Technology
Arts Sciences
TH Köln

# Implementation and evaluation of network slicing in containerized 5G Campus Networks

Edisson Andres Zurita Hidalgo

Research Project
MSc. Communications Systems and Networks

Lecturer: Prof. Dr. Andreas Grebe

January 11, 2023

**Abstract**

Implementation and evaluation of network slicing in Containerized 5G Campus Networks. In this project a 5G Core, Radio Access Network (RAN) and User Equipment (UE) is deployed in Docker containers, in order to configure Network Slices and evaluate their performance with bandwidth throughput tests.

# Contents

# 1 Preface

This project wants to simulate a 5G core network in a containerized environment, to test mobile network scenarios as real as possible with docker containers and try to measure parameters like delay, jitter so it can be considered as an alternative for future commercial (or enterprise) implementations. All tested scenarios were performed in TH Köln on-premise servers so it can be available for future tests and implementations, and to recreate different testbeds according to the researcher requirements.

# 2 Fundamentals

In this section, all the theoretical fundamentals that were used in this project will be briefly described, so any reader could gather all fundamental concepts before diving into the tests and investigation results itself.

## 2.1 Fundamentals of 5G

### 2.1.1 Architecture of Mobile Networks

A mobile telecommunications system is operated by a network operator such as Deutsche Telekom, Telefonica or AT&T and is officially known as a public mobile network (PLMN). As shown in Figure 1, it consists of four main components, namely the core network (CN), the radio access network (RAN), the management system and the user's terminal device. The latter is colloquially referred to as the cellphone and more formally as the user equipment (UE)Cox [1]

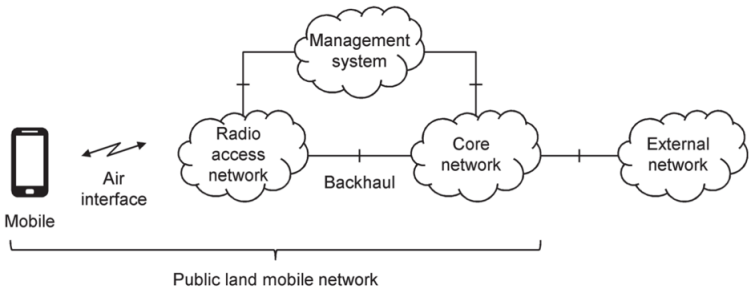


Figure 1: Architecture of a mobile telecommunication system.

The core network transports data traffic between the cellphone and one or more external networks, e.g., the public switched telephone network (PSTN) or the Internet. The core network also controls the cell phone's communication with these external networks and stores information about the network operator's subscribers.

The radio access network controls the network's radio communications with the cellphone. It communicates with the core network through an interface called backhaul and with the cellphone through the air interface, also called radio interface. At this interface, the direction from the network to the cellphone is

called the downlink (DL) or forward link, and the direction from the cellphone to the network is called the uplink (UL) or reverse link.

The network is controlled by its own management system. Its tasks include configuring the various components of the core and radio access network, monitoring their performance, reporting faults to the network operator and billing the user.

### 2.1.2 Evolution of Mobile Core Network

In this section, a brief review of the history of Mobile Networks (with focus on Core Network) will be introduced, as it is important to understand the current architecture used in modern networks.

Mobile Networks began their evolution around the decade of the 1950s, when its first generation (1G) appeared with the purpose of expanding previous broadcasting techniques in major important points that concerned the users at the time, using analog voice modulation with really bulky user equipment, a mobile telephony service (MTS) based on a manual circuit switch core (managed by a group of call operators) Cox [1]. The most important drivers for mobile networks at the moment:

- Simultaneous calls
- Spectrum usage
- Privacy and security
- Area coverage
- Reliability

All of them, as we can imagine, remain as main focus points of development for all the research groups actively working on new standards and improvements of hardware and software (already on production and upcoming). Some other concerns where added to the list, as more services and advantages were needed and the use became more spread (especially by the end of the 90s).

- Data transfer rate
- User mobility
- Quality of service
- Low latency
- End-User security

Given that the service became a massive necessity for all users around the world, and found great applications such as Broadband internet connection (with high-speed mobility), Internet of things applications (IoT) and Machine-to-Machine communication (M2M).

**2.1.2.1 Second Generation (2G)** In the second generation (2G) of Mobile Networks, the Mobile Core Network received a great improvement in *separation of planes*. It was the pioneer that presented a clear separation between radio network and switching network. To accomplish that, the base station were

controlled by a Base Station Controller (BSC) which acted as a traffic aggregator/distributor between Base Transceiver Stations (BTS) and Mobile Core Network. An image depicting 2G Architecture can be find below. There were also great improvements regarding the following points.

- ***Digital modulation combined with multi-access schemes.*** Digital modulation techniques arose as an efficient way to accelerate voice transmission and reduce spectrum usage. At the same time, multiplexing techniques were employed to increase the number of simultaneous calls allowed and enhance signaling mechanisms with dedicated channels for control messages. Time Division Multiple Access (TDMA) and Code Division Multiple Access (CDMA) techniques emerged as the most suitable options for mobile networks.
- ***Security and Handset flexibility.*** GSM introduced the suscriber identity module (SIM) card, which is a small memory card that stores user information and security parameters (e.g. user identity, key pairs and more), providing and extra layer of security for both the user and provider, and allowing the user to change devices at anytime.
- ***Improved Mobile Switching Center and new services.*** Providers started to offer data exchange services. After the introduction of multiple access techniques, a few access channels were destined for data exchange (even though it was at low rate and without simultaneous use of voice, it was a big progress). Short Message Service (SMS) was offered to, allowing users to send up to 160 characters per message between each other.

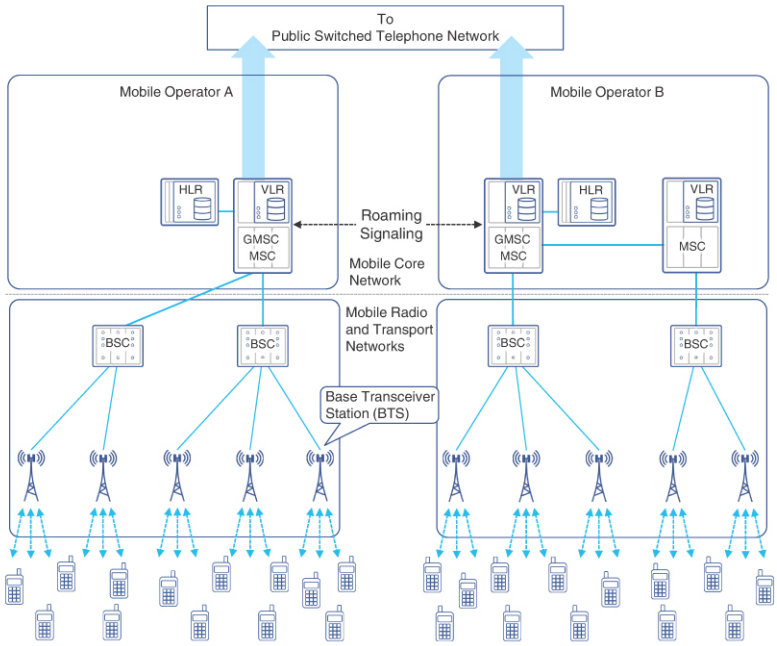


Figure 2: 2G Mobile Network Architecture.

**2.1.2.2 A small step before third generation (2.5G and 2.75G)** Some small technical improvements were introduced in between 2G and 3G, due to the fact that when the user placed a call, the data transmission service became unavailable. Sudhakar Shetty [3]Instead of allocating all time slots for voice channels, as done in GSM, a newly introduced concept called *GPRS (General Packet Radio Service)* carved out a few channels to data transmission, allowing users to keep data connections active and working, even when the user was in a voice call, providing the "always-on" data experience to the user. Other architecture changes were made in GPRS, being the most important the following:

- The introduction of direct connection to the packet switched data network *(packet data network, or PDN)* instead of bypassing all this traffic through PSTN. This includes connection to public internet, as well as private networks or intranets. This was enabled by the use of Serving GPRS Support Node or SGSN (which face the suscriber side of the network) and Gateway GPRS Support Node or GGSN, which faced the PDN. Both blocks encompass the so called GPRS Subsystem.
- Inside the Base Station Controllers, a capability called Packet Control Unit (PDU) was included, which was in charge of redirecting voice traffic to MSC and data traffic to GGRS. For data traffic, the GGSN then could forward traffic directly to internet, or to a GPRS roaming exchange (GRX) in the case the provider was serving traffic to a visiting user, thus, to provide roaming service in 2G data connections.

An image of 2.5G Mobile Network is shown below:

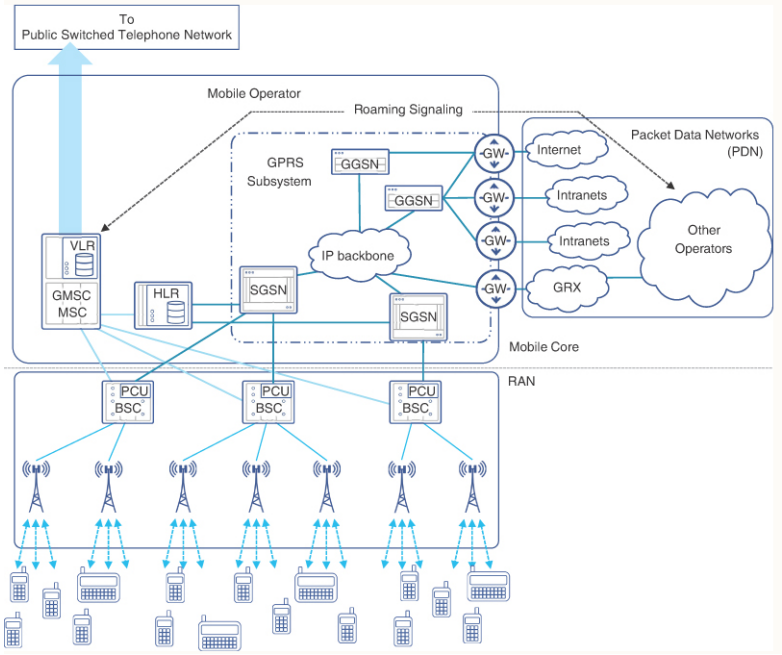


Figure 3: 2.5G Mobile Network Architecture.

Since the maximum data rate of GPRS was still low (57.6 Kbps if the end user occupies all the time slots), new modulation techniques and link adaptation

methods were introduced in the early 2000s (with no major network-architecture changes). This changes were presented to the market as *Enhanced Data Rates for GSM Evolution, or* ***EDGE***. The improvements targeted just the Radio Access Network, and offered a maximum data transmission speed of 384 kbps with all eight TDMA slots concatenated.

**2.1.2.3 Third Generation (3G)** In the early 2000s, the internet service began to spread rapidly among companies and common users across the world, with a everyday increasing offer of DSL lines for private purposes. This also boosted the diversity of *next-generation services* offered through both fixed and mobile networks, such as voice and video calls, car navigation and audio and video streaming. Therefore, the data transmission speed offered by EDGE was not enough for customers, which lead into an important milestone in Mobile Networks, and that is the creation of an international consortium called *3rd Generation Partnership Project (3GPP)*, composed by the most relevant vendors and operators of Mobile System in the world at that time. The first specification of this group was *Release 99* that defined the *Universal Mobile Telecommunications System (UMTS)*. In the figure below the 3G Architecture is shown.

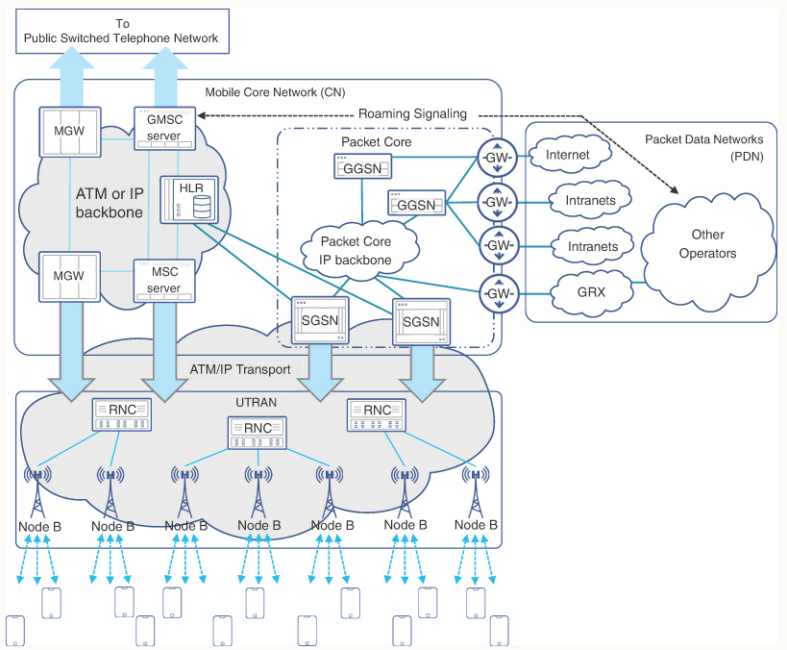


Figure 4: 3G Architecture.

The UMTS comprehends two key components: UMTS Terrestrial Radio Access Network (UTRAN) and UMTS Core Network (CN). In this initial realease the CN components were very similar to the GPRS network but evolved deeply in the next releases, however, the UTRAN part had some huge improvements for the radio network, including new and advanced channel coding techniques and the adoption of Wideband Code Division Multiple Access (WCDMA) to boost up data transmission speeds. It included as well, the concept of *NodeB* as a demarcation point between RAN and mobile backhaul network.

Regarding the Core Network, most of the important core blocks remained unchanged: MSC, GMSC, HLR GGSN and so on. Nevertheless, due to scalability issues for continuously growing service demand pushed a major change in the MSC, which was divided into two functions: MSC Server (MSC-S) and media gateway (MGW).

Given that 3G was a standardized product, it evolved over time with different *"Releases"* that were published. Most important features of each release are:

- **Release 99.** Which was already explained and separated core into two main key components.
- **Release 4.** It began the transition to IP core network, to pave the path from circuit switching to packet switching
- **Release 5.** Improved downlink speed with new radio specifications, reaching a theoretical 14,4 Mbps maximum speed. On the mobile core a *IP Multimedia Subsystem (IMS)* to move voice connections to a packet switched manner. From Release 5 and subsequent releases are called High Speed Packet Access (HSPA).
- **Release 6.** Enhanced uplink speed up to 5,8 Mbps under ideal radio conditions.
- **Release 7.** Also referred as HSPA+, theoretical speeds of 28 Mbps for downlink and 11 Mbps were achieved, with more sophisticated modulation techniques.
- **Release 8 and beyond.** It the new *Evolved Packet Core (EPC)* for major improvements in mobile core, and *Enhanced Universal Terrestrial Radio Access Network (E-UTRAN)* for the radio side. These enhancements are better known as *Long-Term Evolution (LTE)*, leading the path into 4G Networks.

In Figure 5, there is the progression of uplink and downlink speeds across 3GPP releases.

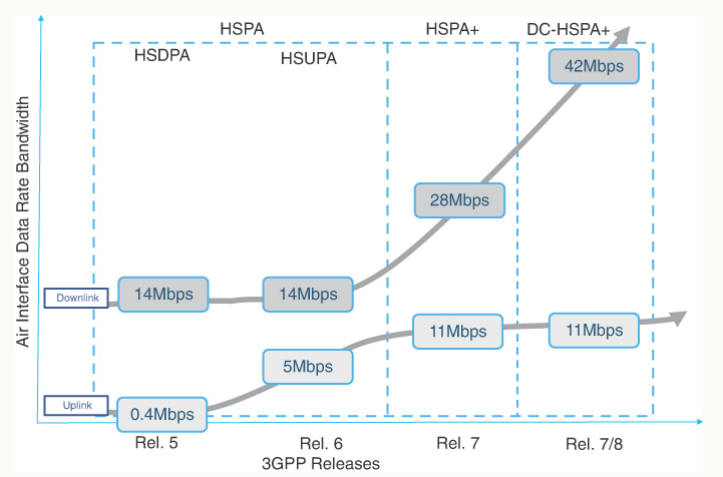


Figure 5: 3GPP Releases and Corresponding Theoretical Speeds.

**2.1.2.4 Fourth Generation (4G)** Release 8 of the 3GPP specifications marked an important transition in Mobile Networks. It paved the way for more modular core and radio networks, with open interfaces to be independent from each other, but trying to keep all CN and RAN blocks connected together smoothly, working constantly on improving data rates for the ever-growing application market. It also tried to provide lower latency with better Quality of Service (QoS) options Hassan [2]. At the same time, it warrantied interoperability with existing 3G systems, so transition expenses could be soften and slowly assumed by operators. In 4G, changes targeting the radio portion of the network were referred to as Long Term Evolution (LTE), while the new mobile core architecture was discussed as *System Architecture Evolution (SAE).*

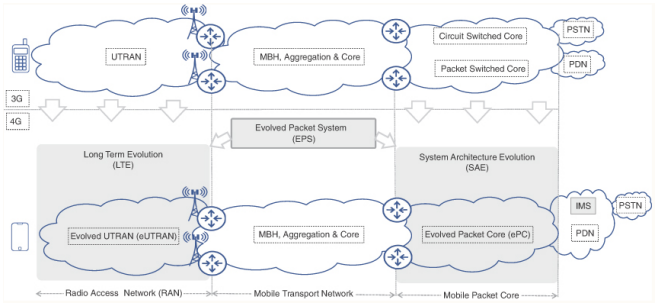


Figure 6: 4G Mobile Network Architecture compared with 3G

**System Architecture Evolution (SAE).** It defined the Evolved Packet Core (EPC) as the foundation of the Mobile Core Network. From that point ahead, all mobile core design try to keep a very modular design, with open interfaces to make each building block vendor-agnostic, with open interfaces to talk to other blocks and guarantee backwards compatibility. It settled Internet Protocol (IP) within the packet core, including Voice over IP (VoIP) for calls and emerging protocols to packet switching within the core (MPLS being the most resourceful).

***Evolved Packet Core (EPC).*** As a major development point of Mobile Core, 4G started the decoupling of Control Plane and User Plane and grouping those functions to individual devices, allowing each blocked to be scaled and improved independently from each other. Therefore, new nodes emerged from the EPC evolution, which are described below.

- ***Mobility Management Function (MME).*** Is the centralized control node of EPC. It inherited the user management role of the MSC-Server as well as the control plane functions of SGSN and some RNC's roles of allocating radio resource and facilitating handoff scenarios; but, it doesn't handle any user traffic.
- ***Serving Gateway (SGW).*** The SGW is an aggregation point terminating the GTP tunnel from the user's device, and starting a new GTP tunnel toward the PDN Gateway.
- ***Public Data Natework Gatewat (PGW).*** It allocates IP address to user's devices with appropriate and allowed connectivity to the PDN network based on the APN, user profile, and type of subscribed services.

PGW also registers and keeps track of data and voice consumption by each user for billing purposes.

- ***Policy Charging and Rule Function (PCRF).*** It stores the policies to allow or deny services to the user, although the enforcing of these rules is still performed by PGW.
- ***Home Suscriber Server (HSS).*** Is consulted for user authentication, authorization, session establishment. It uses a protocol called Diameter, to perform AAA functions.

In the figure 7, the functions from 3G blocks and 4G corresponding successors are depicted.

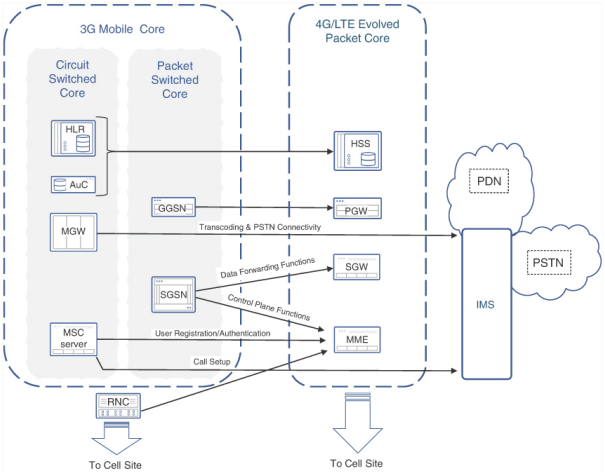


Figure 7: 4G Mobile Network Architecture compared with 3G

In legacy systems, the core network contains two domains that transport different types of traffic using different network technologies. The circuit-switched (CS) domain transports fixed-rate data traffic, such as voice, so that users can make calls to other devices on the PSTN or to the circuit-switched domains of other network operators. This is done using a technique known as circuit switching, where a separate two-way connection is established for each individual telephone call. The CS domain carries voice traffic at a constant data rate with minimal delay, but is unsuitable for services where the data rate may vary.

The packet-switched (PS) domain transports variable-rate traffic, such as Web pages and e-mail, between the user and external data networks such as the Internet. In this process, a data stream is divided into packets, each of which is tagged with the address of the desired destination device. Within the network, routers read the destination addresses of incoming data packets and forward them to these destinations according to the instructions in the internal routing tables. The resources of the network are shared by all users, so this technique is more efficient than circuit switching. However, delays can occur if too many devices attempt to transmit at the same time.

Recently, the widespread use of smartphones has led to data traffic outweighing voice traffic on cellular networks. In response, developers have abandoned circuit switching and introduced mobile communications systems that use only packet

switching. This simplifies the design and allows the system to be optimized for carrying data traffic, but it also means that voice calls have to be handled in a different way.

### 2.1.3 5G Mobile Core Network

There are two key components that redefined the EPC, to become 5GC (5G Core). The first is the well-established separation of control and user plane functions from each other, also known as *CUPS (Control and USer Plane separation)*; and a *Service Based Architecture (SBA)*. In addition, virtualization capabilities began to be offered for 5GC, enabling a new concept called virtual Evolved Packet Core (EPC), leading to a cloud-native packet Core.

**2.1.3.1 Control and User Plane Separation (CUPS)** The proposed solution recognizes that the bulk of the traffic in the mobile networks is user data, which is predominantly sent between the mobile devices and endpoints reachable through the Public Data Network (PDN) interface of the packet core. These endpoints could include content delivery servers, gaming servers, peer-to-peer applications, and so on. Hence, if this PDN connectivity is provided closer to the cell sites, the mobile transport network wouldn't need to carry the burden of transporting this massive amount of data between the mobile user and the centralized packet core locations. CUPS, therefore, takes the plane separation idea one step further and advocates that instead of co-locating the devices implementing user-plane and control-plane functions, these should be deployed independently—with the user-plane devices deployed much closer to the cell site.

While 4G Core has evolved to 5G Core, it gained some new capabilities and some functions are divided into more functions but general architecture is similar. One of the most significant difference between 4G and 5G core network is the separation of control and user plane functions from each other, also known as *CUPS (Control and USer Plane separation)*.

**Towards a Cloud-Native 5G Core.** The application of virtualization techniques and principles of NFV had a major impact on the transformation from EPC to vEPC and subsequently to 5GC. It is therefore meaningful to understand these technologies that facilitate the transition toward a fully virtualized, cloud-native 5G core network.

***Virtualization, Containerization, and Docker.*** In the initial days of server virtualization—to achieve isolation, application security, and resource sharing—hardware-assisted virtualization through a hypervisor was the commonly used implementation. Hypervisor functionality running in a host operating system allows for instantiation of a virtual machine (VM), where a guest operating system (OS) and virtualized applications can independently run. Hence, initial implementations of NFV followed the same model, and the VNFs included an operating system ready to be instantiated as a VM. vEPC followed the same path.

This can be considered just the tip of the iceberg, since there are much more interesting concepts involved in 5G Core architecture and enhancements, just to name a few:

- *Kubernetes:* as an orchestration platform for better management and performance monitoring and tuning, with a enormous toolset that enables automation in many key processes (provisioning, maintenance, configuration management.
- *Microservices architecture:* Since 5GC is broken up in scalable and independant building blocks, each of them can be redesign, tested and version-release controlled, this is a perfect suit for microservices-based implementation; offering a much higher resiliency of the whole core.
- *Multi-Access Edge Compute (MEC).* Since CUPS separates 5GC components from each other, a great idea could be to terminate user traffic as close to the user as possible. MEC makes it possible, providing data and traffic resources in the "edge" of the network facing the user, terminating time-sensitive traffic everytime it is possible.

**2.1.3.2 5G User Plane Function (UPF).** The User Plane Network Function (UPF) is the cornerstone of CUPS and is responsible for the routing and forwarding of mobile user's traffic to an external network. To achieve low latency, UPF may be placed in the edge or far-edge data centers, in addition to the main DC, as previously discussed in the placement considerations for the 5GC user plane. Additionally, to provide ultra-reliability, either redundant UPF or redundant traffic paths to the same UPF may be used.

Just like 4G, the 5G packet core also makes use of GTP-U protocol over IP to encapsulate data traffic between the RAN and gateway to the external network, simply referred to as Data Network (DN) in 5G terminology. This GTP-U tunnel is established between the gNB and UPF over the N3 reference interface (thus sometimes referred to as N3 GTP-U). The UPF terminates this GTP-U tunnel and forwards the user traffic to the DN. The UPF (like other cloud-native implementations) is meant to be scalable to meet traffic demands, and multiple UPFs can be spun up when needed.

The mobile device and the UPF use a construct called a protocol data unit session (PDU session) for traffic forwarding. This is similar in concept to the EPS bearer that was defined in 4G networks, but it has some subtle differences—the most important being the granularity of the QoS mechanism. Unlike the EPS bearer, where a single QoS treatment is applied to all traffic, a PDU session can apply different QoS behaviors to individual traffic flows.

**2.1.3.3 5G Control Plane Network Functions** Even though this technique provided great isolation and security, there was a cost to it in the form of slight wastage of hardware resources and longer instantiation time. For deployments where the balance is more toward agility and resource optimization than a high level of isolation, the alternate approach of OS-level virtualization became attractive. This type of virtualization, also referred to as containerization, leverages the capabilities built into the OS to provide some degree of separation, resource sharing, and independence. This segregated environment is referred to as a container.

Compared to VMs, containers have a smaller footprint and make better utilization of the host's resources while providing a relatively less amount of security

and isolation. For example, containers do not require a guest OS, thus making them light on system resource requirements. However, because they share the host's OS with other containers, they aren't fully isolated from each other. Most importantly, because containers use built-in OS capabilities, they are free of additional software overhead and resource emulation that would have been required for VMs. This makes containers the preferred choice for dynamic and agile environments.

However, additional capabilities are needed for packaging, porting, maintaining, and managing the containerized applications. Docker is a definite frontrunner among the tools that offer these capabilities and has been the most popular choice. Today, the words Docker and container are used interchangeably, almost completely overlooking the fact that a container is a virtualized application while Docker is an application to package, build, manage, instantiate, version-control, and ship a containerized application.

Because some functions are divided, 5G core network has more functions as it mentioned above. 5G Core Network Functions are like following:

- **Access and Mobility Management function (AMF)** supports: Termination of NAS signaling, NAS ciphering and integrity protection, registration management, connection management, mobility management, access authentication and authorization, security context management.
- **Session Management function (SMF)** supports: session management (session establishment, modification, release), UE IP address allocation and management, DHCP functions, termination of NAS signaling related to session management, DL data notification, traffic steering configuration for UPF for proper traffic routing.
- **User plane function (UPF)** supports: packet routing and forwarding, packet inspection, QoS handling, acts as external PDU session point of interconnect to Data Network (DN), and is an anchor point for intra- and inter-RAT mobility.
- **Policy Control Function (PCF)** supports unified policy framework, providing policy rules to CP functions, access subscription information for policy decisions in UDR.
- **Authentication Server Function (AUSF)** acts as an authentication server.
- **Unified Data Management (UDM)** supports: generation of Authentication and Key Agreement (AKA) credentials, user identification handling, access authorization, subscription management.
- **Application Function (AF)** supports application influence on traffic routing, accessing NEF, interaction with policy framework for policy control.
- **Network Exposure function (NEF)** supports: exposure of capabilities and events, secure provision of information from external application to 3GPP network, translation of internal/external information.

- **NF Repository function (NRF)** supports: service discovery function, maintains NF profile and available NF instances.
- **Network Slice Selection Function (NSSF)** supports: selecting of the Network Slice instances to serve the UE, determining the allowed NSSAI, determining the AMF set to be used to serve the UE.

At this point, NFV becomes important because of cloud-native networking concept because it will be possible for core network to request new network functions from VNF catalog into E2E service chains. NFV will provide flexibility and time efficiency to system.

### 2.1.4 Network Functions Virtualization

Network Functions Virtualization (NFV) is a technology that allows network operators to virtualize network functions and services, such as routers, firewalls, and load balancers, and run them on commercial off-the-shelf (COTS) servers or other commodity hardware. This allows operators to deploy and manage network functions and services more efficiently, and reduces the need for specialized hardware and proprietary software.

One of the main benefits of NFV is that it allows network operators to more easily and quickly deploy new network functions and services. By using NFV, operators can create and deploy new network functions and services as virtual appliances, without having to purchase and install specialized hardware. This allows operators to respond more quickly to changing customer demands and market conditions, and to introduce new services and capabilities more rapidly. Zhang [5]

Another benefit of NFV is that it allows network operators to more easily scale their networks to meet changing demand. By using NFV, operators can create virtual networks that can be quickly and easily expanded or contracted, depending on the needs of their customers. This allows operators to more efficiently utilize their network resources, and to better manage their costs.

NFV also offers several other benefits. It allows network operators to more easily migrate their networks to new technologies and architectures, such as software-defined networking (SDN) and 5G. It also enables operators to more easily integrate their networks with other systems, such as cloud computing platforms and internet of things (IoT) networks. Finally, NFV can help operators to improve the security and reliability of their networks, by allowing them to isolate and protect different network functions and services.

**2.1.4.1 Network Slicing** Network slicing is a technology that allows service providers to create multiple virtual networks on top of a shared physical infrastructure. These virtual networks, or slices, can be customized to meet the specific requirements of different types of users or applications. This allows service providers to offer a wide range of services with different performance, security, and reliability characteristics, while still utilizing a common underlying infrastructure.

One of the main drivers for network slicing is the emergence of 5G networks. 5G networks are expected to support a wide range of new applications and ser-

vices, including high-speed broadband, low-latency communications, and massive machine-type communications. These applications have very different requirements in terms of performance, security, and reliability, and cannot be supported by a single network architecture.

Network slicing allows service providers to create virtual networks that are tailored to the specific requirements of each type of application or service. For example, a service provider might create a slice for broadband services that is optimized for high-speed data transfer, and another slice for low-latency applications that is optimized for real-time communications. Some of the typical slices that might be used in a 5G network include:

- ***Enhanced Mobile Broadband (eMBB):*** This type of slice is optimized for high-speed data transfer, and is designed to support applications such as streaming video, online gaming, and virtual reality. eMBB slices are typically characterized by high throughput and low latency, and are able to support a large number of users simultaneously.
- ***Ultra-Reliable Low-Latency Communications (URLLC):*** This type of slice is designed to support applications that require extremely low latency and high reliability, such as remote surgery, self-driving cars, and industrial automation. URLLC slices are typically characterized by very low jitter and packet loss, and are able to support a small number of users with very high-quality connections.
- ***Massive Machine-Type Communications (mMTC):*** This type of slice is designed to support a large number of devices that communicate infrequently and in small data bursts, such as sensors, meters, and wearables. mMTC slices are typically characterized by high connectivity density and low energy consumption, and are able to support a very large number of devices with minimal network overhead.

In the figure below there is a graphical overview of each slice characteristics and trade-offs:

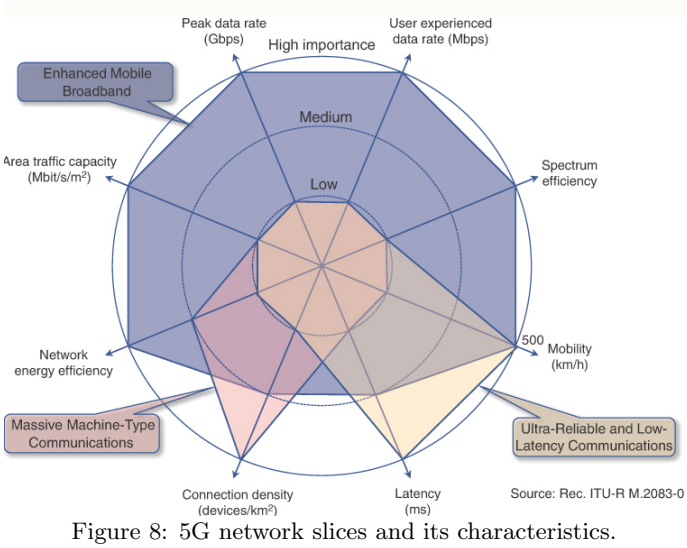


Figure 8: 5G network slices and its characteristics.

These are just a few examples of the types of slices that might be used in a 5G network. Other possible slices include ones optimized for specific industries, such as healthcare or transportation, or ones that support specific types of services, such as virtual private networks or internet of things applications. By using network slicing, service providers can create customized virtual networks that are tailored to the specific requirements of different types of users and applications.

## 2.2 Containerization Fundamentals

Since one of the goals of this project is to simulate a 5G Core with Docker containers, a brief introduction of Containers fundamentals is needed.

### 2.2.1 Introduction: What is a container?

Containers have seen widespread adoption across the Tech industry. Almost every company has started adopting containers for deploying microservices. Container orchestration technology such as Kubernetes(K8) has simplified the management of containers. Companies using Kubernetes have observed improvements in scalability, reliability and operability of their systems. But what is a container, and why is it so widely used?

Containers provide developers (and almost every tech group in every company) a fast, easy and portable method to test, deploy and redeploy applications and new features in a continuous manner. By using the concepts of Namespaces and Cgroups, containers use a "subset" of the hosts operating system to provide isolation from them and offering a lightweight, scalable and flexible alternative to Virtual Machines (VMs).

### 2.2.2 Docker containers

Docker is a powerful tool for creating and managing containers, which are isolated environments that allow you to package and run applications in a predictable and isolated manner. Containers are an increasingly popular way to develop and deploy applications because they make it easy to create consistent, repeatable environments that are portable and can be run on any machine with Docker installed.

To create a Docker container, you first need to create a Docker image. This is done using a Dockerfile, which is a text file that contains all of the instructions for building the image. The Dockerfile specifies the base image to use for the container, as well as any additional dependencies, libraries, or other software that the application needs to run.

Once the Dockerfile is complete, you can use the docker build command to build the image. This will create a new image that you can use to run a container. To run a container, you can use the docker run command, followed by the name of the image you want to run. This will create a new container and start it running.

You can use the docker ps command to see a list of all running containers, and the docker stop command to stop a running container. You can also use the

docker logs command to view the output of a running container, or the docker exec command to run a command in a running container.

Docker is a powerful tool that can greatly simplify the process of building and deploying applications. By using containers, you can ensure that your applications will run consistently and reliably on any machine with Docker installed.

### 2.2.3 Docker compose

Docker compose is a basic container orchestrator, widely used to deploy and manage a Docker applications, especially in testing environments. It is the simplest and first step to bring an application with multiple containers running and ready for testing and evaluating, because it offers the developer/tester to run the application locally or remotely, and specify all involved parameters in a single file.

To use Docker Compose, first a `docker-compose.yaml` file is needed that defines the services that make up your application. This file uses a YAML syntax to specify the details of each service, such as the image to use, any dependencies, ports, and other configuration options.

Once the `docker-compose.yaml` file is complete, the `docker-compose up` command should be used to start all of the services defined in the file. This will create and start containers for each of the services, and link them together so that they can communicate with each other. On the other hand, `the docker-compose down` command to stop all of the services and remove the containers. Some other useful commands are `docker-compose logs` command to view the output of all of the services, or the `docker-compose exec` command to run a command in a specific service.

A small example of a `docker-compose.yaml` file is shown below, it defines a web application, a database, and a cache service:

```
version: '3'

services:
  web:
    image: my-web-app:latest
    depends_on:
      - db
      - cache
    ports:
      - "8080:80"

  db:
    image: postgres:latest
    environment:
      POSTGRES_USER: myuser
      POSTGRES_PASSWORD: mypassword
      POSTGRES_DB: mydb

  cache:
    image: redis:latest
```

# 3 Project Setup

## 3.1 Software tools used in the project

### 3.1.1 Open5GS

Open5GS is an Open-Source implementation written in C, to emulate Mobile Core Network, fulfilling 3GPP group specifications for fourth and fifth generations. Therefore, it has a Service Based Architecture (SBA) and provides full flexibility for developers and researchers, to test, personalize and deploy Mobile Cores based on individual needs. In this project, for the sake of researching we will use 5G Standalone Setup, meaning there is no 4G components in Core Network, neither in RAN. A descriptive picture for Open5Gs architecture is shown below on Figure 9:

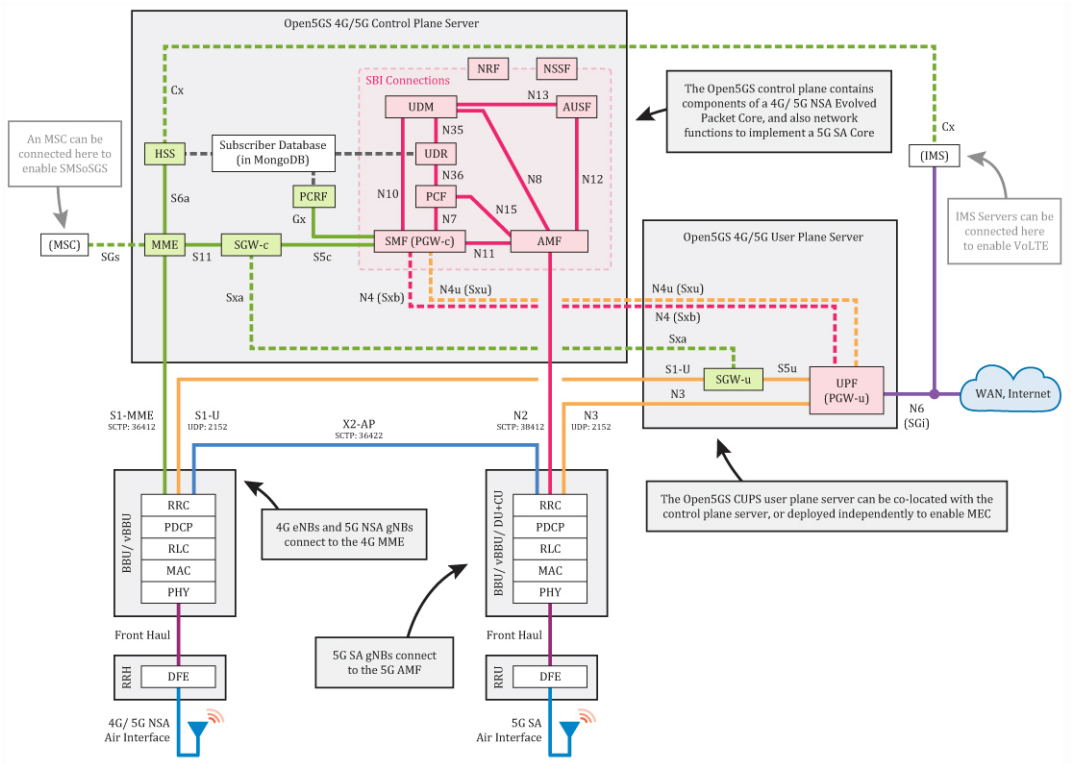


Figure 9: Open5Gs Architecture

The Open5GS 5G SA Core contains the following functions:

- NRF - NF Repository Function
- SCP - Service Communication Proxy
- AMF - Access and Mobility Management Function
- SMF - Session Management Function
- UPF - User Plane Function

- AUSF - Authentication Server Function
- UDM - Unified Data Management
- UDR - Unified Data Repository
- PCF - Policy and Charging Function
- NSSF - Network Slice Selection Function
- BSF - Binding Support Function

The 5G SA core works in a different way to the 4G core - it uses a Service Based Architecture (SBI). Control plane functions are configured to register with the NRF, and the NRF then helps them discover the other core functions. Running through the other functions: The AMF handles connection and mobility management; a subset of what the 4G MME is tasked with. gNBs (5G basestations) connect to the AMF. The UDM, AUSF and UDR carry out similar operations as the 4G HSS, generating SIM authentication vectors and holding the subscriber profile. Session management is all handled by the SMF (previously the responsibility of the 4G MME/ SGWC/ PGWC). The NSSF provides a way to select the network slice, and PCF is used for charging and enforcing subscriber policies. Finally there is the SCP that enable indirect communication.

The 5G SA core user plane is much simpler, as it only contains a single function. The UPF carries user data packets between the gNB and the external WAN. It connects back to the SMF too.

With the exception of the SMF and UPF, all config files for the 5G SA core functions only contain the function's IP bind addresses/ local Interface names and the IP address/ DNS name of the NRF.

### 3.1.2 UERANSIM

UERANSIM is the open source state-of-the-art 5G UE and RAN (gNodeB) simulator. UE and RAN can be considered as a 5G mobile phone and a base station in basic terms. The project can be used for testing 5G Core Network and studying 5G System.

There are three main interface in UE and RAN perspective, 1) Control Interface (between RAN and AMF), 2) User Interface (between RAN and UPF), 3) Radio Interface (between UE and RAN). UERANSIM supports to run with Open5GS and some other Open Source Mobile Core implementations (Free5GC for example). We can connect UERANSIM to one of these 5G Core network and test the functionality.

### 3.1.3 iPerf3

IPerf3 is built on a client-server model and measures maximum User Datagram Protocol, TCP and Stream Control Transmission Protocol throughput between client and server stations. It can also be used to measure LAN and wireless LAN throughput.

The tool is simple to use: A single executable runs on both the client and the server. Command-line parameters indicate which system will take on the role

of the server – the target –and which will be the client. It makes no difference which is which.

### 3.1.4 Wireshark

Wireshark is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education. Originally named Ethereal, the project was renamed Wireshark in May 2006 due to trademark issues

Wireshark is a data capturing program that "understands" the structure (encapsulation) of different networking protocols. It can parse and display the fields, along with their meanings as specified by different networking protocols. Wireshark uses pcap to capture packets, so it can only capture packets on the types of networks that pcap supports.

- Data can be captured "from the wire" from a live network connection or read from a file of already-captured packets.
- Live data can be read from different types of networks, including Ethernet, IEEE 802.11, PPP, and loopback.
- Captured network data can be browsed via a GUI, or via the terminal (command line) version of the utility, TShark.
- Captured files can be programmatically edited or converted via command-line switches to the "editcap" program.
- Data display can be refined using a display filter.
- Plug-ins can be created for dissecting new protocols.
- VoIP calls in the captured traffic can be detected. If encoded in a compatible encoding, the media flow can even be played.
- Raw USB traffic can be captured.
- Various settings, timers, and filters can be set to provide the facility of filtering the output of the captured traffic.

Wireshark's native network trace file format is the libpcap format supported by libpcap and WinPcap, so it can exchange captured network traces with other applications that use the same format, including tcpdump and CA NetMaster. It can also read captures from other network analyzers, such as snoop, Network General's Sniffer, and Microsoft Network Monitor.

Wireshark can color packets based on rules that match particular fields in packets, to help the user identify the types of traffic at a glance. A default set of rules is provided; users can change existing rules for coloring packets, add new rules, or remove rules.

## 3.2 Project Testbed

For this project, a VM with the following characteristics was implemented in the private cloud of TH Köln, in the DN Lab infrastructure:

- ***Operating System:*** Ubuntu 22.04
- ***Core:*** 8 vCPU
- ***Memory*** 32 GB
- ***Disk:*** 200 GB
- ***GPU:*** 16 GB

Inside this VM, a docker implementation of Open5GS was launched, following one of the tutorials from the official documentation of Open5GS, that can be found in the following link:

*https://open5gs.org/open5gs/docs/tutorial/03-VoLTE-dockerized/*

In this tutorial, a dockerized VoLTE network is configured and deployed using docker containers for each 5G Core building block, and docker-compose as orchestration tool. This GitHub project has a broad scope that comprehends a Kamailio SIP Server as Callserver and more Voice-over-IP functions, that are not included in this project since they are out of this project research span.

### 3.2.1 Containerized 5G Campus Network

In figure 10, a diagram showing the container interconnection is shown to depict how the docker network will be set up.

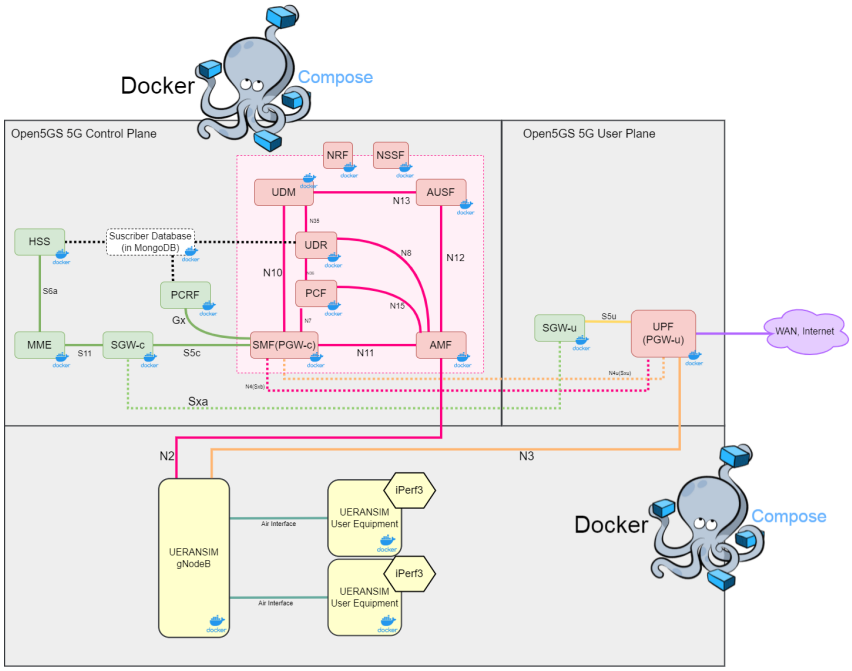


Figure 10: Dockerized 5G Core Network

For this purpose, first we need to build the corresponding docker images for each component of the network.

**3.2.1.1 5G Core in Docker Compose file.** To build base docker image for the whole core network, we need to switch to base directory and then build the docker image. The same procedure is followed afterwards to build UERANSIM images to be used as gNodeB and User Equipment (UE).

```
cd docker_open5gs/base
docker build --no-cache --force-rm -t docker_open5gs .
```

To confirm that the images are correctly build, we could check with the corresponding docker command:

```
andres@5g-simulation:~$ docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED         SIZE
docker_mysql        latest    f61c62007cd7   7 weeks ago     742MB
docker_dns          latest    5100b10cee22   7 weeks ago     251MB
docker_mongo        latest    f1a96ea930a4   7 weeks ago     281MB
docker_ueransim     latest    a20cea925dad   7 weeks ago     168MB
docker_open5gs      latest    7c479d8de8e6   7 weeks ago     567MB
```

Before continuing to build the `docker compose` services, we need to set up environment variables so they can be passed to the `docker-compose.yaml` file. An environment variable is a value that can be passed to the operating system or an application at runtime. These variables are used to configure various aspects of the operating system or an application, such as the location of files or directories, the behavior of an application, or the system path. Environment variables can be set globally for the entire operating system, or they can be set for a specific user or application. They are often used to store information that may change between different environments, such as development, staging, and production environments. The complete environment variable file is show below:

```
andres@5g-simulation:~/docker_open5gs$ cat .env
# Set proper timezone to sync times between docker host and
    containers
#TZ=Europe/Berlin

MCC=001
MNC=01

TEST_NETWORK=172.22.0.0/24
DOCKER_HOST_IP=192.168.1.223

# MONGODB
MONGO_IP=172.22.0.2

# HSS - open5gs
HSS_IP=172.22.0.3

# PCRF
PCRF_IP=172.22.0.4

# SGW
SGWC_IP=172.22.0.5
SGWU_IP=172.22.0.6
SGWU_ADVERTISE_IP=172.22.0.6

# SMF
SMF_IP=172.22.0.7
```

```
# UPF
UPF_IP=172.22.0.8
UPF_ADVERTISE_IP=172.22.0.8

# MME
MME_IP=172.22.0.9

# AMF
AMF_IP=172.22.0.10

# AUSF
AUSF_IP=172.22.0.11

# NRF
NRF_IP=172.22.0.12

# UDM
UDM_IP=172.22.0.13

# UDR
UDR_IP=172.22.0.14

# MYSQL
MYSQL_IP=172.22.0.17

# ICSCF
ICSCF_IP=172.22.0.19

# SCSCF
SCSCF_IP=172.22.0.20

# PCSCF
PCSCF_IP=172.22.0.21

# UERANSIM
NR_GNB_IP=172.22.0.23
NR_UE_IP=172.22.0.24
NR_UE2_IP=172.22.0.35

## USER EQUIPMENT 1

UE1_IMEI=356938035643803
UE1_IMEISV=4370816125816151
UE1_IMSI=001011234567895
UE1_KI=8baf473f2f8fd09487cccbd7097c6862
UE1_OP=11111111111111111111111111111111
UE1_AMF=8000

## USER EQUIPMENT 2

UE2_IMEI=356938035643804
UE2_IMEISV=4370816125816152
UE2_IMSI=001011234567891
UE2_KI=8baf473f2f8fd09487cccbd7097c6865
UE2_OP=11111111111111111111111111111111
UE2_AMF=8000

# OPEN5GS WEBUI
WEBUI_IP=172.22.0.26

# PCF
PCF_IP=172.22.0.27
```

```
# NSSF
NSSF_IP=172.22.0.28

# BSF
BSF_IP=172.22.0.29

# ENTITLEMENT SERVER
ENTITLEMENT_SERVER_IP=172.22.0.30
```

Next, the `docker compose` images must be build so it can be ready when it will start to run. For that we need to set up our environment variables that will help to pass important parameters to the docker network (such as: containers IP addresses, MCC, NCC, IMSI parameters for UE and more).

For that we need to run the following commands:

```
set -a
source .env
docker-compose build --no-cache
```

Then we can move on to build RAN and UE docker images.

**3.2.1.2 Containerized RAN and UEs.** To build the docker images for gNodeB and UEs (known as `docker_ueransim`) we do the following:

```
cd ../ueransim
docker build --no-cache --force-rm -t docker_ueransim .
```

We can check if the image is correctly built with the following command:

```
andres@5g-simulation:~/docker_open5gs$ docker image ls
REPOSITORY           TAG       IMAGE ID       CREATED         SIZE
...
...
docker_ueransim      latest    a20cea925dad   2 months ago    168MB
...
```

The Mobile Core Network and the RAN and USER portion are ready to be deployed, but we will bring it alive in a later step.

### 3.2.2 Network slicing configuration

For an appropriate configuration of network slicing, we need to modify the proper configuration files of the building blocks that handle slicing in 5G core and RAN portion of the network. Below the details of each file are shown and remarked.

**3.2.2.1 AMF.** In the AMF configuration we must include the *Service Differentiator (SD)* and *Slice Service Type (SST)* field values, so the slices can be defined and distinguished among each other and the "default" slice. The file below is *∼/amf/amf.yaml*:

```
    plmn_support:
      - plmn_id:
          mcc: MCC
          mnc: MNC
```

```
        s_nssai:
          - sst: 1
            sd: 00000a
      - plmn_id:
          mcc: MCC
          mnc: MNC
        s_nssai:
          - sst: 2
            sd: 00000b
```

**3.2.2.2 NSSF.** In the NSSF config we must also include the SD value so the NSSF block can provide this details to NRF, so the control plane is aware that two slices are provisioned across the whole 5G core. The file below is *~/nssf/nssf.yaml*:

```
    nssf:
    sbi:
      - addr: NSSF_IP
        port: 7777
    nsi:
      - addr: NRF_IP
        port: 7777
        s_nssai:
          sst: 1
          sd: 00000a
      - addr: NRF_IP
        port: 7777
        s_nssai:
          sst: 2
          sd: 00000b
```

**3.2.2.3 UERANSIM - gNodeB.** Of course we must tell the gNodeB in the RAN section to support slices, so the UEs can properly register and access all the functions needed. The configuration is shown below and can be found in *~/ueransim/open5gs-gnb.yaml*:

```
        # List of supported S-NSSAIs by this gNB
    slices:
    - sst: 1
        sd: 000010
    - sst: 1
        sd: 000011
```

**3.2.2.4 UERANSIM - UE.** Both UEs must specify, to which network slices they want to be registered and served, so this details are configured below. The configuration file is under *~/ueransim/open5gs-ue.yaml* and *~/ueransim/open5gs-ue2.yaml*:

**UE 1 (for eMBB slice):**

```
# Initial PDU sessions to be established
sessions:
  - type: 'IPv4'
    apn: 'internet'
    slice:
      sst: 1
      sd: 000010
    emergency: false
```

```
# Configured NSSAI for this UE by HPLMN
configured-nssai:
  - sst: 1
    sd: 000010

# Default Configured NSSAI for this UE
default-nssai:
  - sst: 1
    sd: 000010
```

**UE 2 (for mMTC slice):**

```
# Initial PDU sessions to be established
sessions:
  - type: 'IPv4'
    apn: 'internet'
    slice:
      sst: 2
      sd: 000011
    emergency: false

# Configured NSSAI for this UE by HPLMN
configured-nssai:
  - sst: 2
    sd: 000011

# Default Configured NSSAI for this UE
default-nssai:
  - sst: 2
    sd: 000011
```

Reaching that milestone, we can say that our test-bed is all setup and ready to be used for network slicing experiments.

### 3.3 Testing procedure

Since the main goal of this project is to check how does a 5G mobile core with network slicing parameters performs using Docker containers as its foundations, the following steps are to be followed. So as a first step, the "bring-up the testbed network" will be presented (including 5G Mobile Core, gNodeB and UEs), depicting logs and packet capture for specific sessions and slices. Then an additional container will be deployed to perform measurements for each slices about throughput and delay with similar parameters.

## 4 Test procedure

### 4.1 Bringing up the Mobile Core Network

With all the images and services correctly built, now we can run the Mobile Core Network in docker environment.

```
docker-compose up -d
```

To verify if the docker compose is running, we can check with `docker compose ls` to see of the compose file is running:

```
andres@5g-simulation:~/docker_open5gs$ docker compose ls
NAME              STATUS        CONFIG FILES
docker_open5gs    running(23)   ~/docker_open5gs/docker-compose.yaml
```

Or `docker container ls` to check the state of every individual container:

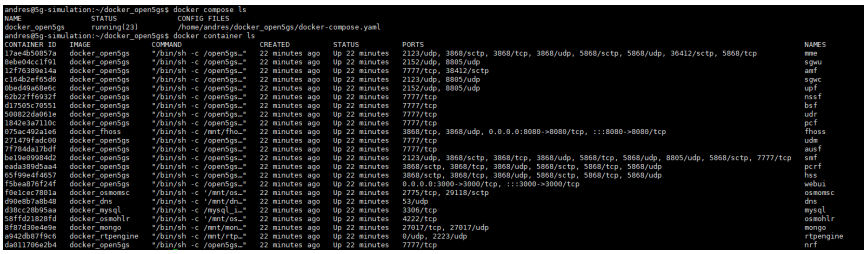

Figure 11: Running containers for 5G Mobile Core

A short version of the *docker-compose.yaml* file is shown below, the complete file will be included in the Gitlab project of the DN Lab form TH Cologne:

```
version: '3'
services:
  mongo:
    build: ./mongo
    image: docker_mongo
    container_name: mongo
    env_file:
      - .env
    volumes:
      - ./mongo:/mnt/mongo
      - mongodbdata:/var/lib/mongodb
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
    expose:
      - "27017/udp"
      - "27017/tcp"
    networks:
      default:
        ipv4_address: ${MONGO_IP}
  webui:
    image: docker_open5gs
    container_name: webui
    depends_on:
      - mongo
    env_file:
      - .env
    environment:
      - COMPONENT_NAME=webui
    volumes:
      - ./webui:/mnt/webui
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
    expose:
      - "3000/tcp"
    ports:
      - "3000:3000/tcp"
    networks:
      default:
        ipv4_address: ${WEBUI_IP}
  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:
      - "7777/tcp"
    networks:
      default:
        ipv4_address: ${NRF_IP}
  ausf:
    image: docker_open5gs
    depends_on:
      - nrf
    container_name: ausf
    env_file:
      - .env
    environment:
      - COMPONENT_NAME=ausf-1
    volumes:
      - ./ausf:/mnt/ausf
      - ./log:/open5gs/install/var/log/open5gs
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
    expose:
      - "7777/tcp"
    networks:
      default:
        ipv4_address: ${AUSF_IP}
  udr:
    image: docker_open5gs
    depends_on:
      - nrf
      - mongo
    container_name: udr
    env_file:
      - .env
    environment:
      - COMPONENT_NAME=udr-1
    volumes:
      - ./udr:/mnt/udr
      - ./log:/open5gs/install/var/log/open5gs
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
    expose:
      - "7777/tcp"
    networks:
      default:
        ipv4_address: ${UDR_IP}
networks:
  default:
    ipam:
      config:
        - subnet: ${TEST_NETWORK}
volumes:
  mongodbdata: {}
  dbdata: {}
```

With the docker compose file up and running, then we must build the docker image of UERANSIM to be used as gNodeB and UE.

## 4.2 Provisioning UEs

Another important step to be started before bringing the RAN containers up, is to add UE configuration via the WEB GUI of Open5GS. That will store all important data (such as IMSI, AMF value, in our case slice configuration and much more) in the MongoDB of the core and then provided to the 5G core functions, so the UEs can be registered and granted all key functions for the tests.

After the `docker-compose up -d` execution, the `webui` container offers a friendly GUI to the user in order to provision User Equipment (UE) information. To enter into the WebUI, one must open a browser and place the IP address of the host machine with port 3000, which is the port exposed for that purposed.

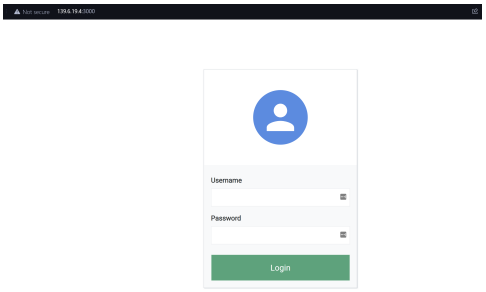


Figure 12: Open5GS Web UI

The default login credentials are `user: admin` and `password:1423`. As a good practice the password was changed after first deployment.

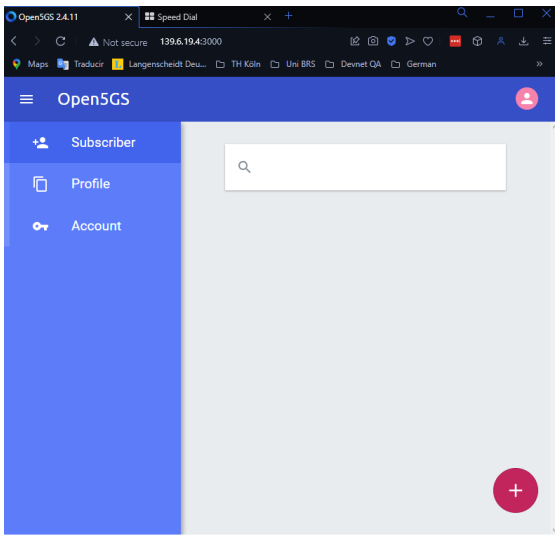


Figure 13: Open5GS Web UI login window

To add a new UE, we must move to the `Subscriber` tab and click on the plus sign and the bottom right corner. Then a new submenu appears to enter all necessary data for the new subscriber. The fields marked with an asterisk `(*)` are mandatory. All these parameters are specified in the `.env` inside the project directory (includes `IMSI`, `Suscriber Key (K)`, `AMF`, `OP Key`. In Figure 14 below, the fields are filled for UE1.

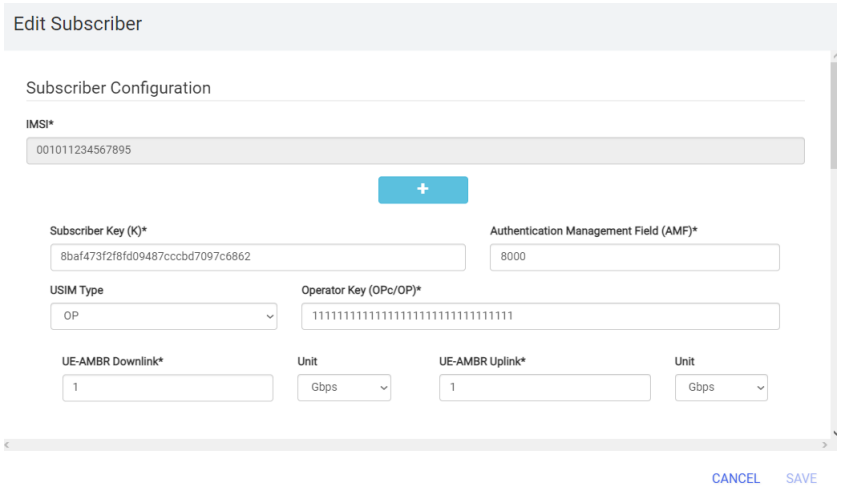


Figure 14: Adding a new UE in Open5GS

For this project purposes, next step is to fill is the `Slice Configuration` section of the subscriber. As of the completion of the practical phase of this project (August 2022), Open5GS has a limit of four different network slices supported. This is enough since the test just involves two different slices to evaluate.

The slice configuration for `UE1` is shown below in Figure 15, with the corresponding values for the eMBB slice which is `sST=1`, `sD=00000a` and rate limit of 1Gbps for Up and Downlink transfer maximum rate.

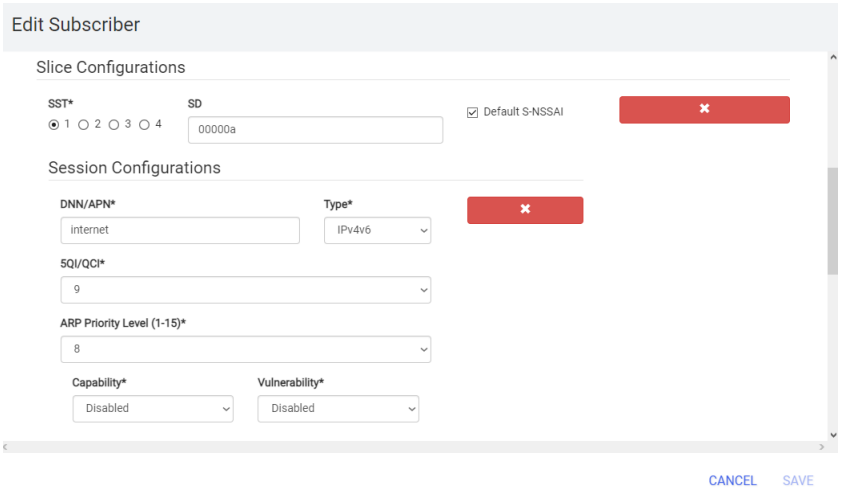


Figure 15: Slice configuration for UE1

The same is depicted for UE2 in Figure 16, which works in mMTC slice, `sST=2`, `sD=00000b` and rate limit of 1Mbps for Up and Downlink transfer maximum rate.

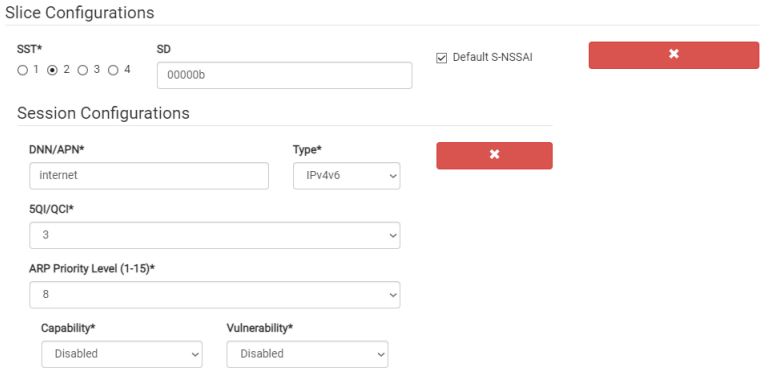


Figure 16: Slice configuration for UE2

One important remark to make, is that both UEs will get registered in the same Access Point Name (APN), so both share the same infrastructure and the slice principle remains in place. After completing all mandatory fields, the final step is to save the configuration by clicking in the save button on the `Suscriber` submenu. Then we can continue to turn up gNodeB and UEs.

## 4.3 Bringing up gNodeB and UEs

To bring up the gNodeB and UEs needed for the tests, and attach them to the previously created docker network, the following commands need to be executed:

```
# UERANSIM gNB
docker-compose -f nr-gnb.yaml up -d && docker attach nr_gnb
# UERANSIM NR-UE 1
docker-compose -f nr-ue.yaml up -d && docker attach nr_ue
# UERANSIM NR-UE 2
docker-compose -f nr-ue2.yaml up -d && docker attach nr_ue2
```

## 4.4 Message exchange and Session establishments

### 4.4.1 gNodeB

**4.4.1.1 gNodeB Logs.** As a first step, when the gNodeB starts to run, it must exchange authentication messages with AMF, so it can be authenticated, checked on capabilities and accepted to offer services to users (UEs). This can be reviewed and analyzed on the logs displayed on the AMF function of Open5GS. In the log picture below, we find the logs in AMF container for the gNB authentication.

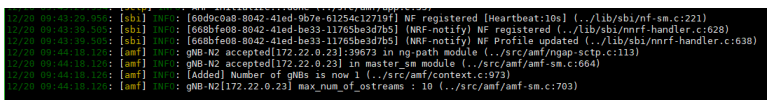


Figure 17: gNodeB authentication with AMF function

Here we can see that the gNodeB gets registered in the AMF function, then the AMF sets up a profile and add the gNB with its IP address (which is `172.22.0.23` in this case), and also a maximum number of streams is fixed. Inita

**4.4.1.2 Packet captures.** To analyze in a deeper level, we can inspect the packets exchanged between AMF and gNB containers. The first packet is a `NGSetUpRequest` message, in which there are four items present:

- **id-GlobalRANNodeID.** Which includes identification values like `gNB-ID` which is the "name" of gNB, in this case is "000001" as is the first base station that gets registered.
- **id-RANNodeName.** Specify a `RANNodeName` for the new gNodeB. In our case is `UERANSIM-gnb-1-1-1`.
- **id-SupportedTAList.** In this item, the gNodeB tells the AMF function (and therefore the whole 5G Core) which slices are available and wants offered to users. Since we already configured in previous steps the slice values, here we can verify that indeed this slices are advertised: `sST=1` and `sD=00000a` which emulates eMBB silce; and `sST=2` and `sD=00000b` which emulates mMTC silce.
- **id-DefaultPagingDRX.** It indicates to the Core the DRX Paging value. Paging in Mobile Networks is an important feature, since it allows UEs to remain in "idle mode" until they actually have "something to say" to basestations and Core Network, to save battery power. UEs in RRC Idle mode use Discontinuous Reception (DRX) also known as paging cycle to reduce power consumption. This DRX cycle determines how frequently UE check for paging messages. In this case it is set to `v128 (2)`, which means 128 radio frames to wait, that is translated to 1.28 seconds.

Below the packet capture is depicted, splitted into two figures to get the whole message complete. Figure 18 shows `id-GlobalRANNodeID` and `id-RANNodeName` fields.

```
279 2.572415    172.22.0.23          172.22.0.10      NGAP              144 NGSetupRequest
Frame 279: 144 bytes on wire (1152 bits), 144 bytes captured (1152 bits)
Linux cooked capture v2
Internet Protocol Version 4, Src: 172.22.0.23, Dst: 172.22.0.10
Stream Control Transmission Protocol, Src Port: 36940 (36940), Dst Port: 38412 (38412)
NG Application Protocol (NGSetupRequest)
  NGAP-PDU: initiatingMessage (0)
    initiatingMessage
      procedureCode: id-NGSetup (21)
      criticality: reject (0)
      value
        NGSetupRequest
          protocolIEs: 4 items
            Item 0: id-GlobalRANNodeID
              ProtocolIE-Field
                id: id-GlobalRANNodeID (27)
                criticality: reject (0)
              value
                GlobalRANNodeID: globalGNB-ID (0)
                  globalGNB-ID
                    pLMNIdentity: 00f110
                      Mobile Country Code (MCC): Unknown (1)
                      Mobile Network Code (MNC): Unknown (01)
                    gNB-ID: gNB-ID (0)
                      gNB-ID: 00000001 [bit length 32, 0000 0000  0000 0000  0000 0000  0000 0001 decimal value 1]
            Item 1: id-RANNodeName
              ProtocolIE-Field
                id: id-RANNodeName (82)
                criticality: ignore (1)
              value
                RANNodeName: UERANSIM-gnb-1-1-1
```

Figure 18: gNodeB Setup Request in AMF (part1)

In Figure 19 `id-SupportedTAList` and `id-DefaultPagingDRX` are shown.

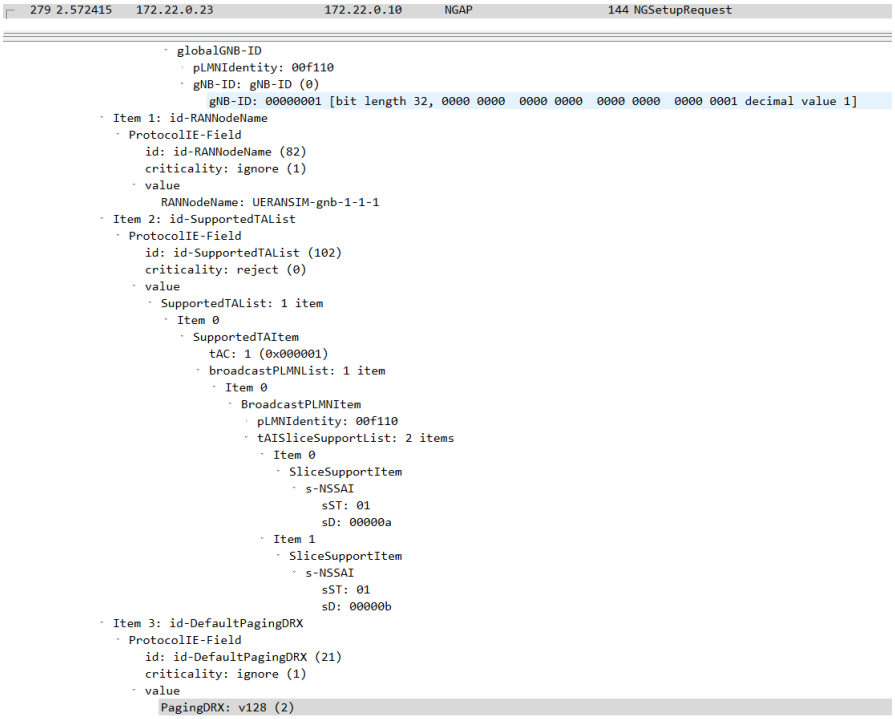


Figure 19: gNodeB Setup Request in AMF (part2)

The next message is the response sent by the AMF function to gNodeB. Within this message, which states a "Successfull Outcome" of the previous request, there are also four items or fields to explain:

- **id-AMFName.** It advertises the `AMFName` which is a basic name of AMF core function, in this case is `open5gs-amf0` for this testbed. More details are exchanged in the next item.
- **id-ServedGUAMIList.** GUAMI (Globally Unique AMF ID) is used to uniquely identify an AMF within a 5G network. It is comprised of the MCC, MNC, `AMF Region ID` (set to 02), `AMF Set ID` (set to 0040) and `AMF Pointer` (set to 00).
- **id-RelativeAMFCapacity.** It indicates the relative processing capacity of an AMF with respect to the other AMFs in the AMF Set in order to load-balance AMFs. In our case is set to 255, which makes sense since it is the only AMF available to process data.
- **id-PLMNSupportList.** Similar to the `id-SupportedTAList` , but in this case the AMF function confirms the offered slices are valid and ready to be used in the Mobile Core. Here we can once again affirm that the slices are correctly accepted: `sST=1` and `sD=00000a` which emulates eMBB silce; and `sST=2` and `sD=00000b` which emulates mMTC silce.

Below the packet capture is depicted, once again splitted into two figures to get the whole message complete. `id-AMFName` and `id-ServedGUAMIList` are shown

in the Figure 20.

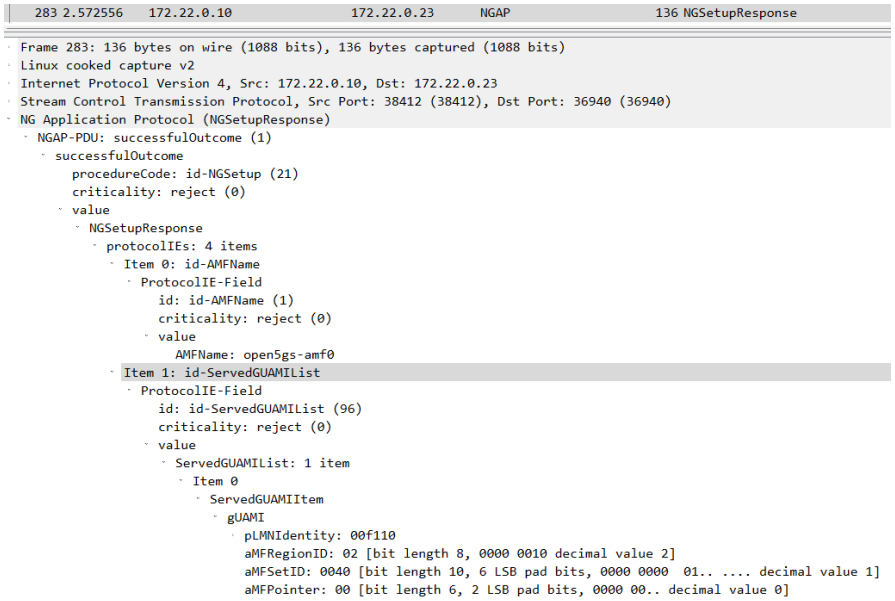


Figure 20: gNodeB Setup Response in AMF (part1)

In Figure 21 below `id-RelativeAMFCapacity` and `id-PLMNSupportList` are shown.

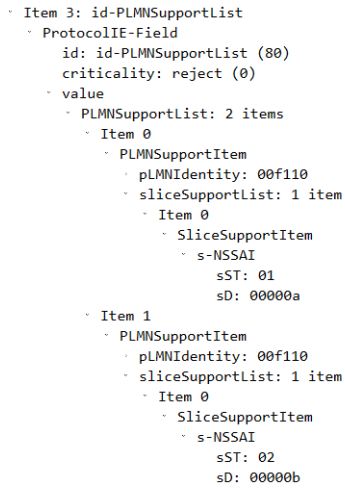


Figure 21: gNodeB Setup Response in AMF (part2)

### 4.4.2 User Equipments

**4.4.2.1 UE logs.** We will check the logs going from "bottoms up" way, meaning from UE up to the 5G mobile core.

In the UE, the entire registration process is illustrated, since there are the following messages:

- Receives and ACK for the registration request
- Switches state to `[MM-REGISTERED/NORMAL-SERVICE]` and sends registration complete.
- Sends a PDU Session Establishment Request.
- After the Core (AMF function) authenticates and grants acccess, a Config Update message is received with initial setup parameters such as IP Address and network slice parameters.
- Finally, PDU Session Establishments is successful and the `uesimtun0` network interface comes up.

The log picture for UE1 is shown below in Figure 22.

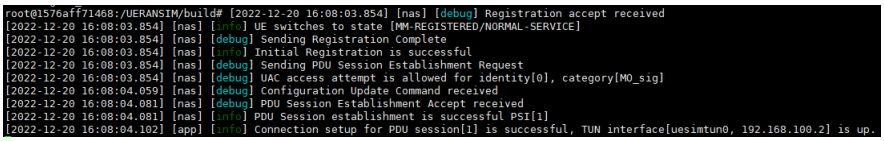


Figure 22: UE 1 turn up log

The same can be checked in Figure 23 for UE2:

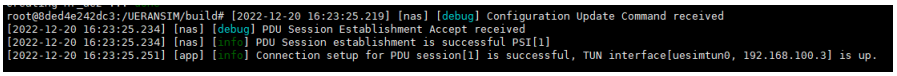


Figure 23: UE 2 turn up logs

In gNodeB we can check logs for bot UEs, the process consists of the following:

- The gNodeB identifies a new signal for each UE and starts Radio Resource Control (RRC) protocol exchange, to setup Air Interface parameters (completely emulated and assumed for UERANSIM). The major functions of the RRC protocol include connection establishment and release functions, broadcast of system information, radio bearer establishment, reconfiguration and release, RRC connection mobility procedures, paging notification and release and outer loop power control
- Next, the NAS (Non-access stratum) protocol exchange starts, in order to maintain a continous comunication between UE and gNB and core as it moves through the network.
- Finally, PDU Session is established between UE and Mobile Core Network, so the UE setup is complete.

The complete log process is illustrated below:

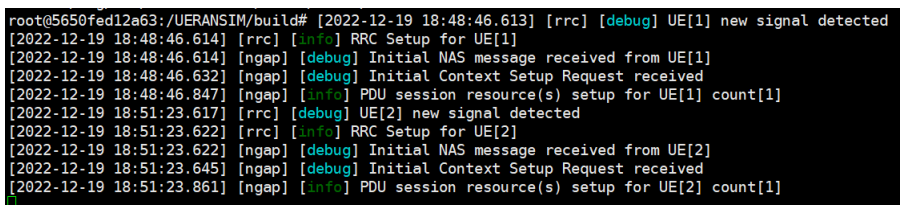


Figure 24: UE setup in gNodeB

In the 5G Core, three important functions will produce logs during the UE setup.

In the AMF function, the authtentication process goes as following:

- AMF receives an `Initial UE Message`, and after verifying that the UE data is correct (registered in Core database), it gets identified and added to the AMF.
- Next, the Registration process starts by specifying which functions and capabilities does the UE support and can actually access. This information is transmited in a `Configuration update command` message.
- Finally, UE gets granted access and capabilities accordingly to the APN and Network Slice that it belongs to.

In the figure below, the process for UE2 is showed, we can see the corresponding values; `DNN:internet, SST:2, SD:0xb`.

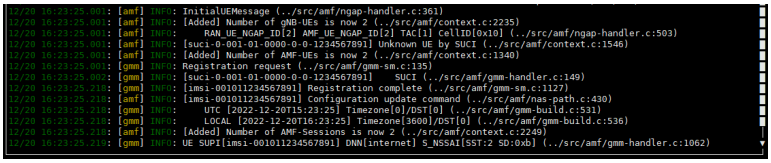

Figure 25: UE2 authentication in AMF

UE1 produces a similar log result in AMF function, which is shown below.

Figure 26: UE1 authentication in AMF

In the UPF function, there is a new UPF-session added everytime a newly registered UE becomes alive. If the UE was already registered, UPF just reactivates the old session checking on parameters.

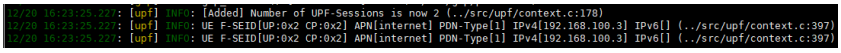

Figure 27: UE logs in UPF function

The behavior in SMF function is similar as UPF, since it handles sessions (PDU Session) to be specific for each UE, so everytime a UE want to establish connections (mostly outgoing connections to internet), this functions gets queried.

Figure 28: UE logs in SMF function

**4.4.2.2 Packet captures.** In the packet capture we can further check message exchange for UE registration with more details. One important point to mention, is that these messages are exchanged between gNodeB and AMF, on behalf of UE, so there is no direct communication from UE to AMF in this case. This can be reflected in the picture below, where all authentication and negotiation messages are sent between gNB `IP add: 172.22.0.22` and AMF container `172.22.0.10`.

| | | | | | | |
|---|---|---|---|---|---|---|
| 528 | 4.892710 | 172.22.0.23 | 172.22.0.10 | NGAP/NAS-5GS | 144 | InitialUEMessage, Registration request |
| 748 | 4.897921 | 172.22.0.10 | 172.22.0.23 | NGAP/NAS-5GS | 152 | SACK (Ack=0, Arwnd=106496) , DownlinkNASTransport, Authentication request |
| 754 | 4.898715 | 172.22.0.23 | 172.22.0.10 | NGAP/NAS-5GS | 152 | SACK (Ack=0, Arwnd=106496) , UplinkNASTransport, Authentication response |
| 800 | 4.900692 | 172.22.0.10 | 172.22.0.23 | NGAP/NAS-5GS | 132 | SACK (Ack=1, Arwnd=106496) , DownlinkNASTransport, Security mode command |
| 806 | 4.901408 | 172.22.0.23 | 172.22.0.10 | NGAP/NAS-5GS | 196 | SACK (Ack=1, Arwnd=106496) , UplinkNASTransport |
| 1118 | 4.909559 | 172.22.0.10 | 172.22.0.23 | NGAP/NAS-5GS | 240 | SACK (Ack=2, Arwnd=106496) , InitialContextSetupRequest |
| 1120 | 4.909900 | 172.22.0.23 | 172.22.0.10 | NGAP | 104 | SACK (Ack=2, Arwnd=106496) , InitialContextSetupResponse |
| 1174 | 5.111685 | 172.22.0.23 | 172.22.0.10 | NGAP/NAS-5GS | 248 | UplinkNASTransport, UplinkNASTransport |
| 1176 | 5.112137 | 172.22.0.10 | 172.22.0.23 | NGAP/NAS-5GS | 148 | SACK (Ack=5, Arwnd=106496) , DownlinkNASTransport |
| 1613 | 5.130231 | 172.22.0.10 | 172.22.0.23 | NGAP/NAS-5GS | 256 | PDUSessionResourceSetupRequest |
| 1631 | 5.135276 | 172.22.0.23 | 172.22.0.10 | NGAP | 108 | PDUSessionResourceSetupResponse |

Figure 29: Message exchange for UE authentication

In Figure 30, the packet capture is shown for id-AMF-UE-NGAP-ID and PDUSessionResourceSetupItemSURe values.

```
1613 5.130231   172.22.0.10          172.22.0.23     NGAP/NAS-5GS          256 PDUSessionResourceSetupRequest

NG Application Protocol (PDUSessionResourceSetupRequest)
  NGAP-PDU: initiatingMessage (0)
    initiatingMessage
        procedureCode: id-PDUSessionResourceSetup (29)
        criticality: reject (0)
      value
        PDUSessionResourceSetupRequest
          protocolIEs: 4 items
            Item 0: id-AMF-UE-NGAP-ID
              ProtocolIE-Field
                  id: id-AMF-UE-NGAP-ID (10)
                  criticality: reject (0)
                value
                    AMF-UE-NGAP-ID: 1
            Item 1: id-RAN-UE-NGAP-ID
              ProtocolIE-Field
                  id: id-RAN-UE-NGAP-ID (85)
                  criticality: reject (0)
                value
                    RAN-UE-NGAP-ID: 1
            Item 2: id-PDUSessionResourceSetupListSUReq
              ProtocolIE-Field
                  id: id-PDUSessionResourceSetupListSUReq (74)
                  criticality: reject (0)
                value
                  PDUSessionResourceSetupListSUReq: 1 item
                    Item 0
                      PDUSessionResourceSetupItemSUReq
                          pDUSessionID: 1
                        pDUSessionNAS-PDU: 7e021e25fe25037e006801004a2e0101c21100090100063131010 1ff01060b00010b0001…
                          Non-Access-Stratum 5GS (NAS)PDU
                            Security protected NAS 5GS message
                                Extended protocol discriminator: 5G mobility management messages (126)
                                0000 .... = Spare Half Octet: 0
                                .... 0010 = Security header type: Integrity protected and ciphered (2)
                                Message authentication code: 0x1e25fe25
                                Sequence number: 3
                              Encrypted data
                        s-NSSAI
                            sST: 01
                            sD: 00000a
```

Figure 30: PDU Session Request message for UE1 (Part 1)

One interesting packet to analyze is the `PDUSessionResourceSetupRequest`, in which many important UE parameters are exchanged. To name the most relevant:

- **id-AMF-UE-NGAP-ID:** which identifies with a number (ID) the sesion between UE and AMF. It is used to distinguished this session among the ones with other UEs so the correct parameters can be passed and controlled between Mobile Core and UE. Technote [4]
- **PDUSessionResourceSetupItemSUReq:** Which includes encryption parameters for the messages, and most importantly `s-NSSAI` information, or in other words, network slice configuration values.

- **id-UL-NGU-UP-TNLInformation:** in this field, important details about GTP tunnelling are exchanged, stating the `TransportLayerAddress`. This is set to `172.22.0.8`.
- **id-QosFlowSetupRequestList:** contains important information about QoS values for 5G mobile core network. The `fiveQI` value is set to `9` in eMBB slice and to `1` in mMTC slice.
- **id-UEAggregateMaximumBitRate:** explicitly states the maximum bit rate (for both Uplink and Downlink channels). In eMBB a maximum rate of `1Gbps` is set, and a maximum value of `1Mbps` is set for mMTC slice.

An important parameter to observe is also the Message Authentication Code (MAC), which is suposed to be encrypted in real world application. We can see as well, that the capture corresponds to UE1 authentication process, since the slices parameters correspond to this particular device, with `sST:01` and `sD:00000a`.

In Figure 31, the packet information about QoS values, GTP tunneling and Max Bit Rate is illustrated:

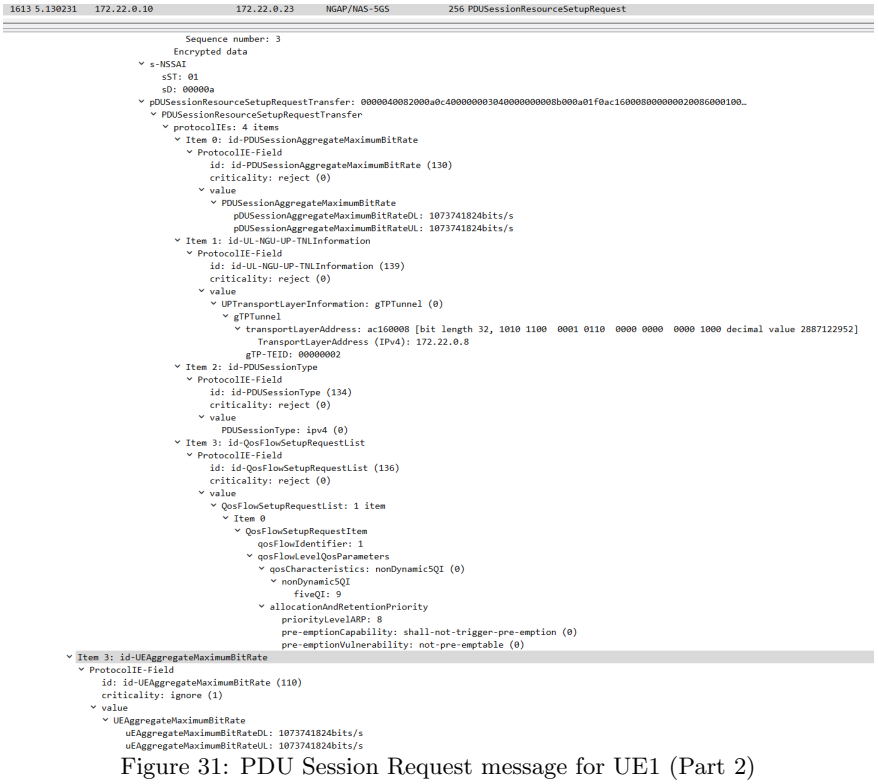

Figure 31: PDU Session Request message for UE1 (Part 2)

Another detail that worth to observe, is how SMF and UPF transmit messages between each other, in order to establish GTP tunnel parameters. This is accomplished using Packet Forwarding Control Protocol (PFCP), and in the `SessionModificationRequest` packet the functions agree regarding the "outer" IP address for future packet forwarding, from the UE to outside. In Figure 32

we can see in detail this packet and its data.

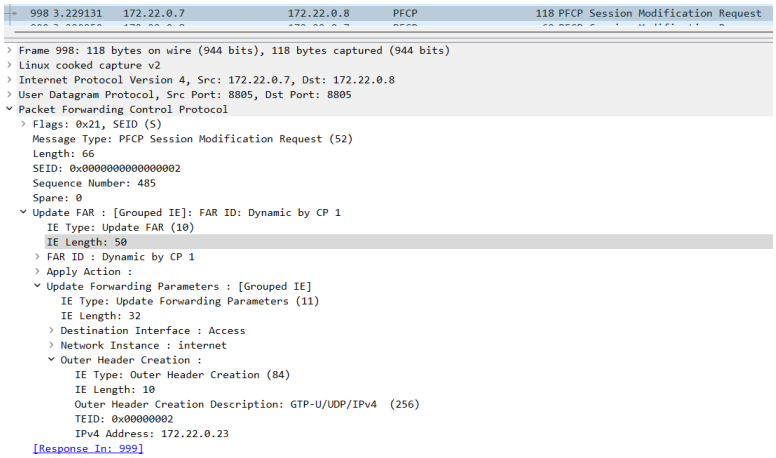


Figure 32: PFCP exchange between UPF and SMF

To check if the GTP encapsulation is working properly (meaning, that the User Plane of 5G Mobile Core is working properly), a simple PING message is sent to one public DNS server to check if it is reachable and duly encapsulated with GTP. In Figure 33 we can see the `ping` command used, noticing the `-I useimtun0` option that forces the ICMP packets to be sent with the "SIM card" IP address as source address.

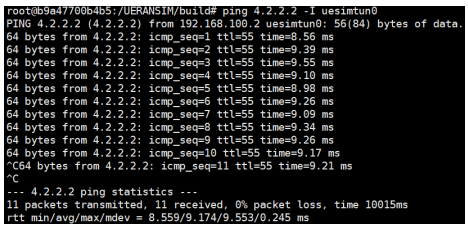


Figure 33: Ping test from UE2 to public DNS

This packets were captured with wireshark and we confirm that GTP is used to encapsulate the ICMP packets that are sent to the outside network. Check Figure 34 were all details are shown:

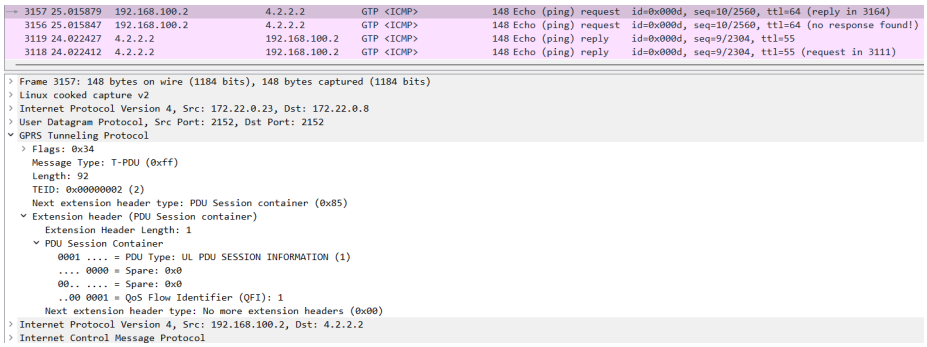


Figure 34: Packet capture of Ping test from UE2 to public DNS

Note that there is a QoS Flow Identifier (QFI) is displayed there, but it doesn't correspond with 5QI values that were configured in UE provisioning, it corresponds to legacy fields for LTE sessions.

## 4.5 iPerf Tests

The chosen tool to make throughput evaluation of network slices is iPerf3, as one of the easiest and more practical way to test a network channel. So, as a first step in the `Dockerfile` used to build `docker_ueransim` container image which is used to run UEs.

As explained in previous sections, iPerf uses a client-server topology. In our case, the UEs will act as "clients" and an external container acting as "server" will be attached to `docker_open5gs_default` container network so it can reach both UEs easily.

### 4.5.1 iPerf server in outside container

The iPerf server is setup as a "standalone container" meaning it is just turned up for testing purposes and then deleted inmediatly. We use a public docker image form Dockerhub `(networkstatic/iperf3)` with the following command:

```
$ docker run  -it --rm --name=iperf3-server -p 5201:5201 --network
    docker_open5gs_default --ip 172.22.0.50 networkstatic/iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

We pass the following options for the docker run command:

- To indicate to which docker network this container must get connected, we use option `--network docker_open5gs_default`.
- An IP address must be set, for that the option `--ip 172.22.0.50` is used.

As we can see the iperf3 container is running in server mode and listening for incoming connections on port `5201`. So next step is to trigger test on client side (UEs).

### 4.5.2 iPerf client on UEs

As soon as we start to run iPerf as client mode in each UE, the test will begin and will print results to `stdout`. So the command used to run the test looks as follows:

```
# iperf3 -c 172.22.0.50 -B 192.168.100.2 -b 100K -O 2 -u -V -t 12 -
    R -p 5201
iperf 3.7
....
[  5] local 192.168.100.2 port 46425 connected to 172.22.0.50 port
    5201
Starting Test: protocol: UDP, 1 streams, 1348 byte blocks, omitting
     2 seconds, 12 second test, tos 0
[ ID] Interval           Transfer     Bitrate         Total
    Datagrams
[  5]   0.00-1.00   sec  13.2 KBytes   108 Kbits/sec  10  (omitted)
[  5]   1.00-2.00   sec  11.8 KBytes  97.1 Kbits/sec  9  (omitted)
```

```
[  5]   0.00-1.00   sec  11.8 KBytes  97.0 Kbits/sec  9
.....
[  5]  11.00-12.00  sec  11.8 KBytes  97.1 Kbits/sec  9
- - - - - - - - - - - - - - - - - - - - - - - - -
Test Complete. Summary Results:
[ ID] Interval           Transfer     Bitrate         Jitter
    Lost/Total Datagrams
[  5]   0.00-12.00  sec   146 KBytes  99.8 Kbits/sec  0.000 ms
    0/111 (0%)  sender
[  5]   0.00-12.04  sec   146 KBytes  99.4 Kbits/sec  0.062 ms
    0/111 (0%)  receiver
CPU Utilization: local/sender 3.8% (2.0%u/1.9%s), remote/receiver
    0.0% (0.0%u/0.0%s)

iperf Done.
```

The iperf3 client options used for the test are the following:

- `-c` option indicates that is running in client mode.
- `-B 192.168.100.2` binds the `uesimtun0` IP address as source address to send packets.
- `-b 100k` sets the maximum bandwidth to be tested (100 kbps in the example above).
- `-O 2` tells iPerf3 to omit the first two tests, to avoid unexpected high values due to TCP slow start mechanism.
- `-u` forces UDP to be used.
- `-t 12` so the test has a 12 seconds duration, but since we are omitting the first two seconds, its 10 effective secconds to be taken into account.
- `-R` sets reverse mode, meaning that the server starts the test by sending packets to client.
- `-V` defines more "verbosity" in this test.
- `-p 5201` the port to which client must connect to.

Since the slices applications differ a lot between each other (in bandwidth terms), two different sets of values are defined per slice. The performance values that are documented are packet loss and jitter (offered directly in the output by iPerf3).

**Values for eMBB Slice (Max Bandwidth 1 Gbps)**

- 500 Kbps
- 1 Mbps
- 10 Mbps
- 50 Mbps
- 100 Mbps
- 500 Mbps
- 1 Gbps

- 1.5 Gbps

**Values for mMTC Slice (Max Bandwidth 1 Mbps)**

- 1 Kbps
- 10 Kbps
- 20 Kbps
- 50 Kbps
- 100 Kbps
- 500 Kbps
- 1 Mbps
- 1.5 Mbps
- 2 Mbps

## 4.6 Plot results

The iPerf test were recorded and will be stored in separate files, so just a table for each test set will be shown here with the final results. Table 1 show results for the test performed for UE1.

| Bandwith (bps) | Jitter (ms) | Packet Loss (%) |
|---|---|---|
| 500 K | 0.081 | 0 |
| 1 M | 0.133 | 0 |
| 10 M | 0.019 | 0 |
| 50 M | 0.027 | 0 |
| 100 M | 0.023 | 0 |
| 500 M | 0.006 | 0.21 |
| 1 G | 0.006 | 3.4 |
| 1.5 G | 0.01 | 27 |

Table 1: Bandwidth test results for UE1 (eMBB slice)

Table 2 depicts test results for UE2.

| Bandwith (bps) | Jitter (ms) | Packet Loss (%) |
|---|---|---|
| 1 K | 0.0003 | 0 |
| 10 K | 0.049 | 0 |
| 20 K | 0.081 | 0 |
| 50 K | 0.129 | 0 |
| 100 K | 0.096 | 0 |
| 500 K | 0.129 | 0.0 |
| 1 M | 0.093 | 30 |
| 1.5 M | 0.102 | 31 |
| 2 M | 0.127 | 49 |

Table 2: Bandwidth test results for UE2 (mMTC slice)

The graphical plots for these experiments are shown below. They clearly depict an expected behavior of increasing packet loss as soon as we reach and pass the bandwidth limit for each slice.

**Graphic plot for UE1**

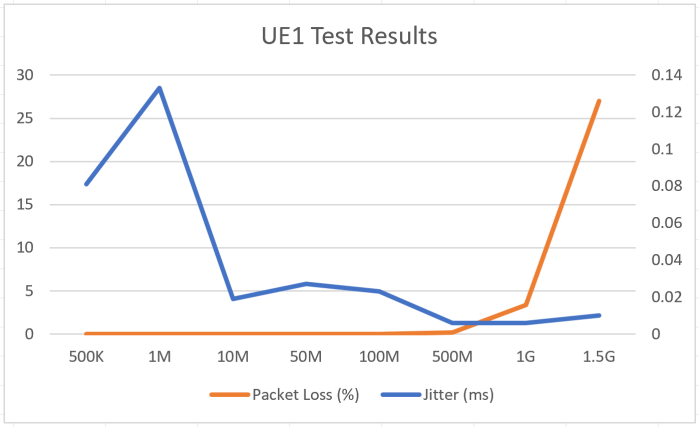


Figure 35: Packet loss and Jitter results for UE1

**Graphic plot for UE2**

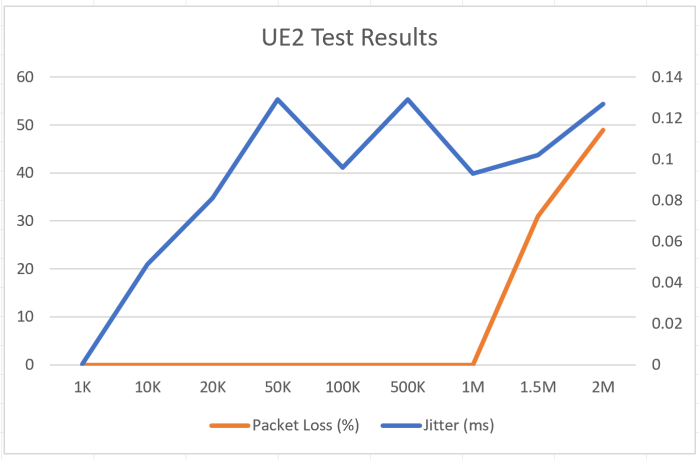


Figure 36: Packet loss and Jitter results for UE2

# 5 Data analysis and Conclusions

In this final sections, an analysis about the results of the tests and experiments is presented for future references and be recalled by possible readers.

## 5.1 Data analysis

### 5.1.1 Docker behavior for future test

- The containerized option of Open5GS offer a flexible solution to easily deploy a 5G Core Network in virtual machines. It could be easily taken also to public clouds since the docker-compose.yaml is already provided and ready to be used.
- All containers handling network functions worked as expected, providing full functionality for the experiment purpose. With all `C` language running smoothly on the background keeping all 5G signaling. In the case of containerized environments, an important consideration is to keep track of the open ports to offer each service for each 5G function.
- Since there is no jitter introduced by "external factors" in these test rounds, it only depends on the machine performance and state during the tests runs, which can be fastly confirmed with the irregular jitter values obtained on both UEs.
- Open5GS and its containerized deployments offer as well the posibility to connect to external RAN portions, so it is really flexible for both experimental and real-world purposes.
- The docker network works perfectly fine for the experiment. There is even proven topologies that include IMS functions to provide voice functionalities.

Another important remark to show, is that CPU consumption for each iPerf test reflected a small growth, depending on the bandwidth tested. In the UPF container, which is the function that handles packet forwarding for this scenario, the CPU utilization went to maximum value of 13.12% for a test of 5 Gbps maximum throughput, which sounds well but might be too much for just one UE demanding top bandwidth.

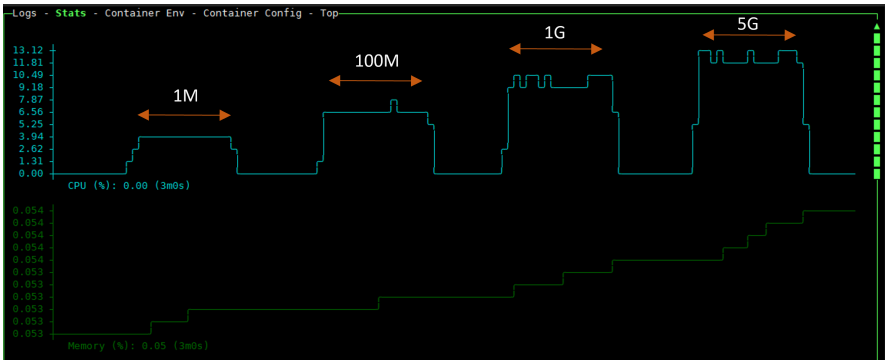


Figure 37: UPF CPU utilization during bandwidth tests

In Figure 37 above, we can check a peak 10.5% CPU utilization for 1Gbps tests, a peak 6.56% for 100 Mbps tests and 3.94% for 1 Mbps tests.

### 5.1.2 Network slicing performance

- Both slices fulfill their main purpose, which was to limit the maximum data rate for Uplink and Downlink channels. The tests reflect a good performance in this matter, despite providing an abnormal high bandwidth at the beggining of the iPerf test.
- Jitter can't be properly simulated and tested since all is deployed on a virtual environment (including the Air Interface which introduces the most disturbance into the UE communication). However, Open5GS has proved that it is reliable in that area keeping jitter to minimum levels.
- It is mandatory to modify all container configuration of 5G core functions so the UEs and Slices are properly authenticated and provisioned.
- Network slicing is a potential boosting feature to improve Quality of Experience for every mobile user, since it can boost applications performance based on specific requirements for each application. Therefore, more experiments including open source solutions for RAN portion could lead to great development in 5G core.

### 5.1.3 Lessons learned

- In the present environment since all is "directly-connected" in a virtual level (docker network), all jitter values only depend on computing power and actual CPU use of host machine. This can be confirmed with the irregular values of jitter obtained in each test run, which doesn't depend on bandwidth and doesn't actually influence on packet loss values.
- Some issues were faced after updating UERANSIM version to the latest stable, so as a good practice is a nice idea to always get the project and results in a git repository, maybe even tag each progress so it can be easily move backwards when needed to previous versions.
- After every change made into the configuration files, a new docker build and docker-compose build must be run, so trying to keep this rebuild as few as possible is a nice idea to save time. Also pushing docker images to a public or private registry is a good practice that can be included.

### 5.1.4 Future work

- For now, in this solution there is no support for IPv6, which drives 5G new deployments in the industry, so it would be useful for future testers to try to include it in the supported protocols list.
- The docker compose solution is not easy to escalate and maintain if more services are included, so a different orchestration tool like Kubernetes could be used. A good manifest file or Helm Chart file could be developed and published to make a 5G Core deployment even more portable and flexible with the user.
- Including real hardware in RAN portion could lead to better and more realistic results, and there are already articles explaining about Voice-over-LTE (VoLTE) deployments in real world scenarios for small business.

## 5.2 Conclusions

- Open5GS keeps a good seed of knowledge in the official documentation and a big community that supports and contributes to its continuous improvement, which is a great advantage over some other open source solutions for 5G core, so it is the better option (in the author's opinion) for 5G tests and projects.
- The docker container technology is perfect for this type of solucions and for fast and "portable" deployments. It behaves as a robust and "normal" virtual machine with much less used resources, even for demanding applications such as a 5G Core.
- More conclusive analysis could be reached with real-world RAN portion experiments, although there is already articles and posts on the internet that proves excellent performance on that matter.

# Bibliography

**1** Cox, Christopher: *An Introduction to 5G*. 1. Wiley, Hoboken, NJ. USA, 2018. – ISBN: 9781119602668

**2** Hassan, Syed F.: *A Network Architect's Guide to 5G*. 1. Pearson Education Limited, Hoboken, NJ. USA, 2022. – ISBN: 0-13-737684-7

**3** Sudhakar Shetty, Rajaneesh: *5G Mobile Core Network : Design, Deployment, Automation, and Testing Strategies*. 1. Apress, Bangalore, Kanata, Inidia, 2018. – ISBN: 978-1-4842-6472-0

**4** Technote, Share: *IP Network - NGAP*. `https://www.sharetechnote.com/html/IP_Network_NGAP.html`. Version: Februar 2021. – Last accessed 18 November 2022

**5** Zhang, Ying: *Network Function Virtualization: Concepts and Applicability in 5G*. 1. Wiley-IEEE Press, Hoboken, NJ. USA, 2018. – ISBN: 9781119390602