Dongbin Yang

Developing and improving 5G telecom message traces software for a DU component

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

2 May 2025

Abstract

Author:	Dongbin Yang
Title:	Developing and improving 5G telecom message traces software for a DU component
Number of Pages:	42 pages + 2 appendices
Date:	2 May 2025
Degree:	Bachelor of Engineering
Degree Program:	Information Technology
Professional Major:	Smart IoT Systems
Supervisors:	Marko Uusitalo, Senior Lecturer Ahmad Banijamali, Team Manager Jere Pylkkänen, Software Engineer

---

The goal of this study was to continue developing a message tracing and debugging tool for a distributed unit component - Cp-rt, which could support the Cp-rt component and improve message sequence tracking, enable system analysis and reduce debugging time.

The study was conducted by an inductive approach for an internal project in case company. The project under a distributed unit context focused on the development of a tool- Endeavour, which is a graphical user interface-based test framework and was developed inhouse for 3G technology and several principles and features were also incorporated into the development of 5G. The development of Endeavour, however, was only manipulated in the distributed unit. The assumption of study is to convert messages into corresponding format and display its sequential flow chart in the Endeavour.

The study covered the software design and implementation of necessary modifications, including message format conversion, protocol handling, and synchronization between Cp-rt and other 5G components. Key developments include the automation of protocol file generation, improved message serialization, and enhanced system compatibility.

The study found that Endeavour tool could be capable of handling new messages in the distributed unit using well-designed software plan, and considering the compatibility with other units in the 5G system. However, the challenges remain in areas such as synchronization, message parsing, and scalability. Such peer development merely lays the groundwork for optimization and ensures that the Endeavour framework continues to support evolving 5G network requirements.

Keywords: 5G, Control Plane, Software Development, Testing, Endeavour, Distributed Unit, Mobile Network

Contents

Appendices

List of Abbreviations

AMF:	Access and Mobility Management Function
AN:	Access Network
BTS:	Base Transmission Station
CU :	Central Unit
CN:	Core Network
Cp-rt:	Control Plane Real Time
DU:	Distributed Unit
DN:	Digital Network
eMBB:	Enhanced Mobile Broadband
EPC:	Evolved Packet Core
FM:	Fault Management
gNB-CU-UP:	gNodeB Central Unit User Plane
gNB-CU-CP:	gNodeB Central Unit Control Plane
gNB-DU:	gNodeB Distributed Unit
JSON:	JavaScript Object Notation
LTE:	Long Term Evolution
MeNB:	Master Enhanced NodeB
MuLTI:	Multi Language Toolkit for Interface
MIMO:	Massive Multiple-Input and Multiple-Output
MME:	Mobility Management Entity
NR:	New Radio
NS:	Networking Slicing
NFV:	Network Function Virtualization
OAM:	Operations, Administration and Maintenance

PIT:	Plane Integration Tests
PDU:	Packet Data Unit
RAN:	Radio Access Network
RIC:	RAN Intelligent Controller
RU:	Radio Unit
SCT:	System Component Test
S-GW:	Serving Gateway
SDN:	Software-Defined-Networking
SBA:	Service-Based Architecture
SMF:	Session Management Function
SCTP:	Stream Control Transmission Protocol
SysCom:	System Internal Communication Protocol
SUT:	System Under Test
TCP:	Transmission Control Protocol
TTCN-3:	Testing and Test Control Notation – Version 3
UMTS:	Universal Mobile Telecommunications Standard
UE:	User Equipment
UPF:	User Plane Function
UT:	Unit Test
WCDMA:	Wideband Code Division Multiple Access
WIMAX:	World Interoperability for Microwave Access
XML:	Extensible Markup Language
3GPP:	3rd Generation Partnership Program
1G:	1st Generation Cellular Network
2G:	2nd Generation Cellular Network

3G:	3rd Generation Cellular Network
4G:	4th Generation Cellular Network
5G:	5th Generation Cellular Network
5GC:	5th Generation Core
6G:	6th Generation Cellular Network

1 Introduction

The development of cellular networking technology has given rise to a need for faster end-user data rates due to constantly increasing performance requirements of the evolving multimedia since the 4th generation Long-Term Evolution (LTE) (Penttinen 2019: 35). The era of the 5th generation along with a massive number of smart devices such as phones, watches, internet-of-things and smart cities has been designed to cope with these challenges by providing more capacity and enhanced user experiences to solve all the current needs even more for the most advanced applications, such as virtual reality.

The 5th generation is the outcome of long development in mobile communication, which roots back to the first-generation mobile communication networks in the 1980s (Penttinen 2019:35). The 3rd Generation Partnership Project (3GPP) organization standardized radio network technology parameters followed by 3G and 4G (IEEE Spectrum, 2018). The 5G technology with high throughput, ultra-reliable low latency and massive communication characters has broken through the limits of 4G (LTE). Therefore, its advanced capacity and complex design of architecture are determined to require comprehensive, effective and efficient testing for new features among multiple components.

The system scale in the 5G architecture design determines the complexity of interaction among various components. While the numbers of test cases increase, it eventually leads to onerous development that not only requires more allocated resources but also a longer time for the testing process. Nokia, the case company in this study, therefore decided to improve their current approach of system testing by developing an extension tool named Endeavour as a part of Emil application, which could improve the testing efficiency and shorten the time taken by procedures, eventually reducing costs.

1.1 Research Purpose

This study is a part of an internal development project at the case company and focuses on the development of the Endeavour tool, which was developed and

maintained by an internal development team. The purpose of this study is to build a foundation and develop new features under Endeavour tool for the Cp-rt component to improve reproducing the log trace of communication effectively as a message sequence chart, which could provide better and clearer understanding of the system log and shorten the time of maintenance efficiently.

The Endeavour system was originally developed inhouse and is not only used in 5G Megazone Operations, Administration, and Maintenance (OAM) development for System Component Test (SCT) and Plane Integration Tests (PIT), but also it has been conducted in the development of Wideband Code Division Multiple Access (WCDMA) (Nokia Endeavour, 2022). Therefore, it would be crucial to make it applicable to involve one of the key components in 5G.

1.2 Research Structure

This study includes six main sections. The first section is the introduction, which briefly introduces the background of the study and the research problems and objectives.

The second section will address the theoretical background about the mobile network, the 5G architecture and the RAN and the core network. Due to the complexity of 5G architecture, this study, however, merely focuses on the development of one of key components, Cp-rt. Moreover, its architecture and functionalities shall be discussed in section 2.

The project design and verification methods will be discussed in Section 3. In this section, the architecture of the case software will be illustrated by going through the research methodology and how to verify the software. For example, unit testing and system testing approaches for the case study will be addressed in the implementation in this section.

Major implementations will be explained in Section 4 and Section 5. Section 4 deals with the entire development of the system for the case study. Description

of the development will begin with a review of the codebase of the Endeavour structure to understand its functionalities and relations among different files. However, due to a non-disclosure agreement, the developed code is not fully presented, and the implementation will merely be described and discussed in section. Subsequently, Section 5 explains how the Cp-rt interface had to be implemented or modified based on the existing codebase of central unit (CU) components. In addition, the section explains how unit tests were applied to verify the feasibility and usability of new development.

Section 6 presents an operational analysis and future development. From an implementation perspective, synchronization with other components is addressed in the section due to the limited number of test cases for the case study. Possible improvements and research will be discussed in this section to provide readers with a clear idea about further research direction.

The last section will provide a conclusion and discuss how the outcome could be utilized in the Endeavour tool for message tracing tasks or how its functionalities could be integrated into other projects.

2 Theoretical Background

Telecommunication of networks is a rather broad topic, and the fifth generation also is a complex technology. The background information of the development of mobile networking, the 5G operation principle and the control plane architecture and its component, are addressed and discussed in this study.

2.1 Mobile Networks

The wireless communication technology on mobile devices has undergone a revolution during several decades from the1st generation (1G) to the 5th generation (5G). Many different wireless connections appear, and data transfer methods have improved from generation to generation in consideration of the maximum throughput, latency and reliability (Sivakurmar, Nasir and Rajanedaran, 2017).

The breakthrough on wireless communication technology of each generation has paved the way for the most advanced and sophisticated technology such as 5G or even 6G for the coming future (Bhatti, n.d.). The differences of each generation are depicted in Figure 1. The initial network (1G) system designed purely for voice call without data handling was introduced in the early 1980s as analog cellular systems. Although the 1G system came with limited capacity, it inspired the upcoming digital cellular system named 2G by improving sound quality, better security and higher total capacity (Sivakurmar, Nasir and Rajanedaran, 2017).

While the 3rd generation (3G) cellular network system provides the much faster data transfer and aims to enable the long expected video-conferencing with UMTS (Universal Mobile Telecommunications Standard), the 4th generation (4G) integrates the 3G cellular networks and Wi-Fi networks with fixed internet as support evolutes the system beyond the limitations of 3G and enhances the quality of services to increase the bandwidth as well as reduce the cost of the resource (Bhatti, n.d.). The 4G cellular system network has adopted Long Term Evolution (LTE) and Worldwide Interoperability for Microwave Access (WIMAX), which was introduced in the end of 2000. This provides even faster data transmission rates and reduced latency, which makes data transmission possible in mobile web access, gaming service, video conferencing, high-definition television and wearable upgrade (Sivakurmar, Nasir and Rajanedaran, 2017).

Technology	1G	2G/2.5G	3G	4G	5G
Deployment	1970/1984	1980/1999	1990/2002	2000/2010	2014/2015
Bandwidth	2kbps	14-64kbps	2mbps	200mbps	>1gbps
Technology	Analog cellular	Digital cellular	Broadbandwidth/ cdma/ip technology	Unified ip & seamless combo of LAN/WAN/WLAN/PAN	4G+WWWW
Service	Mobile telephony	Digital voice, short messaging	Integrated high quality audio, video & data	Dynamic information access, variable devices	Dynamic information access, variable devices with AI capabilities
Multiplexing	FDMA	TDMA/CDMA	CDMA	CDMA	CDMA
Switching	Circuit	Circuit/circuit for access network&air interface	Packet except for air interface	All packet	All packet
Core network	PSTN	PSTN	Packet network	Internet	Internet
Handoff	Horizontal	Horizontal	Horizontal	Horizontal& Vertical	Horizontal& Vertical

Figure 1: Comparison of wireless network technology (Sivakurmar, Nasir and Rajanedaran, 2017)

Since the 5th generation of cellular network (5G) was introduced in 2010, 5G has been rolled out as the next phase of mobile telecommunication standards beyond the 4G standards (Sivakurmar, Nasir and Rajanedaran, 2017). Even though 4G will remain the dominant mobile access technology by subscription, by 2026 it is anticipated that there are 3.5 billion 5G subscriptions globally, which will account for 40% of all mobile subscriptions and 54% of total mobile data traffic carried by 5G networks (Ericsson Mobility Report 2022).

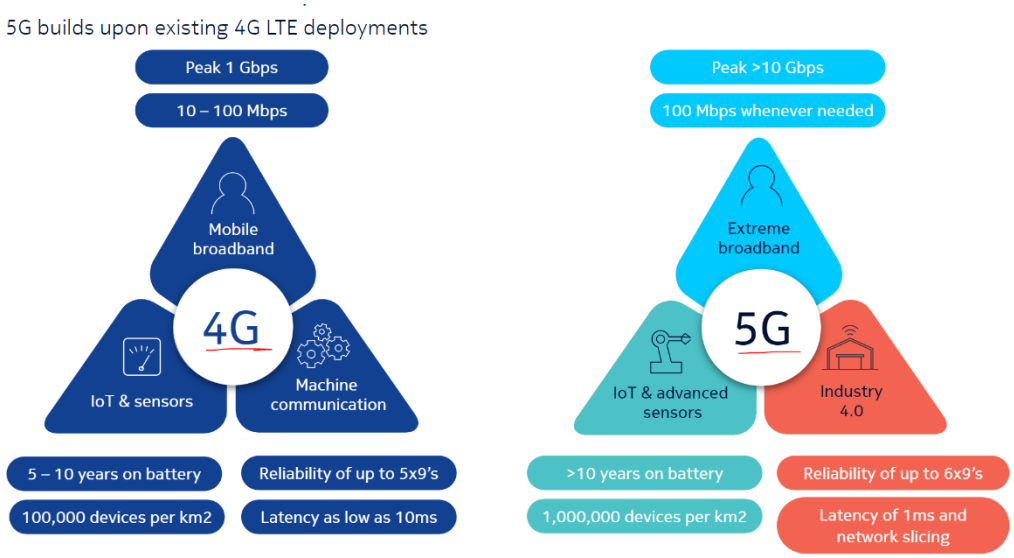

Figure 2: 5G improving over 4G LTE deployments (Nokia, 2022).

The 5G mobile network is the system defined by 3GPP from Release 15 onward (3GPP, 2023). In comparison to predecessors, the operations and architecture of the mobile networks have changed. The 5G network system improves 4G services over higher data rates by Enhanced Mobile Broadband (eMBB), 6x9's high reliability enabled by Edge Computing capability, high traffic densities of devices with a wide range of devices and services anticipated in the 5G timeframe, and dynamic network operations (3GPP, 2023).

The 5G mobile network is envisioned to offer sophisticated connections with different types of devices in a connected society and assist to satisfy the service requirements in various use case scenarios to achieve a real 'Internet of Everything' (Erunkulu et al. 2021). Release 18 in 2022 has significantly symbolized a milestone of 5G system evolution from 5G-Basic to 5G-Advanced. Release 18 evolved in the areas of artificial intelligence to enhance the intelligence of wireless networks for all use cases from single system design on compatibility to a diverse configurability and the human-machine interaction experience (Ericsson, 2021).

2.2 5G Architecture, RAN and Core Network

Classical mobile telecommunication networks comprise four major domains, which are User Equipment (UE), radio access network (RAN), core network (CN) and transport network. The functionality of the major domains formed the foundation of traditional networks architecture, in which RAN uses radio frequencies to provide wireless connectivity to the UE. CN coordinates between different parts of the access network and provides connectivity to the internet. The transport network provides connectivity between RAN and CN (Erisson, 2020). However, 5G is an evolution of the traditional mobile network, which is designed to support diverse services with different data traffic profiles (high throughput, low latency and massive connections) and models (IP data, non-IP data, short data bursts and high throughput transmissions). Packet Data Unit

(PDU) session types also includes a IPv4, IPv6, IPv4v6, Ethernet and Unstructured (3GPP TR, 2019).

The 5G's main key characteristic is not only a new radio interface but also its standard-alone (SA) and non-standalone (NSA) architectures rely on a modular separation between the *5G core network* (5GC) that controls resource allocation and security between network and user devices and *5G RAN* that focuses exclusively on managing radio communication and encrypting the signaling and user plane traffic. This separation allows operators to build resilient multi-vendor 5G networks (Kennedy, 2019).

The feature and functionality for 5G system deployment are defined by 3GPP in system architecture, which is an ongoing development process due to new demands. Besides the two core components, NG-RAN and 5GC, 3GPP also defines the system to be an interaction between the UE and an end point, which could be a server like Application Server (AS) or alternative UE. Therefore, the system establishes communication between DN and UE via AN and CN (3GPP TS, 2018).

Figure 3: 5G system end-to-end architecture (3GPP TS, 2018)

2.2.1 5G Radio Access Network

The 5G radio access networks do not merely depend on a single radio access, which increments not only the flexibility, capacity and reliability of the network but also creates multi-connectivity with various access networks to achieve multiple access typologies combining with 4G or WiFi as part of radio access (Nokia, 2022).

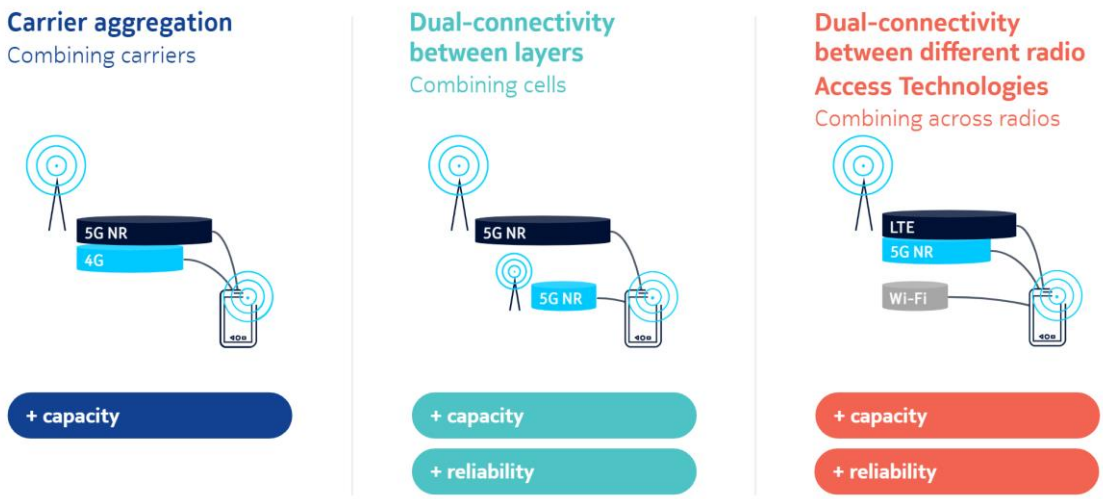

Figure 4: 5G multi-connectivity with various access networks (Nokia, 2022)

The New Radio (NR) introduction and development by 3GPP creates the new standards for the 5G network, which facilitates communication between transmitters and receivers at frequency bands of up to 71GHz on release 17 (Erunkulu et al. 2021). The frequency spectrum is divided into frequency ranges 1 and 2. Frequency range 1 is divided into low band (400MHz - 3GHz) and mid-band (3GHz - 7GHz), and in turn frequency range 2 uses high band, keeping in range of 24.25GHz - 71GHz (Nokia, 2022). The NR intends to be a scalable and flexible air interface, and its framework accommodates massive multiple-input and multiple-output (MIMO), which can enhance the network coverage and capacity performance matching massive data services (Vook et al. 2018).

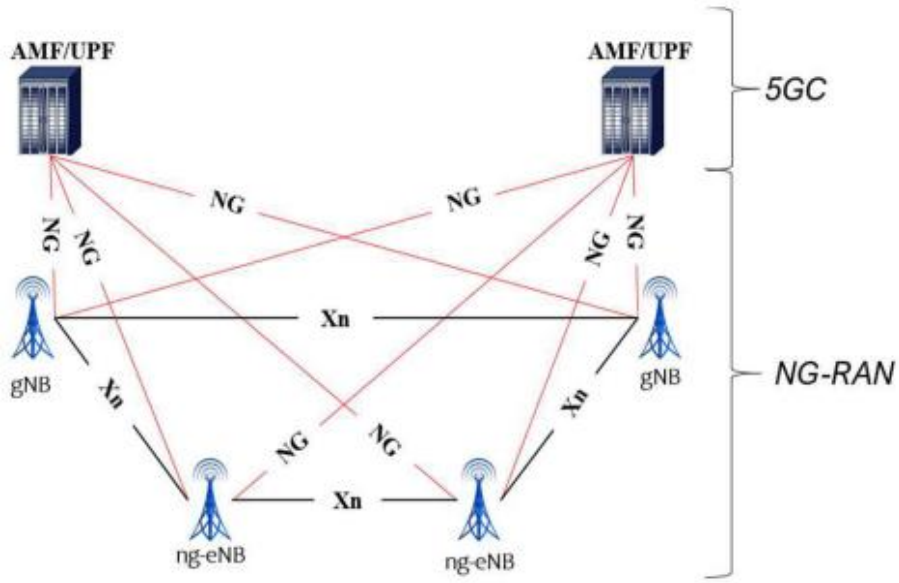

Figure 5: Standalone (SA) architecture for 5G system (3GPP TR, 2019)

The other architecture is NSA, where the 5G RAN and NR interface is used in conjunction with 4G LTE and EPC Core Network to ensure availability of NR technology without the replacement of the network. However, services provided by 4G can enjoy the capacity offered by the 5G NR (3GPP TR, 2019), also named E-UTRA-NR Dual Connectivity. The base station of 5G NR links to the 4G/LTE base station as an enhanced NodeB (eNB) through interface X2, and responsible for establishing the connection between the UE and the 5GC interface (Soos et al, 2020). The S1 and S1-U interface respectively connect eNB and en-gNB to EPC, the front made up of Mobility Management Entity/Serving Gateway components (MME/S-GW), while the latter connects the other gNB through the X2-U interface (Cisco, 2021).

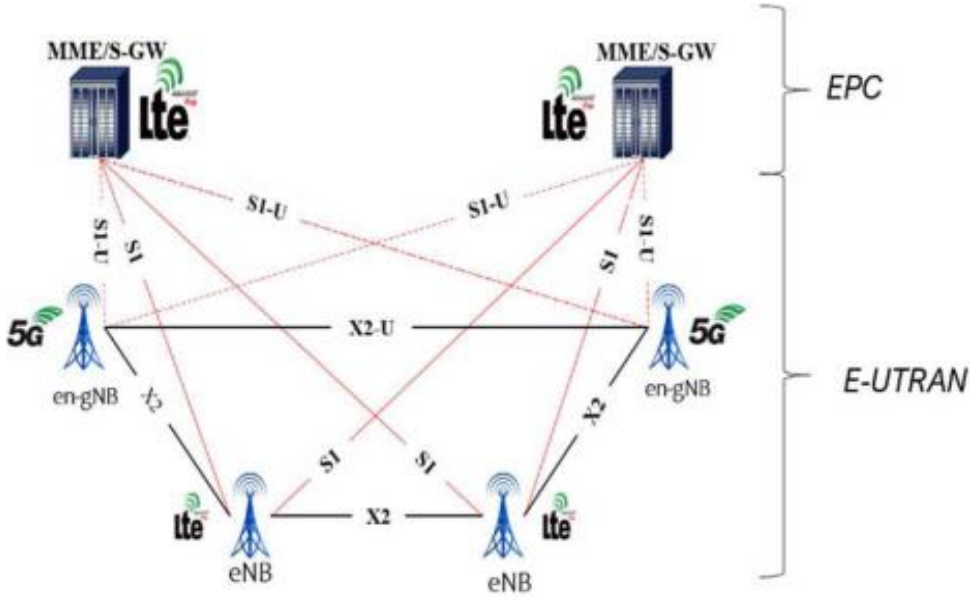

Figure 6: 5G NSA architecture system (3GPP TR, 2019)

The MME and configuration data provide the information to E-UTRAN to determine the mode of connectivity for the UE. Therefore, 4G is fully in charge of handling the signals in the control plane. The 5G-RAN provides both LTE and NR radio access and the 5G-RAN node includes the eNB and gNB node. Furthermore, the eNB and gNB deliver the functions of the control plane and the user plane respectively from LTE or NR towards the UE (Bertenyi et al, 2018). Moreover, the eNB and gNB are interconnected by the Xn interface, which connect them to 5GC that directly connects to Access and Mobility Management Function (AMF) through the NG control plane interface as well as to the User Plane Function (UPF) through the user plane interface (3GPP TS, 2018).

2.2.2 5G Core Network

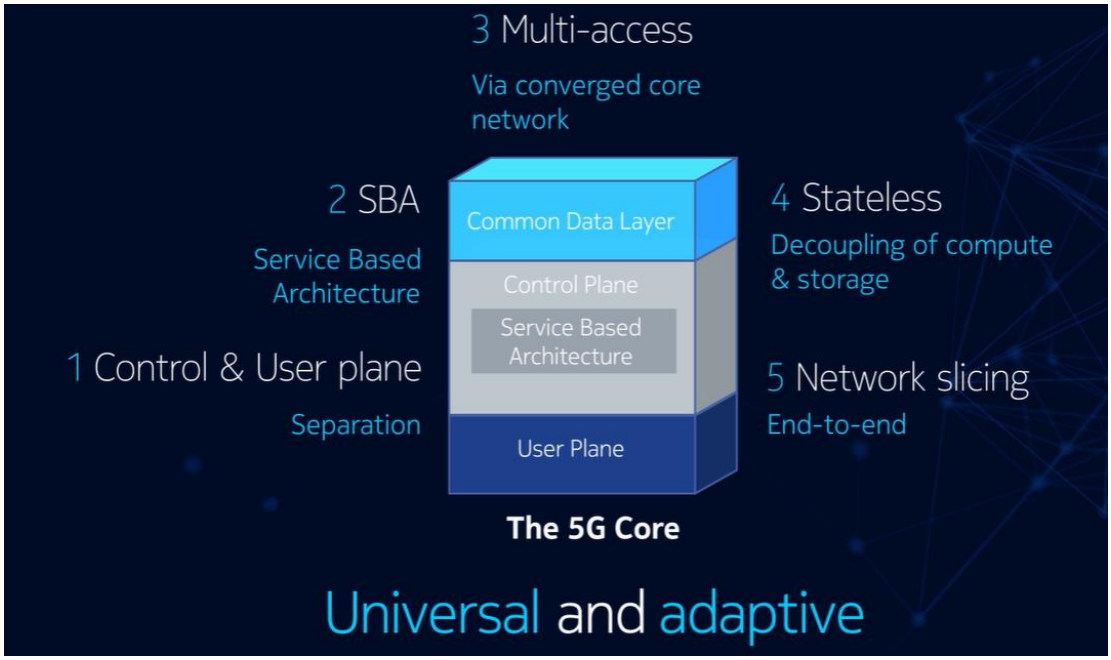

Figure 7: Key features for 5G core (Nokia, 2022)

The 5G network inherently becomes more efficient, handling more connections at low latency per device. The novel technical design of networking slicing (NS), software-defined-networking (SDN) and network function virtualization (NFV) provide the possibility to manage across a wider range of radio frequencies. Moreover, by novel evolution, it could unlock in the midrange and millimeter-wave (mmWave) bands for carriers to expand network offerings (Hollington, 2023) (Erunkulu et al. 2021).

The 5G core with these five key features becomes the universal and adaptive network core. The ***control & user plane separation (CUPS)*** causes the user plane to be closed to user access point, reducing latency and transport network load. The ***Service-Based Architecture*** (SBA) are defined based on service requirements. The ***multi-access via converged core network*** allows wireline and wireless convergence and supports a different use of different phase and mobile network access. ***Stateless*** decouples the compute and storage resources but retain the state in a secured and centralized repository. ***Network slicing*** can assign the network-slice instances simultaneously up to 8 slices for various services and applications (Nokia, 2022). Therefore, the key traits ensure

the capability of managing all the data, voice and internet connections under fast and reliable nature.

Figure 8: Schematic overview of 5GC (3GPP A Global Initiative, 2023)

The 5G core architecture relies on the Service-Based Architecture (SBA) framework, in which elements are defined according to the service requirements (Erunkulu et al., 2021). The session management function (SMF) and AMF govern the functionalities and procedures in the core network. The UPF handles the user data, while the AMF manages the signals to access the UE and RAN in the control plane. The AMF/UPF/SMF delivers control plane functions primarily from the 5G core (3GPP A Global Initiative, 2023). The SBA is defined by 3GPP as a process of delivering the standard data repositories and functionality in the CP through a set of interconnected network functions (NFs) (Erunkulu et al. 2021).

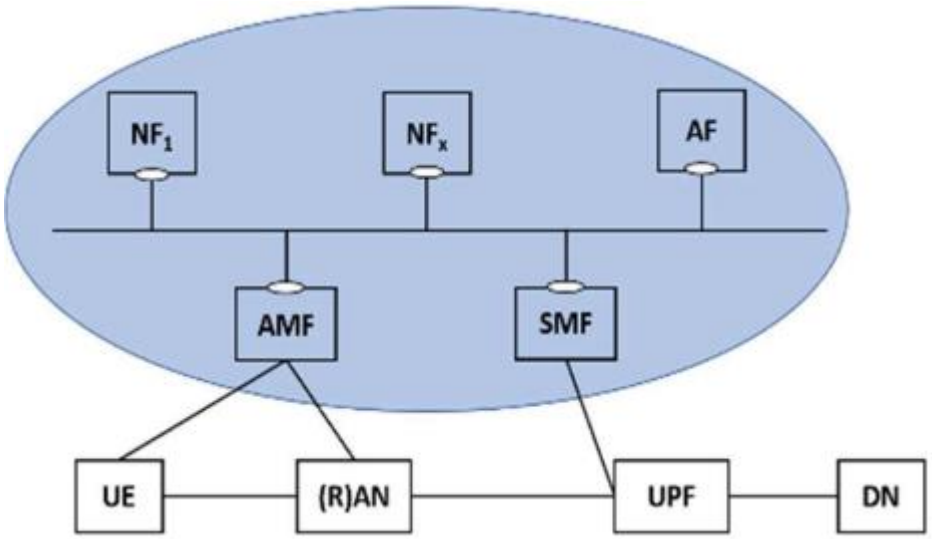

Figure 9: 5G system architecture with a set of interconnected NFs (Erunkulu et al., 2021)

The system architecture is composed of a set of interconnected NFs in 5GC, which offer specific services by NFs through intercommunication on common framework interfaces. The NFs is software manageable in the server rather than a dedicated device, which equips cloud native, making it faster and easier to upgrade, optimize resource utilization and enable access for authorized 3rd parties. Therefore, the SBA provides reusability and modularity and enables virtualized deployment (Zhang et al, 2018).

2.3 Control Plane

The modularity adaptation has been equipped into the 5G core network and the separation between the control and user plane has become the major split in the 5G core architecture, which interacts with RAN respectively through different interfaces (Kennedy, 2019). The control plane is not an exclusive concept to 5G core, and it is a part of the network to manage and control the exchange of data that support the functions in the 5G system that establish and maintain the user plane (Guttman & Ali, 2018).

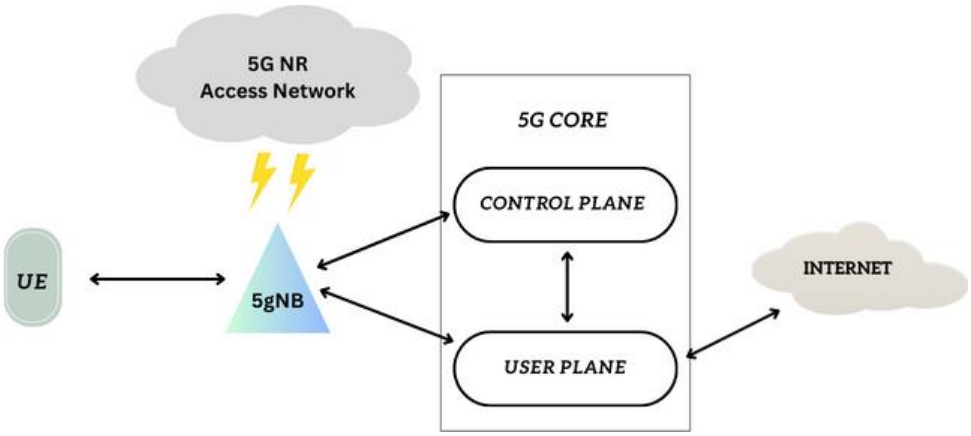

Figure 10: Control and user plane separation in 5G core

Via the interface, the control plane manages the path of information exchange for controlling and enabling the operation of the services. Therefore, efficiency, scalability and reliability determine the quality of service for mobile network operations, which associates with each UE registered within the network (Guttman & Ali, 2018).

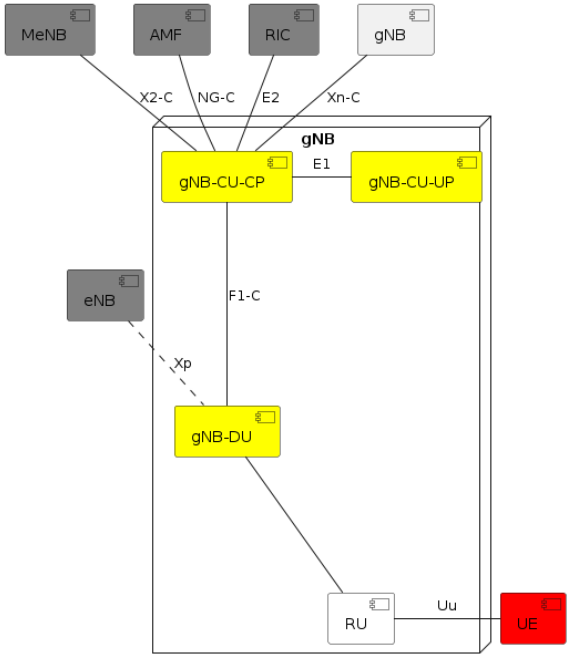

Figure 11: Control plane components overview (Nokia, 2023)

In the mobile telecommunication system, the control plane supports all signaling. CP connects not only to the user plane via the E1 interface but also connects to a distributed unit (DU) via the F1 interface. Through the radio unit (RU), radio signals convert to digital signals for further transmission into a DU and further transmits the signals into the central unit (CU) -CP (Nokia, 2023).

2.4 Cp-rt Component

The Cp-rt stands for Control Plane Real Time in the 5G gNB, which serves as a pivotal element within the architecture, orchestrating real-time control functions essential for the efficient and dynamic operation of the radio access network. According to a 3GPP definition, Cp-rt constitutes a CP implementation within a DU, which means a logical node of the DU includes Cp-rt functions and its operation is controlled by a CU. The deployment of Cp-rt retains merely one in each DU (Nokia, 2023). The critical role of Cp-rt is to manage signaling, connection establishment and maintenance to establish seamless communication between UE and network elements. To ensure communication with other elements in a RAN, different interfaces are used. The F1AP interface enables communication with the Control Plane (CP); L1/L2 interfaces handle

communication with L1/L2-RT and L2-NRT instances; and through the CpRt::Cm interface, CP-RT communicates with Node O&M instances (Nokia Cp-rt design, 2023).

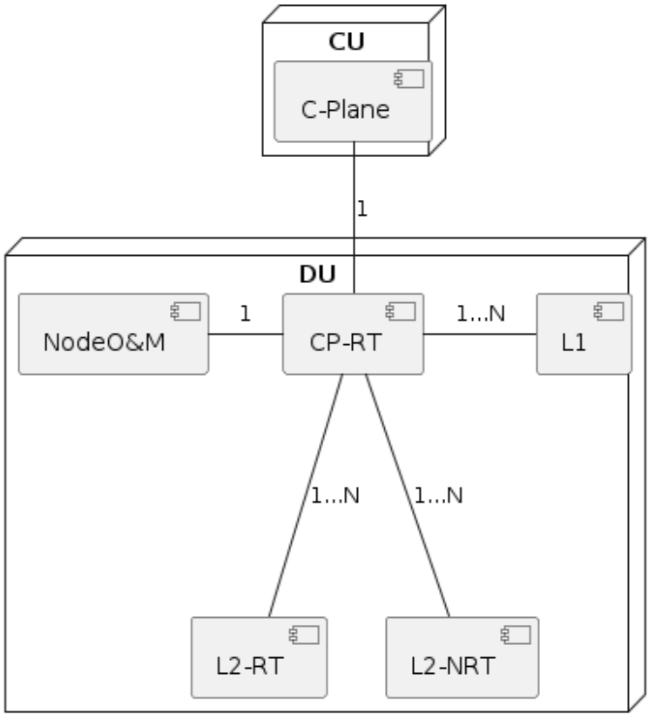

Figure 12: CP-RT system overview (Nokia, 2023)

When investigating in detail, the Cp-rt component is composed of its subcomponents, which can deploy duties to relevant components for best performance. ***CPRTCOMM*** as a fundamental component performs in mainly four areas: Architecture, Capacity/Performance, Common Services and General administration. It provides and supports, for example, the threading models, common library, performance monitoring, internal common services, PM-counting and troubleshooting. Nevertheless, the other components handle more specific management tasks, for example, the ***CPRTOAM/CPRTCELL*** component mainly focuses on configuration that concerns cell management and other interfaces or links, while the ***CPRTRM*** is responsible for radio resource management and admission control. Furthermore, the ***CPRTUE*** component conducts the UE management such as bearer setup/modification or UE specific configuration to L2-PS/L2-LO/L2-Hi-DU and more. (Nokia Cp-rt design, 2023).

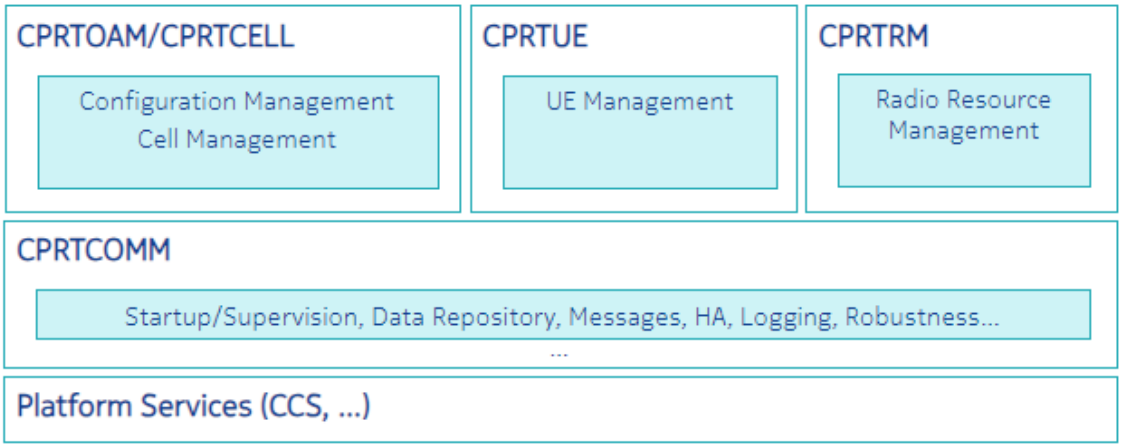

Figure 13: CPRT subcomponent view (Nokia Cp-rt design, 2023)

The Cp-rt component is as a multi-threaded application in the CP of a 5G system, which involves controlling and managing concurrent threads to handle different tasks concurrently. The initialization of the application takes place through the main thread, which activates CpIfDuThread that handle the communication with F1AP Sctp (Stream Control Transmission Protocol) and starts CpRtThread that handles common or cell level procedures. However, it remains idle after initialization during the lifetime of Cp-rt. Cp-rtThread linking to UP signaling and CpRtCm service triggers the other threads such as CpRtUeThread that handles UE level procedures. Both CpRtTraceMgmtThread and CpRtBeThread are background running threads. The former interfaces with RCP trace controller, the latter executes high CPU consuming tasks (Nokia, 2023). The multi-threaded implementation conducted through different interfaces delivers the signal or message concurrently into various services to trigger relevant activations to provide reliability and low latency to UE.

The Cp-rt component deployed in DU but implemented in CP in a gNB system is a crucial element that empowers the network to operate with agility and efficiency. It provides specific services with efficient resource utilization and ensures responsiveness and adaptability for the diverse and dynamic communication in the 5G network.

2.5 Emil and Endeavour

To ensure the functionalities of the development of different features, the testing process is a necessary procedure to examine and maintain the functionalities in the system scope. Emil is a Radio Network diagnostics tool and collects internal and external messages for eNB and performs message analysis to create call scenarios. Also, its features are applied within the 5G network for troubleshooting purposes. Messages sent through Emil, for example, explore the trace and source ID, the duration of calls and network cells. (Emil Training, 2017).

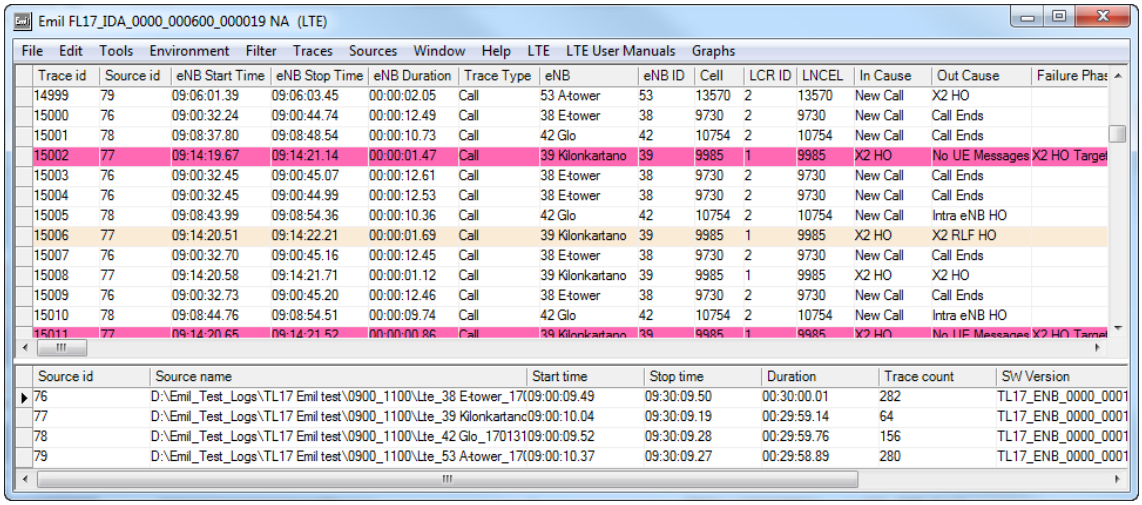

Figure 14: Emil diagnostic tool and messages exploration (Emil Training, 2017)

Endeavour is a test framework that is based on a graphical user interface (GUI). It has been developed inhouse for 3G technology since 2011 and it is an extension to the Emil logging tool (Nokia Emil & Endeavor, 2022). Endeavour was implemented not only in multi-site organization to test many interfaces, but also in 5G Megazone OAM development for SCT and PIT (Nokia Endeavour Introduction, 2022). Several principles and features developed in 3G technology were also incorporated into the development of 5G (Nokia Endeavour PoC Status, 2022). In contrast to code-based SCT, Endeavour presents a message sequence chart that is easy to understand from a software testing perspective (Nokia Endeavour Introduction, 2022).

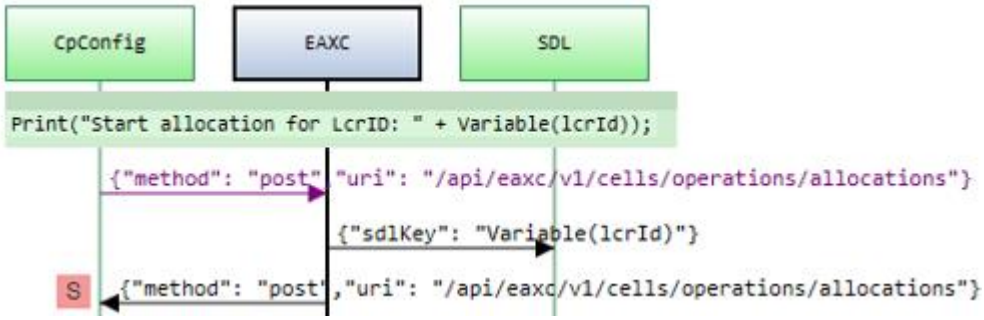

Figure 15: Message sequence chart based SW testing tool

2.6 Endeavour Testing Environment

Endeavour becomes a proprietary tool in the case company. It is developed in C++, but its GUI interface is programmed by C# that is compiled to Intermediate language code (IL) by an open-source compiler. As part of Emil, Endeavour also utilizes some components of Emil like sack, monitoring files and user C# script compilation. While executed, Just-in-Time (JIT) compiles IL code to machine code (Nokia Endeavor architecture and functionality, 2022).

The architecture of Endeavour testing environment is a complicated structure, including a user interface, the Endeavour server and the Endeavour connector. The supported messages are given in three formats, which are Emil sack that is encoded/decoded by types.dat, JavaScript Object Notation (JSON) and the Extensible Markup Language (XML) format (Nokia Endeavour, 2022). The Endeavour server is the major component for implementing testing logic, which de/activates the tested processes as a broker between Endeavour and the tested process or the system under test (SUT).

Due to the role of the Cp-rt component in the c-plane system, the messages delivered to the SUT are in either one message format of Protobuf, ASN.1 and MuLTI (Multi Language Toolkit for Interfaces) formats. The protocols could be used in either one of ZmqProxy, STCP, and SysCom (System Internal Communication Protocols) protocols. ZmqProxy is an embeddable networking library, which is a concurrency framework carrying atomic messages across various transports such as TCP (ZeroMQ, 2023). Sctp is an IP transport layer

protocol for transmitting multiple streams of data simultaneously between two endpoints (TechTarget, 2000-2024). SysCom is an internal protocol used BTS-wide for control and management signaling (Nokia SysCom Introduction, 2018). Furthermore, the communication between the server and connector follows the TCP protocol, which stores various TCP server ports and TCP sockets, as shown in Table 1 representing message size for each field (Nokia Endeavour, 2022).

Table 1: Initial setup of TCP in Endeavour

Field	Size in Message
Meassage_total_size	4 bytes
Message_id	2 bytes
Number_of_REAL_pids	2 bytes
Pid (board, cpu, task, padding)	6 bytes * number_of_real_pids
Number_of_SUT_pids	2 bytes
Pid (board, cpu, task, padding)	6 bytes * number_of_sut_pids
Number_of_FAKE_pids	2 bytes
Pid (board, cpu, task, padding)	6 bytes * number_of_fake_pids
Test case name string length	4 bytes
Test case name string	(Test_case_name_string_length) bytes

All the JSON formatted messages are transferred through a dedicated port to the Endeavour server over the TCP protocol with all the attached headers, bodies and parameters. The Cp-rt component can utilize these formatted data into serialized messages and EndavourConnector then converts them into the JSON format to communicate between the server and connector in Endeavour environment. For example, the connector converts JSON to byte array by Protobuf (Protocol Buffers). This is done for handling or converting the byte array to JSON to display the message in a human readable form in the Endeavour GUI, while sending the JSON validation result as a monitored message to Endeavour interface to present the successfulness also over the TCP protocol. The JSON files created by Endeavour are the dominated data

type to be encoded or decoded according to certain formats between server and connector.

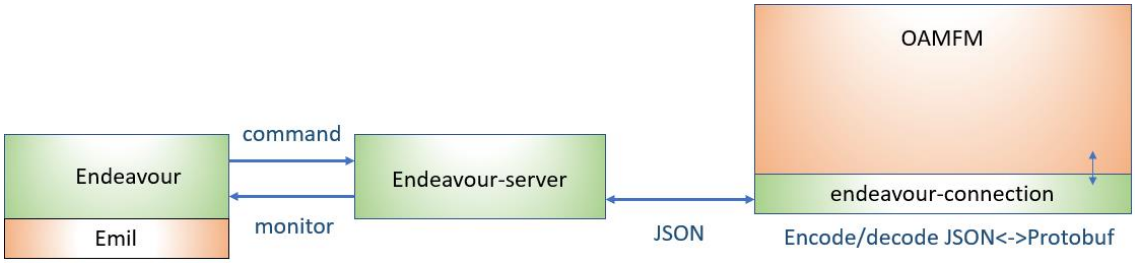

Figure 16: Endeavour test environment (Nokia Endeavour, 2022)

Within the Endeavour testing environment, the formatted stream signals delivered to the Cp-rt component serialize message forwarding to Endeavour server, and the Endeavour connector converts the messages to JSON and communicates over the TCP protocol within the environment to display messages in a human-readable form in the Endeavour GUI for indicating complete messaging circles for testing purposes.

3 Project Method Design Process

The purpose of this study was to design, develop and analyze the viability of the internal Endeavour tool as extra support in the 5G system for optimizing message tracing communication. The study was conducted using an inductive approach with the assumption that it is possible to convert messages into a format by using Endeavour converter to display a message sequence chart for the Cp-rt component. The assumption was presumed to be feasible resulting from the other two components (cp-ue and cp-e2), the messages of which have been converted using the Endeavour converter and illustrated as a sequential messages chart.

3.1 Software Testing

The software development of any project should not neglect the role of software testing. Testing is a set of activities with the objective of identifying failures in a

software or a system and to evaluate the level of quality to obtain information about user satisfaction. A set of techniques and methods is typically used to enable decisions for software delivery to users, while taking into account the objectives of cost reduction, time to market and risks (Homes, 2012).

Testing is a crucial phase in the software development life cycle. It allows analyzing the risks and identifying the failures strategically along with the software development models. Unit tests (UT) traditionally follow software development. Unit testing means testing a unit as the smallest piece of code that can be logically isolated in a system. Testing a unit could mean testing any individual component of software such as a line of code, function and class (SmartBear, 2024). In the final year project, unit testing was carried out using the Nokia discipline of conducting testing. This included using the Gtest framework, which means Google C++ testing and mocking framework (GoogleTest, 2023).

3.2 Structure Design of Endeavour

The codebase of implementation contains several key sections, which include the Endeavour converter, the Endeavour server and the Endeavour workspace. The server, linked with the referred libraries and directories from *BuildDir* executes the *System-Under-Test* source code to run request test cases that demonstrate sequential messages flows in the workspace. The illustrated message chart mainly reflects the functionality of the connector/converter, which the loaded message from the component is converted into *Json* format, serialized accordingly, and linked to different interfaces to explicitly display visibility.

In the code, most of the referred libraries of the server and converter shared identical source code. The preprocessor directives are compiled to conditionally include parts of the source code that is necessary for either the server or converter to implement the task.

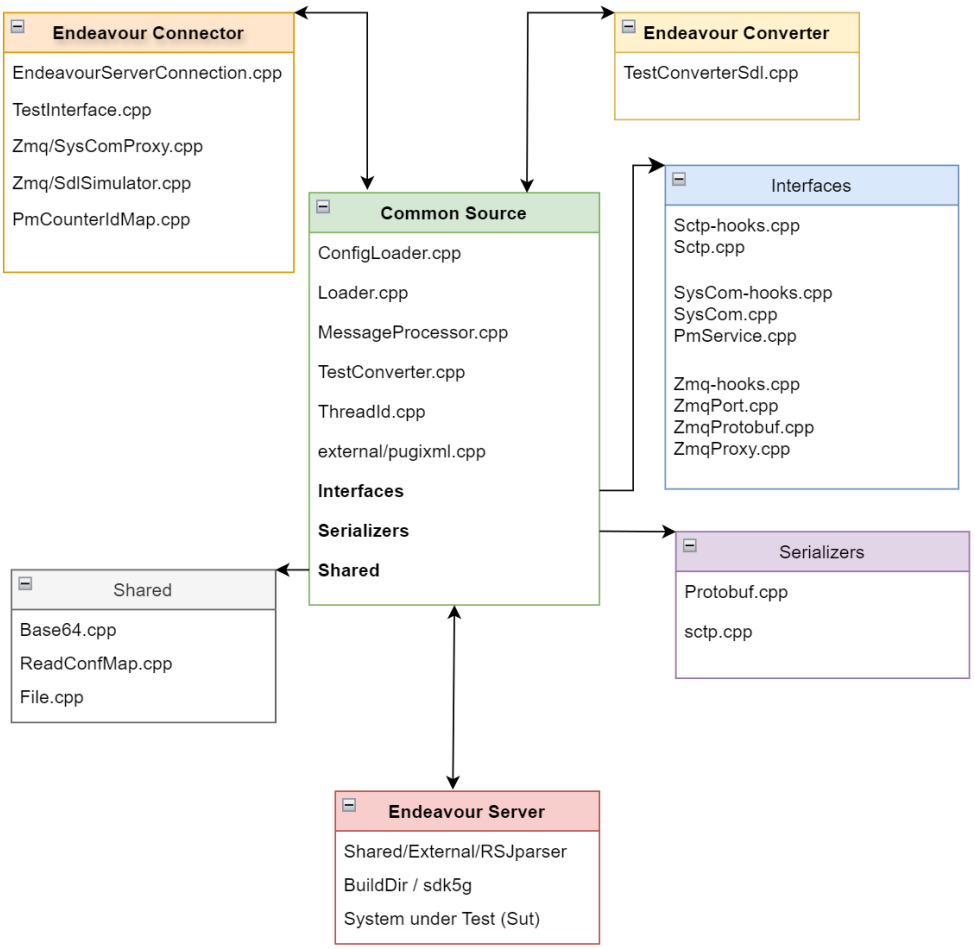

Figure 17: Codebase Structure of Endeavour Implementation

The structure of the codebase can efficiently utilize the various libraries for both the server and converter and revision of any source code would be efficiently applied to both sections, which reduces the cumbersome development and possibility of any mistakes.

4 Project Implementation

4.1 Cp-rt Component Attachment

The viability of tracing communication using Endeavour functions across different components of the 5G system is demontrated, and many existing functionalities are also applicable to other components, which increases the complexity of development on the code base. The intended component needs to attach to the system, the *Components.x* adds a sequential hex number

X(0x302, "Cprt" sv) and *run.sh* is written to preload the relevant libraries and to execute the test case of Cp-rt in Endeavour. The Endeavour server, therefore, should be able to identify the component and to build the relevant libraries into the system. The *build_endueavour_server.sh* modification should become a necessary task to create the implementation for Cp-rt to be identified.

```
if [-f/workspace/build/native/cprt/sct/bin/Cprt]; then
  echo -e "Found ${COLOR_GREEN}CP-RT${END_COLOR}"
  CMAKE_CONFIG_OPTIONS="$CMAKE_CONFIG_OPTIONS -DCP-RT=ON"
else
  CMAKE_CONFIG_OPTIONS="$CMAKE_CONFIG_OPTIONS -DCP-RT=OFF"
fi
```

Listing 1. Script for attaching the Cp-rt component

Any Cp-rt test case run on Docker can be identified by the Endeavour system. The case should indicate the name of the component after building the server of Endeavour. Subsequently the test case will be loaded into the *EndeavourPoC.ews* file to store the meta-data and create the *BtsProcessInstanceMap* corresponding to the specific projects. However, because there is no direct support for Cp-rt, an immediate error may be detected by the system as a runtime error or undefined behavior especially if it fails to differentiate *Sut Wrapper* from other components. A possible solution is to modify the *convert_ttcn3_sct.sh* script to verify the running component with the name of test case.

```
params=()
for arg in "$@"; do
  params+=("$arg")
done
pushd /workspace >/dev/null
run_build(){
    local script="$1"
     shift
    if [ -z "${DOCKER_IMAGE_VERSION}" ]; then
      runon5gdocker env SCT_SUT_WRAPPER="$ENDEAVOUR_SERVER_PATH/EndeavourConnector/$script" ./gnb_build/build.py $@
    else
      SCT_SUT_WRAPPER="$ENDEAVOUR_SERVER_PATH/EndeavourConnector/$script" ./gnb_build/build.py $@
    fi
}
for param in "${params[@]}"; do
  case "$param" in
        "cpue")
          run_build "Ttcn3SutWrapperCu.sh" "${@:1}"
          ;;
        "cprt")
          run_build "Ttcn3SutWrapperCprt.sh" "${@:1}"
          ;;
    esac
done
```

Listing 2. Verifying the Ttcn3Wrapper of component

To make the Endeavour converter perform as expected, the TTCN3- SCT test case needs to be executed, which will create an executable file for Endeavour to convert message data into XML. The development of both scripts could support Endeavour to identify the Cp-rt component and select the responsible *sutwrapper* script to build. With the early version of modification, although the system could identify the component, there are no actual processes to implement further, and at this point, further development will create specific functionalities merely for Cp-rt components.

4.2 Preparation for Integration

The requirement of this system is to trace the message flow of communication for the Cp-rt component. The architecture of 5G technology indicates the critical role of the DU component that determines the communication scope and approach that could differ dramatically from other components. The implementation of Endeavour for Cp-rt shall be defined against the other components in the CU such as cp-ue.

Before proceeding with further conversion, it is important to consider the functionality and role of the gnb-DU components in the 5G system. The existing *CmakeList.txt* file originally designed for gnb-CU components no longer supports both gnb-CU and gnb-DU components. As a result, the corresponding *CmakeList.txt* file will be modified and refactored to separate the source and/or header files for each unit, enabling pre-compilation. Moreover, the created shell script *Ttcn3SutWrapperCprt.sh* will identify the component to utilize the *LD_PRELOAD* environment variable to load relevant libraries for further processing.

To support converting test cases that inject EndeavourConnector through *LD_PRELOAD* without its own API for managing storage and component lifetimes, global variables could be a solution for accessing its internal state in hooked functions. The *ConfigLoader* header file has to define function calls precisely for the gnb-CU and gnb-DU to create a global instance, to obtain components and sdl-addresses via the *configLoader* namespace, which only requests for Cp-rt components.

```
namespace configLoader {

void createGlobalInstance();
void destroyGlobalInstance();

const std::string& getSdlAddress();
const std::string& getComponent();

#ifdef ENDEAVOUR_CU
const std::string& getPort(const std::string& key);
const std::string& getRndAddress();
const std::string& getSysComProxyAddress();
#endif

#ifdef ENDEAVOUR
std::uint32_t getSutTaskId();
#endif}
```

Listing 3. Creating global instance and configuring loader for Cp-rt

4.3 MuLTI Formed Messages

The Cp-rt component involves controlling and managing multiple threads and handles different communications concurrently, which indicates the Endeavour system would manipulate multiple messaging protocols through the various interfaces respectively.

The encoding functionalities of messaging protocols can be supported in the system through their hooked interfaces. For instance, with the F1-Ap interface, it can handle the communication of the Sctp protocol. Moreover, the Cp-rt component manipulates the formatted *.mt* files that are parsed by the MuLTI tool to generate protocol *.pt* files. Each message needs to be assigned a unique message ID, and message name and type. The namespaces are used to distinguish the data type from each other, which allows MuLTI to have the folder structure matching the namespace. To make the namespace unique, if enclosing namespaces are enabled in end- and test-points, MuLTI will move the generated code into an artificial namespace. The root level of the namespace is derived from the filename of the pg-file, and following the name of the end- and test-point, the namespace structure is given as in pt- and mt-files (Nokia MuLTI, 2023).

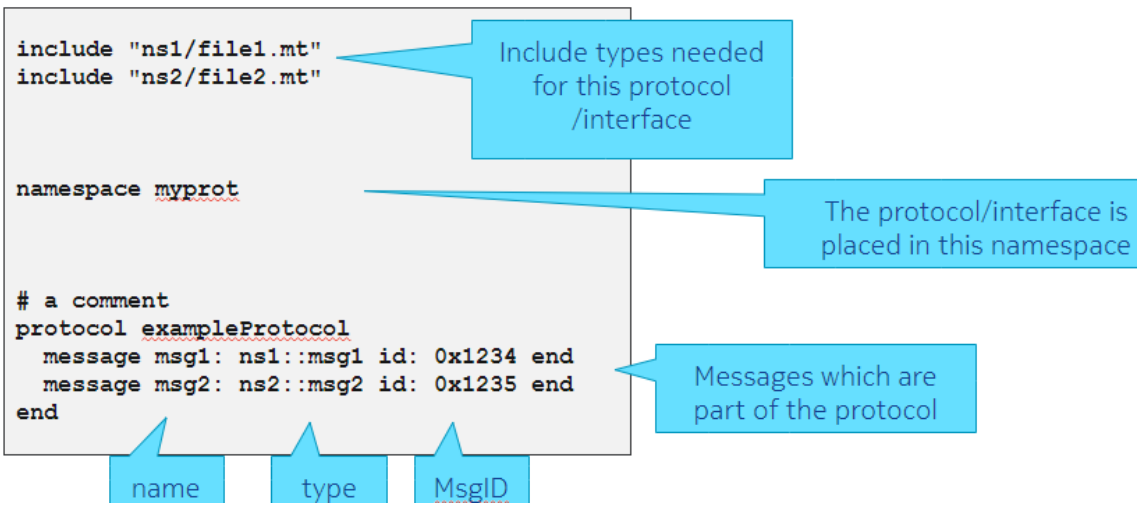

Figure 18: MuLTI formatted .pt protocol file

4.4 Multiple Headers Generation

The key data in the protocol *.pt* files will be manipulated within the SysCom interface, and the message name and ID must be retrieved from these files, which are distinctive by their unique namespace. The solution to retrieve the data is to create a supporting *python* script *generateMultiHeaders.py*, which can read the *.pt* files via a file path as an argument through a *for* loop. While looping, each line of the file can be scanned and key words like *messageSpace, messageName* and *messageId* can be found for extracting and putting message datas into the *messages[ ]* container. While the entire file is scanned, the container will collect all key elements needed to create *SysCom* type and *messageId* as a pair into *msgMap{ }* collector. The script also contains several functions such as *searchMsgIdById()*, *searchMsgIdByName()*, *matchMsgId()*, *getMsgId()* and *getMsgName()*, to provide the possibility of handling the messages in each header. Then the name of an output followed by each name of the component will be generated in the *multiHeaders* folder that contains many various headers files in the user plane and control plane.

```
namespace CPUECPRT {
constexpr TAaSysComMsgId CpRt_ReceiveFromCpUeId {0xDE30};
constexpr TAaSysComMsgId CpRt_SendToCpUeId {0xDE31};

const std::unordered_map<TAaSysComMsgId, std::string_view> msgMap {
      {CpRt_ReceiveFromCpUeId, "CpRt_ReceiveFromCpUe"},
      {CpRt_SendToCpUeId, "CpRt_SendToCpUe"},
      };
}
```

Listing 4. Header files generated by Python script

To reduce the complexity for users and generate these headers automatically while compiling, the decision is to integrate the existing .pt-files in the gnb repository into the *CMakeLists.txt* file. The most relevant .pt-files path has been listed in a *Multi_Protocols* list, and each file will be executed through the path by the *foreach* loop and the name of component will be extracted using *REGEX* to

generate the file name. Furthermore, *add_custom_command()* directs the output and contains the Python command to trigger the execution of *generateMultiHeaders.py* at run-time. Then, the list of generated header files assists Endeavour to identify each message ID after running the test case.

```
foreach(MULTI_PROTOCOL ${MULTI_PROTOCOLS})
 string(REGEX REPLACE "/workspace/itf/l2/" "" OUTPUT_NAME ${MULTI_PROTOCOL})
 string(REGEX REPLACE "/protocol.pt" "" OUTPUT_AGAIN ${OUTPUT_NAME})
 string(REGEX REPLACE "/" "" OUTPUT_FINAL ${OUTPUT_AGAIN})
 string(STRIP ${OUTPUT_FINAL} HEADER_SOURCE)
 add_custom_command(
    OUTPUT ${CMAKE_CURRENT_BINARY_DIR}/${HEADER_SOURCE}.hpp
    DEPENDS ${MULTI_PROTOCOL}
    COMMAND python ${GENERATE_SCRIPT} ${CMAKE_CURRENT_SOURCE_DIR}/multiHeaders ${MULTI_PROTOCOL}
    WORKING_DIRECTORY ${CMAKE_CURRENT_BINARY_DIR}
 )
 list(APPEND HEADER_LISTS ${CMAKE_CURRENT_BINARY_DIR}/${HEADER_SOURCE}.hpp)
endforeach()
```

Listing 5. CmakeList.txt file

4.5 Leveraging the Scripts

The *generateMultiHeaders.py* script not only creates the necessary header files for each component that communicates with Cp-rt but also adds functions that can extract the message name or ID from the header files to assist the SysCom serializer function. However, the generated files include approximately 40 headers, and it becomes cumbersome to generate identical functions to each header file, which could be problematic for future maintenance of the code base.

An idea for optimization is to simplify the function of the script and to preserve the reading functionality and to define the message type. The goal is to form pairs in the *msgMap{ }* container to match each message name with its unique ID. The leveraging process is to write the other helper program to utilize these header files to support requested functions instead of overlapping identical functionalities in each header file.

The helper program *syscomCprtHelper.hpp* includes all generated header files during pre-compilation and creates the new *allMsgMaps* vector *std::vector <std::unordered_map<TAaSysComMsgId, std::string_view>> allMsgMaps { }*. Each element in this vector is an unordered-map container representing message ID and its corresponding message name. This container generates the operation base for retrieving message names and IDs across different namespaces.

```
std::optional<std::pair<TAaSysComMsgId, std::string_view>> getMessagePair(const msgIdOrName param){
  for(const auto& msgMap : allMsgMaps){
    if(std::holds_alternative<TAaSysComMsgId>(param)){
      TAaSysComMsgId msgId = std::get<TAaSysComMsgId>(param);
      auto it = msgMap.find(msgId);
      if(it != msgMap.end())  return std::make_pair(it->first, it->second);
    }else if(std::holds_alternative<std::string_view>(param)){
      std::string_view msgName = std::get<std::string_view>(param);
      for (const auto& [id, message] : msgMap){
        if(message == msgName) return std::make_pair(id, msgName);
      }
    }
  }
  return std::nullopt;
}
```

Listing 6. Syscom helper program for retrieving message data

The *getMessagePair()* function is designed to search through a collection of *allMsgMaps* to find a matching message based on either a message ID or a message name. When this function *std::optional<std::pair<TAaSysComMsgId, std::string_view>> getMessagePair(const msgIdOrName param)* is called (*see Listing 6*), it returns an *std::optional* type containing a pair of *std::pair of TAaSysComMsgId and std::string_view*; otherwise returning *std::nullopt*.

The condition *std::holds_alternative<TAaSysComMsgId>(param)* checks *param* is of the type of *TAaSysComMsgId,* and retrieves the *TASysComMsgId* value using the *std::get<TAaSysComMsgId>(param)* function. It also searches for this ID from *msgMap* as so to return a pair of the ID and corresponding message. Alternatively, the condition *std::holds_alternative<std::string_view>(param)* can

also investigate if the *param* is of the type *std::string_view* and retrieves *std::string_view* using the same function and iterates over *msgMap* to find the message matched *msgName*. Then, it returns a pair of the ID and the message name.

4.6 Handling Unknown Messages

With these new changes in place, although it has passed the case testing, the outcome in Endeavour still indicates many unknown messages that have not been identified or defined. To in-depth study the messages, it has been found out that a certain number of messages while communicating with Cp-rt were created or identified at run-time on the SDK5g folder.

```
[EndeavourConnector] Unsupported SysCom message id 0x2414 (9236) - 76 bytes,
payload: 4d 1d 00 00 00 00 00 00 02 00 00 00 14 00 00 00 01 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 19: Unknown message

To handle these unknown messages in the system, the resolution of the manipulation follows three processes. The first process aims to find the location of message IDs from the SDK5g folder, and then write another Python script to extract the key values to create *.pt* files; the second process is to take advantage of header-generator script to create corresponding header files; to update the CMakelist file to ensure the implementing sequence of these two Python scripts will be the third process to manipulate.

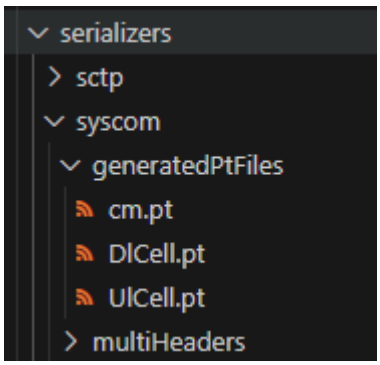

Figure 20: New generated protocols

The protocol generator as the other Python script is composed of three sections that play separate key roles to achieve the purpose. The function *extract_msg_info( )* defines *msgNamePattern* and *msgIdPattern* from each line of a given file. The function can scan the line of files and extract the *msgName* and *msgId* matching with the pattern, also converting the message ID into HEX format.

On the other hand, *generate_pt_files( )* takes the defined grouped list and output directory as parameters to iterate over each group and its associated files in *groupedFiles* to call the *extract_msg_info( )* function to extract the message name and ID. When both *msgName* and *msgId* are successfully extracted, both name and ID will append a dictionary to the messages list and eventually prints a message indicating that the protocol file has been generated.

The last part of the code processes a list of file paths and groups them based on specific substrings and generates protocol files for each group. This section contains the main function that calls the *generate_pt_files ( )* function to generate protocol files in the corresponding directory.

The unknown messages in SDK5g are designed to be manipulated using the script during pre-compilation, which needs to add new commands and executed file paths to the *CMakeList.txt* file and place them in the generatedPtFiles folder. This approach can ensure the protocol files of unknown messages are generated prior to implementing the header script.

4.7 Encoding to Base64 Format

The message serialization aims to enhance readability, allowing users to view the data and its values communicated between Cp-rt and other components. To test the capability of conversion, it is necessary to investigate if the development could encode and format messages in the system. With the support of the *getMessagePair( )* function based on created code, less

development is required since existing functions in *syscom.cpp* in order not to break the link between the existing code and other components.

To include *getMessagePair( )* in *getSycomMsgIdFromName( )* and *getSysComMsgNameFromId( )* by adopting their parameters, it is possible to filter out the requested message information and return corresponding data for further handling. Using the *SerializedMsg syscom::serialize* function, *matchMessageId( )* can check the *if* condition to determine whether the message ID under SUT falls into the scope of Cp-rt components to implement base64-format encoding, which can simply be in the XML file since the messages information and data are converted to *EndeavourWorkspace* as XML file.

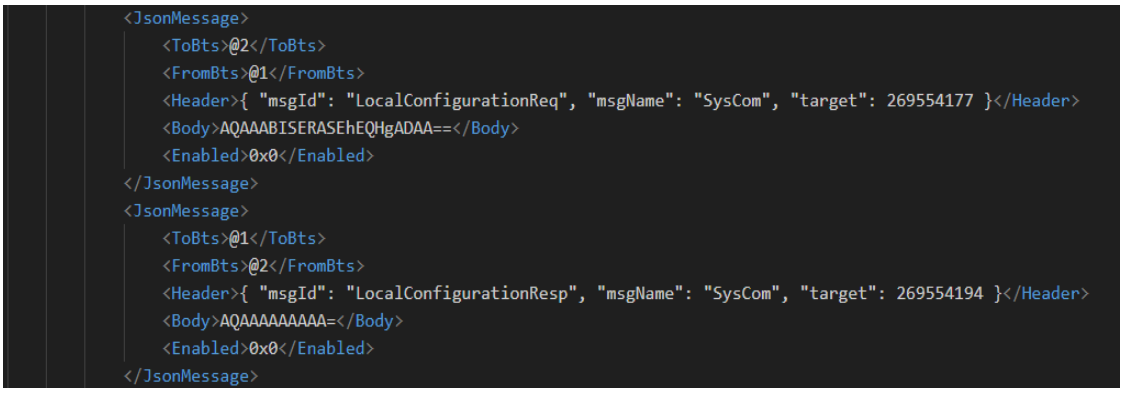

Figure 21:Message display in XML file

5. Interface Implement for Cp-rt

Although the system is supposed to handle various messages, the configuration setup and protocols from the DU component could differ dramatically from the CU, which might result in the reconsideration of software architecture or modification of the existing interface implementation.

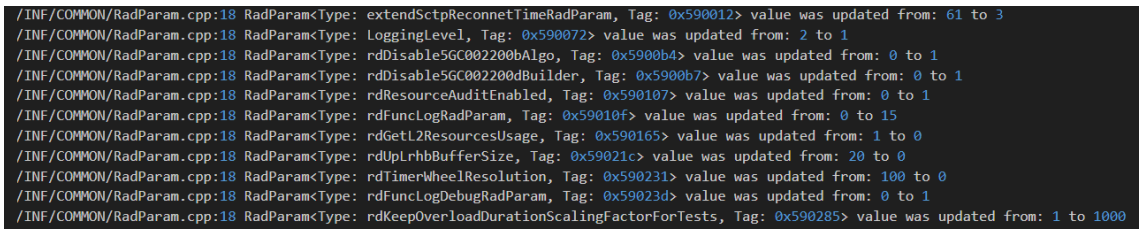

Figure 22:Messages log from Cp-rt

Figure 22 indicates that the system requests certain new RadParams in the configuration process. While *TestInterface* handles the new message commands in *DummyInterface.cpp* to identify the new event ID for configuring and updating RadParams in the runtime. However, this should not break the existing configuration for CU components such as cpue. Then, it is necessary to create a new separate container and new related functionality to check whether the *envTestRadParams* container holds the new parameters. After this, the updated value inserts into the command line.

The topology of the 5G system for the DU component illustrates that Cp-rt inter-communicates with UPlane and CPlane through different interfaces. Therefore, SysCom becomes the dominant protocol to handle the majority of the Cp-rt messages. Furthermore, the interface should be re-designed and developed to serve both CU and DU components.

Therefore, the implementation of the SysCom interface needs further development to sequence the message flow between EndeavourConnector and the SUT, which can help resolve the timing lag issue. While the SUT sends *NOTIFY_READY*, EndeavourConnector will then send the ready message to EndeavourSever through the *EndeavourServerConnection::connectionThreadTask* function to activate a process, in which the received messages are processed, sequenced and displayed on the system's graphical interface to demonstrate the message flow.

6. Operation Analysis and Future Development

The system utility for Cp-rt operates together with TTCN-3 and is layered on top of it even though they are in many ways doing completely different tasks, especially with the way the Endeavour GUI and EndeavourServer are in the middle.

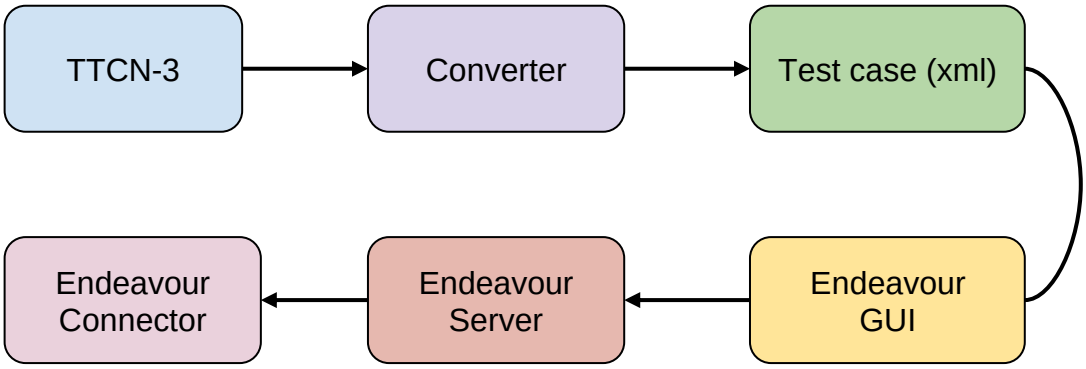

Figure 23: Endeavour operation diagram

The purpose of this project is to integrate the message flow of Cp-rt components into the program and provide a generic mechanism for tracing messages not only in the CU but also in the DU components, establishing a foundation for message tracing that can be easily implemented and further developed in the future.

Due to the role of Cp-rt, the messages could differ from the layers of the 5G system, but they could also vary within various functional sections. The number of messages could be several times greater than that of a single component in Cp-ue. However, the functionalities of the system simply provide the foundation for serializing a large amount of messages from low level to high level of Cplane, enabling each single message to be converted with its proper ID into an XML file, which the system can then process further through the graphical user interface.

6.1 Synchronizing Codebase with Other CU Components

The code base of the system for Cp-rt to some extent can’t be isolated. Furthermore, it should be synchronized with other components in the same code base framework, which has brought a few challenges in understanding its

architecture but also in creating a generic handling base for all various components.

During the implementation, it has been found out that the existing functionalities would be necessary to be revised or modified. This would enable handling performance with the generic mechanism not only in the CU but also in the DU. For the Sctp messages for instance, while the messages were being received, the function *message::SerializedMsg sctp::serializeAsn()* using static buffer that could not dynamically adjust its size, which caused the errors for handling Cp-rt messages. Adapted to generic principle, this container of function should be able to manipulate all messages from Cp-ue and Cp-rt regardless of the number of messages. In order to have such a condition check, the looping functionality for the size was implemented and a macro was used to separate the E2ap and F1ap interface messages to determine whether handling was for Cp-ue or Cp-rt.

Therefore, these modifications do not only happen in a single function; the development extends to all shared functionalities, from simple separation to creating a generic mechanism for message tracing applicable to multiple components.

6.2 Usage Restrictions and Future Development

The goal of software development is to maximize the possibility of generalization and make it applicable to various circumstances. The current software configuration and implementation are built based on a few SCT test cases that could set up the system foundation for future development and micro-modification while considering the massive amount of the test cases of Cp-rt. The more test cases there are running under the system, the fewer problems or new issues occur, which is also a request for further improvement. Moreover, future improvement and development should also be applicable to the interface of Cp-rt, even though the messages have been handled at some point. The optimizable operation on the interface of the system could be developed and improved further.

The purpose of the Endeavour system is ultimately to trace the messages in the SCT test case, demonstrating a clarified communication and message flow. Additionally, details of system information, including key parameters, should be displayed in a human-readable format when looking up the information.

Figure 24: Single MuLTI message on Endeavour

The messages for Cp-rt are formatted using the MuLTI, which is created and used only by the case company. This restricts the possibility to parse the MuLTI messages into JSON format or to reverse them back. Tool MuLTI formating allocates the data into different layers, including the static part and the dynamic part. The backbone structure for MuLTI would be the arrays or nested dynamic variable sized arrays. The static part is allocated to a total of 16 bytes, and for each data element a total 12 bytes will dynamically allocated. Therefore, the architecture of the MuLTI encoded messages becomes a complex procedure that could make parsing the message difficult and reversing the JSON messages back to proper MuLTI format more complicated. This could be studied further in the future. Therefore, parsing messages such as payload in Figure 24, between the MuLTI and JSON messages is not included in this study. The Figure illustrates that the messages were only encoded with the base64 format.

7. Conclusion

The rapid evolution in 5G technology comes along with a complex architecture and high-performance demands. The traits of high throughput, ultra-reliable low latency and massive communications demand efficient message tracing and debugging that becomes an essential task for maintaining network reliability. This study focuses on the development of an internal message tracing tool and establishes the foundation of integrating the Cp-rt component into the framework, which ultimately could enable message sequence tracing efficiency for future improvement in the 5G system.

To integrate the Cp-rt, the study follows a structured software design approach that aligns with existing architecture. This design ensures efficient communication between components, enabling structured message processing and traceability. The modifications in the study allow Cp-rt to be fully incorporated into this framework, ensuring compatibility with existing testing methodologies.

The goal of integrating the DU component is achieved by several critical milestones. The configuration modification enables it to recognize Cp-rt as a traceable component, and support for multiple message formats ensures compatible message conversion. By scanning runtime-generated logs, it is able to dynamically detect and categorize undefined messages and to convert them into structured protocol files. Improvements were made to the SysCom interface to ensure efficient sequencing and synchronization of Cp-rt messages within the framework. These changes ensure that messages are properly processed and visualized.

The developed foundation integrates Cp-rt into the framework. Synchronization with CU and DU components requires further improvement to ensure compatibility across different 5G layers. On the other hand, the complexity of MuLTI messages limitation poses a challenge for broader system interoperability. Future study should explore the parsing techniques and enhance automation in message handling to optimize test case execution.

Increasing the number of test cases and refining interface structures will also improve system robustness.

The study lays the groundwork for further improvements in message handling, automation, and interface optimization. While challenges remain, the study could provide a solid foundation for future development, ensuring that Endeavour can keep supporting evolving 5G technologies and contribute to more efficient telecom network diagnostics.

References

Penttinen, J. (2019) 5G Explained: Security and Deployment of Advanced Mobile Communications. ProQuest [Online] Available from: https://ebookcentral.proquest.com/lib/metropolia-ebooks/detail.action?docID=5721174 (Accessed 27 Nov. 2023)

IEEE (2018) 3GPP Release 15 Overview 3rd Generation Partnership Project (3GPP) members meet regularly to collaborate and create cellular communications standards, IEEE Spectrum Available from: https://spectrum.ieee.org/3gpp-release-15-overview (Accessed 27 Nov. 2023)

3GPP (2013) TTCN-3 Available from: http://www.ttcn-3.org/index.php/about/why-use-ttcn3 (Accessed 29 Nov. 2023)

Nokia Solution & Networks Oy (2022) 'Endeavour PoC presentation and demo for BOAM LT - 20220617' Internal Document, (Accessed 2 Dec. 2023)

Sivakumar D., Mohamad Khairul Anuar bin Mohd Nasir, and Ragu A/L Rajanendaran (2017) International Journal of Information System and Engineering, A Case Study Review: 5G on Future Malaysia Industry, Vol. 5. (Accessed 18 Dec. 2023)

Ahtisham Bhatti (n.d.) Advanced Computer Network Lab, A Survey Report on "Generation of Network: 1G, 2G, 3G, 4G, 5G" (Accessed 20 Dec.2023)

Ericsson (2022) Ericsson Mobility Report. Available from: Ericsson Mobility Report November 2020 (Accessed 20 Dec. 2023)

3GPP A global Initiative (2023) Available from: 5G System Overview (3gpp.org) (Accessed 21 Dec. 2023)

Nokia Solution & Networks Oy (2022) Nokia Bell Labs 5G Foundation|BL00100-K-0220, Internal Document (Accessed 10 April 2023)

Erunkulu, O. O., Zungeru, A. M., Lebekwe, C. K., Mosalaosi, M., and Chuma, J. M. (2021). "5G Mobile Communication Applications: A Survey and Comparison of Use Cases". In: IEEE Access 9, pp. 97251–97295. issn: 21693536. doi: 10.1109/ACCESS.2021.3093213.

Ericsson (2021) 5G evolution toward 5G advanced: An overview of 3GPP releases 17 and 18. Available from: Toward 5G Advanced: overview of 3GPP releases 17 & 18 - Ericsson (Accessed 21 Dec. 2023)

Ericsson (2020) 5 key facts about 5G radio access networks. Available from: 5 Key Facts About 5G Radio Access Networks (ericsson.com) (Accessed 27 Dec. 2023)

Jesse Hollington (2023) What is 5G? Speeds, Coverage, Comparisons and more. Available from: What is 5G? Speeds, coverage, comparisons, and more | Digital Trends (Accessed 28 Dec. 2023)

David Kennedy (2019) The Facts on 5G - How 5G networks are being built in the real world. Ovum Consulting (Accessed 27 Dec. 2023)

3GPP Technical Specification (2018) 5G; System Architecture for the 5G System (3GPP TS 23.501 Version 15.2.0 Release 15), document ETSI TS 123.501, pp. 4-220, vol.15

3GPP Technical Report (2019) Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); LTE; 5G; Release description; Release 15, document 3GPP TR 21.915, V15.0.0, 2019, pp. 1–53.

F.W.Vook, A.Ghosh, E.Diarte and M.Murphy (2018) 5G new radio: Overview and performance, in Proc. 52nd Asilomar Conference Signals, Systems and Computers, pp. 1247-1251

G.Soos, D. Ficzere, P. Varga, and Z.Szalay (2020) Practical 5G KPI measurement results on a non-standalone architecture, in Proc. IEEE/IFIP Netw. Operations Manage. Symp.(NOMS), pp.1-5

Cisco (2021) 5G Non Standalone Solution Overview Available from: 5G Non-standalone Solution Guide, StarOS Release 21.25 - 5G Non Standalone Solution Overview [Cisco ASR 5000 Series] - Cisco (Accessed 03 Jan. 2024)

B. Bertenyi, R. Burbidge, G. Masini, S. Sirotkin and Y. Gao (2018), NG radio access network (NG-RAN), J. ICT Stand,. Vol. 6, nos. 1-2, pp. 59-76

C. Zhang, X. Wen, L. Wang, Z.Lu and L. Ma (2018) Performance evaluation of candidate protocol stack for service-based interfaces in 5G core network, in Proc. IEEE int. Conf. Commun. Workshop (ICC workshops), pp. 1-6

Erik Guttman, Irfan Ali (2018), Path to 5G: A Control Plane Perspective, Journal of ICT, Vol. 6_1 & 2, 87-100.

Nokia Solution & Networks Oy (2023), 5G C-Plane Software Architecture Document, Overall C-Plane architecture, Internal Document (Accessed 8 Jan. 2024)

Nokia Solution & Networks Oy (2023), 5G C-Plane Software Architecture Document, 5G-CP-RT- Design Document, Internal Document (Accessed 11 Jan. 2024)

Nokia Solution & Networks Oy (2023), MuLTI Document, Getting Started with MuLTI, Internal Document (Accessed 15 Oct. 2024)

ETSI (2013) TTCN-3: The Global Testing Language, ETSI The standards People, Available from: TTCN3_Q32022_050822.pdf (ttcn-3.org) (Accessed 15 Jan. 2024)

Nokia Solution & Networks Oy (2022), EMIL and Endeavour History, Endeavour PoC material, Internal Document (Accessed 17 Jan. 2024)

Nokia Solution & Networks Oy (2017), Emil Training Internal Material (Accessed 22 Jan. 2024)

Nokia Solution & Networks Oy (2022), Endeavour PoC presentation and demo for BOAM LT-20220617, Endeavour Introduction, Internal Document (Accessed 22 Jan. 2024)

Nokia Solution & Network Oy (2022), Endeavour PoC Status Overall Summary, SCT Endeavour_Status, Internal Document (Accessed 22 Jan. 2024)

Nokia Solution & Network Oy (2022), Endeavour architecture and functionality, Internal Document (Accessed 23 Jan. 2024)

Nokia Solution & Network Oy (2022) Endeavour Plan- Endeavour Interfaces, Internal Document (Accessed 23 Jan. 2024)

ZeroMQ (2023) An open-source universal messaging library, Available from: https://zeromq.org/ (Accessed 25 Jan. 2024)

TechTarget (2000-2024) What is the Stream Transmission Control Protocol? Available from: https://www.techtarget.com/searchnetworking/definition/SCTP (Accessed 29 Jan. 2024)

Nokia Solution & Network Oy (2018) SysCom Introduction, Internal Document (Accessed 29 Jan. 2024)

Homes Bernard (2012) Fundamentals of Software Testing, Publisher: Wiley-ISTE, [Online], Available from: https://ebookcentral.proquest.com/lib/metropolia-ebooks/detail.action?docID=1120766 (Accessed 12 Feb. 2024)

SmartBear (2024) What Is Unit Testing? Available from: https://smartbear.com/learn/automated-testing/what-is-unit-testing/ (Accessed 12 Feb. 2024)

Google (2023) GoogleTest User's Guide, 1.14.0/August 2. 2023, Google Test Documentation. Available from: https://google.github.io/googletest/primer.html (Accessed 12 Feb. 2024)

Appendix

Headers Generator

The header files generator script was written in the Python programming language during the project. The script is shown in the listing below.

```
import sys

messages=[]
index= 5

#List of filenames passed as arguments to program
outDir = sys.argv[1]
files = sys.argv[2:]
for filename in files:
    with open(filename, "r") as file:
        for line in file:
            line = line.strip()
            if line.startswith("protocol"):
                messageSpace = line.split()[1]
            if line.startswith("message"):
                messageName = line.split()[1][:-1]
                splt_line = line.split()
                if(len(splt_line) > index):
                    messageId = splt_line[4]
                else:
                    messageId = splt_line[3][3:]
                messages.append({
                    "group": messageSpace,
                    "name":messageName,
                    "id":messageId })

outFileName = messageSpace
with open (outDir + "/" + outFileName + ".hpp", "w") as file:
    file.write("// Copyright 2024 Nokia. All rights reserved.\n")
    file.write("// Generated Header File\n\n")
    file.write("#include <IfAaSysCom_Defs.h>\n")
    file.write("#include <string_view>\n")
    file.write("#include <stdexcept>\n")
    file.write("#include <unordered_map>\n")
    file.write("#include <spdlog/fmt/bundled/core.h>\n\n")
    file.write("namespace %s {\n\n" %(messageSpace))
    for message in messages:
        file.write("\tconstexpr TAaSysComMsgId %sId
{%s};\n" %(message["name"], message["id"]))
    file.write("\n\n")
    file.write("\tconst std::unordered_map<TAaSysComMsgId, std::string_view>
msgMap {\n")
    for message in messages:
        file.write("\t\t{%sId, \"%s\"},\n" %(message["name"],
message["name"]))
    file.write("\t};\n\n")
    file.write("}")
```

Protocols Generator

The protocol generator script was written in the Python programming language during the project. The script is shown in the listing below.

```
import sys
import os
import re

def extract_msg_info(file_path):
    msgNamePattern = r'class\s+(\w+);'
    msgIdPattern =
r'static\s+constexpr\s+uint16_t\s+msgId\(\)\s*\{\s*return\s+(\d+u)\s*;\s*\}'

    with open(file_path, "r") as file:
        content = file.read()
        msgNameMatch = re.search(msgNamePattern, content)
        msgIdMatch = re.search(msgIdPattern, content)
        if msgNameMatch and msgIdMatch:
            msgName = msgNameMatch.group(1)
            msgId = hex(int(msgIdMatch.group(1)[:-1]))
            return msgName, msgId
    return None, None

def generate_pt_files(groupedFiles, outDir):
    for group, files in groupedFiles.items():
        messages = []
        for file in files:
            msgName, msgId = extract_msg_info(file)
            if msgName and msgId:
                messages.append({
                    "id": msgId,
                    "name": msgName})
        if messages:
            outPath = os.path.join(outDir, "{}.pt".format(group))
            with open(outPath, "w") as file:
                file.write("// Copyright 2024 Nokia. All rights reserved.\n")
                file.write("// Generated Header File\n\n\n")
                file.write("namespace Cprt::%s\n\n" %(group))
                file.write("protocol %s\n" %(group))
                for message in messages:
                    file.write("\tmessage %s:\t\t%s_t \t\tid:%s
         end\n" %(message["name"], message["name"], message["id"]))
                file.write("\nend")

outDir = sys.argv[1]
filePath = sys.argv[2:]
groupedFiles = {"cm": [], "DlCell": [], "UlCell": []}
for path in filePath:
    if "/cm" in path:
        groupedFiles["cm"].append(path)
    elif "/DlCell" in path:
        groupedFiles["DlCell"].append(path)
    elif "/UlCell" in path:
        groupedFiles["UlCell"].append(path)
    else:
        print("unmatched path {}".format(path))
generate_pt_files(groupedFiles, outDir)
```