Master Individual Project

Determining the Compatibility of the Network Functions of the 5G Core Implementations -Open Air Interface and Open5GS

Submitted by: Deblina Karmakar
Examiner: Prof. Dr.-Ing Ulrich Trick
Date of start: 15.08.2023
Date of submission: 15.02.2024

Statement

I confirm that I have written this thesis on my own. No other sources were used except those referenced. Content which is taken literally or analogously from published or unpublished sources is identified as such. The drawings or figures of this work have been created by myself or are provided with an appropriate reference. This work has not been submitted in the same or similar form or to any other examination board.

15/02/2024,

_____

Date, signature of the student

Content

1 Introduction

The landscape of wireless communication is undergoing unprecedented changes, driven by an escalating demand for connectivity in our increasingly interconnected world. At the forefront of this transformative wave is 5G, a revolutionary mobile communication technology that facilitates seamless communication between a multitude of objects. In this era where connectivity is a fundamental necessity, the guiding principle is evident: "The fabric of modern society is woven with threads of connection, binding everyone and everything together"

5th generation mobile network is a new global wireless standard after 1G, 2G, 3G, and 4G networks. 5G enables a new kind of network that is designed to connect virtually everyone and everything together including machines, objects and devices. 5G wireless technology is meant to deliver higher multi-gigabits per second peak data speeds, ultra low latency, more reliability, massive network capacity, increased availability, and a more uniform user experience to more users. Higher performance and improved efficiency empower new user experiences and connects new industries. [1]

The deployment and design of networks are significantly changed by 5G networks from a static to a fully dynamic perspective. However, while we prepare standards and technologies for future use cases and applications, we also need to align with the state-of-the-art beyond 5G networks in order to establish the foundation for such dynamic networks. [2]

The 3rd Generation Partnership Project (3GPP) is a collaborative project between a group of telecommunications associations with the initial goal of developing globally applicable specifications for third-generation (3G) mobile systems. 3GPP was officially formed in December 1998 when ETSI partnered with different standards and telecom organizations with the goal of developing 3G networks. [3]. 5G made its debut in this release 15 which came in three phases and generally focused on the core network architecture. The first phase focused primarily on mobile broadband for NSA 5G architecture. The second phase introduced SA 5G architecture. The third phase introduced architectures for migration from 4G to 5G. NSA refers to leveraging the LTE core network for control functions and adding 5G radio access to the existing 4G base stations, allowing devices to connect to both networks simultaneously. However, data traffic for 5G in NSA was routed through the 4G core, leading to potential limitations or suboptimal performance for certain 5G features. On the other hand, SA architecture represents a fully independent 5G network with its own core network separate from 4G. This approach delivers superior performance, lower latencies, enhanced network slicing capabilities, and better support for emerging use cases like IoT and ultra-reliable communication.

In 2019, several 5G implementation initiatives got underway globally, adhering to the Third Generation Partnership Project (3GPP) standard. Two of the most popular open source projects are OpenAirInterface(OAI) and Open5gs. OAI is run by the OpenAirInterface Software Alliance (OSA), a French non-profit organisation called "Fonds De Dotation" which was founded in 2014 and is funded by corporate sponsors. On the other hand, Open5gs is run by an admin named Sukchan Lee and sponsored by several financial contributors including Telnet, Voicenter, Coral Telecom etc. Both of these projects aims to provide a platform which facilitates the features of 5G core network that are 3GPP complaint.

The goal of the project is to utilise network functions and test interoperability using different core networks provided by different open source projects. The study focuses on the communication between network functions on an integrated platform. The workings of every component within the 5GCN can be understood with this realization of the 5G network. This study is beneficial for ensuring interoperability, which is crucial for creating a diverse and interoperable 5G ecosystem. Ensuring compatibility also encourages the use of diverse components, reducing dependency on a single project. The whole experiment is being carried out on an open-source Linux operating system platform. This report will go into more detail on the system architecture, implementation strategy, challenges with integration, performance assessment, and discussion of the results.

2 Theoretical Background

The project aims to establish a 5G SA environment engaging 5G core network and NG-RAN (gNB). It also build a compatible environment for network functions to work together. Therefore, a background investigation on every potential part and operation of this implementation is essential. This chapter offers the theoretical underpinnings for every part and technique utilized in this project's implementation (which is covered in Chapter 4).

2.1 5G use cases and usage scenarios

The continuing growth in demand from subscribers for better mobile broadband experiences is encouraging the industry to look ahead at how networks can be readied to meet future extreme capacity. The procedure on the way to 5G differed and still differs significantly from that of previous generations of mobile networks, including 3G and 4G. While in the past, the focus was on communication between and services for people, it is now on providing a networked world for everyone and everything, i.e., not only for people but also for (smart) things and systems. The approach is no longer primarily technology- driven, like up to and including 4G, but use case-driven. Based on a large number of possible use cases, the requirements were derived, and the technology required for implementation was specified [4].

This process was used in a number of projects and organizations amongst them three of the most popular names are project METIS (Mobile and wireless communications Enablers for the Twenty-twenty Information Society), ITU(International Telecommunications Union) and 3GPP (3rd Generation Partnership Project) [35].

The International Telecommunications Union (ITU) is a specialized agency of the United Nations responsible for many matters related to information and communication technologies. The ITU has outlined the broad goals and requirements for the development of 5G technology. In ITU's IMT-2020 (International Mobile Telecommunications 2020) initiative, the ITU has defined three main use cases for 5G, categorized into Enhanced Mobile Broadband (eMBB), Ultra-Reliable Low Latency Communications (URLLC), and Massive Machine Type Communications (mMTC). This particular project started in 2015 and produced over 70 use cases, which were first divided into five crucial categories before being reduced to three. They are distinguished by the specific performance characteristics, although there is some overlap [36]. The figure 2.1 illustrates what is known as a 5G triangle. The descriptions of use cases are as follows:

2.1.1 Enhanced Mobile Braodband (eMBB)

This “addresses” the human-centric use cases for access to multi-media content, services and data. Based on the original scope of the SMARTER (“Study on New Services and Markets Technology Enablers”) approach, related requirements have been defined on high data rates, higher traffic or connection density, high user mobility, and those relevant to various deployment and coverage scenarios. This group of use cases is characterized by broadband data access across a wide coverage area in crowded locations, office areas, and high-speed public transport systems. The target is to provide maximum user experience by providing connectivity both indoors and outdoors while delivering high QoS broadband even in challenging network conditions. This use case falls into the categories of Multi-user interaction, Augmented Reality, and Context Recognition [6]. Some further use cases in this area are as follows:

Hot Spots – Broadband Access in Dense Areas :

This use case relates to providing enhanced broadband access in densely populated areas such as high-rise building complexes, urban city centres, crowded areas, and etcetera. Moderate mobility and high data rates will be required.[6]

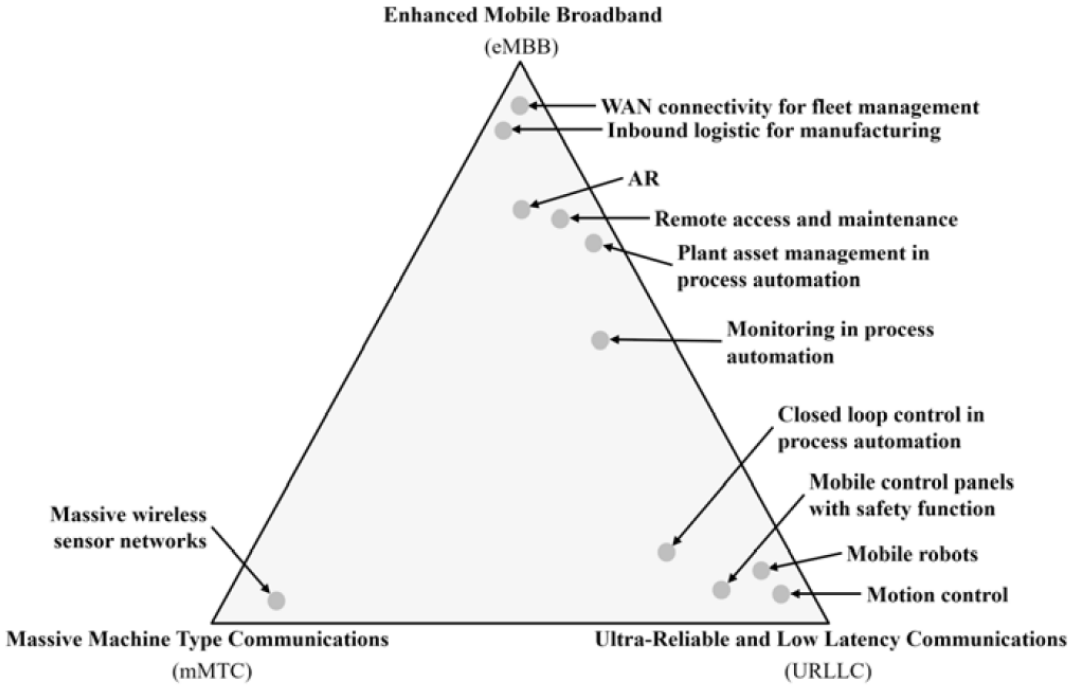


Figure 2.1:5G-ACIA use cases and ITU-R/3GPP usage categories [4]

General Broadband Everywhere:
This use case relates to providing a consistent user experience, guaranteeing user speeds of 50+ Mbps everywhere towards a mobile and a connected society. The user data has to be delivered consistently across the coverage area. High mobility will be required.[6]

Public Transport:
This use case is about providing broadband access to public transport systems such as high-speed trains. The use case consists of providing robust communication link and high-quality Internet for information, entertainment, interaction or work with a high mobility component[6]

Smart Offices :
This use case in characterized by heavy data use in an indoor environment that will require low mobility. This is a use case scenario where hundreds of users require ultra-high bandwidth to serve intense bandwidth applications.[6]

2.1.2 Massive Machine Type Communication (mMTC)

mMTC addresses the challenges encountered in machine-type communication. mMTC uses new communication models that are unlike human-type communication models. The application systems supported by massive machine-type communication prioritize low-rate, uplink-centric transmission, which is different from high-speed, downlink-based, human-type communication.The mMTC use cases give utmost importance to small packet data transmission techniques rather than high-speed, large-volume data transmission. Innovative access strategies are

present in massive machine-type communication, which prevents the collisions that can arise if traditional human-type communication-based schemes are used. The mMTC service can be tailored to combine random access and scheduling strategies when there are thousands of IoT devices waiting for access [7]. Some of the most significant use cases are listed below:

Connected Vehicles:
Wireless connectivity enables new functionalities and services in the automotive industry. Automotive applications optimize both machine and human communication. The mMTC use case in automobiles enables the exchange of data between machines inside the vehicle or with servers or user devices. mMTC use cases in the automotive industry include traffic efficiency, road safety, remote diagnostics, and control.

Smart Grids:
The smart grid is a complex cyber-physical system where energy production is decentralized. There is real-time control and coordination to match the energy supply with demand. The mMTC service provided by 5G wireless technology promotes the establishment of a reliable smart grid. This smart grid, which is a use case of mMTC, is also capable of sending commands and polls.

Factory Automation:
By enabling wireless connectivity between various movable machines present in an integrated control system, it is possible to reduce downtime, shut downs, and monetary losses during manufacturing. The application of 5G mMTC over a wired connection helps engineers lower installation costs and eliminate the clutter and weight issues from cables. The most common application of factory automation is the real-time control of sensors and actuators in a closed loop.

Massive machine-type communication (mMTC) services enabled by 5G technology are integral parts of use cases such as factory automation, autonomous vehicle control, smart buildings, smart city systems, smart grids, smart logistics, and geographically spread devices [7].

2.1.3 Ultra Reliable and Low Latency Communications (URLCC)

These are essential use cases which is related to IoT(Internet of Things) applications, that will have very high demands on reliability, availability and low latency with lower demands on the volume of data, but significantly higher business value. These use cases also fall into the category of mission-critical Machine-Type Communication (MTC). These application requires very low end-to-end latency going down to the millisecond level. These use cases and applications feature interactions across all categories, human-to-human, human-to-machine, and machine-to-machine[6]. Some notable use cases in this area are as follows:

Process Automation:
These use cases are centered on information integration and enabling process automation useful in the oil and gas, chemicals, energy and water industries. The application here covers the pumps, compressors, mixers, monitors of temperature, pressure, flow controllers etc [6].

Automated Factories:
These use cases involve communication transfers enabling time-critical factory automation that are required in many industries across a wide spectrum that includes metals, semiconductors, pharmaceuticals, electrical assembly, food and beverage, etcetera. The applications for the use cases fall into functions related to material handling, filing, labeling, palletizing, packaging, welding, stamping, cutting, metal forming, soldering, sorting, printing presses, web drawing, picking and placing etc [6].

Urgent Healthcare/ Remote Surgery:

These use cases are envisioned around applications that will conduct remote diagnosis and treatment. There is a need for remote patient monitoring and communications with devices measuring vital signs such as ECG, pulse, blood glucose, blood pressure, temperature, and etcetera. The remote treatment and response based on monitored data can be life critical for a patient, requiring immediate, automatic or semiautomatic response [6].

2.2 5G Architecture

3GPP's Release 15 defines the Fifth Generation of Mobile Telephony, or 5G or 5GS(5G System). It was functionally locked in June 2018 and had all of its specifications completed by September 2019. 5G system are consist of three elements- 5G Core (5GC), 5G base stations gNB and User Equipment(UE).
The primary component of the Next-Generation Radio Access Network (NG-RAN) is 5G base station gNB, where "g" stands for "5G" and "NB" refers to "Node B," the designation inherited from 3G onwards for the radio transmitter. It is commonly referred to as an "eNodeB" or "eNB" in 4G-LTE networks. gNodeBs provide the wireless connection between the user equipment (UE) and the core network. It performs tasks including radio resource management, mobility management, and data transmission. UE refers to the end-user‘s mobile devices, including smartphones, tablets, and IoT devices. It communicates with the infrastructure of the 5G network to access numerous services [4].

Another key component of 5G architecture is 5G Core (5GC), which is the heart of a 5G mobile network. It establishes reliable, secure connectivity to the network for end users and provides access to its services. The core domain handles a wide variety of essential functions in the mobile network, such as connectivity and mobility management, authentication and authorization, subscriber data management and policy management, among others. 5G Core network functions are completely software-based and designed as cloud-native, allowing higher deployment agility and flexibility on multiple cloud infrastructures [8]. Next Generation RAN (NG-RAN) that contains gNBs are connected to the 5GC and possibly also to each other. A gNB is divided into a gNB control unit (CU) and one or more gNB distributed units (DU) as shown in Figure 2.2 [4].

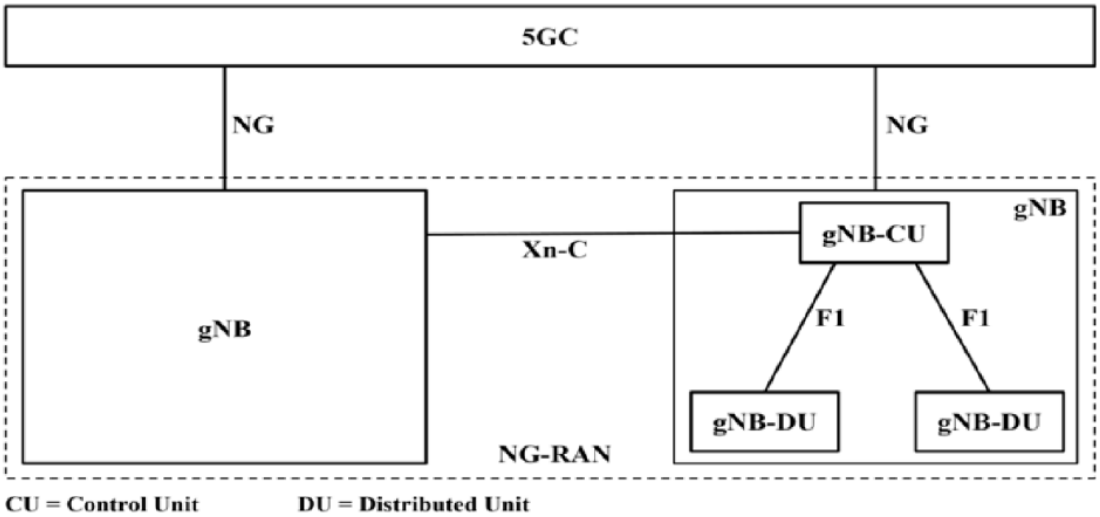


Figure 2.2: NG-RAN architecture with split gNB [4]

2.2.1 Service Base Architecture (SBA)

5G supports complex services relating to diverse use cases such as eMBB, mMTC, URLLC etc. To deliver these facilities Mobile Network Operators (MNOs) face significant challenges with the previous monolithic infrastructure. Thus, SBA is implemented due to the necessity of network transformation [8].

3GPP has defined 'Reference Point Architecture' and 'Service Based Architecture'. In "Reference Point Architecture" the Network Elements (NEs) use point-to-point interfaces to inter-connect each other. Each point-to-point interface has specified certain signaling protocols and procedures. In SBA, the set of Network Elements (NEs) are replaced with a set of Network Functions (NFs). Each Network Function can provide services to other Network Functions, i.e. each Network Function is a service provider. The point-to-point interfaces are replaced by a common bus which inter-connects all Network Functions. The services are specified for the Network Function providing them, instead of each pair providing and consuming Network [31]. Figure 2.3 described the above scenarios.

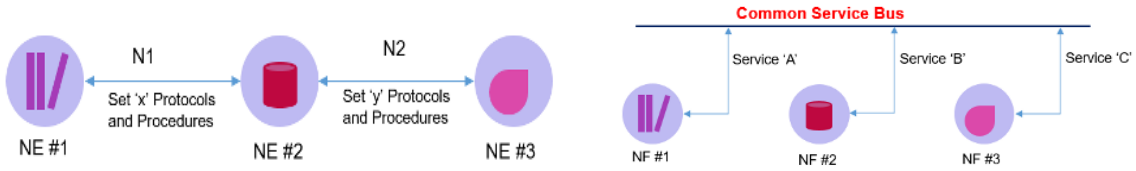


Figure 2.3 a): NE s in reference point architecture [31] *2.3 b): NFs in SBA [31]*

In Service Based Architecture (SBA) for the 5G Core, services are the essential building blocks used to create and execute the functions of a mobile network. NFs communicate with each other via a Service Based Interface (SBI). Each NF is containerized and operates independently, and exposes its functionality to other NFs through a SBI.[8]

As per 3GPP explanation, "That means wherever suitable the architecture elements are defined as network functions that offer their services via interfaces of a common framework to any network functions that are permitted to make use of these provided services."[8]

The Service Based Architecture of the 5G Core is a flat architecture that separates Control Plane (CP) functions from User Plane (UP) functions. The SBA infrastructure is fundamentally different from the Point to Point (P2P) model that preceded it because NFs exist independently and autonomously from one another rather than requiring dependencies that take time and personnel to maintain and optimize[8].

SBA utilizes the concept of network slicing within the 5G Core (5GC) to provide insights into the application scenario. The 5GC network is based on SBA as shown in Figure 2.3, with web-based Application Programming Interfaces (APIs) for control plane functions, which enables a more flexible and agile deployment of innovative services. [9].

3GPP selected OpenAPI version 3 (formerly known as Swagger) as the formal language to be used for the definition of APIs in the Service-Based Architecture. As indicated in 3GPP TS 29.501, each API is documented in a 3GPP technical specification (TS) that describes in natural language the API, and this specification also includes a normative Annex containing an OpenAPI description of the API [33]. Additionally, the use of APIs based on web-based technology makes development easy, as libraries, development tools, specification tools, code generators, security mechanisms and many other components are broadly available. It employed the new web protocol, HTTP/2 from IETF and designed interfaces according to the REpresentational State Transfer (REST) principle [32].

The main principle for the design of 3GPP API can be simply summarized as follows: “API should be designed as *RESTful* API whenever possible”. REST is an architectural style that imposes certain guiding principles to be satisfied by an API to be referred to as RESTful API. In short, the underlying server implementation is hidden by a layer of abstraction: service data are defined as "resource" uniquely identified by URI (Uniform Resource Identifier) and service consumers use HTTP methods to access to only a representation of resources. The representation can be delivered over HTTP using different file formats, JSON being the most popular one. For basic Create, Read, Update, and Delete (CRUD) operations on resource/service data, standard HTTP/2 methods (e.g. GET, POST, PUT, DELETE) are used to operate on resources uniquely identified by URIs [33].

Figure 2.4 shows the 5G system architecture in accordance with Service Based Architecture.

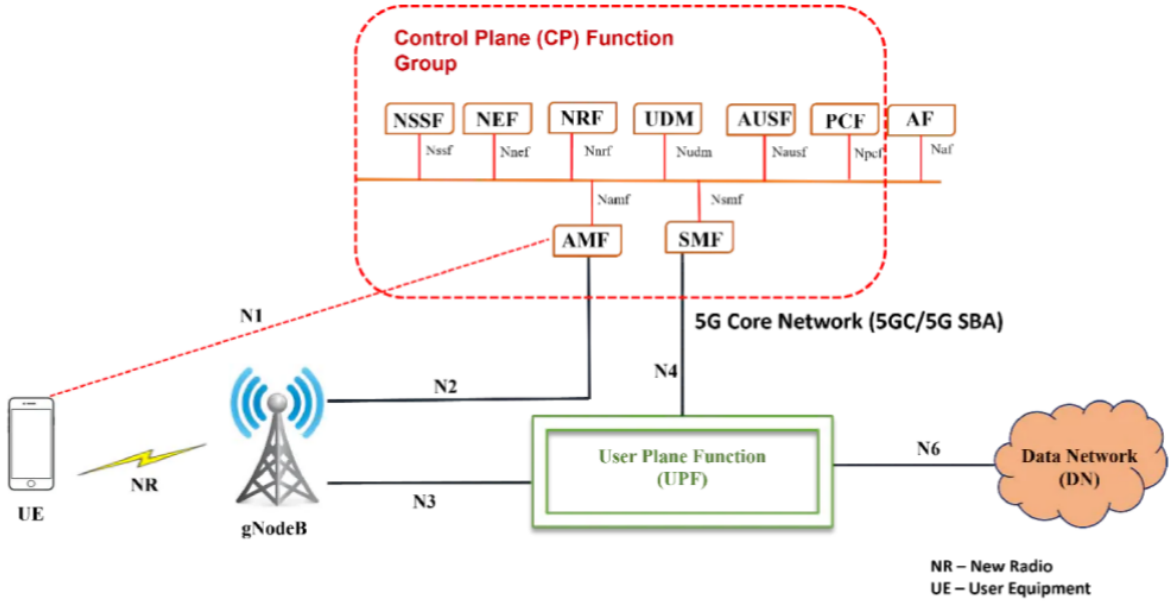


Figure 2.4: 5G system Architecture with Service-based Architecture[9]

2.2.2 SBA Protocols

The 3GPP has selected protocols for service-based interfaces in Release 15.

HTTP/2:
This is chosen as the application layer protocol. HTTP/2 is a binary protocol that brings performance improvements over its predecessor (HTTP/1.1) by introducing features like multiplexing and header compression.

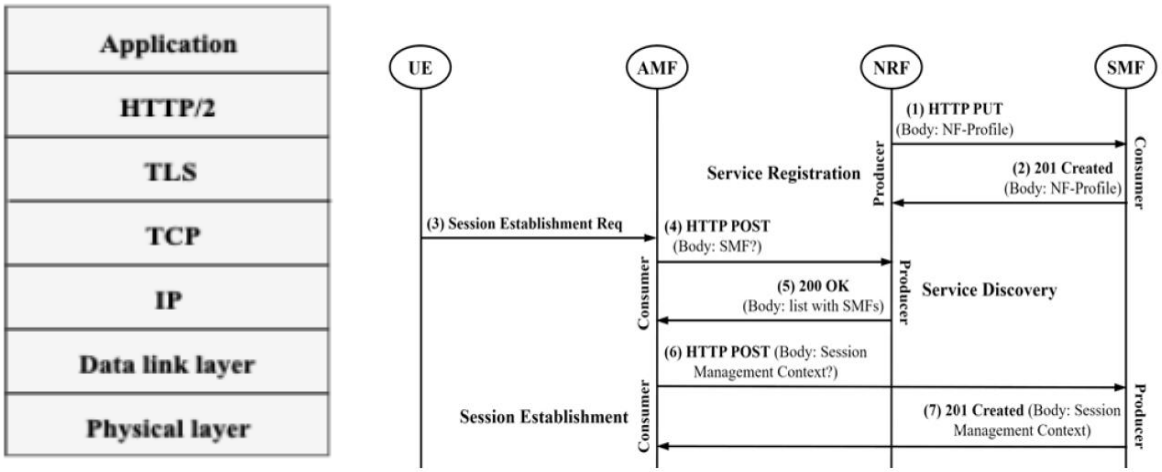
Transport Protocol:
Transmission Control Protocol is chosen as the transport protocol for HTTP/2. TCP provides reliable, connection-oriented communication.

Serialization Format:
JavaScript Object Notation(JSON) is selected as the serialization format. JSON is a lightweight and human-readable data interchange format.

API Design Style:
The API design style is RESTful wherever possible. REST (Representational State Transfer) is an architectural style for designing networked applications. It relies on a stateless, client-server communication model.

Interface Description Language:
OpenAPI, formerly known as Swagger, is used as the Interface Description Language (IDL). OpenAPI provides a standard way to describe RESTful APIs, allowing for easier documentation and code generation.

Figure 2.4(a) displays the basic SBA protocol stack that is in use. Applications are connected to the 5GC over a RESTful API. Figure 2.4(b) shows instances of how NFs can communicate with each other using RESTful APIs.



a) SBA Protocol stack *b) Interaction of NFs via RESTful APIs in the SBA*

Figure 2.4: SBA Protocol stack and RESTful API connection of applications to the 5GC via NEF[4]

2.2.3 5G NSA and 5G SA networks

Many carriers need to improve their radio access technology, which involves upgrading the cell and core from 4G to 5G, since they are transitioning to the 5G world. However, because 5G NR is being deployed quickly and 5G technology is still in its early stages, service providers frequently need to offer 5G alongside LTE. As a result, 3GPP has developed several strategies to enable a seamless upgrade from 4G LTE to 5G [10].

These can be categorized into two modes: Non-standalone mode (NSA) and standalone mode (SA). In the NSA, operators deploy 5G gNB nodes that connect to the existing 4G EPC core through 4G eNB nodes. In this architecture, there is no need for a separate 5G core since the 5G gNB nodes rely on the 4G core network for control information. This offers service providers a cost-effective and expeditious path to roll out 5G by leveraging their existing 4G infrastructure. Moreover, when a new 5G spectrum is introduced, it enhances the capacity and bandwidth within the network. 5G NSA offers certain advantages, such as Multi-RAT Dual Connectivity (MR-DC) for higher throughput [10], easy deployments, reduced cost etc.

In SA mode, all the eNB nodes are replaced by gNB, and the EPC is migrated to the 5G core. This enables the deployment of new services and uses cases, including smart factories, network slicing, and Voice over New Radio (VoNR). SA mode significantly enhances the end-user experience by providing lower latency, which is crucial for time-critical communications like factory automation and autonomous vehicle operation. Additionally, it improves

overall network efficiency[10]. SA requires MNOs to configure a completely new architecture and learn how to manage it.

Even though SA 5G is an improvement over NSA 5G, only few operators have deployed the latest technology. In September 2023, Counterpoint Research reported that only 47 MNOs worldwide have released commercial SA 5G deployments [11].

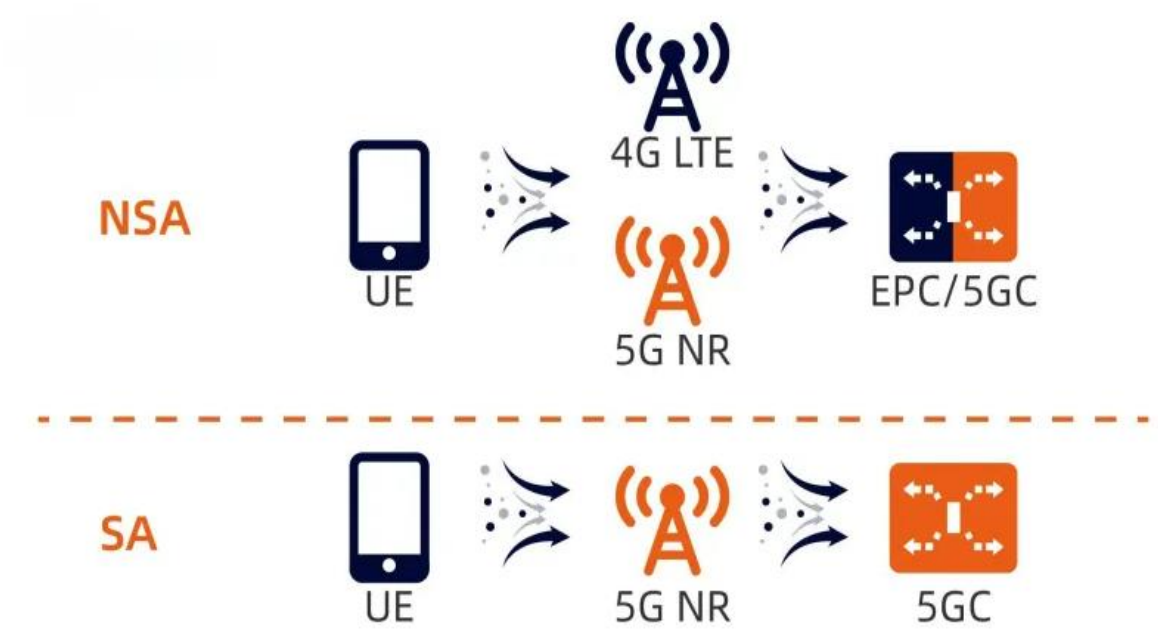


Figure 2.5: 5G NSA and SA architecture[12]

2.3 5GC Network Functions(NFs)

The project's goals are to evaluate the compatibility of various network functions and establish connectivity between NFs from different open source projects. Thus, having a clear understanding of how each and every NF function works is essential. A thorough summary of the same will be given in this section.

2.3.1 User Plane Function(UPF)

User Plane Function is a critical component of the 5G core network architecture It oversees the management of user data during the data transmission process. The UPF serves as a connection point between the RAN and the data network. It takes user data from the RAN and performs a variety of functions like as packet inspection, traffic routing, packet processing, and QoS enforcement before delivering it to the Data Network or Internet. This function allows the data plane to be shifted closer to the network edge, resulting in faster data rates and shorter latencies. Figure 2.6 describes the required protocols for UPF.[13]

Interfaces/reference points with employed protocols:

Interface between the UPF and the RAN (gNB) is N3 (GTP-U). Interface between two UPFs (the Intermediate I-UPF and the UPF Session Anchor) is represented by N9 (GTP-U). N6 (GTP-U) interface connects the Data Network (DN) to the UPF and N4 (PFCP) interface connects the UPF to the SMF [13].

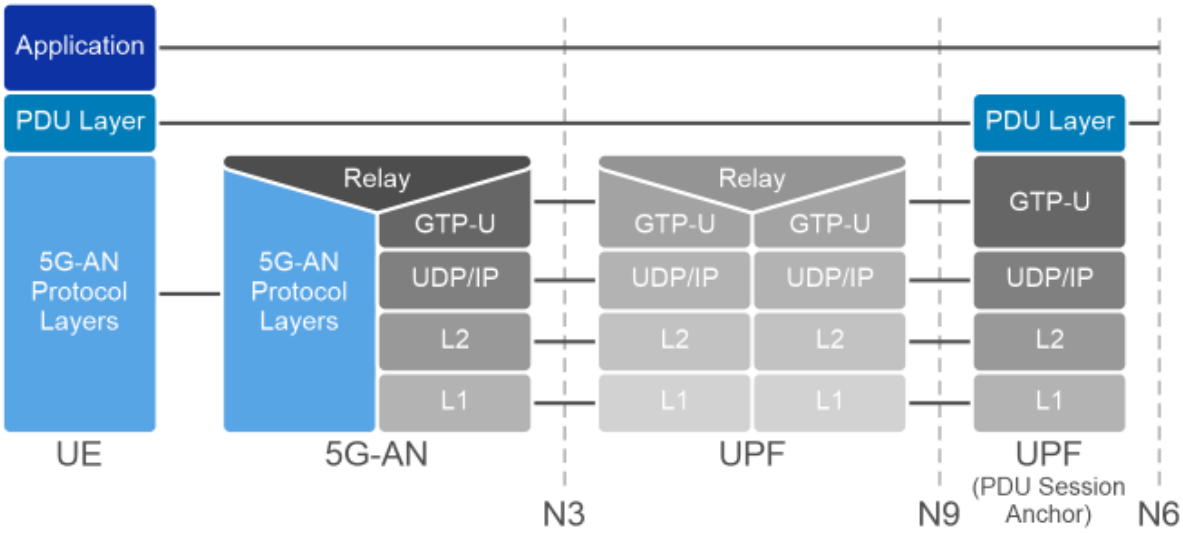


Figure 2.6: GTP-U Interface of UPF[13]

2.3.2 Access and Mobility Management Function (AMF)

It manages access and controls mobility. Connection and session information from end-user equipment or the RAN are received by AMF on N1 and N2 interfaces. Handling of session management tasks is forwarded to SMF. The AMF implements NAS ciphering and integrity protection algorithms by acting the part of the 5G core access point, thereby terminating RAN Control and EU traffic originating at either the N1 or N2 reference interface. The AMF sends an Authentication and Key Agreement (AKA) request in the same way that the MME in Evolved Packet Core (EPC) infrastructures does after receiving the initial NAS message. This occurs before the UE authorization process, in which the Unified Data Management (UDM) feature of the 5G Core Service Based Architecture (SBA) supersedes the Home Subscriber Server (HSS). This is done via N8's reference interface and thus uses the message bus HTTP/2 Service Based Interface (SBI). Finally, the AMF connects with Lawful Intercept (LI) systems to send relevant and appropriate event information (3GPP TS 23.501, 2021).

In addition, AMF performs a number of other crucial tasks for the 5GC network, two of which are covered in the section below:

Next Generation Application Protocol (NGAP):
NGAP is a Next-Generation Application Protocol that is found on the N2 reference point between the gNB and AMF. It is to support both UE and non-UE associated services. This includes operations such as configuration updates, UE context transfer, PDU session resource management, and also support for mobility procedures. It is also used to convey uplink and downlink NAS (Non-Access Stratum) messages as a payload, as well as support CM (Connected Mode) idle and CM connected operations such as paging and UE context release. NGAP is transferred over SCTP (Stream Control Transmission Protocol). NGAP protocol provides transport function between UE and AMF by offering NAS signaling transport. The NAS protocol provides mobility management and session management between the UE and the AMF[14].

5G Globally Unique Temporary Identifier (5G-GUTI):
The AMF allocates a 5G Globally Unique Temporary Identifier (5G-GUTI), which provides greater privacy than IMSI because it’s a temporary identity, which the AMF can re-assign at any time [15].

2.3.3 Session Management Function (SMF)

SMF is a fundamental element of the 5G Service-Based Architecture (SBA). The SMF keeps trace of PDU sessions and QoS Flows in the 5GC for UEs and make sure their states and status are in sync between Network Functions in control and user planes. SMF also acts as a Dynamic Host Configuration Protocol (DHCP) server and an IP Address Management (IPAM) system. It also receives PCC (Policy and Charging Control) rules from PCF (Policy Charging Function) and convert PCC rules into SDF Templates, QoS Profiles and QoS rules for UPF, gNB and UE respectively for QoS Flows establishment, modification and release etc.

2.3.4 Network Repository Function (NRF)

NRF works as a centralized repository for all the 5GC network functions and helps to increase the efficiency, scalability, and flexibility of the 5GC. NRF allows other NFs to register and discover each other via a standards-based API. It reduces the requirement for network configuration each time a new network function is added, increasing the resilience of the 5G network. It supports NF management, discovery, and authorization services. Figure 2.7 will give some insight about call flow:

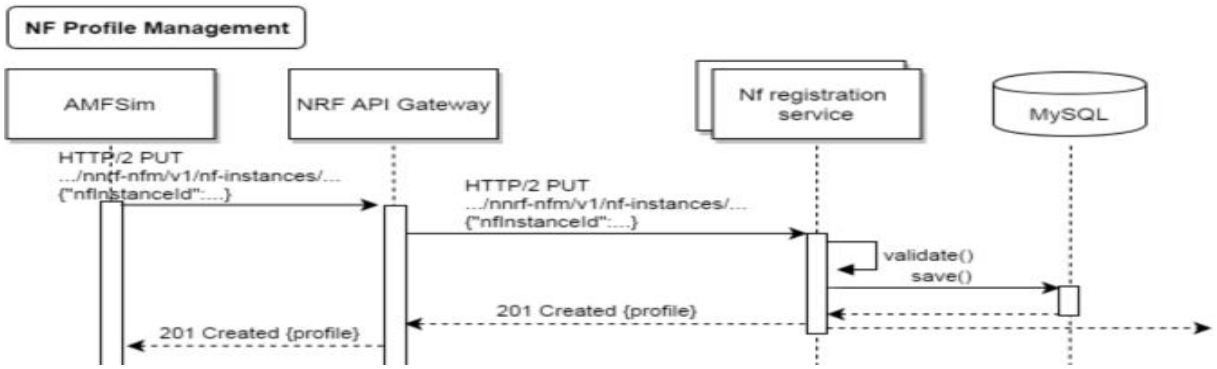


Figure 2.7: Registration of an NF profile in NRF call flow [16]

2.3.5 Service Communication Proxy (SCP)

Since this project involves using open5gs, familiarity with SCP is essential as open5gs uses SCP to connect with other NF via NRF. In the 3GPP Release 15 architecture, each NF, when acting as a service consumer, is responsible for choosing a service producer from among the set of results returned from a query to the Network Repository Function (NRF). 3GPP Release 16 has first introduced SCP which can work together with NRF. SCP acts as a proxy for existing service-based interfaces and does not explicitly expose any service-based interfaces of its own. It enable consumer NFs to outsource responsibilities to a central function that can be more easily configured, monitored, and managed[18]. SCP also helps in providing load balancing and better routing control which bring resiliency to the network. It relieves user NFs from remembering and interpreting complex routing rules associated with next hop selection and at the same time makes re-routing decisions based on load conditions and health status of NF providers within configuration time-period [19].

2.3.6 Policy Control Function (PCF)

Within the 5G networks, the Policy Control Function (PCF) supports a unified policy framework for governing network activity. PCF obtains subscription details from UDM in order to make policy decision and the pass the applicable policy rules to the control plane functions for compliance (3GPP TS 23.501, 2021).

2.3.7 Unified Data Management (UDM)

The UDM manages data for access authorization, user registration, and data network profiles. The technology is similar to the 4G network's home subscriber service (HSS) but is cloud-native and designed for 5G specifically. UDM is a component of the control plane that facilitates communication between the user plane and the control plane using microservices. The storage space of all the UDM's information could reside in a hyperconverged infrastructure (HCI). Virtualization is necessary for 5G, for that HCI and UDM collaborate together. In order for the network to be fully virtualized, HCI applies to the computing, networking, and storage resources.

Stateful and Stateless Versions of UDM:
A stateful UDM keeps data on hand locally, while a stateless version stores data externally in the UDR. A stateful architecture shares data among microservices that orchestrate communication between the network planes. Stateful microservices are preferable for keeping of the record data but the downside is that if a problem arises, then all microservices sharing information must be taken off the network at once [17].

A stateless architecture keeps subscriber data separate from the functions it supports. This way, database access is kept separate from the operation of the network, improving stability and flexibility. The downside is that multiple nodes can't update the same points of information at once, which can cause lag in the network [17].

2.3.8 Unified Data Repository (UDR)

5G UDR stores data grouped into distinct collections of subscription-related information such as subscription data, policy data, structured data for exposure, and application data. Policy data is made available via the N36 interface to the PCF. Application data is placed into the UDR by the external AFs, via the Network Exposure Function (NEF), which can avail different NFs who are authorized to request – subscriber-related information. One intriguing aspect of UDR is the ability of the visiting UDR to locally store the roamers' parameters in roaming scenarios. It functions similarly to the 2G/3G VLR because it can store Policy Data and Data for Exposure of roaming UEs in the UDR in the visited network and make it locally available to the applicable NFs.

According to 3GPP TS 23.501 standard comments: "There can be multiple UDRs deployed in the network, each of which can accommodate different data sets or subsets, (e.g. subscription data, subscription policy data, data for exposure, application data) and/or serve different sets of NFs. Deployments where a UDR serves a single NF and stores its data, and, thus, can be integrated with this NF, can be possible."

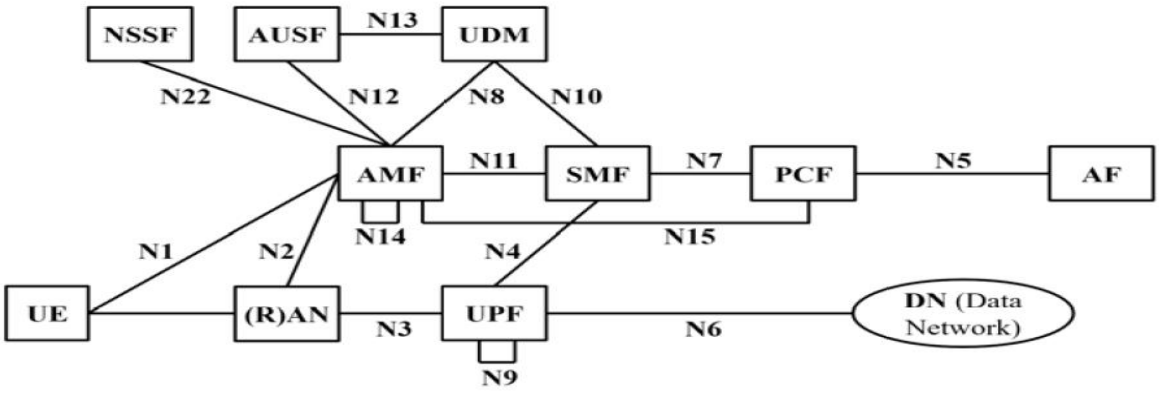


Figure 2.8: Network functions with reference points for a 5G system without roaming [4]

2.4 5G Protocol Stacks and Key Technologies

According to TS 23.501, protocol stack are defined for communication between different network functions. In this section we will discuss about the protocol stacks of user plane(UP) and control plane(CP).

2.4.1 UP and CP Protocol Stack

Figure 2.9 shows the protocol stacks for communication with a UE for both the user and the control plane. In both cases, the lower layers are realized by PDCP(Packet Data Convergence Protocol), RLC (Radio Link Control), and MAC(Medium Access Control). The RLC protocol ensures reliable communication on the air interface in layer 2. The PDCP is primarily responsible for transporting user and signalling data, including encryption and integrity assurance. In the case of the UP protocol stack, the SDAP (Service Data Adaptation Protocol)assigns a corresponding QoS flow and a corresponding Data Radio Bearer (DRB) to the transmitted user data packets. In the CP, the RRC protocol (Radio Resource Control) is used to establish and terminate signalling connections between the gNB and UE, provides Signalling Radio Bearer (SRB), paging, mobility control in the event of handover and QoS management. As shown in Figure 7.15, NAS (Non Access Stratum) messages are exchanged between a UE and the 5GC function AMF (Access and Mobility Management Function) based on an RRC connection to establish and maintain sessions [4].

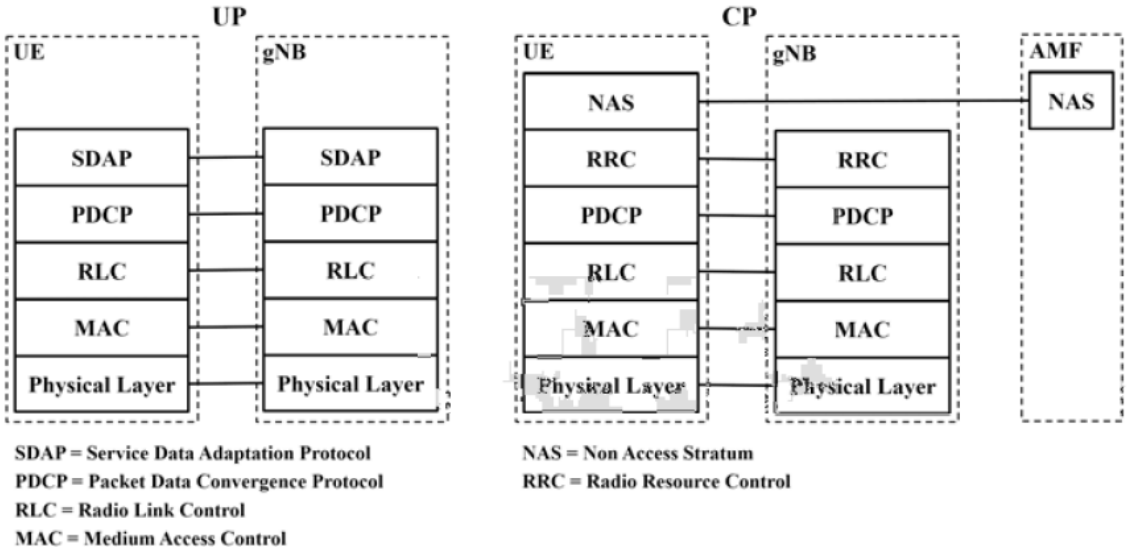


Figure 2.9: UP and CP protocol stack between UE and NG_RAN or 5GC [4]

2.4.2 Software-Defined Networking (SDN)

SDN is considered to be a key technology for the required, flexible handling of data flows in the context of NFV. NFV decouples software and hardware of the network services, SDN separates the control logic with the associated signalling (control plane) from the user data (user plane or data plane) in the network nodes of the IP transport network, i.e., in switches and routers, as shown. This is done by introducing a central SDN controller for the control plane and decentralized, pure SDN switches reduced to data packet forwarding [4].

The monolithic switches and routers are divided by a layer structure into simple SDN switches (in the data plane), which are only responsible for the forwarding of data packets, and SDN controllers (in the control plane), which provide the control logic (separation of control plane and data plane). Concerning flexibility and costs, an SDN controller usually controls a whole range of SDN switches via the Data-Controller Plane Interface (D-CPI), e.g., using the OpenFlow protocol. The protocol processing, i.e., the decision what to do with a flow, a sequence of related data packets, takes place in the SDN controller (central logical control, can still be distributed over several physical or virtual, even redundant network nodes). The rules for forwarding are transmitted to the SDN switches by the SDN controller, e.g via OpenFlow. SDN switches no longer need to be able to understand and evaluate numerous protocols; they must mainly support communication with the SDN controller in addition to packet forwarding. Furthermore, Figure 2.10 shows another advantage of the SDN concept. Via APIs (Application Programming Interface, SDN applications (in the application layer) can program the SDN controller to change its behaviour at runtime and thus implement new network services in the short-term (programmability). The SDN concept enables network administrators to configure and manage the transport network flexibly and dynamically from a central point. This ensures security and optimized use of network resources utilizing self-developed SDN programs. SDN applications can be used for switching, routing, load balancing, QoS, traffic engineering, security, NFV, etc [4].

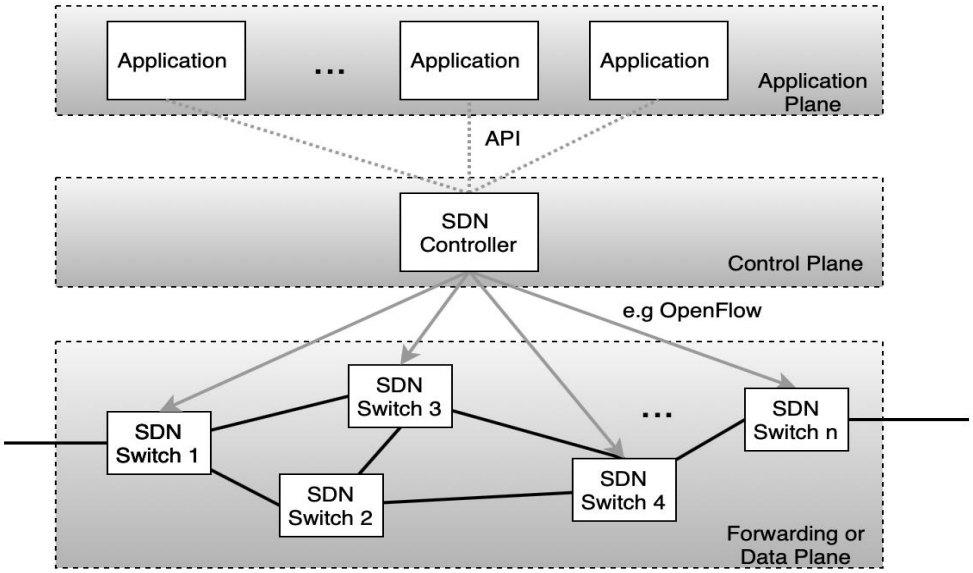


Figure 2.10: SDN Architecture [4]

Software-defined networking is designed to make networks more flexible and agile. SDN was originally defined as an approach to designing, building, and managing networks that separates the network's control and forwarding planes, enabling network control to become directly programmable, and the underlying infrastructure to be abstracted for applications and network services. Primary goal was to move the control plane away from network hardware and enable external control of data through a logical software entity called a controller. The SDN controller, which manages packet-flow control to enable intelligent networking, is situated between network devices and applications. In this architecture, network control becomes programmable. With the controller, administrators will be able to easily manage the 5G network and introduce new services or changes [28].

Key Takeaways, How 5G SDN Will Bolster Networks:

- The virtualization brought by software-defined networking (SDN) to 5G allows administrators to control and change the network remotely.

- 5G can cache content locally, which requires an SDN controller to manage the traffic accessing the cache and orchestrate how the devices are communicating with the network.
- The advantages of the virtualization technology and the next-generation network include automation, the creation of new services that use virtualized resources, lower costs, and higher bandwidth [29].

The ultimate goal of SDN is to create a network that does not need any the design or adjustments of the administrator interference, so, the network can be implemented fully automated administration. The administrators can manage the network through the controller plane more easily with dictating the required policy to the routers and switches, while they have a fully function monitoring over the network [34].

2.4.3 Network Function Virtualization (NFV)

NFV is considered a key technology in the evolution of 5G network. NFV is a concept to decouple network functions from proprietary hardware so they can run as software on standardized hardware and make the network more flexible by minimizing dependence on hardware. It represents a shift in the way telecommunications networks are designed and deployed allowing network operators to manage and expand their network capabilities on demand using virtual, software based applications where physical boxes once stood in the network architecture. This makes it easier to load-balance, scale up and down, and move functions across distributed hardware resources. With continual updates it is possible to keep things running on the latest software without interruption to their customers. Some of the key features for NFV are Network Slicing, Cloud-native NFs, End-to-end Service Management, Edge Computing, RAN Cloudification etc[20]. NFV is standardized by European Telecommunications Standards Institute (ETSI). NFV refers to the substitution of network functions for dedicated devices — such as routers, firewalls, and load balancers with virtualized instances running on commercial off-the-shelf (COTS) hardware. Figure 2.10 shows different layers of NFV architecture.

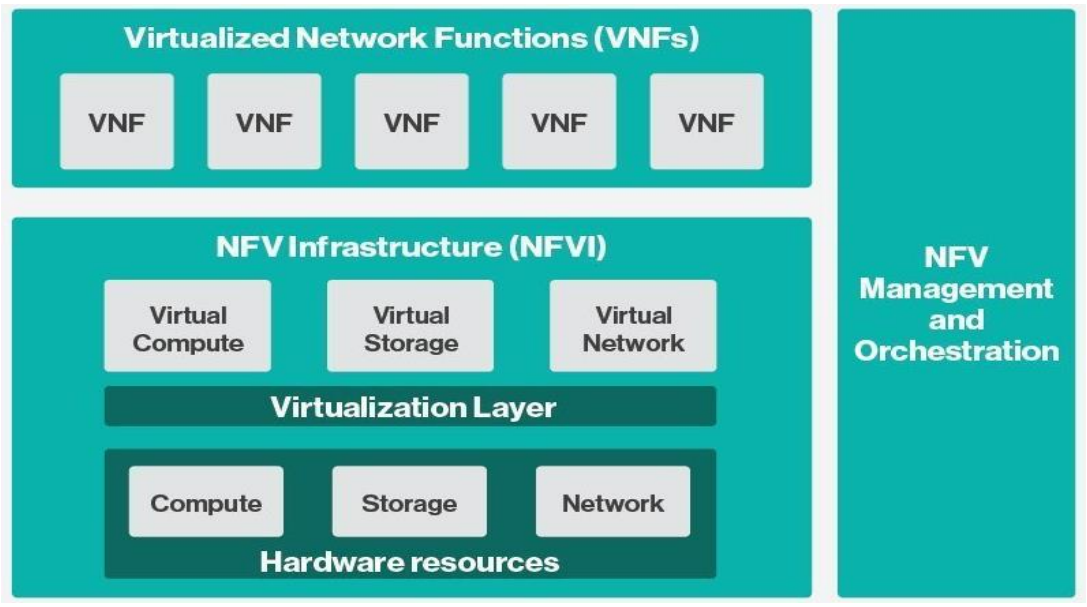


Figure 2.10: Three distinct layers of NFV architecture [21]

2.5 5G open-source platforms and frameworks

There are several open source projects related to 5G networks that are available in today's world and the landscape is evolving every day. These are ongoing initiatives which serve as testbed for 5G networks. Among the well-known initiative are OpenAirInterface [22], open5gs [23], free5gc[24], srsRAN [25]. These are focused on putting 5G standardization into practice. In addition to the projects that are developing the Radio Access Network (RAN) and the core network of 5G, there are additional projects, such as O-RAN [26], Open Network Automation Platform (ONAP) [27], SD-RAN [28], and MOSAIC5G [29], which are assisting the 5G community by enabling additional 5G features. Table 2.5 provides a summary of the ongoing 5G open-source projects.

Project	**Description**
OpenAirInterface	Developing 5G RAN and OAI-5G Core Network (CN) project to provide a 3GPP-Compliant 5G Standalone (SA) CN implementation with a rich feature set. Currently working on 3GPP release 16.
Open5gs	Developed to configure NR/LTE network. Radio access network(gNB/eNB) and USIM are also available. Currently running on 3GPP release 17.
free5gc	Implementation of most service based Virtualized Network Functions (VNF)s with RAN simulators, defined in release 15 and beyond.
srsRAN	Developed for a stable LTE RAN and EPC toward a future 5GC.
O-RAN	Disaggregated RAN with Software Defined (SD) control over Radio Re-source Control (RRC)
MOSAIC5G	It's a platform to develop RAN Intelligent Controller (RIC) and integration of ORAN with OAI as a OAI sub-project.
SD-RAN	Compatible RAN components for O-RAN project
ONAP	Orchestration Platform with build-in management network slicing

The execution of this project mainly uses two famous open source projects, Open5gs and OAI.

2.5.1 Open5gs

Open5gs is an open-source project for 5th generation (5G) mobile core networks. Open5GS contains a series of software components and network functions that implement the 4G/ 5G NSA and 5G SA core functions. It is implemented in C language for 5G core and EPC (Evolved Packet Core). All the Open5GS components have config files, which are presents in github with .yaml extension. Each config file contains the component's IP bind addresses/ local Interface names and the IP addresses/ DNS names of the other components it needs to connect to. This open source project is using SCP that enable indirect communication with other NFs [30].

Package managers can be used to install Open5GS in Debian/Ubuntu and openSUSE environments (for major and minor builds). CentOS, Fedora, and Mac OSX require to build with source code.

2.5.2 OAI

OpenAirInterface (OAI) is a significant open-source project dedicated to providing a flexible and customizable software implementation of the 3rd Generation Partnership Project (3GPP) specifications for mobile communication networks. Designed to support both LTE and 5G technologies, OAI enables researchers, academia, and professionals to experiment with and customize the functionalities of radio access networks (RAN) and core network elements. By adhering to 3GPP standards, OAI ensures compatibility and compliance with industry specifications, fostering a collaborative environment for the exploration and development of cutting-edge wireless communication technologies.

With a strong emphasis on flexibility and experimentation, OAI serves as a platform for hands-on learning and research in telecommunications. The project's open-source community, hosted on GitHub, encourages collaborative development and knowledge sharing among researchers and engineers. OAI also supports the integration of software-defined networking (SDN) principles and network function virtualization (NFV). This integration allows users to create flexible and programmable network configurations, making OAI a valuable resource. As of now, OAI is aligned with 3GPP release 17 specifications and the source code and configuration files of all the NFs are aligned with the same.

2.6 UERANSIM

UERANSIM is an open-source software project developed to simulate the 5G User Equipment (UE) and RAN (gNodeB) in a 5G network. It allows developers, researchers, and telecommunications professionals to emulate the behavior of a 5G UE and RAN in a controlled environment for testing and development purposes. UERANSIM is distributed under GNU General Public License v3.0 which is a free, copyleft license for software and other kinds of works. It can be integrated with any open source 5G core network.

For the purpose of fully functional 5G UE and RAN simulation, UERANSIM offers four executable files. Two executable files (nr-ue and nr-gnb) for running the 5G UE and gNB, can be used respectively. A CLI-based tool (nr-cli) interacts with running 5G UE and gNB. Additionally, a utility called nrbinder is offered to make use of a 5G UE's internet access. It can bind the TUN interface of UE to practically any application (like Firefox, for instance, that runs on the internet) from a 5G UE. For each PDU session, a TUN interface is created.

When a PDU session forms successfully, the UE automatically performs the following actions:

1. The TUN interface is constructed.
2. Configure an IP route, a routing table, and an IP rule.

UERANSIM is written in the C++ programming language and simulates 5G enabled UEs and gNBs. Multiple UEs can be created simultaneously to be connected with the same gNB or multiple different gNBs. It uses the SCTP (Stream Control Transmission Protocol) and GTP (GPRS Tunnelling Protocol) between gNB and Core network components for communication. SCTP protocol with default port number 38412 is designed to use between gNB and AMF (N2 interface), while GTP protocol with default port number 2152 is between gNB and UPF (N3 interface). RLS (Radio Link simulation) protocol designed by UERANSIM itself, is used for communication between UE and gNB.

3 Requirements Analysis

3.1 General Objectives

The main objective of this project is to determine the compatibility of the different network functions in two different open-source projects, Open Air Interface and Open5GS. Both of the projects are 3GPP complaints and provide open-source network implementation for 5G core network. In order to provide the testing environment for 5GC, integration of user equipment and 5G radio access network(5G-RAN) is a necessity. The simulated user equipment (UE) should be able to communicate with 5G core network via 5G radio access network (gNodeB) and utilise the available services.

Below is a general structure of the system that needs to be achieved for this project. The user equipment should be connected to 5G access network, which will be further connected to 5G core network. 5GC would facilitate different network functions for two different projects. Furthermore, from UE/RAN perspective, three main interfaces should be achieved here: control interface (between RAN and AMF), user interface (between RAN and UPF) and radio interface (between UE and RAN). UERANSIM will provide an open-source 5G UE & 5G RAN (gNodeB) implementation. The NFs of Open5Gs should be able to establish connections with the NFs of OAI and vice versa.

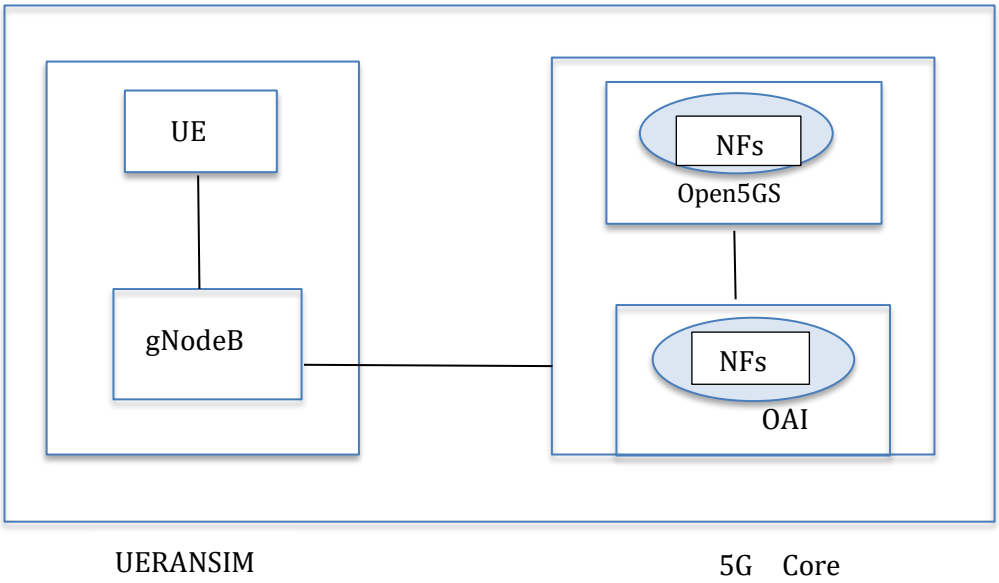


Figure 1.1 General Structure of 5G environment needs to be achieved

3.2 Clarifying the Requirements

Through discussions with supervisors and literature research from books, papers – IEEE (www.ieee.org), ACM (http://dl.acm.org/), access via FRA-UAS Intranet – as well as standardisation organisations – ETSI

(http://www.etsi.org/), ITU-T (http://www.itu.int/en/ITU-T/Pages/default.aspx), 3GPP (http://www.3gpp.org/), IETF (www.ietf.org) – the requirements are to be defined (Trick and Lehmann).

The main functionality required is to produce a standalone 5G environment for core network. The 5G environment should provide the functionalities to try out diverse combination of network functions sourced from different projects. More specifically, first it should check the network functions that are currently being offered by both open-source projects, OAI and Open5GS and then check the interoperability of these NFs.

3.2.1 Network function

In a 5G core network, network functions play a crucial role in enabling the various services and functionalities of the network. Network functions are essentially software components or applications that perform specific tasks within the network architecture. They are designed to handle different aspects of communication, control, and management in order to deliver the desired performance and capabilities of the 5G network.

Open Air Interface: The available network functions in OAI are AMF, SMF, UPF, NRF, UDM, UDR and AUSF. OAI provides Docker Compose files that help orchestrate the different network functions. In the case of the deployment of the OAI components for testing and development purposes, the hosted VM could use docker-compose.

Open5GS: For 5G SA core, the NFs provided by open5GS are AMF, SMF, UPF, NRF, UDR, UDM, PCF, NSSF, SCP and BSF.

To achieve interoperability, the NFs should have a compatible interface to exchange data and signals. For example, to connect the Network Repository Function (NRF) of OpenAirInterface (OAI) to the Access and Mobility Management Function (AMF) and other NFs of Open5GS, it would involve integrating the components from these different projects. It should use compatible communication channels through which the network functions will interact, which could be achieved using REST APIs. For successful communication, both sides of the interaction (e.g., OAI NRF and Open5GS NRF or AMF) must support the same protocols and protocol versions. For communication, it could make use of HTTP/2 and RESTful principles. Data shared between them might be expressed in JSON or XML, and TLS could be used for encryption and authentication. For 3GPP specification, OAI supports 3GPP release 15 and Open5GS supports 3GPP release 16 and 17.

3.3 Time frames

Milestones:

14.08.2023 to 31.08.2023: Literature review and learning the theoretical background

01.09.2023 to 10.09.2023 Analysis of the requirements

11.09.2023 to 15.09.2023: Submission of the requirements analysis and modification based on the feedback

16.09.2023 to 28.09.2023: Elaborating of the concept and planning of the use cases

29.09.2023 to 14.10.2023: Software configuration and PC setup

15.10.2023 :	Implementation and testing of the use case
15.11.2023:	Request for feedback from supervisor
20.01.2024:	Submission of the Project draft to supervisor
23.01.2024 to 08.02.2024: report and documentation	Revise the project based on the proposals from the supervisor,writing
09.02.2024 to 15:02:2024	Final documentation, Submitting the project to the examination office

3.4 Target State

The developed network should consist of one 5G core network from two different projects with available NFs, combined with 5G enabled UEs and 5G RAN. The objective is to develop a matrix of potential NF combinations from Open5G and OAI that can ultimately link with one another despite being located in distinct projects. Information about whether or not one NF from one project can connect to any other/all NFs from another project should be included in the matrix. The compatibility of the NFs will be verified in different ways depending on different circumstances like authentication, registration, mobility management, session management, packet routing & forwarding, traffic steering, and others. By facilitating the experimentation of network function combinations, the 5G environment could be served and the interoperability of these functions could be tested.

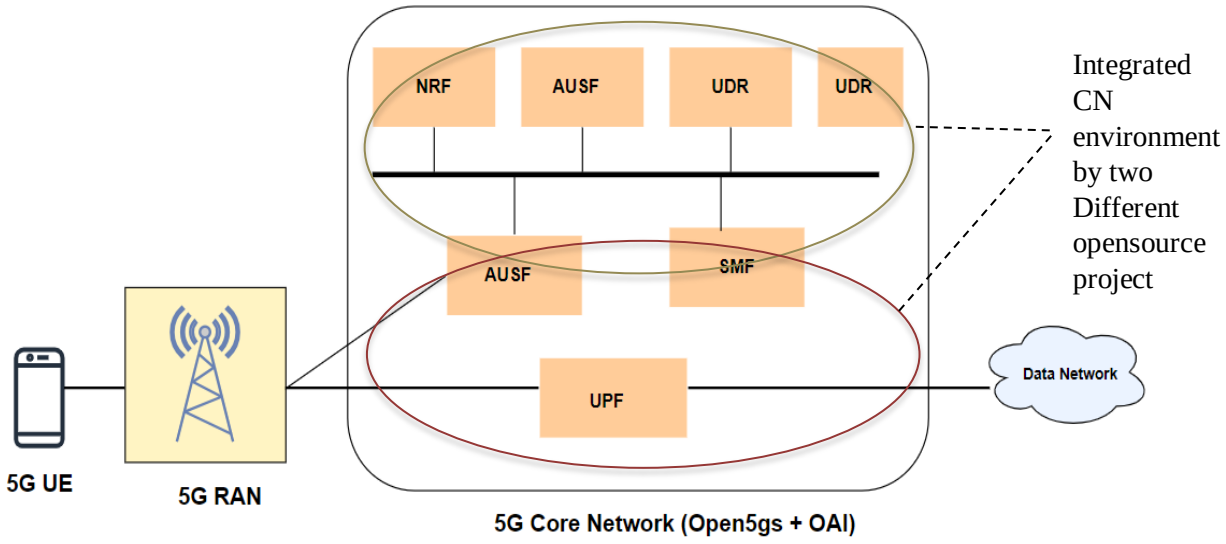


Figure 3.1: Target overview of the system

3.5 Use Cases for the Prototype

The use cases that could be implemented for this project are as follows:-

3.5.1 Integration of OAI NRF with NFs from Open5gs

The developed network should be able to provide functionalities such as “service registration” where other network functions could register with the NRF, attaching their profile and supported services. It is a part of `Nnrf_NFDiscovery` function where NRF as part of its responsibilities, can include service discovery functionalities which involves maintaining a registry or database of available services in the network, including information about which Network Functions (NFs) offer those services.

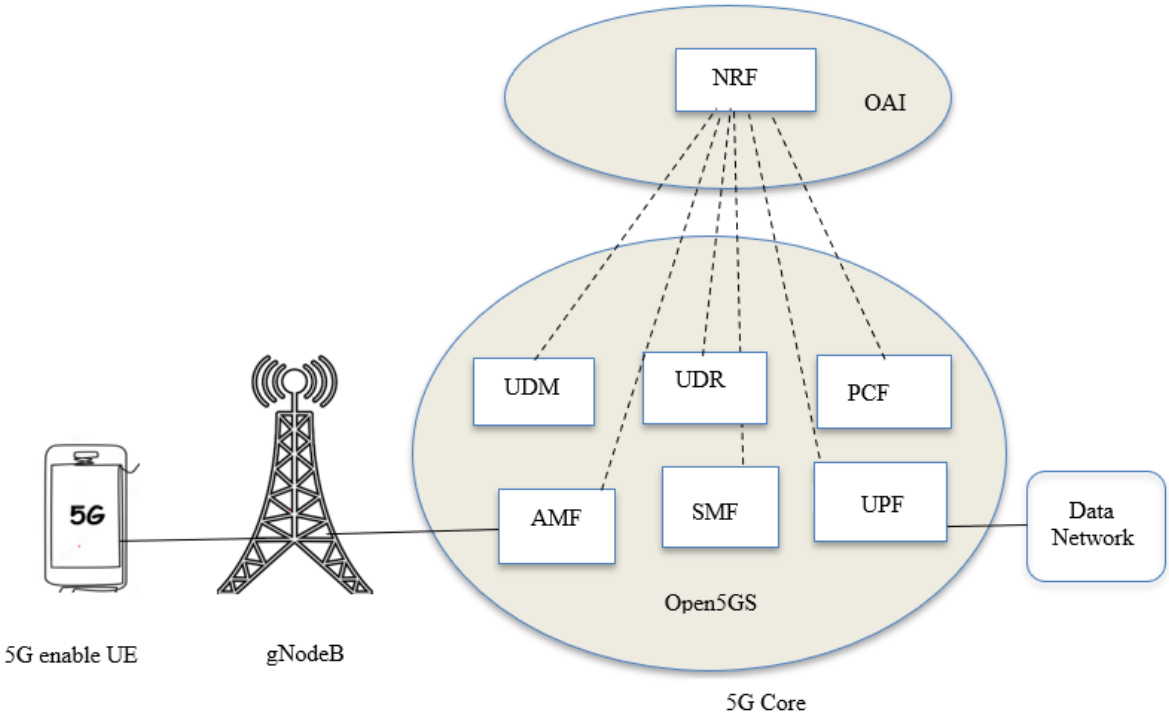


Figure 3.2: Use case 1 architectural overview

It should also provide “NF profile management” using `Nnrf_NFManagement` function provide by NRF. Nnrf_NFManagement provides a list of services in the network such as `NFRegister`, `NFUpdate`, `NFDegister`, `NFStatusSubscribe` and `NFStatusNotify`. These services ensure the capabilities of network so that it can make informed decisions about where to route specific services.

The provided services should be implemented and tested.

3.5.2 Integration of OAI UDR with AMF,SMF,UPF,NRF,UDR from Open5gs

The developed network should be able to provide functionalities for storing and synchronizing user data, subscriber data management and session management etc. According to the specification of "3GPP TS 23.501 V16.0.0 " , OAI UDM is designed to provide functionalities in OpenAirInterface.

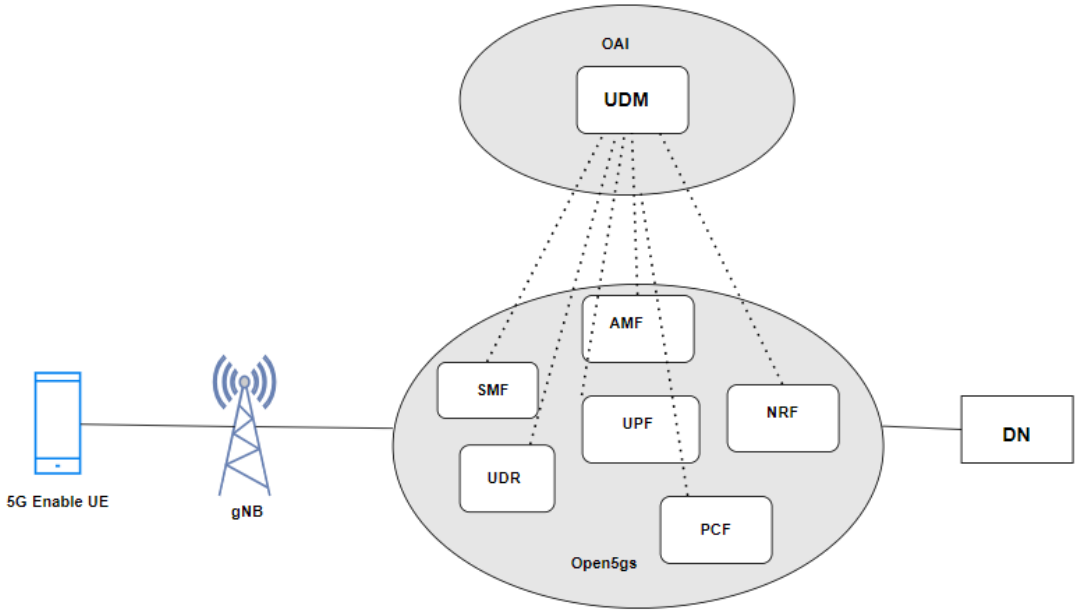


Figure 3.3: Use case 2 architectural overview

The developed network should provide functionalities for generation of 3GPP AKA Authentication Credentials which is a security mechanism used in mobile networks, including 5G. AKA is employed to authenticate users and establish secure communication channels. Authentication credentials, such as keys, are generated during the AKA process to ensure the integrity and confidentiality of communication. It would also provide Identification Handling for users which includes managing and handling user identities within the network. Furthermore, the developed network should support the de-concealment of SUCI. The Subscription Concealed Identifier (SUCI) is used to conceal the Permanent Equipment Identifier (PEI) or International Mobile Subscriber Identity (IMSI), however de-concealment of SUCI is necessary to reveal the actual subscriber identity when required.

3.5.3 Integration of Open5GS NRF with NFs from OAI

The developed network should be able to provide functionalities such as service registration, service discovery, and service routing etc.

In open5gs, Control plane and user plane functions are configured to register with the NRF, and the NRF then helps them discover the other core functions. Here in this case, in the integrated environment NRF would help to

discover other NFs from OAI. The NRF would acts as a central registry or repository for information about all the registered control plane functions within the network.

Each control plane function provides its registration details to the NRF, including information about its identity, supported features, and the types of services it can offer. All the services should be implemented and tested. Several functionalities related this are mentioned in section 3.5.1.

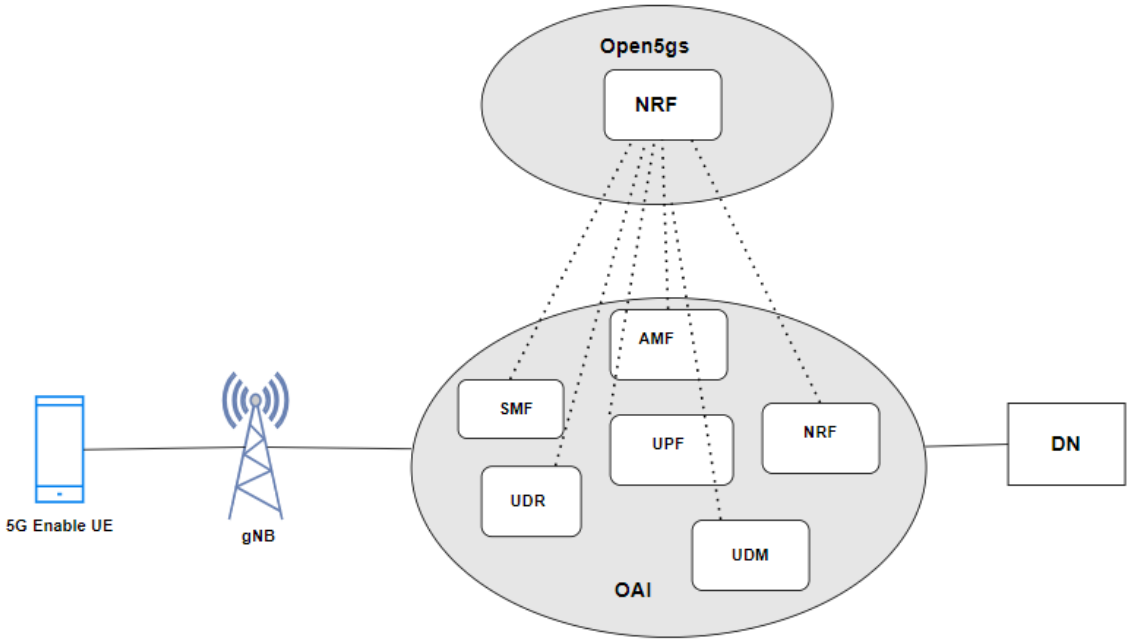


Figure 3.4: Use case 3 architectural overview

3.5.3 Integration of OAI UDR with NRF,AMF,UDM,SMF,UPF from Open5GS

Developed network should store the subscription specific data for users. This User data includes the subscriber's profile, which contains information like the subscriber's identity (SUPI), subscribed services, authentication and authorization data, access and mobility policies, and other related information. Whenever a Network Function (NF) like the AMF or SMF requires user-specific information, it requests the necessary data from the UDM. The UDM retrieves the requested subscription data and provides it to the requesting NF

OAI UDR follows UDR feature list in accordance with the 3GPP version TS 23.501 V16.0.0 §6.2.11. As per the specification mentioned in OAI UDR feature list, UDR can only make communication with UDM using N35 interface. Communication with PCF using N36 interface and also communication with NEF using N37 interface it currently not supported by OAI UDR.

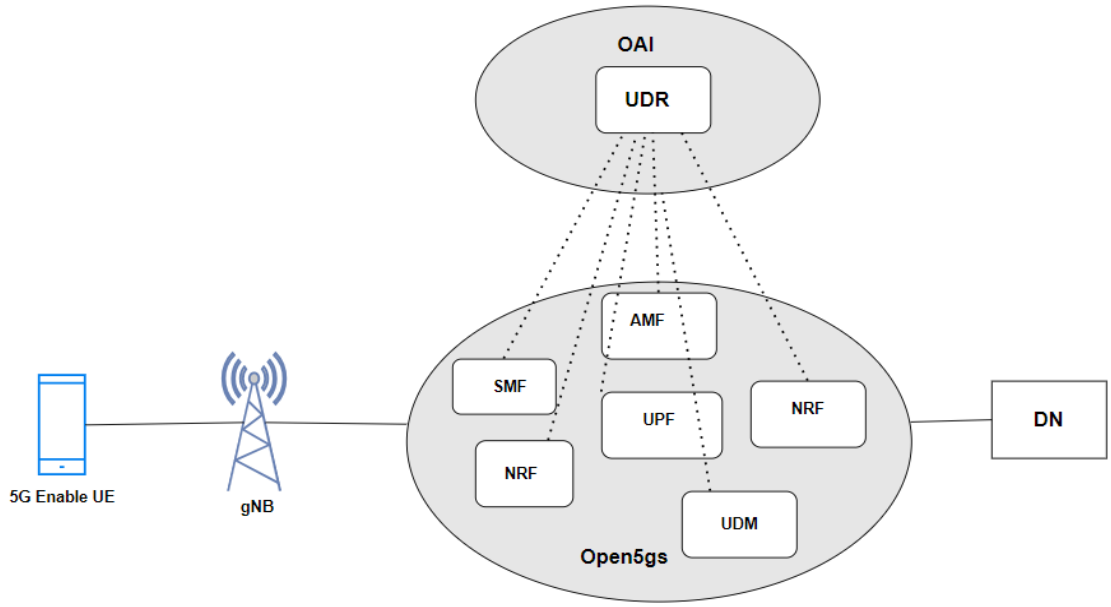


Figure 3.4: Use case 4 architectural overview

4 Realisation

The project's realization is primarily discussed in this chapter. The installation, configuration, and connecting processes for every step of this project are covered in detail. Furthermore, the problems encountered and the actions taken to address them are also covered. The test methodologies used and the observations made are also included in this section.

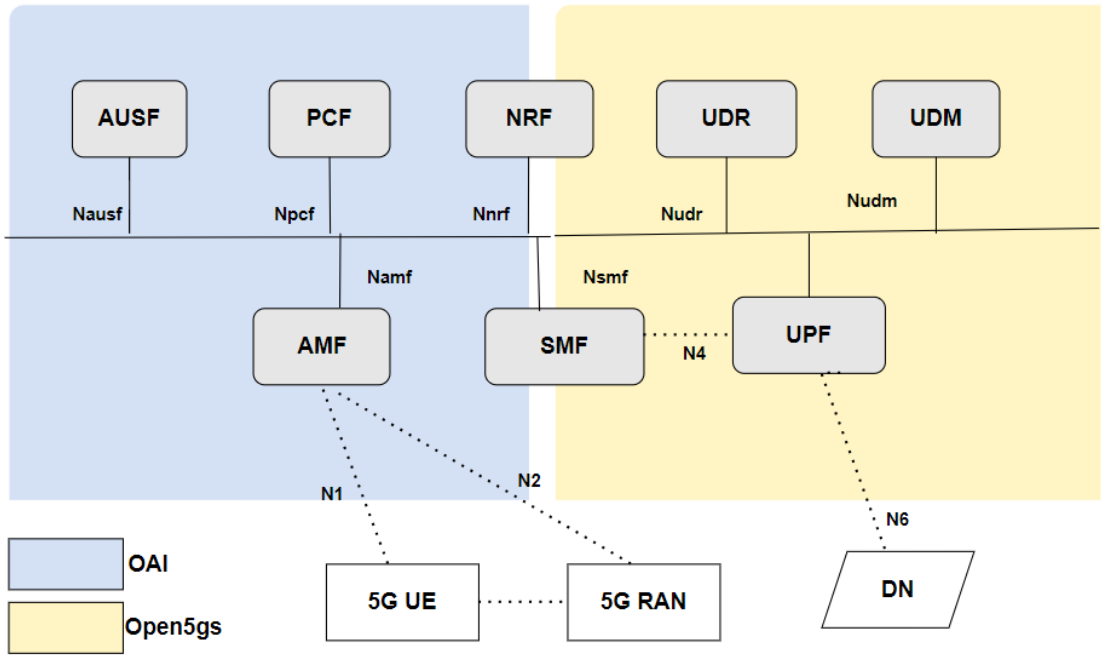


Figure 4.1: Architectural overview of project implementation

Diagram 4.1 illustrates the high level architectural overview of the project implementation that is aimed at establishing in this project, showcasing the connection of different network functions in the core network of a 5G system. The realization focuses on integrating OpenAirInterface (OAI) and Open5GS with various 5G network elements such as Access and Mobility Management Function (AMF), Session Management Function (SMF), and others, to facilitate the interoperability between network functions in different open source projects.

4.1 Prerequisites

Prior to embarking on the actual implementation, certain preliminary steps must be completed to prepare for the initial setup. This part details the procedures for installing VirtualBox and the required virtual machines, along with their respective versions. The subsequent section will elaborate on the installation processes for Open5gs and OAI and their configurations.

4.1.1 Oracle VirtualBox and Ubuntu VM Installation

In this project, virtual machines are used to operate the open-source projects. Consequently, Oracle VirtualBox can be used to install the virtual machines inside it. The project makes use of VirtualBox 7.0, as per the latest upgrade from oracle. Ubuntu Server 20.04 LTS and 18.04 LTS are used for the virtual machines. Since Ubuntu Server 22.04 was not compatible to run along with MongoDB version 6.0, therefore this realisation has been done with Ubuntu focal version and Bionic Beaver versions which are 20.04 and 18.04 respectively . Ubuntu Server is used instead of Ubuntu desktop, as the desktop version needed more hardware space and it could slow down the system and cause performance issues, as a result simple server machine is only used with a limited number of features. A working network interface (ifconfig) and a secure SSH connection is established at the end so that any command can be easily implemented through the host machine's command prompt .

4.2 Setting up 5G Core with OAI NRF and other Open5gs NFs

Figure 3.1 presents an overview of the 5G network architecture, laying out its primary components. Meanwhile Figure 4.2 illustrates the same 5G network architecture with a more specific illustration of components, interfaces, and NFs.

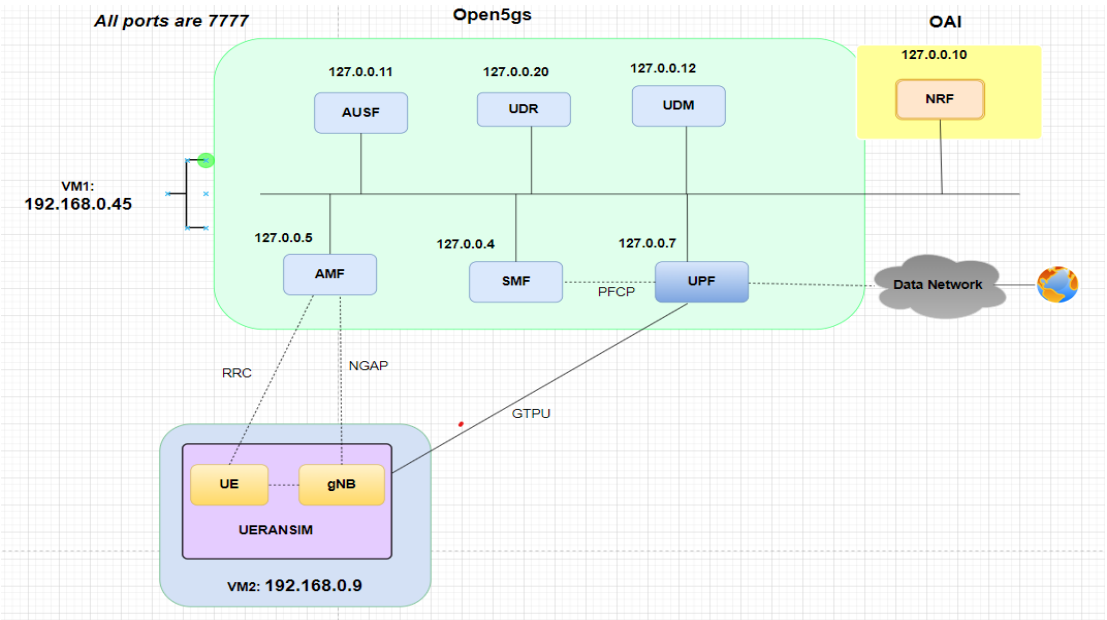


Figure 4.2: Interface and function specific Architecture as per use case 1

From figure 4.2, the simulation of the implemented network can be observed. The core network contains UPF and DN as the UP functions and as CP functions AMF, SMF, UDR, UDM, AUSF and NRF are installed. CP functions are responsible for signalling and UP functions are responsible for data transfer in the 5G network. Here, to check the interoperability between two different projects, NRF is used from OAI project and AMF, SMF, UPF, AUSF, UDR and UDM are used from open5gs project.

4.2.1 Installation

Open5gs Installation:
After successful implementation of Virtual Box and Ubuntu server as prerequisites, the next step is the installation of open5gs. Ubuntu 18.04 LTS version is utilised along with Open5gs version 2.6.4. In open5gs official documentation, there are two types of installation methods mentioned one is “Quickstart” and other one is “Building Open5GS from Sources”. For this project Quickstart process is followed.

Open5gs uses MongoDB as the database for the storage and management of configuration data, subscriber information, and other relevant data within the 5G core network. MongoDB serve as a backend database for Open5GS, providing a reliable and scalable solution for storing the necessary information. Here, MongoDB version 4.4.25 is being used which is perfectly compatible in Ubuntu 20.04. MongoDB is connected to the socket at “127.0.0.1:27017”.

```
1. curl -fsSL https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add –
2. apt-key list
3. echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 multiverse" |
   sudo tee /etc/apt/sources.list.d/mongodb-org-4.4.list
4. sudo apt update
5. sudo apt install mongodb-org
6. sudo systemctl start mongod.service
7. sudo systemctl enable mongod.service
```

Listing 4.1: Commands for installing MongoDB in Ubuntu 20.04

```
1. sudo add-apt-repository ppa:open5gs/latest
2. sudo apt update
3. sudo apt install open5gs
```

Listing 4.2: Commands for installing Open5gs using PPA repository

Open5gs can also be built using Debian and nightly build packages. After successful installation of open5gs next step is to install webconsole for UE registration, which can be done through Nodesource GPG key and also using command line tool. It is significant because WebUI enables us to dynamically modify subscriber data as per the need. When installing WebUI, it is important to take the node.js version into consideration as without proper node.js it cannot function.

```
1. sudo apt install -y ca-certificates curl gnupg
2. sudo mkdir -p /etc/apt/keyrings
3. curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key | sudo gpg --dearmor -o
   /etc/apt/keyrings/nodesource.gpg

4. NODE_MAJOR=20
5. Echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg]
   https://deb.nodesource.com/node_$NODE_MAJOR.x nodistro main" | sudo tee
   /etc/apt/sources.list.d/nodesource.lis
```

```
6. sudo apt update
7. sudo apt install nodejs -y
```

Listing 4.3: Commands for installing WebUI

OAI Installation:

OAI is famous for using docker for installation of different NFs depending on different requirements. But in this realisation, OAI NRF is installed without using docker/docker compose. Every NF from OAI can be installed individually, irrespective of other NFs but it requires huge amount of space to successfully built it. But in this case only NRF was installed so it was manageable for one NF. As stated in the OAI website, this installation process is tested and validated on both Ubuntu 18.04 LTS and Ubuntu 20.04 LTS.

Before installation, OAI NRF source code needs to be downloaded using the following command –

git clone https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-nrf.git

```
1. cd oai-cn5g-nrf/
2. cd ./build/scripts
3. ./build_nrf -I -f
4. ./build_nrf -c -V -b Debug -j
5. sudo nrf -c /usr/local/etc/oai/nrf.conf -o
```

Listing 4.4: Commands for installing OAI NRF

UERANSIM Installation:

There are lots of built-in dependencies shipped with Ubuntu, few built in packages that needs to be installed in order to make UERANSIM work in Ubuntu are mentioned below. The following command must be used to obtain the UERANSIM source code, and the "make" command must be used to build it.

git clone https://github.com/aligungr/UERANSIM

```
1. sudo apt install make
2. sudo apt install gcc
3. sudo apt install g++
4. sudo apt install libsctp-dev lksctp-tools
5. sudo apt install iproute2
6. sudo snap install cmake --classic
```

Listing 4.5: Commands for installing necessary packages

4.2.2 Configuration

This chapter will discuss about the configuration which is used in to conduct this project. Implementation of OAI NFs with Open5gs NRF will be discussed in the section 4.2.2.

Network functions in Open5gs operate differently than network functions in OAI during the initial registration with NRF. In Open5gs, it employs proxy server SCP for the indirection communication instead of the direct communication with NRF. In open5gs, SCP can facilitate service discovery by acting as an intermediary that holds knowledge of network functions and their instances, providing this information to NFs without each one having to query the NRF individually. This way SCP provides an additional layer of resilience in the network.

However, OAI only works with NRF and in the whole architecture there is no use of SCP for indirection communication between NFs. So, in order to enable direct communication with Open5gs and OAI, the first step is to eliminate the need for SCP going forward. As a result, all of the NFs in Open5gs have their configurations changed.

In open5gs, network configuration files are present in the path "etc/open5gs" in ".yaml" format. A few of the configurations of different NFs are mentioned below.

```
amf:
    sbi:
      - addr: 127.0.0.5
        port: 7777
    ngap:
      - addr: 192.168.0.45  #127.0.0.2
    metrics:
      - addr: 127.0.0.5
        port: 9090
    guami:
      - plmn_id:
          mcc: 999
          mnc: 70
        amf_id:
          region: 2
          set: 1
    tai:
      - plmn_id:
          mcc: 999
          mnc: 70
        tac: 1
    plmn_support:
      - plmn_id:
          mcc: 999
          mnc: 70
        s_nssai:
          - sst: 1
    security:
        integrity_order : [ NIA2, NIA1, NIA0 ]
        ciphering_order : [ NEA0, NEA1, NEA2 ]
    network_name:
        full: Open5GS
    amf_name: open5gs-amf0
```

Correct configuration of ngap address is required here, since using this address gNB will communicate with AMF later on. Here Ngap address is set to address of the virtual VM in which UERANSIM gNB is running. For this realization which is 192.168.0.9. It was not needed if there was only one VM for both Open5gs and UERANSIM.

```
#scp:
#    sbi:
#      - addr: 127.0.1.10
#        port: 7777

#
#   <SBI Client>>
#
#   o SBI Client(http://127.0.0.10:7777)
#   sbi:
#     client:
#       no_tls: true
nrf:
    sbi:
      - addr: 127.0.0.10
        port: 7777
```

Fig 4.3: Configuration of AMF of 5G CN of Open5gs

Next for AMF configuration, in order to link the gNB to the AMF of CN, it requires two IP addresses: one for the AMF and another local IP address to which the gNB binds when receiving traffic from the AMF. The values for MNC, MCC, TAC, and SST are set up in accordance with the gNB setup. For gNB and CN, these values have to be the same

Next for SMF configuration, also the NR should be active and SCP should be removed.

```
#scp:
#    sbi:
#      - addr: 127.0.1.10
#        port: 7777

#
#   <SBI Client>>
#
#   o SBI Client(http://127.0.0.10:7777)
#   sbi:
#     client:
#       no_tls: true
nrf:
    sbi:
      - addr: 127.0.0.10
        port: 7777
```

Fig 4.4: Configuration of SMF of 5G CN of Open5gs

Important note to consider here is that work has been done with only IPv4 addresses for NRF for all the cases and disabled IPv6 addresses as OAI is only dealing with IPv4 address.

In the case of UPF, the bind address needs to be specified. The gNB links to the CN via the N2 interface to the AMF and the N3 interface to the UPF. GPRS Tunnelling Protocol for the User Plane (GTP-U) is used for carrying user data between the gNB and the UPF. Here the gtpu IP address is set as per the host machine's IP address where UPF is running.

```
upf:
    pfcp:
      - addr: 127.0.0.7
    gtpu:
      - addr: 192.168.0.45 #127.0.0.2
    subnet:
      - addr: 10.45.0.1/16
      - addr: 2001:db8:cafe::1/48
    metrics:
      - addr: 127.0.0.7
        port: 9090
```

Fig 4.5: Configuration of UPF of 5G CN of Open5gs

In figure 4.6 the OAI NRF configuration is shown. After building NRF as mentioned in listing 4.4, now next we have to run it using the following command. As we are using "-o", so the NRF log will be generated in the console only as the log level is set into "debug" mode.

NRF run command: **`sudo nrf -c /etc/open5gs/nrfOAI.conf -o`**

```
NRF =
{
    # 0 is the default
    INSTANCE      = 1;
    # /var/run is the default
    PID_DIRECTORY = "/var/run";
    LOG_LEVEL = "debug";

    SBI_INTERFACE :
    {
        # NRF binded interface for SBI interface (e.g., communication with other NFs e.g., AMF, SMF, UDM)
        INTERFACE_NAME = "lo";
        IPV4_ADDRESS   = "127.0.0.10/8";
        PORT           = 80;
        HTTP2_PORT     = 7777;
        API_VERSION    = "v1";
    };
};
deblina96@linux:/etc/open5gs$
```

Fig 4.6: Configuration of NRF of 5G CN of OAI

The next stage is to upload UE data to the CN database so that authentication and registration of UE can be done. So, during subscriber creation IMSI, OPC, Subscriber Key need to be added. All these data from UE should be identical with open5gs-ue configuration of UERANSIM.

To register the subscriber information:

Connect to http://192.168.0.45:3000 with "username = admin" and "password= 1423", which is default username and password to login open5gs webui. The password can be changed from account menu as per need.

After, entering the subscriber details using this tool, the subscriber profile will be saved in the UDR MongoDB database backend. Without having to restart any daemons, subscribers will be added using this utility register in the Open5GS UDR right away. If we make changes to a subscriber profile using the WebUI, a restart has to be done in the Open5GS AMF daemon in order for the modifications to take effect.

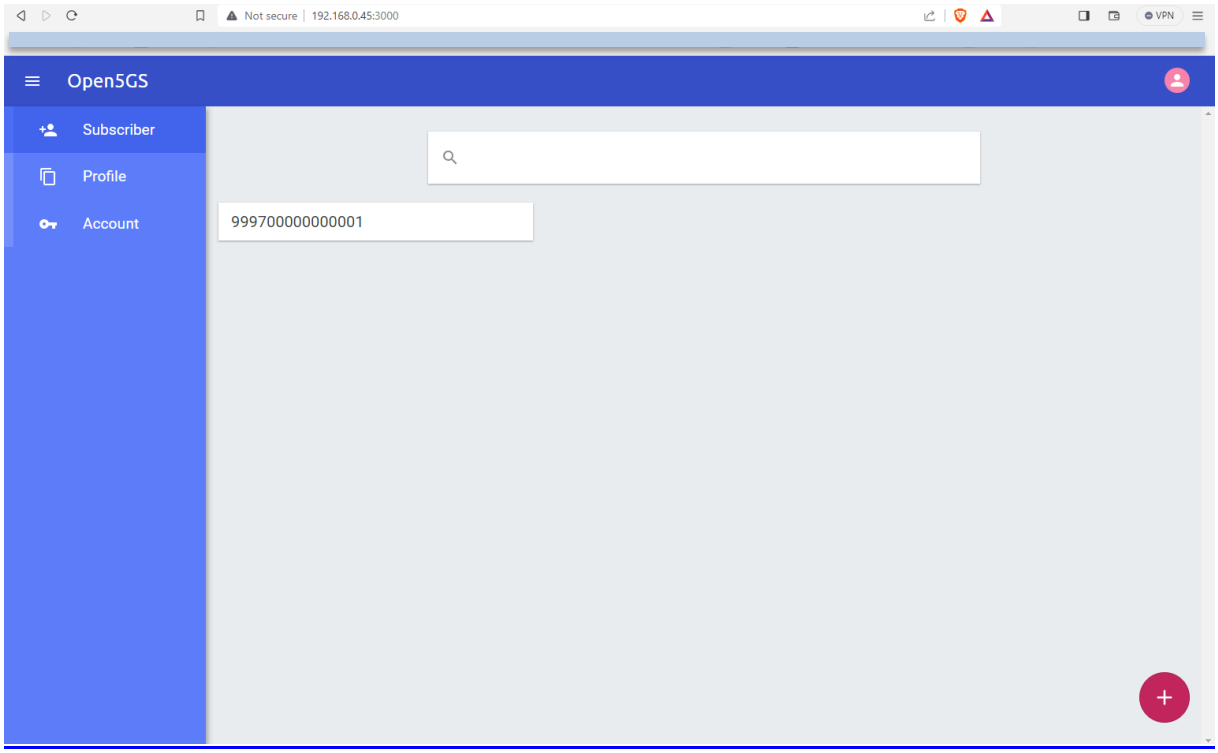


Fig 4.7: Browser snap of Web-UI in Open5gs after subscriber addition

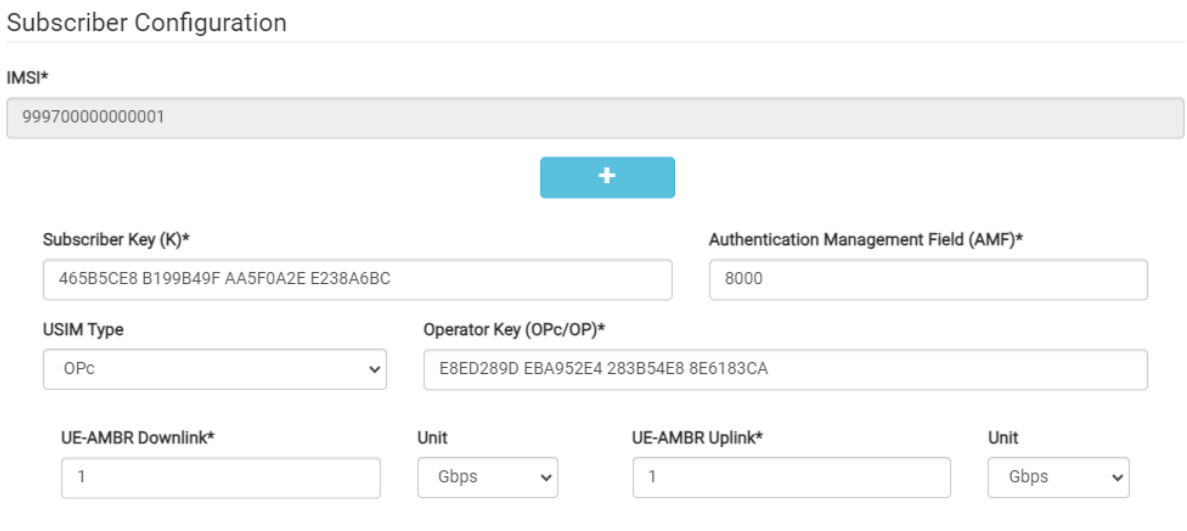


Fig 4.8:Subscriber Configuration in Web-UI of Open5gs

Before running the Open5gs core, a tun interface needs to be created for the CN operation. The name of the interface is ogstun. The interface contains the IP address specified in the UPF configuration.

Create the TUN device with the interface name ogstun.

```
$ sudo ip tuntap add name ogstun mode tun
$ sudo ip addr add 10.45.0.1/16 dev ogstun
$ sudo ip addr add 2001:db8:cafe::1/48 dev ogstun
$ sudo ip link set ogstun up
```

Figure 4.9: Creating ogstun interface for 5GC operation in Open5gs

Next step includes running UERANSIM after building UERANSIM according to the listing 4.5**.** So, these UE and gNB configuration should match to the Webui configuration in order to UE successful initial registration with CN.

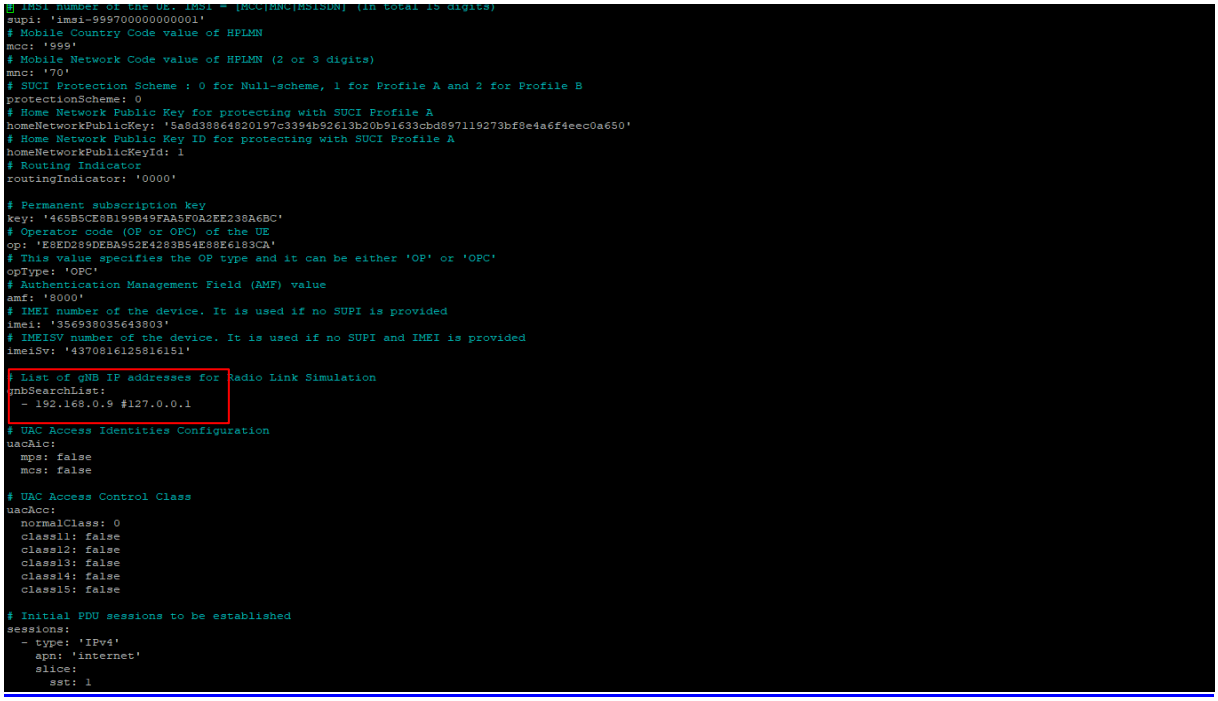


Figure 4.10: Configuration of GNB in UERANSIM

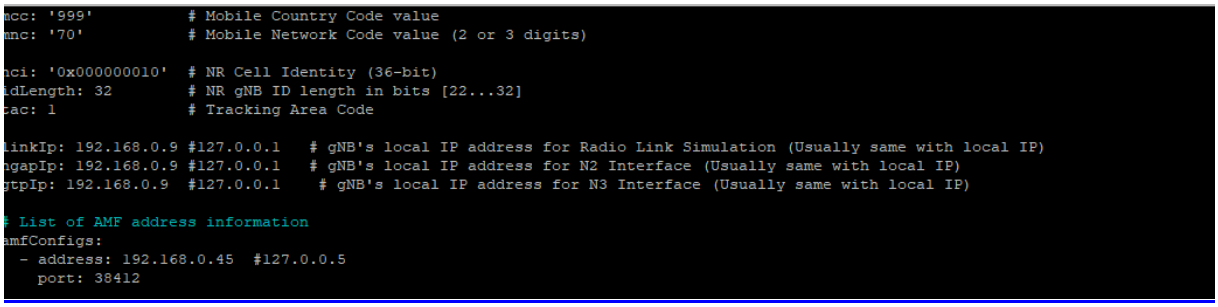


Figure 4.11: Configuration of UE in UERANSIM

5G CN would be launched after the completion of all NFs. The NRF run command is used to initiate the nrfOAI, as shown in figure 4.6. Next, it must remain in operating mode and require the opening of a second terminal in order to bring open5gs on the air. All of these NFs from open5gs—AMF, SMF, UPSF, AUSF, PCF, UDR, and UDM—will be in a running state. As soon as the NFs from open5gs goes into a running state, it can be noticed that the initial registration of different NFs from open5gs is being established with the OAI NRF.

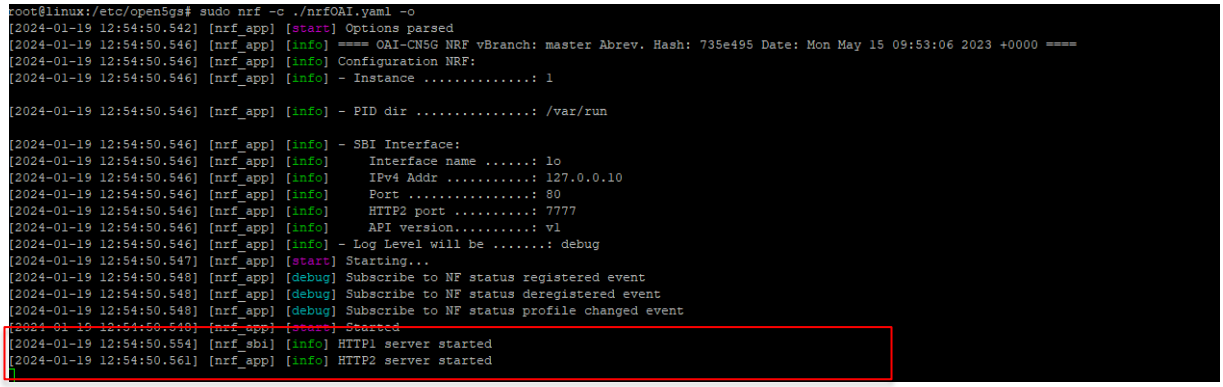


Figure 4.12: OAI NRF started in console

From figure 4.13 to figure 4.18, the registration of NFs to NRF is confirmed by logs captured at NRF node.

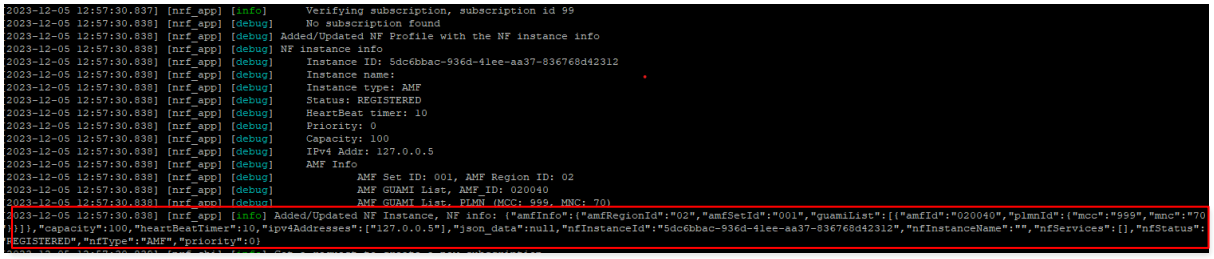


Figure 4.13: Registration of open5gs AMF with OAI NRF

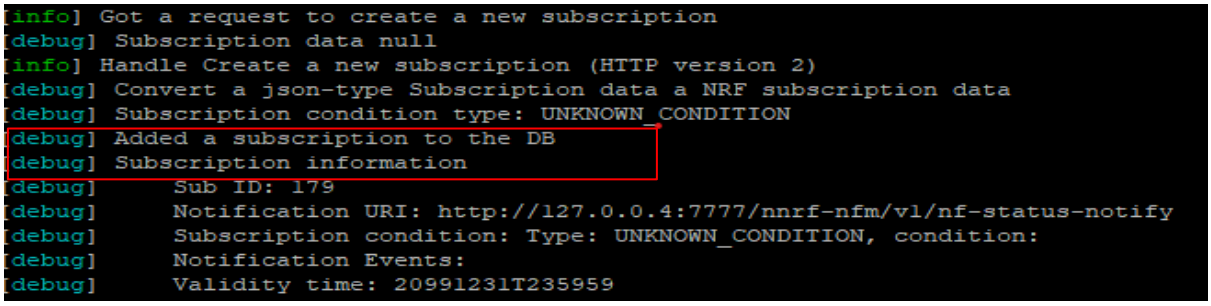


Figure 4.14: Addition of SMF subscription in MongoDB

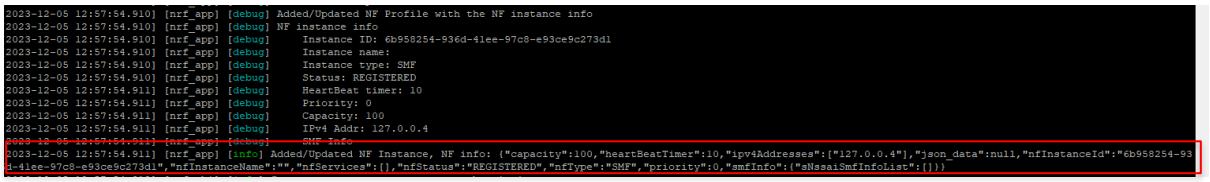


Figure 4.15: Registration of open5gs SMF with OAI NRF

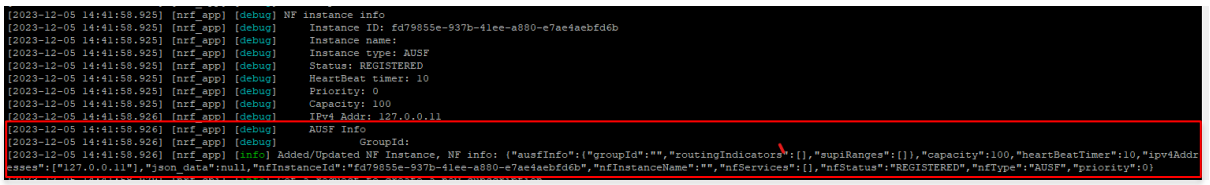


Figure 4.16: Registration of open5gs AUSF with OAI NRF

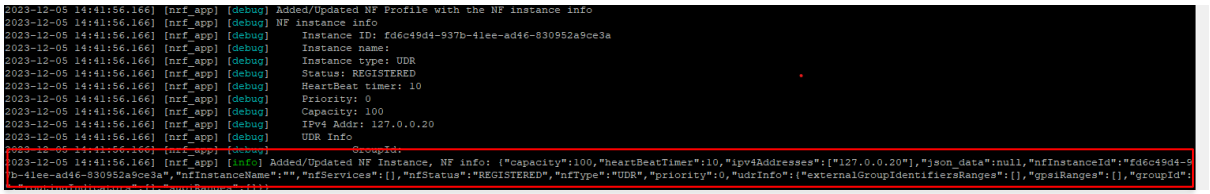


Figure 4.17: Registration of open5gs UDR with OAI NRF

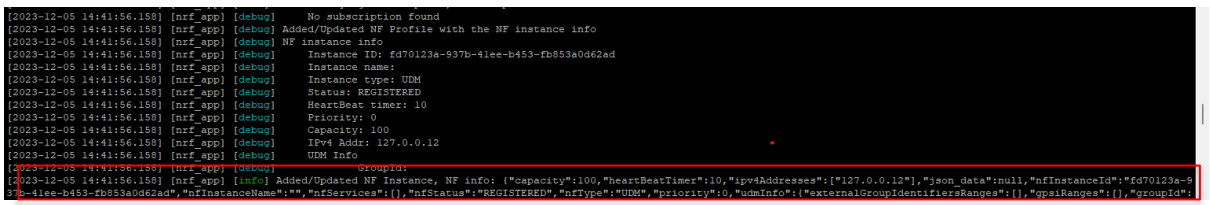


Figure 4.18: Registration of open5gs UDM with OAI NRF

The association between Open5gs NFs with OAI NRF can also be verified if we take a Wireshark capture at the NRF interface. Here, every NF is utilising its loopback address. All NFs instances are binding to the host computers' loopback IP because all of the NFs are installed on the same virtual machine. As a result the host machine's IP address is coming in source in the Wireshark capture at packet number 6466 as shown in figure 4.19. On further analysis of the pcap file, the observation is made between the communication of different NFs from Open5gs with OAI NRF. NRF address is set to 127.0.0.10 as per the figure 4.2. However, the actual loopback address of Open5gs NFs are not directly listed in the pcap file as all NFs are binding to the host machine's loopback IP which is set to 127.0.0.1. But it is possible to check the actual IP if the response messages are expanded to reveal the name of NFs, which is described in figure 4.20.

By capturing the traffic on the `loopback` interface after starting the OAI 5G CN, it is possible to observe the association between various NFs. As soon as the CN is invoked, the loopback interface starts working. To observe the association between the NFs by capturing the traffic on the `loopback` interface, Wireshark (version 4.0.3) is used in this project. The following figure shows the packet exchange between different NFs of the CN.

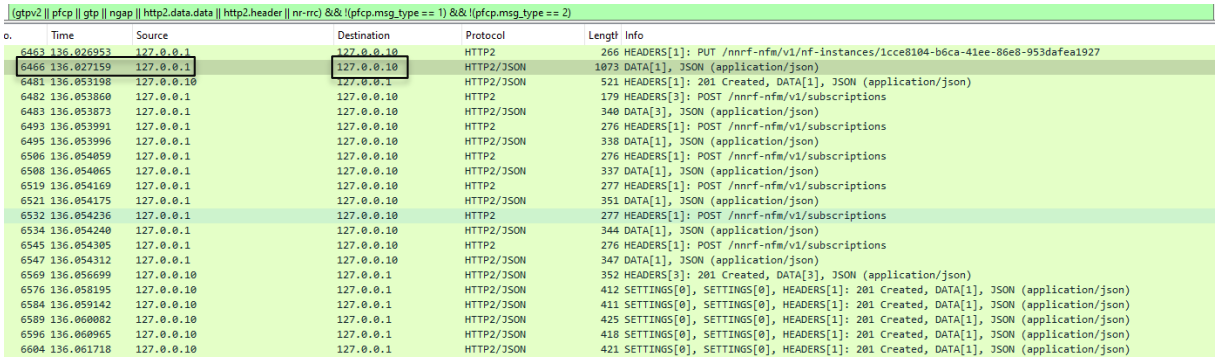


Figure 4.19: Association between different NFs of 5G CN

From figure 4.19, if it the packet number 6466 is expanded, then the source NF is visible with the Ipv4 address, which is AMF in this case. From figure 4.20.1 we can realise the NF name through the member "nfType" which is AMF in this case. Now AMF from Open5gs is sending a registration request to OAI NRF that NRF can set the status of AMF as "REGISTERED" that can be discovered through "nfStatus" member.

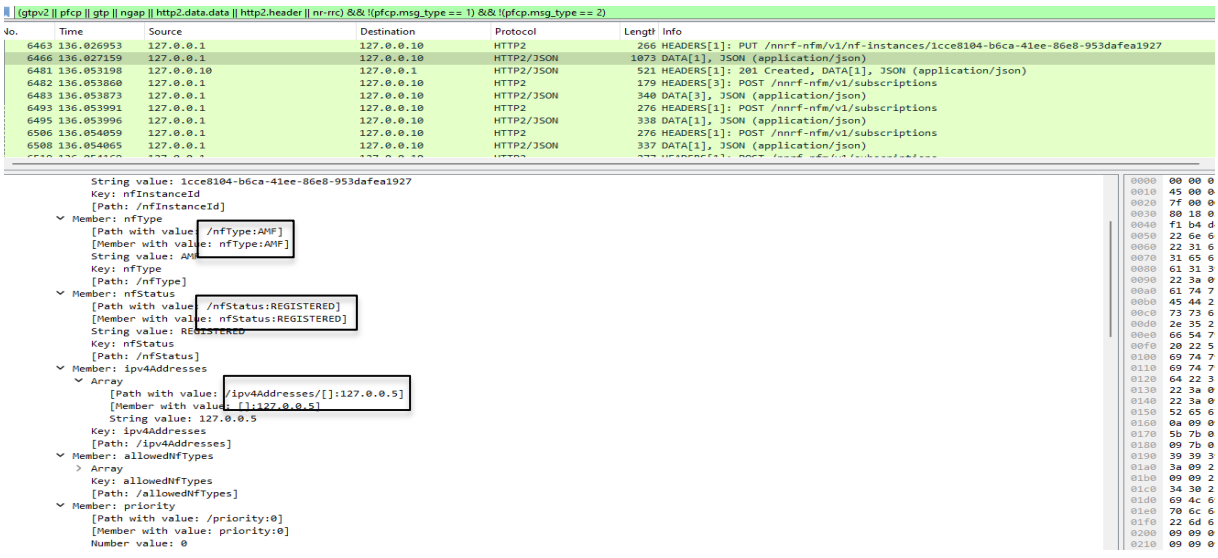


Figure 4.20.1: Extension of packet 6466

Now as a reply to the packet 6466, NRF is registering AMF and sending a response with “201 Created”, which is observed in figure 4.20.2.

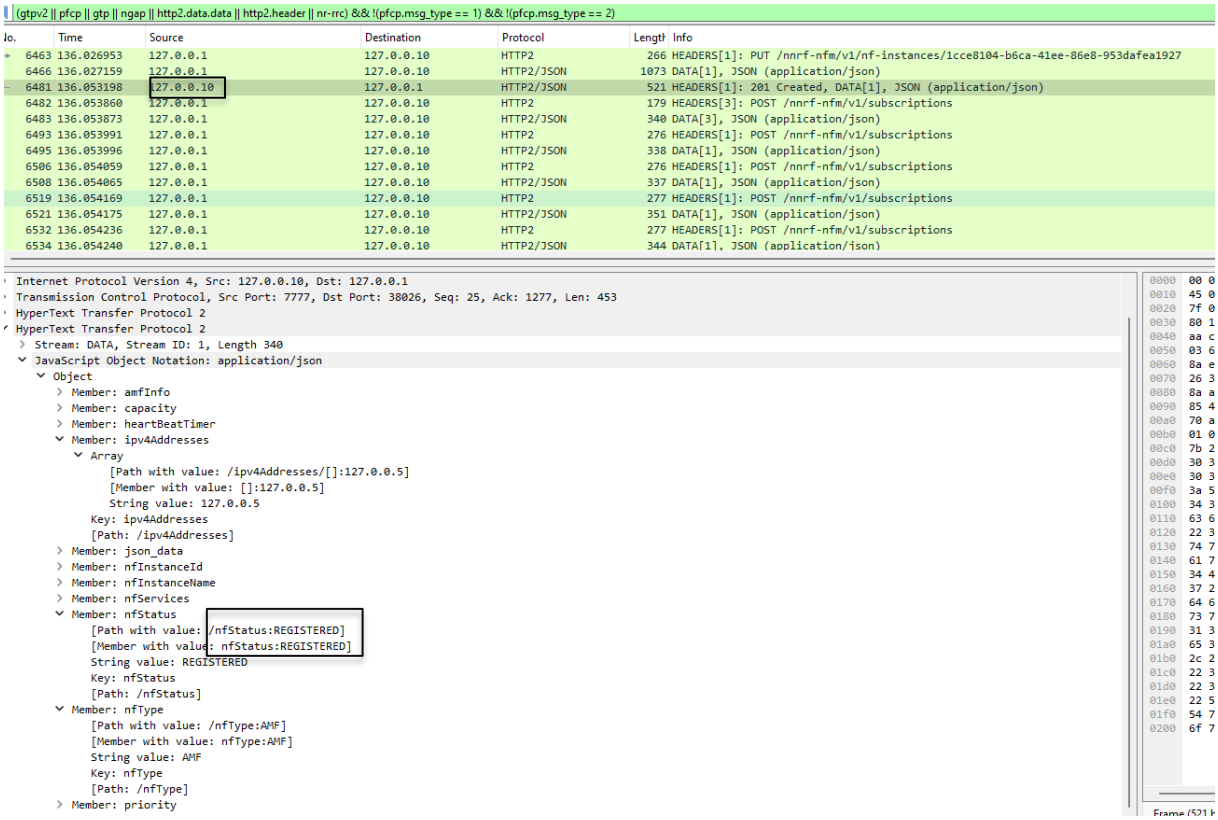


Figure 4.20.2: Response by OAI NRF for packet 6466

This is one example of a registration request, similarly all other NFs also send an HTTP PUT request to the OAI NRF for registration. In the above figure Open5gs AMF is sending the request to NRF with a JSON payload describing the functions and capabilities of AMF, so that other Network Functions can find it.

The **nfServices** key contains an array of services and details of those services, in the below figure the key feature is the serviceName which is “`namf-comm`” which means the `Namf_Communication` Service offered by the AMF.

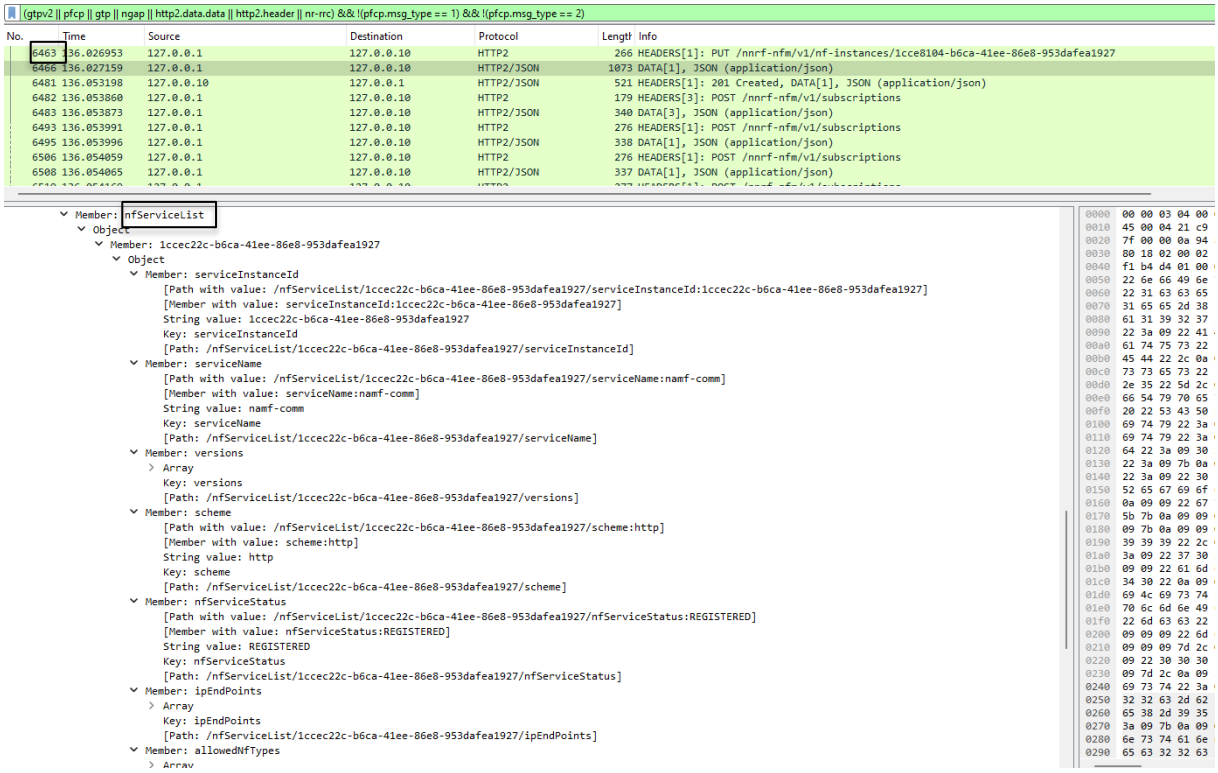


Figure 4.21: Further extension of packet 6466

The next step is add subscription in NRF, this Subscribing is done by sending a HTTP POST with JSON payload indicating which NFs it is interested in. Whenever a Network Function registers on the NRF that related to the type that has been subscribed to, a HTTP POST is sent to each subscriber to let them know.

From figure 4.22 pcap file it is observed that when a AMF registers to the network, SMF gets a notification with information about the AMF that has joined the network. In the below capture “reqNfType” is SMF so SMF is getting the information about AMF through `nf-status-notify` service provided by NRF.

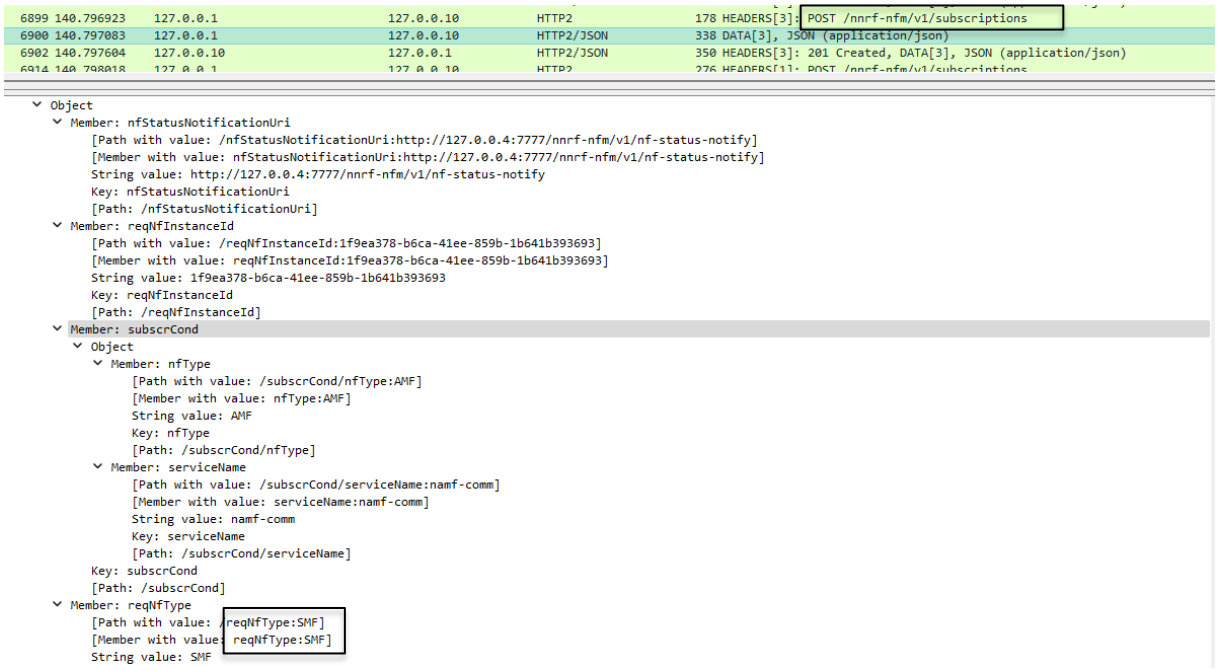


Figure 4.22: Contents of a Subscription message to be notified for SMF in the network

Next, PFCP association setup packets between UPF and SMF can be seen in Wireshark trace capture at UPF's interface shown in figure 4.23.

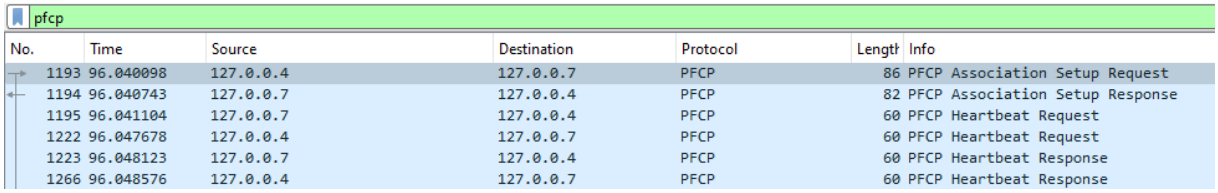


Figure 4.23: Wireshark trace captured at UPF (N4 interface)

4.2.3 Realisation Matrix

This section will be dealing with the realisation matric, which is crucial observation for this realisation. This matrix serves as an essential analytics for assessing the interoperability and feature support across various components of the OpenAirInterface (OAI) with Open5GS systems.

OAI	Open5gs					
NRF Functionalities	**AMF**	**SMF**	**AUSF**	**UDR**	**UDM**	**UPF**

Supports NF registration function	✓	✓	✓	✓	✓	✓
Communication with other NF through Nnrf interface	✓	✓	✓	✓	✓	✓
Maintains the NF profile of available NF instances and their supported services	✗	✗	✗	✗	✗	✗
Support NF service discovery function	✗	✗	✗	✗	✗	✗
NF Subscription	✓	✓	✓	✓	✓	✓

4.2.4 Integration with UE & GNB

Next, a connection to gNB with integrated 5G CN has been realised in accordance with use case 1. The second virtual machine is operated for that. By launching the "nr-gnb" service using the configuration file indicated in figure 4.9 in "config/open5gs-gnb.yaml," it is able to initiate the gNodeB service from the UERANSIM directory. An NG setup between AMF and gNB will be the first step in the 5G RAN and 5GC signalling process. To all available AMFs, a gNB sends an NG setup request with the required data, as illustrated in figure 4.24.

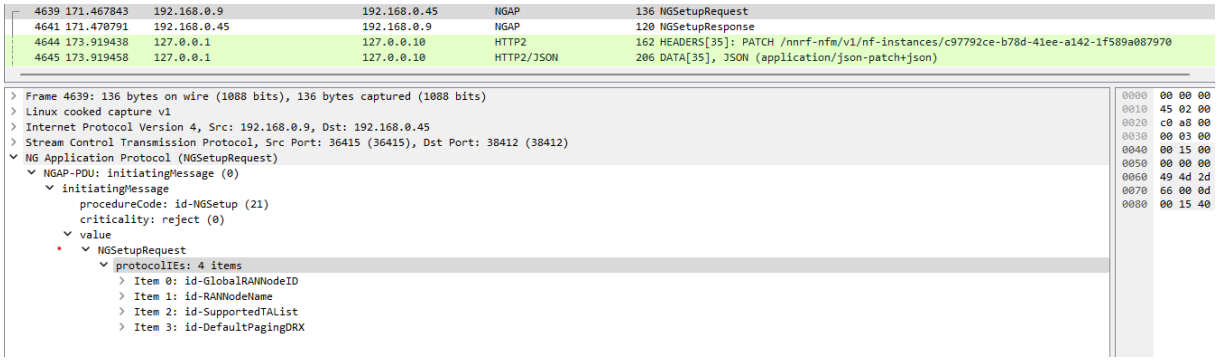


Figure 4.24 NG Setup request by gNB to AMF

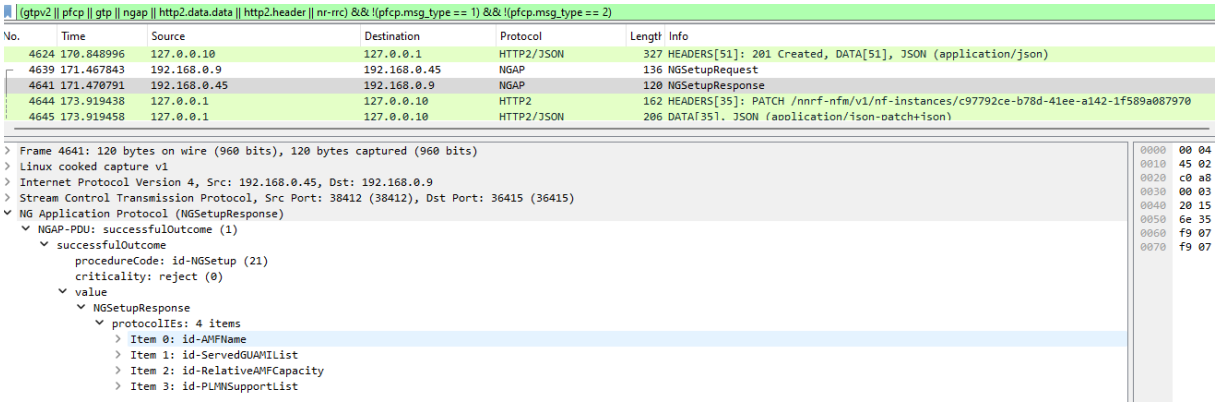


Figure 4.25 NG Setup response by AMF to gNB

Following the successful completion of the NG set up request, the UE will search its search space for accessible gNBs and establish a Radio Resource Control (RRC) connection with gNB. After that, UE sends gNB a registration request, and gNB forwards it to AMF in the form of an initial UE message, as packet number 5563 from figure 4.26 illustrates.

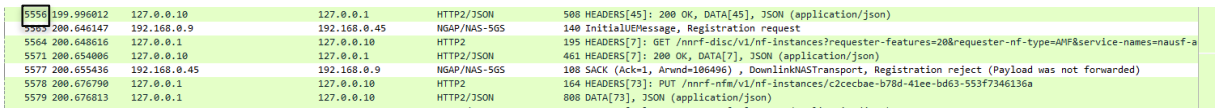


Figure 4.26 NG Setup response by AMF to gNB

But after the registration request goes to AMF, AMF sends an UE authentication request to AUSF to authenticate SUCI or SUPI of UE. But as shown in the packet number 5577, the registration got rejected with the message "`Payload not forwarded`" as AMF couldn't discover AUSF from NRF. In the AMF log it is mentioned that it is unable to find "`Nausf_auth`". Error message is visible through Wireshark capture along with log at AMF node. The specifics of the problem, together with an explanation are mentioned in section 4.2.5.

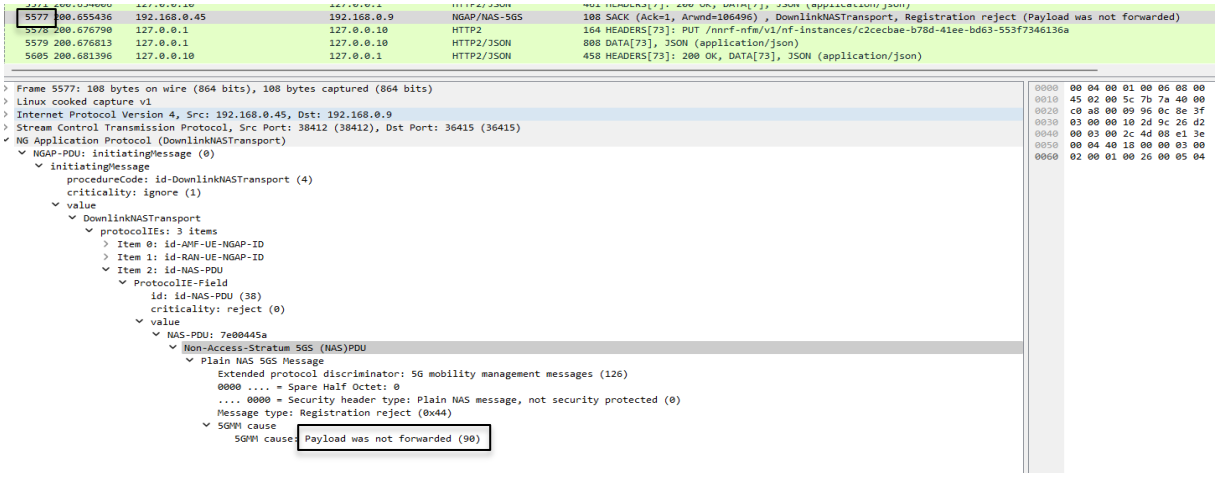


Figure 4.27: Extended packet 5577 with error message from AMF to gNB

LOG Description:

Jan 20 12:19:02 linux open5gs-amfd[2544]: 01/20 12:19:02.490: [sbi] WARNING: Try to discover [nausf-auth] (../lib/sbi/path.c:349)

Jan 20 12:19:02 linux open5gs-amfd[2544]: 01/20 12:19:02.498: [sbi] INFO: [c2cecbae-b78d-41ee-bd63-553f7346136a] (NF-discover) NF registered (../lib/sbi/nnrf-handler.c:829)

Jan 20 12:19:02 linux open5gs-amfd[2544]: 01/20 12:19:02.498: [sbi] INFO: [c2cecbae-b78d-41ee-bd63-553f7346136a] (NF-discover) NF Profile updated (../lib/sbi/nnrf-handler.c:856)

Jan 20 12:19:02 linux open5gs-amfd[2544]: 01/20 12:19:02.498: [amf] ERROR: [suci-0-999-70-0000-0-0-0000000001] (NF discover) No [nausf-auth] (../src/amf/nnrf-handler.c:72)

Jan 20 12:19:02 linux open5gs-amfd[2544]: 01/20 12:19:02.498: [amf] WARNING: [suci-0-999-70-0000-0-0-0000000001] Registration reject [90] (../src/amf/nas-path.c:219)

Figure 4.28: Captured log at AMF after initial registration request

4.2.5 Observations

Observation and Analysis on Use case 1:

Observation 1:

During the installation phase of Open5gs, it is realised that WebUI is not accessible from web browser, because it is unable to identify localhost on Port 3000, which is a default port used by Open5gs. Some additional configuration is made to access WebUI which is listed below.

```
  [1/1]                                /lib/systemd/system/open5gs-webui.service

[Unit]
Description=Open5GS WebUI
Wants=mongodb.service mongod.service

[Service]
Type=simple

WorkingDirectory=/usr/lib/node_modules/open5gs
Environment=NODE_ENV=production
Environment=HOSTNAME=0.0.0.0
Environment=POST=3000
ExecStart=/usr/bin/node server/index.js
Restart=always
RestartSec=2
```

Observation 2:

Once the NRF started registering other NFs, in NRF log few errors can be observed, as the source code will try to validation information of SCP, which is currently disabled.

```
ERROR: yuarel_parse() failed (../lib/sbi/message.c:915)
ERROR: Cannot parse http.location [127.0.0.10/nnrf-nfm/v1/subscriptions] (../lib/sbi/nnrf-handler.c:543)
```

`handle_scp_info()` function which is a part of nnrf-handler.c source code, will try to access network functions instance about SCP, which is disabled in our case.

This is possible to resolve if we disable SCP information from source code completely from every file which is trying to connect and confirm communication through SCP NF.

Observation 3:

During this execution of use case 1, another error message occurred regarding the HTTP2 PATCH request.
A PATCH request is used when an NF needs to make a partial update to its profile, in terms of service base architecture all NFs would ask to keep itself as REGISTERED in the network . In this case as observed in figure 4.29.1,from packet number 1494, "user-agent" is AUSF, meaning AUSF is asking NRF to keep its status as "REGISTERED" which is observed in figure 4.29.2. It uses HEARTBEAT mechanism which is sent through HTTP2 PATCH request.

As a response to that OAI NRF is sent a "400 Bad Request" as shown in figure 4.29.1 in packet number 1498, as it is not supporting the PATCH request sent by different NFs from Open5gs. Each NF that has previously registered in NRF shall contact the NRF periodically (heart-beat), by invoking the NFUpdate service operation, in order to show that the NF is still operative. But the from pcap it is shown that PATCH UPDATE is not successful.

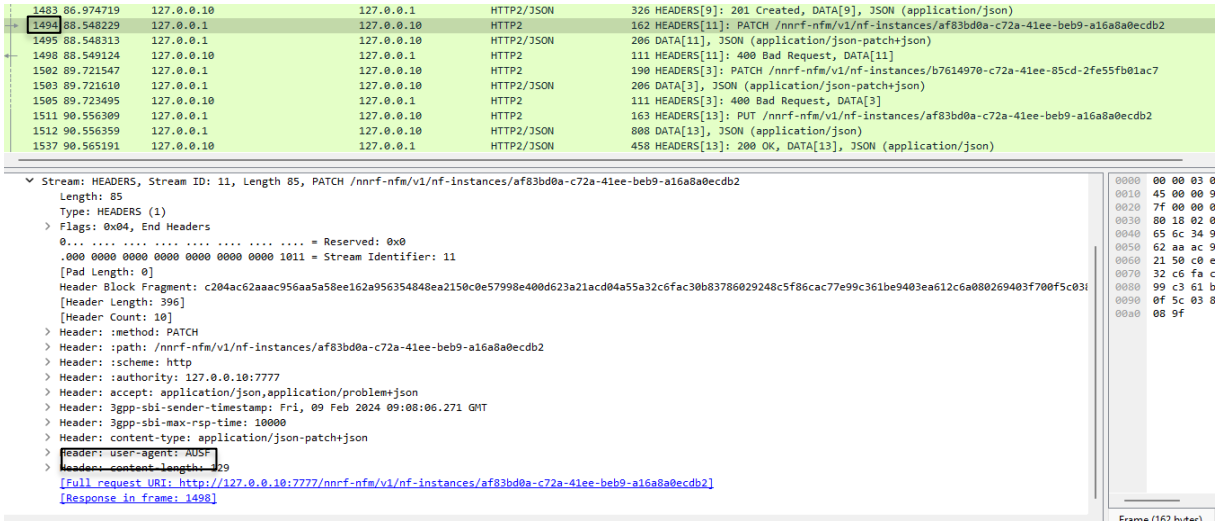


Figure 4.29.1: Wireshark trace of PATCH request from AUSF to NRF

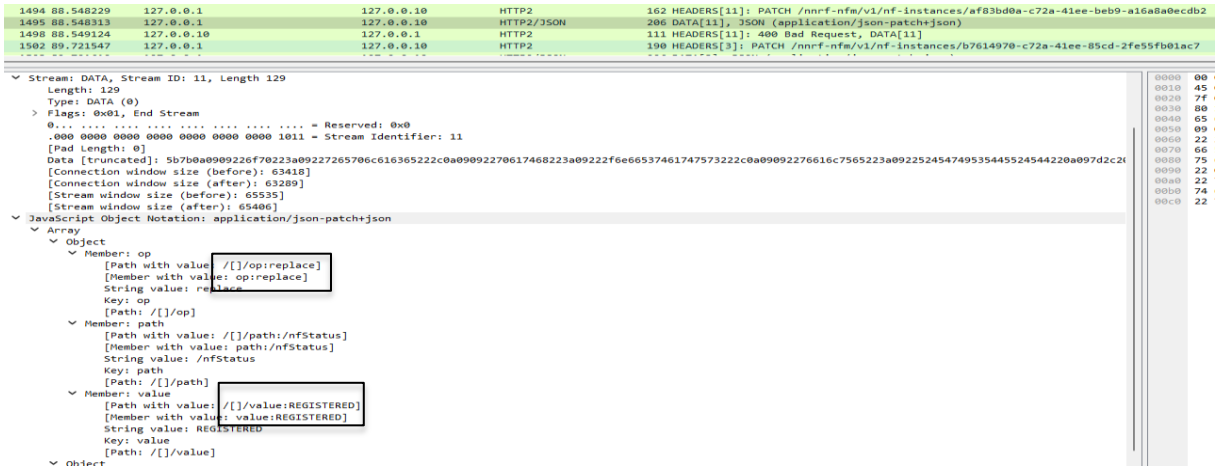


Figure 4.29.2: Wireshark trace of PATCH request JSON data

Observation 4:

The next observation includes the failure of Initial UE registration. UE sends the initial registration request to AMF through gNB. AMF attempts to locate AUSF and UDM services from NRF when UE registration request reaches to it. As seen in figure 4.28, AMF is unable to locate the service off "Nausf_auth" vis NF service discovery function while performing this. The rationale behind this is that NRF is unable to keep the NF profile listed as "REGISTERED" once AUSF completes the initial registration. Thus, AMF cannot locate the services of AUSF to which it has already subscribed.

nnrf-handler.c() function which can be observed from figure 4.28 in the yellow highlighted part, that is a function which handles the registration of a NF profile with the NRF. The code can be found in "Open5gs/lib/sbi/nnrf-handler.c" path which is a part of source code of Open5gs.

This particular function is used to ensure that the necessary objects (nf_instance and NFProfile) and their properties (nf_instance_id, nf_type, and nf_status) are valid and not NULL. It also process information about heartbeat timer. In this case this function is unable to process the information of the JSON payload sent via PATCH request. As a result the error is occurring.

4.3 Setting up 5G Core with OAI UDM and other Open5gs NFs

In this approach, UDM is used from OAI and other NFs are used from open5gs. UDM needed a lot of RAM to install so, it is deployed on a separate virtual machine. In order to separate the CN from the RAN, 5G RAN is also installed in a separate VM. The general architecture of this method is depicted in figure 4.27.

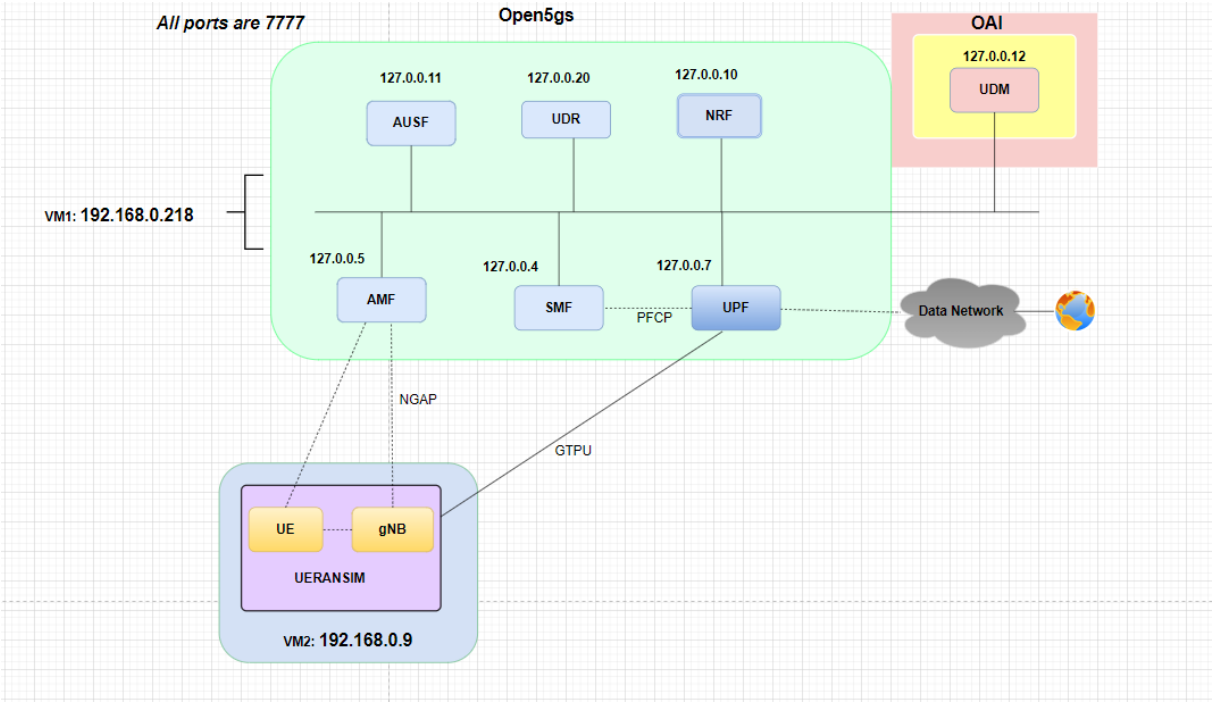


Figure 4.30: : Interface and function specific Architecture as per use case 2

4.3.1 Installation

The installation of open5gs and UERANSIM are already mentioned in the previous section, need to follow the same Listing 4.1 to 4.3 and 4.5.

OAI Installation

First the source code of UDM needs to downloaded from the following path, once downloaded the rest of the installation can be done by following below.

Git clone **https://gitlab.eurecom.fr/oai/oai-cn5g-udm.git -b master**

```
1. cd oai-cn5g-udm/
2. oai-cn5g-udm$ cd ./build/scripts
3. ./build_udm -I -f
4. ./build_udm -c -V -b Debug -j
```

Listing 4.6: Commands for installing UDM from OAI

4.3.2 Configuration

Once the UDM is successfully built then the next step requires configuring the UDM. One important note to remember here, as per the recent update in OAI gitlab repository, once the UDM is built, two different types of configuration files are generated in "oai-cn5g-udm/etc" folder. But "config.yaml" file can be used by all OAI NFs it is not a specific configuration related to UDM. So for this realisation "`udm.conf`" configuration file is used which is only used for UDM configuration. Figure 4.31 illustrates the whole configuration of UDM.

```
## UDM configuration file
UDM =
{
  INSTANCE_ID = 1;
  PID_DIRECTORY = "/var/run";
#  UDM_NAME = "{{ env["UDM_NAME"] }}";
  LOG_LEVEL = "debug";

  INTERFACES:{
    # UDM binded interface for SBI interface (e.g., communication with UDR, AUSF)
    SBI:{
        INTERFACE_NAME = "lo";
        IPV4_ADDRESS   = "127.0.0.12/8";
        # YOUR NETWORK CONFIG HERE (default: 80)
#        PORT           = 80;
        PPID           = 60;
        # YOUR NETWORK CONFIG HERE (default: v1)
        API_VERSION    = "v1";
        # YOUR NETWORK CONFIG HERE (default: 8080)
        HTTP2_PORT     = 7777;
    };
  };

  # SUPPORT FEATURES
  SUPPORT_FEATURES: {
    USE_HTTP2    = "yes";
    REGISTER_NRF = "yes";
  }

  UDR:{
#    {%- if env["UDR_HOSTNAME"] is defined %}
    IPV4_ADDRESS   = "127.0.0.20";
    PORT           = 7777;
    # YOUR API VERSION FOR UDR CONFIG HERE (default: v1)
    API_VERSION    = "v1";
#    HTTP2_PORT     = 7777;
#    FQDN           = "{{ env["UDR_FQDN"] if "UDR_FQDN" in env.keys() else 'localhost' }}"
  };

  NRF :
  {
 #   {%- if env["NRF_HOSTNAME"] is defined %}
    IPV4_ADDRESS   = "127.0.0.10";
    PORT         = 7777;
    # YOUR NRF API VERSION HERE
    API_VERSION  = "v2";
#    HTTP2_PORT   = 7777;
#    FQDN         = "{{ env["NRF_FQDN"] if "NRF_FQDN" in env.keys() else 'localhost' }}";
  };
};
```

Figure 4.31 : Configuration details of UDM in OpenAirInterface

From the above configuration, it is visible that OAI UDM needs the information about NRF where it can register, so NRF details from open5gs configuration are mentioned here. UDR information is also required so the IP address and port from Open5gs UDR configuration is also given here. The source code is written in such a way that it will only allow one port for UDR and NRF configuration. That the reason only one is port is enable in the configuration file of UDM.

Once the configuration is done, the next step is run the udm.conf file. But before that, it should be checked that the UDM is not running from Open5gs as here only one VM is used for core network configuration.

The UDM can be stopped from Open5gs using the below command:

sudo systemctl stop open5gs-udmd

Once the UDM is stopped then NRF,SMF,AUSF,UDR are started from Open5gs using the following command:

sudo systemctl restart open5gs-nrfd

sudo systemctl restart open5gs-amfd

sudo systemctl restart open5gs-smfd

sudo systemctl restart open5gs-ausfd

Now UDM is started from OAI using the below command, as soon it started logs will be generated in the console itself as log level is here set to “debug”.

Sudo udm -c /oai-cn5g-udm/etc/udm.conf -o

```
root@udm:/home/deblina96/oai-cn5g-udm/etc# sudo udm -c ./udm.conf -o
[2024-02-08 15:23:14.914] [udm_server] [start] Options parsed
[2024-02-08 15:23:14.926] [udm_server] [debug] Parsing the configuration file, file type CONF.
[2024-02-08 15:23:14.926] [config] [debug]
Load UDM system configuration file(./udm.conf)
[2024-02-08 15:23:14.931] [config] [info] ==== OAI-CN5G UDM vBranch: master Abrev. Hash: 74224d7 Date: Tue Jan 23 16:34:56 2024 +0100 ====
[2024-02-08 15:23:14.931] [config] [info] ============== UDM =============
[2024-02-08 15:23:14.931] [config] [info] Configuration UDM:
[2024-02-08 15:23:14.932] [config] [info] - Instance.................: 1
[2024-02-08 15:23:14.932] [config] [info] - PID Dir..................: /var/run
[2024-02-08 15:23:14.932] [config] [info] - UDM name.................:
[2024-02-08 15:23:14.932] [config] [info] - SBI:
[2024-02-08 15:23:14.932] [config] [info]     Iface name............: lo
[2024-02-08 15:23:14.932] [config] [info]     IPv4 Addr.............: 127.0.0.12
[2024-02-08 15:23:14.932] [config] [info]     HTTP1 Port ...........: 0
[2024-02-08 15:23:14.932] [config] [info]     HTTP2 Port............: 7777
[2024-02-08 15:23:14.932] [config] [info]     Api Version...........: v1
[2024-02-08 15:23:14.932] [config] [info] - Supported Features:
[2024-02-08 15:23:14.932] [config] [info]     REGISTER NRF ..  ......: Yes
[2024-02-08 15:23:14.932] [config] [info]     Use FQDN .............: No
[2024-02-08 15:23:14.932] [config] [info]     Use HTTP2.............: Yes
[2024-02-08 15:23:14.932] [config] [info] - UDR:
[2024-02-08 15:23:14.932] [config] [info]     IPv4 Addr.............: 127.0.0.20
[2024-02-08 15:23:14.932] [config] [info]     Port..................: 7777
[2024-02-08 15:23:14.932] [config] [info]     API version...........: v1
[2024-02-08 15:23:14.932] [config] [info] - NRF:
[2024-02-08 15:23:14.932] [config] [info]     IPv4 Addr ............: 127.0.0.10
[2024-02-08 15:23:14.932] [config] [info]     Port .................: 7777
[2024-02-08 15:23:14.932] [config] [info]     API version ..........: v1
[2024-02-08 15:23:14.932] [config] [info] - Log Level will be .......: debug
```

Figure 4.32 : Log details of UDM in OAI

After starting UDM from OAI, then UDR and UPF is also started from Open5gs using the following command:

sudo systemctl restart open5gs-udrd

sudo systemctl restart open5gs-upfd

To check all the running instances in Open5gs the below command is used.

ps aux | grep open5gs

Once the core network is active and running, it requires to check UDM’s functionalities. For that the next step is to bring UERANSIM online. gNB’s configuration in UERANSIM needs to be modified in order for gNB to connect to the CN's AMF.

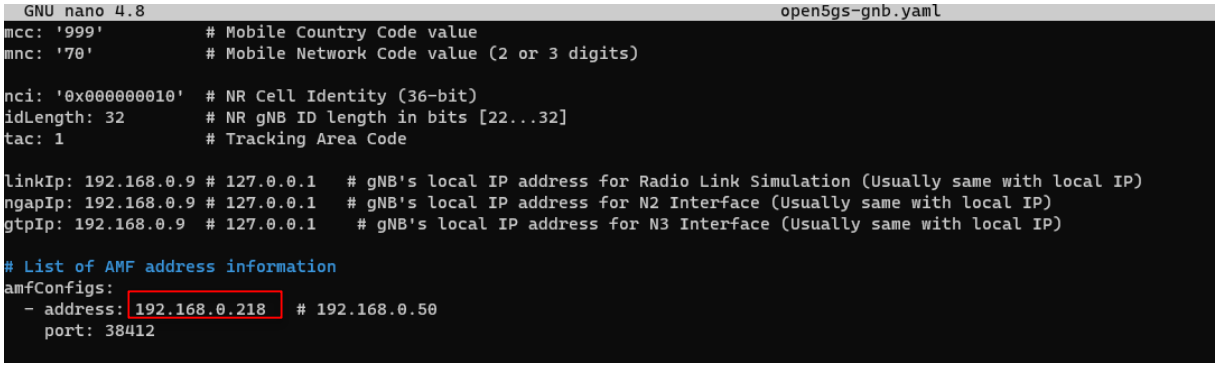


Figure 4.33 : Configuration of gNB

Amfconfigs address should be the IP of the VM in which AMF is running in this case which is 192.168.0.218 as per the Figure 4.33.

Now, once gNB is deployed using the command “build/nr-gnb -c config/open5gs-gnb.yaml” then in UERANSIM terminal we can see connection of gNB with AMF.

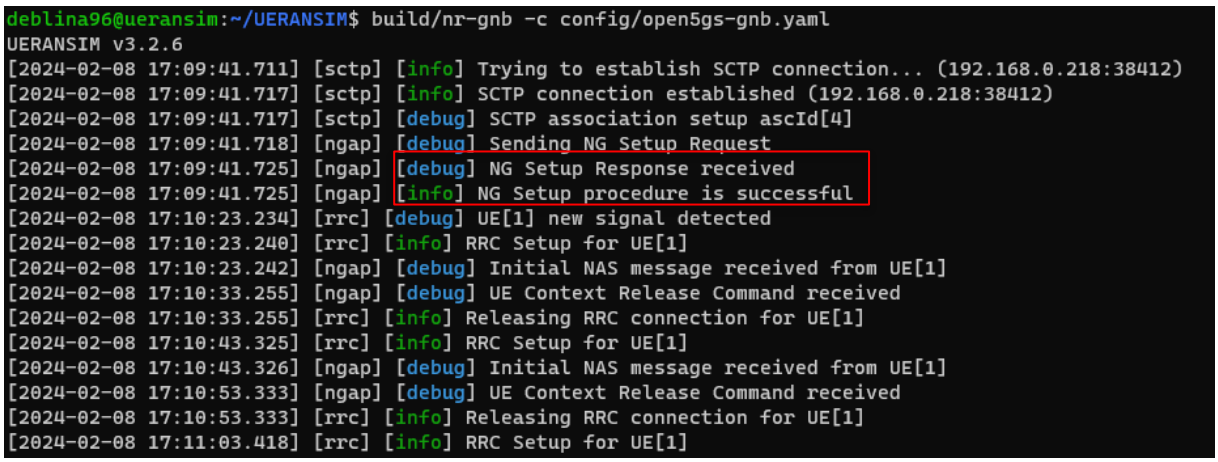


Figure 4.34 : Connection details of gNB with integrated CN

Upon the successful initiation and operation of the gNB, attention is then shifted towards the User Equipment. The primary objective in this phase is to initiate the UE and to closely observe and verify the UDM’s capabilities in handling UE registration and authentication. But from the figure 4.35 it is observed that Initial registration of UE to the 5G CN is failed with the same error which is observed earlier in the case of use case 1.

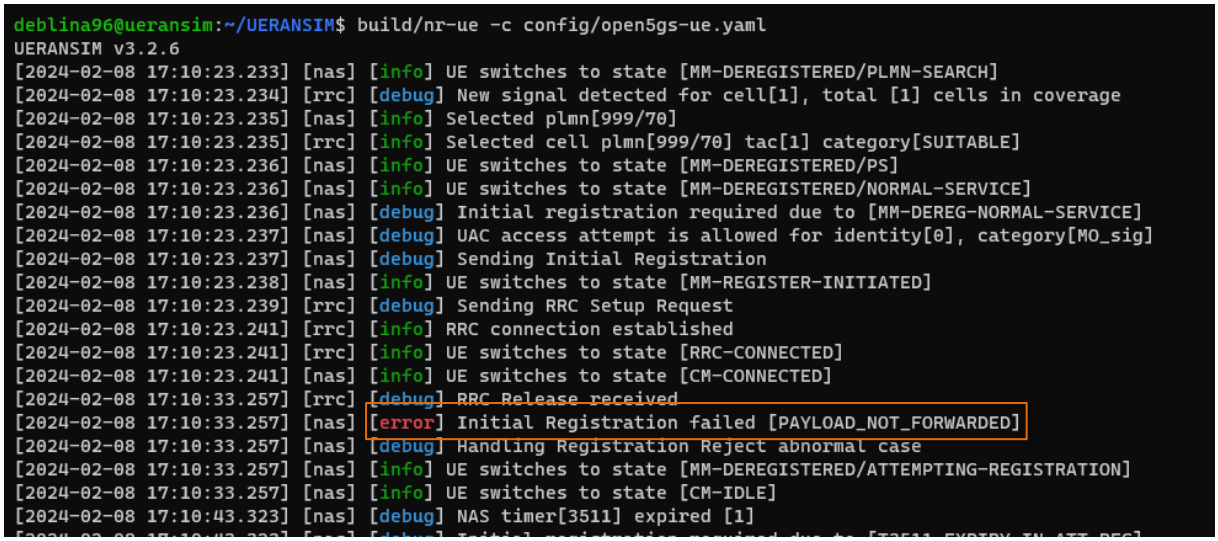


Figure 4.35 : Log details from UERANSIM for UE

4.3.3 Realisation Matrix

Next section represents a matrix comparing the functionalities of the UDM component as implemented in the OpenAirInterface (OAI) against Open5GS within the 5G network infrastructure. The matrix evaluates the presence of certain UDM functionalities and their support across two different platforms:

OAI	Open5gs		
UDM Functionalities	NRF	AUSF	UE
Registration with NRF	✓	NA	NA
Communication with AUSF on UE authentication request	NA	✕	NA
De-concealment of SUCI	NA	✕	✕
Generation of 3GPP AKA Authentication Credentials	NA	NA	✕

4.3.4 Observations

Observation and analysis of Use case 2:

An excellent illustration of how AMF, AUSF, and UDM are used for initial UE registration may be found in Figure 4.36. However, in this instance, it is evident from UE log analysis that UE registration is unsuccessful with the error message "payload not forwarded," which is caused by the absence of the highlighted portion shown in the picture.

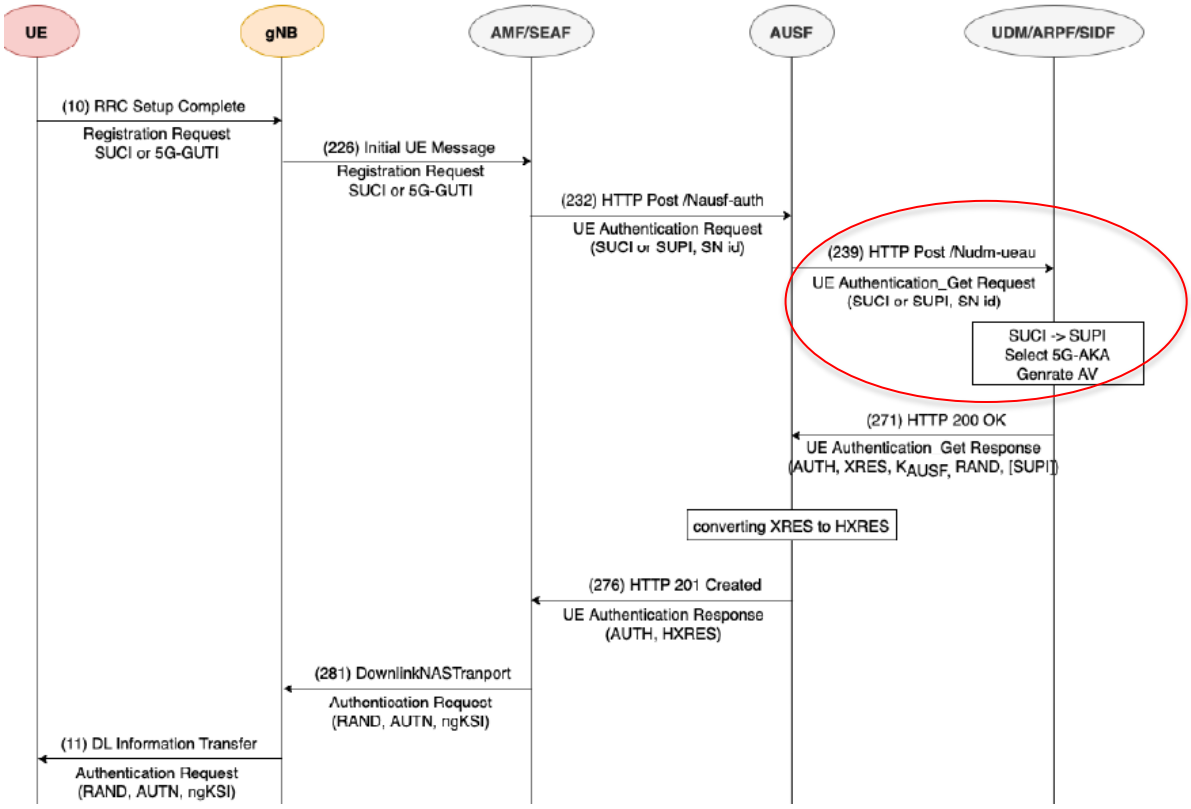
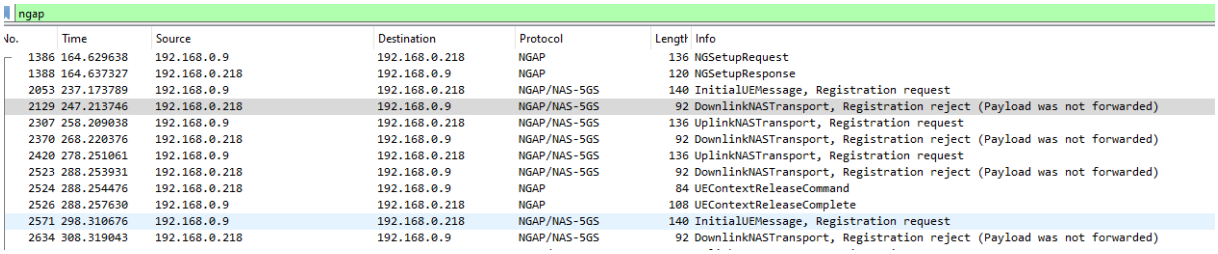


Figure 4.36 : Flow graph for authentication process of UE

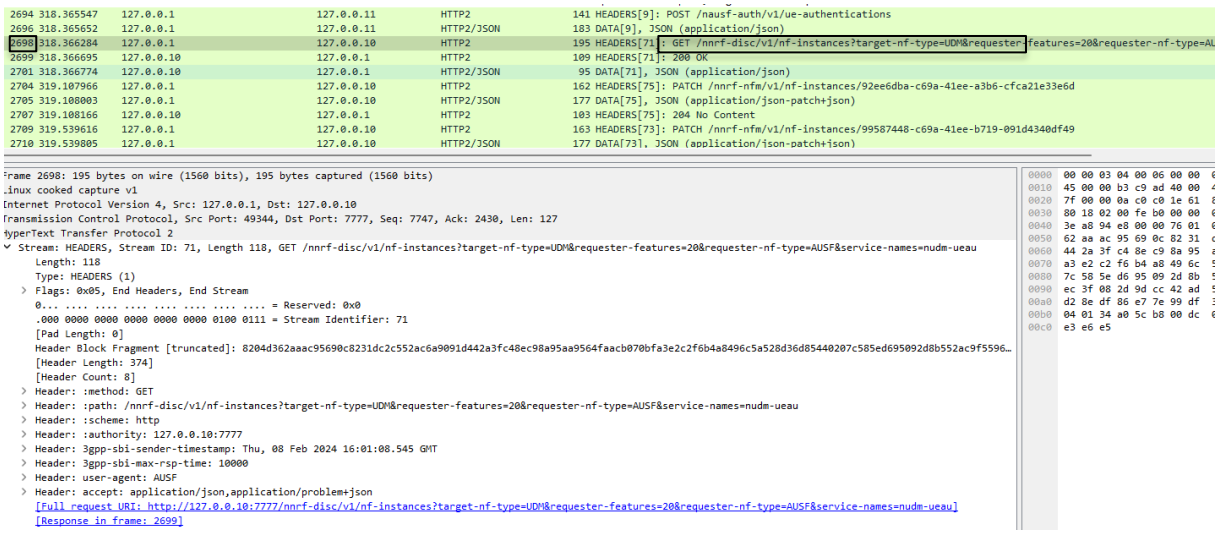
There is a problem with the service discovery procedure, based on the observations made through Wireshark captures, notably in figures 4.37 and 4.38. Specifically, the AUSF is unable to locate the “nudm-ueau:AUSF” service using the NF-discover function. Figure 4.38's packet number 2698 highlights the GET request that indicates an effort to locate this service. Because of this service discovery failure, the AUSF is unable to authenticate the User Equipment which is a necessary step before allowing access to the network. This means that the UE's authentication process is unable to be finished, which stops the network access procedure from moving forward.



ngap

No.	Time	Source	Destination	Protocol	Length	Info
1386	164.629638	192.168.0.9	192.168.0.218	NGAP	136	NGSetupRequest
1388	164.637327	192.168.0.218	192.168.0.9	NGAP	120	NGSetupResponse
2053	237.173789	192.168.0.9	192.168.0.218	NGAP/NAS-5GS	140	InitialUEMessage, Registration request
2129	247.213746	192.168.0.218	192.168.0.9	NGAP/NAS-5GS	92	DownlinkNASTransport, Registration reject (Payload was not forwarded)
2307	258.209038	192.168.0.9	192.168.0.218	NGAP/NAS-5GS	136	UplinkNASTransport, Registration request
2370	268.220376	192.168.0.218	192.168.0.9	NGAP/NAS-5GS	92	DownlinkNASTransport, Registration reject (Payload was not forwarded)
2420	278.251061	192.168.0.9	192.168.0.218	NGAP/NAS-5GS	136	UplinkNASTransport, Registration request
2523	288.253931	192.168.0.218	192.168.0.9	NGAP/NAS-5GS	92	DownlinkNASTransport, Registration reject (Payload was not forwarded)
2524	288.254476	192.168.0.218	192.168.0.9	NGAP	84	UEContextReleaseCommand
2526	288.257630	192.168.0.9	192.168.0.218	NGAP	108	UEContextReleaseComplete
2571	298.310676	192.168.0.9	192.168.0.218	NGAP/NAS-5GS	140	InitialUEMessage, Registration request
2634	308.319043	192.168.0.218	192.168.0.9	NGAP/NAS-5GS	92	DownlinkNASTransport, Registration reject (Payload was not forwarded)

Figure 4.37 : Wireshark trace of UE, gNB with 5G CN



2694	318.365547	127.0.0.1	127.0.0.11	HTTP2	141	HEADERS[9]: POST /nausf-auth/v1/ue-authentications
2696	318.365652	127.0.0.1	127.0.0.11	HTTP2/JSON	183	DATA[9], JSON (application/json)
2698	318.366284	127.0.0.1	127.0.0.10	HTTP2	195	HEADERS[71]: GET /nnrf-disc/v1/nf-instances?target-nf-type=UDM&requester-features=20&requester-nf-type=AU
2699	318.366695	127.0.0.10	127.0.0.1	HTTP2	109	HEADERS[71]: 200 OK
2701	318.366774	127.0.0.10	127.0.0.1	HTTP2/JSON	95	DATA[71], JSON (application/json)
2704	319.107966	127.0.0.1	127.0.0.10	HTTP2	162	HEADERS[75]: PATCH /nnrf-nfm/v1/nf-instances/92ee6dba-c69a-41ee-a3b6-cfca21e33e6d
2705	319.108003	127.0.0.1	127.0.0.10	HTTP2/JSON	177	DATA[75], JSON (application/json-patch+json)
2707	319.108166	127.0.0.10	127.0.0.1	HTTP2	103	HEADERS[75]: 204 No Content
2709	319.539616	127.0.0.1	127.0.0.10	HTTP2	163	HEADERS[73]: PATCH /nnrf-nfm/v1/nf-instances/99587448-c69a-41ee-b719-091d4340df49
2710	319.539805	127.0.0.1	127.0.0.10	HTTP2/JSON	177	DATA[73], JSON (application/json-patch+json)

```
Frame 2698: 195 bytes on wire (1560 bits), 195 bytes captured (1560 bits)
Linux cooked capture v1
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.10
Transmission Control Protocol, Src Port: 49344, Dst Port: 7777, Seq: 7747, Ack: 2430, Len: 127
HyperText Transfer Protocol 2
▾ Stream: HEADERS, Stream ID: 71, Length 118, GET /nnrf-disc/v1/nf-instances?target-nf-type=UDM&requester-features=20&requester-nf-type=AUSF&service-names=nudm-ueau
    Length: 118
    Type: HEADERS (1)
  › Flags: 0x05, End Headers, End Stream
    0... .... .... .... .... .... .... .... = Reserved: 0x0
    .000 0000 0000 0000 0000 0000 0100 0111 = Stream Identifier: 71
    [Pad Length: 0]
    Header Block Fragment [truncated]: 8204d362aaac95690c8231dc2c552ac6a9091d442a3fc48ec98a95aa9564faacb070bfa3e2c2f6b4a8496c5a528d36d85440207c585ed695092d8b552ac9f5596…
    [Header Length: 374]
    [Header Count: 8]
  › Header: :method: GET
  › Header: :path: /nnrf-disc/v1/nf-instances?target-nf-type=UDM&requester-features=20&requester-nf-type=AUSF&service-names=nudm-ueau
  › Header: :scheme: http
  › Header: :authority: 127.0.0.10:7777
  › Header: 3gpp-sbi-sender-timestamp: Thu, 08 Feb 2024 16:01:08.545 GMT
  › Header: 3gpp-sbi-max-rsp-time: 10000
  › Header: user-agent: AUSF
  › Header: accept: application/json,application/problem+json
    [Full request URI: http://127.0.0.10:7777/nnrf-disc/v1/nf-instances?target-nf-type=UDM&requester-features=20&requester-nf-type=AUSF&service-names=nudm-ueau]
    [Response in frame: 2699]
```

```
0000  00 00 03 04 00 06 00 00  0
0010  45 00 00 b3 c9 ad 40 00  4
0020  7f 00 00 0a c0 c0 1e 61  a
0030  80 18 02 00 fe b0 00 00  0
0040  3e a8 94 e8 00 00 76 01  0
0050  62 aa ac 95 69 0c 82 31  c
0060  44 2a 3f c4 8e c9 8a 95  a
0070  a3 e2 c2 f6 b4 a8 49 6c  5
0080  7c 58 5e d6 95 09 2d 8b  5
0090  ec 3f 08 2d 9d cc 42 ad  5
00a0  d2 8e df 86 e7 7e 99 df  3
00b0  04 01 34 a0 5c b8 00 dc  0
00c0  e3 e6 e5
```

Figure 4.38 : Wireshark trace of UE, gNB with 5G CN

4.4 Setting up 5G Core with Open5gs NRF and other OAI NFs

In this approach, 5G NR and UE along with CN from open5gs and OAI, all running on a singular virtual machine. The 5G Core (5GC) system is seamlessly installed and operational on an Ubuntu 22.04 OS environment. The OpenAirInterface platform offers a selection of OAI 5GC deployment configurations and UPF variants, such as Slicing 5GC, Basic 5GC, and Minimalistic 5GC, presenting foundational options for deployment. For this particular project, the `DEPLOY_SA5G_BASIC` deployment mode has been chosen. This basic deployment configuration incorporates Network Functions (NFs) like AMF, SMF, NRF, UPF, UDM, AUSF, and UDR. Furthermore, these NFs are effortlessly deployed within Docker containers, facilitated by Docker-Compose, which serves as the deployment mechanism.

Therefore, installing Docker Engine and Docker Compose on the CN machine is the initial step in this direction. In order to prevent using the docker command in sudo mode, the user name must be added to the docker group. The next step is to download and copy the OAI 5G CN configuration files. After getting the config files, Docker images are pulled from Docker hub. Figure 4.39 illustrates the whole setup along with IPv4 addresses.

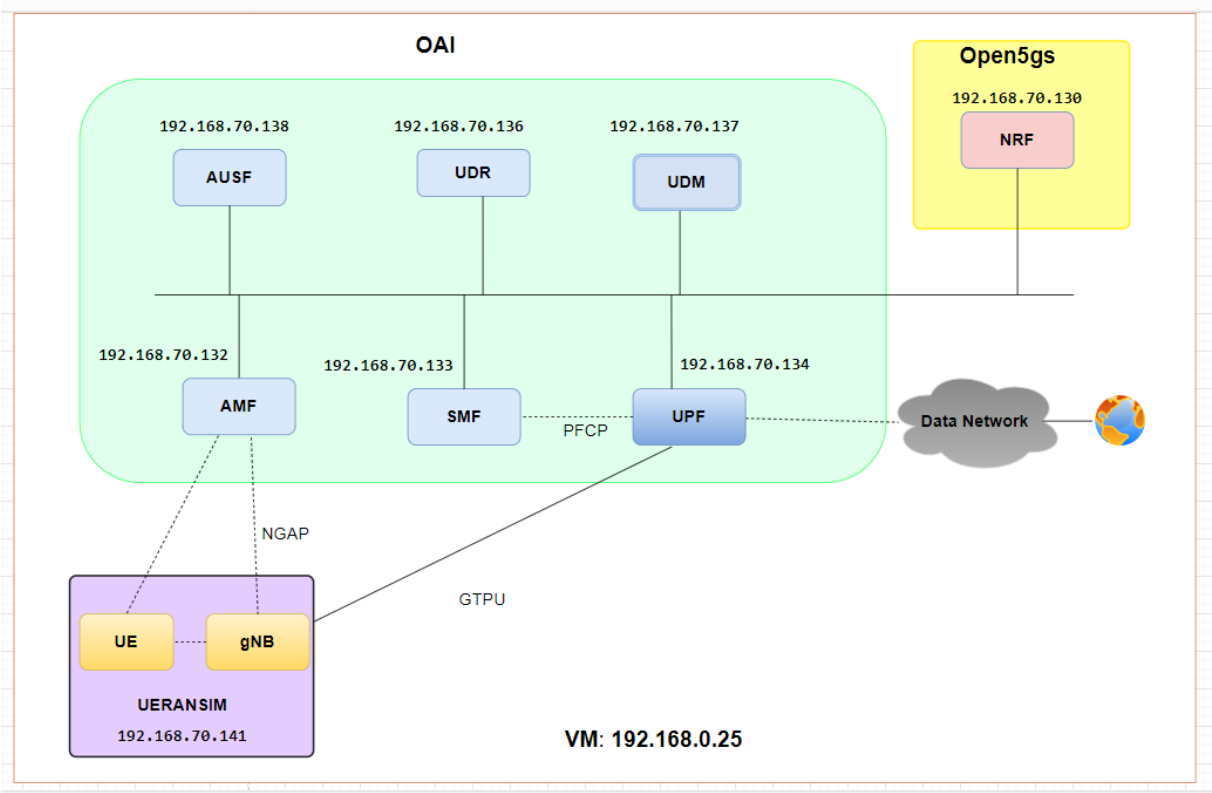


Figure 4.39 : Interface and function specific Architecture as per use case 3

4.4.1 Installation

Starting the setup requires initially installing Wireshark and Docker-compose on the host machine. For this purpose, Wireshark version 4.0.6 and Docker-compose version 1.29.2 are employed. This phase of implementation leverages Docker to facilitate the deployment of both OAI and Open5GS.

Installation with OAI:
For the configuration file of OAI 5G CN needs to be downloaded using the following command:

git clone --branch v2.0.0 **https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed.git**

After getting the config files, the next step is to pull the Docker images. In this implementation, the mysql:8.0 Docker image is pulled. In addition, the required NFs docker images for the basic OAI 5GC deployment are pulled. After pulling the required docker images, the building process initiated using the docker images.

```
#!/bin/bash
docker pull oaisoftwarealliance/oai-amf:v2.0.0
docker pull oaisoftwarealliance/oai-upf:v2.0.0
docker pull oaisoftwarealliance/oai-smf:v2.0.0
docker pull oaisoftwarealliance/oai-udr:v2.0.0
docker pull oaisoftwarealliance/oai-udm:v2.0.0
docker pull oaisoftwarealliance/oai-ausf:v2.0.0
docker pull oaisoftwarealliance/oai-nssf:v2.0.0
docker pull oaisoftwarealliance/oai-pcf:v2.0.0

# Utility image to generate traffic
docker pull oaisoftwarealliance/trf-gen-cn5g:latest
```

Listing 4.7: Commands for pulling and building OAI 5GCN docker images

Now, the 5G CN can be started by deploying the NF containers. Here, Docker containers are built using docker-compose file name "`docker-compose-basic-nrf.yaml`". The following image shows the deployment of the OAI 5G CN. The deployment occurs in the same order as following

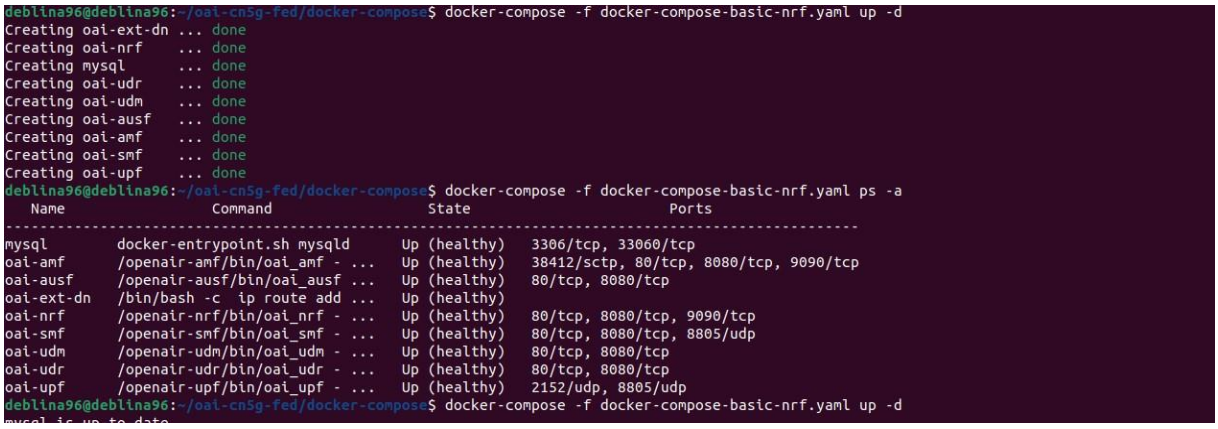


Figure 4.40: Deployment of OAI 5G CN

Now the association between different NFs can be seen by capturing the traffic in `demo-oai` interface. NFs. As soon as the CN is invoked, the `demo-oai` interface is created and packets will be captured on demo-oai bridge which is configured on the docker-compose-host machine. The associated IP addresses of different NFs are addressed in figure 4.41.

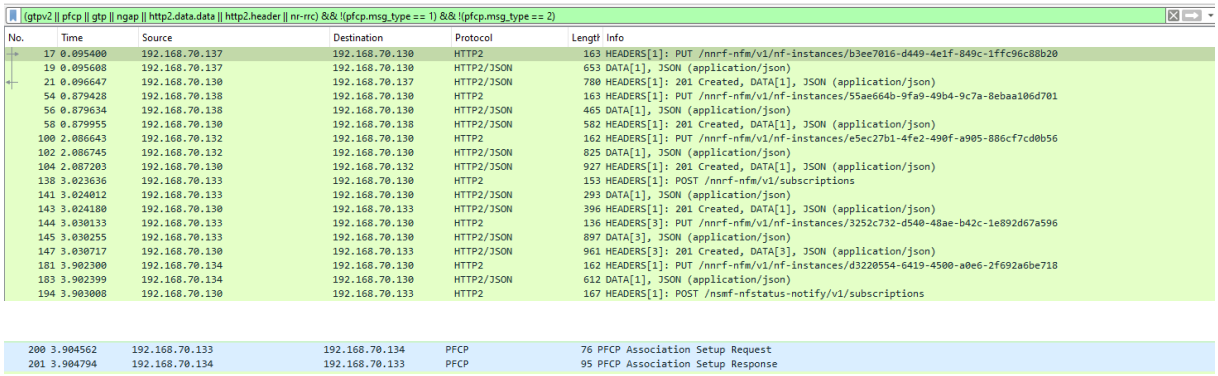


Figure 4.41: Association between different NFs of OAI 5G CN

The following command is used to stop the OAI 5G CN. First, all the containers will be stopped and then using the following command all containers will be deleted one by one.

```
#To stop the containers

docker-compose-host $: docker-compose -f docker-compose-basic-nrf.yaml down

#To stop the containers with zero graceful period
docker-compose-host $: docker-compose -f docker-compose-basic-nrf.yaml down -t 0
```

Listing 4.8: Commands for stopping OAI 5G CN

Installation with Open5gs:
For this installation, meeting the prerequisites of Docker Compose and Docker Engine versions is essential. The versions adhered to are listed below.

```
deblina96@deblina96:~$ docker compose version
Docker Compose version v2.17.3
deblina96@deblina96:~$ docker --version
Docker version 24.0.7, build afdd53b
```

Figure 4.42: Required version of docker and docker compose for Open5gs installation

The source code can be downloaded by using the below command . After downloading the source code from github, then the base image for open5gs is built by using the following command:

```
#To download source code

git clone https://github.com/herlesupreeth/docker_open5gs

# To build source image

  docker build --no-cache --force-rm -t docker_open5gs
```

Listing 4.9: Commands for downloading source code and docker image

For a smooth execution of the compose file, it is advisable to modify the values of "DOCKER_HOST_IP" and "UPF_ADVERTISE_IP" in .env file, which used to manage environment variables for the Docker containers in Open5gs. The DOCKER_HOST_IP is set to the Ip in which the virtual machine is running.

After configuring the host IP then by running docker-compose file name sa-deploy.yaml the NRF container is created from open5gs.

```
# For 5G deployment
docker compose -f sa-deploy.yaml build
# 5G Core Network deployment
docker compose -f sa-deploy.yaml up
```

Listing 4.9: Commands for deploying NRF from Open5gs

4.4.2 Configuration and Deployment in integrated environment

TEST_NETWORK needs to be set up to correspond with the IP address of the network where the containers are supposed to run. In this case we have use a similar network for both OAI and Open5gs, in order to make communication between containers part of a different open source project.

The first issue will arise when both of the docker-compose files try to access the same network, then "network pool overlap" error will occur. In order to resolve that one "`Common External Network`" name "`my_common_network`" is created using one particular subnet. This is aimed at facilitating communication between containers across different Docker Compose projects. This approach has delivered a unified network layer for all the services from different NFs, regardless of the Compose file they originate from.

Command for creating external network:

docker network create --driver bridge --subnet=192.168.70.128/24 my_common_network

```
deblina96@deblina96:~$ docker network inspect my_common_network
[
    {
        "Name": "my_common_network",
        "Id": "00a2472c34841c9a128f754deee7665929de686584a1b8e0a89f83f0b4c951cc",
        "Created": "2024-02-05T23:23:52.219539132Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "192.168.70.128/24",
                    "Gateway": "192.168.70.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

Figure 4.42: Details of custom external network

After the network has been established, the subsequent step involves incorporating this network into each Docker Compose file**.** Figure 4.43 demonstrates an example of how this external network is utilized. The complete configuration file details can be found in the Appendix section.

```
networks:
  my_common_network: #default:
    external: true
#    ipam:
```

Figure 4.43: Description of Docker Compose file

Now, after completing the previous section “Installation with Open5gs” , NRF deployed would be possible using the sa-deployment.yaml file. The further configuration details of NRF is present in “nrf_init.sh” script. Figure 4.44 illustrates the Open5gs NRF configuration using a Docker Compose file.

In this setup, the Service Communication Proxy (SCP) is disabled to establish direct communication between other Network Functions (NFs) from OAI with the NRF.

```
  nrf:
    image: docker_open5gs
    container_name: nrf
    env_file:
      - .env
    environment:
      - COMPONENT_NAME=nrf-1
    volumes:
      - ./nrf:/mnt/nrf
      - ./log:/open5gs/install/var/log/open5gs
#       - /etc/timezone:/etc/timezone:ro
#       - /etc/localtime:/etc/localtime:ro
    expose:
      - "80/tcp"
      - "9090/tcp"
    networks:
      my_common_network: #default:      #default:
        ipv4_address: 192.168.70.130 #${NRF_IP}
#  scp:
#    image: docker_open5gs
#    container_name: scp
#    env_file:
#      - .env
#    environment:
#      - COMPONENT_NAME=scp-1
#    volumes:
#      - ./scp:/mnt/scp
#      - ./log:/open5gs/install/var/log/open5gs
#      - /etc/timezone:/etc/timezone:ro
#      - /etc/localtime:/etc/localtime:ro
#    expose:
#      - "7777/tcp"
#    networks:
#      default:
#        ipv4_address: ${SCP_IP}
```

Figure 4.44: Configuration of NRF in Open5gs

```
deblina96@deblina96:~/oai-cn5g-fed/docker-compose$ docker compose -f sa-deploy.yaml up
[+] Running 1/1
 ✓ Container nrf  Recreated
Attaching to nrf
nrf  | Deploying component: 'nrf-1'
nrf  | Open5GS daemon v2.6.6
nrf  |
nrf  | 02/05 14:58:08.747: [app] INFO: Configuration: '/open5gs/install/etc/open5gs/nrf.yaml' (../lib/app/ogs-init.c:126)
nrf  | 02/05 14:58:08.752: [app] INFO: File Logging: '/open5gs/install/var/log/open5gs/nrf.log' (../lib/app/ogs-init.c:129)
nrf  | 02/05 14:58:08.770: [sbi] INFO: nghttp2_server() [http://192.168.70.130]:8080 (../lib/sbi/nghttp2-server.c:395)
nrf  | 02/05 14:58:08.771: [app] INFO: NRF initialize...done (../src/nrf/app.c:31)
```

Figure 4.45: Open5gs NRF running log through container nrf

```
[+] Running 1/1
 ✓ Container nrf  Stopped
canceled
deblina96@deblina96:~/oai-cn5g-fed/docker-compose$ docker compose -f sa-deploy.yaml down
[+] Running 1/0
 ✓ Container nrf  Removed
```

Figure 4.46: Command for stopping and removing nrf container

After the "`nrf`" container successfully transitions to an active and operational state, the Open Air Interface (OAI) is initiated in a separate window, employing the configuration file outlined previously in Figure 4.31. However, in this iteration, the "`oai-nrf`" container is omitted, as the NRF is now derived from Open5GS, utilizing a distinct Dockerfile. The illustration below depicts the operational status of additional containers, such as UDR, UDM, AMF, SMF, and UPF, which are executed from OAI.

```
deblina96@deblina96:~/oai-cn5g-fed/docker-compose$ docker-compose -f docker-compose-basic-nrf.yaml up -d
Creating oai-ext-dn ... done
Creating mysql      ... done
Creating oai-udr    ... done
Creating oai-udm    ... done
Creating oai-ausf   ... done
Creating oai-amf    ... done
Creating oai-smf    ... done
Creating oai-upf    ... done
deblina96@deblina96:~/oai-cn5g-fed/docker-compose$
```

Figure 4.47: Running containers from OAI

```
deblina96@deblina96:~/oai-cn5g-fed/docker-compose$ docker ps
CONTAINER ID   IMAGE                                      COMMAND                  CREATED             STATUS                    PORTS                                          NAMES
c5720847325c   docker_open5gs                             "/bin/sh -c /open5gs…"   About a minute ago  Up About a minute         80/tcp, 9090/tcp                               nrf
279b101cdcd4   oaisoftwarealliance/oai-smf:v2.0.0         "/openair-smf/bin/oa…"   3 minutes ago       Up 3 minutes (healthy)    80/tcp, 8080/tcp, 8805/udp                     oai-smf
7fed845e12f2   oaisoftwarealliance/oai-amf:v2.0.0         "/openair-amf/bin/oa…"   3 minutes ago       Up 3 minutes (healthy)    80/tcp, 8080/tcp, 9090/tcp, 38412/sctp         oai-amf
6f0687bf4b9e   oaisoftwarealliance/oai-ausf:v2.0.0        "/openair-ausf/bin/o…"   3 minutes ago       Up 3 minutes (healthy)    80/tcp, 8080/tcp                               oai-ausf
46994f369a573  oaisoftwarealliance/oai-udm:v2.0.0         "/openair-udm/bin/oa…"   3 minutes ago       Up 3 minutes (healthy)    80/tcp, 8080/tcp                               oai-udm
0709fe009972   oaisoftwarealliance/oai-udr:v2.0.0         "/openair-udr/bin/oa…"   3 minutes ago       Up 3 minutes (healthy)    80/tcp, 8080/tcp                               oai-udr
d6296f761db1   mysql:8.0                                  "docker-entrypoint.s…"   3 minutes ago       Up 3 minutes (healthy)    3306/tcp, 33060/tcp                            mysql
e95dd524d804   oaisoftwarealliance/trf-gen-cn5g:latest    "/bin/bash -c ' ip r…"   3 minutes ago       Up 3 minutes (healthy)                                                   oai-ext-dn
deblina96@deblina96:~/oai-cn5g-fed/docker-compose$
```

Figure 4.48: All running containers from OAI & Open5gs

Once all the containers are in running state then by capture the traffic using Wireshark it is possible to check the association between different NFs. The provided packet capture (pcap) file clearly demonstrates that the NFs originating from OAI successfully complete registration with the Open5gs Network Repository Function (NRF) within this integrated setup.

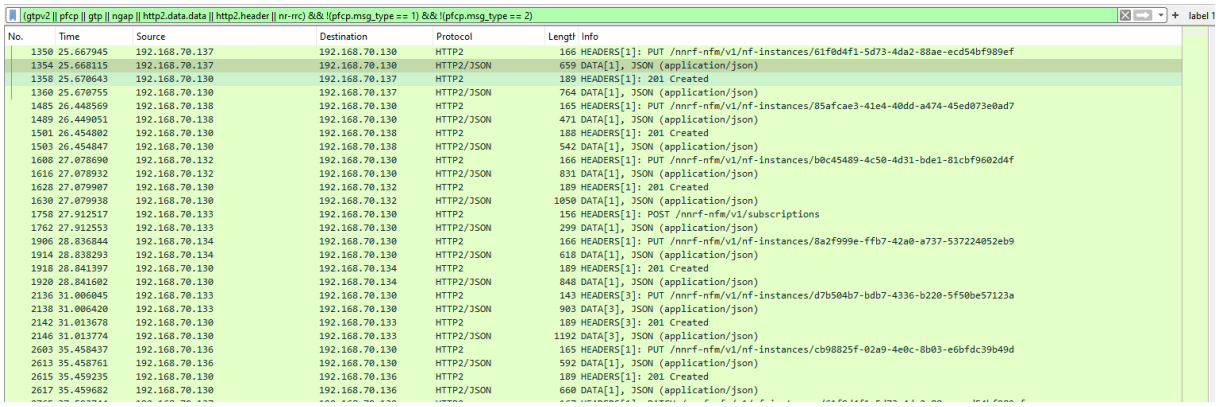


Figure 4.49: Wireshark trace capture at NRF interface in integrated environment

Now, illustrated by Figure 4.49**,** it is observed that numerous packets are originating from the NRF and vice versa, in order to have a better understanding of the registration process certain packets are expanded below to analyze the JSON parameters contained within the packets. Few important sections are also highlighted.

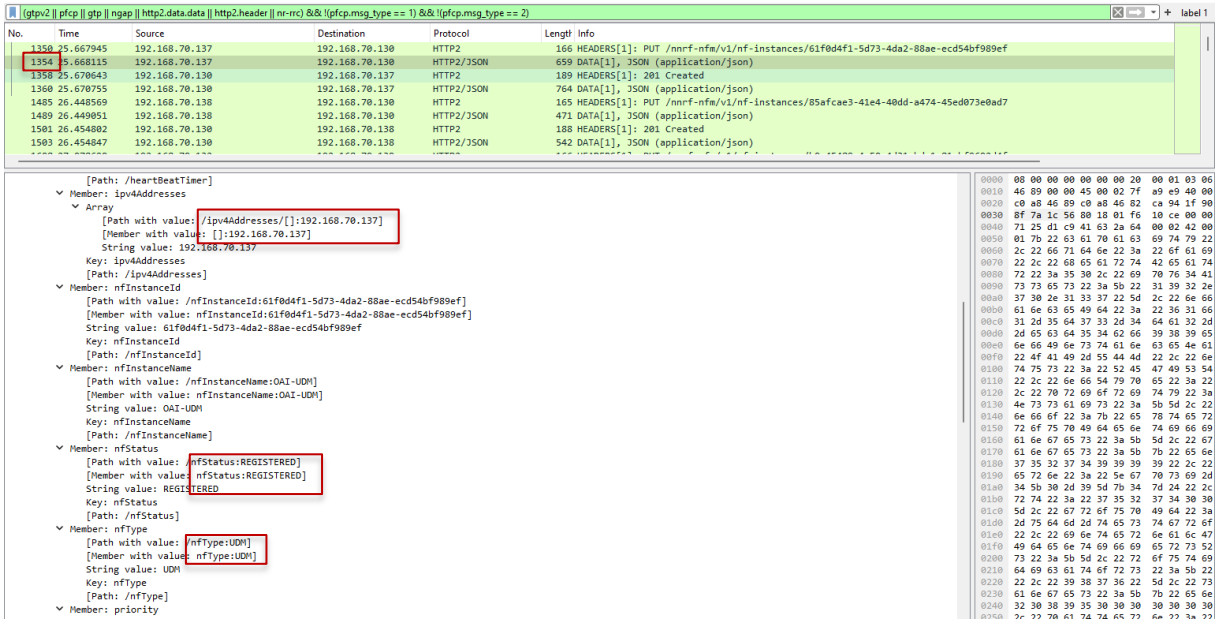


Figure 4.50: Wireshark trace of extended packet 1354

The information provided in "packet 1354" indicates that the UDM, with the IP address 192.168.70.137, is initiating a registration request to the Network Function Repository Function (NRF) at the IP address 192.168.70.130. This registration attempt is being made by supplying its IP address within an array through the

"ipv4address" member. Additionally, the UDM is seeking to confirm its registration status with the NRF by setting the "nfStatus" member to "REGISTERED". This implies that UDM is asking the NRF to acknowledge its status as an active and registered network function within the 5G system.

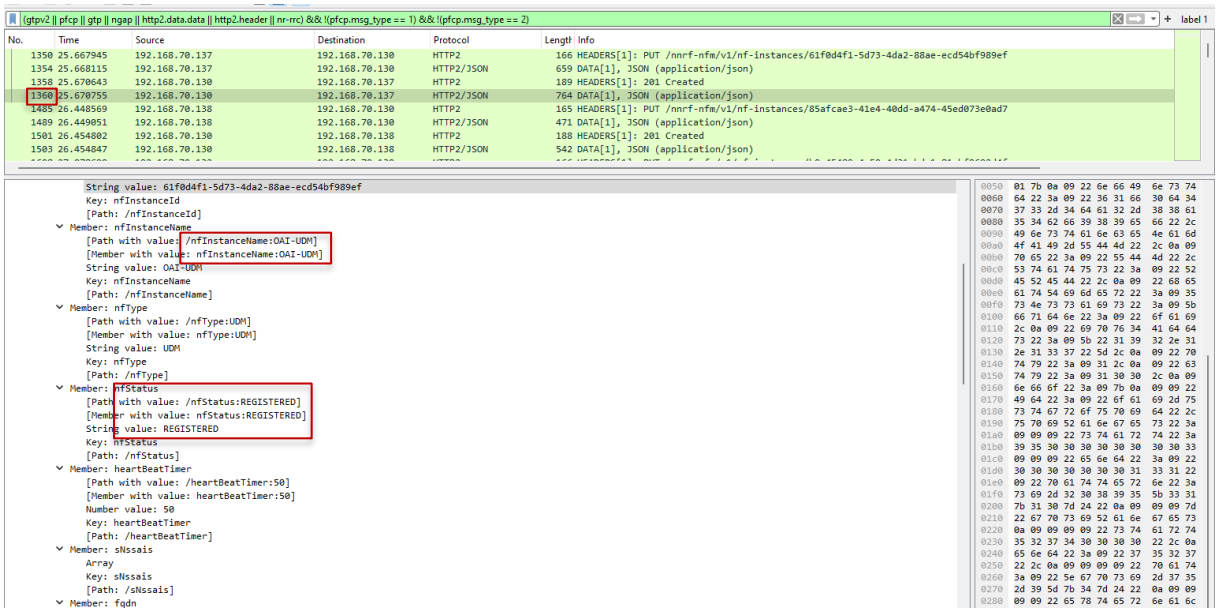


Figure 4.51: Wireshark trace of extended packet 1360

Packet 1360 is a display capture of the reply from NRF to UDM. Here it can be seen a successful response to the registration request. The NRF, recognizing the request from UDM at IP address 192.168.70.137, and processed the registration and is confirming the UDM's status as "REGISTERED."

The reply also contains several key pieces of information such as “instance ID”, which uniquely identifies the UDM in the NRF's records. The “nfStatus” with the value “REGISTERED” confirms that the UDM's status in the network is now registered. The “heartBeatTimer” value suggests there's a keep-alive timer mechanism in place, allowing the NRF to expect regular updates in form of "heartbeats" from the UDM, ensuring it remains active and responsive. The “sNssais” array indicates service slice selection information, which is used in networks to identify the service slices the UDM is associated with or can handle.

The same successful registration response for AMF can be seen through packet 1630.

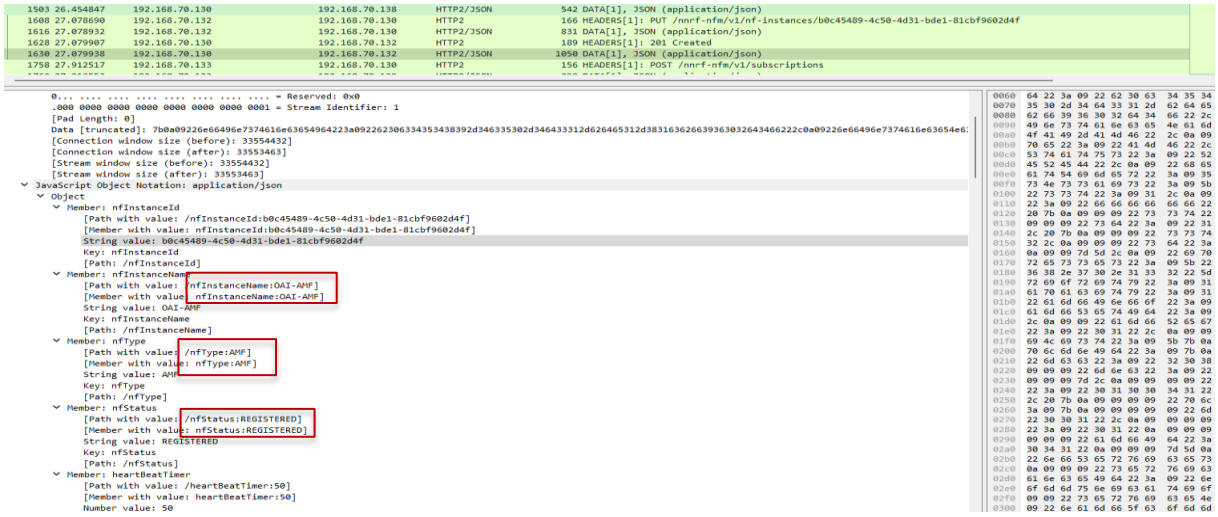


Figure 4.52: Wireshark trace for AMF reply from NRF through packet 1630

The registration of all these NFs to NRF can also be confirmed by logs captured in different Docker Container. The logs from Docker container “oai-amf” and “oai-smf” are mentioned below.

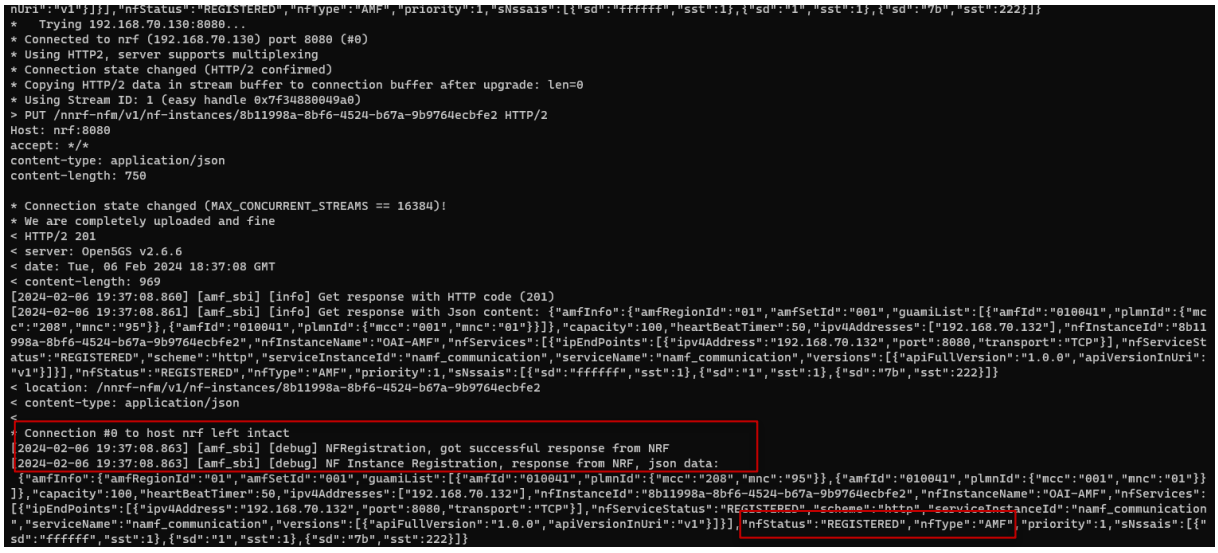


Figure 4.53: Log showing AMF registration at NRF

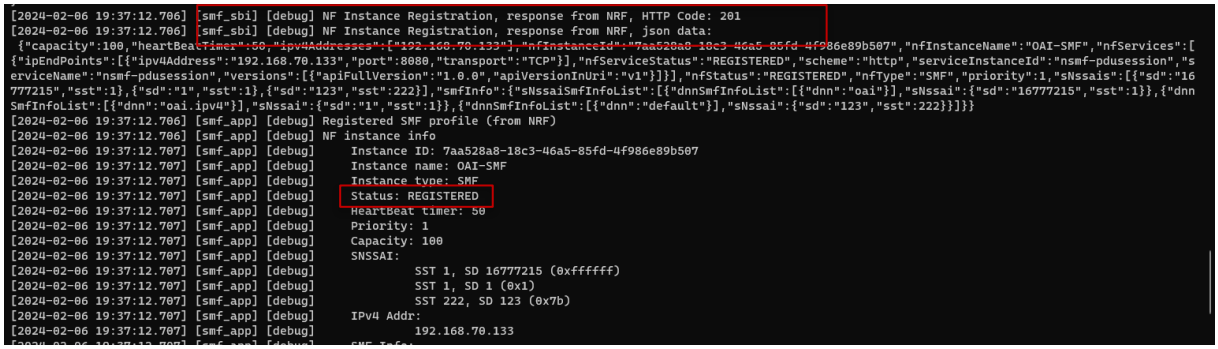


Figure 4.54: Log showing SMF registration at NRF

4.4.3 Realisation Matrix

For assessing the feature support and interoperability of the various OpenAirInterface (OAI) components with Open5GS NRF, this matrix serves as an essential analytical tool.

Open5gs	OAI					
NRF	AMF	SMF	AUSF	UDM	UDR	UPF
Service discovery and Orchestration	✕	✕	✕	✕	✕	✕
NF registration	✓	✓	✓	✓	✓	✓
NF Subscription add in NRF	✓	✓	✓	✓	✓	✓
NF profile Update in NRF & Heartbeat timer	✕	✕	✕	✕	✕	✕

4.4.4 Observations

Observations and analysis on use case 2:

From the above realization it is visible that OAI NFs are unable to send heartbeat message to Open5gs NRF once the initial registration is completed. This is realized from the pcap file, where NRF is sending “400 Bad Request” as response of the HTTP2 PATCH request initiated by NFs from Open5gs. As per the 3GPP 5G core specification

there is one mechanism named as the Heartbeat / keepalive mechanism, to indicate the NRF that the other NF in the 5G network are still there and working.

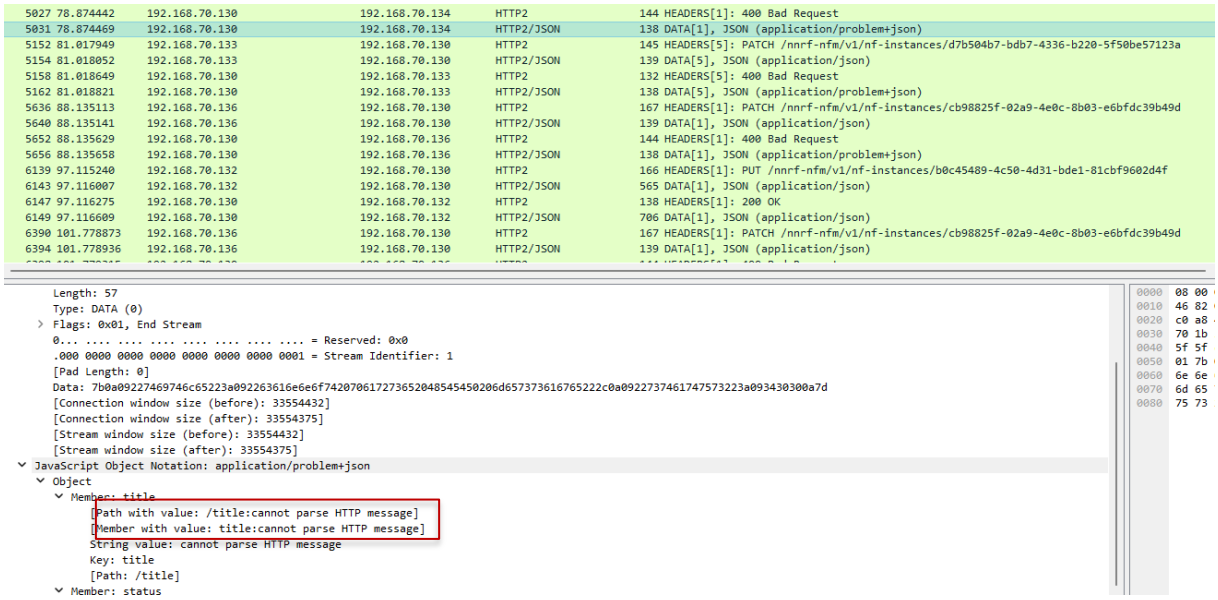


Figure 4.55: Wireshark trace for bad request and its JSON data

HTTP PATCH request is always sent with a JSON payload. Open5GS NRF is in line with the 3GPP specifications for the NRF's service-based interface. The JSON contain the target NF's (in this case which is UPF for the packet 5031) NF profile, including its NF type, NF instance ID, IP addresses, Heartbeat timer and other supported features as per the 5G Core Network schema.

Further error details can be found by capturing the log at NRF interface of Open5gs.

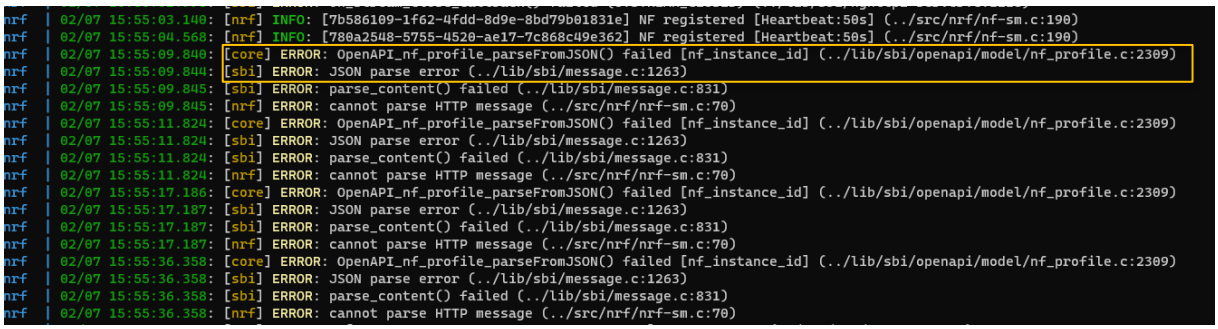


Figure 4.56: Error details from nrf container log from Open5gs

The specific function which is failing is "`OpenAPI_nf_profile_parseFromJSON()`", which creates an `OpenAPI_nf_profile_t` object from JSON input. The function can be found in Open5gs source code in the mentioned file "open5gs/lib/sbi/openapi/model/nf_profile.c".

The `nf_instance_id` is a crucial identifier for network function instances, and it follows a valid UUID (Universally Unique Identifier) version 4, as described in IETF RFC 4122. In both Open5GS and OpenAirInterface, the nf_instance_id is a unique identifier for a Network Function instance. To ensure that nf_instance_id is unique, It is generated using a method UUID 4 that guarantees uniqueness, This a 128-bit number used to identify information in computer systems.

As the error message suggests from Figure 4.56 that the OpenAPI_nf_profile_parseFromJSON()" function, which is a part of Open5gs source code, cannot parse the JSON data as it is unable to identify **nf_instance_id** . This validation is essential for NRF so that it can update NF profile within its database.

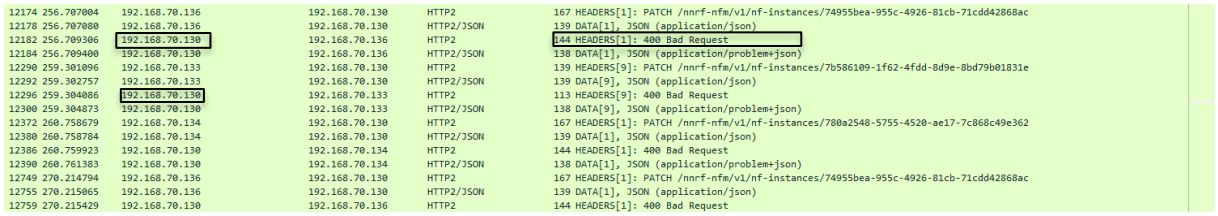


Figure 4.57: Error details from Wireshark trace

Moreover, when OAI is sending PATCH request to Open5gs by attaching the JSON payload in the HTTP2 request, open5gs is unable to recognize the "nf_instance_Id" which was generated by the OAI through software using a UUID function for the PATCH request and sent to Open5gs. Therefore the error is occurring

5GRAN and 5GC singaling:

After the initial registration, the subsequent step involves establishing a connection between the 5G Radio Access Network (RAN) and User Equipment (UE) with the 5G Core Network (CN) to enable data network access. This is accomplished using a Docker Compose file named "docker-compose-ueransim-vpp," which also utilizes the previously established custom external network. The deployment of UERANSIM can be achieved through Docker within the same network, as demonstrated in the below figure.

From Figure 4.58, the log of the ueransim container is visible , where first gNB sends an NG setup request providing the necessary information to AMF. AMF responds to the requests by sending the NG setup response. After successful response, the NG setup procedure is built as shown in figure 4.58. Thus successful setup of N1/N2 interface communication with 5G CN.

Once the NG Setup procedure is successful, then UE will try to connect to CN providing the necessary details of UE such as PLMN, TAC etc as shown in figure 4.46. Then the authentication process of UE is done by using AUSF and UDM in the CN. But, as per log of UE, it is visible that NAS error is coming by saying "Can not be accepted as UE doesn't have an emergency."

This happened as the UERANSIM Docker Compose file which is currently present in OAI repository advice that this version of UERANSIM currently does not support any integrity and ciphering algorithm NIA0, NEA0 respectively. Hence the security process is failing and not able to establish a successful PDU session.

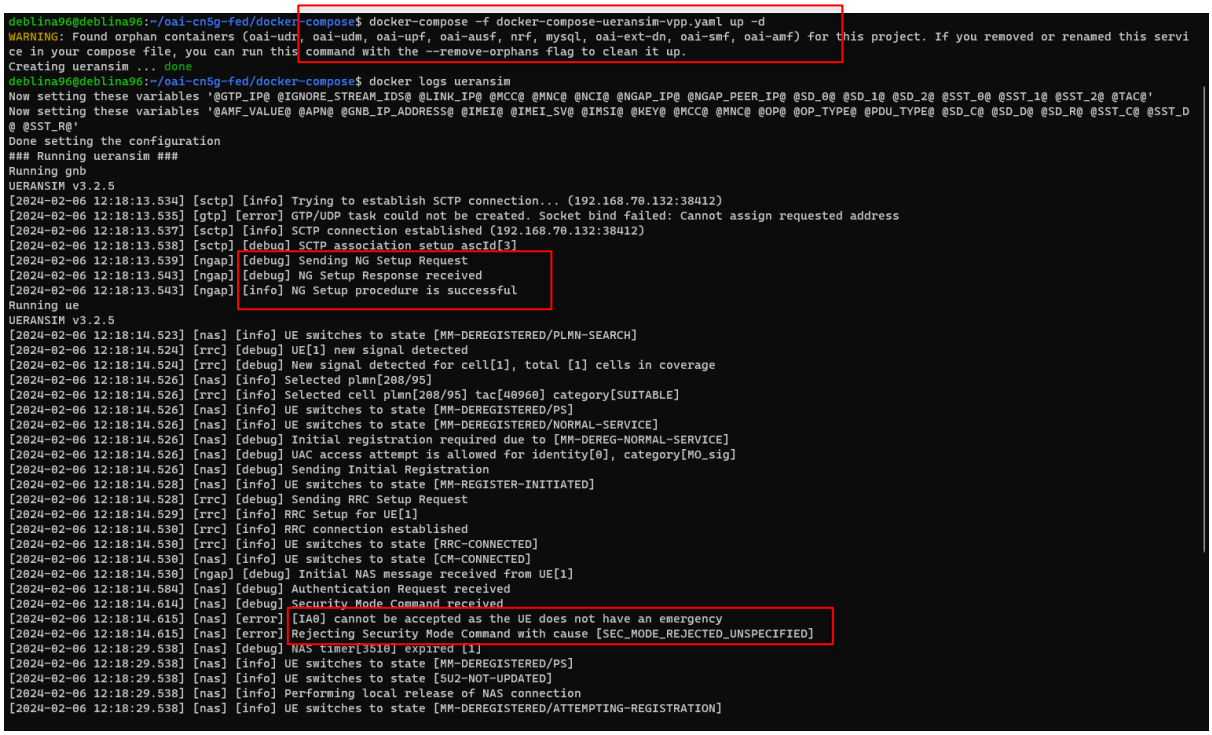


Figure 4.58: Log details of gNB and UE from ueransim container

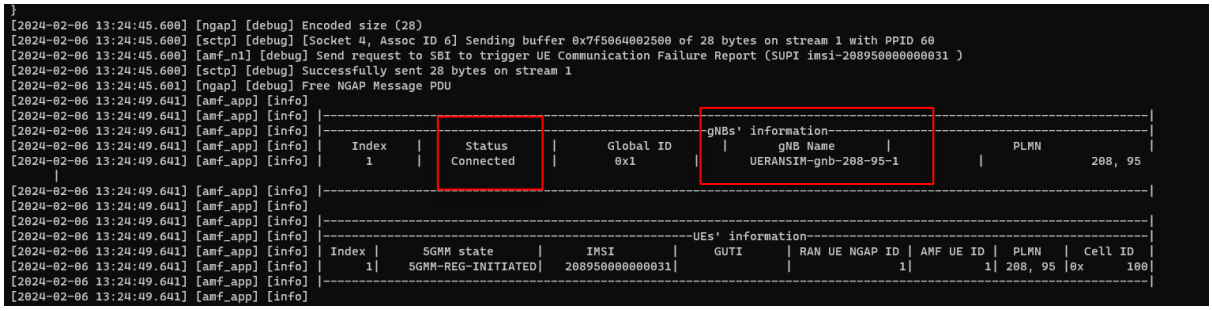


Figure 4.59: Log details from oai-amf container after attaching ueransim container

The successful execution of UE in OAI project could be done by using other Dockerfile. One of the example could be using COTSUE which is also mentioned in OAI repository. But in integrated environment the UE registration with Core Network is unable to achieve because of the observations made earlier in section 4.4.4.

4.5 Setting up 5G Core with OAI UDR and other Open5gs NFs

In this architecture, the Core Network (CN) components from Open5GS and OpenAirInterface (OAI) are orchestrated on a single virtual machine. The entire 5G Core (5GC) ecosystem is built in an Ubuntu 22.04 OS environment. The OpenAirInterface framework extends a variety of deployment configurations for the OAI 5GC, alongside multiple User Plane Function (UPF) alternatives such as the Slicing 5GC, Basic 5GC, and the streamlined Minimalistic 5GC, each laying the groundwork for versatile deployment strategies. For the scope of this endeavour, we have opted for the DEPLOY_SA5G_BASIC mode, a fundamental yet comprehensive setup that encompasses an array of Network Functions (NFs) including the AMF, SMF, NRF, UPF, UDM, AUSF, and UDR. These NFs are containerised, with Docker-Compose paving the way for a smooth and efficient deployment process.

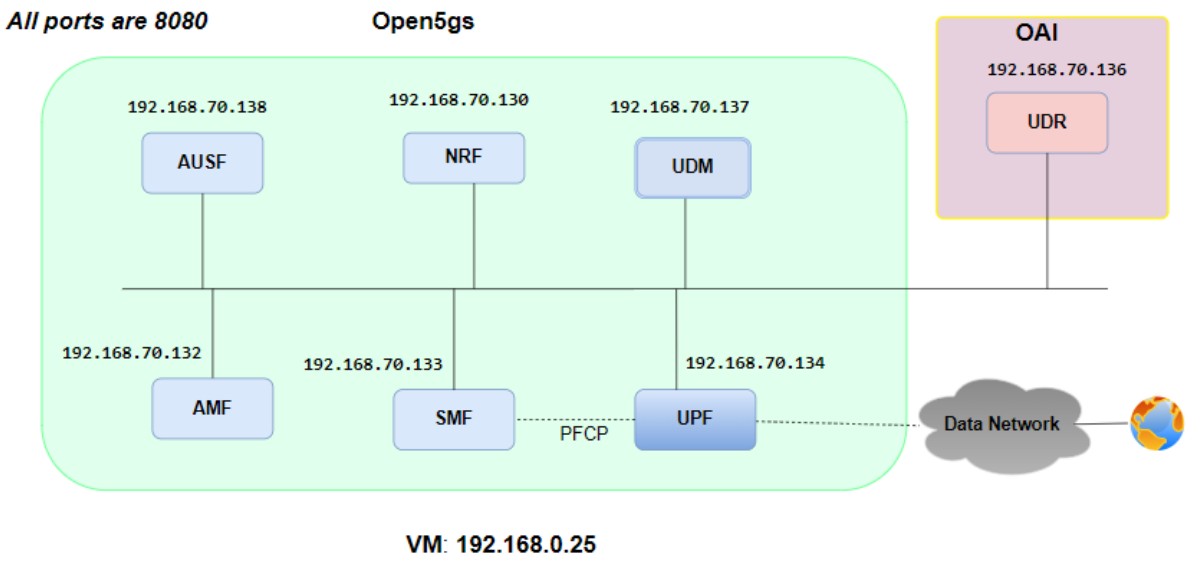


Figure 4.60: Interface and function specific Architecture as per use case 4

The procedure for installing OpenAirInterface and Open5gs using Docker and a Docker Compose file was previously covered in section 4.4.1. The same procedures have been conducted with the new UDR setup here as well.

4.5.1 Configuration

The subnet that was previously assigned to the custom network which was created in figure 4.42 is also used here. All NFs are operational with IP addresses inside the same subnet's range.

```
version: '3.8'
services:
    mysql:
        container_name: "mysql"
        image: mysql:8.0
        volumes:
            - ./database/oai_db2.sql:/docker-entrypoint-initdb.d/oai_db.sql
            - ./healthscripts/mysql-healthcheck2.sh:/tmp/mysql-healthcheck.sh
        environment:
            - TZ=Europe/Paris
            - MYSQL_DATABASE=oai_db
            - MYSQL_USER=test
            - MYSQL_PASSWORD=test
            - MYSQL_ROOT_PASSWORD=linux
        healthcheck:
            test: /bin/bash -c "/tmp/mysql-healthcheck.sh"
            interval: 10s
            timeout: 5s
            retries: 30
        networks:
            my_common_network:          #public_net:
                ipv4_address: 192.168.70.131

    oai-udr:
        container_name: "oai-udr"
        image: oaisoftwarealliance/oai-udr:v2.0.0
        expose:
            - 80/tcp
            - 8080/tcp
        volumes:
            - ./conf/basic_nrf_config.yaml:/openair-udr/etc/config.yaml
        environment:
            - TZ=Europe/Paris
#        depends_on:
            - mysql
#           - oai-nrf
        networks:
            my_common_network: #public_net:
                ipv4_address: 192.168.70.136
```

Figure 4.60: UDR configuration in OAI

OAI use mysql database, whereas Open5gs use mongodb as its backend DB. In order to work with UDR, setting up a database it a necessity as UDR is responsible for storing and managing subscriber data.

`MYSQL_DATABASE` defines the name of the database to be created as soon as the MySQL container is first launched. After that, this database will be available for use with UDR by using the preconfigured login and password.

Once the configuration is done at OAI side, UDR from Open5gs side will be disabled from the Docker Compose file used for Open5gs containers.

```
deblina96@deblina96:~/oai-cn5g-fed/docker-compose$ docker compose -f sa-deploy2.yaml up
[+] Running 6/6
 ✓ Container nrf   Created
 ✓ Container udm   Created
 ✓ Container ausf  Created
 ✓ Container amf   Created
 ✓ Container smf   Created
 ✓ Container upf   Created
```

Figure 4.60: Containers running from Open5gs

```
deblina96@deblina96:~/oai-cn5g-fed$ cd docker-compose
deblina96@deblina96:~/oai-cn5g-fed/docker-compose$ docker-compose -f docker-compose-basic-nrf1.yaml up
WARNING: Found orphan containers (udm, amf, ausf, upf, smf, nrf) for this project. If you removed or renamed this service in your compose file, you can run this comman
d with the --remove-orphans flag to clean it up.
Creating oai-udr   ... done
Creating mysql     ... done
Creating oai-ext-dn ... done
Attaching to oai-udr, mysql, oai-ext-dn
```

Figure 4.60: Containers running from OAI

As soon as the containers started running it is possible to see the communication between containers through logs as well as Wireshark capture.

```
nrf    | Open5GS daemon v2.6.6
nrf    |
nrf    | 02/11 19:13:27.532: [app] INFO: Configuration: '/open5gs/install/etc/open5gs/nrf.yaml' (../lib/app/ogs-init.c:126)
nrf    | 02/11 19:13:27.532: [app] INFO: File Logging: '/open5gs/install/var/log/open5gs/nrf.log' (../lib/app/ogs-init.c:129)
nrf    | 02/11 19:13:27.539: [sbi] INFO: nghttp2_server() [http://192.168.70.130]:8080 (../lib/sbi/nghttp2-server.c:395)
nrf    | 02/11 19:13:27.553: [app] INFO: NRF initialize...done (../src/nrf/app.c:31)
ausf   | Deploying component: 'ausf-1'
ausf   | Open5GS daemon v2.6.6
ausf   |
ausf   | 02/11 19:13:27.982: [app] INFO: Configuration: '/open5gs/install/etc/open5gs/ausf.yaml' (../lib/app/ogs-init.c:126)
ausf   | 02/11 19:13:27.983: [app] INFO: File Logging: '/open5gs/install/var/log/open5gs/ausf.log' (../lib/app/ogs-init.c:129)
ausf   | 02/11 19:13:27.985: [sbi] INFO: NF Service [nausf-auth] (../lib/sbi/context.c:1438)
ausf   | 02/11 19:13:27.985: [sbi] INFO: nghttp2_server() [http://192.168.70.138]:7777 (../lib/sbi/nghttp2-server.c:395)
ausf   | 02/11 19:13:27.986: [app] INFO: AUSF initialize...done (../src/ausf/app.c:31)
udm    | Deploying component: 'udm-1'
udm    | Open5GS daemon v2.6.6
udm    |
udm    | 02/11 19:13:28.089: [app] INFO: Configuration: '/open5gs/install/etc/open5gs/udm.yaml' (../lib/app/ogs-init.c:126)
udm    | 02/11 19:13:28.089: [app] INFO: File Logging: '/open5gs/install/var/log/open5gs/udm.log' (../lib/app/ogs-init.c:129)
udm    | 02/11 19:13:28.091: [sbi] INFO: NF Service [nudm-ueau] (../lib/sbi/context.c:1438)
udm    | 02/11 19:13:28.091: [sbi] INFO: NF Service [nudm-uecm] (../lib/sbi/context.c:1438)
udm    | 02/11 19:13:28.091: [sbi] INFO: NF Service [nudm-sdm] (../lib/sbi/context.c:1438)
udm    | 02/11 19:13:28.092: [sbi] INFO: nghttp2_server() [http://192.168.70.137]:7777 (../lib/sbi/nghttp2-server.c:395)
udm    | 02/11 19:13:28.092: [app] INFO: UDM initialize...done (../src/udm/app.c:31)
amf    | Deploying component: 'amf-1'
amf    | Open5GS daemon v2.6.6
amf    |
amf    | 02/11 19:13:28.444: [app] INFO: Configuration: '/open5gs/install/etc/open5gs/amf.yaml' (../lib/app/ogs-init.c:126)
amf    | 02/11 19:13:28.446: [app] INFO: File Logging: '/open5gs/install/var/log/open5gs/amf.log' (../lib/app/ogs-init.c:129)
amf    | 02/11 19:13:28.454: [metrics] INFO: metrics_server() [http://192.168.70.132]:9091 (../lib/metrics/prometheus/context.c:299)
amf    | 02/11 19:13:28.454: [sbi] INFO: NF Service [namf-comm] (../lib/sbi/context.c:1438)
amf    | 02/11 19:13:28.456: [sbi] INFO: nghttp2_server() [http://192.168.70.132]:7777 (../lib/sbi/nghttp2-server.c:395)
amf    | 02/11 19:13:28.456: [amf] INFO: ngap_server() [192.168.70.132]:38412 (../src/amf/ngap-sctp.c:61)
amf    | 02/11 19:13:28.458: [sctp] INFO: AMF initialize...done (../src/amf/app.c:33)
```

Figure 4.60: Log snippet from Open5gs console

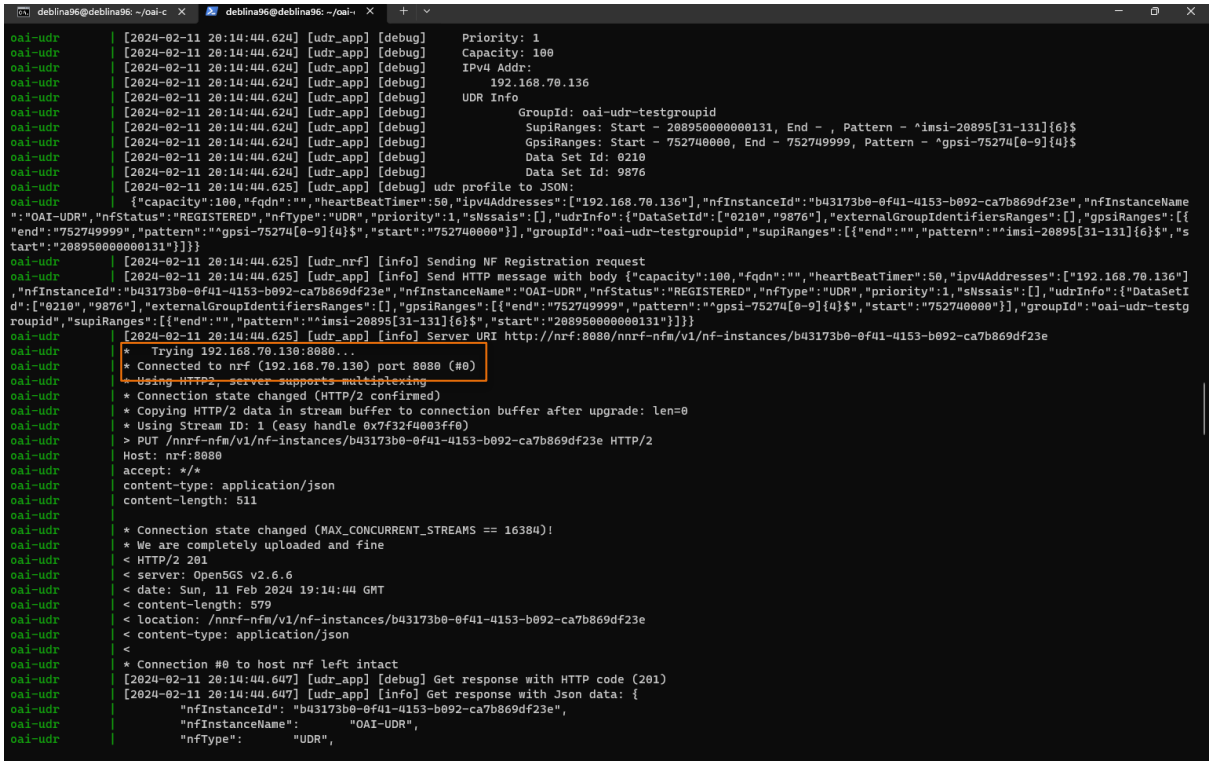


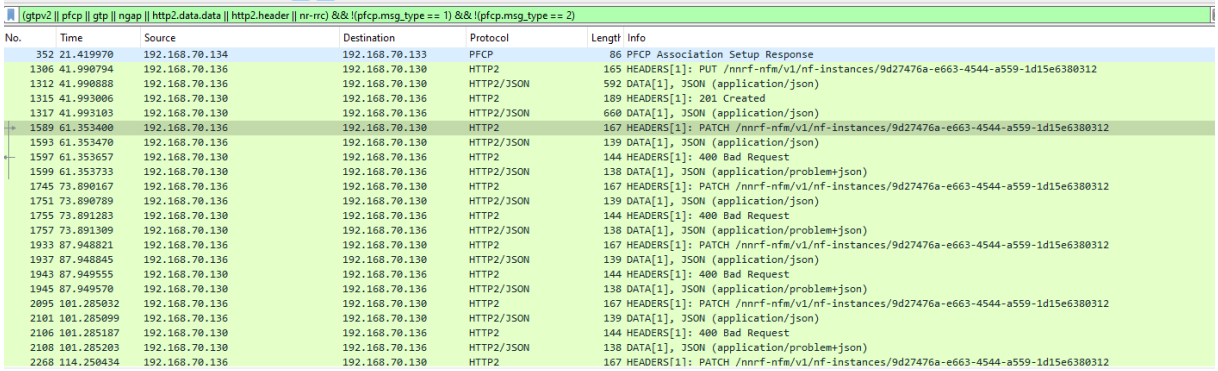
Figure 4.60: Log snippet from OAI console

From log it is visible that UDR is successfully registered with Open5gs NRF, which was achieved earlier also with use case 1. But the then If we analysis the Wireshirk capture we can see the association, which is mentioned

4.5.2 Realisation Matrix

OAI UDR	Open5gs UDM	User Equipment
Communication with UDM	✕	NA
Storage and retrieval of subscription data by the UDM	NA	✕

4.5.3 Observation



(gtpv2 || pfcp || gtp || ngap || http2.data.data || http2.header || nr-rrc) && !(pfcp.msg_type == 1) && !(pfcp.msg_type == 2)

No.	Time	Source	Destination	Protocol	Length	Info
352	21.419970	192.168.70.134	192.168.70.133	PFCP	86	PFCP Association Setup Response
1306	41.990794	192.168.70.136	192.168.70.130	HTTP2	165	HEADERS[1]: PUT /nnrf-nfm/v1/nf-instances/9d27476a-e663-4544-a559-1d15e6380312
1312	41.990888	192.168.70.136	192.168.70.130	HTTP2/JSON	592	DATA[1], JSON (application/json)
1315	41.993006	192.168.70.130	192.168.70.136	HTTP2	189	HEADERS[1]: 201 Created
1317	41.993103	192.168.70.130	192.168.70.136	HTTP2/JSON	660	DATA[1], JSON (application/json)
1589	61.353400	192.168.70.136	192.168.70.130	HTTP2	167	HEADERS[1]: PATCH /nnrf-nfm/v1/nf-instances/9d27476a-e663-4544-a559-1d15e6380312
1593	61.353470	192.168.70.136	192.168.70.130	HTTP2/JSON	139	DATA[1], JSON (application/json)
1597	61.353657	192.168.70.130	192.168.70.136	HTTP2	144	HEADERS[1]: 400 Bad Request
1599	61.353733	192.168.70.130	192.168.70.136	HTTP2/JSON	138	DATA[1], JSON (application/problem+json)
1745	73.890167	192.168.70.136	192.168.70.130	HTTP2	167	HEADERS[1]: PATCH /nnrf-nfm/v1/nf-instances/9d27476a-e663-4544-a559-1d15e6380312
1751	73.890789	192.168.70.136	192.168.70.130	HTTP2/JSON	139	DATA[1], JSON (application/json)
1755	73.891283	192.168.70.130	192.168.70.136	HTTP2	144	HEADERS[1]: 400 Bad Request
1757	73.891309	192.168.70.130	192.168.70.136	HTTP2/JSON	138	DATA[1], JSON (application/problem+json)
1933	87.948821	192.168.70.136	192.168.70.130	HTTP2	167	HEADERS[1]: PATCH /nnrf-nfm/v1/nf-instances/9d27476a-e663-4544-a559-1d15e6380312
1937	87.948845	192.168.70.136	192.168.70.130	HTTP2/JSON	139	DATA[1], JSON (application/json)
1943	87.949555	192.168.70.130	192.168.70.136	HTTP2	144	HEADERS[1]: 400 Bad Request
1945	87.949570	192.168.70.130	192.168.70.136	HTTP2/JSON	138	DATA[1], JSON (application/problem+json)
2095	101.285032	192.168.70.136	192.168.70.130	HTTP2	167	HEADERS[1]: PATCH /nnrf-nfm/v1/nf-instances/9d27476a-e663-4544-a559-1d15e6380312
2101	101.285099	192.168.70.136	192.168.70.130	HTTP2/JSON	139	DATA[1], JSON (application/json)
2106	101.285187	192.168.70.130	192.168.70.136	HTTP2	144	HEADERS[1]: 400 Bad Request
2108	101.285203	192.168.70.130	192.168.70.136	HTTP2/JSON	138	DATA[1], JSON (application/problem+json)
2268	114.250434	192.168.70.136	192.168.70.130	HTTP2	167	HEADERS[1]: PATCH /nnrf-nfm/v1/nf-instances/9d27476a-e663-4544-a559-1d15e6380312

Figure 4.60: Wireshark trace between UDM and NRF

The identical problem has been seen with the HTTP2 PATCH request throughout this realisation. It has been noted that following the initial registration, there are no more stages or opportunities for communication between various NFs. The section before this one contains the error information in details.

5 Summary and Perspectives

Through the implementation of several use cases, the project's goal- testing the integration between the two open-source projects, OAI and Open5gs have been accomplished. This section highlight overall interoperability challenges encountered when integrating network functions (NFs) from different distributions within one 5G core architecture, specifically within the Network Repository Function interactions.

Several obstacles have come up during the conduction of this research, from putting up different open-source projects to changing configurations and looking into interoperability scenarios. Numerous test setups have been made to determine which version of MongDB is compatible with the most recent version of Open5gs due to compatibility issues between MongoDB and Open5gs. Chapter 4 mentions all the compatible versions. Numerous adjustments have been made to the IP address, port number, interface name, functions etc, to enable communication between NFs of different distributions. Right now, two projects—one using SCP's assistance and the other relying solely on NRF—are operating significantly in a different manner. Setting up the initial connection between the two open-source projects needs a deep understanding of their architecture and specially the source code.

The ability of NFs from one distribution to register with an NRF from another distribution and to able to add the subscription to NF status indicates a level of interoperability at the initial stage. This is promising for multi-distribution environments, suggesting that there's a foundational compatibility at least at the level of initial NF registration and status subscription. But the heartbeat mechanism's incompatibility between different open source projects suggests a deeper level of integration issue.

The final observation includes that after the successful initial registration NF Service Consumers send a PATCH request to the resource URI using same NF Instance and Instance ID which was used earlier during initial registration through HTTP2 PUT messages. It is sent to replace their nfStatus as registered in the network so that NF and its services can be discovered by other NFs available in the network. However, in response NF Service Consumers get a “400 Bad Request” error by NRF. The NF instances are failed to be processed by NRF due to errors in the encoding of the NFProfile JSON object which includes NF Instance. The NF Instance, identified by the "nfInstanceID", is not available in the list of registered NF Instances in the NRF's database. This issue is compounded by the NRF's behavior of deleting NF instances generated by a different distribution, suggesting an inability to maintain state for foreign or unrecognized NF instances.

But If the same thing is done within only one distribution then NRF returns “204 No Content” as a successful response to the PATCH request.

Thus, it has been observed that after the initial registration, further stages or chances for communication among different NFs are lacking. Specifically noting the absence of sustained communication between network functions beyond the initial registration phase. This gap underscores a significant challenge in achieving full interoperability within the 5GS with two different project. However, it also opens avenues for future research and development efforts aimed at overcoming these obstacles.

In future, research can be conducted on investigating the possibility of keeping the NF Instance ID registered in the NRF database, irrespective of the distribution origin, could facilitate ongoing communication between NFs. If it is possible to use and generate the same format JSON data for NF instances belonging to both distributions, then this approach would allow for the maintenance of the heartbeat mechanism across integrated environments, enhancing interoperability. Encouraging both the OpenAirInterface (OAI) and Open5GS communities to adopt this strategy could be pivotal.

Currently, the OAI community primarily supports deploying NFs using Docker containers, limiting testing scenarios and interoperability research. Expanding the deployment options to include non-containerized environments could offer more flexibility in testing configurations and integration scenarios. Addressing the

challenge of significant storage requirements for building individual components like NRF, UDR, and UDM outside of containerized environments would also be essential.

One key feature should also be highlighted here that, for use case 1 the same implementation is not possible now with OAI NRF. As per the recent update in OAI repository it is not possible to build OAI NRF alone without using docker. The configuration file which is used in this research named “nrfOAI.yaml” is currently not getting generated by using the latest source code by GitHub. But it is possible to use “config.yaml” which is a general file maintained for all NFs.

Conducting deep analysis and making necessary adjustments to the source code of various NFs could enable sustained connections between different distributions. Modification of source code in Open5gs is a must for error free execution because of the SCP issue mentioned in section 4.2.5. SCP related parameters should be completely disabled in source code so that no NF in open5gs tries to make a communication through SCP.

In future, collaborative efforts from different open source community can lead to the development of standardized protocols, interfaces, and testing methodologies that accommodate diverse 5G implementations.

6 Acknowledgment

I want to express my gratitude towards Prof. Trick and Prof. Lehmann for providing me an opportunity to work on such an interesting and insightful topic. The project work has taught me a lot about the 5G world, and the intricacies of communication involved it. I cordially thank Prof. Trick for his guidance, supervision, feedback, and patience on this project. Thank you so much for the time you took for the project discussion. I also thank Prof. Lehmann for his valuable suggestions. I really appreciate the time he made for the discussion related certain issues of this project.

7 Abbreviations

1

1G	First Generation Cellular Networks

2

2G	Second Generation Cellular Networks

3

3G	Third Generation Cellular Networks
3GPP	Third Generation Partnership Project

4

4G	Fourth generation of Cellular Networks

5

5G	Fifth generation of Cellular Networks
5GCN	Fifth generation Core Network
5GC	Fifth generation Core
5GS	Fifth generation System

A

ARFCN	Absolute radio-frequency channel number
API	Application Programming Interface
AKA	Authentication and Key Agreement

C

CN	Core Network
CU	Control Unit
CP	Control Plane
CRUD	Create, Read, Update, and Delete
COTS	Commercial Off-The-Shelf

D

DU	Data Unit
DP	Data Plane
DHCP	Dynamic Host Configuration Protocol
DRB	Data Radio Bearer
D-CPI	Data-Controller Plane Interface

E

ETSI	European Telecommunications Standards Institute
EMBB	Enhanced Mobile Broadband
ECG	Electrocardiogram.
EPC	Evolved Packet Core

G

GTP	GPRS Tunneling Protocol
GUTI	Globally Unique Temporary Identifier
GTP-U	GPRS Tunnelling Protocol User Plane

H

HTTP	Hyper Text Transfer Protocol
HSS	Home Subscriber Server

HCI Hyperconverged Infrastructure

I

IOT Internet of Things

ITU International Telecommunications Union

IMT-2020 International Mobile Telecommunications 2020

IETF Internet Engineering Task Force

IPAM IP Address Management

IMSI International Mobile Subscriber Identity

IDL Interface Description Language

J

JSON Javascript Object Notation

L

LTE Long Term Evolution

M

METIS Mobile and wireless communications Enablers for the Twenty-twenty Information Society

MMTC Massive Mobile Type Communication

MNO Mobile Network Operator

MRDC Multi-RAT Dual Connectivity

MME Mobile Management Entity

MCC Mobile County Code

MNC Mobile Network Code

MAC Medium Access Control

N

NSA	Non Standalone Architecture
NG-RAN	Next Generation RAN
NF	Network Function
NG-RAN	Next Generation Radio Access Network
NE	Network Element
NAS	Non-Access Stratum
NGAP	Next Generation Application protocol

O

OAI	OpenAirInterface
OSA	OpenAirInterface Software Alliance
ONAP	Open Network Automation Platform

P

P2P	Point To Point
PFCP	Packet Forwarding Control Protocol
PDU	Protocol Data Unit
PCC	Policy and Charging Control
PDCP	Packet Data Convergence Protocol

Q

QoS	Quality of Services

R

REST	REpresentational State Transfer
RAN	Radio Access Network
RLC	Radio Link Control

RRC Radio Resource Control

RIC RAN Intelligent Controller

RLS Radio Link simulation

S

SA Standalone Architecture

SBA Service Base Architecture

SBI Service Base Interface

SCTP Stream Control Transmission Protocol

SUCI Subscription Concealed Identifier

SUPI Subscription Permanent Identifier

SDAP Service Data Adaptation Protocol

U

UE User Equipment

URLLC Ultra-Reliable Low Latency Communications

URI Unified Resource Identifier

UDP User Datagram Protocol

V

VoNR Voice over New Radio

VLR Visitor Location Register.

8 References

1. Qualcomm vlog, “Everything you need to know about 5G”, https://www.qualcomm.com/5g/what-is-5g
2. Malcolm Atkinson a, Sandra Gesing b, Johan Montagnat c, Ian Taylor: “Future Generation Computer Systems Volume 75”, October 2017.
3. Wesley Chai, Corinne Bernstein: “3GPP (3rd Generation Partnership Project)”, https://www.techtarget.com/searchnetworking/definition/3rd-Generation-Partnership-Project-3GPP
4. Trick, U. (2021): An Introduction to the 5th Generation Mobile Networks. Frankfurt am Main: De Gruyter Oldenbourg.
5. 5G-ACIA: 5G for Connected Industries and Automation, Whitepaper, 2nd edition. 5GACIA, February 2019
6. 5gamericas: 5G Services and Use Cases, Nov 2017, https://www.5gamericas.org/5g-services-use-cases/
7. Cadence PCB solutions: “Massive Machine-Type Communication (mMTC) for Smart Systems”,https://resources.pcb.cadence.com/blog/2022-massive-machine-type-communication-mmtc-for-smart-systems
8. RCR Wireless News, “Service Based Architecture is cloud-native”, September 9, 2022
9. Sree Lekshmi : “5G Service Based Architecture (SBA)”, calsoftinc.com/blogs/2022/, September 21, 2022
10. ufiSpace Article: “Demystifying 5G Technology: Understanding 5G NR, NSA vs SA, and Use Case Introduction”, August 2023
11. Deanna Darah : “5G NSA vs. SA: How do the deployment modes differ?”, TechTarget.com, https://www.techtarget.com/searchnetworking/feature/5G-NSA-vs-SA-How-does-each-deployment-mode-differ
12. info@iplook.com, “IP LOOK end-to-end mobile network solution”, , https://iplook.com/info/5g-nsa-sa-i00028i1.html
13. Gaurav Gangwal, Kevin Gray: “The 5G Core Network Demystified”, August 17th, 2023
14. TELCOMA.com: “NGAP- Next-Generation Application Protocol over N2 Interface”, https://telcomaglobal.com/p/ngap-next-generation-application-protocol-n2-interface
15. emblasoft.com blog: “Exploring the 3GPP AMF – Access & Mobility Management Function”, September 2022
16. DELL Technologies: “Dell Technologies 5G Core Validated Design with Oracle and VMware Reference Architecture Guide”, https://infohub.delltechnologies.com/l/dell-technologies-5g-core-validated-design-with-oracle-and-vmware-reference-architecture-guide-1/use-cases-161/
17. Connor Craven: “What Is Unified Data Management (UDM)?”, https://www.sdxcentral.com/5g/definitions/key-elements-5g-network/what-is-unified-data-management/
18. MAVENIR: “Service Communication Proxy (SCP)”, https://www.mavenir.com/resources/service-communication-proxy-scp/
19. Oracle docs: “Service Communication Proxy (SCP) Cloud Native User's Guide” , https://docs.oracle.com/en/industries/communications/cloud-native-core/2.2.0/scp_user_guide/service-communication-proxy-architecture1.html#GUID-6C3A599D-AB9F-430D-B7E8-66AD51B69006
20. ETSI Paper: “Network Functions Virtualisation (NFV) ”, https://portal.etsi.org/nfv/nfv_white_paper_5g.pdf
21. Andrew Froehlich, West Gate Networks: “network functions virtualization (NFV)”, https://www.techtarget.com/searchnetworking/definition/network-functions-virtualization-NFV
22. https://openairinterface.org/
23. https://open5gs.org/
24. https://free5gc.org/
25. https://www.srslte.com/
26. https://www.o-ran.org/
27. https://cloudify.co/what-is-onap/

28. Alifya Hussain: “5G SDN and NFV”, https://medium.com/@alifyahussain/5g-sdn-and-nfv-e411dbe927b1, Jan 20, 2020
29. Connor Craven: “How 5G SDN Will Bolster Networks”, October 2020, https://www.sdxcentral.com/5g/definitions/key-elements-5g-network/5g-sdn
30. https://open5gs.org/open5gs/docs/guide/01-quickstart/
31. Cubro: “”What is Service Based Architecture for 5G System , 29. April 2021, https://www.cubro.com/en/blog/service-based-architecture-for-5g-system/
32. Jai Arkko: “Service-Based Architecture in 5G”, September 14, 2017, https://www.ericsson.com/en/blog/2017/9/service-based-architecture-in-5g
33. Lionel Morand: “3GPP TSG CT Chair, “OpenAPIs for the Service-Based Architecture”, May 30 2022, https://www.3gpp.org/technologies/openapis-for-the-service-based-architecture
34. Maede Zolanvari: “SDN for 5G”, October 2015, https://www.cse.wustl.edu/~jain/cse570-15/ftp/sdnfor5g.pdf
35. Horizon 2020: “Mobile and wireless communications Enablers for Twenty-twenty (2020) Information Society-II”, https://cordis.europa.eu/project/id/671680
36. Etsi.org: ” WHY DO WE NEED 5G?”, https://www.etsi.org/technologies/mobile/5