTS macro in ANAUNV.MAC for a complete listing.

The following TCP statistics are always collected:

Statistic Symbol	Contents
BGRNC	Number of times the TCP background process (BG) has run.
BYTRC	Total number of data octets received.
BYTSC	Total number of data octets sent.
DGRNC	Number of times the TCP delayed action process (DG) has run.
DUPKC	Total number of duplicate packets received.
FINRC	Total number of FIN packets received.
FINSC	Total number of FIN packets sent.
IPPKC	Total number of packets received.
IPRNC	Number of times the TCP input process (IP) has run.
OPPKC	Total number of packets (instances) sent (includes retransmissions).

OPRNC Number of times the TCP output process (OP) has run.
PZPKC Total number of packets generated.
PZRNC Number of times the TCP packetizer process (PZ) has run.
RAPKC Total number packets reassembled.
RARNC Number of times the TCP reassembler process (RA) has run.
RSTRC Total number of RESET packets received.
RSTSC Total number of RESET packets sent.
RXPKC Total number of packet (instances) retransmitted.
RXRNC Number of times the TCP retransmiter process (RX) has run.
SYNRC Total number of SYN packets received.
SYNSC Total number of SYN packets sent.
TASKC Total number of times a TCP process has been run.

The following TCP statistics are not usually collected and will probably be zero.

Statistic Symbol	Length	Contents
ACDLA	17.	Activation delay histogram.
BGUSE		Total time charged to background process.
DGUSE		Total time charged to delayed action process.
IPDLA	17.	Input delay histogram.
IPUSE		Total time charged to input process.
OHUSE		Total time charged to TCP overhead.
OPDLA	17.	Output delay histogram.
OPUSE		Total time charged to output process.
PZDLA	17.	Packetizer delay histogram.
PZUSE		Total time charged to process.
RADLA	17.	Reassembler delay histogram.
RAUSE		Total time charged to reassembler process.
RXDLA	17.	Retransmiter delay histogram.
RXUSE		Total time charged to retransmiter process.

5. Multinet

The first implementation of networking on TOPS-20 was the ARPANet. At that time networking was a new technology, so the possibility that a system might be connected to more that one network or use more than one protocol suite wasn't considered. As other networks were developed, various methods were used to connect to them, but each connection was special purpose: each network had a hardware interface, and each interface had a module driving that network's protocol. All these connections were independent. In addition, the hooks to drive the hardware were usually an integral part of the protocol module.

To expand such a configuration to handle multiple interfaces, or to allow more than one protocol suite to use a single interface, is not a simple task. The advent of internetwork communication emphasized this problem, since this introduced a module that should be able to use all the interface hardware, without regard for the individual network protocols. The first Internet implementation did this by hard wiring knowledge about each network type, and its protocols, in the Internet software. Thus is unwieldy, difficult to modify, and basically unmodular. Figure 3 shows how a representative network configuration might look under this scheme.

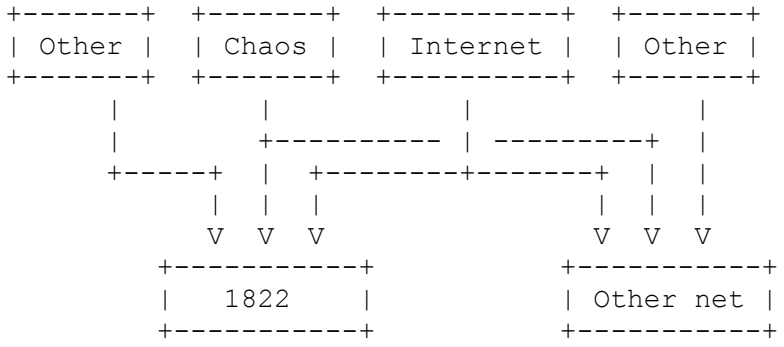


Figure 3. Network Configuration Without Multinet

Multinet's purpose is to provide network (device and protocol suite) independence by creating a common interface between the hardware driving the networks and protocol suites. Figure 4 shows such a configuration. The higher level protocol suites must now only be concerned with their protocols and their application interface. The multinet layer handles:

- o the various addresses by which the host is known,
- o managing the routines driving the interface hardware, and
- o packet dispatching (taking a packet from the protocol suites and enqueuing it for output; receiving a packet from a network and delivering it to the proper protocol suite).

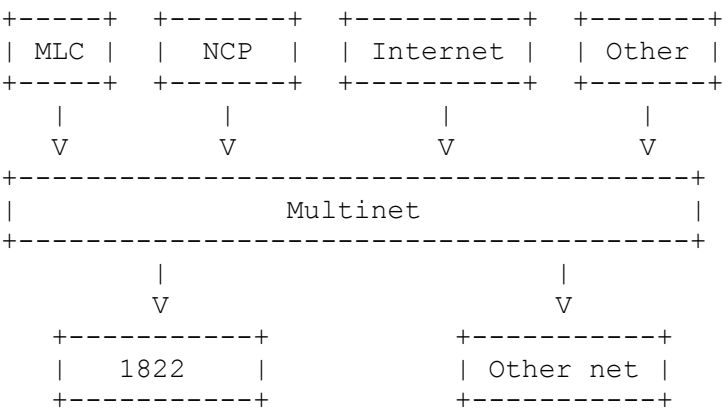


Figure 4. Network Configuration With Multinet

One of the design goals of multinet is to be as transparent as possible to the existing applications (old software). New (or updated) applications may take advantage of the multiple networks by using the GTHST% JSYS described below. Note that network control functions are invoked using the IPOPR% JSYS, described above.

The remaining sections describe the multinet layer from several different perspectives. The applications programmer will be interested in the description of the GTHST% JSYS. The host administrator should read Part 2, describing how to install the configuration files that are required by the network software. Part 3 contains sections describing the overall structure of the network software and how to add:

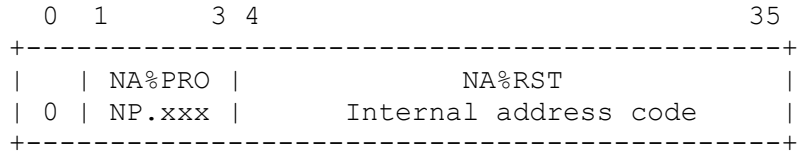
- o a network device driver,
- o a protocol suite, and
- o a protocol within IP.

Finally, for those who may have to maintain the software, there is a section describing how the multinet layer is organized.

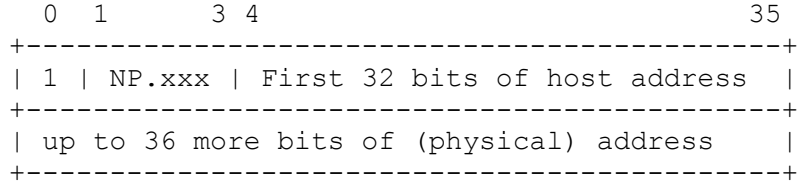
5.1. Network Address Formats

The ability to support multiple protocol suites means that network addresses may not all have the same format. Internet addresses contain 32 bits, Ethernet addresses contain 48 bits, and Chaos addresses contain 16 bits. In an attempt to hide these differences from the applications developer, the network addresses given to and returned by GTHST% are translated into a single word quantity (analogous to translating file specification strings into JFNs).

The format of these addresses need not be visible but will be shown for completeness. The names of the three fields in a standard format network address are NA%FLG (B0), NA%PRO (B1:3), and NA%RST (B4:35). NP.xxx is the protocol suite code (defined in ANAUNV). The "internal address code" is either the physical address, for suites with 32 or fewer physical bits in the address, or a translation table address, for suites that have more than 32 bits in the physical address.



In certain special internal situations where the actual physical address is required, NA%FLG is set and the following format is used:



5.2. GTHST% JSYS 273

The GTHST% JSYS is used to obtain information about network hosts and local interfaces. The number of hosts has grown so large that maintaining a complete table is not practical. A limited number of host entries are kept in the host translation cache. If information is requested for a host not found in the cache, a Name Server is consulted. This may take time. Information is also kept for selected gateways and networks.

The .GTHSN function, which translates a host name string to an address, accepts either completely specified or abbreviated domain names. Names which do not end in a dot are assumed to be fully specified, e.g., "BBNA.BBN.COM" or "BBNA", a nickname. Names which end in a dot are assumed to be abbreviations; the name is completed with the host's default completion string, e.g., the name "BBNA." would be completed to "BBNA.BBN.COM" if the host's string were "<3>BBN<3>COM" (see DOMPAR+7).

Note that the name strings are getting longer. The maximum length string is now about 330 characters. Soon old programs, which only provide space for 39 characters, will stop working correctly.

Accepts In AC1: Flags ! Function Code.

AC2: Function-specific argument.

AC3: Function-specific argument.

AC4: Function-specific argument.

Returns +1: Failure, TOPS-20 error code in AC1.

+2: Success, function-specific data in ACs 2-4.

Several function flags are defined. They allow an application to control the translation process.

Flag	Meaning
GH%ANY	Return a Host, Gateway, or Network.

GH%GWY Return a Gateway.

GH%NET Return a Network.

None of the above (old way), return a Host.

GH%MOD Mode used to find answer. The possible values are:

Mode	Description
.GHDEF	Use a resolver, if necessary, to find the answer (default mode). Note that some time may be required to find the answer; the application is willing to wait.
.GHLCL	Only local information should be consulted to find an answer. Fail if the answer is not locally available.
.GHNRF	If the resolver is required, it should not allow referrals.
.GHRF	If the resolver is required, it should allow referrals.

GH%INI If the answer is not locally available, initiate resolution but return a failure. When the answer is received, it will be cached. The application will repeat the request later; the answer should then be locally available.

GH%MBA The answer must be authoritative.

GH%VIR The resolver should use a virtual circuit because the reply is expected to be lengthy. Note that using GH%VIR might be better for replies which exceed a maximum packet size (536 octets), since correct reception of a fragmented datagram requires that all fragments be correctly received.

GH%RRF Source/Destination data is in resource record format. Host names consist of multiple strings (binary length octet, text octets), without dots. For example, "SRI-NIC.ARPA." would be "<7>SRI-NIC<4>ARPA<0>".

GH%CNM If the source name is a nickname, replace it with the canonical name. Note that there must be sufficient space for the canonical name to be returned.

The following flags are associated with host name strings. They are used by .GTHSN to determine which fields should be parsed and are updated based on the fields that were found. They are used by .GTHNS to specify how the name string should be printed.

Flag	Meaning
GH%SNM	Function .GTHSN did not parse an ASCII name string; .GTHNS will not print a name string.
GH%ADR	Function .GTHSN should parse an ASCII address string; .GTHNS will print the host address in ASCII.
GH%PSU	Function .GTHSN parsed a Protocol Suite Identifier; .GTHNS will include a Protocol Suite Identifier, including the "#" prefix character, in the name string.
GH%PRT	Function .GTHSN should parse a port number. .GTHNS will print a port number after a separating space.
GH%SPC	Function .GTHNS only. Print a trailing space.

The following flags may be returned by GTHST%:

Flag	Meaning
GH%AKA	Returned if the specified name is an alias.
GH%TRN	Returned if the reply was too long to fit into the space provided.

Several of the functions return information about a host. Some of the information is "static" and some is "dynamic". Note that the dynamic information may be stale; it is not possible keep track of all the hosts within the internet. The dynamic information is updated by observing traffic flow and processing error messages. The format of the host status word is defined in MONSYM.MAC, and reproduced below:

Field	Mask	Description
HS%UP	1B0	If HS%VAL is set: Host is up if 1, not known to be up if zero.
HS%VAL	1B1	HS%UP is valid.
HS%DAY	7B4	If known, Day when host will be up.
HS%HR	37B9	If known, Hour when host will be up.
HS%MIN	17B13	If known, 5 minute interval, within hour, when host will be up.
HS%RSN	17B17	If known, Reason why host is down.
HS%SRV	1B18	Host is a server.
HS%USR	1B19	Host is a user.
HS%NCK	1B20	Host name string is a nickname.
HS%STY	77B26	System type:
	1B26	.HS10X TENEX
	2B26	.HSITS ITS
	3B26	.HSDEC TOPS10
	4B26	.HSTIP TIP
	5B26	.HSMTP MTIP
	6B26	.HSELF ELF
	7B26	.HSANT ANTS
	10B26	.HSMLT MULTICS
	11B26	.HST20 TOPS20
	12B26	.HSUNX UNIX
	13B26	.HSNET Network
	15B26	.HSVMS VMS
	16B26	.HSTAC TAC
	17B26	.HSDOS MSDOS
HS%NEW	1B27	Host does new protocol (obsolete).
HS%NAM	1B28	Host has a name string.
HS%SLF	1B29	One of the entries for the local host. (Not yet implemented.)
HS%NET	1B30	Entry is a network.
HS%GAT	1B31	Entry is a gateway.
	0B30:31	Entry is a host.
HS%PRM	1B32	Entry should not be deleted from the (cache) table.

The GTHST% functions are:

Function	Description
.GTHSZ	Returns the negative number of host names, negative maximum number of hosts, and a local host network address. Use of the information returned in AC2 and AC3 is not recommended since the entries in network caches are always changing. Application supplied arguments: None. Returned data: AC2: -Number of host names,,0. (Do not use.) AC3: -Maximum number of hosts,,0. (Do not use.) AC4: One of the local host's network addresses.
.GTHIX	This function is obsolete and will no longer be supported. It formerly returned the name string, primary network address, and status of the specified host. Application supplied arguments: AC2: Destination descriptor. AC3: Index into name table. Returned data: AC2: Updated destination descriptor. AC3: Host's network address. AC4: Host's status.

.GTHNS Returns the name string, primary network address, and status of the specified host.

Application supplied arguments:

AC2: Destination descriptor.

AC3: Host's network address.

AC4: If GH%PRT is set, a right justified port number.

Returned data:

AC2: Updated destination descriptor.

AC3: Host's network address.

AC4: Host's status.

The format of the returned string depends upon the flag bits set in AC1. In general, the format of the string is as follows:

[<Name>][<ProtocolSuite>][<Address>][<Port>][<Space>]

Item	Description
Name	ASCII Host name, unless GH%SNM is set.
ProtocolSuite	'#' followed by protocol name (e.g., '#Internet' or '#Chaos'), if GH%PSU is set. Note the protocol suite is currently synonymous with a domain class. It also specifies how the Address should be parsed.
Address	ASCII Host address appropriate for the protocol, if GH%ADR is set or GH%SNM is not set and a name is not available.
Port	ASCII Port number, if GH%PRT is set.
Space	Space if GH%SPC is set.

.GTHSN Returns the primary network address and status of the host identified by the specified name string. The host status information is returned only if GH%PRT is not set. HS%NCK will be set if the name is a nickname. If GH%PRT is set, a port number following the host name or address string will be parsed and returned in AC4.

Application supplied arguments:

AC2: Source descriptor.

Returned data:

AC1: Updated flags, GH%SNM, GH%PSU, GH%ADR, if the associated fields were processed.

AC2: Updated destination descriptor.

AC3: Host's network address.

AC4: Host's status (GH%PRT zero) or Port number (GH%PRT set).

.GTHHN Returns the specified host's status. Note that the status for a host depends on which of its network addresses is specified.

Application supplied arguments:

AC3: Host's network address.

Returned data:

AC3: Host's network address.

AC4: Host's status.

.GTHHI This function is obsolete and will no longer be supported. It formerly returned the network address and status of the host specified by the index in AC3.

Application supplied arguments:

AC3: Host index.

Returned data:

AC3: Host's network address.

AC4: Host's status.

.GTHLN Returns a local host network address on a specified network.

Application supplied arguments:

AC2: Network number, network address, or a host on that network.

Returns

+1: No interface exists on the specified network.

+2: A interface exists (it may not be operational).

AC2: One of the local host's address on the specified network.

.GTHNT Returns the status of one of the local host's network interfaces.

Application supplied arguments:

AC2: Network number.

AC3: Address for returned block of information.

AC4: -Number of entries to be returned,,Offset of first entry to return.

Returns

+1: No interface exists on the specified network.

+2: A interface exists; data has been returned.

Returned data:

Offset	Description
0	Interface status. 0 means down, >0 means going down, and <0 means up.
1	Desired Interface status. 0 means off, <0 means on, >0 means cycle.
2	Network has an unreported state change if >0.
3	Network has output enabled if -1.
4	Time and Date. Last time network came on internally.
5	Time and Date. Last time network went off externally.
6	Time and Date. Last time network came on externally.
7	Time and Date. Last time network went off internally.

.GTHRT Returns the network address of the first hop in the current route to the specified network.

Application supplied arguments:

AC2: Network number, or network address of a host on that network.

Returns

+1: There is no current route to the specified network. If error code GTHSX5 (Route not known - would try gateway) was returned, the address of the gateway to which a packet would have been sent is returned in AC3. Since gateways are chosen "randomly", the address returned in the latter case will vary from call to call.

+2: The current first-hop route to the specified network is returned in AC3.

.GTHRR Returns Resource Records associated with Network Domains. See RFC 882 [15] and 883 [16] for detailed descriptions of domain names and Resource Record Formats.

The lookup operation to be performed is specified in the GH%OPR field. It has 4 possible values (only .GHSTD is currently implemented).

Value	Description
.GHSTD	Standard query. The source is a domain name.
.GHINV	Inverse query. The source is the "answer", the "question" will be returned to the destination.
.GHCM	Completion query; multiple replies may be returned.
.GHCU	Completion query; the reply must be unique.

The class or protocol suite of the information to be returned is specified in the GH%CLA field. It currently has 3 possible values.

Value	Description
.GHIN	Internet.
.GHCS	CSNet.
.GHSTR	Wild.

The type of information to be returned is specified in the GH%TYP field. It has 18 possible values.

Value	Description
.GHA	Host address.
.GHNS	Authoritative name server.
.GHMD	Mail destination.
.GHMF	Mail forwarder.
.GHCNA	Canonical name.
.GHSOA	Start of zone of authority.
.GHMB	Mailbox domain name.
.GHMG	Mailbox group member.
.GHMR	Mail rename.
.GHNUL	Null resource record.
.GHWKS	Well known service description.
.GHPTR	Inverse Address.
.GHHIN	Host information.

.GHMIN Mail information.

.GHZON Zone transfer.

.GHWMB Wild mailbox (e.g., .GHMB, .GHMG, .GHMR).

.GHWMA Wild mail agent (e.g., .GHMD, .GHMF).

.GHSTR Wild, i.e., * .

Application supplied arguments:

AC2: Source designator. The "question" for standard inquires.

AC3: Query operation, type, class, and length fields.

Field	Description
GH%OPR	Query operation, .GHTSD.
GH%TYP	Query type; one of the type codes listed above.
GH%CLA	Query class; one of the protocol suite codes listed above.
GH%LEN	Maximum number of octets available in the destination area; zero implies no limit.

AC4: Destination descriptor.

Returned data:

AC2: Updated source descriptor.

AC3: GH%LEN updated to number of bytes returned.

AC4: Updated destination descriptor.

The data returned consists of zero or more strings of octets. Each string has the format:

<type><class><ttl><data length><data>

Item	Description
<type>	The type of resource; 16 bits. One of the possible GH%TYP values listed above.
<class>	The protocol suite or class; 16 bits. One of the possible GH%CLA values listed above.
<ttl>	The resource's time to live, in seconds; 32 bits. The validity of the resource expires after the time interval has passed.
<data length>	The number of octets of data which follows; 16 bits.
<data>	Data describing the resource; the data format is a function of the class and type fields.

GTHST% Error Mnemonics:

ARGX02 Invalid function.

ARGX17 Invalid argument block length.

GTHSX1 Unknown host address.

GTHSX2 No number for that host name.

GTHSX3 No string for that host address.

GTHSX4 Invalid network number.

GTHSX5 Route not known - would try gateway.

GTJIX1 Invalid host index.

GTDX1 Invalid domain specification.

GTDX2 Referenced domain name does not exist.

GTDX3 Requested domain data not present at name.

GTDX4 Requested domain data not available.

GTDX5 Bad domain output specification.

GTDX6 Domain system error.

6. BBN TCP JSYS Interface

The BBN TCP interface was used prior to DEC's development of the TCP: device interface. The BBN interface will probably be phased out when the functionality of the DEC interface is complete. Descriptive information about TCP was presented with the TCP: device JSYS interface above and will not be repeated here.

The BBN interface has 9 TCP related JSYSi. They are used to:

OPEN%, Specify the ends of a connection and synchronize, a connection.
SEND%, Send data to the foreign end.
RECV%, Receive data from the foreign end.
CLOSE%, Begin closing a connection.
ABORT%, Break a connection.
STAT%, Obtain status information about a connection or TCP.
CHANL%, Specify interrupt channels.
SCSLV%, Specify security levels.
ATNVT%, Translate a connection into an NVT descriptor.

6.1. Creating a TCP Connection

A TCP connection is created by passing the address of a Connection Descriptor block (which identifies the two ends of the connection), a Transmission Timeout, and a Retransmission Parameters word to the OPEN% JSYS. A successful OPEN% will return a Job Connection Number or "JCN". The JCN is a small job-relative number which is given to the other BBN interface JSYSi to identify the connection. (The use of a JCN with the BBN JSYSi is analogous to the use of JFNs with the file system JSYSi.)

Each connection must be unique. This is guaranteed by the selection of a 16-bit local port value. Unfortunately, there is no easy way to find which port values are unused except by trial and error. Use of local port values less than 256 is restricted as they have been reserved for "well known" services. A suggested local port value contains the user's job number in the left-most 8 bits and anything else in the right-most 8 bits. If the connection is not unique, OPEN% will return a Connection Already Exists error and another local port value should be selected and tried.

Active connections are specified using the TCP%FS flag to the OPEN% JSYS; if it is not set, the connection will be passive.

The OPEN% JSYS has both blocking and non-blocking forms which are differentiated by the setting of the TCP%WT flag. The blocking form (TCP%WT flag set) will not return control to the application process until either the connection has been successfully synchronized or an error has occurred. The non-blocking form (TCP%WT flag not set) will return a success indication immediately (unless there was a syntax error or the Connection Already Exists). The application can subsequently check whether the connection has opened correctly by attempting to send or receive data or requesting the connection status (STAT% JSYS). A program interrupt may also be enabled for the connection opening (or closing) event.

6.2. Connection Descriptor Block

Information describing a connection is passed to TCP in a Connection Descriptor block via the OPEN% JSYS. The block contains .TCPCS words. Unspecified words and bits should be zero.

Word Offset	Field	Meaning
.TCPLH	B4-35	The local host's internet address (a host may have more than one). If the default local host address is to be used, specify zero.
.TCPLP	B20-35	A unique non-zero local port number. It is recommended the left-most 8 bits contain the job number to avoid conflicts with other connections. The GJINF% JSYS can be used to determine the job number.
.TCPFH	B4-35	Either a non-zero Foreign Host internet address, or, in a passive open, zero to allow a connection from any foreign host.
.TCPFP	B20-35	Either a non-zero Foreign Port number, or, in a passive open, zero to allow a connection from any foreign port.

.TCPOP B0-17 The address of an IP Option block, or zero.
B18-35 The address of a TCP Option block, or zero.
An option block consists of 10 words containing the packed options to be used. See sections 4.13, IP Options, and 2.2, TCP Options, for the formats of the individual options.

.TCPIP IP datagram parameters:
B0-1 Do-not-Fragment this datagram. If B1 is set IP is not allowed to fragment a datagram. Packets which are too large for an intermediate network will be discarded, not delivered (in fragments).

B10-17 Maximum Time-to-Live, in seconds. Datagrams which have not been delivered in the specified number of seconds will be discarded. The default value of 60 seconds is used if the field is zero.

B26 Set the IP Type-of-Service field from B27-35. (Note: a non-zero type of service will be loaded from B27-35 even if this bit is not set for compatibility with prior versions.)

B27-35 IP Type-of-Service field. See section 4.1, IP Type-of-Service.

.TCPIC B0-35 If TCP%IC is set in AC1 of the OPEN% JSYS, this word contains the initial Interrupt Channel word. See section 6.17, CHANL% JSYS, for the format of the Interrupt Channel Word.

6.3. Sending and Receiving Data

Data is transferred between the application and TCP in Data Buffers. A Buffer Header containing control information is associated with each Data Buffer. Data to be sent to the foreign end is placed into one or more Data Buffers, the associated Buffer Headers are filled in, and the address of each Buffer Header is passed to TCP using the SEND% JSYS.

The TCP collects data from the Data Buffers and decides, based on the current Window, etc., when to create a packet and transmit it to the foreign end. The application can force TCP to transmit the contents of a Data Buffer by setting the PUSH flag, TCP%PU, in the Buffer Header flag word before doing the SEND%.

When TCP has finished extracting the data from a SEND% Data Buffer it sets the DONE flag, TCP%DN, in the Buffer Header flag word, and optionally generates an interrupt. It is also possible to cause a process to block until a buffer has been processed by setting the TCP%WT flag in AC1 of the SEND% JSYS; the TCP%PU flag should also be set in the Buffer Header. (Beware of setting TCP%WT without TCP%PU -- the process may be blocked forever.) The SEND% will not return control to the application process until the data has been placed into a packet and, due to the presence of TCP%PU, sent (or an error occurs).

Data is received from the foreign end by first passing TCP the address(es) of a Buffer Header for an empty Data Buffer using the RECV% JSYS. When data arrives, TCP places it into the Data Buffers. The TCP%WT flag may be specified in AC1 of the RECV% JSYS, if control should not be returned to the application process until the Data Buffer has been filled and the associated Buffer Header updated (or an error occurs).

A buffer passed to TCP is returned to the application when it is full or when a packet arrives with the PUSH flag set. Before returning a buffer, TCP sets the TCP%DN flag and places an error code in the Buffer Header. The TCP%PU flag will be set if the received packet had the PUSH flag set. (The buffer might not be full in this case.)

Several buffers may be passed to TCP using SEND% and/or RECV% if the non-blocking form of the calls is used (TCP%WT was not set in AC1). The SEND% or RECV% will block if an attempt is made to queue more than a few (site dependent) buffers.

TCP also provides an asynchronous signal mechanism called URGENT. If a SEND% Buffer Header has the TCP%UR flag set, an urgent signal is associated with the last data octet in the buffer. Its sequence number is communicated to the receiving TCP. The receiving application will find the TCP%UR flag set in all RECV% Buffer Headers whose associated Data Buffer contains lower sequence numbered octets. An interrupt may be enabled for the "Urgent data has arrived" event. If the sender signals another URGENT, the new sequence number is used. The receiver may only get one URGENT interrupt if previous URGENT data has not yet been received (i.e., data is being processed slowly or the system is heavily loaded).

6.4. Buffer Headers

Data is transferred between the application process and TCP in Data Buffers. Each Data Buffer contains zero or more octets of data, packed four octets per word, in bits 0-31. Data is transferred between TCPs beginning with the left-most bit of the left-most octet. There is no padding between individual buffers.

Associated with each Data Buffer (but not necessarily contiguous to it) is a Buffer Header containing control information describing the Data Buffer. A Buffer Header is .TCPBS words long. The extended address of the Buffer Header is passed to TCP in AC2 by the SEND% and RECV% JSYSi. Note that individual Buffer Headers and Data Buffers may not cross section boundaries, but a Buffer Header and its associated Data Buffer need not be in the same section.

The format of a Buffer Header is:

Word Offset	Field	Meaning
.TCPBF	B0-17	Flags (note that the following flags have full-word values, not half-word).
	TCP%ER	This buffer has an error associated with it.
	TCP%LE	The error occurred at the local end, not the foreign end.
	TCP%PE	The error is permanent, not temporary.
	TCP%EC	Field containing the BBN error code. See section 6.8, Error Returns and Codes, for a listing of BBN error codes.
	TCP%DN	Asynchronous flag, cleared by TCP when it receives the buffer, set by TCP when processing associated with the buffer; has been completed.
	TCP%UR	SEND%: The last octet in the buffer should be communicated to the receiver as Urgent Data.

RECV%: There is additional Urgent Data.

TCP%PU SEND%: Promptly send (any previously buffered data and) data in this buffer to the receiving application (do not wait for more data to fill a packet).
RECV%: Sender requested data in this buffer be delivered promptly, even if the buffer is not full.

TCP%WM The Data Buffer is formatted as 36-bit bytes. Off if the Data buffer has four left-justified 8-bit bytes per word. Note: 36-bit mode is not yet implemented.

B18-35 Reserved for application process; not used by TCP.

.TCPBA B0-5 Byte position of the first data octet in the Data Buffer, a la an LDB pointer.

B6-11 Byte size; currently restricted to 8, or zero, which implies 8. If both the Position and Size fields are zero, a POINT 8,<extended address> pointer is used; if word mode (TCP%WM) is set, POINT 36,<extended address> is used.

B12 Must be zero.

B13-35 Extended address of first data octet.

Each word of the Data Buffer (except possibly the first and last) contains four left-justified 8-bit data octets. Usually, data is transmitted beginning with the left-most bit of the first word, but the position and size fields may be specified.

.TCPBC B0-35 Data Buffer count.

SEND%: The number of octets of data in the Data Buffer. When TCP has finished with the buffer, the field is updated to the number of octets not sent (usually 0).

RECV%: The number octets available for data. When TCP has finished with the buffer, this field is updated to the number of unused data octets.

.TCPBO B0-17 The address of an IP Option block, or zero.
B18-35 The address of a TCP Option block, or zero.

SEND%: If the address is not zero, merge the options from the Option block with those which received from the foreign end for inclusion in future packets.

RECV%: If the address is not zero, return any options from the most recently received packet in the Option block. Note that both option blocks are in the same section as the Buffer Header.

.TCPBI Specifies IP parameters (only used with SEND%).
B0 The (zero) field B1 should be used for the IP Don't Fragment flag.
B1 IP Don't fragment flag, if non-zero or B0 is set.
B10-17 IP Time to Live, if non-zero.
B27 The (zero) field B28-35 should be used for the IP Type of Service.
B28-35 IP Type of Service, if non-zero or B27 is set.

Any unspecified bits in the Buffer Header should be zero.

6.5. Closing a Connection

When an application has no more data to be sent, the connection should be "closed". The CLOSE% JSYS tells to the foreign end of the connection that no more data will be sent to it. Since CLOSE% implies a PUSH, any data which TCP has internally buffered will first be sent; then a FIN will be sent. The FIN notifies the foreign application that the connection is "Closing". After the CLOSE%, no more SEND%s are allowed.

CLOSE%, by itself, does not fully "close" the full-duplex connection, since there may be data from the other end to process (and the other end may even have more data to be sent).

There are two recommended procedures for closing a connection. If the application is notified (either by a RECV% error code or an interrupt) that the connection is "Closing", no more RECV%s are necessary. Any data for the foreign end should be sent using SEND%, and then a CLOSE% with the TCP%WT flag set should be issued. The CLOSE% will block until the connection is fully "Closed". If CLOSE% returns without error, it has also released the JCN. If an error is returned, the JCN has not been released; it should be released using the ABORT% JSYS.

If a CLOSE% is issued before being notified that the connection is "Closing", the TCP%WT flag should not be set. (Setting the TCP%WT flag in this case is a frequent cause of "connections failing to close".) Control will be returned to the application process to allow RECV%s to be issued until the foreign end (CLOSE%s and consequently) sends a FIN, which will cause a "Closed" error to be returned. (Failure to RECV% this data (or "unexpected data") is another common cause of "connections failing to close"; see section 6.11, the TCP%FR OPEN% bit.) The JCN should then be released using the ABORT% JSYS.

A connection may be abnormally terminated at any time by issuing the ABORT% JSYS. ABORT% also releases the JCN associated with the connection.

After a connection has been fully CLOSE%d or ABORT%ed, the internal connection block (TCB) is retained by TCP for several seconds to allow any packets, which might still be in the network, (e.g., retransmissions) to arrive and be processed. During this period, an attempt to reuse the same connection (host addresses and port numbers) will fail.

6.6. Connection Status

The status of a connection (or the values of TCP's internal counters) may be examined using the STAT% JSYS (or function .IPRIP of IPOPR% or function .TCRTC of TCOPR%). The most frequent use of STAT% is to determine the addresses/ports associated with a connection, the state of its send and receive sides, or an error code. Use of the symbolic name form of STAT% call is recommended to reduce site and software release dependencies. The following table lists the names of a few of the variables. The .IPRIP function of IPOPR% and the .TCRTC function of TCOPR% list other available quantities. See section 6.16, STAT% JSYS, for details.

ASCII Name	Contents
TFH	Foreign Host address.
TLH	Local Host address.
TFP	Foreign Port.
TLP	Local Port.
TRSYN	Receive Side State (see below).
TSSYN	Send Side State (see below).
TERR	BBN error code.
TCERR	TOPS-20 error code.

Symbol	Value	Meaning
.TCSYA	4	Waiting for SYN packet to arrive or be sent.
.TCSYS	5	SYN packet received or sent.
.TCSYN	7	Connection synchronized, data may be sent.
.TCFIN	2	FIN packet received or sent.
.TCTIM	1	Waiting to guarantee that the final ACK is delivered.
.TCNOT	0	FIN acknowledged, connection closed.

6.7. Interrupts

TCP allows the application to specify software interrupt channels for five TCP related events:

- State change,
- SEND% buffer done,
- RECV% buffer done,
- presence of URGENT data, and
- error (or timeout).

They may each be assigned to no channel, assigned to separate channels, or share a common channel. An interrupt may be reassigned to another channel and deassigned at any time. Note that, in general NO interrupts will occur if the application has ABORT%ed the connection.

Since the TCP runs at a higher priority than most application processes, successive interrupts may occur more quickly then the application can process them. One can not expect, for example, the number of RECV% buffer full interrupts to be the same as the number of RECV%s issued; there may be less if two buffers were filled without running the application process.

The state change interrupt occurs when:

- the connection is opening (SYN received),
- open (synchronized, SYNs exchanged and acked),
- closing (FIN received), or
- closed (FINs exchanged and acked).

The error interrupt occurs when a RESET is received (Connection Rejected), when a transmission timeout has occurred (Transmission Timeout), or a "significant event" has occurred.

The urgent data interrupt occurs when a packet containing urgent data arrives and the connection is not already in Urgent Receive mode. If another packet arrives with additional urgent data, a second interrupt will NOT be given unless the application has already RECV%ed all of the urgent data from the first packet.

The Buffer Done interrupts occur when the TCP%DN flag is set in the Buffer Header.

6.8. Error Returns and Codes

The BBN TCP JSYSi are called with arguments in one or more of ACs 1 through 4. If the call is successful, the JSYS will skip an instruction when control is returned to the application process. The contents of ACs 1 through 4 may be changed depending on the particular JSYS involved. If the call was not successful, control is returned to the instruction in the application process following the JSYS. In this case AC1 contains a BBN error code identifying the cause of the problem. The error codes returned (with the exception of ATNVT%, the User Queue JSYSi, and the DEC TCP: JSYS interface) are not in the format of other TOPS-20 errors, but are in the form of the Buffer Header error field (a TOPS-20 style error code may be found by examining the TCERR field using the STAT% JSYS).

The error code consists of three flag bits and a 5-bit error code, right-justified in AC1. The AC1 bits and the corresponding Buffer Header symbols are listed in the table below.

Buffer Header	AC1 Bit	Meaning
TCP%ER	1B28	Set if an error was detected ("E"), zero otherwise ("X").
TCP%LE	1B29	Set if the error was detected locally ("L"), zero if the error was the due to the foreign TCP ("F").
TCP%PE	1B30	Set if the error is permanent ("P"), zero if the error is temporary ("T").
TCP%EC	37B35	BBN Error Code.

The Error flag indicates an error.

The Local flag indicates whether the problem was detected at the local or foreign end.

The Permanent flag indicates whether the problem is permanent or temporary. Temporary errors are usually associated with resource shortages; repeat the request later. Permanent errors should not be retried.

Some error codes are only associated with a BBN TCP JSYS, some may be returned only by a BBN TCP JSYS if the TCP%WT flag was set in AC1, and some are returned only in a Buffer Header.

The error flags and their meanings are:

Key	Meaning
j	Error can be returned in AC1 by a BBN TCP JSYS.
w	Error can be returned in AC1 by a BBN TCP JSYS with TCP%WT set.
b	Error code can be returned in a Buffer Header.

Codes	Meaning (TOPS-20 equivalent)
0 b	No Error. The buffer was processed without error. (TCPX25)
1 j	Argument Error in JSYS. An AC1 flag (TCP%xx) was set which is inappropriate for the JSYS; the TVT, connection index, or JCN is invalid. (TCPX25)
2 j	Invalid Options specified. The routing option does not parse or an option length exceeds the amount of option space available. (TCPX47)
3 j	Connection Not Open. Attempt to CLOSE% a connection which is not open. (TCPX30)
4 j	Temporarily Out of Resources. A new connection cannot be created because of a lack of JCNs for the job, free storage for the TCB, an available TVT, or the system limit on simultaneous connections would be exceeded. (TCPX20)
5 j	Wild foreign host/port only allowed if listening. (TCPXX3)
6 j	Connection Already Exists. Attempt to OPEN% a connection which has already been OPEN%ed. (TCPX19)
7 wb	Connection Rejected or Reset. A RESET was received for the connection. (TCPX31)
9 w	Transmission Timeout. (TCPX32)
12 wb	Connection Closed or Closing. Closed locally or remotely. Cannot (re)activate the connection because it

was not fully closed. Note that the code, without the "error" flag set, is returned in RECV% Buffer Headers after a FIN has been received (the foreign end is closing). (TCPX33)

13 j Wild local port is illegal. A zero local port is not allowed. (Note that this error can also be caused by improper control flags or an improperly specified local host address. (TCPXX8)

14 wb Connection Reset. A Transmission Timeout has occurred or the connection is being aborted. (TCPX25)

15 j Bad Buffer Argument. The Buffer Header count (.TCPBC) is negative, the Buffer Header or Data Buffer is not accessible or crosses section boundary, or the position or byte size (not 8) is wrong. (TCPX25)

16 j Insufficient resources to process data buffer. Either insufficient free storage or TCP%WT was specified and there are no free "wait flags". (TCPXX1)

17 j Invalid interrupt channel. A channel number is not 0-5, 23-35, 77 (ignore), or 76 (clear). (TCPX34)

20 j Bad STAT% offset. Offset to too large for TCB or Statistics area. (TCPX34)

21 j Bad STAT% count. Count is zero, positive, or forms an offset which is too large. (TCPX34)

22 j Bad STAT% name. An ASCII name is not in STAT%'s table. (TCPX34)

29 j Changing security levels is not allowed. (TCPX34)

30 j Use of TCP%VT not allowed. Only Job0 (internet fork) can create TVTs using TCP%VT. (Use the ATNVT% JSYS.) (TCPX25)

31 j TCP not available. TCP has not been initialized or has been turned off. (TCPX16)

6.9. Options

Options may be specified with OPEN% or SEND%. Options are specified using the .TCPOP word in the Connection Descriptor block and/or the .TCPBO word in the Buffer Header block. The left-half contains the address of a block for IP options. The right-half contains the address of a block for TCP options (the addresses must be in the same section as the Connection Descriptor Block or Buffer Header). Each block is .TCPOW words long and contains one or more option octets, packed four left-justified octets per word.

Once specified, the options are sent in each packet generated. If an Option block address is zero, the corresponding options remain unchanged. All the current options may be cancelled by specifying an address pointing to an Option block which begins with an End-of-Options octet (zero).

Options contained in packets which are received may be obtained by specifying non-zero address(es) in the .TCPBO word associated with a RECV%.

The formats of the IP and TCP options have been previously described.

6.10. TCP Retransmission Parameters

Both OPEN% and SEND% require a Transmission Timeout and Retransmission Parameters. The Transmission Timeout is the number of seconds for which retransmissions of unacknowledged octets should be attempted before a Transmission Timeout error is returned to the application process. A Transmission Timeout will abort the connection unless the TCP%RX bit was specified in AC1 of the OPEN% JSYS or interrupts are enabled on the Error Channel. Values permissible are 1 to 3600 seconds, or zero. A zero value means that there is no timeout and retransmissions should continue "forever" (until the other end sends a RESET or the program is ^C'd and the connection ABORT%ed).

If you are not interested in the details of retransmissions, use zero for the Retransmission Parameters. The standard retransmission algorithm, based on measured round trip delay, will be used. See section 2.1, TCP Retransmission Algorithms and Parameters.

The Retransmission Interval is the time between successive transmissions of a packet. The Retransmission Parameters word specifies which, of two algorithms, will be used to compute the Retransmission Interval. Both algorithms are based on an estimate of the round trip delay associated with the connection. The default algorithm is used if the left-half value of the Retransmission Parameters word is zero.

A second algorithm, which allows the application specify its factors, is activated if the left-half of the Retransmission Parameters word is non-zero. Once activated, the second algorithm is used for the duration of the connection. The right-half, if not zero, is the Initial Retransmission Interval, in seconds. If the right-half is zero, the Initial Retransmission Interval will be computed based on the estimated round trip delay.

The left-half of the Retransmission Parameters word contains two 9-bit quantities, called the Backoff Numerator (in B0-8) and the Backoff Denominator (in B9-17). In computing the Next Retransmission Interval for a packet, the last interval is multiplied by the Numerator and then divided by the Denominator (using integer arithmetic). The Backoff Numerator must be greater than or equal to the Backoff Denominator. Default values are used if a factor is zero.

6.11. OPEN% JSYS 742

Specifies a TCP connection and returns its JCN.

Accepts In AC1: Flags ! Address-of-Connection-Descriptor-Block.

TCP%ET Messages concerning significant events related to the connection are sent to the job's controlling terminal. (Not yet fully implemented.)

TCP%FR Discard any data which is received after a CLOSE%.

TCP%FS Set to initiate synchronization (active open). Cleared to await foreign connection attempt (passive open).

TCP%IC Set to indicate that word .TCPIC of the Connection Descriptor Block contains an initial Interrupt Channel Word.

TCP%PS If set and the connection attempt fails, wait a few seconds and restart the connection procedure. Note that this flag should not be used.

TCP%PT Events concerning this connection should be traced. Tracing only occurs if it has been enabled in the monitor.

TCP%RX Do not abort the connection when a retransmission timeout has occurred; give an interrupt on the error channel.

TCP%VT Create a TCP virtual terminal connection. Note that only Job 0 may use this flag. Application programs should use the ATNVT% JSYS.

TCP%WT Block until the connection has been synchronized or an error has occurred.

AC2: Transmission Timeout, in seconds (3600 second maximum).

AC3: Retransmission Parameters Word.

B0-8 Backoff Numerator.
B9-17 Backoff Denominator.
B18-35 Initial Retransmission Interval.

Returns +1: Failure; BBN error code in B28-35 of AC1. If a JCN had been assigned before the error was detected, it is in the left-half of AC1 (it should be ABORT%ed).

+2: Success; AC1 contains the JCN in the right-half with the TCP%JS flag set.

The format of the .TCPCS-word Connection Descriptor Block, whose address is contained in the right-half of AC1, was described in the Connection Descriptor Block section. The JCN returned in AC1 should be used to specify this connection to the other BBN TCP JSYSi.

If the TCP%WT flag is not set, control will be returned to the application process as soon as the connection parameters have been validated, the internal control block (TCB) created, and, if the TCP%FS flag was set, the connection synchronization process begun. The connection will probably not be fully synchronized. An interrupt may be enabled for connection state changes (which includes leaving the listening state and becoming fully synchronized), or the state of the internal TSOPN flag may be checked using the STAT% JSYS.

Note that SEND%s, RECV%s, and CLOSE% may be issued before a connection is fully synchronized.

OPEN% Error Codes:

ELP+1 Argument Error in JSYS. An inappropriate AC1 flag (TCP%xx) was set. The Connection Descriptor block is invalid.

ELP+2 Invalid Options specified. The routing option does not parse or an option length exceeds the amount of option space available.

ELT+4 Temporarily Out of Resources. A new connection cannot be created due to a lack of JCNs (for the job), free storage for the TCB, an available TVT, or because the system limit on simultaneous connections would be exceeded.

ELP+6 Connection Already Exists. Attempt to OPEN% a connection which has already been OPEN%ed.

EFP+7 Connection Rejected or Reset. A RESET was received for the connection.

ELP+9 Transmission Timeout.

ELP+12 Connection Closed or Closing. Closed locally or remotely; cannot (re)activate the connection because it was not fully closed.

ELP+13 Wild local port is illegal. A zero local port is not allowed.

ELP+14 Connection Reset. A Transmission Timeout has occurred or the connection is being aborted.

ELP+30 Use of TCP%VT not allowed. Only Job0 (internet fork) can create TVTs (Use ATNVT%).

ELT+31 TCP not available. TCP has not been initialized or has been turned off.

6.12. CLOSE% JSYS 743

Close a TCP connection.

Accepts In AC1: TCP%JS ! Flags ! JCN.

TCP%FR Discard any data which is received after a CLOSE%.

TCP%JS Must be set because the right-half has a JCN.

TCP%WT Wait for a close in both directions or for an error.

Returns +1: Failure, BBN error code in B28-35 of AC1.

+2: Success.

The CLOSE% JSYS causes any data in TCP's buffers to be sent to the receiver. A FIN is then sent to notify the receiver that no more data will be sent. After a CLOSE%, SEND%s are illegal.

Note that if the TCP%WT flag is not set, the JCN will remain valid, allowing more RECV%s, etc. RECV%s should be repeated until a Connection Closed error is returned. ABORT% must then be used to release the JCN.

If the TCP%WT flag is set, the CLOSE% JSYS will block until FINs have been exchanged and acknowledged to fully close the connection, or an error is detected. Note that a connection cannot be fully closed if there is data which has not been RECV%ed (unless the TCP%FR flag was specified in OPEN% to discard any data received after CLOSE%). If no error was detected, the JCN is also released; the JCN must be released using ABORT%.

CLOSE% Error Codes:

ELP+1 Argument Error in JSYS. An inappropriate AC1 flag (TCP%xx) was set. The JCN is invalid.

ELP+3 Connection Not Open. Attempt to CLOSE% an unopened connection.

ELT+4 Temporarily Out of Resources. A new connection cannot be created because of a lack of JCNs for the job, free storage for the TCB, an available TVT, or the system limit on simultaneous connections would be exceeded.

EFP+7 Connection Rejected or Reset. A RESET was received for the connection.

ELP+7 Connection does not exist. There is no TCB for the specified connection.

ELP+9 Transmission Timeout.

ELP+14 Connection Reset. A Transmission Timeout has occurred or the connection is being aborted.

6.13. SEND% JSYS 740

Send a buffer of data over a TCP connection.

Accepts In AC1: TCP%JS ! Flags ! JCN.

TCP%JS Must be set because the right-half has a JCN.

TCP%WT Wait for data to be copied into a packet, or for an error. Note that the packet may not be sent if the TCP%PU flag was not set in the Buffer Header.

AC2: Extended Buffer Header Address.

AC3: Transmission Timeout, in seconds (3600 second maximum).

AC4: Retransmission Parameters Word.

B0-8 Backoff Numerator.
B9-17 Backoff Denominator.
B18-35 Initial Retransmission Interval.

Returns +1: Failure, BBN error code in B28-35 of AC1.

+2: Success.

The format of the .TCPBS-word Buffer Header Block whose extended address is in AC2 was described in the Buffer Header section. Unused words and bits should be zero.

When TCP accepts the Buffer Header, the Error field and TCP%DN flag are cleared. Retransmissions occur as specified by the Retransmission Parameters word, until the data has been acknowledged by the receiver, the Transmission Timeout has passed, or an error has been detected. When the data has been processed, an error code is placed into the Error field, the count word is changed to the number of octets not sent (normally 0), and the TCP%DN flag is set, and an interrupt on the Send Buffer Done channel occurs. The buffer will remain in use until the TCP%DN flag is set.

If the TCP%WT flag is not specified, control returns to the application process.

SEND% Error Codes:

ELP+1 Argument Error in JSYS. An AC1 flag (TCP%xx) was set that is inappropriate for the JSYS; the JCN is invalid.

ELP+2 Invalid Options specified. The routing option does not parse, or an option length exceeds the amount of option space available.

EFP+7 Connection Rejected or Reset. A RESET was received for the connection.

ELP+7 Connection does not exist. There is no TCB for the specified connection.

ELP+9 Transmission Timeout.

ELP+12 Connection Closed or Closing. Closed locally or remotely; cannot (re)activate the connection because it was not fully closed.

ELP+14 Connection Reset. A Transmission Timeout occurred. The connection is being aborted.

ELP+15 Bad Buffer Argument. The Buffer Header count (.TCPBC) is negative, the Buffer Header or Data Buffer crosses a section boundary, or is not accessible.

ELT+16 Insufficient resources to process data buffer. Either insufficient free storage, or TCP%WT was specified and there are no free "wait flags".

ELT+31 TCP not available. TCP has not been initialized. TCP has been turned off.

6.14. RECV% JSYS 741

Receive a buffer of data from a TCP connection.

Accepts In AC1: TCP%JS ! Flags ! JCN.

TCP%JS Must be set because the right-half has a JCN.

TCP%WT Wait for data buffer to be filled. Wait for an error.

AC2: Extended Buffer Header Address

Returns +1: Failure, BBN error code in B28-35 of AC1.

+2: Success.

The format of the .TCPBS-word Buffer Header Block whose extended address is contained in AC2 is described in the Buffer Header section. Unused words and bits should be zero.

The count word in the Buffer Header should contain the (maximum) number of octets that the Data Buffer will hold. When TCP accepts the Buffer Header, the Error field, the TCP%UR flag, TCP%PU flag, and TCP%DN the flag are cleared. The buffer will remain in use until the TCP%DN flag is set. Data received is placed into the buffer until either it is full or a PUSH is received. When TCP is finished with the buffer, an error code will be placed into the Error field. The count word is set to the number of unused octets remaining in the buffer. If there is additional Urgent data, the TCP%UR flag is set. If the buffer is being returned because of a PUSH, the TCP%PU flag is set. Finally, the TCP%DN flag is set, and an interrupt occurs on the Receive Buffer Done channel.

If the TCP%WT flag is not specified, control returns immediately to the application process.

RECV% Error Codes:

ELP+1 Argument Error in JSYS. An inappropriate AC1 flag (TCP%xx) was set. The JCN is invalid.

EFP+7 Connection Rejected or Reset. A RESET for the connection was received.

ELP+7 Connection does not exist. There is no TCB for the specified connection.

ELP+9 Transmission Timeout.

ELP+12 Connection Closed or Closing. Closed locally or remotely; cannot (re)activate the connection because it was not fully closed. Note that this code, without the "error" flag set, is returned in RECV% Buffer Headers after a FIN has been received (the foreign end is closing).

XLP+12 Connection Closing. Closed remotely, no more data to be RECV%ed.

ELP+14 Connection Reset. A Transmission Timeout occurred or the connection is being aborted.

ELP+15 Bad Buffer Argument. The Buffer Header count (.TCPBC) is negative; the Buffer Header or Data Buffer crosses a section boundary or is not accessible.

ELT+16 Insufficient resources to process data buffer. Insufficient free storage or TCP%WT specified and there are no "wait flags" free.

ELT+31 TCP not available. TCP has not been initialized or has been turned off.

6.15. ABORT% JSYS 747

Abort a TCP connection.

Accepts In AC1: TCP%JS ! JCN.

TCP%JS Must be set because the right-half has a JCN.

Returns +1: Failure; BBN error code in B28-35 of AC1.

+2: Success, connection broken; JCN released.

The local end of the connection is forgotten; partially processed packets are deleted, queued SEND% and RECV% buffers are returned, and the retransmission queue is purged.

If FINs have not been exchanged and acknowledged, a RESET packet is sent to the remote end. If the RESET packet is not delivered, the other end discovers its half-open connection the next time it uses it.

Note that TCP's internal connection block (TCB) remains for several seconds to allow any packets in transit to arrive and be processed. An attempt to reuse the same connection (host addresses and ports) will fail until the waiting time has passed.

ABORT% Error Codes:

ELP+1 Argument Error in JSYS. An inappropriate AC1 flag (TCP%xx) was set. The JCN is invalid.

6.16. STAT% JSYS 745

Obtain information about a TCP connection or the system's internal network statistics tables.

Accepts In AC1: Flags ! (TVT-line number, connection-index, JCN, or 0).

TCP%IX Return information about the connection specified by the index in the right-half of AC1. The index is a system-wide identifier whose binding may change from call to call. (See TCP%NI.)

TCP%JS Return information about the connection whose JCN is in the right-half of AC1.

TCP%NI Return an AOBJN counter for the currently active connections in AC2. (See TCP%IX.)

TCP%NT Return an AOBJN counter for the currently active TVTs in AC2. (See TCP%TV.) (Note that the right-half of AC2 will contain the terminal line number of the first TVT.)

TCP%SD An LDB pointer should be returned for each of the specified symbols instead of the value. (The byte pointers returned assume AC14 (octal) is pointing to a TCB image, and AC13 (octal) is pointing at STAT0 (offset 0 of the statistics area)). Requires TCP%SY to be set and a valid connection identifier (e.g., TCP%IX!1).

TCP%SL AC4 points to an area where the length, in LDB bytes, of each symbol should be placed. Requires TCP%SY and TCP%SD to be set.

TCP%ST Return network statistics information instead of information about a TCP connection. The right-half of AC1 is ignored. The .IPRIP function of IPOPR%

and the .TCRTC function of TCOPR% list other available quantities. (See TCP%SY).

TCP%SY The right-half of AC2 contains the address of a list of ASCII variable names associated with either a TCP connection (TCB) or network statistic for which information should be returned. See TCP%SD and TCP%SL.

TCP%TV Return information about the TVT whose line number is specified in the right-half of AC1. (See TCP%NT.)

AC2: -N,,Offset into TCB or Statistics area (TCP%SY clear). N successive words are desired beginning at the specified offset.

or

-N,,Address of symbol names (TCP%SY SET). N symbols are given beginning at the specified address.

AC3: -M,,Address in application's space where information is returned; contains M words.

AC4: -M,,Address in application's space where lengths associated with the specified symbols are placed, if TCP%SL is set; contains M words.

Returns +1: Failure; BBN error code in B28-35 of AC1.

+2: Success; Data copied to application specified area: counts and addresses in AC2, AC3, and AC4 updated.

The TCB offset identifies the transfer starting point. The Address in application space identifies the starting point of the destination area. If successful, min(M,N) words have been transferred from the TCB (or network statistics area) to the application's space.

Note that the order/entries are site dependent; do not use numeric offsets. Use TCP%SD and the ASCII names.

STAT% Error Codes:

ELP+1 Argument Error in JSYS. An inappropriate AC1 flag (TCP%xx) was set. The TVT, connection index, or JCN is invalid.

ELP+20 Bad STAT% offset. Offset is to too large for TCB or Statistics area.

ELP+21 Bad STAT% count. Count is zero, positive, or forms an offset that is too large.

ELP+22 Bad STAT% name. An ASCII name is not in the STAT% table.

ELT+31 TCP not available. TCP has not been initialized or has been turned off.

6.17. CHANL% JSYS 746

Specify PSI channels for TCP interrupts.

Accepts In AC1: TCP%JS ! JCN.

TCP%JS Must be set because the right-half has a JCN.

AC2: Six channel numbers.

B0-5	Urgent data available.
B6-11	RECV% buffer done.
B12-17	SEND% buffer done.
B18-23	Error.
B24-29	State change (opening, open, closing, closed).
B30-35	Reserved, must be 77 (octal).

Returns +1: Failure; BBN error code in B28-35 of AC1.

+2: Success; This fork will receive TCP PSIs for the specified events. Note that this word may be specified in the Connection Descriptor block as an initial argument to the OPEN% JSYS (if the TCP%IC is set).

Each of the 6-bit bytes may be: a decimal channel number (0-5 or 23-35), 77 (octal) if no change is desired for the corresponding event, or 76 (octal) to disable interrupts for the event. PSIs may be dropped or may be tardy on heavily loaded systems. Defensive programming is required to guard against these problems.

CHANL% Error Codes:

ELP+1 Argument Error in JSYS. An inappropriate AC1 flag (TCP%xx) was set. The JCN is invalid.

ELP+17 Invalid interrupt channel. A channel number is not 0-5, 23-35, octal 77, or 76.

ELT+31 TCP not available. TCP has not been initialized or has been turned off.

6.18. SCSLV% JSYS 744

OBSOLETE: Set the security level for a TCP connection. Use the appropriate IP security option; see section 4.13, IP Options.

Accepts In AC1: TCP%JS ! JCN.

TCP%JS Must be set because the right-half has a JCN.

AC2: 36-bit security value

Returns +1: Failure; BBN error code in B28-35 of AC1.

+2: Success; The security value has been associated with the connection.

The security value is not interpreted by the TCP (except to see that it matches). The number of bits of the security value used is variable and depends on the security procedure. In all cases, the right-most bits of the word are used.

SCSLV% Error Codes:

ELP+1 Argument Error in JSYS. An AC1 inappropriate flag (TCP%xx) was set The JCN is invalid.

ELP+29 Changing security levels is not allowed.

ELT+31 TCP not available. TCP has not been initialized or has been turned off.

References

[1] Kevin W. Paetzold, "TCPIP-SPEC.DOC.4", Digital Equipment Corporation, Marlboro, Massachusetts, September 13,1984.

[2] "TOPS-20 Monitor Calls Reference Manual", AA-4166D-TM, Digital Equipment Corporation, Marlboro, Massachusetts, 1980.

[3] Bolt Beranek and Newman Inc., "Specifications for Interconnection of a Host and an IMP", Technical Report 1822, Bolt Beranek and Newman, 1978.

[4] J. Postel, Information Sciences Institute, University of Southern California, "Internet Protocol", RFC 791, Network Information Center, SRI International, March 1982.

[5] J. Postel, Information Sciences Institute, University of Southern California, "Internet Control Message Protocol", RFC 792, Network Information Center, SRI International, March 1982.

[6] J. Postel, Information Sciences Institute, University of Southern California, "Transmission Control Protocol", RFC 793, Network Information Center, SRI International, March 1982.

[7] J. Postel, Information Sciences Institute, University of Southern California, "Telnet Protocol Specification", RFC 854, Network Information Center, SRI International, June 1983.

[8] J. Postel, Information Sciences Institute, University of Southern California, "File Transfer Protocol", RFC 765, Network Information Center, SRI International, March 1982.

[9] J. Postel, Information Sciences Institute, University of Southern California, "Simple Mail Transfer Protocol", RFC 821, Network Information Center, SRI International, November 1982.

[10] D. L. Mills, Linkabit, "Internet Delay Experiments", RFC 889, Network Information Center, SRI International, December 1983.

[11] J. Postel, Information Sciences Institute, University of Southern California, "Assigned Numbers", RFC 900, Network Information Center, SRI International, June 1984.

[12] J. Postel, Information Sciences Institute, University of Southern California, "Telnet Option Specifications", RFC 855, Network Information Center, SRI International, May 1983.

Note: RFC 856 to RFC 861, describing individual options, were published concurrently with RFC 855.

[13] J. Postel, Information Sciences Institute, University of Southern California, "User Datagram Protocol", RFC 768, Network Information Center, SRI International, March 1982.

[14] J. Haverty, Bolt Beranek and Newman, "XNET Formats for Internet Protocol Version 4", IEN 158, Network Information Center, SRI International, October 1980.

[15] P. Mockapetris, Information Sciences Institute, University of Southern California, "Domain Names - Concepts and Facilities", RFC 882, Network Information Center, SRI International, November 1983.

[16] P. Mockapetris, Information Sciences Institute, University of Southern California, "Domain Names - Implementation and Specification", RFC 883, Network Information Center, SRI International, November 1983.

Index of Network Application Writer's Guide

7. Configuring the Networks

7.1. Files in SYSTEM:

Each site must supply files specifying the local network configuration and any changes to network parameters that may be desirable. Once the networks are configured and running, the only maintenance needed is to keep the (permanent) host and local host address tables up to date. These tables are built from files in SYSTEM: and are processed at system startup. The files required in SYSTEM: are described in subsequent paragraphs.

The SYSTEM:INTERNET.GATEWAYS and SYSTEM:HOSTS.TXT files for the DOD Internet are available from the ARPANet Network Information Center (SRI-NIC.ARPA). Sites installing ARPANet with TCP/IP support should be sure to obtain the most recent versions of these files.

7.1.1. SYSTEM:INTERNET-ETHERNET-MAPPINGS.BIN

This file contains information about translations between Internet addresses and Ethernet hardware addresses.

7.1.2. SYSTEM:INTERNET-LOGIN-MESSAGE.TXT

This optional text file contains a special login message sent whenever a TCP connection is established to the TELNET port.

7.1.3. SYSTEM:SITE-ADDRESS.TXT (or INTERNET.ADDRESS)

The SYSTEM:SITE-ADDRESS.TXT file (which DEC has called INTERNET.ADDRESS) contains information about the addresses and interfaces of the customer's system. It replaces the "HOST" keyword in the X-CONFIG.CMD file read by SYSJOB, and must be updated with the address(es) of the customer's host. The file contains a line entry for each network interface present in the system configuration. It may contain additional line entries if a single physical interface supports multiple protocol suites or addresses. Additional entries may be specified to restrict forwarding of packets between multiple interfaces, and to alter configuration parameters.

7.1.3.1. Interface Lines

Lines used to specify network interfaces are composed of several fields separated by commas. The first field is a keyword that specifies the type of hardware interface used to connect to a network.

The keyword may be followed by a list of octal codes, separated by colons. The device initialization routine interprets the codes. The most common interface is an AN20, which has both input and output device codes. The first field for the default AN20 is:

```
AN20:520:524
```

Other hardware devices, whose names are recognized, but not fully supported are: ALTO, IMP10, IPNI, and NETWORK-DTE. If an interface supports multiple network addresses or protocol suites, additional lines beginning with the keyword VIRTUAL follow the line specifying the common hardware interface.

The second field specifies the protocol used to communicate between the host and the network (front-end); keywords identifying the types of networks are contained in the TYPNAM table in STG. Note that some networks may not require any protocol at this level. The protocol used by the ARPANET is 1822. Other recognized keywords are: CHAOS, ETHERNET, and NETWORK-FE.

The third field specifies the (first) network address associated with the interface. In the case of the DARPA Internet, the INTERNET keyword is followed by a colon and the

host's internet address, expressed in the "dotted four byte" format, e.g., INTERNET:[10.0.0.79]. Other recognized address keywords are: DECNET, PUP, and XNS.

Thus, the minimal specification for an 1822 ARPANET AN20 interface with address 10.0.0.79 is:

```
AN20:520:524,1822,INTERNET:[10.0.0.79]
```

Other required parameters would be specified by the device's initialization code.

Optional fields, which may be present, are:

OUTPUT-RATE:n An estimated transmission rate into the network, in microseconds per byte; specified as a decimal number. It is used for flow control within the host.

PACKET-SIZE:n The maximum packet length supported by the network, in octets (exclusive of any local network leader); specified by a decimal number.

PRIORITY:n Systems with multiple network addresses may wish to specify relative priorities using the PRIORITY keyword. The highest priority is 63, the lowest 0.

LOGICAL-HOST-MASK:[a.b.c.d]
Some protocols allocate a portion of the network address space to support multiplexing of a single network interface. This field indicates those bits that should be excluded from routing and address comparisons. The mask has the same format as the host address.

MONITOR:NCT<i>A[:NCT<i>C]
Sites wishing to monitor network utilization and performance may specify this field. The keyword is followed by a name of the form NCT<i>A , where <i> is a digit, beginning with 1. If the line entry specifies a hardware interface (i.e., not a VIRTUAL line), additional information will be

collected if a second argument, NCT<i>C, is specified. The maximum value of i is determined by the number of interfaces in the system configuration (%NETS in PARAN.MAC or PARAMS.MAC).

SET:<flag-name> There may be one or more SET fields in a line entry. The parameters currently supported are:

RFNM-COUNTING
For 1822 networks, setting this parameter causes the host to count outstanding messages on each connection. The host will not attempt to send a message to a destination if doing so would cause the IMP to block all outgoing traffic, but will look for another message that can be sent (to some other destination). Unless there is a good reason not to set the parameter, it is recommended that it be set (a very bad link from the AN20 to the IMP is one example -- the poor link could corrupt enough packets to cause resulting in poor performance).

HANDLING-TYPE
For 1822 networks, setting this parameter enables use of multiple 1822 connections from the host. It might profitably be used in situations where a host has so much traffic to another system that the 1822 limit of eight outstanding messages on an 1822 connection causes poor performance, i.e., a lot of traffic going to a single gateway or terminal concentrator. Its use in other situations is not recommended.

TYPE-3-MESSAGES
For 1822 networks, setting this parameter enables use of "uncontrolled" 1822 messages. It should not be set unless the IMP has been configured to allow such communication.

ID:<name-string>
A name (text string) may be associated with an interface for use in an UNRESTRICTED-FORWARDING command described below.

7.1.3.2. Command Lines

In addition to lines associated with interfaces and network addresses, command lines may be present to alter system network parameters. There are currently two recognized commands: SET and UNRESTRICTED-FORWARDING.

SET:<keyword>:<value>
SET:<network parameter>[+<decimal byte number>]:<value>

This command modifies system control flags and parameters. Value may be:

- a decimal number,
- "#" followed by an octal number, or
- "&" followed by an internet address in the form "a.b.c.d".

The various network parameters may be specified using the parameter (or parameter+offset) form. Consult the definition of DEFSTS in ANAUNV.MAC for a complete list of parameters and their standard values. Note that the parameters are not all independent, so changing one may require changing others.

The currently recognized keywords (and their default values) are:

Keyword (Default)	Description
BBNOK (1)	When 1, use of the old BBN network JSYSi is allowed.
DECOK (1)	When 1, use of the DEC network JSYSi is allowed.
DMS%ED (1)	Enables use of the Domain Database by the GTHST% JSYS.
DMS%EC (0)	Enables use of locally cached data by GTHST% (caching of data is being implemented).

DMS%ER (0) Allow GTHST% to request that the Resolver program consult foreign Name Servers to answer a request which is not in the local database (the resolver program is under development).

DMS%ES (1) Enable a local Name Server program to answer queries from remote Resolvers.

IPS%FR (0) When 1, only one fragment of a fragmented IP packet is placed into the output queue at a time. When clear, all fragments are queued "at the same time". The inter-packet spacing resulting from setting the flag may help in cases where receivers cannot handle back-to-back packets, or there are "short queues" within the network.

IPS%LP (0) When 1, the Internet fork loops until there is nothing more for it to do. Setting it might improve interactive network performance at the expense of running user processes.

LOCAL-LOGICAL-HOST (0)
Setting this variable to -1 allows the host to use Logical Host addresses. It has no use if none of the network(s) support logical addressing.

MNS%MP (0) When 1, remote monitoring uses a private protocol.

MNS%MT (1) When 1, remote monitoring uses TCP.

MNS%RF (0) When 1, restricted forwarding is in effect. Set by the UNRESTRICTED-FORWARDING command.

TCS%LA (0) When 1, TCP's minimum retransmission interval is the integer load average plus 5 (seconds).

TCS%LP (0) When 1, the TCP fork loops until there is nothing more for it to do. Setting it might improve interactive network performance at the expense of running user processes.

TCS%O1 (0) When 1, additional flow control information (option) is used with TCP connections. Note that some systems (e.g., TACs) cannot handle "unknown" options, so setting this flag may prevent communication with them.

TCS%O2 (0) When 1, TCP connections are hashed using sequence numbers instead of local port (not yet implemented).

TCS%RX (1) When 1, information from retransmitted packets is used in TCP round-trip-time computations. Due to the protocol, such information is "unreliable" and may cause bad estimates (low throughput).

Network parameters associated with the Domain Name functions are specified in the DOMPAR block.

DOMPAR Offset (Default)	Description
0.	Contains the DMS%xx flags described above.
1. (10)	Maximum depth of nicknames; used to protect against naming loops.
2. (2000)	Millisecond check interval for acquiring a database lock.
3. (500)	Milliseconds to wait before GH%INI returns a not locally available error.
4. (200)	Millisecond check interval for acquiring the master database lock.
5. (30000)	Maximum number of milliseconds to wait for resolution of a query by the Resolver.
6. (0)	Default class, as used by GTHST% .GTHSN and .GTHNS. (Zero defaults to Internet.)
7.-13. (0)	Resource record format domain name string to be added to names which do not end with a dot (primitive form of "completion").

Network parameters associated with the internal Telnet Server that supports network logins are contained in the TVTPR block.

TVTPR Offset (Default)	Description
0. (30)	Telnet synchronization timeout, in seconds. The connection synchronization process must complete in 30 seconds or the connection attempt will be aborted.
1. (900)	Transmission timeout, in seconds, used after connection has been established. If data has not been acknowledged within this 15 minute interval the connection is aborted.
2. (500)	Milliseconds per character allowed for system messages to be sent before the message is flushed.
3. (30000)	Minimum number of milliseconds to wait for a negotiation to be answered. The maximum is twice this number.
4. (60000)	Millisecond interval of background checks to make sure that the listener isn't hung.
5. (60000)	Millisecond interval used to probe closed send windows.
6. (0)	Retransmission Parameters Word for Telnet connections.
7. (0)	Special flags specified when the connection is opened. See section 6.11, OPEN% JSYS; note only TCP%FR and TCP%RX might be useful.
8.-14. (0,23,0,0,0,0,0)	Connection Descriptor Block used with the OPEN% JSYS to listen for network login requests. The Telnet port is 23; if a local port of zero is specified, the internal Telnet Server is disabled. Specification of local host or foreign host fields may be used to limit which connections will be accepted; an Access Control

Job or network server program may be used if more complex criteria are to be used.

UNRESTRICTED-FORWARDING (No forwarding permitted)
UNRESTRICTED-FORWARDING:<ID1>,<ID2>[:<ID3>,<ID4>]

If the command is not specified, packets are forwarded between any of a host's interfaces. It may thus act as a simple routing gateway between networks. (An extreme example might be for a (non-routing) work station to send all packets to the TOPS-20, which would then forward/route them to the correct destination.)

The UNRESTRICTED-FORWARDING command should be specified if forwarding should not be permitted. Given alone, no forwarding will be allowed.

If it is desirable to allow forwarding between some set of the interfaces, fields consisting of pairs of IDs may be specified (see the ID keyword in the the interface line entries). Pairs are separated by colons; IDs within a pair are separated by a comma. The pair ID2,ID2 means that packets arriving on interface ID1 may be forwarded via interface ID2. Note another field is necessary to permit forwarding from ID2 to ID1.

7.1.3.3. Example of a SITE-ADDRESS.TXT File

The following is an example SYSTEM:SITE-ADDRESS.TXT file. Each site should change the address (following the "IN:") to correspond to the internet address for the site. The address of DEC-TOPS20 is 10.0.0.79. [Note that while two lines are shown for clarity in the AN20 command, all information for an interface must be on a single line.]

```
;;; SITE-ADDRESS file for host DEC-TOPS20

AN20:520:524,1822,IN:[10.0.0.79],1822,PACKET-SIZE:1004,SE (cont.)
T:RFNM-COUNTING,LOGICAL-HOST-MASK:[0.0.255.0],MONITOR:NCT1A:NCT1C

SET:LOCAL-LOGICAL-HOST:-1
```

7.1.4. SYSTEM:INTERNET.GATEWAYS

The SYSTEM:INTERNET.GATEWAYS file specifies the addresses of (some) of the Internet gateways -- those to be tried if no better route has been found (via an ICMP Redirect), or in cases where a particular gateway is to be used (instead of one which might be specified via an ICMP Redirect). In the usual case, the file should contain entries for a couple of routing (PRIME) gateways on each of the networks to which the host is connected.

Each line entry in the file begins with an identifying keyword and specifies a single gateway. Historically, there were four keywords:

- PRIME — These gateways perform Internet routing, and communicate routing information between themselves to establish the "shorest" route.
- DUMB — These gateways can route packets, but do not participate in routing updates; they have been outlawed.
- ALWAYS-UP — These gateways may be used to indicate the only path to a stub network; alternate routes are not tried.
- HOST — A host that will forward packets, but shouldn't be used if another route exists.

The keyword is followed by a list of internet addresses (separated by commas). One address in the list should be on a network to which the host is connected (if none are, the gateway entry is ignored). If that is the only address, the gateway may be tried when the routing cache does not contain a route to a particular network (round-robin among all such gateways). If there are additional addresses (and the gateway is up), this gateway will be used for traffic destined to the networks identified by the additional addresses. (Note that it is not necessary for the gateway to actually have physical connections to the networks identified by the additional addresses.)

The following is an example of a SYSTEM:INTERNET.GATEWAYS file.

```
CREATED 2 JAN 1985 800-EST
; GW/PRIME Internet Gateways.
;  Routing gateway on ARPANET
PRIME 10.5.0.5

;  Routing gateway on MILNET
PRIME 26.0.0.103

;  Administratively desired gateway for traffic between ARPANET
;  and 192.1.7 (or 128.23) etc.
PRIME 10.6.0.5, 192.1.7.2, 128.23.0.11
PRIME 10.5.0.51, 128.18.1.0

; GW/ALWAYS-UP gateways. Only path to stub network.
ALWAYS-UP 10.3.0.89, 192.5.43.1

; Host IP/GW gateways.  Use only as a last resort.
HOST 10.3.0.5, 192.1.2.68
```

7.2. Domain Name Services

Domain name support requires the presence of files containing the main and backup tables. Resolving names not in the local tables requires that the resolver program be running; it runs as a user job and should be started in the SYSJOB.RUN file. If the host is a name server the server program must also be run. See the description of the DOMPAR parameter block under the SET: command in section 7.1.3.2.

See RFC 882 [15] and 883 [16] for a complete description of the domain concept and procedures required to specify domains, and how to include portions of the domain space in the local databases.

7.2.1. Files in DOMAIN:

The domain software uses the logical name DOMAIN: to find the databases, definition files, and programs. A definition for it must be added to the x-x-CONFIG.CMD file.

7.2.1.1. DOMAIN:FLIP.DD and FLOP.DD

The file containing the master database is called FLIP.DD; the backup copy is called FLOP.DD. Both copies must have identical generation numbers. DOMAIN: is searched for the highest generation of FLIP.DD and an identical generation of FLOP.DD. If a pair is found and at least one has not been corrupted (due to a crash while being updated), they are used. If neither file is suitable, the next lower generation is examined.

7.2.1.2. DOMAIN:DSV.EXE

The host may support a domain name server; one is required only if the host is to be authoritative for some portion of the domain space. The server program is named DSV.EXE. If there will not be a server running on the local host the DMS%ES flag in the DOMPAR parameter block should be set to zero.

7.2.1.3. DOMAIN:PPLEGALS.TXT

The name-value correspondence of fields used within the domain system are specified in this file. It must correspond to the latest RFC.

7.2.1.4. DOMAIN:resolver.EXE

The resolver program is still being developed; until one is ready, all host names and addresses of interest will either have to be built into the database or included in the SYSTEM:HOSTS.TXT file. Until the resolver program is available, requests to the resolver by GTHST% should be disabled by specifying a value of zero for the DMS%ER flag in the DOMPAR parameter block.

7.2.2. Building a New Domain Database

A new domain database, FLIP.DD, is built using the MAKEDB utility program. It requires a parameters file (PPLEGALS.TXT), a configuration file, zone files, and optional $INCLUDE files. The PPLEGALS.TXT file contains mappings between keywords found in the files and their value.

7.2.2.1. DOMAIN:MAKEDB.EXE

MAKEDB prompts for the name of the configuration file. The configuration file specifies the names of the files containing each of the zones to be loaded. Each line begins with a load command (only ZLOADF and CLOADF are currently implemented) followed by the domain name of the zone to be loaded, the class, a refresh interval (in seconds), and a file specification of a zone file.

Example lines from a configuration file that specify the BBN.COM zone, part of the IN-ADDR.ARPA zone for a network administered by BBN, and the initial contents of the cache are:

```
ZLOADF BBN.COM.              IN 3600 DOMAIN:BBN.ZONE
ZLOADF 1.192.IN-ADDR.ARPA.   IN 3600 DOMAIN:NET-192-1.ZONE
CLOADF                               DOMAIN:BBN-CACHE.ZONE
```

A zone file contains resource records and $INCLUDE lines. Each resource record begins with an optional domain name for a node; if the node name is omitted the last specified name is used.

The second field specifies restrictions; it is optional. UNAVAILABLE means that use of the record is administratively prohibited, CORESIDENT restricts use of the record to the host, INZONE allows the information to be given to any host within one of the server's authoritative zones, and ANY (the default) allows unrestricted use.

The third field, also optional, specifies the CLASS; if omitted, the last class specified is used.

The fourth field contains the TYPE. Additional fields may be required, depending on the CLASS and TYPE.

The BBN.ZONE file might contain:

```
BBN.COM.    IN  SOA  BBNA.BBN.COM.  Hostmaster.BBN.COM. (
            001     1800     300    604800  3600)
;           serial  refresh  retry  expire  minimum (seconds)

BBN.COM.    IN  NS   BBNA.BBN.COM.  ;Authoritative servers
            IN  NS   BBNG.BBN.COM.

*.BBN.COM.  IN  MF   BBNA.BBN.COM.  ;Take mail for all BBN
            IN  MF   BBNG.BBN.COM.

$include <DOMAIN>BBN.HOSTS BBN.COM. ; BBN under COM
```

The $INCLUDE lines specify a file name and an optional domain name string. If the string is specified, it is appended to all relative domain names (not ending in ".") contained in the file.

A host entry in the BBN.HOSTS file might be:

```
BBNA  INZONE  IN  A      192.1.2.68
              IN  A      10.3.0.5
              IN  A      192.1.2.68
              IN  A      8.5.0.4
              IN  HINFO  DEC-2050T  TOPS20
              IN  MD     BBNA
              IN  MF     BBNG
              IN  WKS    192.1.2.68 ICMP
              IN  WKS    192.1.2.68 TCP FTP SMTP TELNET TIME
              IN  WKS    192.1.2.68 UDP TIME
              IN  WKS    8.5.0.4 ICMP
              IN  WKS    8.5.0.4 TCP FTP SMTP TELNET TIME
              IN  WKS    8.5.0.4 UDP TIME
              IN  WKS    10.3.0.5 ICMP
              IN  WKS    10.3.0.5 TCP FTP SMTP TELNET TIME
              IN  WKS    10.3.0.5 UDP TIME
BBN           IN  CNAME  BBNA
```

The corresponding entry in the DOMAIN:NET-192-1.ZONE file is:

```
68.2.1.192.IN-ADDR.ARPA.  IN  PTR  BBNA.BBN.COM.
```

The DOMAIN:BBN-CACHE.ZONE file might contain:

```
.  IN  NS  SRI-NIC.ARPA.          ; List of root servers
   IN  NS  F.ISI.ARPA.

SRI-NIC.ARPA.  IN  A  10.0.0.51    ; Addresses of (foreign)
               IN  A  26.0.0.73    ; root servers
F.ISI.ARPA.    IN  A  10.2.0.52

$include <DOMAIN>ARPA.HOSTS  ARPA. ; ARPA
$include <DOMAIN>BBN.CNAMES  ARPA. ; BBN Nicknames
```

The ARPA.HOSTS file duplicates the HOSTS.TXT file; it is not really needed as the host tables will be used if the domain system cannot resolve a query. The BBN.CNAMES file contains nicknames used at BBN. Examples of entries are:

```
LABSB.         IN  CNAME  BBN-LABS-B.BBN.COM.
ISIA.          IN  CNAME  A.ISI
```

The resulting FLIP.DD should be copied into DOMAIN: both as FLIP.DD and FLOP.DD; they must be given the same generation number.

7.3. Access Control Job

If an Access Control Job is present in the system (i.e., ACJFN is non-zero), it is used to validate all requests to open a TCP connection via OPENF% (or OPEN%), and all requests to assign a User Queue via ASNIQ%. The default validation test is not used. This permits connections to be restricted on an individual basis, if desired. It also means that the Access Control Job distributed by DEC must be modified to contain the default test if individual control is not desired. Code to implement the default validation for the .GOANA and .GOAIQ functions follows.

7.3.1. Function .GOANA

This function restricts access to TCP. If no ACJ is present, access is not allowed for passive connections to the "well known" (local) TCP ports in the range 1 to 255 unless the user has WHEEL, OPERATOR, ABSOLUTE-ARPANET-SOCKETS, or ARPANET-WIZARD capabilities enabled.

If the ACJ is present, the .GOANA function is called during the OPENF% (and OPEN%) JSYSi. The application argument block pointed to by ARGBLK+.RCARA contains the foreign host address, foreign port number, local port number, and local host address. Values of zero indicate that the parameter was not specified.

Code to perform the test described above is:

```
STANA:  HLRZ T1,ARGBLK+.RCNUA   ; Get number of arguments
        CAIGE T1,4+1            ; FH,FP,LP,LH, plus error word?
         RET                    ; There isn't, deny access
        MOVE T3,ARGBLK+.RCARA   ; Application argument address
        MOVE T1,2+1(T3)         ; Get Local Port
        CAIL T1,^D256           ; Smaller than 256?
        IFSKP.                  ; Yes, see if user has caps
          HRRZ T2,ARGBLK+.RCCAP ; Get user's capabilities
          TRNN T2,SC%WHL!SC%OPR!SC%NWZ!SC%NAS ; Enabled?
          ANSKP.                ; No,
            RET                 ; No, deny access
        ENDIF.
        RETSKP                  ; Allow access
```

7.3.2. Function .GOAIQ

This function restricts access to IP User Queues. If no ACJ is present, access is allowed if:

1) the user has WHEEL, OPERATOR, ABSOLUTE-ARPANET-SOCKETS, or ARPANET-WIZARD capabilities enabled, or

2) if the user did not have the capabilities but either:

2a) the Source Port Mask ANDed with the Source Port Value is greater than 255., or

2b) the Logical Host Mask ANDed with the Logical Host Value is greater than the System Reserved Value (from the variable INTLHX);

otherwise, access is disallowed.

If the ACJ is present, the .GOAIQ function is called during the ASNIQ% JSYS. The application argument block pointed to by ARGBLK+.RCARA contains the six (Value,Mask) pairs: logical host & protocol value, foreign host address value, local host address value, local & foreign port value, and their respective masks, in B0-31.

Code to perform the test described above is:

```
STAIQ:  HLRZ T1,ARGBLK+.RCNUA   ; Get number of arguments
        CAIGE T1,.IQLEN+1       ; Entire QDB plus error word?
          RET                   ; There isn't, deny access
        MOVE T3,ARGBLK+.RCARA   ; Application argument address
        LDB T1,[POINT 16,.IQPTV+1(T3),15] ; Get Local Port Value
        LDB T4,[POINT 16,.IQPTM+1(T3),15] ; Get Local Port Mask
        AND T1,T4               ; Make minimum value
        CAIL T1,^D256           ; Smaller than 256?
        IFSKP.                  ; Yes, see if user has caps
          MOVX T1,SC%WHL!SC%OPR!SC%NWZ!SC%NAS ; Possible caps
          TDNE T1,ARGBLK+.RCCAP ; Enabled?
          ANSKP.                ; No, check if using Logical Host
            LDB T2,[POINT 24,.IQPRV+1(T3),23] ; Log. Host Value
            LDB T4,[POINT 24,.IQPRM+1(T3),23] ; Log. Host Mask
            AND T2,T4           ; Make minimum value
            CALL GETLHX         ; Get INTLHX in T1
              RET               ; Failed, deny access
            CAMG T2,T1          ; In system reserved range?
              RET               ; Yes, deny access
        ENDIF.
        RETSKP

; Look up the system value of INTLHX, the System Reserved Logical
; Host Maximum.  Return it in T1.  Clobbers no other ACs.

GETLHX: SAVEAC <T2,T3>
        STKVAR <<IPBLK,.NTLNP+1>,LHX>
        MOVEI T1,.IPRIP         ; Read network info
        MOVEI T2,IPBLK          ; Address of IP Status Block
        MOVEI T3,.NTLNP+1       ; Block length including header
        MOVEM T3,.NTLEN(T2)
        MOVX T3,<NT%SY!NT%ST>   ; Using symbols, want net info
        MOVEM T3,.NTFLG(T2)
        HRROI T3,[ASCII\INTLH\] ; -1,,Address of ASCII INTLHX
        MOVEM T3,.NTNMP(T2)
        HRROI T3,LHX            ; -1,,Address for return value
        MOVEM T3,.NTDTP
        IPOPR%
          ERJMP R               ; Failure, return
        MOVE T1,LHX             ; Return in AC1
        RETSKP                  ; Success
```

7.4. Telnet Server

The default Telnet Server allows network logins on local port 23 from any foreign host and port provided that network logins are enabled. The parameters specifying the connection are contained in the TVTPR block described above under the SET: command. Timeouts, retransmission parameters, and limited forms (e.g., by network) of restricting remote requests may be specified directly. More complicated requirements may require a separate network login server program or use of an Access Control Job. Note that the internal Telnet Server can be disabled by specifying a local port of zero.

7.5. Compilation Parameters

There are several compilation parameters related to the TOPS-20 internetworking code. The following table lists the parameters, their default values, a description of their function, and the circumstances under which they should be changed. When changing a parameter, include its definition in the sites PARAN.MAC file.

Symbol	Default Value	Function
ALTN	0	The number of ALTO network interfaces to be monitored. Must be non-zero to include ALTO code. Note Alto is unsupported.
ANXN	1	The number of DEC AN20 IMP interfaces to be monitored. Must be non-zero to include AN20 code (modules 1822DV.MAC and IMPANX.MAC).
BBNN	0	The number of BBN IMP10 IMP interfaces to be monitored. Must be non-zero to include IMP10 code (modules 1822DV.MAC and IMPBBN.MAC). Note IMP10 hardware is unsupported.
CHAOS	0	Set non-zero if ChaosNet is included in the system. Note Chaos is not a supported protocol suite.
ETHER	0	Set non-zero if Ethernet is included in the system.
ETHRTS	64.	Maximum number of entries in the "ethernet" address translation table.
IHSHSZ	11.	Size of host's local host address hash table; it must be a prime number. For efficiency, the size should be at least twice the number of local host addresses.
INMNPR	70.	The protocol number used for IP host monitoring. It should be changed when an official number is assigned.
INMNPT	241.	The host (server) IP monitoring port. Port INMNPT-1 is used by the monitoring host. It

should be changed when an official number is assigned.

INTFSZ 200000 The number of words allocated for DARPA Protocol suite buffers (IP packet reassembly, gateway blocks, TCP connection blocks, buffer headers) in the network buffer section (only an integral number of pages is allocated).

IPNIN Minimum of 1 or KNIN
Number of KLNIs for IP.

MAXJCN NTTTVT Maximum number of TCP connections per job. Since Job 0 "owns" TCP virtual terminals, it should be at least as large as the maximum number of TVTs (see also NTTTVT). If other servers are run under Job 0 (FTP, SMTP, etc), the number should be correspondingly larger. Space is allocated within the JSB.

MAXTCB Maximum of 50 or NTTTVT*2
Administrative limit on maximum number of simultaneous TCP connections (listening plus active). The value is contained in the network parameter TCMTC+1, and may be changed statically (include the command "SET:TCMTC+1:<site max>" in the SYSTEM:SITE-ADDRESS.TXT file) or dynamically (e.g., MDDT); recompilation is not necessary.

MNTBSZ 400000 Number of words allocated in the network buffer section for packet buffers (only an integral number of pages is allocated).

NETHSZ 53. Maximum number of entries in the host's routing table cache; it must be a prime number. For efficiency, the size should be twice the number of networks in simultaneous use.

NFEN 0 The number of Network Front End (DTE-20) interfaces to be monitored. Must be non-zero to include NFE code (module NFEPHY.MAC). Note that the front end hardware is unsupported.

NFIXED 11. Number of entries in Internet free storage hash table; should be a prime number. There are only about 4 block sizes which are regularly used within the DARPA protocol suite.

NHOSTS 4001. Maximum number of host address entries in host status tables (cache); it must be a prime number. Until name server support has been completed, the host tables will have to grow to hold all hosts of interest at a site. For efficiency, the size should be twice the number of host addresses. The host tables are allocated in the network buffer section.

NHSTN NHOSTS*4
Number of words allocated for the names and nicknames of hosts in the host tables. The host tables are allocated in the network buffer section.

NHSTNL NHOSTS*2
Number of words allocated for the addresses of hosts in the host tables. The host tables are allocated in the network buffer section.

NIQ 32. Maximum number of IP User Queues assigned. It may be increased if the host supports many application protocols which are implemented externally to the monitor.

NTTBL 1 Number of TTY buffers per line, and
NTTBL1 2 Number of TTY buffers per "fast" line.

The number of data bytes per packet for TCP connections using TVTs is often limited by the number of characters that fit into the TTY output buffer. Therefore, increasing number of buffers allocated per line increases network efficiency when large amounts of data are to be transferred (video screen refreshes, etc.). (It is also possible to change the size of the output buffers allocated, see TTSIZ.)

NTTTVT 32. Maximum number of TCP Virtual Terminals (TVTs). See also MAXJCN.

RFNTSZ 64. Size of 1822 RFNM hash table; must be a power of 2. Should be larger than the maximum number of hosts communicating simultaneously via 1822 networks.

TCBHSZ	17.	Size of TCP connection hash table; must be a prime number. (Note: since the current key is local port, making the table larger (to increase efficiency for a large number of remote TELNET connections), will not help; they will all be hashed into the same bucket. The TCS%O2 option, when fully implemented, will reduce this problem.
TTSIZ	40	Size of a TTY output buffer; must be a power of 4. The number of data bytes per packet for TCP connections that use TVTs is often limited by the number of characters that will fit into the TTY output buffer. Increasing its size will increase network efficiency when large amounts of data are to be transferred (video screen refreshes, etc.). (It is also possible to change the number of output buffers allocated, see NTTBL and NTTBL1.)

8. Structure of the TOPS-20 Internetworking Software

8.1. Network Software Modules

The internetworking software is composed of several modules which parallel Figure 1. At the lowest levels, there are two hardware devices which can be connected to an 1822 network. The DEC supported AN20 and the BBN IMP10; their respective device drivers are IMPANX.MAC and IMPBBN.MAC. Inclusion of either of these devices in a system requires the 1822 Protocol module, 1822DV.MAC. The module IPNIDV.MAC contains routines necessary to use the DARPA Protocol suite over a KLNI interface. The Network Front End software, for both the device driver and the network protocol, are contained in NFEPHY.MAC.

The multinet layer is contained in the module MNETDV.MAC. It contains the general interface to the internetworking code, and includes:

- ATNVT%
- GTHST%, including access to the domain name services
- A free storage package for network buffers
- Routines to start, monitor, and stop network operations, etc.
- Diagnostic and Monitoring Routines

The DARPA Protocol suite consists of the module IPIPIP.MAC, which contains IP, ICMP, and the User Queue interface to IP. IP also requires IPFREE.MAC, a free storage package, and TCPCRC.MAC, which contains checksum routines used by all protocols within the suite.

The TCP layer currently consists of three modules: TCPTCP.MAC, which contains the JSYS interface independent portions of TCP, TCPBBN.MAC, which contains the BBN TCP JSYS interface, and TCPJFN.MAC, which contains the TCP: device JSYS interface and includes TCOPR% and IPOPR%. (Note that TCPJFN currently uses TCPBBN.)

The module TTANDV.MAC is included within TTYSRV to implement TVTs (TCP network terminals), the TELNET protocol, and related MTOPR% functions.

STG.MAC contains all of the internetworking tables and site dependent code.

ANAUNV.MAC contains all of the network related definitions which form the networking library ANAUNV.UNV.

Adding protocol suites and network drivers should only require source modules ANAUNV.MAC and STG.MAC.

8.2. Compiling and Linking a New Monitor

Compiling a new internetworking monitor the first time requires a merge as the changes which were given to DEC in March and October of 1984 have not yet been distributed. The changes currently available are based on DEC's Autopatch 7 tape, 5.4(1022) over 5.1(3046).

The size of ANAUNV, PROLOG, MACSYM, and MONSYM have exceeded the ability of MACRO to compile most of the network modules. To get around this problem, the networking modules are all compiled with special versions of MONSYM (it purges .ERCOD at the end of pass 2) and PROLOG (it does not include the BUGS.MAC file). All bug macros in the networking code (and MEXEC, TTYSRV, and TTPHDV) have been replaced by the BUG. macro.

TCPTCP may still be too large to compile on some systems; if so, it should be compiled twice -- once including a file which contains PART==0 and once with a file which contains PART==2 . The resulting REL files should be called TCPTCP.REL and TCPT2P.REL.) (A separate LOADMODULE for TCPT2P is not necessary as it is included in TCPTCP, if required.) Note that PART==1, the default, compiles the whole module.

Since the network modules distributed by DEC are so far behind, source comparisons are usually not effective. The modules:

1822DV.MAC	ANAUNV.MAC	IMPBBN.MAC	IPFREE.MAC
IPIPIP.MAC	MNETDV.MAC	NFEPHY.MAC	TCPBBN.MAC
TCPCRC.MAC	TCPJFN.MAC	TCPTCP.MAC	TTANDV.MAC

should replace the DEC modules of the same name. STG is also a problem. The easiest way to merge it is probably to replace the section in the middle which involves the networks then do a source comparison to get the rest of the changes. The network globals at the end of GLOBS.MAC should also be replaced.

The following monitor modules must be merged:

ANNLND.MAC	FILMSC.MAC	FORK.MAC	GLOBS.MAC
IPNIDV.MAC	JSYSB.MAC	JSYSF.MAC	MACSYM.MAC
MEXEC.MAC	MONSYM.MAC	PAGEM.MAC	PARAMS.MAC
PARAN.MAC	PHYKNI.MAC	POSTLD.MAC	PROLOG.MAC
SCHED.MAC	STG.MAC	TTPHDV.MAC	TTYSRV.MAC

A CTL file used to compile and link a monitor follows. The directory names, etc., will have to be changed for your site. Check the switch settings at the end. The job runs in a directory which is empty except for the KDDT.REL and MDDT.REL files. <5-4-NET> contains the modified sources, <PAETZOLD> contains the 5.4(1022) DEC sources, and <PAETZOLD.AP7> the 5.1(3046) DEC sources. Dependencies on the ANAUNV library are noted by the /COMPILE switch.

```
@NOERROR
!
DEF DSK:
DEF MON: DSK:,PS:<5-4-NET>,PS:<PAETZOLD>,PS:<PAETZOLD.AP7>
DEF R:   DSK:
DEF SYS: DSK:,SYS:
DEF UNV: DSK:
DEF UTL: DSK:
@ENA
!
@DEL PROLOG.UNV.1000
@EXP
!
MONSYM::
@COMPILE MON:MONSYM.MAC R:MONSYM
MACSYM::
@COMPILE MON:MACSYM.MAC R:MACSYM
!
PROLOG::
@COMPILE MON:PROLG0.MAC+MON:PROLOG.MAC+MON:GLOBS.MAC R:PROLOG
@REN PROLOG.*.0 PROLOG-NO-BUGS.*.*
@COMPILE MON:PROLG0.MAC+MON:PROLOG.MAC+MON:BUGS.MAC -
+MON:GLOBS.MAC R:PROLOG
@COP PROLOG.*.0 PROLOG-BUGS.*.*
!
PROKL::
@COMPILE MON:PROKL.MAC R:PROKL
PHYPAR::
@COMPILE MON:PHYPAR.MAC R:PHYPAR
NSPPAR::
@COMPILE MON:NSPPAR.MAC R:NSPPAR
SERCOD::
@COMPILE MON:SERCOD.MAC R:SERCOD
!
ANAUNV::
@COMPILE MON:ANAUNV.MAC R:ANAUNV
!
@COP PROLOG-NO-BUGS.UNV.0 PROLOG.UNV.1000
```

```
!
NIPAR::
@COMPILE MON:NIPAR.MAC R:NIPAR
NISYM::
@COMPILE MON:NISYM.MAC R:NISYM
NISRV::
@COMPILE MON:REL5.MAC+MON:TOPS.MAC+MON:NISRV.MAC -
+MON:PHYKNI.MAC R:NISRV
NIUSR::
@COMPILE MON:NIUSR.MAC R:NIUSR
NITEST::
@COMPILE MON:REL5.MAC+MON:NITEST.MAC R:NITEST
IPNIDV::
@COMPILE /COMPILE MON:IPNIDV.MAC R:IPNIDV
!
MNETDV::
@COMPILE /COMPILE MON:REL5.MAC+MON:MNETDV.MAC R:MNETDV
1822DV::
@COMPILE /COMPILE MON:1822DV.MAC R:1822DV
IMPANX::
@COMPILE /COMPILE MON:IMPANX.MAC R:IMPANX
IMPBBN::
@COMPILE /COMPILE MON:IMPBBN.MAC R:IMPBBN
NFEPHY::
@COMPILE /COMPILE MON:NFEPHY.MAC R:NFEPHY
!
IPFREE::
@COMPILE /COMPILE MON:REL5.MAC+MON:IPFREE.MAC R:IPFREE
IPIPIP::
@COMPILE /COMPILE MON:REL5.MAC+MON:IPIPIP.MAC R:IPIPIP
!
TCPCRC::
@COMPILE /COMPILE MON:TCPCRC.MAC R:TCPCRC
TCPTCP::
;@COMPILE /COMPILE MON:TCPTCP.MAC R:TCPTCP ; INSUFFICIENT MEMORY FOR 1
PART
@COMPILE /COMPILE PART0.MAC+MON:TCPTCP.MAC R:TCPTCP
TCPT2P::
@COMPILE /COMPILE PART2.MAC+MON:TCPTCP.MAC R:TCPT2P
TCPBBN::
@COMPILE /COMPILE MON:REL5.MAC+MON:TCPBBN.MAC R:TCPBBN
TCPJFN::
@COMPILE /COMPILE MON:REL5.MAC+MON:TCPJFN.MAC R:TCPJFN
!
TTYSNV::
@COMPILE /COMPILE MON:ARPAF.MAC+MON:KLPRE.MAC+MON:TTYSRV.MAC -
+MON:TTANDV.MAC+MON:TTPHDV.MAC R:TTYSNV
TTYSDN::
```

```
@COMPILE /COMPILE MON:ARPAD.MAC+MON:KLPRE.MAC+MON:TTYSRV.MAC -
+MON:TTANDV.MAC+MON:TTPHDV.MAC R:TTYSDN
!
STGAN::
@COMPIL/COMP MON:REL5.MAC+MON:KLPRE.MAC+MON:ANPLND.MAC -
+MON:PARSYF.MAC+MON:PARAMS.MAC+MON:STG.MAC R:STGAN
!
JSYSB::
@COMPILE MON:JSYSB.MAC R:JSYSB
!
MEXEC::
@COMPILE /COMPILE MON:MEXEC.MAC R:MEXEC
!
@DEL PROLOG.UNV.1000
@EXP
!
EFILIN::
@COMPILE /COMPILE MON:EXPRE.MAC+MON:FILINI.MAC R:EFILIN
FILMSC::
@COMPILE /COMPILE MON:FILMSC.MAC R:FILMSC
FORK::
@COMPILE /COMPILE MON:FORK.MAC R:FORK
JSYSF::
@COMPILE /COMPILE MON:JSYSF.MAC R:JSYSF
PAGEM::
@COMPILE /COMPILE MON:PAGEM.MAC R:PAGEM
POSTLD::
@COMPILE /COMPILE MON:POSTLD.MAC R:POSTLD
SCHED::
@COMPILE /COMPILE MON:SCHED.MAC R:SCHED
!
TAPE::
@COMPILE MON:TAPE.MAC R:TAPE
DTESRV::
@COMPILE MON:KLPRE.MAC+MON:DTESRV.MAC R:DTESRV
GETSAV::
@COMPILE MON:GETSAV.MAC R:GETSAV
LDINIT::
@COMPILE MON:LDINIT.MAC R:LDINIT
APRSRV::
@COMPILE MON:KLPRE.MAC+MON:APRSRV.MAC R:APRSRV
COMND::
@COMPILE MON:COMND.MAC R:COMND
DEVICE::
@COMPILE MON:DEVICE.MAC R:DEVICE
DIRECT::
@COMPILE MON:DIRECT.MAC R:DIRECT
```

```
DISC::
@COMPILE MON:DISC.MAC R:DISC
SWPALC::
@COMPILE MON:SWPALC.MAC R:SWPALC
DSKALC::
@COMPILE MON:DSKALC.MAC R:DSKALC
ENQ::
@COMPILE MON:ENQ.MAC R:ENQ
FREE::
@COMPILE MON:FREE.MAC R:FREE
FUTILI::
@COMPILE MON:FUTILI.MAC R:FUTILI
GTJFN::
@COMPILE MON:GTJFN.MAC R:GTJFN
IO::
@COMPILE MON:IO.MAC R:IO
IPCF::
@COMPILE MON:IPCF.MAC R:IPCF
JSYSA::
@COMPILE MON:JSYSA.MAC R:JSYSA
LINEPR::
@COMPILE MON:KLPRE.MAC+MON:LINEPR.MAC+MON:LPFEDV.MAC R:LINEPR
CDPSRV::
@COMPILE MON:CDPSRV.MAC R:CDPSRV
CDRSRV::
@COMPILE MON:KLPRE.MAC+MON:CDRSRV.MAC+MON:CDKLDV.MAC R:CDRSRV
LOGNAM::
@COMPILE MON:LOGNAM.MAC R:LOGNAM
LOOKUP::
@COMPILE MON:LOOKUP.MAC R:LOOKUP
MAGTAP::
@COMPILE MON:MAGTAP.MAC R:MAGTAP
MSTR::
@COMPILE MON:MSTR.MAC R:MSTR
PHYH2::
@COMPILE MON:PHYH2.MAC R:PHYH2
PHYM2::
@COMPILE MON:PHYM2.MAC R:PHYM2
PHYM78::
@COMPILE MON:PHYM78.MAC R:PHYM78
PHYP4::
@COMPILE MON:PHYP4.MAC R:PHYP4
PHYX2::
@COMPILE MON:PHYX2.MAC R:PHYX2
PHYP2::
@COMPILE MON:PHYP2.MAC R:PHYP2
PHYSIO::
```

```
@COMPILE MON:PHYSIO.MAC R:PHYSIO
PLT::
@COMPILE MON:PLT.MAC R:PLT
PTR::
@COMPILE MON:PTR.MAC R:PTR
PTP::
@COMPILE MON:PTP.MAC R:PTP
DIAG::
@COMPILE MON:DIAG.MAC R:DIAG
SYSERR::
@COMPILE MON:SYSERR.MAC R:SYSERR
TIMER::
@COMPILE MON:TIMER.MAC R:TIMER
FESRV::
@COMPILE MON:KLPRE.MAC+MON:FESRV.MAC R:FESRV
MFLIN::
@COMPILE MON:MFLIN.MAC R:MFLIN
MFLOUT::
@COMPILE MON:MFLOUT.MAC R:MFLOUT
DATIME::
@COMPILE MON:DATIME.MAC R:DATIME
NSPSRV::
@COMPILE MON:NSPSRV.MAC R:NSPSRV
!
@COMPIL/COMP MON:ANNLND.MAC+MON:VERSIO.MAC R:VERSIO
!
@NOERROR
!
LINK::
@R LINK
TTY:/LOG/LOGL:5
AMON/SAVE,/HASHSIZE:12007, -
/FRECOR:0, -
/SET:RSCOD:1000, -
/SET:INCOD:121000, -
/SET:RSDAT:143000,/SYMSEG:PSECT:RSDAT,/PATCHSIZE:100, -
/SET:PPVAR:205000, -
/SET:RSVAR:211000, -
/SET:SYVAR:327000, -
/SET:PSVAR:330000, -
/SET:JSVAR:346000, -
/SET:NRVAR:424000, -
/SET:BGPTR:450000/UPTO:447777, -
/SET:NRCOD:452000, -
/SET:BGSTR:727000, -
/SET:POSTCD:741000, -
/SET:NPVAR:727000, -
```

```
R:MONSYM.REL, -
R:LDINIT, -
R:VERSIO,R:STGAN, -
R:APRSRV.REL/s,R:SCHED.REL/s,R:PAGEM.REL/s,R:FORK.REL/s, -
R:MEXEC.REL/s,R:GETSAV.REL/s,R:SYSERR.REL/s,R:COMND.REL/s, -
R:DEVICE.REL/s,R:DIRECT.REL/s,R:ENQ.REL/s,R:FREE.REL/s, -
R:FUTILI.REL/s,R:GTJFN.REL/s,R:IO.REL/s,R:IPCF.REL/s, -
R:JSYSA.REL/s,R:JSYSF.REL/s,R:LOGNAM.REL/s,R:LOOKUP.REL/s, -
R:MSTR.REL/s,R:SWPALC.REL/s,R:DISC.REL/s,R:EFILIN.REL/s, -
R:FILMSC.REL/s,R:MFLIN.REL/s,R:MFLOUT.REL/s,R:DATIME.REL/s, -
R:PHYSIO.REL/s,R:DIAG.REL/s,R:DSKALC.REL/s,R:PHYH2.REL/s, -
R:PHYP4.REL/s,R:PHYP2.REL/s,R:TTYSDN.REL/s,R:FESRV.REL/s, -
R:MAGTAP.REL/s,R:TAPE.REL/s,R:TIMER.REL/s,R:PHYM2.REL/s, -
R:PHYM78.REL/s,R:PHYX2.REL/s,R:DTESRV.REL/s,R:LINEPR.REL/s, -
R:CDRSRV.REL/s,R:PLT.REL/s,R:PTR.REL/s,R:PTP.REL/s, -
R:CDPSRV.REL/s,R:TTYSNV.REL/s, -
JSYSB,1822DV/s,IMPBBN/s,NFEPHY/s, -
R:IMPANX.REL/s,R:IPIPIP.REL/s,R:IPFREE.REL/s, -
R:TCPTCP.REL/s,R:TCPT2P.REL/s,R:TCPCRC.REL/s,R:MNETDV.REL/s, -
R:TCPJFN.REL/s,R:TCPBBN.REL/s, -
R:NSPSRV.REL/s, -
R:IPNIDV.REL/s,R:NISRV.REL/s,R:NITEST.REL/s,R:NIUSR.REL/s, -
R:KDDT.REL,R:MDDT.REL,R:POSTLD.REL, -
*/G
@EXP
AMON::
@GET AMON
@START 142
=BUGHLT<HLTADR□12B
=BUGCHK<CHKADR□11B
DBUGSW/1
DCHKSW/1
BUGH0+1/JFCL
=□G
!
@
%TERR::
@CONT
@
%FIN::
!
@DEL PROLOG.UNV.1000
@EXP
!
```

8.3. Support and Utility Routines

Several routines have been written during the course of development of the internetworking software; they are listed below. They all use the ANAUNV.UNV library.

Program	Function
PICKLE	Files monitoring datagrams and can request them from remote hosts. (See sections 16.1.3 and 16.1.4 for examples of using PICKLE.)
PKTPRN	Filters a binary packet trace, prints a text listing. (See section 15.2.2 for an example of using PKTPRN.)
STS	Lists monitoring reports previously filed by PICKLE, or writes custom reports based on them. (See section 16.1.5 for an example of using STS.)
TCPEEK	Snapshots the local host's monitoring information; can examine gateway tables and TCP connection blocks. (See section 16.1.2 for an example of using TCPEEK.)
TCPTST	Tests some TCP functions, including data Source, Sink, and Echo. Provides IP Pinger and current route functions. (See section 15.4.1 for an example of using TCPTST.)
TCPU	Uses the Logical Host feature either to watch (and file) packets between two hosts or to be an ill-behaved TCP data sink (long delays, dropping packets, sending ICMP messages, etc). (See section 15.4.2 for an example of using TCPU.)
TSTATS	Displays several parameters for each of the specified TCP connections. (See section 16.1.1 for an example of using TSTATS.)

9. Overview of Internetworking Structure

Protocol suite and driver independence is achieved through the use of dispatch vectors and Network Control Tables (NCT).

9.1. Protocol Suites

The link between the multinet layer and the protocol suites is through fifteen tables in STG.MAC. Each protocol suite is assigned (in ANAUNV.MAC) a symbol in the form of NP.xxx, where xxx is a mnemonic for the suite. The nth entry in the tables corresponds to the suite whose NP.xxx value is n. (Note that the current code only supports seven suites, numbered 1 through 7. Historically, IP was code zero; this is being phased out in favor of a non-zero code. Some suites have been defined, but not implemented in the standard monitor, e.g., CSNet and OSI. Unused codes may be reassigned freely, they are site-dependent definitions.) Protocol suite codes and domain classes currently use identical codes.

The tables have names in the form of PROxxx. An entry is placed into a table using the PENTRY macro. It has three arguments:

- the protocol symbol, NP.xxx,
- the monitor assembly flag that, if non-zero, includes the protocol in the monitor, and
- the value for the table entry.

Note that the PENTRY macros must appear in numerical order according to the value of the protocol symbol NP.xxx. The table names and their respective functions are:

Name	Function
PROCZF	Dispatch to a routine called to close and release any files for a job-relative fork handle (i.e., CLZFF%).

PROEIN Dispatch to a routine which accepts a packet for the protocol suite which has arrived from the network.

PROHST Dispatch to a routine that can translate a standard form binary network address and port into an ASCII string. It implements the GH%ADR and GH%PRT flags of the GTHST% .GTHNS function. It is the inverse of PRONUM.

PROIAD Dispatch to the routine that translates a standard form binary network address into an inverted domain name string. It is part of the GTHST% .GTHNS function.

PROINZ Dispatch to the routine that initializes the protocol suite.

PROKFK Dispatch to a routine that releases any resources assigned to it when a fork kills itself (i.e., KFORK%).

PROKJB Dispatch to a routine that releases any resources assigned to a job when the job is killed (i.e., LGOUT%).

PRONAM Address of the ASCIZ protocol suite name.

PRONET Dispatch to a routine that extracts a network number from a standard form address. (Note: One way to perform sub-net routing to return a "subnetwork number".)

PRONUM Dispatch to a routine that can parse an ASCII string for a numeric host address and, if the protocol suite uses ports, a port number. It implements the GH%PRT flag of the GTHST% .GTHSN function. It is the inverse of PROHST.

PRONDN Dispatch to a routine called when a interface goes down.

PRONUP Dispatch to a routine called when a interface comes up.

PROODN Dispatch to a routine that performs any necessary operations after a packet has been transmitted on a network.

PROON Indirect address of a flag word; indicates whether or not the protocol has been initialized and turned on.

PROOVH Contains the number of words of protocol-suite specific packet buffer overhead.

9.2. Network Interfaces

There are two basic layers of software below the multinet layer. The lower layer (device driver) consists of the software required to drive the hardware interface to a network. Each type of hardware interface, such as an AN20, is assigned a symbol in ANAUNV in the form of NH.xxx, where xxx is a mnemonic for the hardware interface.

The upper layer has two functions. The first is to translate between (internal address, protocol code) pairs and the local leader fields required by the network. The second is to implement any protocol necessary to interact with the network or front end, independent of the actual hardware interface used. Each protocol, such as 1822, is assigned a symbol in ANAUNV in the form of NL.xxx, where xxx is a mnemonic for the network protocol. Most of the knowledge of how each protocol suite communicates to a network is present in this layer.

The interface to each of these layers is through a dispatch vector pointed to by the NCT, the Hardware Vector and Protocol Vector. The vector names and functions are briefly described below.

9.2.1. NCTs

There is one NCT for each (local host address, device driver) pair. An NCT consists of two logical areas: the first is associated with a local host address (and hence, indirectly, with a protocol suite), and the second (if present) is associated with a device driver (it contains all of the driver's variables). NCTs containing both parts are called "physical NCTs". If an interface supports multiple local addresses (or protocol suites), other "virtual NCTs" exist which contain only the first part and a link to the physical NCT containing the device driver's variables.

The following diagram describes the relationships between the protocol suite tables, the network Protocol Vector entries, the device Hardware Vector entries, and the network layers and monitor JSYSi.

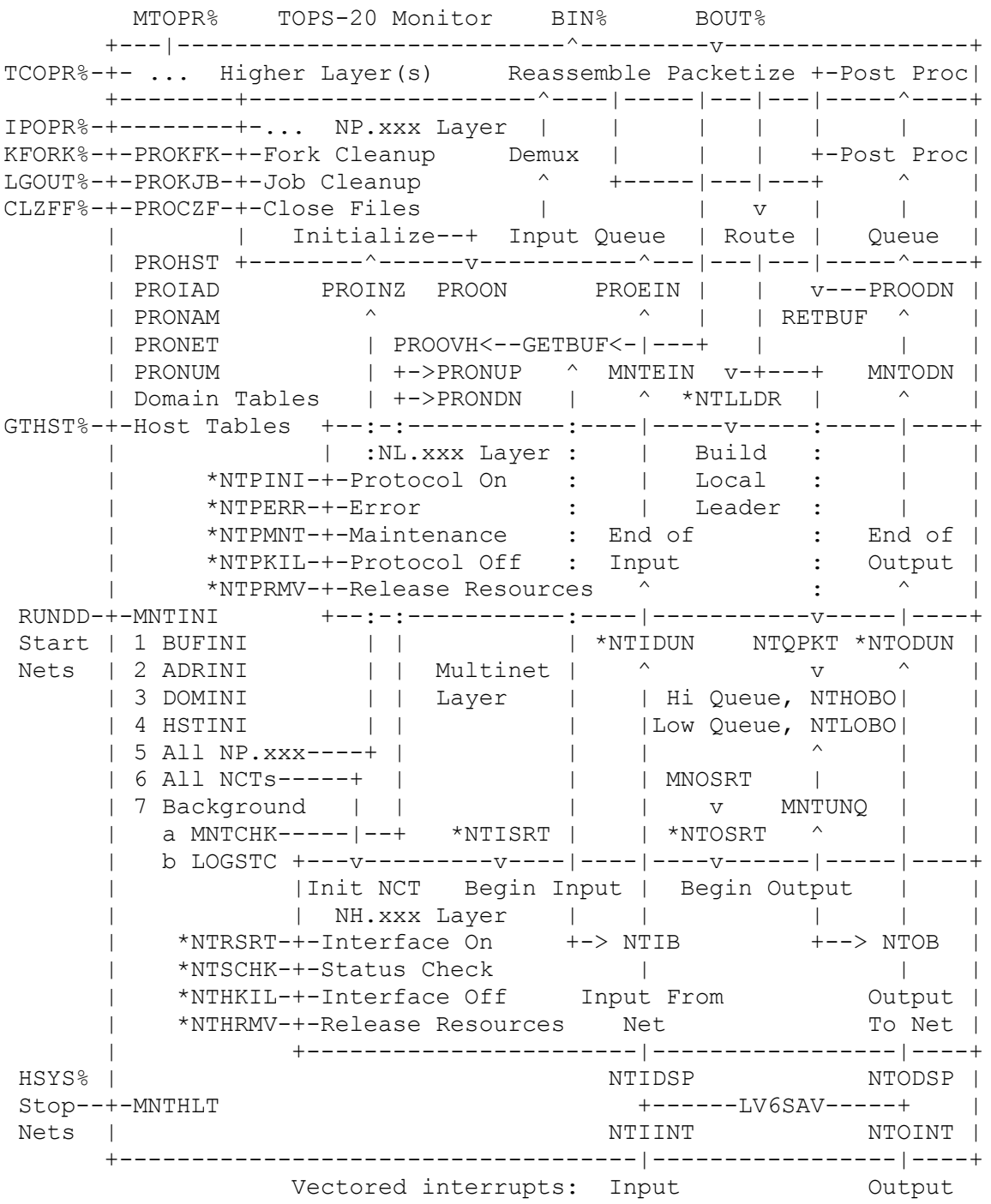


Figure 5. Protocol Layers within the TOPS-20 Monitor

9.2.2. Protocol Vector

Each NCT points to a Protocol Vector which has eight entries:

Routine	Description
NTIDUN	Address of a routine to process a packet received from (one of) the hardware interface(s). After performing any operations related to the network protocol, it passes the packet to MNTEIN, which then passes it to the proper protocol suite via the PROEIN table.
NTLLDR	Address of a routine which builds the local leader required to send a packet to a specified network address (next hop).
NTODUN	Address of a routine called when the hardware has completed transmission of a packet. After performing any operations related to the network protocol, it passes the packet to MNTODN, which then passes it to the proper protocol suite via the PROODN table.
NTPERR	Address of a routine called when a hardware error is detected. The routine should perform any operations necessary to make the network protocol operational.
NTPINI	Address of a routine which enables the local host address associated with the NCT.
NTPKIL	Address of a routine which disables the local host address associated with the NCT.
NTPMNT	Address of a routine which is called periodically to perform any protocol related maintenance operations.
NTPRMV	Address of a routine called just before the NCT is to be flushed to release any system resources acquired when the NCT was initialized.

9.2.3. Hardware Vector

Each NCT points to a Hardware Vector which has six entries:

Routine	Description
NTHKIL	Address of a routine to disable the device associated with the NCT.
NTHRMV	Address of a routine called just before the NCT is to be flushed to release any resources which were acquired when the NCT was initialized.
NTISRT	Address of a routine to begin reception of another packet from the network. The multinet routine GETBUF should be used to obtain packet buffers (which are locked in memory). When a packet is available, it is passed to the protocol module via NTIDUN.
NTOSRT	Address of a routine to begin transmission of another packet. Packets to be sent should be removed from the output queue(s) using the multinet routine MNTUNQ. When a packet has been transmitted, it is passed back to the protocol module via NTODUN.
NTRSRT	Address of a routine to enable the hardware device associated with the NCT.
NTSCHK	Address of a routine called to make sure a packet is OK to process (i.e., no errors were detected while the packet was being sent or received). It should, if possible, check that the device is still operational.

10. Control Structure

This section provides an overview of the control structures used by the internetworking software.

10.1. Initialization and Multinet Background

Initialization of the internetworking software begins when MNTINI is called (at RUNDD7 in MEXEC). After the network buffer pool has been initialized (BUFINI), the multinet background and utility forks are created and started. Each fork first calls the MNTFKI routine in order to initialize the fork to run at high priority in monitor context.

The utility fork runs on demand in order to assist processing host down status information supplied by the networks, and to write out packet trace data, if it exists.

Before entering the background loop, the background fork first:

- calls ADRINI to process the SYSTEM:SITE-ADDRESS.TXT file (creating and initializing the NCTs),
- calls DOMINI to initialize the domain databases from the files DOMAIN:FLIP.DD and FLOP.DD,
- calls HSTINI to process the SYSTEM:HOSTS.TXT file (entering the permanent host names into the host translation cache) and initialize the host status tables,
- calls NETHSI to initialize the network hash table, and
- for each protocol suite whose @PROON flag has 1B35 set, calls the PROINZ routine to initialize the respective protocol suite (but they should not start normal execution until bit 1B0 of NETSUP is set).
- If DBUGSW is less than 2, the NETSUP flag is set to -1, which allows the protocol suites to begin execution. The MNETON routine is also run to enable the local addresses associated with each of the NCTs (NTPINI dispatch in the Protocol Vector) and to turn on the networks associated with the physical NCTs (NTRSRT dispatch in the Hardware Vector).

Each execution of the multinet background loop calls MNTCHK, which checks on the status of each of the NCTs, and reschedules itself to run later (usually every minute, or when awoken by a call to MNTWAK).

For each NCT, the MNTCHK routine first calls the protocol maintenance routine (via the NTPMNT dispatch in the Protocol Vector) and then checks on the status of the associated protocol and hardware (if any). Status changes are logged on the CTY by the LOGSTC routine. If the SCTLW indicates that the system is being shutdown, the physical networks are turned off (via the NTHKIL dispatch in the Hardware Vector).

MNTCHK also calls the GCBUF routine to garbage collect the network buffer pools.

10.2. Packet Buffers

The packet buffer consists of two basic areas, possibly with a Gap between them. The structure of a packet buffer is shown in Figure 6. In the Packet-Private Local Variables area variables are grouped by protocol layer; each group consists of a common and protocol specific sub group. The Packet Image Area contains a subarea for each protocol leader (or data). Each layer has a size (PxxBZ) and location (PxxDT) variable. Size is measured in 8-bit bytes, including the protocol's leader and data areas, as shown in the figure. (Note that the location variables should be written using the standard STOR macro, and read with the special PNTLDR macro.)

Packets being received from a network should place the local leader at LCLPKT(PKT) (i.e., the "Unused" area is zero). Packets being formed by one of the protocols should begin the data area MAXOVH words into the packet; each protocol layer should build its leader before going to the next level (there will usually be a Gap in these packets).

10.3. Input Packet Flow

Reception of packets from a network is initiated through the NTISRT entry in the Hardware Vector. There are many ways to implement the device driver; the suggested method is to enable an input interrupt, which dispatches via the NTIDSP address in the NCT, and then start the network device. (See Figure 5.) As input interrupts occur, the dispatch address can be changed.

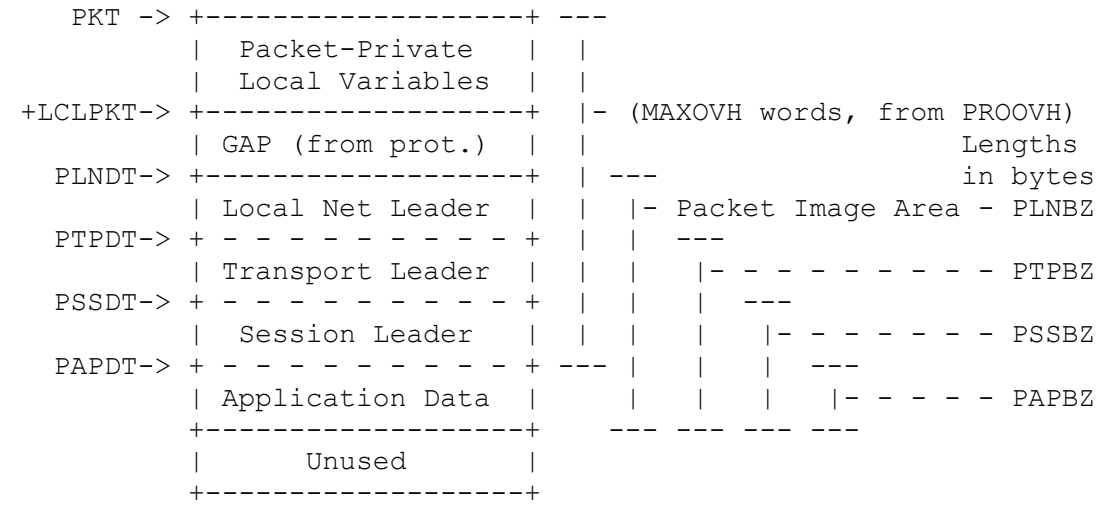


Figure 6. General Packet Buffer

The device driver decides the size of the packet and its protocol suite. The GETBUF routine is called with this data (or some assumptions) to obtain a packet buffer for the packet (GETBUF consults the PROOVH table to find out how much protocol dependent overhead is required and zeros the Packet Private Local Variables area in the buffer).

The address of the packet buffer is placed into the NTIB variable in the NCT. The PLNBZ and PLNDT variables in the packet buffer header should be set to the packet length, in bytes, and the address of the local leader.

After the packet has been placed into the buffer, the device driver passes the packet to the network layer (NL.xxx) via the NTIDUN dispatch in the Protocol Vector.

When the network supports multiple protocol suites or local addresses, the network layer must find the matching NCT, possibly dispatching via NTIDUN a second time.

After processing the local leader, the transport level variables PTPBZ and PTPDT should be set to the transport level length and header location. After completing any operations required by the network (or front end) protocol, the packet and protocol suite code (NP.xxx) are passed to the MNTEIN routine.

MNTEIN first performs common operations for all packets received, and then passes the packet to the protocol suite via

the PROEIN dispatch table. Since the PROEIN routine can be called from interrupt level, it should queue the packet for later processing by the protocol suite's process level fork and cause the fork to be made runnable.

Before returning, MNTEIN automatically initiates reception of the next packet via the NTISRT dispatch in the Hardware Vector.

The process level fork which implements the protocol suite then:

- removes the packet from its input queue,
- processes the transport level leader,
- sets the session level length and leader location in PSSBZ and PSSDT, and
- passes it to the appropriate session level processing routine.

After the session level leader is processed and the application level length and data location set into PAPBZ and PAPDT, the packet data can be passed to the application (e.g., via BIN%s).

The packet buffer should be returned by passing it to the RETBUF routine.

10.4. Output Packet Flow

Output operations usually begin by obtaining a packet buffer by calling the GETBUF routine with the protocol suite code, NP.xxx, and the amount of space needed. Space should be reserved in the packet buffer for the longest set of session, transport, and local leaders (i.e., MAXOVH words). PAPDT should be set to point after this area, and PAPBZ should be set to the number of data bytes placed into the packet (e.g., from BOUT%s).

When the packet is full, a session leader should be placed before the data. PSSBZ and PSSDT should set to the total packet length and location of the session leader.

The protocol suite's transport leader is then constructed and PTPBZ and PTPDT are set.

To send the packet the first hop destination of the packet and the NCT must be determined. The destination address, NCT, and packet are passed to the local leader building routine specified by the NTLLDR dispatch of the NCT's Protocol Vector.

Finally, if the leader is constructed successfully and the interface is functional, NTQPKT is called to place the packet into one of the NCT's output queues.

The transmission of a packet begins when the MNOSRT routine is called (initially via the multinet background loop). After checking that no output is in progress (NTOB is zero), the device driver is called via the NTOSRT dispatch in the Hardware Vector.

It usually calls the MNTUNQ routine to get the next packet to be sent and saves the packet's address in the NTOB variable in the NCT.

The hardware is started after setting the address of an interrupt routine into the NTODSP NCT offset and enabling output interrupts.

After the final interrupt has been processed, the NTODUN dispatch in the Protocol Vector passes the packet back to the network level (NL.xxx). Transmission of the next packet should then be initiated by executing JRST MNOSRT.

The network level routine performs any operations the network needs (including processing any detected transmission failures) and passes the packet to the MNTODN routine. MNTODN passes the packet to the appropriate protocol suite via the PROODN dispatch table. That routine can return the packet buffer by calling RETBUF, or can queue the packet for further processing by the protocol suite (in process context).

11. Adding a Protocol Suite

Addition of a protocol suite requires:

1) Defining a compilation symbol; e.g., xxxN, which controls whether (non-zero) or not (zero) the protocol suite should be included in the monitor build.

2) Adding an NP.xxx definition to ANAUNV.MAC.

3) Adding an entry to the TYPNAM keyword table in STG to associate a protocol suite name string (domain class) with a routine to parse an ASCII network address in the appropriate format. Note that these routines are currently all in MNETDV.MAC and share a common set of parsing routines; see ADIPP.

4) The name must also be specified to the domain system as a "dtype" in the PPLEGALS.TXT file.

5) An inverse address domain name must be added to the IAORG table.

6) Adding entries to the fifteen PROxxx tables in STG at the offset defined by NP.xxx, including the address of the protocol name string in PRONAM, the on/off variable's address in PROON, and the maximum leader length in PROOVH.

7) Writing the twelve routines that correspond to the remaining PROxxx tables. Note that the routines for PROEIN and PROODN should be resident; they may be called from interrupt level.

A routine may appear in more than one entry in a table if two or more protocols share common formats.

11.1. External Implementations

Protocol suites can be added in two ways: internal to the monitor and external to the monitor. In the latter case (which is not supported at present), monitor calls (JSYSi) will have to be added to allow application data (i.e., from BIN% and BOUT%), control information, and (partial) packet images to be passed between the monitor and the external protocol suite implementation.

The PROINZ routine will use MNTFKI to create and initialize any forks the protocol suite requires. The other routines , which implement the protocol suite, need to call the GETBUF and RETBUF routines and may want to access the host tables. The routine which sends packets will probably call the PRONET routine (to get a net number) and the NETNCT routine (to determine the NCT to which the packet should be queued), MNTCALL NTLLDR (to construct the local leader), and the NTHSND or NTLSND routines to queue the packet (see NTSNDI for an example).

11.2. Internal Implementations

Detailed descriptions of the PROxxx tables follow. Some of the tables contain information or identify routines which can be called even if the protocol suite is not on (see PROON), as may be the case when a protocol suite was on but has been turned off. These tables are described first.

Entries that may be accessed if the protocol is OFF:

Entry	Description
PRONAM	The extended address of the ASCIZ string containing the protocol suite name. It is used in the identification of interfaces which come up or go down. It is recognized by GTHST%, and may be included in address to string translations if the GH%PSU flag is set. It may also be specified in the HOSTS.TXT file.
PROOVH	Contains the number of words of protocol suite specific packet buffer overhead. This determines how much space is allocated for protocol leaders in a packet buffer obtained via the GETBUF routine. The table is resident as it may be accessed at interrupt level.
PROON	The on/off variable whose extended address is in PROON has three assigned bits. B35, when initially set, indicates that the PROINZ routine should be called at system startup to initialize the protocol suite. The protocol initialization routine may use B19-34 as flags to indicate what it has completed, and should leave B19-35 zero when it is done. 1B18 means that the

protocol should also be turned on when the system is started. It should be cleared and B0 should be set to indicate that the protocol suite is ON when the protocol suite is useable. If B18 is not set, B0 should be left zero to indicate that the suite is OFF. The variable should be resident as it may be accessed at interrupt level.

PROINZ Contains the extended address of the routine which initializes the protocol suite. It is called with the protocol suite code (NP.xxx) in T1 and should act according to the flags pointed to by PROON, described above, creating and starting any forks required. The routine itself should not wait for anything, but any forks which are created should CALL DISL to wait until B0 of NETSUP is set. The routine MNTFKI should be used to create forks to run under multinet.

PROHST Contains the extended address of a routine which can translate a standard form binary network address and port into an ASCII string. It implements the .GTHNS GTHST% function for the protocol suite. The MNOUTD routine is available to output decimal numbers (NOUT% cannot be used because the previous context will be incorrect). The routine's arguments are:

0 Flags: GH%ADR is set if address string should be printed, GH%PRT if a port is to be printed.
1 Destination designator.
2 NP.xxx protocol suite code.
5 Standard form network address.
6 Additional address bits or optional port number.
7 Instruction that when executed will output the character in 2.

It should return +1 if an error is detected or +2 otherwise with:

1 Updated destination designator.

PROIAD Contains the extended address of a routine repeatedly called to translate the next field of a standard form binary network address into an inverted domain label. Each label in the name written to the destination string must begin with a label length octet (the length octet itself is not included in the count) followed by that many label characters. For example, the INTERNET protocol suite routine would return the four labels "<3>234", "<1>1", "<2>90", and "<1>8" for the address "8.90.1.234". It is part of the GTHST% .GTHNS function. The routine's arguments are:

1	Destination byte pointer.
2	NP.xxx protocol code.
3,4	-1 the on first call, should be used to distinguish subsequent calls.
5,6	Standard form network address.

The routine should return +1 if more labels are required and skip return after the last label has been written to the destination string. It should return:

1	Updated destination byte pointer.

PRONET Contains the extended address of a routine called to extract a "network number" from a standard form host address. T1 contains the host address, with the protocol code NP.xxx, in field NA%PRO. The routine should return a "network number" in T1, also containing the protocol code in field NA%PRO. The (sub)network number can be anything as long as it can be distinguished from a host address, the PROHST routine can print it, and the PRONUM routine can parse it. The network number is used to cache a route to the network.

PRONUM Contains the extended address of a routine to parse an ASCII string representing a host (or net) address and translate it into a one word standard format network address; see section 5.1, Network Address Formats. Since the presence of the address field is optional, the routine must be able to detect that the item is not an address. If an address is found, the GH%ADR flag should be set. If GH%PTR is set, a port should be parsed after the (optional) address. This routine implements part of

the GTHST% .GTHSN function. It is the inverse of PROHST. The routine's arguments are:

0 Flags: GH%PRT is set if a port should be parsed.
1 Source designator.
2 NP.xxx protocol suite code.
7 Instruction that when executed will input the next character into 2.

It should always return +1 if an error is detected (e.g., GH%PRT is set but there is no port number) or +2 if successful with:

0 Updated flags: GH%ADR is set if an address was parsed.
1 Updated source designator.
5 Standard form network address (if found and GH%ADR was set).
6 Additional address bits and, if GH%PRT was set, port number.

PROKJB Contains the extended address of a routine which releases any protocol suite related resources assigned to the job whose job number is in JOBNO. PROKJB is called when a job is killed, and saves all registers. It should check that the protocol suite has been initialized, but the suite may be OFF.

PROKFK Contains the extended address of a routine which releases any protocol suite related resources assigned to the system fork whose number is in FORKX. PROKJB is called when a fork kills itself, and saves all registers. It should check that the protocol suite has been initialized, but the suite may be OFF.

PROCZF Contains the extended address of a routine which closes and releases any protocol suite related files given the job-relative fork handle in T1 (e.g., a FORKN). PROCZF implements the CLZFF% function and saves all registers. It should check that the protocol suite has been initialized, but the suite may be OFF.

Routines only called if the protocol suite is ON are:

Routine	Description
PROEIN	Contains the extended address of a routine which should be called when a packet for the protocol suite has arrived. The routine should be resident as it is called at interrupt level; it is suggested that the routine queue the packet for later processing by the protocol suite's process level fork (and make the process runnable). T1 has the extended address of a standard packet buffer; PTPBZ and PTPDT have been set to the number of 8-bit bytes at the transport level and the address (via the PNTLDR macro) of the transport leader. When the protocol suite is finished with the packet, it releases the storage by indirectly calling PROEIN+NP.GEN (i.e., RETBUF) with the extended packet address in T1.
PROODN	Contains the extended address of a routine called when a packet for the protocol suite has been transmitted on the selected network (or was not sent due to interface error, see PLNXO). If the suite does not perform any post transmission processing, and does not need to retain the packet (e.g., for retransmissions by a higher level protocol), the RETBUF routine may be specified. If a routine is supplied, it should be resident as it is called at interrupt level. It is suggested that the routine queue the packet for later processing by the protocol suite's process level fork (and make the process runnable). T1 has the extended address of a standard packet buffer. When the protocol suite is finished with the packet, it releases the storage by indirectly calling PROODN+NP.GEN (i.e., RETBUF) with the extended packet address in T1.

PRONDN Contains the extended address of a routine called when a local host address or interface goes down. VNCT (i.e., P1) has the NCT address of the failing local address; NTNET and NTLADR in the NCT contain the standard form network number and address.

PRONUP Contains the extended address of a routine called when a interface comes up. VNCT (i.e., P1) has the NCT address of the now functioning local address; NTNET and NTLADR in the NCT contain the standard form network number and address.

12. Adding a Protocol above IP

Internet protocols may be implemented either internally or externally to the monitor. When the protocol is implemented externally, the IP User Queue facility, described previously, is used to assign the IP protocol number to the application program which then may send and receive internet packets.

Protocols implemented internal to the monitor communicate with IP via direct subroutine calls. IP provides several routines (in IPIPIP.MAC); (see the documentation in IPIPIP.MAC). There are routines associated with:

- IP event flags (ASNWTB, SETWTB, CLRWTB, RELWTB),
- IP scheduler tests (INTBZT, INTBOT, INTOOT, INTZOT),
- IP locks (CLRLCK, SETLCK, UNLCK, RELLCK, LCKCAL),
- IP queues (INITQ, NQ, DQ),
- sending internet packets (SNDGAT),
- reporting an errors via ICMP (ICMERR),
- computing checksums (DATCKS), etc.

AOSing the variable INTFLG causes the internet fork to be run.

Communication between IP and the protocol is via a dispatch table in STG generated by using the PCLTAB macro. The table requires that seven routines be defined, defines five variables, and includes three constants. The macro should be invoked before "PCLTAB (INQ,-1)", near the label INTPIX. The negative count of internal protocols in INTPIX should be incremented. The macro has three arguments: a three character IP protocol name ("xxx"), an optional protocol code ("prt"), and an optional flag.

If a symbol in the form of .xxxFM has not been defined (in ANAUNV.MAC) equal to the IP protocol number of the protocol being implemented, the second argument should be that number. The other two constants are used by the output queue monitoring code. They are created using a macro analogous to the TCPSTS macro in ANAUNV.MAC (the macro should be invoked wherever TCPSTS is). The symbols xxxAB and MxxxAB (defined by the NTITEM macro) will placed into the table. If this monitoring is not to be

performed, a non-blank third argument should be given in the PCLTAB invocation.

The five global variables defined in the table are:

Symbol	Description
xxxFLG	A flag to be AOSed when the protocol's main routine, xxxPRC, is to be run.
xxxIPQ	Should be set to the address of a queue head where packets received for the protocol will be placed. The GETBLK routine can be used to obtain the storage; and the queue head should be initialized using the INITQ routine. Use the DQ routine to remove packets from the queue; note that DQ should be called while NOINT if the caller is not the internet fork.
"xxxON	Indicates whether or not the protocol should be initialized and run. If B35 is one, the protocol is initialized when the system restarts. If the B18 is one, the protocol should be turned on. B0 indicates whether or not the protocol is turned on (see xxxINI below).
xxxSID	Used for a protocol unique counter, such as the IP segment id.
xxxTIM	Contains the next scheduled time (TODCLK) the main processing routine, xxxPRC, should be run.

The seven routines which must be defined in the protocol module are described below. Note that they should never DISMS. Doing so will hang all network operations.

Symbol	Description
xxxINI	The protocol's initialization routine, called in the context of the internet fork at initialization if B35 is set in xxxON. It may use B19-34 of xxxON as flags to indicate what it has completed, and should leave B19-35 zero when it is all done. 1B18 means that the protocol should also be turned on when the system is started. It should be cleared and B0 set to indicate that the protocol is ON. If B18 is not set, B0 should be left zero to indicate that the protocol is OFF. If the protocol wants a separate fork, the routine MNTFKI should be used to create it.
xxxPRC	The main processing routine for the protocol; it is called in internet context. First, it should zero the run request flag (xxxFLG), process any packets in the input queue (pointed to by xxxIPQ), and anything else required to implement the protocol. It computes the next time it should be run (TODCLK units) and places the result in xxxTIM before returning.
xxxCHK	A routine which compares the time in T1 to the time in xxxTIM and sets T1 to the minimum. It is used to find the next scheduled run time for the internet fork.
xxxICM	A routine called when an ICMP packet is received for the protocol. PKT contains an extended pointer to the datagram. The routine should RETPKT the datagram before returning.
xxxKJB	A routine called when job JOBNO is being logged out, so that any protocol related resources can be released.

xxxKFK A routine called when fork FORKX is being killed, so that any protocol related resources can be released.

xxxCZF A routine called (from CLZFF%) to close files given the job-relative fork handle (e.g., a FORKN) in T1.

The current structure of packets given to IP (at SNDGAT) is restricted; they must already have an IP header and it must begin at offset PKTELI in the packet buffer. The use of this method is historical (the more general technique would be to pass a packet without an IP header and a parameter block with all of the IP parameters specified by the higher level protocol).

13. Adding Network Device Drivers and Network Protocols

Each distinct network interface requires code to:

- o drive the hardware (accessed via the Hardware Vector),
- o support any required network protocol (accessed via the Protocol Vector), including create and process any local leaders, and
- o an NCT to hold variables and configuration information.

There is one NCT for each (local host address, device driver) pair. An NCT consists of two logical areas. The first, of length VNCTSZ, is associated with a local host address (and indirectly with a protocol suite). The second is associated with the device driver, and is only present in Physical NCTs; it is sub-divided into a region, of length NCTBAS, common to all devices, followed by a region unique to a specific device.

Addition of a network protocol requires:

1) Adding an NL.xxx definition to ANAUNV.MAC for the network protocol.
2) Adding an entry to the TYPNAM keyword table in STG which associates a network protocol name string with a routine in MNETDV, that which sets the NTHDRL, NTPSIZ, NTPSTI, NTTYP, and NTPVEC entries in the NCT.
3) Write a network protocol module which implements the eight routines of the Protocol Vector.

Addition of a device driver requires:

1) Adding an NH.xxx definition to ANAUNV.MAC for the hardware device.
2) Adding an entry to the INTNAM keyword table in STG which associates the hardware device name string with a routine (e.g., xxxNCT) to initialize the hardware portion of the NCT, including the NTDEV, NTHVEC, NTORAT, NTPSTS, and interrupt entries.

3) Write a device driver module which implements the six routines of the Hardware Vector.

13.1. NCT Creation and Initialization

The ADRINI routine is called at system startup time (from MNTINI) and by the .IPNIF function of IPOPR% to create and initialize NCTs for each of the network interfaces specified in the SYSTEM:SITE-ADDRESS.TXT file.

When the hardware device name string keyword and device codes (if any) have been read, the INTNAM keyword table in STG.MAC is examined to find the address of the NCT initialization vector (e.g., xxxNCT, and implicitly an NH.xxx). The ININCT routine creates an NCT, initializes any constants specified in the initialization vector, then calls the device dependent routine for any additional processing which might be required.

The initialization vector begins with a header word, a region of (offset into NCT, value) pairs, followed by a code segment to initialize any variables which are not compile time constants (e.g., IO instructions which are functions of the device codes). The format of the vector is:

Field	Contents
IVLEN	Number of words required for the NCT. It is NCTBAS plus the number of words of storage unique to the network device driver. Offsets into the NCT of the unique variables should begin at NCTBAS.
IVINI	Number of words, excluding this word, containing initialization pairs.

Each of the initialization pair words contain:

B0-17 Offset into NCT of word to be initialized.
B18-35 Address of a word containing the initial value.

After the initialization pairs have been processed, the code segment is called with:

T1: Number of device codes following the hardware device name string keyword.

T2: Address of a block containing the device codes.

P1: Address of the NCT.

The routine should return+1 if it fails and skip return if it succeeds. The success return does not imply that the interface is useable; only that the NCT has been initialized.

The following table lists the currently defined vectors and their respective keywords.

ALTNCT ALTO NCT initialization vector, dummy in STG.MAC.

ANXNCT AN20 NCT initialization vector, in IMPANX.MAC.

IMPNCT IMP10 NCT initialization vector, in IMPBBN.MAC.

NINCT IPNI NCT initialization vector, dummy in STG.MAC.

NFENCT NETWORK-DTE NCT initialization vector, in NFEPHY.MAC.

VIRNCT VIRTUAL NCT initialization vector, in MNETDV.MAC.

After the initialization routine successfully returns any remaining keywords are parsed, their processing routines identified using the TYPNAM keyword table in STG.MAC, and called. Included among the keywords should be:

- the network protocol name string,
- the protocol suite name string, and
- the local host address for the NCT.

13.2. Common NCT Entries

The names and descriptions of the entries in the common area of the NCT are given in the following table. A few elements defined as structures are:

Element	Description
NTPRO	This field should contain the protocol suite code NP.xxx. It is usually specified by one of the address keywords in TYPNAM.
NTTYP	This field should contain the network protocol code NL.xxx. It is usually specified by one of the network keywords in TYPNAM.
NTDEV	This field should contain the hardware interface code NH.xxx. It is usually specified by an NCT initialization pair (or code) specified by the hardware interface keyword in INTNAM.
NTHSH	This hash code identifying the NCT is automatically initialized; some network protocols use it to identify the NCT (the number is small integer).
NTPRIO	This field should contain the interface priority; it is usually specified by the PRIORITY keyword.

Full word elements are:

Offset	Description
NTPVEC	The extended address of the Protocol Vector.
NTHVEC	The extended address of the Hardware Vector.

NTPHY Zero in Physical NCTs. In Virtual NCTs, automatically set to the address of the associated Physical NCT.

NTLADR The protocol suite address associated with the NCT is parsed by the routine identified by the address type keyword. See ADIPP (in MNETDV.MAC) for an example. Note that the NA%PRO field should be set to NP.xxx to identify the protocol suite.

NTNET The network number associated with the NTLADR address. It can be obtained from the address by the PRONET routine.

NTSBNM The subnetwork mask to be used if subnet routing is desired.

NTNLHM The logical host mask for the network, if it exists, is usually specified using the LOGICAL-HOST-MASK keyword. The NTPRO field can be used to identify the parsing routine used by NTLADR.

NTNLHB This field is automatically specified from NTNLHM.

NTHDRL This offset contains the difference between MAXLDR and the number of words of network leader required by the network, i.e., the number of unused words.

NTPSIZ The right half of this offset contains the maximum number of 8-bit bytes that a packet to be sent using this interface can hold, exclusive of the local leader. The left half, if non-zero, is used to indicate a restricted size, e.g., more efficient than the maximum size permitted. It is usually set using the PACKET-SIZE: keyword in the SITE-ADDRESS.TXT file.

NTORAT An estimated network transmission rate, in microseconds per byte. This constant is used to compute the estimated length of the output queue for flow control purposes. It defaults to 75. It is usually set using the OUTPUT-RATE: keyword in the SITE-ADDRESS.TXT file. NTOFCT is an associated variable which is currently unused.

NTOMSC Estimated output queue length, in milliseconds; used for flow control purposes. It is maintained by the multinet routines using the NTORAT parameter.

NTPSTI Negative length and address of network address monitoring block, or zero if the interface is not to be monitored. It is set using the MONITOR: keyword in the SITE-ADDRESS.TXT file.

NTPSTS Negative length and address of additional network device monitoring block, or zero if the device is not to be monitored. It is set using the MONITOR: keyword in the SITE-ADDRESS.TXT file.

NETON This variable contains the desired status of the hardware interface; 0 is off, -1 is on, and <0,,-1> is cycle. It should be set to -1 to turn the network interface on when the system restarts. The IPOPR% functions .IPSNT and .IPRNT can be used to set or read the state.

NTRDY This variable contains the actual state of the hardware interface; zero is down and 1B0 set is up. Other values are used by the NTRSRT and NTHKIL routines of the Hardware Vector. It should initially be left zero.

NTORDY This variable contains the actual state of the host address associated with the NCT; zero is down, 1B0 set is up. Other values are used by the NTPINI and NTPKIL routines of the Protocol Vector. It should initially be left zero. When zero, output to the network via the NCT is not allowed. (Input via the NCT is also discarded if the 1B0 is not set in the flag word pointed to by the PROON table entry identified by NP.xxx (from NTPRO).)

NTSTCH This variable should be AOSed whenever the status of the NCT changes, e.g., the hardware interface goes on or off, or the local address is enabled or disabled. When non-zero, the multinet background routine calls the LOGSTC routine to log a message on the CTY.

NTIUPT
NTIDNT The internal date and time that the hardware (or address) associated with the NCT was internally enabled or disabled.

NTXUPP
NTXDNT The internal date and time that the hardware (or address) associated with the NCT was externally enabled or disabled.

NTDCLK A (TODCLK) timer used to timeout attempts to disable the host address or network associated with the NCT.

NTHDWN
NTHDWI The protocol suite dependent (NA%PRO) address of a host on the network which has gone down, and any associated status information which might be available. When non-zero, the information is used to automatically update the host tables, and is zeroed when the data has been processed. Note that this information is not available for all types of networks.

NTERRF This flag is set negative when a (temporary) error is detected by the hardware (usually by the NTSCHK routine from the Hardware Vector). Any packet sent or received when an error is detected is assumed lost.

NTHOBO and NTHOBI
NTLOBO and NTLOBI
The heads and tails of a list of packets to be sent using high and low (standard) priority. The chain uses the NBQUE word in the packet buffer; NBQUE in the last packet in the list is zero. The NTHSND and NTLSND routines are used to add packets to the lists. The MNTUNQ routine is used to remove the next highest priority packet.

NTINRS When non-zero, the address of a routine to be called to restart/continue input of a packet when reception was previously delayed (e.g., have a packet but no input buffer).

NTIB Input lock/packet pointer. When Zero the NTISRT routine is called to begin reception of the next packet. To make sure that no errors were detected, the NTSCHK routine from the Hardware Vector should be called after each packet is received.

NTOB Output lock/packet pointer. When zero, the NTOSRT routine is called to begin transmission of the next packet. To make sure that no errors were detected, the NTSCHK routine from the Hardware Vector should be called after each packet is received.

Note: The NTIB and NTOB entries are used to control input and output operations, respectively. When zero, there is no operation in progress. Testing for zero (driver not active), the PI should be OFF. When found to be zero, the entry is set to -1 (i.e., 1B0) to indicate driver active. When a standard form packet buffer is associated with the operation the entry is changed to the extended address of the packet buffer. Drivers which perform operations with packets NOT in a standard buffer (e.g., control messages associated with the network protocol, etc.) may place the IPDV%R+extended address of the local leader into the entry.

NTTOUT Two words, each is a time (TODCLK) that an output (input) operation that is in progress should finish. Zero if no output (input) is in progress. The multinet background fork uses it to detect hung output (input) to a network.

If the operation is hung, the driver is automatically restarted (by calling NTRSRT via the Hardware Vector).

NTGEN A block of 6 words for general use by the hardware interface code. Offsets for any additional storage required should be defined in the device driver module and begin at offset NCTBAS (see IVLEN, above).

13.3. Standard Vectored Interrupt Support

The TOPS-20 device codes for these devices are usually specified in the SYSTEM:SITE-ADDRESS.TXT file immediately following the interface keyword. They are passed to the NCT initialization vector routine in the block pointed to by AC2. Standard support for hardware devices which use vectored interrupts consists of a state and stack saving and restoring routine. Note that it is assumed that the network interfaces all use PI level 6; if one doesn't, it will have to duplicate the saving and restoring routines (LV6SAV) and guarantee that the stack it sets up has sufficient space. Each vectored interrupt (e.g., input and output) uses a block of eight words in the NCT. The structure of the words in the NCT is described below.

NCT Offset	Contents	Comments
NTxINT	XPCW <address of NTxIPC>	The interrupt address given to the hardware device.
NTxJSR	JSR LV6SAV	Entry to state save routine; the ACs are saved and a stack is provided.
NTxDSP	Interrupt routine address	After processing the interrupt, the routine updates NTxDSP with the extended address of the routine to process the next interrupt, and RETs to dismiss the interrupt.

NTxNCT NCT address

NTxIPC Four word interrupt flags and PC block.
The first two words hold the flags and PC at the time of the interrupt. The second two words contain the flags and extended address of the NTxJSR offset.

13.4. Protocol Vector Routines

The interface to the network protocol layer is via a Protocol Vector which has eight entries. In each case, the routine is called with VNCT (i.e., P1) containing the address of the (virtual) NCT. The vector names and descriptions of the associated routines are given below.

Vector Name	Function

NTPINI This vector position contains the global address of a routine called to enable the host address associated with the NCT. It is called when:

- the networks are first turned on,
- each time that a failed interface begins to function, and
- each time the address is enabled after having been off (see IPOPR% function .IPSNT).

When the host address for the NCT becomes operational, the routine should set B0 of NTORDY and skip return. Until such time (e.g., the device driver is not yet functional as B0 of NETON is still zero), the routine should simply return without setting B0 in NTORDY.

NTLLDR This vector position contains the global address of a routine called to build a local leader for a packet. When it is called:

T1: Packet's (first hop) standard format destination address.
T2: Extended address of the packet buffer.
T3: Protocol suite code, NP.xxx.

The routine builds the local leader in the area preceding the transport level leader (located by PNTLDR xx,PTPDT,(T2)), and sets PLNDT and PLNBZ to the local leader buffer address and data count (including PTPBZ). The routine must return an error code in T1:

- MNTX00 if everything is ok,
- NE%DRP+an error code if a fatal error was detected, or
- a warning code (such MNTX04).

NTIDUN This vector position contains the global address of a routine called at interrupt level when a packet is received from (one of) the hardware interfaces. In addition to performing any operations related to the network protocol, the routine should:

1) Call the NTSCHK routine of the (physical) NCT to make sure that no hardware errors were detected while the packet was being received.

2) Call the NTPRNG routine with T2 containing the flags IPDV%I, IPDV%E, and either the extended address of the packet buffer from NTIB, or, for packets not in standard buffers (e.g., control packets), the flag IPDV%R plus the extended address of the local leader. The IPDV%D flag should also be set if the packet should be discarded.

3) Histogram the type (and subtype) of packet received, if appropriate, into the MNMXR monitoring block located via NTPSTS using the INHSTI macro.

4) Find the (virtual) NCT corresponding to the local host address contained in the packet.

5) Exchange 0 and and the contents of NTIB (allowing reception of the next packet to begin).

6) Set the PLNDT pointer to the local leader and the PLNBZ length of the packet (including the local leader).

7) CALLRET MNTEIN with T1 containing the NP.xxx code (from NTPRO) (or NP.GEN if no NCT could be found in step 4), T2 containing the extended address of the packet, T3 containing the length of the local leader, and VNCT containing the (virtual) NCT address. Note MNTEIN executes MNTJRST NTISRT to restart input for the device.

NTODUN This vector position contains the global address of a routine called at interrupt level when the hardware completes transmission of a packet. The routine is called with T1 containing a trace code (usually PT%SLN if transmission was successful, or PT%KOL if unsuccessful). In addition to performing any operations related to the network protocol, the routine should:

1) Call the NTSCHK routine of the (physical) NCT to make sure that no hardware errors were detected while the packet was being sent (switching to the PT%KOL code).

2) CALLRET MNTODN with T1 containing the trace code (PT%SLN or PT%KOL), PNCT containing the physical NCT address, and NTOB containing the extended pointer to output packet buffer.

NTPERR This vector position contains the global address of a routine called when an error has been detected by the hardware. The routine should perform any operations necessary to get the network protocol operational.

NTPMNT This vector position contains the global address of a routine called periodically to perform any protocol related maintenance operations.

NTPKIL This vector position contains the global address of a routine called when the host address associated with the NCT is disabled. It should clear B0 of NTORDY to stop the protocol suites from placing more packets into the

output queue. If the network protocol requires some action to disable the address, they should be initiated after setting a timeout (TODCLK in NTDCLK). When the address has been disabled, the timeout should be cleared and a skip return taken.

NTPRMV This vector position contains the global address of a routine called just before the NCT is flushed. Any system resources which were acquired when the NCT was initialized should be returned. This function is only called if the IPOPR% function .IPNIF (or ADRINI is called from MDDT) is invoked to reprocess the SYSTEM:SITE-ADDRESS.TXT file.

13.5. Hardware Vector Routines

The interface to the device driver layer is via a Hardware Vector which has six entries. In each case, the routine is called with PNCT (i.e., P1) containing the address of the (physical) NCT. The vector names and descriptions of the associated routines follow.

Vector Name	Function
NTRSRT	This vector position is called when the networks are first turned on, each time that a failed interface begins functioning, and each time the device is enabled after having been off (see IPOPR% function .IPSNT). It contains the global address of a routine called from the multinet background fork (which may block for short periods of time if necessary) when the hardware device associated with the NCT is to be enabled. It should assume an unknown state of the NCT and begin by cleaning up where possible; clearing B0 of NTRDY, returning any NTIB or NTOB buffers, resetting any interrupt dispatches, etc. If the hardware is operational, it should set B0 of NTRDY and RETSKP. If the hardware is not functional, it should RET.
NTSCHK	This vector position contains the global address of a routine called each time a packet is sent or received over the interface to make sure that no errors have been detected (the packet is ok to process). It should, if possible, check that the device is still operational and should skip return if everything is ok, or simply return otherwise.
NTISRT	This vector position contains the global address of a routine called to begin reception of another packet from the network. The routine should: 1) Call the NTSCHK routine to make sure the hardware is still operational.

2) Turn off interrupts (PIOFF) while the NTIB variable is checked to see if input is in progress. If none is, NTIB should be set to -1 before reenabling interrupts (PION).

3) Set-up the hardware device (including NTIDSP to begin input interrupt).

4) When input actually begins, a timeout (TODCLK+reasonable time) should be set in NTTOUT+1. If input is hung, the driver is automatically restarted (by calling NTRSRT via the Hardware Vector).

5) The NTPRNG routine should be called with T2 containing IPDV%I, IPDV%B, and either the extended address of the packet buffer from NTIB, or, for packets not in standard buffers (e.g., control packets), the flag IPDV%R plus the extended address of the local leader.

   The address of any input packet buffer should be stored in NTIB after address has been obtained (some devices might not obtain an input buffer until part of the local leader has been read to determine the size required). The routine GETBUF should be used to obtain packet buffers (which are locked in memory). If GETBUF fails, a restart address may be placed into NTINRS and the MNTWAK routine called so more buffers will be created. Control then returns to the restart address to retry the GETBUF call and continue. The PLNQD and PLNDB variables in the packet buffer should be set to the TODCLK when the buffer was obtained and input began, respectively. PLNDT should be set to point to where the local leader will be placed in the packet buffer.

6) When a packet has been received (usually in response to an interrupt), the timeout NTTOUT+1 should cleared, the PLNDE variable should be set to the current value of TODCLK, PLNBZ should be set to the number of bytes received, and the resulting packet buffer should be passed to the network protocol level by MNTCALL NTIDUN. The NTIDUN routine will dispose of the packet

(zeroing NTIB) and eventually "MNTJRST NTISRT" to begin reception of the next packet before returning (the stack should left in the same state it was when the NTISRT routine was called).

NTOSRT This vector position contains the global address of the end of a routine to begin transmission of another packet. The multinet routine MNOSRT is the beginning of the routine. It verifies that the hardware device is on (B0 set in NTRDY) and disables interrupts while checking the NTOB variable to determine if output is already in progress. If output is not in progress, NTOB is set to -1 before reenabling interrupts and jumping to the NTOSRT code which performs the following operations:

1) Call the MNTUNQ routine which returns the address of the next packet to be sent in T1 (if simple first-in first-out behavior is acceptable). MNTUNQ skips if there is a packet available; its address should be placed into NTOB. If no packet is found, NTOB is zeroed and NTOSRT should return.

2) Call the NTPOSS routine with T2 containing IPDV%O, IPDV%B, and either the address of the packet to be sent, or, for packets not in standard buffers (e.g., control packets), the flag IPDV%R plus the extended address of the local leader. T1 should contain a message type code in the right half and a message sub-type code (or zero) in the left half, which will be histogramed into the MNMXS monitoring block located via NTPSTS using the INHSTI macro.

3) Before starting the hardware device, a timeout (TODCLK+reasonable time) should be set in NTTOUT. If output is hung, the driver is automatically restarted (by calling NTRSRT via the Hardware Vector).

4) Set the output interrupt dispatch address in NTODSP.

5) Start the hardware device.

After the final output interrupt has been processed,

6) Clear the timeout, NTTOUT, and call NTPRNG with T2 containing IPDV%O, IPDV%E, and either the address of the packet to be sent, or, for packets not in standard buffers (e.g., control packets), the flag IPDV%R plus the extended address of the local leader.

7) If the packet was not a control message, MNTCALL NTODUN with T1 containing the trace code PT%SLN and T2 containing the address of the packet which was sent. (Control messages should be traced differently, see IMPANX.MAC for an example.)

8) Clear the timeout (NTTOUT), the output in progress flag (NTOB), and JRST MNOSRT to begin processing the next packet to be sent.

NTHKIL This vector position contains the global address of a routine called when the device associated with the NCT is to be disabled. It skips return when the hardware has been disabled.

NTHRMV This vector position contains the global address of a routine called just before the NCT is flushed. Any system resources acquired when the NCT was initialized should be returned. This function is only called if the IPOPR% function .IPNIF is invoked (or ADRINI is called from MDDT) to reprocess the SYSTEM:SITE-ADDRESS.TXT.

14. Multinet Data Structures and Routines

The following list describes data structures maintained by the multinet layer:

Structure	Description
NCT	The NCT has been described. They are created and removed by the ADRINI routine which processes the SYSTEM:SITE-ADDRESS.TXT file.
NLHOST	A list of standard form local host addresses terminated by -1. Entries are added and removed by ADRINI; the LCLHST (and LOGHST) routines use the table to check whether an address is one of the local host's addresses.
ADRHSH	A hash table of local host addresses. Entries are added by ADRADR or HSHADR, removed by ADRINI, and referenced by HSHADR.
NCTTBL	Translates NCT (hash code) index to NCT address; B0 is set if the interface is down. Entries are added by HSHNET, removed by ADRINI, and referenced by HSHNET, NETLUK, NETCHK, and LCLNET.
NETHTB NETIFC NETGWY	These parallel hash tables are used to perform routing. They contain a standard form network number (key), the address of the NCT (used to send a packet to that net), and the gateway (or host) address to which the packet should be sent. Entries are added by NETHSH (which returns the table index), modified by routing updates (which have been received), and timed out.
HOSTNN HSTSTS HOSTPN	These parallel hash tables form the host status cache. They contain the standard form host address (key), host

status word, and, if known, the extended address of the corresponding HOSTN table entry for the primary host name. The HSTHSH routine returns the hash index into the tables and locates empty slots for new entries. Entries are added by HSTINI, or by observing traffic flow. HOSTPN will be removed when the Domain software is fully operational; HOSTNN and HSTSTS will probably remain.

HOSTN
HSTNAM
HSTNML These tables form the host name - address translation cache. A (2 word) entry in HOSTN exists for each host name string and contains extended addresses of entries in HSTNAM and HSTNML. HSTNAM contains the ASCIZ host name; HSTNML contains a -1 terminated list of standard form host addresses for the host. Entries are added by HSTINI. Note that these three tables will be removed when the Domain software is fully operational.

MNTFWT A table used when restricted forewarding is enabled. The first word contains a byte pointer, which, when adjusted by the NTHSH value of a receiving NCT, is used to ILDB a byte of bits whose 36-i th bit (where i is the NTHSH value of a desired forewarding NCT) is set if forewarding has been enabled. The second word in the table contains a count of the number of remaining words in the table; the remaining words comprise the data bytes.

The following lists describes several routines that maintain the data structures:

Routine	Description
MNTWAK	Is called by any fork which wants to run the multinet background process.
MNTBPT	Scheduler test for multinet background fork.
DISRE	A scheduler test which waits for the right half of the section zero variable to become zero.

PVNCT Called to locate the (first) NCT for a protocol suite. When called:

T1: Contains the protocol suite code (NP.xxx).

PNCT: Contains the physical NCT address. It preserves all ACs but returns with PNCT undated.

VNCT: Set to the NCT address of the first NCT (for the hardware device) matching the protocol suite. Successive NCTs, linked by NTLNK, may also be for the same suite; the last NCT (for the hardware device) is identified by either a zero NTLNK, or a zero NTPHY.

VNCTFN Called to perform a function on a physical NCT and all of its associated virtual NCTs. When called:

T1: Contains the address of the function to be called.

T2-T4: May be passed to the function, if desired.

PNCT: Contains the address of the physical NCT. If the function skipped for any of the NCTs, VNCTFN skip returns.

LV6SAV Called in response to a network hardware interrupt to save the current context. It passes control to the interrupt dispatch for the associated NCT, e.g., NTIDSP or NTODSP.

NTPOSS Called to record the message's Type and Sub-type before a beginning transmission of a packet or control message, which will be histogramed into the MNMXS monitoring block located via NTPSTS using the INHSTI macro. Then it joins the NTPRNG routine. When called:

T1: Contains the Type code in the right half and an optional Subtype code in the left half.

T2: Contains flags (see NTPRNG) and the extended pointer to the output buffer, or IPDV%R and the address of the local leader of a control message.

PNCT: Contains the physical NCT address.

NTPRNG Called after completing an input/output operation or before beginning reception of a packet or control message. It records statistics related to network hardware utilization. When called:

T2: Contains flags and the extended pointer to the packet buffer, or IPDV%R and the address of the local leader or control message.

PNCT: Contains the physical NCT address.

Other flags are:
IPDV%B Beginning operation.
IPDV%E Ending operation.
IPDV%I Input operation.
IPDV%O Output operation.
IPDV%D Message will be discarded at the network level (e.g., it is a control message).

(It returns T2 unchanged; all flags but IPDV%R have been cleared.)

MNTEIN Is called when the network protocol layer has completed processing a packet which has just been received. It is called at interrupt level. When called:

T1: Contains the protocol suite code (NP.xxx).

T2: Contains the extended pointer to the input buffer with PLNDT and PLNBZ set.

T3: Contains the number of bytes in the local leader.

VNCT: Contains the Virtual NCT address.

Then it executes MNTJRST NTISRT to begin reception of the next packet.

NTSNDI A specialized routine to create a local leader and send an IP packet. When called:

T1: Contains the first-hop destination host address on a directly connected network.

T2: Contains the extended address of the packet buffer with PTPDT and PTPBZ set.

T3: Containing the protocol suite code NP.xxx.

VNCT: Containing either <0,,NCT> or <LoopbackpNCT,,LogicalNCT>:

It returns with:

T1: Contains an error or warning code (NE%DRP set if packet was rejected).

T2: Contains the estimated output queue length in milliseconds, as computed for NTOMSC.

NTHSND
NTLSND Are called to place a packet in a driver's high (NTHOBO/I) or normal (NTLOBO/I) priority output queue. When called:

T1: Contains the destination network code.

T2: Contains the extended address of the packet buffer with PLNDT and PLNBZ set.

VNCT: Contains either <0,,NCT> or <LoopbackpNCT,,LogicalNCT>.

They returns with:

T1: Contains an error or warning code (NE%DRP set if packet was rejected).

T2: Contains the estimated output queue length in milliseconds, as computed for NTOMSC.

NTQPKT Called to place a packet into a driver's output queue. When called:

T1: Contains the offset into the NCT of the queue (i.e., NTHOBO or NTLOBO).

T2: Contains the extended address of the packet buffer with PLNDT and PLNBZ set.

VNCT: Contains either <0,,NCT> or <LoopbackpNCT,,LogicalNCT>.

They return with:

T1: Containing an error or warning code (NE%DRP set if packet was rejected).

T2: Containing the estimated output queue length in milliseconds, as computed for NTOMSC.

MNOSRT Called, possibly at interrupt level, to begin transmission of a packet. When called:

VNCT: Contains the address of an NCT which might have output waiting to be sent.

MNTUNQ Called by NTOSRT routines to dequeue the next packet to be sent from a driver's output queues. When called:

PNCT: Contains the physical NCT address.

It skip returns if a packet was found in one of the queues, containing with:

T1: The extended address of the packet buffer, if successful. If unsuccessful T1 contains zero.

MNTODN The network protocol layer jumps here, while at interrupt level, after completion of an output operation. When called:

T1: Contains a trace code (PT%SLN if successful, or PT%KOL if unsuccessful).

PNCT: Contains the physical NCT address with

NTOB: containing the extended address of the packet buffer, with PLNDT and PLNBZ set.

Virtual NCTs are used to associate multiple addresses with a single physical network interface. They are shorter than physical NCTs as they do not have any variables associated with the hardware device. The NTPHY offset in a virtual NCT contains the NCT address of the physical NCT which controls the network interface. The NCT initialization vector for virtual NCTs is:

VIRNCT The NCT initialization vector for a Virtual NCT.

The loopback device is used to send datagrams to the local host. Its use is required for those networks which cannot send datagrams to themselves and also permits intra-host communication even if all of the physical interfaces are down (or don't exist). The NCT initialization vector for the loopback interface is:

LUPNCI The initialization vector for the loopback NCT.

The loopback device has a local leader and output routine (that copies packets that are to be sent); it has no network protocol.

LUPLDR A local leader routine used to maintain consistency with other protocols.

LUPOUT The NTOSRT function for the loopback device. It:

- obtains a packet buffer from GETBUF,
- copies the output packet into it,
- passes the resulting packet upwards via the NTIDUN vector,
- passes the original output packet back via the NTODUN vector, and
- executes CALLRET MNOSRT to restart output for the "loopback device".

HSTHSH The hashing function which hashes;

T1: containing the standard form host address.

It returns with:

T2: the <section,,index> value required to access the HOSTNN, HSTSTS, and HOSTPN host status cache.

It skips if the entry was present in the tables. If it does not skip the entry did not exist, but would be in the position specified (if the tables are full, -1 is returned).

.ATNVT Implements the ATNVT% JSYS. It dispatches to one of the following processing routines:

TATNVT for AN%TCP calls.

TVTJFN for TCP: connections.

CATNVT for CHAOS: connections.

PATNVT for PUP: connections.

.CVHST Obsolete; implemented the CVHST% JSYS.

.GTHST Implements the GTHST% JSYS.

.GTDOM Implements the GTDOM% JSYS. Note that all of GTDOM%s functions have been integrated into the GTHST% JSYS, which should be used to maintain compatibility with older programs.

HSTLUK Lookup an ASCII host name in the (old) host tables; it should be superseded by the domain database when the latter is fully implemented.

T1: contains the length of the string to be matched.

T2: contains a one word global byte pointer to the string.

T3: contains flags from GTHST% call in the left half; GH%ANY, GH%GWY, and GH%NET limit which matches will be accepted.

It returns normally if the name was not found, or skip returns if the name was found with:

T1: containing the updated pointer.

T3: containing the best address for that host.

T4: containing the host's status.

FHPADR Is called to find which host address, in a given list, is on the highest priority interface. The interface does not have to be enabled. When called:

T1: contains the address of a list of standard format host addresses terminated by a -1.

It returns with:

T1: containing the highest priority address, or the first host address in the list, if none are on connected nets.

LCLNET Called by IP routines to check if a host address is on a connected network. It skip returns if an interface on the network exists. When called:

T1: contains a standard format host address.

NETCHK Is called to check if an interface on a given net is up; it skip returns if an interface on the net is enabled. When called:

T1: contains a standard form network number.

NETLUK Is called to find an interface on a given directly connected network. When called:

T1: contains a standard format network number.

It returns with:

T1: 0 if there is no interface on the net.

NCT if an interface exists and is up or

1B0+NCT if the interface exists but is down.

HSHNET A work routine for finding an NCT address. When called:

T1: contains a standard format network number.

T2: contains the address of table of two functions. The first function is executed if no NCT is found. The second function is executed if an NCT is found (it does not have to be enabled).

MSK2BP Called to convert a contiguous bit mask into a prototype byte pointer. When called:

T1: contain a mask

It returns with:

T1: containing a prototype byte pointer for the field.

NETNCT Called to translate a standard format network number to an NCT address; the NCT does not have to be enabled. It preserves T1 through T4. When called:

T1: contains a standard format network number.

It returns normally if there is no NCT on the given network. It skip returns with:

VNCT: containing an NCT address on the given network.

LCLHST Called to check if a given (non-zero) address is one of the local host's addresses. The address does not have to be enabled. It preserves all registers. When called:

T1: contains a host address.

It skip returns if the address is a local address; returns normally if the address is not a local address.

HSHADR A work routine to determine if a given address is one of the local host's addresses. The address does not have to be enabled. When called:

T1: contains a standard format host address.

T2: contains the address of table of two functions. The first function is executed if the address is not a local address. The second function is executed if the address is local (it does not have to be enabled).

LCKSTR Called to lock a region of memory in core. When called:

T1: contains the extended address of the start of the region.

T2: contains the extended address of the end of the region.

ETHRAH Called to translates physical addresses containing 37 to 72 bits into the "rest" part of a standard format host address. The one word standard format address may then be used to simplify subsequent host address comparisons. When called:

T1: contains the first 32 physical bits.

T2: contains the second 32 physical bits.

It returns with:

T1: containing the extended address of a block containing:

+0 a flag word; 1B1 means that the block is currently in use.

+1 the first 32 address bits from T1.

+2 the second 32 address bits from T2.

+3 available for software use.

The standard format host address is formed by inserting the appropriate NP.xxx protocol code into the NA%PRO field of T1.

MNTFKI A system/OS dependent routine used to initialize a new fork, which runs under multinet, in monitor context, in section 1, with high priority. When called:

Q2: contains the in-section address for an illegal interrupt routine, or 0 if no address exists.

It returns the FORKX in T1.

MNTRED The routine called by the .IPRNT function of IPOPR% to read a network's status. When called:

T1: contains the standard form network number, or host address.

It returns in:

T1: MNTX00 if successful, an error code if unsuccessful.

T2: contains the actual status; the possible values are: -1 = on, 0,,-1 cycle, 0 = off.

MNTSET The routine called by the .IPSNT function of IPOPR% to set a network's status. When called:

T1: contains the standard form network number, or host address.

T2: contains the desired status; the possible values are: -1 = on, 0,,-1 cycle, 0 = off.

It returns in:

T1: MNTX00 if successful, an error code if unsuccessful.

MNTCZF The routine called from CLZFF% to close and release any network related files given the job-relative fork handle (e.g., a FORKN) in T1. After saving all of the registers, it calls the PROCZF routine for each of the protocol suites.

MNTKFK The routine called from KFORK% to release any network resources assigned to the fork specified by FORKX. After saving all the registers, it calls the PROKFK routine for each of the protocol suites.

MNTKJB The routine called from LGOUT% to release any network resources assigned to the job specified by JOBNO. After saving all the registers, calls the PROKJB routine for each of the protocol suites.

MNTCHK Called each time the multinet background loop runs to check the status of each of the NCTs. When called:

VNCT: contains the NCT address.

For each NCT, the protocol maintenance routine is called (via the NTPMNT dispatch in the Protocol Vector), and the status of the associated protocol and hardware (if any) is checked by calling the CHKHDW routine. If the NCT has had any unreported status changes (NTSTCH is non-zero), the LOGSTC routine is called to log them on the CTY. Finally, if the SCTLW indicates that the system is being shutdown, the physical networks are turned off (via the NTHKIL dispatch in the Hardware Vector).

The GCBUF routine is also called to garbage collect the network buffer pools.

CHKHDW Called to check the status of an NCT's hardware (if any). When called:

VNCT: contains the NCT address.

LOGSTC Called to log any unreported status changes for an NCT on the CTY. When called;

VNCT: contains the NCT address.

MNTCLQ Called to clear an interface's NTIB, NTOB, and output queues (NTHOBO/I & NTLOBO/I). The interface should be turned off or I/O page failures may result. When called:

PNCT: contains the NCT address.

FREBFL Is called to release an NBQUE linked list of packet buffers. When called:

T2: contains the extended address of first packet in the list.

PNCT: contains the physical NCT address (with the hardware OFF and NTOB cleared).

FREBUF releases individual packet buffers. When called:

T1: contains the extended address of packet buffer.

PNCT: contains the physical NCT address (with the hardware OFF and NTOB cleared).

NCTUP
NCTDWN These routines are called when the local address or hardware associated with an NCT is enabled or disabled. They update B0 in the NCTTBL table, and, if the protocol suite is ON (PROON), call the PRONUP or PRONDN routine to notify the protocol suite of the change. When called:

VNCT: contains the NCT address.

MNTHLT Called twice at system shutdown time, in system shutdown fork context, to tell all networks that the system is going down. First, it stops any ICMP pinging, initiates transmission of the last monitoring report, and calls DOMKIL to release any domain related resources. At the second call, the NCTs are all turned off and there is a

short delay so that the device drivers can stop the hardware. When called:

T1: .lt. 0 System is (starting to) shutdown (first call), or
.ge. 0 Reason for going down (last call).

T2: Indicates when the system will be back up, if known.

MNOUTD Converts a decimal number to an ASCII string in the user's address space. It is used by the PRONUM routines because of prvious context problems. When called

T1: Destination designator.

T2: Number to be converted to ASCII.

7: Instruction to be executed to move a byte to the caller's address space.

ADRINI Called at system startup (from MNTINI) and by the .IPNIF function of IPOPR% to process the SYSTEM:SITE-ADDRESS.TXT file. It uses the INTNAM, TYPNAM, SETTAB, and CMDNAM keyword tables (in STG) and their associated routines and calls ININCT to initialize an NCT. The tables NLHOST, ADRHSH, and NCTTBL also are initialized. See section 13.1, NCT Creation and Initialization, for a more complete description.

Network protocol names are associated with their NCT setup routine via the TYPNAM keyword table in STG. The following 4 routines setup parameters then call ADPVEC to set the associated NCT network protocol parameters.

AD1822 1822 protocol initialization routine, in 1822DV.MAC.

ADCHAN A prototype CHAOS network protocol processing routine, in MNETDV.MAC.

ADETHR A prototype Ethernet network protocol processing routine, in MNETDV.MAC.

ADNFE Network front end network protocol processing routine, in NFEPHY.MAC.

ADPVEC A common end for network protocol processing routines that accepts:

T2: Contains the address of the Protocol Vector (or zero if unsupported) for NTPVEC.

T3: Contains the local leader padding (for NTHDRL).

T4: Containing the network protocol code NL.xxx, (for NTTYP).

VNCT: Contains the address of the NCT being initialized.

Protocol suite names are also associated with their NCT setup routine via the TYPNAM keyword table in STG. The following 5 routines setup parameters then call ADLNPA to set the associated NCT network protocol parameters.

ADCHAP An initialization routine for the CHAOS protocol suite that supports CHAOS NVTs over IP; it doesn't call ADLNPA.

ADDNAP A prototype initialization routine for the DECNET protocol suite.

ADIPP An initialization routine for the DARPA protocol suite, i.e., INTERNET (or IP for backwards compatibility).

ADPUPP A prototype initialization routine for the PUP protocol suite.

ADXNSP A prototype initialization routine for the XNS protocol suite.

ADLNPA A common ending for the initialization routines which expects:

T1: containing the standard format network number (for NTNET).

T2: containing the protocol suite code NP.xxx (for NTPRO).

P2: containing the standard format network address (for NTLADR).

The remaining keywords processed by ADRINI from the interface lines of the SITE-ADDRESS.TXT file using the TYPNAM table in STG are:

ADIDKW The processing routine for the ID keyword.

ADLHM The processing routine for the LOGICAL-HOST-MASK keyword (for NTNLHM). It calls the appropriate PRONUM routine to parse a network address mask.

ADMON The processing routine for the MONITOR keyword (for NTPSTI and NTPSTS).

ADPSIZ The processing routine for the PACKET-SIZE keyword (for NTPSIZ).

ADPRIO The processing routine for the PRIORITY keyword (for NTPRIO).

ADSET The processing routine for the SET keyword.

The SET keyword associates configuration parameter keywords with a processing routine via the SETTAB table in STG. The following 4 routines are currently defined:

STRHTY Processes the 1822 HANDLING-TYPE keyword; it is in 1822DV.MAC.

STORAT Processes the OUTPUT-RATE keyword (for NTORAT); it is in MNETDV.MAC.

STRFNC Processes the 1822 RFNM-COUNTING keyword; it is in 1822DV.MAC.

STAR3 Processes the 1822 TYPE-3-MESSAGES keyword; it is in 1822DV.MAC.

Commands in the SITE-ADDRESS.TXT file are associated with a processing routine using the CMDNAM keyword table in STG. Routines for the 2 currently defined commands are:

ADURFC The processing routine for the UNRESTRICTED-FORWARDING command.

ADSETC The processing routine for the SET command. It uses the following 3 routines:

ADSRVL to read a value.

ADSSWT to set switch bits.

ADSSYM to set a parameter's value.

DOMINI Initializes the Domain Name system by finding a usable pair of domain database files (FLIP.DD and FLOP.DD), mapping them in, and reinitializing all locks.

DOMKIL Disables the Domain Name system, waits for any active resolver requests to complete, and unmaps the domain database.

HSTINI Called at system startup to process the SYSTEM:HOSTS.TXT file. It enters host data into the host cache tables, HOSTNN, HSTSTS, HOSTPN, HOSTN, HSTNAM, and HSTNML.

Both ADRINI and HSTINI use a common set of text parsing routines:

GTFIL Called to open the input text file. The JFN is stored in HTBJFN. The file should be CLOSF%ed using the JFN in HTBJFN.

GBOL Called to find the beginning of the next line and skipping over white space and comments. BOL is set to the byte number of the start of the line.

RDFLD Called to parse the next field into TMPBUF up to the next "," or ":". The terminator is saved in TERM.

SKPFLD Called to skip over the next field (up to a ":").

RDNCH Called to read the next character. When called:

T3: contains a desired character.

If the character in T3 matches the next character in the file, the routine skip returns. Otherwise, the character is put back so the next call will reread it. In each case:

T2: contains the character just read from the file.

RDNUM Called to parse a number. T3 contains the radix, T2 is set to the number parsed; the terminator is saved in TERM.

RDHNUM Called to parse an internet address in the [a.b.c.d] format. The host address is returned in T1. It should be phased out and the PRONUM routine used instead.

PERROT Called when an error is detected. It outputs the line containing the error and inserts a "^" where the error was detected using the PMARK routine.

14.1. Packet Free Storage

The multinet free storage package for (resident) packet buffers consists of six routines:

Routine	Description
BUFINI	Called at system startup to initialize the packet buffer variables.
GETBUF	Called to allocate a buffer from the pool. When called: T1: contains the total number of words required. T2: contains the protocol suite code (NP.xxx). It returns with: T1: containing the extended pointer to packet buffer, or 0 if no storage was available. The packet buffer has NBSIZ set to the number of words allocated. GETBUF consults the PROOVH table to find out how much protocol dependent overhead is required and zeros the Packet Private Local Variables area in the buffer.
GETPAG	Called when another page is needed.
STEALP	Called to obtain a free memory page and make it resident.
RETBUF	Called to return a buffer to the system with: T1: containing the extended packet buffer address.
GCBUF	Called periodically to collect excess (PULOPT-PULCNT) buffers, using the GC1B routine.
GC1B	Merges excess buffers into pages possible; calls RETRNP.

RETRNP Return excess memory pages to the system.

LINK
UNLINK Used to link and unlink packet buffers from the lists maintained by the free storage package.

BUFCHK Called to determine if an extended address is within the memory region defined by PULBAS and PULTOP. The routine skip returns if the address is in the region and returns normally if it is not.

T1: An extended address.

The following constants and variables are associated with the packet buffer free storage routines:

PULBAS
PULTOP Constants containing the (page aligned) extended address of the base of the packet buffer area, and the top address (sec,,xxx777).

PULMAP A bitmap containing a 1 for each available page. B0 corresponds to the page containing PULBAS.

The following parallel tables are indexed by the free storage pool number:

PULSIZ The size of buffers in the pool.

PULLST Address of a list of free buffers.

PULCNT Count of free buffers in PULLST.

PULMIN Minimum number of free buffers to have available.

PULOPT Optimal number of free buffers to have available.

PULMSK Mask used to locate the buffer's buddy.

15. Network Tracing and Monitoring

The network code has extensive tracing facilities which can be enabled for selected protocol levels, selected classes of events, selected connections, or a specified job. Each point where a sample might be collected is assigned a trace code of the form PT%xxx. They are defined in ANAUNV.MAC within the PT%XXX macro. The trace code specifies the data to be sampled:

- a packet (at selected protocol layers)
- the registers
- a TCP connection block
- a TCP buffer.

When a sample is taken, the trace code, TODCLK, and the specified data are written into a region of memory. When the region becomes half full, the data are appended to a previously existing file, PS:<OPERATOR>MNTRAC.BIN (the name may be changed by changing the ASCIZ constant at MNTRFN). If the tracing file cannot be accessed, samples are discarded.

The MNTRAC data structure controls the tracing facilities. The MNTRAC data is changed using MDDT (the ability to use MDDT is required as the data from the trace may contain sensitive information).

Variable	Description
MNTRAC	When non-zero, tracing is enabled. Bits are set specifying the events to be traced. Common settings are: 1B0 Traces a TCP connection 4000 (PT%%GW) Samples each packet sent and received 2000 (PT%%NT) Traces events at the network protocol or device driver level. Consult the PT%%xx definitions in ANAUNV.MAC for other settings.

+1 The foreign host address for which packet samples are to be collected. (Left justified for protocol suite samples. Right justified for network interface level samples.)

+2 A right justified port number.

MNPTJ A field containing a job number whose TCP connections are to be traced.

MNPTO A field containing the offset of the first word in a TCB to be sampled for a TCB sample; the default is 0.

MNPTN A field containing the number of words of TCB information to be collected for a TCB sample; the default is the whole TCB.

The following routines, called from MDDT, are associated with tracing:

Routine	Description
MNTPIN	Allocates a large region of memory to collect samples before they are written to a file. Should be called once only before tracing is started (it has not been called if MNTRAC+6 is not in the packet buffer region between c(PULBAS) and c(PULTOP); calling it twice looses the previous memory buffer).
MNTPWR	Automatically called whenever the in memory tracing buffer becomes half full. It should also be called from MDDT after a tracing session to write out any samples remaining in memory.

15.1. How to Collect Data

To begin tracing:

- NUL: should be copied to PS:<OPERATOR>MNTRAC.BIN
- MDDT entered
- The MNTPIN routine called (once per system reload)
- Appropriate bits set in MNTRAC (see below).

To stop tracing:

- MNTRAC should be set to zero
- The MNTPWR routine called to write out any samples remaining in memory.

Tracing may be started before the system comes up (to observe what transpires with the networks e.g., control messages) by setting MNTRAC from EDDT (MNTPIN is automatically called). The PS:<OPERATOR>MNTRAC.BIN should already exist. Tracing may also be enabled when the system is shut down; MNTHLT will disable tracing and request that the memory buffer be written just before it returns.

15.2. Tracing Example

An annotated example of collecting a packet trace and using the PKTPRN utility to list the result follows.

15.2.1. Collecting Tracing Samples

The packet trace is collected by:

```
$cop nul: z:MNTRAC.BIN.1                ; Create the tracing file
 NUL: => <OPERATOR>MNTRAC.BIN.1 [OK]
$^Em                                                ; Enter MDDT
MDDT
mntrac 6[   5,,250004                ; Already have memory buffer
mntrac[   0   4000             ; Collect samples at gateway level
xxxx                                  ; Type something for trace
U^G
mntrac[   4000   0                     ; Stop collecting samples
call mntpwr$x                        ; Write out the memory buffer
<>
1,,mretn$g                                          ; Leave MDDT
$
```

15.2.2. Using PKTPRN to List Tracing Samples

The binary trace file is formatted using the PKTPRN utility program. PKTPRN has extensive facilities for filtering out unwanted samples and specifying the packet fields, etc., to be listed. (Note: the example has been edited to fit the page.)

```
$pktprn                                       ; Formatting program
TCP Packet Trace Listing Program   V10.2(164)
Parameters:? one of the following:             ; Display commands
 EXIT       FILTER    FORMAT    FUNCTION    GO      HELP    INPUT
 OUTPUT     PUSH      SHOW      SKIP        TAKE    WAIT

Parameters:filter                               ; Display filters
Filter term: ? one of the following:
 BUFFER-SAMPLES  CLEAR           CODES             DONE
 ENTER           HELP            HOSTS-PORTS       INTERFACE
 LINK-PROTOCOL   NETWORK-DRIVER  NOT               PROTOCOL-SUITE
 PROTOCOLS       PSUITE          REGISTER-SAMPLES  SHOW
 TCB-SAMPLES     TRACE-CODES
Filter term: done                    ; Finished specifying filter

Parameters:fun (to be applied to data is) ? one of the following:
 DRIVER-UTILIZATION        EXTRACT        HELP       LIST-PACKETS
```

```
Parameters:formAT ? one of the following:    ; Formatting options
 ASCII                 B-FIELDS              BINARY
 BUFFER-FIELDS         CORE-INPUT-FILE       DATA-OCTETS
 FILE-POSITION         GARBAGE-OCTETS        HELP
 I-FIELDS              I-LEADERS             ICMP-FIELDS
 INTERNAL-FIELDS       INTERNAL-LEADERS      IP-FIELDS
 L-FIELDS              L-LEADERS             LOCAL-FIELDS
 LOCAL-LEADERS         LONG-FORMAT           MAXIMUM-DATA-LENGTH
 NO                    OMITTED-COUNT         PAGINATE-LISTING
 PARITY-BIT            SHORT-FORMAT          TCB-FIELDS
 TCP-FIELDS

Parameters:formAT tcp-FIELDS ? one of the following:
 ACK              ACK-SEQUENCE     ALL          CHECKSUM
 CONTROL          DATA-OFFSET      DEFAULT      DESTINATION-PORT
 END-SEQUENCE     FIN              NO           PUSH
 RESET            SEND-RIGHT       SEQUENCE     SOURCE-PORT
 SYN              URG-POINTER      URGENT       WINDOW
  or confirm with carriage return

Parameters:i z:MNTRAC.BIN.1                     ; Format samples
Parameters:o MNTRAC.LST.1                         ; collected in
Parameters:fiLTER (expressions are)                  ; the example
Filter term: clear all                          ; List all packets
Filter term: done
Parameters:show                             ; Show current settings
 Input PS:<OPERATOR>MNTRAC.BIN.1
Output PS:<CLYNN>MNTRAC.LST.1
Function LIST-PACKETS.
Format Long
Filter All packets will be processed.
```

```
Parameters:go                                    ; Begin processing
  [BBNF.ARPA]PS:<CLYNN>MNTRAC.LST.1          ; Summary information
Dump of PS:<OPERATOR>MNTRAC.BIN.1,10, 1-Apr-85 21:08:07

Protocol         Recv    Sent
TCP-PACKETS        19      40
ICMP-PACKETS        0       0
BBN-RCC-MONI        0       2

TCP Port         Recv    Sent
TELNET             19      40

   Trace         Recv    Sent
RGI     5257        0      21
QLN     5230        0      21
RGW     5130       19       0
  61 samples read, 61 headers, 40 samples processed.

Parameters:exit
$
```

A few of the formatted samples from the file are shown below. Samples 1 and 2 show the echo of the carriage return which turned the tracing on; sample 3 is the acknowledgement. Sample 4 is the reception of the first "x" (it is shown in octal because the "FORMAT NO PARITY" command was not specified; samples 5 and 6 are it's acknowledgement and echo. Samples 57 and 58 are associated with a monitoring protocol running on the system. See PKTPRN.INFO for a detailed explanation of the fields and formats.

```
$ty mntRAC.LST.1
  [BBNF.ARPA]PS:<CLYNN>MNTRAC.LST.1
Dump of PS:<OPERATOR>MNTRAC.BIN.1,10, 1-Apr-85 21:08:07

 Input PS:<OPERATOR>MNTRAC.BIN.1
Output PS:<CLYNN>MNTRAC.LST.1
Function LIST-PACKETS.
Format Long
Filter All packets will be processed.

+      0.   165028352., Sample 1.,5220401 Note: PT%RGI
+      0.   165028352., Sample 2.,5220401 Note: PT%QLN
IP: Ln=42 ID=2715 TTL=59 6TCP S=192.1.2.66 0.23 D=192.1.2.41 0.14
TCP: Seq=4041298226 Ack=113431392  Ack  Psh    Wnd=2224 Urgp=0
  TCP Data: <15 12>

+     99.   165028451., Sample 3.,5220301 Note: PT%RGW
IP: Ln=40 ID=1 TTL=14 6TCP S=192.1.2.41 0.14 D=192.1.2.66 0.23
TCP: Seq=113431392 Ack=4041298228   Ack     Wnd=456 Urgp=0

+   2116.   165030567., Sample 4.,5220301 Note: PT%RGW
IP: Ln=41 ID=1 TTL=14 6TCP S=192.1.2.41 0.14 D=192.1.2.66 0.23
TCP: Seq=113431392 Ack=4041298228   Ack  Psh    Wnd=456 Urgp=0
  TCP Data: 370

+      6.   165030573., Sample 5.,5220401 Note: PT%RGI
+      0.   165030573., Sample 6.,5220401 Note: PT%QLN
IP: Ln=41 ID=2716 TTL=59 6TCP S=192.1.2.66 0.23 D=192.1.2.41 0.14
TCP: Seq=4041298228 Ack=113431393   Ack  Psh    Wnd=2223 Urgp=0
  TCP Data: x
...
+     97.   165044728., Sample 57.,5220301 Note: PT%RGI
+      0.   165044728., Sample 58.,5220301 Note: PT%QLN
IP: Ln=52 ID=0 F=-0 TTL=9 aSMS S=192.1.2.66 D=192.1.2.67
RCC Data: 0  0  0  3  0  0  264  115  0  0  133  16  30  0  0  5
```

15.3. Adding Trace Points

To add a new tracing point, a trace code must be defined (equivalent points in different network level or hardware level code currently use the same trace code). After selecting a unique three letter mnemonic and 6-bit code, another line should be added to the PT%XXX macro. Each line contains a PT macro which defines a trace code. Its arguments are the three letter mnemonic, the 6-bit code, and a list of PT%%xx codes which describe properties of the point to be traced. The software assumes that three letter mnemonic and right half of the resulting value are unique.

A tracing routine is conditionally called at points in the code where samples are to be collected. The routine called depends on the program context. B0 of AC0 is used within the protocol suites to indicate that events associated with a connection are being traced (center example below). At lower levels, the "trace this packet" flag, PTRAC, in the packet buffer is tested (left example).

```
PKT has pointer to      AC0 has trace
standard packet         connection flag         No flags in AC0

LOAD CX,PTRAC(PKT)
MOVX T1,PT%xxx          MOVX T1,PT%xxx
SKIPN CX                TXNN FR,1B0             MOVX T1,PT%xxx
 TDNE T1,MNTRAC          TDNE T1,MNTRAC         TDNE T1,MNTRAC
  CALL PRNyyy             CALL PRNyyy             CALL PRNyyy
```

The currently defined tracing routines are:

MNTRAP Called to log events (by dumping the registers). It has not been fully implemented. It preserves all the registers. When called:

CX: Contains a trap code (currently same as a trace code).

ACs: Contain information to allow the event to be analyzed.

PRNPKH Called to trace events at the network protocol or device driver level; possibly in interrupt context. If space for the sample is available in the in-core buffer, it is reserved and the data inserted. If no space is available (the process level code to write out the data has not had a chance to run), the counter MNTPTD is incremented. (The position and count of lost samples will be indicated in the listing produced by the PKTPRN utility.) All registers are preserved. When called:

T1: contains the trace code PT%xxx, identifying the call

T2: 0 If a packet is not being traced
0B0+Extended pointer to top of packet buffer to be traced.
1B0+Extended pointer to local leader (e.g., a control message)

T3: Contains formatted DPRO,DTYP,DDEV fields for control message.

VNCT: contains the virtual NCT address.

PRNPKI Called by IP (in process context). If no space is available in the in-core buffer, the routine which dumps the in-core buffer to disk is invoked (it is part of the Multinet Utility fork, which owns the tracing file JFN). Overflow is not a normal occurrence (unless interrupt level tracing is also enabled). All registers are preserved. When called:

T1: Contains the trace code, PT%xxx, identifying the call.

PKT: Contains the address of the packet buffer to be traced (PT%%VI set).

PRNPKT Called by TCP (in process context). If no space is available in the in-core buffer, the routine which dumps the in-core buffer to the disk is invoked (it is part of the Multinet Utility fork, which owns the tracing file JFN). Overflow is not a normal occurrence (unless interrupt level tracing is also enabled). All registers are preserved. When called:

T1: Contains the trace code, PT%xxx, identifying the call.

BFR: Contains the address of the TCP buffer to be traced (PT%%VB set).

PKT: Contains the address of the packet buffer to be traced (PT%%VI set).

TCB: Contains the address of the TCP connection block to be traced (PT%%VH set).

15.4. Diagnostic Facilities

The TCPTST and TCPU programs are used for diagnostic functions. TCPTST can function as a TCP data Source, Echo, or Sink process. TCPTST can also send specially constructed packets using the User Queue facility (the default packet is a standard ICMP Echo).

15.4.1. Using TCPTST to Find a Route

The example below illustrates how an Echo packet including an IP Record Route option is specified. The packet is then sent to host ISIA. The function to display the current route to a host is used before and after sending the ECHO packet to show that a route was added to the routing tables. The IP Record Route option in the returned packet shows the route that the packet followed (at least it would if IP were fully implemented everywhere). See TCPTST.INFO and TCPU.INFO for more information.

```
$tcptst
TCP Test Program   V5.3(46)-7
BBNF.ARPA [8.4.0.12] on Mon 1-Apr-85 21:00-EST
*? one of the following:                    ; The possible commands
 CHECK           CONTINUE         ECHO             EXIT
 HELP            KILL             LIST             LOCAL-HOST
 LOGICAL-HOST    MODE             NO               OPTIONS
 PACKET          PAUSE            PING             RETRANSMISSION
 ROUTE           RUN              SINK             SOURCE
 STATUS

*pacKET (protocol) ICMP (tos) 0 (ttl) 60 (version) 4 (flags) NONE
 (fragment-offset) 0                              ; Default values

Option: ? one of the following:                        ; IP options
 ABORT                End                    Loose-source-route
 LSR                  None                   Nop
 Record-route         Security               SSR
 Stream-id            Strict-source-route    Timestamp
Option: record-route (for) 5 (entries)
Option: end
```

```
ICMP Type: ? one of the following:            ; ICMP packet types
 ABORT              Destination    ECHO           Echo-Reply
 ER                 Info           Info-Reply     IR
 Parameter-problem  Redirect       Source-quench  SQ
 TE                 Time-exceeded  Timestamp      Timestamp-reply
 TR
ICMP Type: Echo (id) 4864 (seq) 8 (data-length) 0  ; Default Echo

*route isia                                  ; No route before ping
No current route to USC-ISI.ARPA [26.3.0.103], would have tried B
BN-FIBER-TEST-GW [192.1.2.6]

*list                               ; Display the received packets

*ping                                                ; Send the ping
*
>PING   Echo Reply received from USC-ISI.ARPA [26.3.0.103] in 902
 msec.
Packet returned:
113 0 0 64                                             ; IP header
0 12 0 0
65 1 56 70
32 3 0 147
10 4 0 14
7 27 24 300                                           ; IP options
1 2 102 32
3 0 147 32
3 0 147 10
4 0 14 0
0 0 0 0
0 0 354 367                                       ; ICMP Echo Reply
23 0 0 10

*route isia                           ; Route to ISIA after pinging
First hop to USC-ISI.ARPA [26.3.0.103] is BBN-FIBER-GATEWAY [192.
1.2.1]
*
*exit
$
```

15.4.2. Using TCPU

The TCPU utility program is used to either "watch" packets between two processes or to be a mis-behaved TCP sink. In either case, the User Queue and Logical Host facilities are used to "implement" a TCP which can drop a few packets (to test a TCP's retransmission algorithms) or add a few ICMP error messages (to see how the hosts will respond). The TCP sink can forever retransmit the final FIN (to check the TIME-WAIT state), etc.

The packets received and sent can be logged either in ASCII or BINARY. The ASCII format is limited but gives a real time display (it would be helpful if PKTPRN were buried inside). Thus the "network" appears to have a low bandwidth or long delay. The BINARY files, however, can be processed by PKTPRN so its full capabilities are available. In addition, the packet samples contain ALL of the data portion of the packets (the internal tracing facilities using MNTRAC sample only the beginning of the packets).

To use the WATCHer, the host addresses of the two processes are specified. One process then opens a connection to the WATCHer's address (instead of the other process). The WATCHer switches the destination address in the packet, recomputes the TCP checksum, and SNDIN%s the packet on to the second process. The second process receives a packet from the WATCHer and sends its packets back to it.

The following example watches a connection between a terminal concentrator (HOST-A) and a VAX (HOST-B). The "a" and "b" indicate reception of a packet from host-a and host-b, respectively.

```
$tcpu
Dummy TCP via User Queue Program   V0.0(121)
BBNF.ARPA [8.4.0.12] on Wed 10-Apr-85 20:52-EST
Parameters:? one of the following:                    ;Commands
 A          ACK              ASCII-OUTPUT
 B          BINARY-OUTPUT    DESTINATION-UNREACHABLE
 DISPLAY    DROP-PACKET      EXIT
 FIN        FLAGS            GATEWAY
 GO         HOST-A           HOST-B
 INPUT      LOCAL-HOST       LOGICAL-HOST
 MASK       REDIRECT         SEGMENT-SIZE-OPTION
 SHOW       SINK             SOURCE-QUENCH
 WATCH      WINDOW
```

```
Parameters:flAGS ? one of the following:
 ACK-RETRANSMITED              HELP
 NO                            NONE
 QUEUE-PACKETS-TO-SEND         SEPARATE-FIN-AND-ACK-OF-FIN
  or confirm with carriage return

Parameters:watCH (protocol) 6                    ;Watch TCP packets
Parameters:logICAL-HOST (number is) 4    ;Set logical host number
Parameters:host-a (is) 192.1.2.41                      ;Process A
Parameters:host-b (is) bbn-vax                         ;Process B
Parameters:binARY-OUTPUT (to file) example.BIN.3   ;Trace packets

Parameters:show                           ;Show parameter settings
Binary output to EXAMPLE.BIN.3
Display OFF
Gateway:    BBNF.ARPA [8.4.0.12]
Host-a:     [192.1.2.41]
Host-b:     BBN-VAX.ARPA [8.1.0.8]
Local host: [8.4.4.12]
Logical host 4 in position [0.0.255.0]
ACKing whole packets
Segment size option off
Window * 2 segsiz extending, initially open
Function: Watch protocol 6
Logical+Protocol  Foreign host  Local host  Port      ;User Queue
[0.0.4.6]         [0.0.0.0]     [8.4.4.12]  [0.9.0.0] ;Value+Mask
[255.255.255.255] [0.0.0.0]     [255.255.255.255]   [255.255.0.0]

Parameters:go                                      ;Begin watching
% No output listing file specified.
Watching abaabbbbabab^C                          ;Packets going by
$ree                      ;Back to deassign queue and close files
$
```

To stop the watcher, TCPU is Control-C'd and then REENTERed to close the output files and deassign the User Queue.

The following example shows what the terminal concentrator does when a few of its FINs are lost. The initial packets synchronize the connection, then TELNET negotiations are received, a character is typed, and the terminal concentrator closes the connection, retransmitting its FIN until it is ACKed.

```
$start
Dummy TCP via User Queue Program   V0.0(121)
BBNF.ARPA [8.4.0.12] on Wed 10-Apr-85 20:56-EST
Parameters:sinK (received TCP data on port) 9
Parameters:fin drOPPED 4 (times when received)
Parameters:logICAL-HOST (number is) 4
Parameters:disPLAY (passing packets) on

Parameters:go
% No output listing file specified.
% No output trace file specified.
Waiting for a SYN ...
    1 2058:33 Received:                                    ;Initial SYN
 IP:  5  00  40      1  25 6TCP [192.1.2.41] 0.14  [8.4.4.12] 0.9
TCP:   33339102          0 5 Psh Syn   512     0
    1 2058:34 Sent:                                       ;SYN-ACK
TCP:  175276032   33339103 5 Ack Psh Syn  1000     0
    3 2058:34 Received:                                          ;ACK
 IP:  5  00  40      1  26 6TCP [192.1.2.41] 0.14  [8.4.4.12] 0.9
TCP:   33339103  175276033 5 Ack Psh   512     0
    4 2058:34 Received:                        ;TELNET negotiations
 IP:  5  00  46      1  26 6TCP [192.1.2.41] 0.14  [8.4.4.12] 0.9
TCP:   33339103  175276033 5 Ack Psh   512     0
  TCP Data: 377 375   3 377 375   1
    4 2058:34 Sent:                      ;SINK ACKs but ignores them
TCP:  175276033   33339109 5 Ack Psh  1000     0
    6 2058:51 Received:                                ;User types "x"
 IP:  5 00    41      1  26 6TCP [192.1.2.41] 0.14  [8.4.4.12] 0.9
TCP:   33339109  175276033 5 Ack Psh   512     0
  TCP Data: 370
    6 2058:51 Sent:                                     ;It is ACKed
TCP:  175276033   33339110 5 Ack Psh  1000     0
    8 2059:03 Received:                     ;User closes connection
 IP:  5 00    40      1  26 6TCP [192.1.2.41] 0.14  [8.4.4.12] 0.9
TCP:   33339110  175276033 5 Ack Psh Fin   512     0
Ignoring til specified # FINs dropped.
    9 2059:03 Received:              ;Several arrive while printing
 IP:  5 00    40      1  26 6TCP [192.1.2.41] 0.14  [8.4.4.12] 0.9
TCP:   33339110  175276033 5 Ack Psh Fin   512     0
Ignoring til specified # FINs dropped.
```

```
   10 2059:03 Received:                    ;Retransmission received
 IP:  5 00    40      1  26 6TCP [192.1.2.41] 0.14  [8.4.4.12] 0.9
TCP:   33339110  175276033 5 Ack Psh Fin   512     0
Ignoring til specified # FINs dropped.
   11 2059:04 Received:                    ;Retransmission received
 IP:  5 00    40      1  26 6TCP [192.1.2.41] 0.14  [8.4.4.12] 0.9
TCP:   33339110  175276033 5 Ack Psh Fin   512     0
Ignoring til specified # FINs dropped.
   12 2059:10 Received:                    ;Retransmission received
 IP:  5 00    40      1  25 6TCP [192.1.2.41] 0.14  [8.4.4.12] 0.9
TCP:   33339110  175276033 5 Ack Psh Fin   512     0
Ignoring til specified # FINs dropped.
   13 2059:13 Received:                    ;Retransmission received
 IP:  5 00    40      1  26 6TCP [192.1.2.41] 0.14  [8.4.4.12] 0.9
TCP:   33339110  175276033 5 Ack Psh Fin   512     0
   13 2059:13 Sent:                   ;Finally ACK it and send FIN
TCP:  175276033   33339111 5 Ack Psh Fin  1000     0
   15 2059:13 Received:                             ;ACK of final FIN
 IP:  5 00    40      1  26 6TCP [192.1.2.41] 0.14  [8.4.4.12] 0.9
TCP:   33339111  175276034 5 Ack Psh   512     0
$
```

16. Host Monitoring

The network software has extensive monitoring facilities. Data is collected in a resident area of memory. A snapshot of the data is periodically taken (every hour, INMPR) and sent to a monitoring host (INMAD) running the PICKLE utility for collection and analysis. Snapshots can also be requested remotely (using the PICKLE utility), or examined locally (using the TCPEEK utility). The STS utility can write detail or summary reports from the collected samples.

16.1. Monitoring Examples

The log below is included to give an idea of the facilities that are available (Note: the example has been edited to fit the page).

16.1.1. Using TSTATS to Display TCP Connection Information

TSTATS is used to display summary information about the host's TCP connections. Shown are a listening, established, and closing connection. The established connection is the one making the log; it shows 250 unacknowledged octets on the way to the terminal. See TSTATS.INFO for a detailed description of the information presented.

```
@tstats
TCP Connection Status Program   V5.3(14)
Flg: O OPEN%  A ABORT%      Urgent recv data
     o Open   v SSEQ valid  urgent send data  (C=*100)

TCB ID: >? one of the following:
 JOB     LINE     TCBID
  or Enter Unique TCB ID#, line 2, entry 2
  or To get all TCBs,  confirm with carriage return
```

```
TCB ID: >
Processing all connections.
BBNF.ARPA on Mon 1-Apr-85 20:54:06-EST
J/L JCN/Id Flg State Sequence   Una Wndow RTT/MxS Un/Baud
 21.   1. R:    CLS 4123459593.          0.  1986.    0.
    584. S:OV  CLS 4199350294.    0.    0.   964.   12C
 5604703  F:192.1.2.68 0.23     L:192.1.2.66 133.67

  0.   1. R:O   EST  113430784.       2832.   411.    0.
 66  652. S:OV  EST 4041279125.  250.  540.   536.   75C
 5610535  F:192.1.2.41 0.14     L:192.1.2.66 0.23

  0.   2. R:O   LSN          0.       2920.  3000. 3000.
    688. S:    LSN          0.    0.    0.262143.    0C
 5607052  F:0.0.0.0 0.0         L:0.0.0.0 0.23
@
```

16.1.2. Using TCPEEK to Snapshot & Display Monitoring Data

The TCPEEK utility can be used to display more detailed information about one of the connections. Information about the established connection in the above example is shown below (TSTATS lists the TCB ID and MONITOR ADDR that TCPEEK needs). See TCPEEK.INFO for more information about TCPEEK and ANAUNV.MAC for descriptions of the fields in the TCB.

```
$tcpeek
PEEK at TCP in operation   V5.4(170)
READY
?  > = take snapshot
  / = print status of symbol, optional +offset or ,limit
      end with / = 2-dimensional, \ = 2-D with 1822 text,
      | = histogram by value
    = print TCB
  E = exit, releasing resources
  G = print gateway information
  M = print guide map
  N = next (/sym+) offset
  R = print route
  S = set RFNTST
  X = transmission time, optional NCT number (defaults to 0)
```

```
  TCB ID(10): 652
 MONITOR ADDR(8): 5610535

TCBID TJCN TOWNR TOFRK TFH          TFP TLH          TLP
652   1    0     12    #30000201051 #16 #30000201102 #27
TSMRT TRXP TSTO   TERR TSTAT    TVTL
562   #0   900000 #0   #2077164 #66
TRIS      TRLFT     TRLAK     TRWND TRLWN     TRURP     TRBS
113430695 113430985 113430985 2631  113433616 113430695 0
TCRCV TCRPC TCDUP TCIGN TCROS TCRZW TCEWN TCPCC TCRST TCRPU TCRUR
0     364   0     0     0     0     0     0     0     0     0
TSIS       TSLFT      TSSEQ      TSWND TSURP      TSBYT TSCB
4041277440 4041286255 4041286507 399   4041278244 0     #0
TSCPK TCSND TCSPC TCRXP TCRXF TCPZA TCFWN TCFWT TCSPU TCSUR
#0    0     381   60    15    76    6     1238  374   1
TCICM TCISQ TCIRD TCFWP
0     0     0     0
TSMXS TCRTM     TCATM     TCLTM     TCBTP     TSPRB
536   157162767 164136854 164494570 164494570 164136854
TCSAG TCFAK TIFDF TTTL TTOS TABTF TCABI
0     0     0     60   #0   #0    0
```

A snapshot is taken and information about 1822 performance and gateway status shown.

```
>BBNF.ARPA snapshot 1,  1-Apr-85 20:59:03 (164504400 msec)
1822 RFNM status:
   RFN8TO/   1
Interface counts:
total [delta] counts from NCT1C

153553 [153553]  Regular Message (0)
     1 [     1]  Dest Host/IMP Dead (7)
     5 [     5]  Interface Reset (10)

Prime gateway up by 192.1.2.6 = BBN-FIBER-TEST-GW 575076 msec ago
   Connecting 192.1.2.6 = BBN-FIBER-TEST-GW
              192.1.4.1 = BBN-FIBER-TEST-GW
              128.11.0.1 = BBN-CRONUS-GW
              192.1.7.1 = BBN-RCCE-GW
              192.1.3.1 = BBN-JERICHO-GW
```

```
Prime gateway up by 192.1.2.1= BBN-FIBER-GATEWAY 3575668 msec ago
   Connecting 192.1.2.1 = BBN-FIBER-GATEWAY
              8.0.0.13 = BBN-FIBER-GATEWAY

Dumb gateway up by 8.7.0.2 = BBN-LABS-B.ARPA 1775348 msec ago
   Connecting 8.7.0.2 = BBN-LABS-B.ARPA
              192.1.9.1 = no name
              192.1.11.1 = SPC-COD

e  quitting...
$
```

16.1.3. Using PICKLE to File Monitoring Data

The PICKLE utility is used in Receive mode to file monitoring reports automatically sent to it by other systems. Those systems specify the system running PICKLE as the remote monitoring host by include its IP address, a.b.c.d, in the SYSTEM:SITE-ADDRESS.TXT file:

```
SET:INMAD:&a.b.c.d
```

PICKLE may either be run as a batch job or as a subjob of SYSJOB. Note that PICKLE must be able to use a User Queue; ARPANET-ABSOLUTE-SOCKETS or ARPANET-WIZARD capability or permission of the Access Control Job.

A typical control file, MON.CTL, would be:

```
@SUBMIT MON/TIME:0:30:0         ; In case of shut downs
@ENA                            ; Need capabilities
@GET PICKLE                     ; Might need directory
@REENTER                        ; Begin in Receive Mode
@
```

Alternatively, commands in SYSJOB.RUN specifying subjob 7 would be:

```
JOB 7 %
LOG OPERATOR
ENA
GET PICKLE
REENTER
%
```

The amount of data collected is specified by setting the MNTMN compilation parameter in ANAUNV.MAC before compiling the network sources; 0 is the minimum level, 9 is the maximum level, and 5 is the default.

16.1.4. Using PICKLE to Request Monitoring Data

PICKLE can also be run in Send mode to request a monitoring sample from another system. In order to accommodate variations between different sites, the first (automatic) monitoring report sent and all reports specifically requested by PICKLE include a symbol table which covers the monitoring data (the same symbol table that IPOPR% .IPRIP and TCOPR% .TCRTC functions use). The follwoing example shows a request for monitoring data.

```
$cwl:pickle
Tops-20 Monitoring Statusgram Filer   V0.0(66)-7
  R (receive) command first ==> monitor mode
  S (send) command first ==> manual mode
READY
v  Verbose mode on
Date & time in pkt/up H:M/ seq # /   .bin ptr/ len/src host

s  Host to tickle: bbnf
  Tickle # 1 to host 30000201102  = 192.1.2.66 = BBNF.ARPA
 1-Apr-85 21:37:13  46:19  19,1             0 7941 BBNF.ARPA

e  Stopping inferiors ... stopped,  quitting...
$

$vd bbnf*
   PS:<CLYNN>
 BBNF.ARPA.BIN.1;P777700   16 7941(36)    1-Apr-85 21:37:14 CLYNN
   .INDEX.1;P777700         1 62(7)       1-Apr-85 21:37:15 CLYNN
```

16.1.5. Using STS to Write Monitoring Reports

The STS utility is used to write detail or summary monitoring reports. It has an interactive mode which accepts the names of quantities to be written into a custom report. It accepts overlays for both a specific host's monitoring format (so the host's symbol table does not have to be LOADed from the monitoring data) and for custom reports. It can either SCAN a set of data filed by PICKLE or be limited to a specific sample

using the OFFSET command. An example of using the STS utility to print a detail report (from the sample requested in the last example) is shown below. (See STS.INFO for more information.) First the INDEX file is examined to determine the OFFSET to be reported (the first is "0"; the sequence number, "19,1", indicates that 19 regular monitoring reports have been sent and that this is the first remote request that has been requested).

```
$ty bbnf.ARPA.index.1
 1-Apr-85 21:37:13  46:19  19,1             0 7941 BBNF.ARPA
$

$cwl:sts
Status>? one of the following:
 DIFFERENCES      EXIT           FORMAT          GET         LOAD
 OFFSET           QUIT           REPORT          SAVE        SCAN
 WRITE

Status>oFFSET (into input file) 0
Status>lOAD (from file) bbnf.arpa.bin
% Loading symbol information from sample...
Status>wRITE (to file) STS.LST.2 !New generation!
Status>exit
$
```

A few entries from the detail report are listed below.

```
TCP/IP Status from file
[BBNF.ARPA]PS:<CLYNN>BBNF.ARPA.BIN.1, 1-Apr-85 21:37:14
Data from host BBNF.ARPA sample 4865 (length 7941) at position 0
For 30-Mar-85 23:18:38 (up 0:01) to  1-Apr-85 21:37:13 (up 46:19)
Data header: 17405200000  212001740400  46004000000  -1350000000
             -177773755740  -177773755740  1704230000  12431

Network Statistics Area - Len  5400.
Sample GTAD         1-Apr-85 21:37:13
Sample TODCLK       166794457.(1-Apr-85 21:37:13.000)
Init GTAD           30-Mar-85 23:18:38
Init TODCLK         80452.(30-Mar-85 23:18:38.000)
# NCTs/Version      3.,,7.
```

```
Telnet port          23.
 Sync timeout(s)     30.
 Estab timeout(s)    900.
 Msg ms/char         500.
 Negotiation timeout(ms)  30000.
 Check listener(ms)  60000.
 Probe window(ms)    60000.
 Default RX pars     0

TCP RX               Process statistics:
  Process lock       Lock  Msecs     Wait.gt.0,,Old Conflicts,,New
                     -1.   30. msec        20.,,0          0.,,0
  Next run           166794781.(1-Apr-85 21:37:13.324)
  Last signal        166793781.(1-Apr-85 21:37:12.324)
  Queue head
    Count/Next       1. / 5610625
    Max  /Prev       7. / 5610625
    When /Head       166400. / 142644
  Run counter        49708.
  Run time(msec)     21915.
  Sigs received      59229.
   TCBs examined     22093.
  TCBs processed     49708.

Total lock conflicts  27.
Bad checksum version etc  0.
TCP SYNs received    679.
 SYNs Sent           698.
 probes              0.  0.
 RSTs received       66.
 RSTs sent           338.
 FINs received       615.
 FINs sent           464.
 pkts received       154597.
 duplicates          1142.  0.
 reassembled         99614.
 bytes received      1501594.
 bytes sent          9347767.
 packets formed      130127.
 retransmitted       7539.
 packets output      138312.
TCP tasks run        486496.
```

```
ICMP codes recv     18. Bkts  815. Cnt
0 Echo Rep          195.
3 Dest. Unreachable 29. Cnt
 net                6.
 host               9.
 protocol           14.
4 Source Quench     54.
5 Redirect          519. Cnt
 net                519.
11 Time Exceeded    18. Cnt
 ttl                18.

ICMP Codes Sent     906. Cnt
0 Echo Reply        39.
3 Dest. Unreachable 19. Cnt
 net                6.
 port               13.
4 Src Quench        688.
8 Echo              160.

ICMP Codes Sent to SELF 20. Cnt
3 Dest. Unreachable 4. Cnt
 protocol           4.
11 Time Exceeded    16. Cnt
 ttl                16.
```

16.2. Adding New Monitoring Points

To add new monitoring points space must be allocated for the data and code must be written to collect it.

Space for a monitoring datum is allocated by including another NTITEM macro to the definition of DEFSTS (in ANAUNV.MAC). (Note that for efficiency only B4 to B35 are available in the reports sent to the monitoring host.) NTITEM has seven arguments.

```
NTITEM(nam,len,lvl,typ,init,fmt,symb)
```

Argument	Description
nam	The symbolic name of the cell (vector); it should contain 5 characters.

len The length, in words, of the vector (must be specified, ge 1).

lvl The minimum monitoring level required.

typ One of the following:

G "nam" has the form bbbAA and is a global symbol, defining the beginning of a new block, bbb. Space is allocated within the monitoring area. Subsequent NTITEMs in the same block are "O"s.

O "nam" has the form bbbnn and identifies a named sub group of data in block bbb (it is an offset (not global) from the last "G"). Space is allocated within the monitoring area.

R Obsolete; used to get the TCB address from register TCB via an LDB.

S "nam" is a previously defined variable or table; not in the standard monitoring area.

T "nam" is a TCB structure entry.

init A list of initial values. Blank entries are cleared by the STSINI call to reset the counters; non-blank entries are not cleared. The symbol REST0 may be used to specify that the rest of the vector should be zeros (i.e., the unspecified values should not be cleared).

fmt Describes the format of the vector. It is primarily used by the STS utility program which analyzes the data and writes reports. Fmt has one of five possible values:

	Elements in vector	Description of elements
B1	n,<x>*(len-1)	(computed) block length, followed by "len"-1 words of data.
V1	<x>*(len)	a vector of "len" words.
V2	n,s,c,<y,z>*(n)	histogram by range consisting of a fixed bucket count (n), the computed sum of the entries (s), the computed

count of entries (c), and n <instance count, maximum value> pairs; "len" is 2*n+3. The INHIST macro can be used to histogram values into vectors of this format. N and the maximum values for the buckets must be specified in the "init" list. The max values must be in increasing numerical order. The last value may be -1 to signify an overflow bucket.

M1	n,s,c,<x>*(n)	histogram of small integers consisting of a fixed bucket count (n), the computed sum of the entries (s), the computed count of entries (c), and n <instance count> buckets; "len" is 2*n+3. The INHSTI macro can be used to histogram integers from 0 to n-1 into vectors of this format. N must be specified in the "init" list.
M2	n,s,c,<x,oM1>*(n)	two dimensional histogram of small pairs of integers consisting of a fixed bucket count (n), the computed sum of the entries (s), the computed count of entries (c), and n <instance count, address of second dimension vector> buckets; "len" is 2*n+3. The INHSTI macro (with size 2) can be used to histogram the first dimension; it leaves the pointer pointing to the word containing the address of the second dimension; the appropriate macro can then be used to histogram the second dimension value. The number of buckets in the second dimension can vary, but the second dimension must have a uniform fmt. oM1 is the address of the second dimension vector. N and the oM1 addresses must be specified in the "init" list. The values to be histogramed are pairs of integers, from 0 to n-1, for the first dimension.

symb A list of short descriptions for the individual entries. The descriptions are used by the report writer, STS, to identify the data. The five possibilities are:

B1 block name, n-1 names.

V1 n names.

V2 name, n names.

M1 name, n names.

M2 name, n names.

Detailed information about the NTITEM macro, including examples, is contained in ANAUNV.MAC.

Within the network code, data collection code segments are included in MNTMx macros, where MNTM0, MNTM2, MNTM5, and MNTM9 are currently defined. If the monitoring level is greater than or equal to the digit, the code is compiled.

References within the code to the entries defined by NTITEM are made via the CELL macro.

```
MNTM5<  AOS CELL(nam,off,idx,blk) > ; Count event
```

The following table lists the arguments for the CELL macro:

Argument	Description
nam	A nam appearing in an NTITEM.
off	The word offset from nam, in the range 0 to len-1. CELL makes sure the offset is within the defined range.
idx	An index, typically (ACx). The range cannot be validated by CELL.
blk	The block, "bbb", in which nam appears.

In the usual case, index is omitted and the monitoring block blk is specified. The offset may be increased by the contents of an index register, e.g., (T2), or may be used and blk omitted when the index register points to the block.

An alternate form, XCELL(nam,off,ind,blk) should be used when the index points to the named entry. It does not use nam or blk, but they may be included as documentation.

17. Things to Do

There are several areas where the network code could be improved:

Accounting functions when a TCP connection closes.

Better estimate of output queue length - NTQPKT - ought to include time packet currently being sent has been in progress - it will help catch the spikes in local net delay.

Better source quench processing; finish limits on # sent per unit time.

Bit in ASNIQ% to trace User Queue packets.

Check how well parameters for the routing tables are working - routes timing out too soon or too slow; selectively time them out instead of flushing all at once (get flood of redirects).

Check that things work when more than 2**32 octets have passed over a TCP connection.

Check where page faults are happening; could things be improved?

Clean up the monitoring area; sort items into better levels; make NTITEM use the level to reduce section zero space when data not collected.

Extend routing table to use source routes instead of just a first hop address; insert source routes automatically when one interface fails.

Finish advisory messages to user terminal (source quench, routes, unreachable, etc.).

Finish allowing users to trace their packets - lock on facility & file in user area.

Finish implementing MNTRAPs, write the data to MNTPTP & get it written as required (locally or to monitoring host).

Finish integrating GTJFN% attributes; make it setup TCB directly instead of going through BBN interface.

Finish "optimization 2" - use high 5-6 bits of send sequence for hash instead of local port (all telnets end up in same bucket).

Finish the rewrite of the multinet check hardware routine; better support for virtual ncts.

Get the name resolver and cache code from ISI when available.

.GTHRR return format - would be nice if had fields indicating syntactic items, fields were aligned for easier use.

Implement IP broadcast.

Implement security and preemption.

JFN interface should put data directly into/remove from packets, not go through another layer of buffering. Might as well make it support all byte sizes - TCPSIM.MAC has the code to do it.

Make SITE-ADDRESS parser accept symbols and their sums.

More work on "optimization 1" to gain information about lost packets, retransmissions, etc.

More work on packetizer to slow it down so more characters get put into a packet - how long can you wait before the user starts to notice?

Only one source quench from TCP per group of receiver packets.

Performance improvements.

Processing of system messages over TELNET protocol is racy.

Reduce offered window if past packets received are pushing little packets.

Remove acked data when repacketize.

Source quench for User Queues.

Test the fragment chaining code; how well does it work at providing fairer access to the local network.

Index to Parts 1, 2, and 3