UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

5G-ENABLED EDGE DEPLOYMENTS FOR EMERGING REAL-TIME SERVICES

IMANOL ROJAS PÉREZ

Thesis supervisor: ELLI KARTSAKLI (Barcelona Supercomputing Center (BSC-CNS))

Thesis co-supervisor: EDUARDO QUIÑONES MORENO

Tutor: FELIX FREITAG (Department of Computer Architecture)

Degree: Master's Degree in Innovation and Research in Informatics (Computer Networks and Distributed Systems)

Master's thesis

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Acknowledgements

Quiero expresar mi más sincero agradecimiento a Elli Kartsakli por su ayuda en el desarrollo de este proyecto, tanto a nivel técnico como personal, y por su apoyo decisivo en la redacción de esta memoria. Asimismo, agradezco a Eduardo Quiñones por brindarme la oportunidad de formar parte del grupo de investigación y a su vez de este proyecto.

Debo un especial reconocimiento a mis compañeros de trabajo y del máster: Alba, Oriol y Riccardo, por su constante apoyo y ayuda en todo momento, tanto durante el máster como en el desarrollo de este proyecto. También quiero agradecer a Kiara por acompañarme y apoyarme en este trabajo y hacer el proceso mucho más ameno.

Finalmente, gracias a mi familia, que siempre ha creído en mí y me ha apoyado hasta el final. Otra vez, gracias a todos de corazón.

Abstract

This thesis examines the integration of 5G technology with edge computing to address the challenge of reducing latency in telecommunications, towards enhancing performance for real-time applications such as autonomous driving, telemedicine, online gaming, and smart city infrastructures. Central to this research is the concept that hosting applications, processing and routing data at the network's edge, closer to its source, can significantly decrease transmission delays, offering critical improvements in safety, decision-making speed, and interactive experiences. By deploying a practical 5G testbed with edge computing capabilities, the study not only demonstrates considerable latency reductions but also validates the feasibility of hosting and managing application data directly at the edge, thereby presenting a more efficient, cost-effective, and secure alternative to conventional cloud processing. This work underscores the transformative potential of combining 5G and edge computing in creating responsive, high-performance telecommunications systems.

Table of contents

List of figures **6**

List of tables **9**

Nomenclature **10**

1 Introduction **1**
1.1 Context and hypothesis 1
1.2 Scope, objectives and contributions 2
1.3 Thesis structure 3

2 Background and literature review **4**
2.1 Overview of the 5G architecture 4
2.1.1 Core Network (CN) 7
2.1.2 Access Network (AN) 12
2.1.3 Deployment Options 14
2.2 Edge computing in the context of 5G 16
2.2.1 The edge computing concept and its importance in 5G 16
2.2.2 Where is the edge of the network? 17
2.3 State of the art 19

3 Methodology **22**
3.1 Introduction to the methodology 22
3.2 Employed frameworks 23
3.2.1 Open5GS 23
3.2.2 UERANSIM 25
3.2.3 Docker and Docker Compose 26
3.2.4 Jetson Inference 27
3.3 Employed equipment 28

3.3.1 Workstation . . . 28
3.3.2 AMARI Callbox Classic . . . 28
3.3.3 Edge devices . . . 31
3.3.4 UEs . . . 32
3.4 Testbed architecture . . . 33
3.5 Performance metrics . . . 35
3.6 Application Scenarios . . . 36

4 Testbed deployment 37
4.1 First deployments . . . 37
4.1.1 Example deployments . . . 37
4.1.2 Docker based deployments on a single host . . . 45
4.1.3 Other initial deployments . . . 52
4.2 Final deployment . . . 54

5 Metrics and performance evaluation 60
5.1 Cloud physical delay simulation and delay measurements . . . 60
5.2 Throughput and jitter metrics discussion . . . 64

6 Testbed applications 67
6.1 Messaging service . . . 67
6.2 Video streaming service . . . 71
6.3 Real time object detection service . . . 75

7 Conclusions and future work 83
7.1 Conclusions . . . 83
7.2 Limitations and future work . . . 86
7.2.1 Limitations . . . 86
7.2.2 Future work . . . 87

References 90

List of figures

2.1 Generations of mobile communications and its main features 5
2.2 Service categories in 5G [1] 7
2.3 Reference Point architecture representation [2] 8
2.4 Service Based Architecture representation [2] 8
2.5 NG-RAN architecture [3] 13
2.6 UP and CP protocol stacks in NR [4] 14

3.1 Open5GS Core diagram [5] 23
3.2 Amarisoft Core architecture [6] 30
3.3 Main testbed architecture 34

4.1 *Example 1*: Open5GS and UERANSIM example from the Open5GS documentation, which selects the UPF based on the configured DNN on the UEs 38
4.2 *Example 2*: Open5GS and UERANSIM example from the Open5GS documentation, which selects the UPF based on the configured NSSAI on the UEs 39
4.3 Table of IPs and VM characteristics for the second example extracted from the GitHub Tutorial 40
4.4 Virtual Box showing the group of configured VMs for *Example 2* and the Open5GS initial VM, Open5GSZero, which has been cloned. 45
4.5 *Example 2* VM1 C-Plane screenshot showing the commands that are executed automatically during boot time 46
4.6 *Deployment 1* and *Deployment 2* diagram, showing the architecture of the network. 46
4.7 AMF list in the gNB configuration of AMARI Callbox Classic 51
4.8 PLMN list in the gNB configuration of the AMARI Callbox Classic 51
4.9 Final testbed diagram 55

4.10 AMF Open5GS logs showing correct functioning and UE registration. . . . 58
4.11 SMF Open5GS logs showing correct functioning and UE session creation. . 59
4.12 UPF Open5GS logs showing correct initialization and UE given IP and 5G services using the edge APN. 59
4.13 Curl made from the UE to check internet connection. 59

5.1 Testbed architecture 62
5.2 Comparison of communication paths using cloud and edge APN routing . . 63
5.3 Worst case scenario experimental paths for edge and cloud APNs 64
5.4 First measurements using `iperf3` with the simulated delay using `tc` applied in the cloud. 65

6.1 Response to the command `docker ps` after deploying the MQTT broker. . 68
6.2 MQTT application working. The message sent by the UE is received on the edge. 69
6.3 *ogstun* interface IP shown on the edge UPF container shell. 70
6.4 `tcpdump` shows that the *ogstun* interface routes all the MQTT messages. These messages are distinctive because they use port 1883. 70
6.5 Streaming service is deployed successfully. 72
6.6 Binary contents from the streaming are outputted using `curl` on the UE (above) and `tcpdump` in the edge UPF shows HTTP traffic from the streaming going to the UE (below). 73
6.7 VLC network streaming configuration. 74
6.8 Streaming works on the UE side. 74
6.9 Diagram of the architecture behind the application scenario 76
6.10 Routes in Jetson Nano after adding the route to communicate with Jetson AGX using the *usb1* interface 76
6.11 Output of the `lsusb` command on the Jetson Nano shows that the camera and the modem are connected to the USB ports. 77
6.12 Different video devices of the camera connected to the UE are mounted on the *jetson-inference* container 77
6.13 Screenshot of the UE screen that shows the GStreamer pop up window with the raw footage of the video camera connected to it 78
6.14 The video from the camera is sent to the Jetson AGX as an RTP streaming on port 1234 79
6.15 `tcpdump` command in the Jetson AGX shows how the RTP streaming sent by the UE is arriving at port 1234 79

6.16 `tcpdump` performed over the *ogstun* interface on the UPF shows how the UE traffic arrives to the edge using the 5G GTP tunnel 80
6.17 Output of the *detectnet* command that executes the deep neural network inference over the streaming video received 80
6.18 The streaming chain works using 5G and the AI inference is displayed over the raw camera video. These images are captured from the video saved on the Jetson AGX but could be streamed to the edge or other parts of the 5G network. 81

List of tables

5.1 Results of the delay tests using `ping`. 63
5.2 Summary of transfer, bitrate, and jitter values for Sender and Receiver. . . . 65

Nomenclature

Roman Symbols

AAA Authentication, Authorization, and Accounting

AES Advanced Encryption Standard

AF Application Function

AI Artificial Intelligence

AMF Access and Mobility Management Function

AN Access Network

API Application Programming Interface

APN Access Point Name

ARP Address Resolution Protocol

AS Application Server

AT Attention (Command prefix in communications)

AUSF Authentication Server Function

AWS Amazon Web Services

BSC Barcelona Supercomputing Center

BSF Binding Support Function

CCTV Closed-Circuit Television

CLI Command Line Interface

CN Core Network

CO Central Office

CORD Central Office Re-architected as a Datacenter

COTS Commercial Off-The-Shelf

CP Control Plane

CSCF Call Session Control Function

CUPS Control and User Plane Separation

DHCP Dynamic Host Configuration Protocol

DNN Data Network Name

DNS Domain Name System

DSL Digital Subscriber Line

EIR Equipment Identity Register

eMBB enhanced Mobile Broadband

eMBMS enhanced Multimedia Broadcast Multicast Service

eNB evolved Node B

EPC Evolved Packet Core

ePDG evolved Packet Data Gateway

ETSI European Telecommunications Standards Institute

FDD Frequency Division Duplexing

gNB Next Generation NodeB

GPRS General Packet Radio Service

GPS Global Positioning System

GTP GPRS Tunneling Protocol

GUAMI Globally Unique AMF ID

HLR Home Location Register

HPC High Performance Computing

HSS Home Subscriber Server

HTTP Hypertext Transfer Protocol

HUD Heads-Up Display

ICMP Internet Control Message Protocol

IEEE Institute of Electrical and Electronics Engineers

IMS IP Multimedia Subsystem

IMSI International Mobile Subscriber Identity

IoT Internet of Things

IP Internet Protocol

ISIM IP Multimedia Services Identity Module

IT Information Technology

JSON JavaScript Object Notation

LAN Local Area Network

LTE Long Term Evolution

MAC Media Access Control

MCC Mobile Country Code

MCE MME Code Emulator

MEC Multi-access Edge Computing

MIMO Multiple Input Multiple Output

MME Mobility Management Entity

MMS Multimedia Messaging Service

mMTC massive Machine Type Communications

MNC Mobile Network Code

MQTT Message Queuing Telemetry Transport

MTU Maximum Transmission Unit

NAS Network Attached Storage

NAT Network Address Translation

NB Narrow Band

NE Network Elements

NF Network Function

NFV Network Functions Virtualization

NG New Generation

NR New Radio

NRF Network Repository Function

NSA Non-Standalone

NSI Network Slice Instance

NSSAI Network Slice Selection Assistance Information

NSSF Network Slice Selection Function

NTN Non-Terrestrial Network

NVMe Non-Volatile Memory Express

OAI OpenAirInterface

PCF Policy Control Function

PCI Physical Cell Identity

PCIe Peripheral Component Interconnect Express

PCRF Policy and Charging Rules Function

PDCP Packet Data Convergence Protocol

PDP Packet Data Protocol

PDU Protocol Data Unit

PFCP Packet Forwarding Control Protocol

PGW Packet Gateway

PGWC Packet Gateway Control

PGWU Packet Gateway User

PHY Physical Layer

PLMN Public Land Mobile Network

POP Point of Presence

PPP Point-to-Point Protocol

RA Radio Access

RAM Random Access Memory

RAN Radio Access Network

RAT Radio Access Technology

RF Radio Frequency

RLC Radio Link Control

RRC Radio Resource Control

RTOS Real-Time Operating System

RTP Real-time Transport Protocol

RTT Round Trip Time

SA Stand Alone

SBA Service-Based Architecture

SBI Service-Based Interface

SCP Service Communication Proxy

SCTP Stream Control Transmission Protocol

SD Secure Digital

SDAP Service Data Adaptation Protocol

SDK Software Development Kit

SDN Software Defined Networking

SDR Software Defined Radio

SDRAM Synchronous Dynamic Random Access Memory

SGW Serving Gateway

SGWC Serving Gateway Control

SGWU Serving Gateway User

SIM Subscriber Identity Module

SMF Session Management Function

SMS Short Message Service

SSL Secure Sockets Layer

SST Slice/Service Type

SUCI Subscription Concealed Identifier

SUPI Subscription Permanent Identifier

TAC Tracking Area Code

TAI Tracking Area Identity

TCP Transmission Control Protocol

TDD Time Division Duplex

TLS Transport Layer Security

TOPS Tera Operations Per Second

TS Technical Specification

UDM Unified Data Management

UDR Unified Data Repository

UE User Equipment

UMTS Universal Mobile Telecommunications System

UP User Plane

UPF User Plane Function

URLLC Ultra-Reliable Low Latency Communications

USIM Universal Subscriber Identity Module

VF Virtual Function

VLC Visible Light Communication

VM Virtual Machine

VNF Virtual Network Function

VoIP Voice over Internet Protocol

VoLTE Voice over LTE

VoNR Voice over New Radio

VPN Virtual Private Network

vRAN virtualized Radio Access Network

YAML Yet Another Markup Language

Chapter 1

Introduction

1.1 Context and hypothesis

The introduction of 5G technology and edge computing marks a significant paradigm shift in telecommunications, enhancing speed, connectivity, performance, and flexibility. This thesis is conducted at the Barcelona Supercomputing Center (BSC)[1] within the Predictable Parallel Computing (PPC) group[2]. It forms a part of the Horizon Europe Verge project[3] and the national PROXIMITY project[4]. These projects aim to enhance the capabilities and applications of 5G and edge computing across various real-world scenarios. The thesis specifically focuses on the transformative potential of 5G networks combined with edge computing technologies to reduce latency for delay-sensible applications using an experimental 5G testbed.

The core hypothesis of the thesis posits that processing data and hosting applications at the network's edge, near the point of origin, rather than relying on distant servers, significantly reduces data transmission paths. This reduction minimizes delays, offering substantial benefits for real-time applications, where even slight reductions in latency can dramatically improve performance and reliability. For instance, autonomous driving technology, which requires millisecond-level decision-making for safety, and telemedicine, including remote surgical procedures and real-time patient monitoring, stand to benefit immensely from reduced latency. In these scenarios, the ability to transmit and process crucial data like surgical commands or patient vitals almost instantaneously is indispensable, enabling medical professionals to make prompt and accurate decisions.

[1] https://www.bsc.es/es

[2] https://ppc.bsc.es/

[3] Horizon Europe research and innovation program under Grant Agreement 101096034. Online at: https://www.verge-project.eu/

[4] Proyectos de Generación de Conocimiento 2021 PID2021-124122OA-I00

The relevance of this research extends beyond these fields, potentially revolutionizing areas such as online gaming, where reducing lag is essential for competitive play, and smart city technologies, which depend on efficient traffic management and emergency response systems. These systems benefit from the immediate processing and action upon data from various sensors and devices, without the delays associated with traditional cloud processing. Additionally, big data and Internet of Things (IoT) applications, which require processing vast volumes of data, stand to gain from edge computing. Transferring such data to the cloud not only introduces latency due to data routing but also impacts energy consumption and the cost associated with transport networks. Hosting applications, processing and routing data at the edge offers a cheaper, more efficient, and secure alternative.

Despite the apparent advantages of combining 5G and edge computing, there is a gap in the availability of real 5G edge-enabled experimental platforms or testbeds equipped with real applications to evaluate these technologies in practice. This thesis addresses this gap by deploying a 5G testbed that demonstrates the substantial latency reductions achievable for delay-sensitive applications through the characteristic flexibility of 5G, coupled with the use of edge devices within the 5G network infrastructure. Furthermore, this project includes hands-on demonstrations within the testbed, implementing real applications that confirm the feasibility of hosting and routing their data at the edge within the 5G network. These practical examples serve as evidence that edge computing can indeed fulfill the promise of enhanced performance and efficiency in a 5G environment.

1.2 Scope, objectives and contributions

The primary scope of this project is to develop a 5G testbed with edge computing capabilities that enables us to test and validate the potential of these emerging technologies with metrics and real applications. Throughout the project, three applications are implemented on this network to show its effectiveness in supporting real-time applications and to illustrate the practicality of edge computing, edge routing and edge hosting together with 5G connectivity.

The project's objectives and contributions include:

1. Understand the 5G architectural concepts, deployments and key features such as its network functions, network slicing, Stand Alone (SA) and Non Stand Alone (NSA) deployments, etc.

2. Explore proprietary and open-source frameworks to deploy an end-to-end 5G testbed, from real end devices to the 5G core network (5G CN).

3. Extend the 5G testbed to be more flexible and to support edge computing, by adding the necessary modifications to enable routing data from the edge and hosting applications.

4. Collect network performance metrics, such as delay, throughput, and jitter, to assess and support the project's hypothesis.

5. Test the testbed's capabilities by deploying a variety of applications at the network edge, including a messaging service, a video streaming service, and a real-time object detection service.

1.3 Thesis structure

This first introductory section has set the research context, the hypothesis, the scope, the objectives and the contributions. The remaining part of the thesis is structured into six chapters, Chapter 2, offers a comprehensive literature review and background on edge computing and 5G, exploring their essential concepts, architectures, and the current state of research. The Methodology chapter, Chapter 3, details the frameworks, equipment, and performance metrics used, including the testbed architecture and application scenarios. Chapter 4, Testbed deployment, describes the practical aspects and challenges of setting up the testbed. Chapter 5, Metrics and performance evaluation, analyzes the performance based on key metrics such as delay, throughput, and jitter. Chapter 6, Testbed applications, illustrates the deployment of specific applications to demonstrate the functionality of the testbed and the integration of edge computing with cloud routing. The final chapter, Chapter 7, summarizes the conclusions and future work, highlighting the findings and suggesting avenues for further research. This organization ensures a coherent flow, gradually building up from foundational concepts to practical applications and their evaluation and finishing with conclusions and future work.

Chapter 2

Background and literature review

2.1 Overview of the 5G architecture

5G technology is a significant step forward in the wireless communications landscape, introducing improvements over its predecessors in terms of speed, connectivity, latency, and channel capacity. Unlike the previous generations, which primarily focused on enhancing the bandwidth and efficiency of communications, 5G is designed to meet the growing demands of interconnections of today's societies, supporting a vast array of devices and services like IoT, robotics, V2X, etc. 5G is also a shift towards more decentralized network architectures, moving data processing closer to the edge of the network. This edge computing approach minimizes delays, reduces the load on core networks, and supports the massive influx of data from IoT and other devices.

The evolution of mobile technologies has been characterized by the development and implementation of different generations, each with specific objectives of solving the drawbacks and the limitations of their predecessors and adapting to the demands of their period. Figure 2.1 provides a comprehensive overview of the evolution across different communication generations.

- **1G**: Introduced in the 1980s, 1G marked the beginning of mobile communication with analog systems that supported voice. Its focus was to provide basic mobile telephony communication.

- **2G**: Deployed in the early 1990s, 2G introduced the digitalization of mobile communications, enabling services such as SMS and MMS, in addition to improving call quality, efficiency and security of the established network.

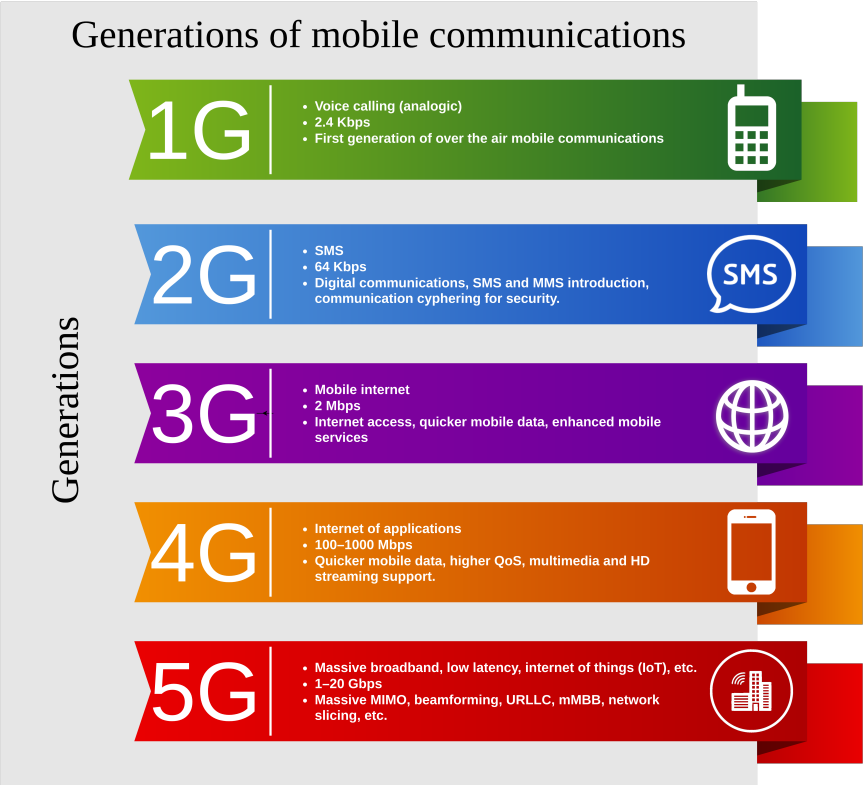


Fig. 2.1 Generations of mobile communications and its main features

- **3G**: Starting in the 2000s, 3G enabled higher data transmission speeds, facilitating mobile internet access and video calling services for greater connectivity and to offer multimedia services.
- **4G**: From 2010 onwards, 4G has offered even faster internet connection speeds, support for HD video, online gaming, and advanced communication services. The objective of 4G was to increase internet access speed to support high bandwidth applications.
- **5G**: Beginning in 2019, 5G started to be deployed with the goal of providing ultra-fast speeds, low latency, and the capacity to massively connect IoT devices, thus supporting a new era of smart applications.

Focusing on the 5G System (5GS), it introduces new concepts and technologies within its core network architecture, setting a new standard for mobile communication systems (described by 3GPP). Two key enabling concepts are Control/User Plane Separation (CUPS) and Network Slicing.

CUPS, with its roots in 4G LTE architecture and formally established in the 3GPP TS23.214 [7], has been put to work with other technologies (such as network slicing) to

create more efficient but also scalable and flexible networks. By decoupling control plane (CP) functions from those of the user plane (UP), CUPS facilitates the creation of agile networks[1].

Network Slicing expands the capabilities of 5G by allowing the configuration of multiple virtual networks, or "network slices," on a single physical infrastructure. Each slice is optimized for specific services or user groups, ensuring the core network can efficiently distribute resources for specific application needs, user groups, etc. [8] [9].

Building upon these foundational technologies, the 5G network architecture integrates additional innovations like Network Function Virtualization (NFV), Software-Defined Networking (SDN), and Cloud-Native Architecture. NFV transforms network functions into virtualized components, utilizing IT virtualization technologies. These components, or Virtual Network Functions (VNFs), are deployed on various software environments, such as virtual machines or containers, enhancing the network's adaptability and operational efficiency. SDN, on the other hand, introduces a programmable network architecture, where network intelligence is managed through software applications. This innovation centralizes and simplifies the control mechanism, making it reconfigurable and significantly more flexible. Lastly, Cloud-Native Architecture brings the principles of cloud computing into the mobile networking domain, especially within 5GS. It adopts a Service-Based Architecture (SBA), enabling scalable and robust interactions between network elements in a manner reminiscent of large-scale web applications [8] [9]. This architecture is built around the idea of using cloud principles—such as microservices, containers, and dynamic management—to ensure that network functions and applications are as resilient, flexible, and scalable as those found in the cloud. By breaking down traditional monolithic network functions into smaller, independent processes, Cloud-Native Architecture allows for more rapid development, deployment, and maintenance of network services, significantly enhancing the agility and efficiency of 5G networks.

These technologies aim to augment the flexibility, agility, and scalability of the 5G core network. Within this context, 5G introduces three distinct service categories based on the concept of network slicing: Enhanced Mobile Broadband (eMBB), Massive Machine Type Communication (mMTC), and Ultra-Reliable and Low Latency Communications (URLLC) [10].

The eMBB service slice is tailored for end-users, offering high-speed mobile connectivity with downlink speeds up to 1Gbps indoors and 300Mbps outdoors in densely populated areas.

[1] "Agile networks", refer to a network design and operation concept that emphasizes flexibility, scalability, and adaptability. These networks are engineered to easily introduce changes in demand, configuration, or technology, enabling operators and organizations to efficiently respond to shifting market needs or business priorities.

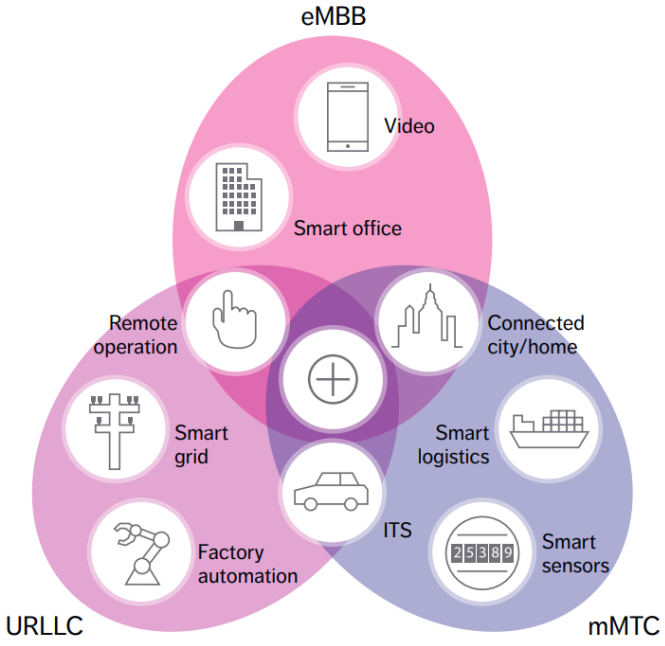


Fig. 2.2 Service categories in 5G [1]

This slice supports bandwidth-intensive activities such as web browsing, video streaming, and messaging applications. In contrast, the mMTC is designed for IoT applications, providing efficient connectivity for a vast number of devices, such as sensors and actuators. This slice facilitates the deployment of Machine to Machine (M2M) communications, enabling smart city initiatives and industrial automation. Finally, the URLLC slice is dedicated to mission-critical applications, offering high reliability and ultra-low latency essential for applications requiring immediate responses, such as factory automation, V2X communication, and remote surgical procedures.

Summing up, 5G is a technology that not only gives advances on internet speed, it also signifies a shift to agile networks. This means flexibility, efficiency and scalability which leaves the door open to designing tailored applications and deployment based on the needs of the society.

2.1.1 Core Network (CN)

There are two ways of describing the core network of 5G depending on the focus of the organization or communication of the microservices on the network. These two different representation models are Reference Point architecture and Service Based Architecture (SBA) [8].

On the Reference Point architecture representation model, the representation of the 5G network involves using a set of Network Elements (NE) and interfaces (point-to-point) that interconnect the Network Elements. Interactions between NEs on the Control Plane of the network are called Signaling Procedures, and they are specific for each of the point-to-point interfaces. Figure 2.3 illustrates the Reference Point architecture schematic.

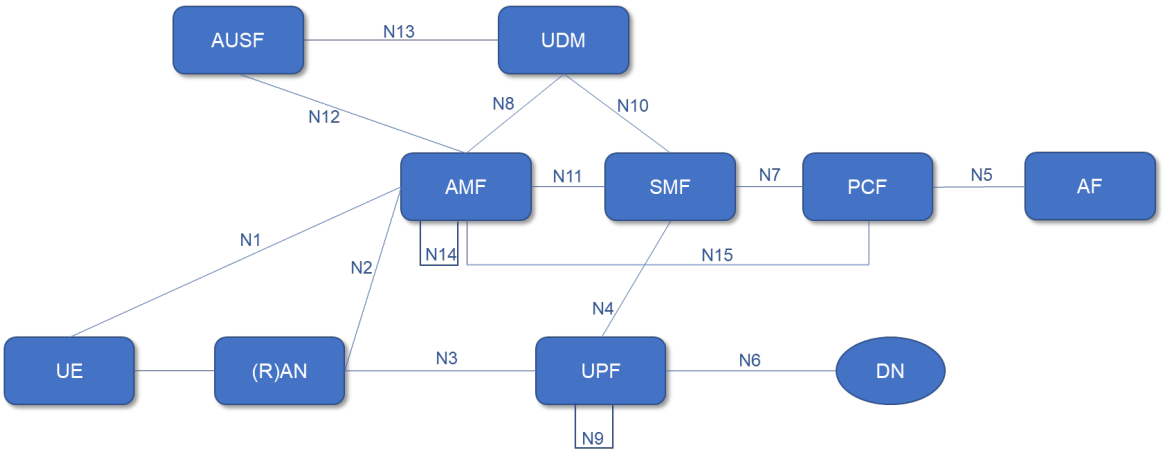


Fig. 2.3 Reference Point architecture representation [2]

Figure 2.3 represents the non roaming case, where the UE stays inside its native network. This is why there is only one UPF and the N9 interface is self connected. On a real network there are UPF handovers and the N9 interface has to communicate UPFs between them.

On the SBA representation, Network Functions (NF) are used instead of NEs and the point-to-point interfaces are replaced with a common bus, enabling the NFs to use the services provided by other NFs. Figure 2.4 represents the 5G network in an SBA oriented way.

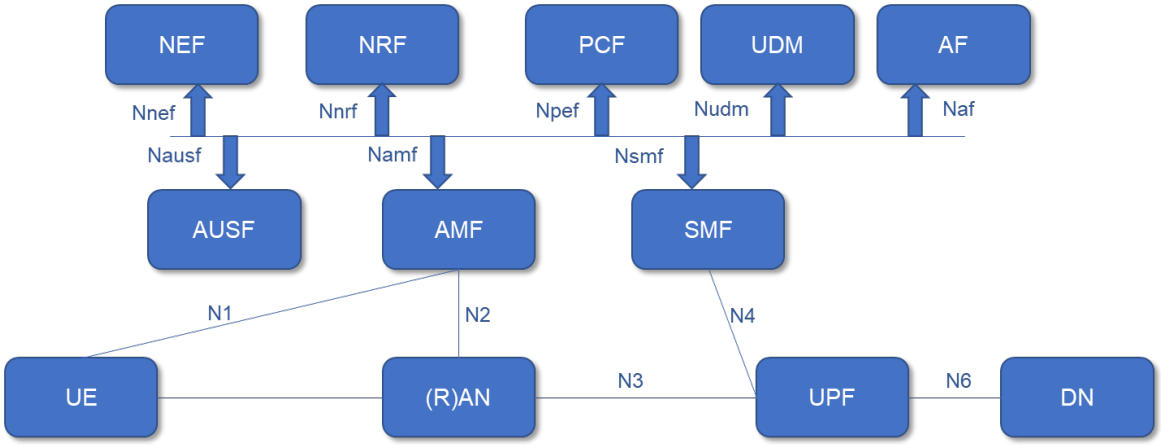


Fig. 2.4 Service Based Architecture representation [2]

The most relevant NFs of the 5GC and its related reference points or SBI interfaces can be described as follows:

- **Access and Mobility Management Function (AMF):** handles all signaling messages between the NG-RAN (Next Generation Radio Access Network) and the 5G Core Network (CN). It primarily deals with user access and mobility procedures. The AMF coordinates authentication and authorization in collaboration with the AUSF (Authentication Server Function) and the UDM (Unified Data Management). During Session Management procedures, it forwards the user equipment (UE) request/response to the SMF (Session Management Function). However, it does not actively manage the User Plane.

  The related reference points and the service-based interface for the AMF include:

  - The N1 reference point is for all control messages exchanged with the UE.
  - The N2 reference point is for all control messages exchanged with the gNB (gNodeB).
  - The N8 reference point is used to collect user subscription data from the UDM during the registration process.
  - The N11 reference point is for session control messages with the SMF.
  - The N12 reference point is for user authentication during the registration process.
  - The N15 reference point is for policy control messages with the PCF (Policy Control Function).
  - The Namf service-based interface allows access to services provided by the AMF.

- **Authentication Server Function (AUSF):** Manages authentication procedures and security protection for the network. The service requires collecting information from the UE and the UDM to provide security parameters in the UE Registration procedure.

- **Network Repository Function (NRF):** serves as a central repository for Service Consumers [2] within the core network, accessible via the Nnrf service-based interface. Service Consumers, like the AMF, need only initial configuration with the necessary parameters to locate the NRF. Once connected, the Service Consumer can gather information about other Network Functions (NFs) from the NRF. This repository maintains and supplies data related to NFs, including details like the NF Type, the

[2] Service Consumers are all the functions that are consuming services from other Service Producers that offer them.

services an NF supports, and the IP address necessary for discovery and selection processes.

- **Unified Data Repository (UDR):** The UDR serves as a database for the subscription data of the UEs over the Nudr service–based interface. It also stores user policies and network policies. The UDM and the PCF access consume the services of UDR to access this data.
- **Unified Data Management (UDM):** UDM gives its services using the Nudm service-based interface The UDM services are:
  - The execution of UE authorization for access to the DNN.
  - The generation of the authentication credentials that are transferred to the AUSF.
  - The management of permanent identity privacy. NFs consume the UDM service to resolve the concealed permanent identity (SUCI) to the real permanent identity (SUPI).
  - To keep track of the AMF instances for related UE.
  - To keep track of the SMF instances for related PDU Sessions.
- **Policy Control Function (PCF):** The Npcf is the service–based interface of the PCF. The PCF produces the service which enables to control of UE policies in the Session Management Procedure. The PCF provides information for UE access selection or PDU Session Selection procedures.
- **Application Function (AF):** Represents applications that influence traffic routing, detect session information, or have the need to access network exposure services.
- **Network Slice Selection Function (NSSF):** The NSSF service provides the necessary information to the Service Consumer to select the (set of) network slice instances for a particular UE. Besides, it gives information about AMFs that should serve the UE. The Nnssf is the service–based interface of the NSSF.
- **Session Management Function (SMF):** SMF is responsible for all procedures related to Session Management. These functionalities include the establishment of a connection between the User Equipment (UE) and the Data Network (DN), as well as the management of the User Plane for that connection. Upon receiving a PDU Session Request from the AMF, the SMF initiates the Session Establishment process in collaboration with the Unified Data Management (UDM) and the Policy Control

Function (PCF). Additionally, the SMF configures the User Plane Function (UPF) with rules for handling User Plane Data, which are communicated via Packet Forwarding Control Protocol (PFCP) control messages. Interaction with the UPF is facilitated through the N4 Reference Point, which is used by the SMF to manage UPF functions, including UPF discovery and selection.

The SMF's services, accessible to other NFs via the Nsmf service-based interface, include:

- Controlling user sessions, encompassing the establishment, modification, and release of sessions.
- Allocating IP addresses to UEs for IP PDU sessions.
- Managing service and session continuity across different end-user protocol types.
- Configuring the UPF for a PDU session, including policy setting in coordination with the PCF.

- **User Plane Function (UPF):** Its integration into the network's user plane path can be either predefined or dynamically adjusted by the SMF, which may also reassign the UPF for active PDU Sessions as needed.

  Following the initiation of the Session Establishment process, the UPF receives directives from the SMF. These directives comprise a series of rules and parameters aimed at guiding the establishment of the session. The UPF's core responsibility involves the management of user plane data in alignment with the specifications conveyed through PFCP (Packet Forwarding Control Protocol) packets. This management is broadly categorized under Packet Processing activities, which can be adapted both prior to and following the session's establishment. The specific tasks assigned to the UPF can vary, reflecting its designated role within the network.

  In certain scenarios, the UPF functions as a firewall, implementing drop instructions from the SMF, whereas in others, it acts as a conduit for data, managing the traffic flow between uplink and downlink packets. The UPF's positioning is also tailored to the service requirements, with potential placements near the UE to facilitate low-latency connections, using Cloud Computing technologies. Both the functionalities and the location of the UPF are considered when the SMF makes its selection. The primary functions of the UPF, as identified, include:

  - Serving as an anchor point for mobility.

- Acting as the external PDU Session point for connecting to the Data Network (i.e., N6).
- Facilitating packet routing and forwarding.
- Conducting packet inspection, such as application detection.
- Implementing the upper part of policy rule enforcement, including gating, redirection, and traffic steering.
- Performing lawful intercept (UP collection).
- Reporting on traffic usage.
- Managing QoS for the User Plane.
- Verifying uplink traffic.
- Marking packets at the transport level in both uplink and downlink directions.
- Buffering downlink packets and triggering downlink data notifications.
- Sending and forwarding end markers to the source NG-RAN node.
- Handling ARP and IPv6 neighbor discovery requests for Ethernet PDUs.

It is important to highlight the usage of the GPRS Tunneling Protocol (GTP) across interfaces N3 and N9, seen in Figure 2.3. The N3 interface establishes a connection between the gNB and the UPF, employing GTP-U (GTP-User plane) to encapsulate and transport user plane data. Similarly, the N9 interface, uses GTP-U to enable communication between two UPFs which is crucial for the transport of user data in scenarios necessitating UPF handovers or internal data routing, particularly during roaming situations. Additionally, for control plane signaling, which orchestrates session management and mobility between network components, 5G uses GTP-C (GTP-Control plane). GTP-C is mainly utilized between the SMF and the NRF for session management. GTP has a fundamental role across these interfaces, enabling the segregation of data packets from the underlying transport network. This separation allows for adaptable routing, efficient mobility management, and uninterrupted session continuity across the cellular network. This is vital for achieving interoperability among equipment from various vendors and ensuring a uniform user experience across mobile networks.

2.1.2 Access Network (AN)

The 5G Next Generation Radio Access Network (NG-RAN) is illustrated in Figure 2.5. Here, the over the air interface is referred to as NG-RA, and it establishes the communication path

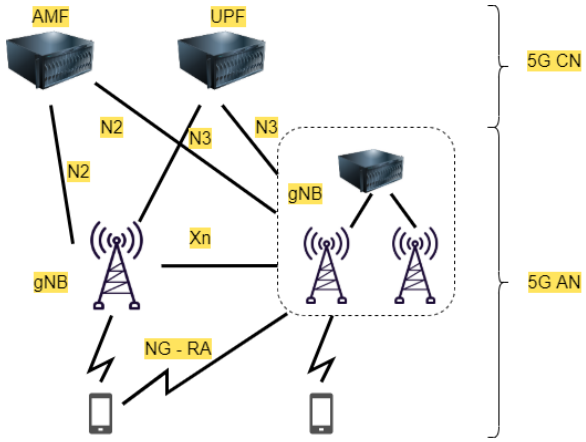


Fig. 2.5 NG-RAN architecture [3]

between the User Equipment (UE) and the gNodeB (gNB). Within this interface, the carriers of radio resources are known as radio bearers. The gNBs are interconnected through the Xn interface, which facilitates the exchange of control messages between them, essential for coordinated operations such as packet duplication and elimination as dictated by the PDCP protocol or handover between gNBs. Additionally, NG-RAN network elements possess interfaces for interacting with the AMF in the Core Network (CN), primarily for the exchange of control messages related to any service within the 5G system. The AMF communicates with the UE via the N1 interface and with the gNB via the N2 interface. Furthermore, the N4 interface establishes a link between the UE and the UPF for the transmission of application data packets (both uplink and downlink).

With the adoption of CUPS, the control and user plane protocols diverge. The user plane protocol stack for NR includes the Radio Link Control (RLC), Packet Data Convergence Protocol (PDCP), and the Service Data Adaptation Protocol (SDAP), with an additional Access Stratum (AS) layer employed above the PDCP to manage SDAP packets. The RLC layer is tasked with error correction, Maximum Transmission Unit (MTU) segmentation and resegmentation, as well as the transfer of payloads to upper layers. The PDCP layer facilitates user data transfer, header compression, and packet handling (duplication/elimination), integrating control sequence numbering with the RLC. The SDAP layer aligns data radio bearers with the required QoS.

The Physical (PHY) and Media Access Control (MAC) layers remain consistent across both user and control planes, with the former managing air transmission and the latter allocating logical channels to transport channels, alongside multiplexing and demultiplexing of RLC Protocol Data Units (PDUs). The Radio Resource Control (RRC) protocol, which administers radio resource management through the establishment, modification, and release

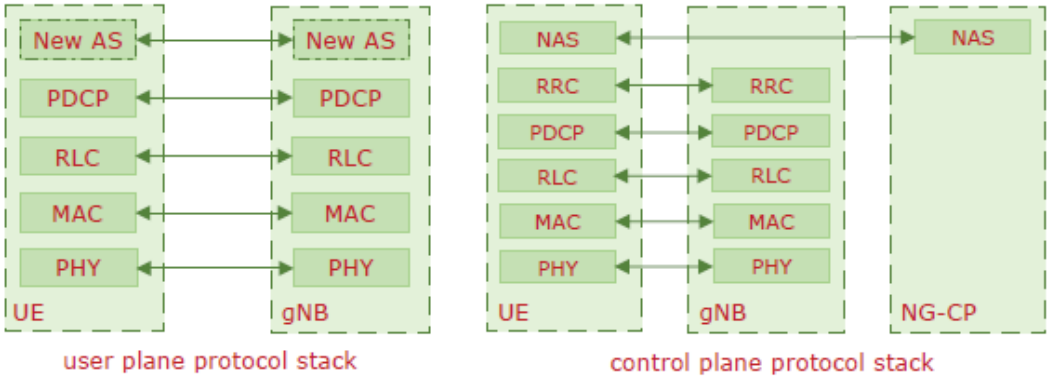


Fig. 2.6 UP and CP protocol stacks in NR [4]

of radio bearers, encapsulates RRC control messages within the PDCP header. In contrast, the Non-Access Stratum (NAS) serves as a functional layer housing protocols essential to the 5G System's operations, such as the 5GS Mobility Management (5GMM) and the 5GS Session Management (5GSM), facilitating control message exchanges between the UE and the 5G CN.

2.1.3 Deployment Options

During the 72nd general meeting of the 3rd Generation Partnership Project (3GPP), several options for the 5G network architecture were proposed. These options can be divided in two main types:

- **Non-Standalone Deployment (NSA):** NSA mode, is defined by its reliance on existing 4G LTE networks. It leverages the existing 4G infrastructure (Enhanced Packet Core or EPC) alongside new 5G NR (New Radio) technologies. This approach allows for faster deployment of 5G services and provides immediate enhancements to network capacity and speed, but it does not fully utilize the 5G core network's capabilities.
- **Standalone Deployment (SA):** SA mode refers to 5G networks that operate independently of existing 4G networks. It utilizes its own 5G core network (5G CN) and New Radio (5G NR), enabling all the features and capabilities of the new 5G technology, including improved efficiency, lower latency, and the ability to support new applications and services such as massive machine type communication (mMTC) and ultra-reliable low latency communication (URLLC).

TR38.912 [11], describes various deployment options, categorized under Non-Standalone (NSA) and Standalone (SA) architectures, to address the diverse needs of network operators,

ranging from using existing infrastructure to deploying entirely new network cores for 5G services. The purpose of these options is to facilitate a flexible, phased transition towards 5G, allowing for both immediate enhancements to existing 4G networks and the eventual realization of fully standalone 5G networks.

The deployment options discussed by 3GPP offer a spectrum of choices for operators. Options 1 and 2 are focused on SA architectures, which do not rely on existing LTE networks and are designed to fully exploit the capabilities of 5G by using the New Radio (NR) access technology in conjunction with the 5G CN. These options are envisioned to support the full suite of 5G features, enabling innovative applications and services that require high bandwidth, low latency, and massive connectivity.

On the other hand, Options 3/3a/3x introduce NSA architectures, enabling operators to deploy 5G services more rapidly by utilizing existing LTE networks as an anchor for mobility management and coverage, with 5G NR serving as a secondary radio access technology. These options are particularly attractive for operators seeking to boost network capacity and speeds while preparing for a future transition to a fully standalone 5G network.

Options 4/4a further refine the NSA approach by transitioning towards a more 5G-dominant operation, creating a mid-point in the evolution from NSA to SA deployments. Similarly, Option 7/7a/7x extends the concept of dual connectivity found in NSA deployments, using LTE as the primary connection for control information while using the speed capabilities of 5G NR as the secondary radio access technology.

Finally, Option 5 is an SA architecture that introduces advanced capabilities in the 5G Core, enhancing network slicing and QoS management while using 5G NR access. This option, along with Options 1 and 2, signifies the end goal for many operators: a fully standalone 5G network that leverages all the technological advancements 5G has to offer.

As the telecommunications industry moves forward, it is the SA architecture options (specifically Options 1, 2, and 5) that represent the ultimate goal for the deployment of 5G networks. These options facilitate the realization of the full potential of 5G, from high-speed data services to IoT applications, by enabling operators to deploy networks that are not constrained by the limitations of existing LTE infrastructure.

2.2 Edge computing in the context of 5G

2.2.1 The edge computing concept and its importance in 5G

The definition that the Linux Foundation gives to the term *edge computing* in its Open Glossary[3] is the following:

The delivery of computing capabilities to the logical extremes of a network in order to improve the performance, operating cost and reliability of applications and services. By shortening the distance between devices and the cloud resources that serve them, and also reducing network hops, edge computing mitigates the latency and bandwidth constraints of today's Internet, ushering in new classes of applications. In practical terms, this means distributing new resources and software stacks along the path between today's centralized data centers and the increasingly large number of devices in the field, concentrated, in particular, but not exclusively, in close proximity to the last mile network, on both the infrastructure and device sides.

Managing expanding datasets, particularly in the era of AI and Big Data's growth since the 90s, is challenging. The increasing size of datasets has made data mobility progressively difficult, thus increasing the distance between applications on smartphones and services in the cloud or operator's core networks. This separation often results in latencies exceeding 50-100 ms, due to data traversing through various networking entities, leading to potential congestion and the inability to guarantee QoS or throughput, which has even more impact on delay-sensible applications.

In this context, edge computing is a crucial solution to reduce the distance between users that consume data-driven applications and services, ensuring the required latencies and throughputs for modern applications, a need that has become particularly evident with the digitization of sectors such as Industry 4.0, collaborative and automated driving, and e-health among others [12].

The importance of edge computing in 5G networks can be explained with four critical requirements that ensure its successful deployment and operation. Each of these requirements must be balanced based on specific application needs.

- **Real-time interaction:** The primary advantage of edge computing over traditional cloud computing is its ability to provide low latency, which is essential for delay-sensitive URLLC applications such as remote surgery, tactile internet, and unmanned vehicle operations. Edge servers can process data and make decisions in real-time, significantly enhancing the QoS.

[3] https://github.com/State-of-the-Edge/glossary/tree/master

- **Local processing:** Edge computing enables data and user requests to be processed locally at the edge servers instead of being sent to the cloud. This reduces the volume of traffic between small cells and the core network, which can enhance bandwidth and reduce congestion in the core network.
- **High data rate:** To handle the substantial data volumes generated by applications like virtual reality and remote surgery, edge computing uses high data rates facilitated by mmWave frequency bands. This allows edge servers, embedded within base stations, to transmit large amounts of data efficiently without relying on the core network.
- **High availability:** Ensuring the availability of services at the edge is crucial, as edge computing decentralizes data processing and application logic. The reliability of edge clouds is vital for the continuous operation of 5G services, especially in environments requiring consistent and uninterrupted connectivity.

2.2.2 Where is the edge of the network?

Edge computing does not have a prescribed location or specific set of locations for deployment. It encompasses incorporating edge nodes within various elements of the network infrastructure such as network routers, cellular or radio towers, Wi-Fi hotspots, DSL boxes, and local data centers. As outlined in Section 2.2.1, edge computing aims to position computational resources nearer to the end-users. Essentially, any device with computing capabilities and situated close to or at the user's location qualifies as an edge computing device, provided it has the capacity to handle computational workloads.

The typical placement of edge computing falls between the user's devices and centralized computing data centers, including both public clouds and telecommunications cloud facilities. Managing the computing resources on devices is a challenge due to their heterogeneity and their connection to network environments, usually local area networks (LANs).

Examples of edge computing deployments, illustrating the diversity of locations, include:

- **On premise:** Corporations implementing 4G/5G Private Networks might deploy a complete Network Core within their premises, integrating it with business applications.
- **RAN/Base Station:** Some entities install infrastructure collocated with RAN in urban areas, utilizing cabinets or mini datacenters.
- **Central Offices (COs):** Positioned at the network edge of Cloud Service Providers (CSPs), COs act as aggregation points for both fixed and mobile traffic from end-users.

An ongoing transformation seeks to mitigate issues on some of these locations by deploying a virtualized, distributed network at the edge. Initiatives such as Central Office Re-Architected as a Datacenter (CORD)[4] and projects under Open Platform for NFV (OPNFV) have commenced a process where data center economies and the flexibility of SDN, combined with cloud design and network disaggregation principles, aim to address these challenges. These technological advancements facilitate the deployment of edge computing in various environments, enhancing connectivity, processing capabilities, and response times some of these changing technologies are:

- **Private datacenters:** Telecommunications companies and others are setting up private datacenters to accommodate edge computing infrastructure, requiring interconnection with Mobile Network Aggregation Points of Presence (POP) for user traffic.
- **Hyperscalers edge locations:** Public cloud firms define their edge locations, such as AWS Cloudfront[5] by Amazon, typically established in a few physical sites per country in Europe.
- **Industrial IoT hubs:** In industries such as manufacturing and logistics, edge computing is implemented at IoT hubs, directly on factory floors or within logistics centers, to process data from IoT devices in real-time.
- **Smart city infrastructure:** Edge computing is also deployed within infrastructure elements like traffic lights and CCTV networks in smart cities, enabling local data processing and faster response times.

While edge locations can vary, they are not mutually exclusive, and a network deployment may utilize multiple edge locations [12].

The data of the 5G services is processed in the cloud or the edge, depending on its kind. There are different kinds of data in 5G services:

- **Hard real-time data** demands strict latency adherence.
- **Soft real-time data** allows for minimal latency deviations.
- **Non-real-time data** tolerates delayed processing.

In case of services that have hard real-time requirements, the processing of the data will happen at the edge which is close to UEs and this substantially decreases latency. In cases

[4]https://opennetworking.org/cord/

[5]https://docs.aws.amazon.com/en_us/AmazonCloudFront/latest/DeveloperGuide/Introduction.html

where applications and services require soft-real time data processing, with a permissible bounded delay, edge servers take on the tasks if the delay between the UEs and the cloud exceeds these bounds, if not, these tasks may be shifted to the cloud. Finally, for services that do not require strict or soft real-time processing, typically the workload is offloaded to the cloud.

2.3 State of the art

The integration of edge computing and cloud-native technologies in 5G networks has been a focus of substantial research, aiming to use the benefits of distributed computing and network agility. Existing studies, such as those cited by [13] and [14], focus on broad concepts such as the integration of emerging technologies, architectural developments and deployment models in edge computing, research in network slicing and orchestration/scheduling, performance evaluation and metrics, standards such as MEC [15] and describe the challenges and open research areas. As remarked in [14], *"as an emerging technology concept, edge computing has not yet fully matured, and there is confusion in the industry about the understanding of cloud-native edge computing and MEC concepts"*. This made it difficult to find research works specifically focused on the scope of this project which, as highlighted in the introduction of this thesis, is to develop a 5G testbed incorporating an edge computing slice. This setup facilitates various tests, including hosting applications, routing requests, and processing data directly at the edge. The focus of these tests is to demonstrate the potential for delay reduction achievable through this approach. It is specifically hard to find works that describe practical approaches on testbed deployment or employing real applications and services.

While practical implementations using real testbeds are notably scarce, some proposals can be found. Cao et al. [16] explore an experimental setup utilizing Kubernetes[6] and KubeEdge[7] using Free5GC [8] to deploy a 5G control plane and user plane, with user plane

[6]Kubernetes, also known as K8s, is an open-source platform designed to automate deploying, scaling, and operating application containers. Developed by Google and now maintained by the Cloud Native Computing Foundation, it helps manage containerized applications across a cluster of machines, providing tools for deploying applications, scaling them as necessary, and managing changes to existing containerized applications. Kubernetes simplifies container orchestration, enhances scalability, and maintains reliability and efficiency in cloud, on-premises, and hybrid environments. More information can be found on their official website: https://kubernetes.io/.

[7]KubeEdge is an open-source system aimed at extending native containerized application orchestration capabilities to hosts at the Edge. It is built on Kubernetes and provides core infrastructure support for networking, application deployment, and metadata synchronization between the cloud and edge. It is designed to address the network reliability, bandwidth limitation, and resource constraints of edge computing environments. More information can be found on their official website: https://kubeedge.io/.

[8]free5GC is an open-source project implementing the 3rd Generation Partnership Project (3GPP) 5G Core (5GC) network specifications. It aims to support the 5G standalone architecture's features and enable rapid

functions at the edge. However, their approach does not involve real UEs or gNBs, nor does it include hosting applications on the edge. Similarly, Vázquez et al. [17] propose a cloud-native platform utilizing Kubernetes and Open5GS, with user plane functions at the edge. Yet, their model also relies on a simulated RAN environment using UERANSIM, without any application deployment at the edge. These studies illustrate a trend where research deploying testbeds or platforms typically simulate the RAN environment and do not deploy real applications, thereby limiting the scope for real-world use cases.

The research conducted by Barrachina et al. in [18] and [19] has been a reference for constructing the testbed for this project, since it is the closest approach to the objective of this thesis. The authors, both from the same research group, identify a gap in the existing literature. While numerous studies explore cloud-native deployments and monitoring systems, few address both elements in the context of MEC-enabled 5G networks comprehensively. They propose a unified experimental platform to bridge this gap, using two different approaches. Specifically, in [18], this platform integrates an open-source 5G core network deployed within a Kubernetes cluster, with the RAN implemented using both physical components (featuring the AMARI Callbox[9] as a standalone 5G gNB for over-the-air transmissions and the AMARI UE Simbox for user equipment) and simulated environments. The paper primarily showcases infrastructure and monitoring capabilities without focusing on specific end-user applications or achieving low latency. In [19], the focus shifts to a disaggregated approach to network function deployment, enabling dynamic, on-demand deployment and scaling of UPFs across multiple edge locations, thereby enhancing the adaptability and responsiveness of the network to changing demands. They use a double gNB and double UE approach using real RAN equipment from Amarisoft, but the UEs are simulated as in [18]. The paper, again, does not talk about real delay-sensible applications or the impact of edge 5G dataplanes in reducing delay.

This project goes beyond the existing state of the art by extending the exploration of practical 5G experimentation using testbeds. It not only researches and develops the infrastructure and demonstrates lower delay using metrics but also assesses the feasibility of deploying delay-sensitive services at the edge of the 5G network. Unlike the practical approaches detailed in this section, which primarily use simulated UEs and gNBs, this thesis project deploys real applications and utilizes commercial off-the-shelf (COTS) real UEs and gNBs, thus moving closer to practical implementation and real-world testing of cloud-native

prototyping and testing of 5G innovations. free5GC provides a flexible and scalable platform for the deployment of a complete 5GC system, facilitating the development of new 5G services and applications. More information can be found on their official website: https://www.free5gc.org/.

[9] AMARI Callbox will be detailed in the Methodology chapter, specifically in section 3.3

5G systems. Furthermore, real-time applications are deployed in the edge of the network to further contribute to the topic.

Chapter 3

Methodology

3.1 Introduction to the methodology

The methodology employed throughout this project is outlined in a sequential and structured manner below, acting as a prelude to the chapter where various critical concepts, technologies, and approaches are explained. These foundations are essential for comprehending the subsequent chapters. The following steps have been followed for the implementation of this project:

1. Selection and evaluation of the software frameworks necessary for constructing the 5G testbed, as discussed in Section 3.2.

2. Definition of the hardware outlined in Section 3.3 to be be utilized in conjunction with the software frameworks for the testbed deployment.

3. Integration of the specified frameworks and hardware to develop a testbed following the architecture described in Section 3.4.

4. Establishment of performance indicators that will substantiate the thesis's hypothesis, detailed in Section 3.5.

5. Implementation, deployment and testing of real applications to evaluate the feasibility of edge hosting, as explained in Section 3.6.

3.2 Employed frameworks

3.2.1 Open5GS

Open5GS is a software-based platform designed to deploy and operate core network functions of 4G LTE and 5G networks. It offers support for both the 4G/5G NSA and 5G SA Core network architectures. These two architectures and the CUPS can be seen in Figure 3.1. Notice, as explained in Section 2.1, the CUPS is a crucial trait of 5G and is also visible in 4G architectures.

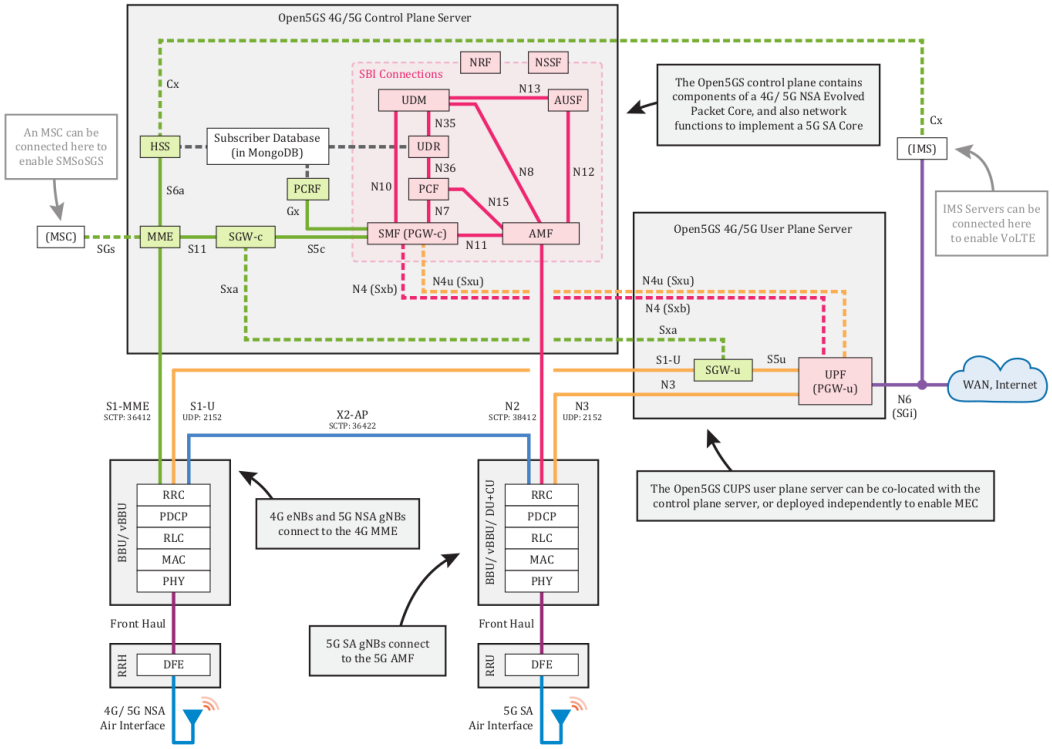


Fig. 3.1 Open5GS Core diagram [5]

The Open5GS platform is divided into two principal segments: the 4G/5G NSA Core and the 5G SA Core. Each segment is equipped with specific network functions and components designed to handle different aspects of network operations, from user authentication to data packet routing.

In the 4G/5G NSA Core architecture, the network's control and user plane functions are distinctly separated to optimize the routing of data packets and the management of network signals. Key components of this architecture include:

- **Mobility Management Entity (MME):** Serves as the central control node, managing user sessions, mobility, and paging.

- **Home Subscriber Server (HSS):** Acts as the database for user information, handling authentication and maintaining subscriber profiles.
- **Policy and Charging Rules Function (PCRF):** Responsible for policy enforcement and charging rules for data services.
- **Serving Gateway Control Plane (SGWC) and Serving Gateway User Plane (SGWU):** Facilitate the routing and forwarding of user data packets.
- **Gateway Control Plane (PGWC) and Gateway User Plane (PGWU):** Provide connectivity to external networks and handle user IP address allocation.

On the other hand, the SA option has the following daemons that simulate 5G services:

- **AMF Daemon – amfd:** Simulates some functionalities of the AMF, providing the N2 interface towards the eNB and Service Base Interface (SBI) for interactions with other NFs in the 5G CN.
- **SMF Daemon – smfd:** Simulates some functionalities of the SMF, offering the N11 interface towards the AMF and the N4 interface towards the UPF, alongside SBI for further NF interactions.
- **UPF Daemon – upfd:** Replicates some UPF functionalities by providing the N11 interface towards the SMF and the N3 interface towards the gNB.
- **NRF Daemon – nrfd:** Mimics NRF functionalities with SBI for NF communications within the 5G CN.
- **AUSF Daemon – ausfd:** Simulates AUSF functionalities, engaging in interactions with other NFs through SBI.
- **UDM Daemon – udmd:** Emulates certain functionalities of the AUSF, utilizing SBI for NF communications.
- **PCF Daemon – pcfd:** Simulates PCF functionalities, providing SBI for interactions within the 5G CN.
- **NSSF Daemon – nssfd:** Imitates NSSF functionalities with SBI for NF communications.
- **UDR Daemon – udrd:** Simulates UDR functionalities, employing SBI for interactions with other NFs.

- **BSF** [1] **Daemon – bsfd:** Replicates BSF functionalities for the system.

How to install and configure the Open5GS SA segment requires detailed knowledge on the configuration files, installation and execution of the daemons. This detailed explanation is given in the testbed deployment guide in section 4.2.

3.2.2 UERANSIM

UERANSIM [20] is an open-source 5G UE and RAN simulator that supports the simulation of 5G network protocols and procedures. UERANSIM enables testing and experimentation with the 5G NSA and SA modes, providing a comprehensive toolset for the simulation of 5G NR cells and UE devices. This simulator is particularly useful for network researchers, developers working on 5G network equipment, and anyone interested in learning about the operational aspects of 5G networks. UERANSIM supports various features including, but not limited to, NR RRC, PDU sessions, and NAS protocols, making it a versatile tool for 5G network simulation and analysis. Once built, UERANSIM has three principal binaries:

- **nr-gnb:** This binary simulates a 5G gNB, the base station in 5G networks. It is responsible for handling radio communications between the network and user equipment (UE), including signal transmission, reception, and various radio resource management tasks.
- **nr-ue:** The nr-ue executable simulates a 5G user equipment (UE), such as a smartphone or any other device capable of connecting to a 5G network. It covers functionalities related to network access, mobility management, session management, and communication with the gNB.
- **nr-cli:** This Command Line Interface (CLI) tool provides users with the ability to interact with the nr-gnb and nr-ue simulations. It allows for the configuration, control, and monitoring of both the gNB and UE components during simulation runs.

This framework has been used only for the first deployments of this project as a tool to learn how the 5G core works. Later, Amarisoft Callbox was used as gNB and a commercial phone and a Quectel modem as UEs. The detailed explanation of the configuration files and deployment is given in section 4.1.

[1] BSF refers to the Binding Support Function, which is a support function of Open5GS used for policy control and charging within the network's architecture. It manages application detection, policy enforcement, and the binding of user sessions to specific policies via the Diameter protocol. BSF in Open5GS primarily interacts with components like PCRF to ensure appropriate policy and charging rules are applied to user sessions.

3.2.3 Docker and Docker Compose

Docker is an open-source platform designed to simplify the process of developing, shipping, and running applications by using containerization technology. Containers allow developers to package an application with all its dependencies into a standardized unit for software development. Docker containers are lightweight, standalone, executable packages that ensure software runs reliably when moved from one computing environment to another. More information about Docker can be found on its official website[2].

The main components of Docker include:

- **Docker Engine:** The core service that builds and runs containers using Docker's components and services.
- **Docker Images:** Read-only templates used to build containers. Images are built from a Dockerfile, which is a script of instructions that automatically builds a given image.
- **Docker Containers:** Runnable instances of Docker images that encapsulate the application and its environment at a point in time.
- **Dockerfile:** A text document that contains all the commands a user could call on the command line to assemble an image.
- **Docker Hub:** A service provided by Docker for finding and sharing container images with your team. It is the world's largest library and community for container images.

Docker Compose, on the other hand, is a tool for defining and running multi-container Docker applications. With Compose, a YAML file is used to configure the application's services, networks, and volumes. Then, with a single command, Docker Compose is able to create and start all the components from the configuration. This makes Docker Compose particularly useful for deploying multi-container applications or microservices. The official documentation and more details can be found at the Docker Compose website[3].

Both Docker and Docker Compose enhance CI/CD pipelines by allowing developers to quickly build and test isolated changes to code or system configurations. This isolation helps in reducing conflicts between running applications and their environments, ensuring more reliable deployments.

[2] https://www.docker.com/
[3] https://docs.docker.com/compose/

3.2.4 Jetson Inference

The *jetson-inference* GitHub repository is a resource for deploying deep-learning models on NVIDIA Jetson platforms, utilizing NVIDIA's TensorRT for optimized neural network execution. It supports various deep neural network (DNN) vision primitives such as image classification (*imageNet*), object detection (*detectNet*), semantic segmentation (*segNet*), pose estimation (*poseNet*), and action recognition (*actionNet*). These models are compatible with multiple NVIDIA Jetson devices, enhancing the library's versatility for embedded AI applications. This resource is used in the third real case scenario (Section 6.3) of this thesis, to manage streaming fluxes between UE and edge and applying AI inference on top of these streamings.

The repository provides multiple modes for vision tasks:

- Image Classification - Categorizing images into predefined groups.
- Object Detection - Identifying and locating objects within images.
- Semantic Segmentation - Classifying each pixel of an image into categories.
- Pose Estimation - Detecting human figures in images and estimating their poses.
- Action Recognition - Recognizing human actions in videos.

For streaming, the repository integrates various inputs and outputs, supporting live camera feeds, video files, and network video streams. The *videoSource* class is utilized to manage input streams, capable of decoding and processing video from formats such as H.264 and MJPEG, and supporting various image formats like JPG, PNG, and BMP.

Moreover, comprehensive tutorials and documentation are included, guiding through the setup and usage of these models for real-time applications. Both Python and C++ APIs are provided, facilitating the integration of deep learning capabilities into projects, whether handling static images or streams from live cameras.

These capabilities are applicable in a diverse range of scenarios, from simple image classification tasks to complex real-time object detection and tracking in video streams. The flexibility in streaming options enhances deployment in varied environments, improving the adaptability and utility of the repository in real-world applications or projects such as the one described in this thesis.

For further details and setup instructions, the repository can be explored directly on GitHub[4].

[4]https://github.com/dusty-nv/jetson-inference

3.3 Employed equipment

3.3.1 Workstation

The workstation is a PC configuration assembled from individually selected components. This PC configuration is designed to serve as a cloud computing platform and control plane, featuring high-performance components suited for demanding tasks such as data processing, virtualization, and hosting applications. At its core is the Intel LGA1700 i7-12700 processor, operating at 2.1GHz, which offers a blend of performance and efficiency for cloud workloads. It's supported by the ASUS PRIME B760-PLUS motherboard, a reliable foundation with DDR5 support, ensuring high-speed memory operations with two Kingston DDR5 16GB RAM sticks at 5200MHz.

For power, it uses the NOX HUMMER GD750W power supply with an 80 PLUS certification, indicating high energy efficiency. Storage is handled by a Samsung M.2 1TB 990 PRO NVMe PCIe 4.0 x4 SSD, offering fast data access speeds crucial for cloud operations. The inclusion of a PNY GeForce RTX 4080 16GB XLR8 Gaming Verto graphics card allows for GPU-accelerated tasks, enhancing the system's ability to handle AI, machine learning, and complex graphical workloads.

Network connectivity is assured by a TP-Link TG-3468 PCI Express Gigabit network card, providing reliable and fast internet connection vital for cloud services.

3.3.2 AMARI Callbox Classic

Amarisoft provides a 5G in a box solution, providing the software stack and SDR hardware. This solution is called AMARI Callbox Classic [6] and it is a tool designed for device testing across multiple network technologies such as 5G NSA and SA, LTE, LTE-M, and NB-IoT. It mimics the functionality of both eNB/gNB and EPC/5GC in compliance with 3GPP standards, facilitating the evaluation of functional and performance aspects of devices. Originally crafted to support three LTE sectors in a 20MHz 2x2 MIMO setup, it has since been updated to also handle configurations for SA and NSA with lower bandwidths, offering a versatile platform for a wide range of testing scenarios. Some of the capabilities and highlighted features are:

- **5G Features:** Operates in both 5G standalone (SA) and non-standalone (NSA) modes, supporting 5G Non-Terrestrial Network (NTN) and 5G Reduced Capacity (RedCap).
- **LTE:** Provides robust LTE technology support, optimized for 5G.
- **LTE-M:** Supports LTE-M connectivity for CAT-M1 devices in both FDD and TDD.

- **NB-IOT:** Enables connection for NB1 and NB2 devices via standalone, in-band, and guard-band NB-IOT. Includes Non-Terrestrial Network (NTN) NB-IOT support.
- **User Equipment Support:** Can manage up to 1000 active UEs, depending on the callbox model and configuration.
- **Throughput:** Offers up to 600 Mbps downlink and 210 Mbps uplink, depending on configuration and UE capabilities.
- **Handover:** Supports intra eNB/gNB handover with a single callbox; inter eNB/gNB handover supported using two callboxes.
- **Carrier Aggregation:** Capable of aggregating multiple TDD and FDD LTE, NR FR1, and NR FR2 cells for high throughput testing.
- **Voice and SMS Services:** Embedded IMS Server allows for VoLTE, VoNR, SMS, emergency calls, and VoWiFi through an external Wi-Fi access point via the embedded N3IWF.
- **Logging and Measurements:** Selective logging and display of all layers of 3GPP LTE and NR stacks, with useful graphs and analytic tools.
- **Automatic Test Setup and Scripting:** Extensive WebSocket API allows remote commands to network elements for easy test automation.
- **Easy Configuration:** Simplified process using JSON files, with example configurations included in each software release.
- **End-to-End Data Testing:** Runs on standard Linux in user space mode, allowing easy integration with IP services.
- **Channel Simulation:** Simulation of different downlink channel types as per 3GPP standards.
- **Feature Testing:** Enables testing features to simulate error cases by overriding normal protocol behavior.
- **High Performance:** Supports multiple UEs and cells and high data rates in LTE and NR, optimized for performance.
- **Early 3GPP Feature Access:** Provides early access to developing 3GPP features for rapid validation.

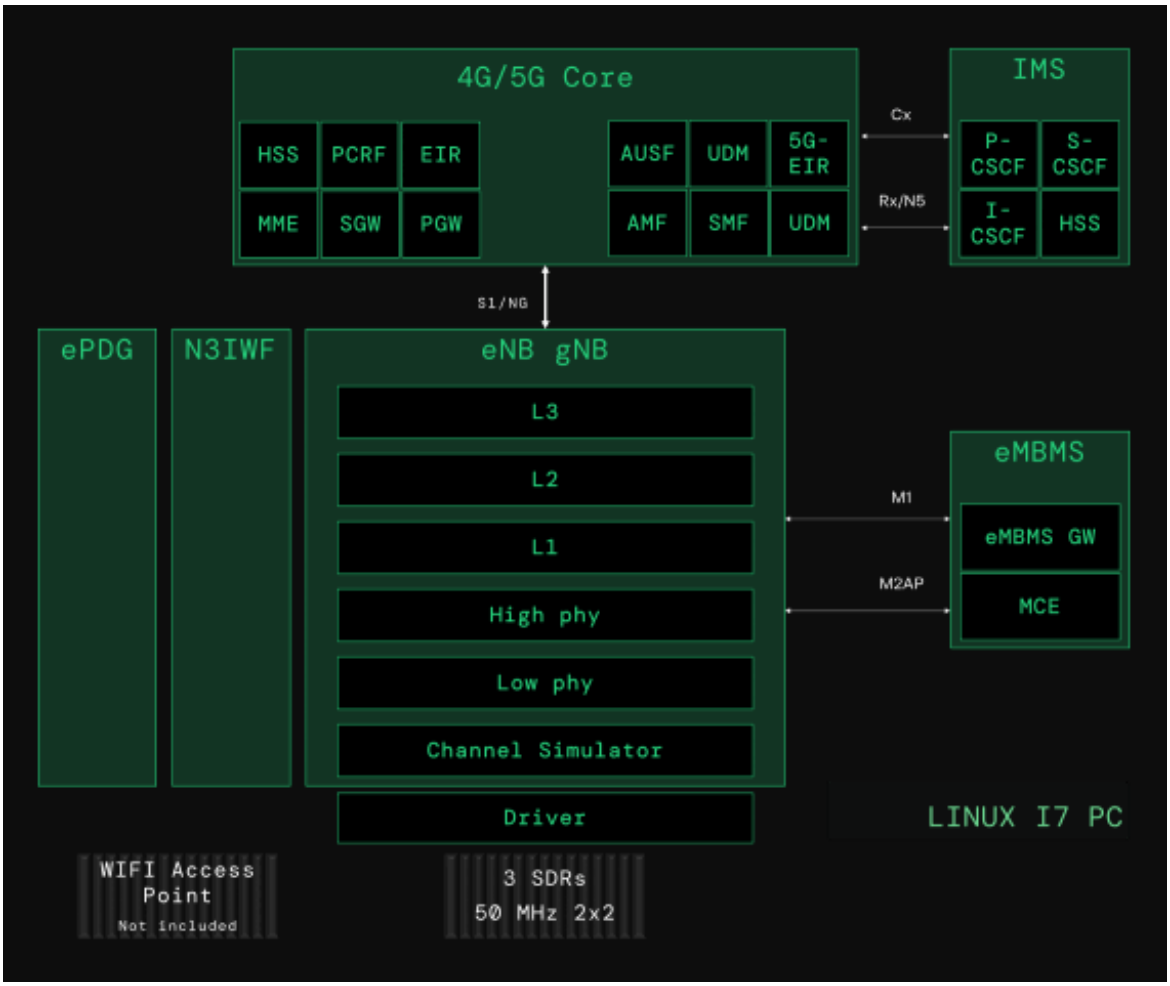


Fig. 3.2 Amarisoft Core architecture [6]

- **Frequency Agnostic:** Supports a wide range of FDD and TDD frequencies, including non-standard ones for Sub-6GHz and mmWave.

The AMARI Callbox uses the AMARI PCIe SDR 2x2 card, which is a software-defined radio (SDR) equipped with the AD9361 2x2 RF transceiver. This setup is designed for generating and processing signals, supporting MIMO 2x2, and capable of operating across a wide frequency range from 500 MHz to 6 GHz. It includes an integrated GPS for precise synchronization of time and frequency. The card allows for easy connection with others of its kind to support more complex testing scenarios, thanks to its daisy-chaining capability, which ensures synchronization of clock and pulse per second signals. It has a bandwidth of 56 MHz and provides an output power of around 0 dBm, which can vary based on the frequency used.

Finally, as depicted in Figure 3.2, the product has the following software components:

- **vRAN, eNodeB and gNodeB:** A Release 17 full software stack for eNodeB and gNodeB, including all necessary layers and a channel simulator. It connects to radio front ends through an open API and supports S1/NG interfaces for 4G or 5G core network connectivity. Configurable via JSON, it also features a WebSocket API for automation and a command-line interface for manual adjustments.
- **4G and 5G CORE:** A compact, Release 17 core network software integrating MME, SGW, PGW, PCRF, HSS, EIR, ePDG, AMF, AUSF, SMF, UPF, UDM, and 5G-EIR, supporting the deployment of both 4G and 5G networks.
- **IMS Server:** An IMS standalone server equipped with P-CSCF, I-CSCF, S-CSCF, and HSS, designed for managing Internet Protocol (IP) multimedia services.
- **eMBMS Gateway:** An LTE multimedia broadcast multicast services gateway featuring a built-in Multicast Coordination Entity (MCE) for broadcasting to multiple users simultaneously.

3.3.3 Edge devices

During the project's development, two main edge devices have been used, one Raspberry Pi 5 where the edge part of the 5G core is deployed, and NVIDIA Jetsons responsible for the High Performance Computing part of some implementations.

The Raspberry Pi 5 [21] is equipped with a Broadcom BCM2712 quad-core Arm Cortex-A76 CPU at 2.4GHz and offers configurations with up to 8GB of LPDDR4X-4267 SDRAM. These specifications mark a notable improvement in processing capabilities compared to its predecessor, making it suitable for handling more complex tasks. The Raspberry Pi 5 is used as an edge node mainly because Raspberry Pis are generally recognized as effective for such purposes. They offer a balance of computing power, connectivity options, and a versatile software ecosystem. However, for applications requiring more substantial computational resources at the edge, NVIDIA Jetson modules are preferred due to their superior processing capabilities.

NVIDIA Jetson [22] is a series of embedded computing boards designed by NVIDIA to accelerate machine learning applications. These compact, powerful boards are used for edge AI applications, where computing power is required close to the data source to reduce latency and data transmission costs. The Jetson family is popular for robotics, autonomous vehicles, and other AI-driven applications due to their low power consumption and high performance.

The Jetson series includes several models, each tailored to different performance levels and power requirements. For instance, in the scope of this project, the Jetson Nano [23],

released in 2019, is known for its 128-core NVIDIA Maxwell architecture GPU and quad-core ARM Cortex-A57 processor, offering 472 GFLOPS of performance at a power draw of 5-10W. The more advanced Jetson AGX Orin [24], released in 2023, dramatically increases performance to 200-275 TOPS with up to 2048-core NVIDIA Ampere architecture GPU, targeting more demanding AI applications.

NVIDIA provides the JetPack SDK [25] for the Jetson series, which includes the Linux for Tegra (L4T) operating system and other development tools. This package supports application development across the Jetson family, enabling developers to create and deploy AI and machine learning applications efficiently. Additionally, RedHawk Linux, a high-performance RTOS, is available for the Jetson platform, offering enhancements for real-time applications.

3.3.4 UEs

In the project, two main devices serve as UE: a Samsung Galaxy A23 5G phone and a Raspberry Pi 4 Model B connected to a Quectel RM502Q-AE 5G modem. Also, as detailed in Section 6.3, a Jetson Nano also operates as a UE, not for its computational capabilities but to address compatibility requirements with specific streaming protocols.

The Samsung Galaxy A23 5G [26] is a mid-range smartphone featuring a Qualcomm Snapdragon 695 chipset with 5G connectivity, enhancing data transmission speeds for telecommunications applications. It comes equipped with a 6.6-inch FHD+ display, offering a 120Hz refresh rate, and it is powered by a 5,000 mAh battery, aimed at providing long-lasting use on a single charge. It is used in the project mainly because of the 5G capabilities and also to use a Commercial Off-The-Shelf (COTS) device on the testbed. In telecommunications projects, COTS refers to commercial available products that can be directly utilized without modifications. Employing COTS devices like the Samsung Galaxy A23 5G in testbed environments enables the assessment of network and application performance under real-world conditions using commonly available hardware.

Given the lack of flexibility in terms of software modifications of the COTS, a modem that could be connected to a Raspberry was acquired. The Quectel RM502Q-AE [27] is a 5G module designed primarily for IoT and eMBB applications, supporting both NSA and SA modes as per 3GPP Rel-15 LTE technology. It's notable for its worldwide coverage, compatibility with multiple Quectel LTE-A modules, and integration with Qualcomm's IZat location technology. This module gives an internet interface to the device which it is connected to. In this case, the device which has the modem connected is a Raspberry Pi 4 that gives the flexibility and programmability of a Linux system for the experiments needed. The Raspberry Pi 4 Model B [28] with 8GB RAM, is powered by a Broadcom BCM2711,

Quad core Cortex-A72 (ARM v8) 64-bit SoC running at 1.5 GHz and equipped with 8GB of LPDDR4-3200 SDRAM.

Regarding the SIMs, the UEs use sysmoISIM-SJA2 [29] which is a versatile and programmable SIM/USIM/ISIM card compliant with ETSI/3GPP standards, designed for cellular technology R&D and small or laboratory network operations. It offers full programmability, allowing modification of file contents, authentication keys, service activations, and application management. With capabilities for remote interaction and support for 3GPP Release 16 features, it's suitable for use in various cellular infrastructure projects. The card is available in multiple form factors for broad compatibility. It's aimed at network operators and researchers. Also, the Amarisoft SIM cards [5] are used on the first deployments with the Amarisoft core, but then all the deployments are done using the sysmoISIM cards.

3.4 Testbed architecture

This section focuses on describing the architecture and frameworks/equipment used for the 5G-enabled edge testbed deployed in this project. This section also enumerates the multiple prior deployments done using, in some cases, other mentioned frameworks and equipment.

The frameworks and equipment used for the target testbed are:

- Open5GS for the 5G CN deployed using docker compose.
- Workstation described in section 3.3.1 which acts as the control plane (cloud) where all the basic 5G SA NFs will be deployed.
- Amarisoft gNB (AMARI Callbox Classic) for the gNB
- Raspberry Pi 5 acting as edge node where the edge UPF and SMF will be deployed.
- Samsung Galaxy A23 5G or Raspberry Pi connected to Quectel RM502Q-AE acting as user equipment. Only one of them is used at a time.

As depicted in Figure 3.3 all the testbed is deployed inside the same rack, therefore, all the components are inside the same IP range which is 192.168.50.0/24. These components are Amarisoft gNB (192.168.50.60), cloud node or workstation (192.168.50.61) and edge node or Raspberry Pi 5 (192.168.50.82). All the containers deployed in both nodes (edge or cloud) have their own docker network. Both nodes, workstation (cloud) and Raspberry Pi 5 (edge) have their own docker network, both have the same range (100.0.0.0/24) but do

[5] Amarisoft SIM cards seem to be model sysmoUSIM-SJS1 [30]

not conflict since they are on different machines. The UE will acquire a different IP inside the range of the subnetwork defined in the configuration of the network and depending on the selected APN (10.0.100.0/24 for cloud and 10.0.101.0/24 for edge). The Raspberry Pi 4 (modem UE) will have an interface called *enxXXXXX* given by the modem and with IP given by DHCP.

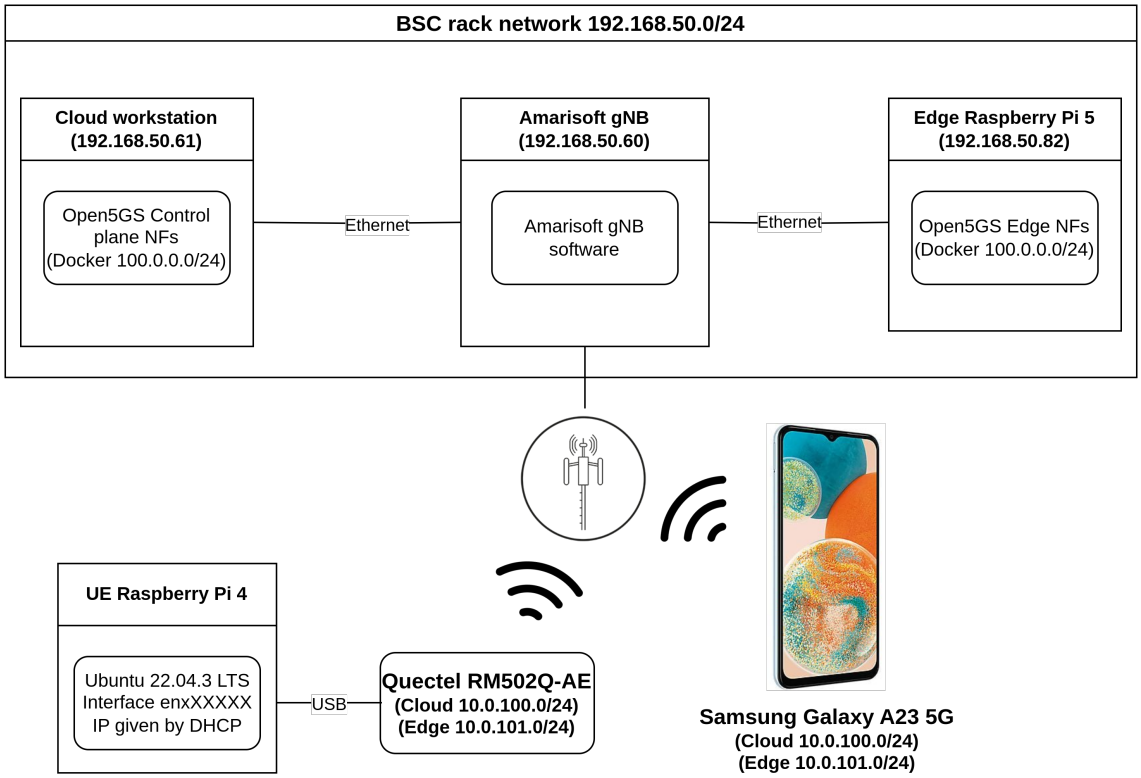


Fig. 3.3 Main testbed architecture

The cloud has the following containerized NFs of the 5G CN (Open5GS): AMF, NRF, AUSF, BSF, NSSF, PCF, UDM, UDR, SMF1, UPF1 and SCP [6]. On the other hand, in the edge device there are the following containerized NFs: SMF2 and UPF2. The cloud and edge nodes use the SCP for internode communication, ensuring efficient service discovery and message exchange between NFs distributed across both locations.

Prior to the final deployment, there were several other deployments, which served the purpose of learning how Open5GS worked and become familiar with the Amarisoft callbox. These initial deployments were crucial for acquiring the necessary expertise to implement

[6]Service Communication Proxy. It is designed to facilitate the communication between other NFs within the 5G CN. SCP acts as a mediator that simplifies the routing and forwarding of messages among the NFs, thereby enhancing the efficiency of signaling traffic management across the network.

the final testbed deployment described above. These first deployments are described in section 4.1. Here is a list of the different first deployments that have been done over the project:

- **Example deployments:** Example deployments from the tutorials listed in Open5GS documentation page[7]. In these tutorials, VMs are used to deploy the desired configuration of the network and UERANSIM is used as gNB and to test connectivity from the UEs.
- **Docker based deployments on a single host:** Open5GS deployed locally with just one data plane (one UPF and one SMF) using Docker Compose. The gNB used is UERANSIM on the first tests and then Amarisoft gNB in the end. UEs are first tested using UERANSIM and finally the Samsung Galaxy A23 5G and the Quectel modem.
- **Other deployments:** Three additional deployments were also tested to explore different possibilities and configurations. These deployments will not be described in detail, since they were not considered in the final deployment, but they will be briefly mentioned, highlighting the lessons learnt.

3.5 Performance metrics

To convincingly argue that content routing and hosting at the edge reduces latency, specific performance metrics are essential. Chapter 5 thoroughly examines and analyzes these metrics. The metrics employed include:

- **Delay:** delay refers to the time it takes for a packet of data to travel from its source to its destination across a network. This includes all the delays caused by the route taken by the packet, processing time at any nodes encountered, and any other type of latency introduced by the network infrastructure. It will be measured using `ping`.
- **Throughput:** throughput is the rate at which data is successfully transmitted over a network from one point to another within a given time frame. It's a measure of how much data can be moved successfully from source to destination, often considered in terms of bits per second (bps), and is a key indicator of network performance. It will be measured using `iperf3`.
- **Jitter:** Jitter refers to the variation in latency or delay between data packets over a network. It will be measured using `iperf3`.

[7] https://open5gs.org/open5gs/docs/

3.6 Application Scenarios

The last step of the project is the examination of specific application use cases. These use cases are designed to validate the testbed's functionality and to demonstrate the practical application of 5G technologies and edge computing within the testbed. The objectives include:

- Verifying the operational status of the testbed, ensuring that UE can access 5G services and the internet.
- Demonstrating the feasibility of deploying applications at the network edge, which operate concurrently with the UPF service (on the same node).
- Illustrating the capability to select the desired UPF through specified Access Point Names (APN) or Data Network Names (DNN) on the UE.
- Confirming the network's ability to route all UE-associated data via the chosen UPF, whether located at the edge or in the cloud.

These applications are elaborated upon in Chapter 6 encompassing the following implementations:

- Messaging service: deployment of an MQTT Broker at the network edge.
- Video streaming service: Streaming service from the edge using HTTP.
- Real time object detection service: UE video streaming and processing at the edge using HPC with NVIDIA Jetsons.

The first application serves as a proof of concept for the viability of running services on the edge where a UPF is deployed. The second application underscores the importance of latency in streaming services, thus, it demonstrates the capability of the edge of offering good quality delay-sensible content/services. Lastly, the final application provides a practical example of how this real-time data streaming can be processed on the edge, adding AI inference though deep learning, and this can benefit latency-sensitive applications, offering users quicker access to processed content. The results and the process of deploying these applications is explained in Chapter 6.

Chapter 4

Testbed deployment

4.1 First deployments

4.1.1 Example deployments

At the beginning of the project, several deployments of the documentation page of Open5GS[1] were followed with the objective of becoming familiar with the framework and with the 5G core. In this subsection, only two of these examples are explained, selected because of their similarity with the final deployment and with the purpose of explaining the diverse deployments and the basics of Open5G and UERANSIM in a progressive manner.

Both examples (and most of the documentation examples) are executed using Virtual Machines (VMs). The Oracle open-source hosted hypervisor for x86 virtualization Virtual Box is used to execute these multiple VMs. This is because it also has the option of creating groups of multiple VMs that can be executed at one, and this is useful for these examples where the different 5G slices and the RAN simulators are spread across 5 VMs. Both of the examples use UERANSIM as the RAN simulator, simulating both gNB and UE. These examples are used to show two different ways of duplicating the user plane using two UPFs.

The first example, *Example 1*[2], consists of one control plane (C-plane) and two user planes (U-planes), with one UPF on each. The data plane to connect to is selected using the desired DNN in the configuration of the UE, denoted *internet* or *internet2*. There is also an *ims* DNN that corresponds to the IMS service of 5G but is also used for internet connection. Figure 4.1 shows the 5 different VMs and Open5GS and UERANSIM function daemon that will run inside each of them. It also shows the assigned IPs for each of the elements, the

[1] https://open5gs.org/open5gs/docs/

[2] https://github.com/s5uishida/open5gs_5gc_ueransim_sample_config

IMSI[3] numbers of the UEs and the different *ogstun* GTP tunnels that represent each of the DNNs of the network.

The second example, *Example 2*[4], again consists of one control plane (C-plane) and two user planes (U-planes), with one UPF on each. In this example, each data plane is assigned to a different network slice. This is similar to the final testbed deployment, where network slicing is used to separate the traffic routed through the edge. The data plane to connect to is selected using the desired NSSAI on the configuration of the UE, *SST:1, SD:0x000001* or *SST:1, SD:0x000001*. In this example, *ims* DNN is not present. Two internet DNNs are configured, one for each of the U-planes. The IPs of all the elements of the network, the IMSI of the UE and all the different VMs with its services are shown in Figure 4.2.

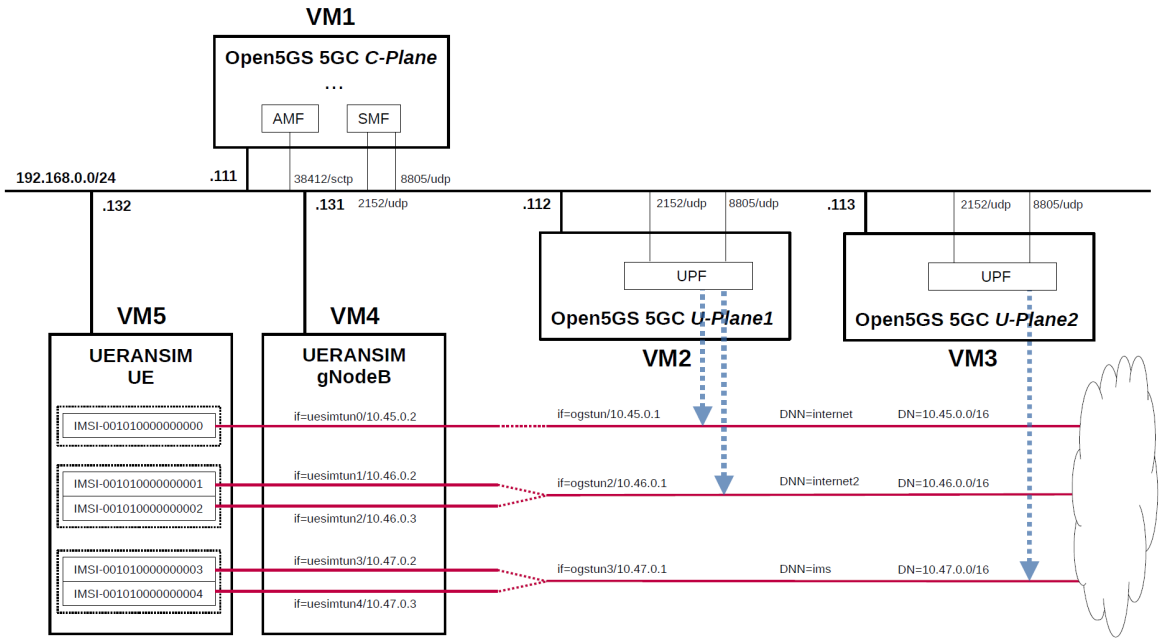


Fig. 4.1 *Example 1*: Open5GS and UERANSIM example from the Open5GS documentation, which selects the UPF based on the configured DNN on the UEs

The process to execute both examples is similar. First, a VM image using Desktop Ubuntu 22.04[5] is downloaded and installed in VM Box. Then the Open5GS repository has to be built from source using the tutorial provided by Open5GS [6]. The same has to be done

[3]IMSI (International Mobile Subscriber Identity) is a unique number associated with all cellular networks. It is used for identifying the users of a cellular network and is stored as a 64-bit field and is sent by the phone to the network. It is also used for acquiring other details of the mobile in the home location register (HLR) or as locally copied in the visitor location register.

[4]https://github.com/s5uishida/open5gs_5gc_ueransim_snssai_upf_sample_config

[5]https://ubuntu.com/download/desktop

[6]https://open5gs.org/open5gs/docs/guide/02-building-open5gs-from-sources/

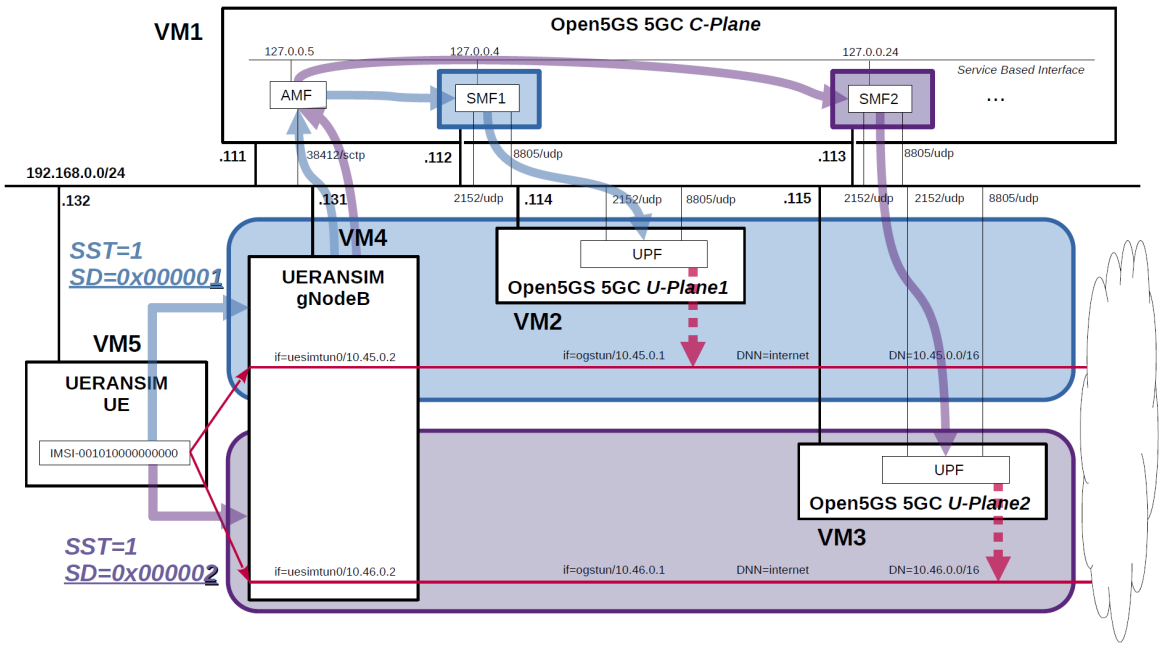


Fig. 4.2 *Example 2*: Open5GS and UERANSIM example from the Open5GS documentation, which selects the UPF based on the configured NSSAI on the UEs

with UERANSIM using the UERANSIM installation guide[7]. Once the VM has both of the frameworks installed, it can be replicated (copied) as many times as needed, in this case 5 times for both of the examples. During the replication step, it is recommended to rename the VMs with something that describes the functionality that VM has inside the deployment. Also, a common NAT network should be created with all the different VMs. It is important to note that since different IPs will be assigned by DHCP to the VMs, it is recommended to note the IPs in a table similar to the table in Figure 4.3. Then they can be grouped in Virtual Box so they can be executed all at once.

The next step is to make changes in the configuration files of Open5GS and UERANSIM. These configuration files are YAML format and can be found in *open5gs/install/etc/open5gs/*. There is a file for each of the 4G and 5G functions, with the name of each file corresponding to the respective core service, i.e.,*amf.yaml*, *smf.yaml*, *upf.yaml*, etc. In this project, the focus is on the deployed 5G functions.

All default YAML configuration files for the services maintain a similar structure at the root level. It's essential to note that this structure might be adjusted in upcoming deployments to meet specific needs and requirements. Despite these adjustments, the configuration files continue to be intuitive and straightforward, as the variations in structure are similar to those explained below. The primary root level keys include:

[7]https://github.com/aligungr/UERANSIM/wiki/Installation

VM #	SW & Role	IP address	OS	Memory (Min)	HDD (Min)
VM1	Open5GS 5GC C-Plane	192.168.0.111/24 192.168.0.112/24 192.168.0.113/24	Ubuntu 22.04	2GB	20GB
VM2	Open5GS 5GC U-Plane1	192.168.0.114/24	Ubuntu 22.04	1GB	20GB
VM3	Open5GS 5GC U-Plane2	192.168.0.115/24	Ubuntu 22.04	1GB	20GB
VM4	UERANSIM RAN (gNodeB)	192.168.0.131/24	Ubuntu 22.04	1GB	10GB
VM5	UERANSIM UE	192.168.0.132/24	Ubuntu 22.04	1GB	10GB

Fig. 4.3 Table of IPs and VM characteristics for the second example extracted from the GitHub Tutorial

- `logger`: Configuration related to the logging and the path of the logs file. There is one log file for each of the services.
- `global`: Settings that apply universally across various components of the network or service. It typically includes parameters that affect the operation, performance, and management of the network on a system-wide level. The default configuration is the max number of UEs accepted on the network.
- `amf`, `smf`, `upf`, etc.: The configuration of the service of the configuration file. Each file has its service configuration tag, i.e, *amf.yaml* has the `amf` tag inside and *smf.yaml* has the `smf` tag. In the default configuration files available on Open5GS's GitHub repository, each service's YAML file primarily contains settings specific to that particular service. However, it's important to note that these configuration files are capable of including SBI configurations and other parameters for different services as well. As it will be seen in future deployments, additional services and parameters can be configured within the same YAML file.
  - `sbi`: As explained before in this thesis, SBI refers to the set of API-based interfaces that allow different NFs to communicate with each other. Each NF exposes its services via APIs on the SBI, and other NFs consume these services using a client-server model.
    - `server`: Specifies the address where the service listens to requests of other NFs. Default is set to `localhost`, it can be setup to listen on another machine.
    - `client`: Specifies the address of the NFs that the service consumes from. All of them consume service from SCP, this is because SCP acts as a central

hub that facilitates the communication and data exchange between different NFs. It helps in orchestrating and managing service discovery, ensuring that requests from one NF are correctly routed to the appropriate providing NFs based on service availability, load, and other operational policies.

Regarding the service names on the root levels, some of the possible configuration fields per service are listed below. It should be noted that parameters that are common across all services and are identical in function and structure will only be described once, to avoid redundancy in the explanation.

- `amf`:
  - `sbi`
  - `ngap`: gNB IP address.
  - `guami`: Globally Unique AMF Identifier (GUAMI) uniquely identifies the AMF on a global scale. It contains information such as:
    - `plmn_id`: This includes the Mobile Country Code (MCC) and Mobile Network Code (MNC), which together identify the home network of the subscriber.
    - `amf_id`: This is composed of the region and set identifiers.
  - `tai`: Tracking Area Identity (TAI), used primarily for tracking and paging UEs as they move between different geographical areas within a network. It contains information such as:
    - `plmn_id`
    - `tac`: Tracking Area Code (TAC) specifies a particular area within the network's coverage. This is smaller than regions covered by PLMN, but larger than cells.
  - `plmn_support`: Specifies settings related to the support of PLMNs. It consists of:
    - `plmn_id`
    - `s_nssai`: Single Network Slice Selection Assistance Information (S-NSSAI) defines one or more Slice/Service Type (SST) identifying a specific type of network slice and Slice Differentiators (SD) which is an optional field used to further distinguish between multiple network slices within the same SST.

  - `metrics`: some of the services have the metrics server configuration. This metrics server is an embedded Prometheus[8] server that tracks certain jobs[9] specified on the metrics server configuration files of Open5GS.
  - `security`: Specifies the preferences for security algorithms used within a network component, such as an AMF in a 5G system. It is composed of:
    * `integrity_order`: Array, which lists the preferred order of integrity protection algorithms. These algorithms ensure that the data transmitted over the network has not been tampered with.
    * `ciphering_order`: Array, which indicates the preferred order of encryption algorithms used to secure data.
  - `network_name`: The network that would appear on the operator search on a phone, for example.
  - `amf_name`: The name of the AMF inside the network.
  - `time`: Management of various timers used in network operations.

- `smf`:
  - `sbi`
  - `pfcp`: Packet Forwarding Control Protocol (PFCP) is a protocol used for managing the user plane, which is responsible for handling data packets as they travel through the network. In this setting the server side address is set and for the client side the UPF service address is set up.
  - `gtpc`: GPRS Tunneling Protocol Control (GTP-C) used for signaling in GPRS and LTE networks, primarily to manage and maintain mobile data sessions between network gateways and mobile devices. Here, the GTP-C server address is set up.
  - `gtpu`: GPRS Tunneling Protocol User (GTP-U) is used for tunneling user data within mobile networks, particularly in transferring data packets between the mobile device and the external network. This parameter sets up the GTP-U server address.
  - `session`: This section defines IP subnets and gateways for network sessions. Additionally, it specifies the DNN associated with each pair of subnet and gateway.

[8] https://prometheus.io/

[9] In the context of Prometheus, a job is a collection of similar instances that are grouped together for monitoring purposes. Each job consists of multiple endpoints that Prometheus can scrape to collect metrics. These endpoints usually correspond to instances of a single application or service.

  - `dns`: This configuration specifies the DNS servers used for resolving domain names associated with requests transmitted across the subnets managed by the SMF. The project is conducted within a highly secure environment using a rack that includes a Raspberry Pi configured with OpenWRT[10]. In this setup, the Raspberry Pi serves both as a router and a VPN gateway. Due to security policies, the use of public DNS servers is restricted, therefore, the designated DNS is the internal IP address of the Raspberry Pi itself.
  - `metrics`
  - `mtu`: Specifies the Maximum Transmission Unit (MTU) for the network.
  - `freeDiameter`: Specifies the configuration file location for freeDiameter, a framework for Diameter protocol implementations. The Diameter protocol is an authentication, authorization, and accounting (AAA) protocol used in telecommunications networks. It evolved from and replaces the older RADIUS protocol, offering more robust and flexible capabilities.

- `upf`:
  - `sbi`
  - `pfcp`
  - `gtpu`
  - `session`
  - `metrics`
- `nrf`:
  - `sbi`: In this case, SBI only has server sinde NRF does not consume any services.
  - `serving`:
    - `plmn_id`
- `nssf`:
  - `sbi`
    - `server`
    - `client`
      - `scp`

[10]https://openwrt.org/

        - `nsi`: Network Slice Instances (NSI) are the pairs of addresses and `s_nssais` of the slices configured in the network. It has to be equivalent to the ones in AMF.

- `pcf`:
  - `sbi`
  - `metrics`
  - `policy`: Settings for policy control and rule enforcement within the network, which determine how network resources are allocated and managed based on predefined rules and dynamic conditions.
- `scp`:
  - `sbi`
  - `info`: Specifies the network and port details for how different components within the network communicate with the SCP.
- `udm`:
  - `sbi`
  - `hnet`: Stands for home network and sets configurations for cryptographic keys used in network security operations, particularly for handling encryption or authentication processes.
- `bsf`, `ausf`, `udr`:
  - `sbi`

Similarly, UERANSIM utilizes two main types of YAML configuration files, which are stored in the *UERANSIM/config/* directory. These files are designed to be user-friendly and include helpful comments, making them straightforward to understand and modify. It is also important to notice that to subscribe or use more than one UEs, the YAML file that configures UEs can be replicated.

After implementing the YAML configuration changes (mostly IP changes across the multiple YAML files to communicate services between them) outlined in both guides and adding the subscriber UE information in the WebUI[11], it is necessary to execute specific network commands on the respective machines. These commands are crucial for adding

[11]This is a web service where the UE subscription information can be set up in an easy visual manner.

SMF routes to the control planes, and enabling port forwarding and creating tunnel devices on the user planes. This can be done manually, as detailed in the tutorials, or more efficiently through a shell script that automatically executes these commands at boot time on each of the VMs.

In this project, the latter method using a shell script was adopted to automate the deployment. Consequently, the VMs have been organized by deployment in Virtual Box, corresponding to each tutorial guide. Each deployment group contains five VMs, named after the network component or function they represent. These VMs are connected using the same NAT network and can be launched simultaneously since all network configuration and execution scripts are run at boot time. The group of VMs for *Example 2* is shown by Figure 4.4 and the scripts inside the control plane VM (VM1) of that group can be seen in Figure 4.5. This approach significantly automates the deployment process even with regular VMs and highlights the need for an even more automated setup. Once everything is deployed, network functionality can be verified by performing commands such as `tcpdump` or `ping` to ensure the network is operating correctly and going through the desired UPF[12].

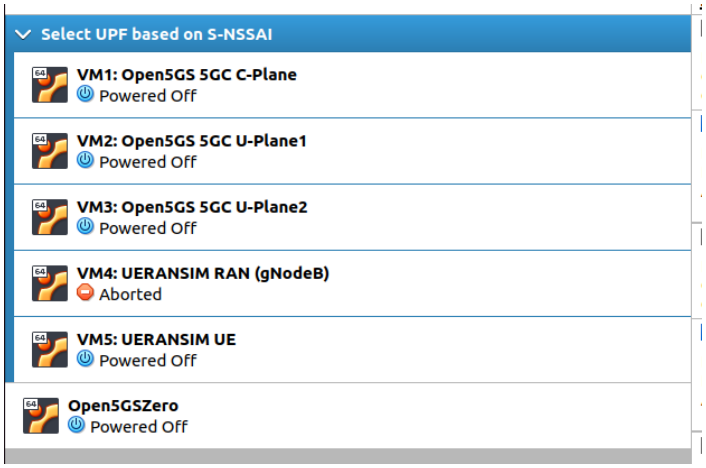


Fig. 4.4 Virtual Box showing the group of configured VMs for *Example 2* and the Open5GS initial VM, Open5GSZero, which has been cloned.

4.1.2 Docker based deployments on a single host

As seen in the previous section, deploying all the services using VMs as devices or planes is not optimal. This section explains two deployments that automate the execution process of the testbed using Docker Compose. These deployments are called *Deployment 1*[13] and

[12]The exact configuration or steps to select the UPF on both examples is detailed on the tutorials
[13]https://github.com/Neruzzz/Open5GS-DockerLocal

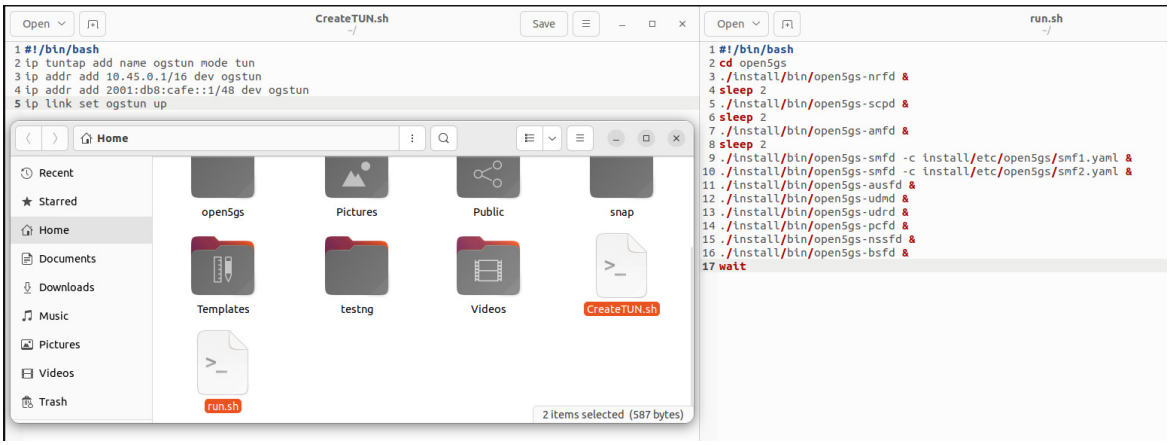


Fig. 4.5 *Example 2* VM1 C-Plane screenshot showing the commands that are executed automatically during boot time

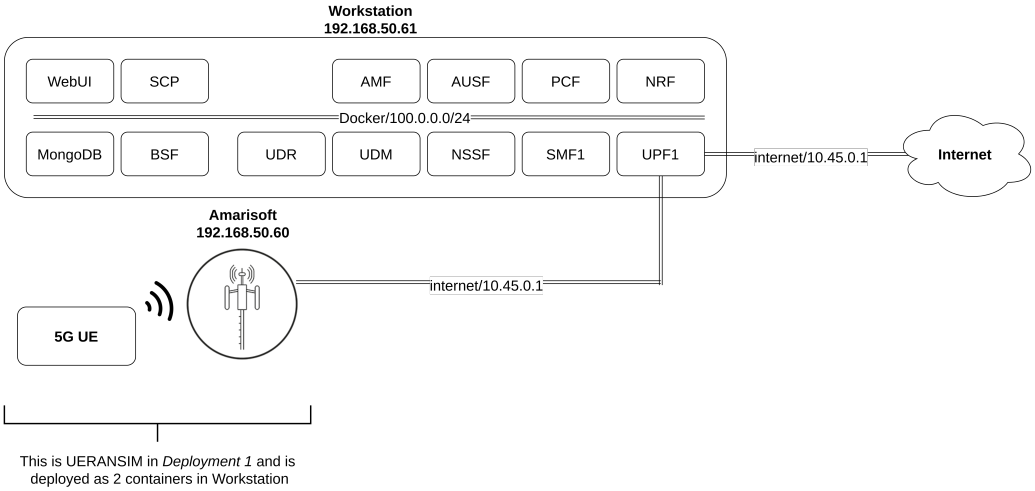


Fig. 4.6 *Deployment 1* and *Deployment 2* diagram, showing the architecture of the network.

Deployment 2[14]. The difference between *Deployment 1* and *Deployment 2* is the RAN, which in the case of 1 is simulated by UERANSIM and in the case of 2 is realistic using Amarisoft gNB and a COTS device (Galaxy A23 or Quectel modem).

The deployed network consists of a single slice where all services are executed in Docker containers, and a single DNN called *internet*. The deployments are depicted by Figure 4.6.

These deployments are based on the *docker_open5gs* GitHub repository[15] created by Supreeth Herle. The functionality of the base project is discussed first, as it is not straightforward, then *Deployments 1 and 2* will be explained.

[14]https://github.com/Neruzzz/Open5GS-DockerCompose-Amarisoft
[15]https://github.com/herlesupreeth/docker_open5gs

All the needed services for deploying a SA 5G core are described in a Docker Compose file called *sa-deploy.yaml*. This Compose file, available on the repository, has the following structure:

- `version`: Specifies the version of the Docker Compose file format being used. This is crucial because it determines which features and syntax are supported in the Compose file.
- `services`: This is a fundamental component of the Compose file. It defines all the applications or services that are part of a project in a Docker Compose file, specifying how these services are built, run, and interconnected. Each service in the services block can represent a container or a set of containers that operate together.
- `networks`: This tag is used to define and configure custom networks within the Docker environment. In this case, a network called `default` with its own range used by all the services on the Docker Compose.
- `volumes`: Is used to define named volumes that can be used by containers within the Docker environment. In this case, the only volume is `mongodbdata` which is used by the `mongo` service on the same compose. This ensures that data is not lost when containers are destroyed or recreated. So the UE data is persistent between executions.

Then each service has its own structure inside the Compose file that defines things such as the image to use, the network to use, the ports to expose, the volumes to use inside the container from the local machine, etc. The structure of these services can be found below:

- `servicename`: Name of the service.
  - `build`: Used to specify configuration options for building a Docker image from a Dockerfile, rather than pulling an existing image from a registry. It can be the directory where the Dockerfile is located, so it is built at runtime.
  - `image`: Specifies the Docker image to be used for running the service.
  - `container_name`: Assigns a static name to the container for easier referencing.
  - `command`: Overrides the default command specified by the container image. This ca also be used to execute commands at the beginning of the service.
  - `depends_on`: Defines startup dependencies between services. The container will only be created and run once the containers it depends on are running, regardless of the status of the applications or services inside those containers.

- `env_file`: Specifies a file to load environment variables from.
- `environment`: Adds or overrides environment variables within the container. In this case, there is an environment variable called `COMPONENT_NAME` with the service name and the number of replicas it has (`service-i`).
- `volumes`: Mounts host paths or named volumes into the container.
- `ports`: Maps ports from the container to the host.
- `expose`: Exposes ports to other containers, but not to the host.
- `networks`: Connects the container to specified Docker networks.

This Docker Compose file incorporates three distinct Docker images: the official MongoDB image (`mongo:6.0`), a metrics server using Prometheus available from the GitHub files, and the Open5GS Dockerfile from the repository. The Docker images for UERANSIM and Open5GS include two levels of script execution, the first is at the end of their Dockerfiles after cloning and compiling the main applications. The process involves copying an initialization script to the root directory during the build phase, altering its execution permissions, and subsequently executing it when the container starts. This is illustrated with the following Dockerfile commands:

```
COPY open5gs/open5gs_init.sh /
RUN chmod +x open5gs_init.sh
CMD ["./open5gs_init.sh"]
```

The initialization script executes different commands based on the `COMPONENT_NAME` environment variable, allowing specific actions for each service. For example, for the UERANSIM image:

```
if [[ -z "$COMPONENT_NAME" ]]; then
        echo "Error: COMPONENT_NAME environment variable not set"; exit 1;
elif [[ "$COMPONENT_NAME" =~ ^(ueransim-gnb-[[:digit:]]+$) ]]; then
        echo "Deploying component: '$COMPONENT_NAME'"
        /mnt/ueransim/open5gs_gnb_init.sh && \
        ./nr-gnb -c ../config/open5gs-gnb.yaml & \
        bash
elif [[ "$COMPONENT_NAME" =~ ^(ueransim-ue-[[:digit:]]+$) ]]; then
        echo "Deploying component: '$COMPONENT_NAME'"
        /mnt/ueransim/open5gs_ue_init.sh && \
        ./nr-ue -c ../config/open5gs-ue.yaml & \
        bash
```

```
else
        echo "Error: Invalid component name: '$COMPONENT_NAME'"
fi
```

These scripts check the `COMPONENT_NAME`, display which component is being deployed, potentially wait for other dependencies (using `sleep`), adjust permissions, and run the service's initialization script followed by the service daemon. The service initialization scripts are the ones that change the configurations inside the YAML files that both Open5GS and UERANSIM use to configure the daemons. An example of these initialization scripts is the following:

```
cp /mnt/amf/amf.yaml install/etc/open5gs
sed -i 's|AMF_IP|'$AMF_IP'|g' install/etc/open5gs/amf.yaml
sed -i 's|SCP_IP|'$SCP_IP'|g' install/etc/open5gs/amf.yaml
sed -i 's|NRF_IP|'$NRF_IP'|g' install/etc/open5gs/amf.yaml
sed -i 's|MNC|'$MNC'|g' install/etc/open5gs/amf.yaml
sed -i 's|MCC|'$MCC'|g' install/etc/open5gs/amf.yaml
```

This is the AMF initialization script, but all services work in the same way. The script adjusts the specified configuration YAML files using environment variables that were declared in the *.env* file and made available in the container at runtime by Docker Compose. The *.env* file contains a list of configuration variables and their corresponding values, streamlining the configuration process due to the two-level service initialization. Without this mechanism, configuring each service would require manually editing each YAML configuration file for every change. The command to execute all the Compose is `docker compose up` and to choose the services to deploy `docker compose -f composefile.yaml up service1 service2 service3 ... service-n`. A snippet of the *.env* is:

```
OPEN5GS_IMAGE=registry.gitlab.bsc.es/ppc/software/open5gs/base-open5gs:local:latest
MONGO_IMAGE=registry.gitlab.bsc.es/ppc/software/open5gs/open5gs-mongo:latest
UERANSIM_IMAGE=registry.gitlab.bsc.es/ppc/software/open5gs/ueransim:local:latest

MCC=001
MNC=01

COMMON_NETWORK=162.22.0.0/24
DOCKER_HOST_IP=192.168.20.153

# MONGO IP
MONGO_IP=162.22.0.2
```

```
13
14  # OPEN5GS WEBUI
15  WEBUI_IP=162.22.0.26
16
17  # OPEN5GS SERVICES
18  # SMF
19  SMF_IP=162.22.0.7
20
21  # UPF
22  UPF_IP=162.22.0.8
23  UPF_ADVERTISE_IP=162.22.0.8
24   ...
```

Until now, we described the key features of the base deployment. This deployment has undergone two major modifications, designated as *Deployment 1* and *Deployment 2*. *Deployment 1* focuses on cleaning up the base repository, refining the Docker Compose file, and simplifying the folder structure to retain only essential functions. *Deployment 2*, on the other hand, incorporates Amarisoft gNB functionality and removes UERANSIM services from the Docker Compose file.

For these projects, the Docker images for Open5GS, MongoDB, UERANSIM, and even WebUI were initially manually created, with various versions located within the *open5gs* and *webui* folders in *Deployment 1*. Initially, these builds required full application compilations within the images, which introduced several dependency-related challenges. Ultimately, the definitive versions of these images were adopted from the base GitHub repository. Both *Deployment 1* and *Deployment 2* incorporate Makefiles to facilitate image building. Additionally, *Deployment 1* includes a Docker manifest that enables the images to operate across multiple device architectures, a feature termed enabling multiarchitecture, which distinguishes it from the practices followed in the base GitHub project.

In contrast to the base GitHub project, where UERANSIM is configured across two separate Docker Compose files (`nr-gnb.yaml` and `nr-ue.yaml`), UERANSIM is integrated directly within the Docker Compose files of *Deployment 1*. This integration strategy aims to simplify operations by allowing a single file to execute all services concurrently. It is important to note that in *Deployment 2*'s Docker Compose file, the gNB service is not included because the gNB functionality is provided by Amarisoft Callbox.

In *Deployment 2*, the configurations regarding the Amarisoft gNB are performed on the AMARI Callbox Classic. A copy of the required configuration file, typically found at *enb/config/gnb-sa.cfg*, is created and then modified. Subsequently, a symbolic link from the modified configuration file to *enb.cfg* is established to ensure the setup points to the *enb.cfg*

file that the gNB daemon uses as default. The IP of the AMF must be configured as indicated in Figure 4.7 to enable the gNB to initiate the initial registration procedure. It is also essential that the PLMN configuration matches that of the AMF, as depicted in Figure 4.8. The gNB daemon within Amarisoft is then restarted to apply the changes. This configuration allows the gNB to connect to the 5G CN and serve as the network's gNB within the testbed.

```
amf_list: [
  {
    /* address of AMF for NGAP connection. Must be modified if the AMF runs on a different host. */
    amf_addr: "192.168.50.61",
  },
],
/* GTP bind address (=address of the ethernet interface connected to
   the AMF). Must be modified if the AMF runs on a different host. */
gtp_addr: "192.168.50.60",

gnb_id_bits: 28,
gnb_id: 0x12345,
```

Fig. 4.7 AMF list in the gNB configuration of AMARI Callbox Classic

```
  },
#endif
    ssb_period: 20, /* in ms */
    n_id_cell: 500,

    plmn_list: [ {
      tac: 100,
      plmn: "99970",
      reserved: false,
      nssai: [
        {
         sst: 1,
         sd: 0x000001,
        },
        {
         sst: 1,
         sd: 0x000002,
        },
       ],
      },
    ],
```

Fig. 4.8 PLMN list in the gNB configuration of the AMARI Callbox Classic

The SIM cards have to use the Milenage algorithm[16] to work with Open5GS. This is programmed using the pySIM software developed by Osmocom that can be found in this GitHub repository[17]. The guide to follow to program the SIMs to use Milenage instead of XOR (the default algorithm used by the Amarisoft SIMs) can be found in this link[18]. Then the SIM information of the UEs has to be introduced in the WebUI interface.

[16]Milenage is the standard algorithm set used for authentication and key generation in UMTS and LTE networks. It is based on the AES cipher and was designed to provide robust security measures for mobile communications. Milenage and similar algorithms are essential for the operation of SIM cards, as they secure the process of verifying subscriber identities and encrypting data. For further details of Milenage and other SIM security algorithms, refer to the 3GPP TS 35.205 specification.

[17]https://github.com/osmocom/pysim

[18]https://tech-academy.amarisoft.com/update.wiki

Unlike the base project and *Deployment 1*, which relied on an emulated RAN, *Deployment 2* implemented real UEs and a gNB, leading to the emergence of incompatibilities and errors during service requests y the UEs. The execution of all the stack of *Deployment 2* was successful and the UEs registered to the network. However, due to a bug in the Open5GS code, the devices were unable to establish an internet connection. A debug message in the logs of the AMF in the 5G CN indicated: `DNN Not Supported OR Not Subscribed in the Slice`. It is believed to have been a bug that involved the UE requesting a PDU session that the core network did not recognize. The core's inability to manage this issue (specifically, its failure to inform the mobile device that the session did not exist) resulted in an infinite loop of queries from the UE and error messages from the core.

4.1.3 Other initial deployments

During the early stages of the project, several experimental deployments were carried out. To keep the thesis concise and focused, these have not been discussed in detail. However, a brief overview of these initial deployments is provided below, highlighting their main objectives and the key learnings derived from each:

- **Amarisoft 5G SA/NSA deployments**: Upon the arrival of Amarisoft equipment in the laboratory, two initial deployments were conducted to understand its operation, specifically the NSA and SA 5G stacks of the AMARI Callbox. Key learnings from these deployments included the identification of Amarisoft modules, which are the configurable components of the software and comprise *mme*, *enb*, *ims*, and *mbmsgw*. There was also a deeper understanding of the configuration filesystem, which utilizes symlinks to template configuration files to modify component functionalities. Additionally, the use of the `screen` functionality proved essential for accessing information from the core and the gNB, allowing for monitoring of each module and providing insights into the network's status. Moreover, skills were developed in using, programming, and subscribing Amarisoft SIM cards and various types of UEs, including the Samsung Galaxy A23.

- **Single node deployment using docker compose and Open Air Interface (OAI)**[19]: During a certain period, as detailed in the previous deployment section, COTS UEs were unable to receive services from the 5G core, despite being correctly registered.

[19]Open Air Interface (OAI) is an open-source software alliance and community that develops software for 5G wireless research and experimentation. They provide an open-source 5G CN that allows researchers and developers to prototype advance wireless technologies, enhancing the implementation and testing of new communication protocols. More information can be found on their official website: http://www.openairinterface.org/.

This deployment was initiated as a workaround to determine if OAI could resolve this issue. Unfortunately, the problem persisted with OAI as well. Key insights from this deployment included learning how to swiftly replace the core network using Docker, which facilitated a smooth transition. Additionally, it provided an understanding of the OAI architecture, revealing its similarities to Open5GS. This experience also highlighted the differences in community support and documentation between the two projects: while Open5GS benefits from a robust community but suffers from sparse documentation, OAI, in contrast, has more comprehensive documentation but lacks a strong community support network.

- **Two nodes docker swarm deployment**: The initial attempt to deploy two nodes or two UPFs was based in the same GitHub project as *Deployments* 1 and 2 and employed Docker Swarm. The architecture of the repository presented several challenges: it relies on local volumes for initializing the Open5GS docker image and sets up the IPs statically usnig the *.env* file. So it relies on the knowledge of the IPs of deployed services or containers. In Docker Swarm, unlike Docker Compose, IPs are assigned dynamically and cannot be set statically using an *.env* file. The adopted strategy involved starting all containers with docker swarm and delaying the execution of service initialization scripts and daemons using a wait command at the beginning of the Open5GS image initialization script[20]. Then, all IPs were collected using scripts within the initialization script. Subsequently, these IPs were recorded into the environment (*.env*) file. The next problem was how to send this information and the UPF and SMF volumes to the edge precess. Due to limited experience with Swarm, the implemented workaround involved pushing necessary files (UPF volume and environment file) to GitHub, followed by cloning them onto the edge machine before any service execution commenced (also using wait commands and scripts on the Open5GS initialization script). This approach, though not ideal, was functional at the time. However, the COTS UE failed to operate correctly. Over time, the repository became outdated due to updates in the Open5GS repository and incompatibilities with the Docker images that were created. This repository is available on this GitHub link[21]. This deployment provided significant knowledge in the functionality of Docker Swarm, enhanced understanding of the GitHub repository initially used for single node deployment to adapt it for two nodes, and was instructive in the management of

[20]Remember that the software architecture used, utilizes an Open5GS docker image that operates with two distinct levels of execution. Initially, the main image initialization script runs when the Docker image starts. Following this, the same script triggers the execution of service initialization scripts and then starts the service daemons.

[21]https://github.com/Neruzzz/Open5GS-DockerSwarm

volumes and IP information in distributed deployments among other lessons learned. It was also a significant step that helped to learn how the sequence of container execution works and how to alter it using commands, resulting in a substantial amount of shell scripting knowledge gained during this stage.

4.2 Final deployment

This final deployment focuses on providing the desired testbed using the learned concepts given by the latter deployments. It uses two nodes, one for the cloud and the other for the edge. It also uses Docker Compose to automate and to speed up the deployment. COTS UEs are used in this deployment and the AMARI Callbox is also used as the gNB of the network.

This deployment is based on the *slices_2upf_2smf* branch from the GitHub repository used in *Deployments* 1 and 2. The key difference is the deployment of two distinct projects, thus, two distinct Docker Compose files, one for cloud services and one for edge services and the modifications done to those projects in order to have internode communication. The repository with the modifications for the cloud slice can be found in this GitHub[22] (called cloud from now on). The repository with the changes for the edge can be found in this GitHub repository[23] (called edge from now on).

These repositories deploy the architecture seen in Figure 4.9. The RAN, edge and core devices (i.e, the Amarisoft Callbox, workstation and edge Raspberry Pi) are interconnected by the Ethernet network provided by BSC, with IPs in the 192.168.50.0/24 range. On the other hand, the UE connects solely via the 5G network, through the two available DNNs, named cloud and edge, depending on the desired configuration.

Services in the cloud are categorized into two groups: 5G CN functions (AMF, NRF, etc.) and 5G CN framework services required by the Open5G framework (the WebUI, mongo, etc.). Regarding the edge, the services deployed are SMF and UPF.

Given that the deployment spans two different nodes, certain service configurations are necessary to ensure visibility and connectivity between nodes. Each node has its own Docker network, allowing services within the same Docker network to communicate directly. These networks have the same ranges in both nodes, but they can take any range that does not conflict with the host machine. Additionally, the Raspberry Pi utilized at the edge has to have Stream Control Transmission Protocol (SCTP) which may not come installed as default in edge devices such as Raspberry Pis or NVIDIA Jetsons. SCTP is extensively used in telecommunications networks, particularly in the implementation of the signaling plane in

[22] https://github.com/Neruzzz/docker_open5gs_2upf_cloud
[23] https://github.com/Neruzzz/docker_open5gs_2upf_edge

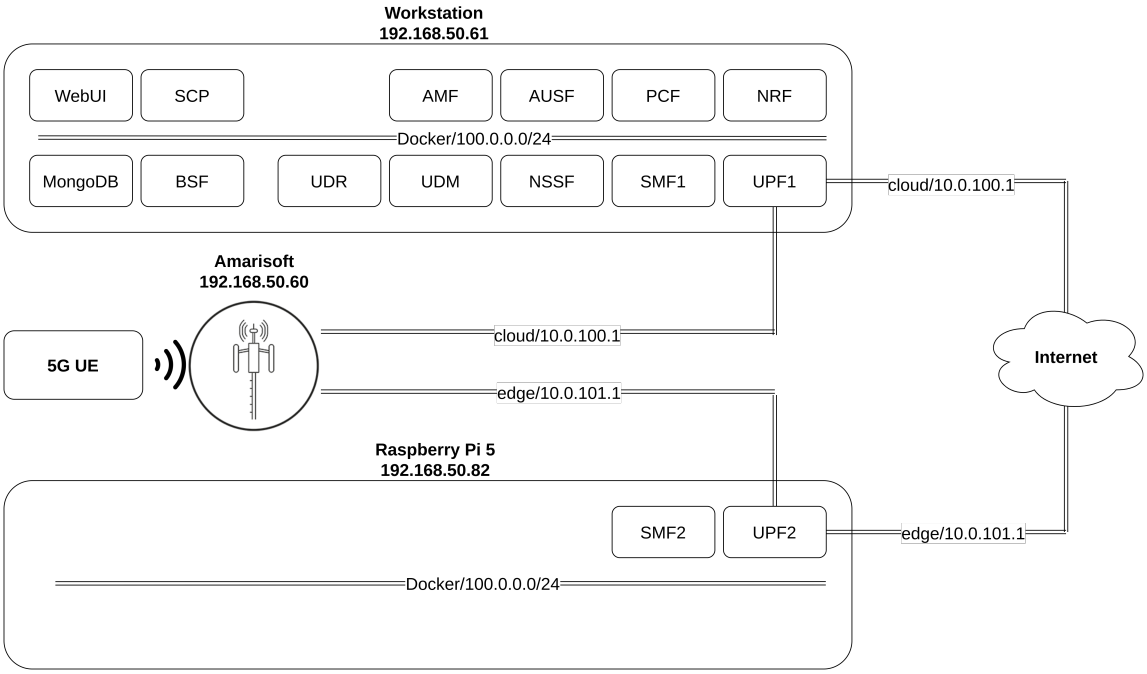


Fig. 4.9 Final testbed diagram

5G cores and systems like Open5GS. The base repository (branch *slices_2upf_2smf*) was first cloned on each of the devices, workstation for cloud and Raspberry Pi 5 or edge. Then the differences were applied on each of the projects as follows:

- Workstation or cloud (cloud repository)
  - Docker Compose file *sa-deploy.yaml*
    - The port of the SCP service is mapped to
      ‘‘`${DOCKER_HOST_IP}:7777:7777/tcp`’’
      so it is reachable from outside the host machine.
  - *.env* file
    - `DOKER_HOST_IP` is changed to the host IP. In this case 192.168.50.61.
    - Set up the desired configurations
  - Service configuration YAML files
    - Add `advertise: DOCKER_HOST_IP` into the `sbi` section of the SCP service.
  - Service initialization scripts

        * Add the `sed -i ...` command to the initialization script of the SCP so it is able to substitute the Docker host IP from the .env file to the configuration YAML file.

- Raspberry Pi 5 or edge (edge repository)
  - Docker Compose file *sa-deploy.yaml*
    * The `depends_on` services of SMF2 and UPF2 is deleted, since the depending on functions are not deployed in the edge except for the `smf2` dependency in UPF2 which is still deployed.
  - *.env* file
    * `DOKER_HOST_IP` is changed to the host IP. In this case 192.168.50.82.
    * `SCP_IP` and `NRF_IP` are set up to the IP of the Workstation (192.168.50.61) so the deployed functions (SMF and UPF) know where to send the requests.
    * Set up the desired configurations
  - Service config YAML files
    * Add `advertise: DOCKER_HOST_IP` into the `sbi` section of the SMF2 service
  - Service initialization script files
    * Add the `sed -i ...` command to the initialization script of the SMF so it is able to substitute the Docker host IP from the .env file to the configuration YAML file.

These changes are reflected on the respective repositories mentioned at the start of the section.

Then the names of the DNNs are changed from *internet* and *private* to *edge* and *cloud*, modifying the corresponding YAML files. Since this is a distributed deployment, the configuration specified in the README file of the *slices_2upf_2smf* branch (of the base GitHub) in section *Multihost setup configuration* inside *5G SA deployment* must be applied too. Finally, the Amarisoft gNB has the same configuration as explained in Section 4.1.2.

The SIM cards used for the UE are the Sysmocom sysmoISIM-SJA2. These SIM cards must be enabled to utilize multiple slices because they typically lock onto the first network slice they connect to and remain bound to it. To address this, the SIM cards were programmed using pySIM to access multiple slices and to select the desired slice through the configured APN on Samsung phones. This code can be found in the cloud repository mentioned at the start of the section. However, this functionality ceased to work following a phone update and

was also incompatible with the Quectel modem previously. As a resolution, two separate SIM cards were employed, one for cloud services and the other for the edge.

Finally, the UEs must be configured to connect to the two DNNs set up in the testbed. In the case of Samsung Galaxy A23 smartphones, multiple APNs can be configured through the settings menu. Entering the *Connections* section and then to *Mobile Networks*, followed by *Access Point Names*, new APNs can be added by specifying names and corresponding identifiers. For instance, in the scope of this project, *Cloud* with an APN named *cloud*, and *Edge* with *edge*. Each APN must be set with a type of *default, internet*, ensuring they serve standard internet services. Then, the user can select the APN to connect to, choosing in the list of configured APNs being able to switch seamlessly from edge to cloud.

In the case of the Quectel module, a similar setting has to be performed, in this case using AT commands [24]. After following this starter guide[25] which helps to connect the modem to a Raspberry Pi using the USB to M.2 B key and installing Minicom[26] to configure internet connections. This guide is the example of the low quality and scarce information on this topic that is mentioned in this thesis. It is incorrect in some parts and does not give any insight in the setting of Packet Data Protocol contexts (PDP contexts). These contexts are used to define and manage the data connection between a mobile device and a mobile network. Each PDP context corresponds to a specific set of configurations, such as IP type and APN, among others. The following set of AT commands performed using *minicom* would set up the needed PDP contexts and activate one of them. Activating a context means connecting to the specified APN of the context, in this case the edge APN. The information of all the AT commands used during the project can be found in the following guide[27] provided by Quectel, which also has some incompatibilities and errors.

```
AT+CGDCONT=1,"IP","cloud" # Configures IPv4 PDP context with cloud APN
AT+CGDCONT=2,"IP","edge"  # Configures IPv4 PDP context with edge APN

AT+CGACT=1,2              # Activates (1) the context with ID 2 (edge)
```

To verify if the testbed has an internet connection via the Quectel modem, the following steps were followed while connected to the edge APN. These procedures are equally applicable to the cloud APN. To avoid redundancy, they are not repeated here, but the result of the

[24] AT commands are a set of instructions used to control modems and other devices. They are essential for setting up communications, including the configuration of APNs.

[25] https://www.waveshare.com/wiki/USB_TO_M.2_B_KEY#Working_With_Raspberry_Pi

[26] Minicom is a text-based modem control and terminal emulation program for Unix-like operating systems, commonly used to send AT commands to modems and other serial devices.

[27] https://www.quectel.com/wp-content/uploads/2021/05/Quectel_RG50xQRM5xxQ_Series_AT_Commands_Manual_V1.2.pdf

test was successful for the cloud APN too. The first step is to deploy the entire testbed using the following commands:

```
# Cloud side inside the project folder
docker compose -f sa-two-slices-deploy.yaml up amf mongo webui scp nrf smf upf
    ausf udm udr pcf bsf nssf

# Edge side inside the project folder
docker compose -f sa-two-slices-deploy.yaml up upf2 smf2
```

The logs of the AMF in Figure 4.10 show how the AMF daemon is running, initialized, all the NFs have connected successfully, a working gNB is connected and, finally, a UE has registered and connected to the network on the edge APN. Then in Figure 4.11 the logs show that the SMF daemon is running, initialized, it is able to connect to all the needed NFs and has a UE session established that starts triggering service requests (requests for internet connectivity). From the UPFs side, Figure 4.12, shows how the UPF connects with all the needed NFs and gives a IPv4 address inside the defined IP range of the edge APN (10.0.101.0/24) to the connected device (UE). There are more NFs involved, but if these three work correctly, this shows that the core is working.

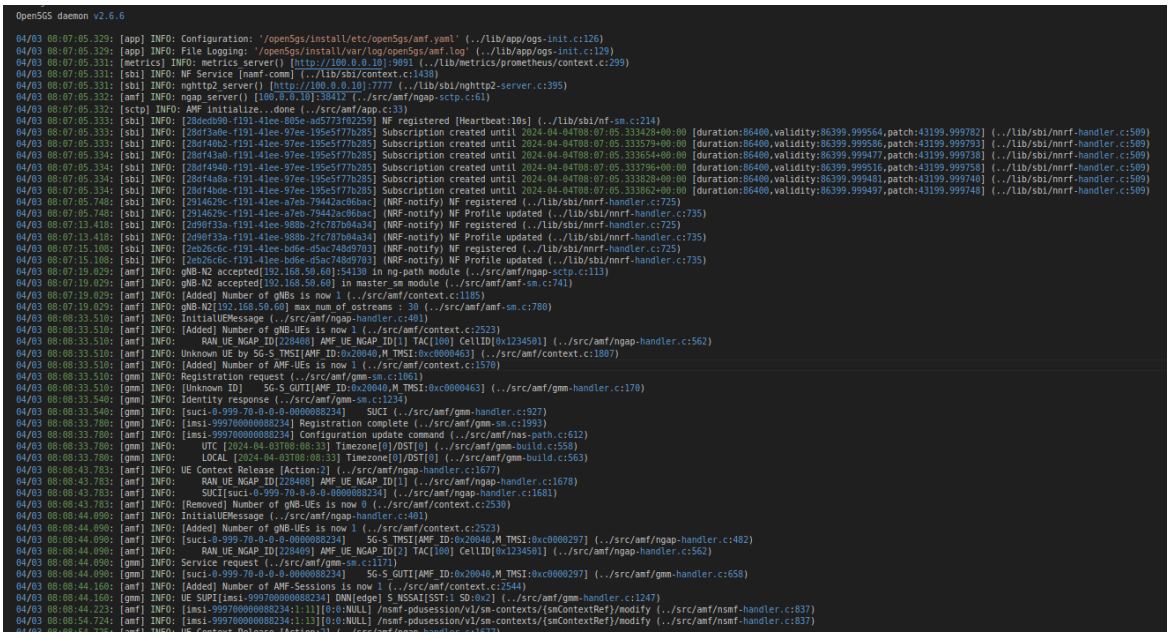


Fig. 4.10 AMF Open5GS logs showing correct functioning and UE registration.

There are cases where the 5G network works and the UE is connected and registered to it but the Internet connection does not work (DNS does not solve addresses, requests are

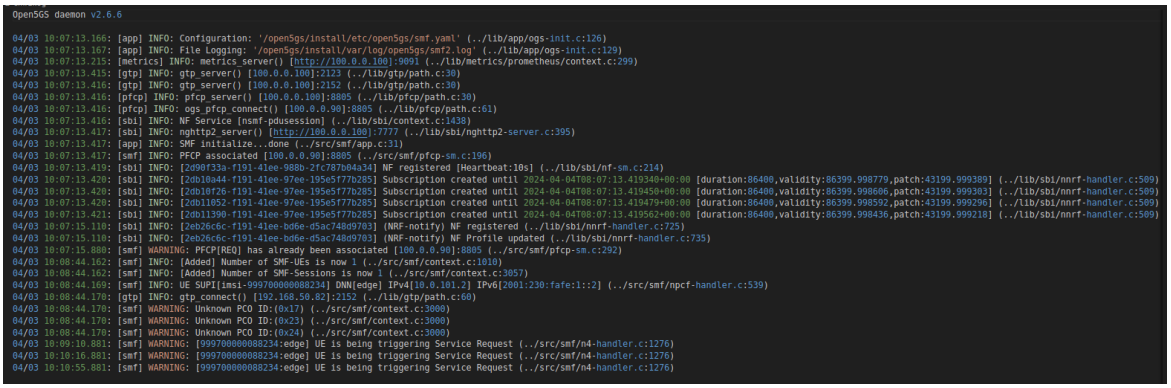


Fig. 4.11 SMF Open5GS logs showing correct functioning and UE session creation.

Fig. 4.12 UPF Open5GS logs showing correct initialization and UE given IP and 5G services using the edge APN.

not triggered, network fails to reach internet, etc.). To ensure full functionality, the internet connectivity from UE side is checked using `curl`, as in Figure 4.13. Notice that to ensure that the data connection passes from the 5G tunnel, instead of the available BSC's Ethernet network connecting the RAN and core components, the option `-interface` is used to force `curl` to use the modem's interface on the connection.

These steps ensure full functionality including internet connection from the UE side and enable the deployment of the applications.

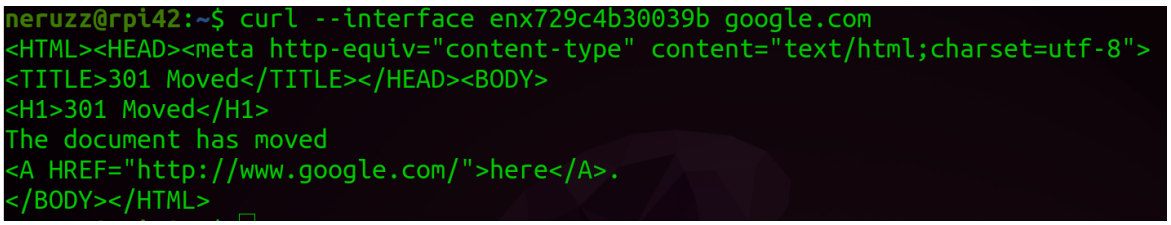


Fig. 4.13 Curl made from the UE to check internet connection.

Chapter 5

Metrics and performance evaluation

5.1 Cloud physical delay simulation and delay measurements

In this section, the focus is primarily on evaluating delay, specifically aiming to demonstrate the reduced physical propagation delay achieved through the project's strategic placement of the UPF in the edge. Since the testbed has been deployed within the lab premises, and several restrictions for accessing external networks apply due to security policies, both edge and cloud nodes were collocated within the same rack. Hence, to make an assessment of potential delay enhancements, the network interface of the workstation, acting as the cloud server, has been modified with the addition of a simulated delay on outgoing packets. This is done to make the differences in delay noticeable between the edge and the cloud since both elements are inside the same rack.

To simulate a theoretical scenario, the value of the delay has been calculated having as a reference point a cloud datacenter located in Ireland. Typically, a UPF for conventional telecommunication operators would be strategically located in regions with extensive network infrastructure, such as major European data centers in big cities to reduce delay and enhance all the 5G network capabilities. The purpose behind choosing Ireland is to exaggerate the delay so it is noticeable on the experiments yet realistic in the sense that Ireland could have its own UPF.

Considering the distance between Barcelona and Dublin, approximately 1500 km, we calculate the propagation delay to simulate network conditions accurately. The formula used to calculate the one-way propagation delay in milliseconds is:

$$\text{Propagation Delay} = \frac{\text{Distance (km)}}{\text{Speed of Light in Fiber (km/s)}} \times 1000$$

Given that the speed of light in fiber is approximately 67% of its speed in vacuum (which is about 299.792 km/s), the speed in fiber optic cables is roughly 200.761 km/s [31]. Thus, for a distance of 1500 km, the one-way propagation delay is:

$$\text{Propagation Delay} = \frac{1500}{200.761} \times 1000 \approx 7,47\,\text{ms}$$

Since the delay is applied only to outgoing packets, it is necessary to double this value to accurately simulate the Round-Trip Time (RTT) in ping measurements, accounting for both the trip to the destination and the return. Therefore, the total delay to be applied is approximately 15 ms.

To simulate this delay in a realistic network environment, the following *tc* (Traffic Control) command from the Linux *iproute2*[1] package can be used to apply the calculated delay to the workstation's network interface:

```
1 sudo tc qdisc add dev eno1 root netem delay 15ms
```

This command introduces a delay of 15 ms to the *eno1* (Ethernet) interface of the workstation, effectively simulating the network conditions between Barcelona and Dublin. Notice that this is only the propagation delay and that normally a route to a cloud would introduce several other delays that would make this number to grow. Since this is an experimental environment, this will be enough to notice the differences on the delay measurements between cloud routes and edge routes.

Several delay measurements are carried out, *pinging* different parts of the network to cover all possible paths. The diagram that will be used to describe these paths can be seen in Figure 5.1. The tunnels depicted are the links between the UPFs and the gNB. These connections are also physically established using Ethernet, but the internal link is a virtual tunnel that uses GTP (*ogstun* interface on the UPF side). In this case no path needs to go through the Internet since all nodes of the network are on the same rack as explained in section 3.4.

The first analyzed path, depicted in Figure 5.2a, is the one that goes from the UE to the cloud or from cloud to the UE, using the cloud APN. This means that the requests that go from or to the UE will be routed by the UPF in the cloud (UPF1). This path is expected to be affected by the simulated propagation delay to reach the cloud. The downlink path should have a similar delay compared to the uplink path.

The second analyzed path, depicted in Figure 5.2b, corresponds to the route that goes from the UE to the edge or from edge to the UE, using the edge APN. This means that the

[1] https://manpages.debian.org/unstable/iproute2/tc.8.en.html

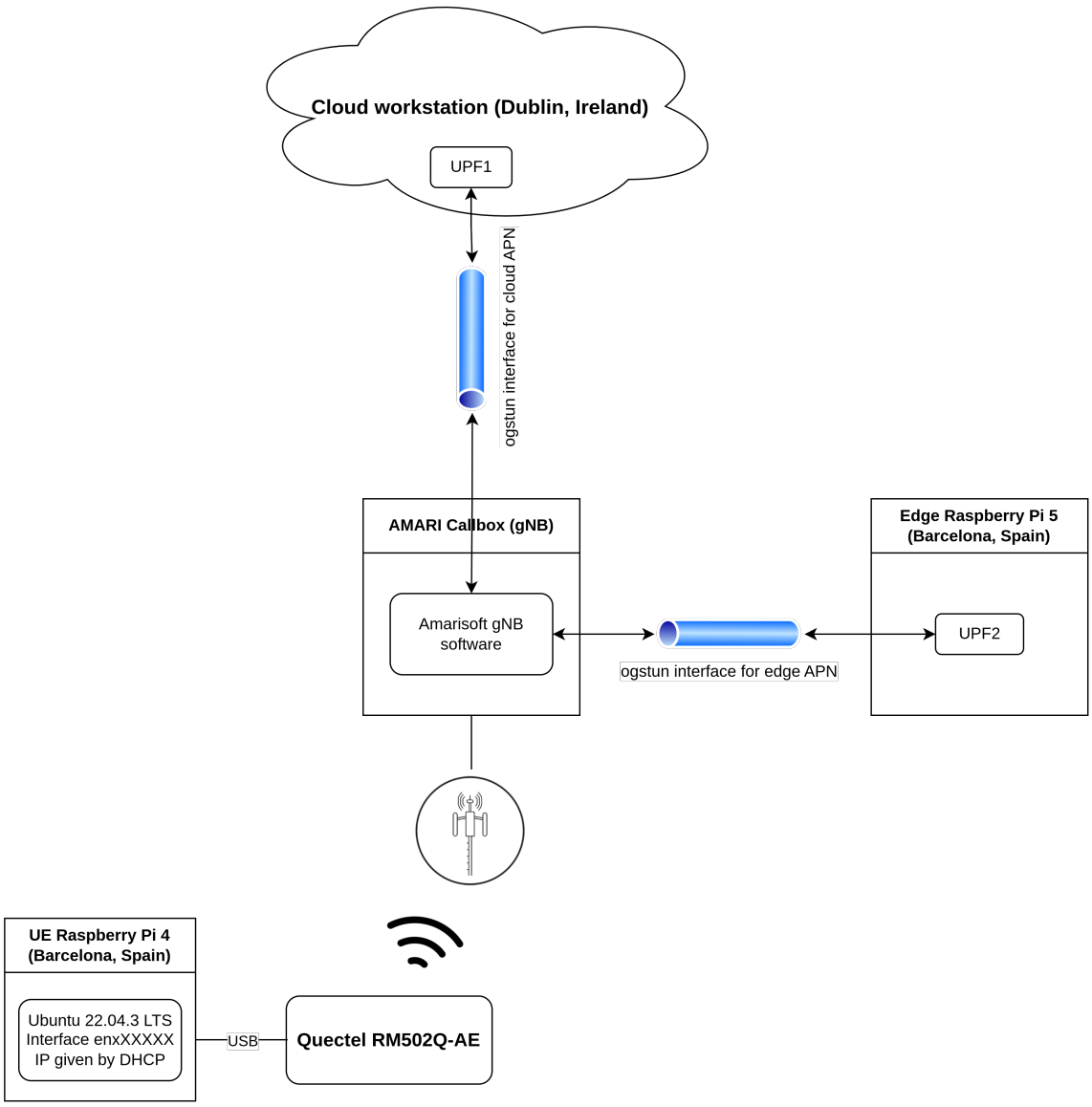
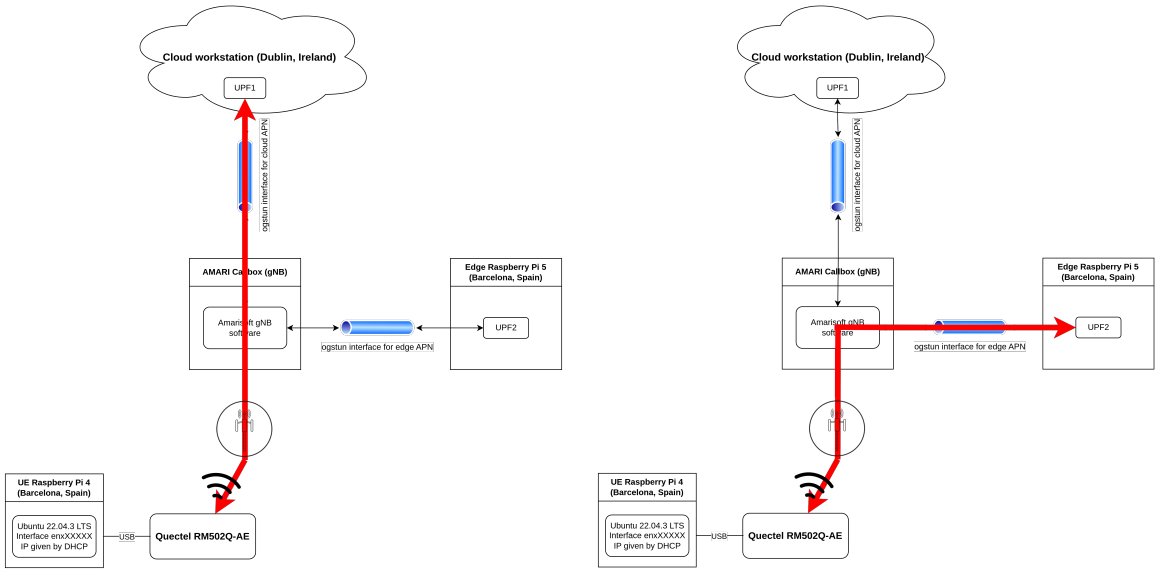


Fig. 5.1 Testbed architecture

requests that go from or to the UE will be routed by the UPF in the edge (UPF2). The delay on this path is expected to be smaller than the one described in Figure 5.2a. This happens because both the endpoint of the ping and the UPF that routes the requests of this path are close to the source of the request (both located in Barcelona). As in the last scenario, the uplink and downlink paths are expected to have similar delays.

The last two paths tested are the furthest end-to-end points of the network for each of the APNs. The first case, depicted in Figure 5.3a is the longest path for the UE to ping the cloud since it has to arrive to it going to the UPF in the edge on the first place. The worst



(a) UE to cloud or cloud to UE using cloud APN (routed from the cloud)

(b) UE to edge or edge to UE using edge APN (routed from the edge)

Fig. 5.2 Comparison of communication paths using cloud and edge APN routing

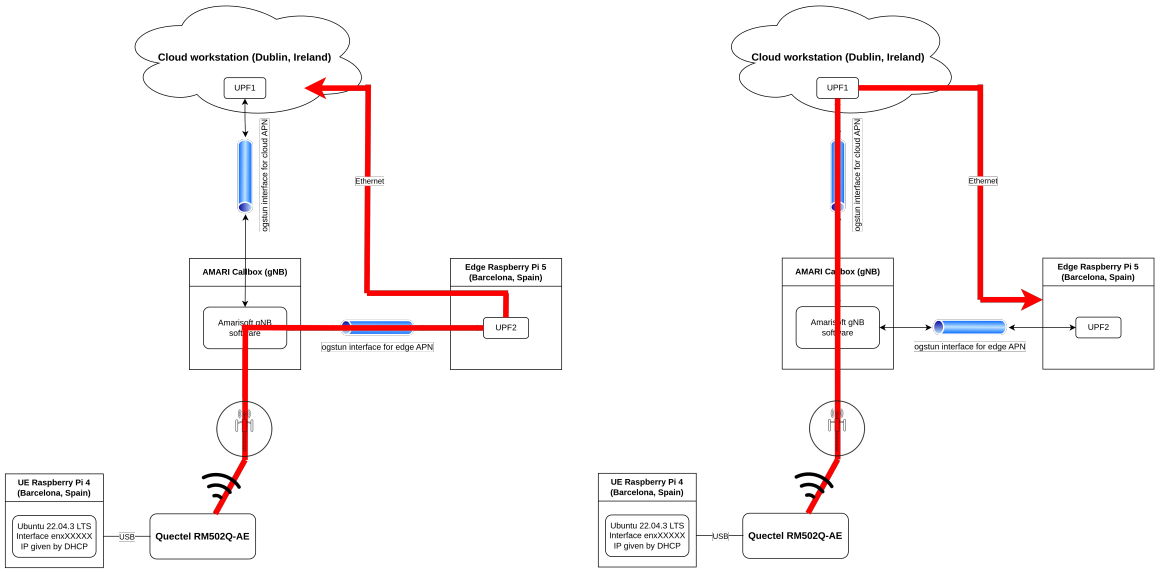
case scenario is depicted in Figure 5.3b since the UE has to ping the edge which is located near, using the furthest point of the network as router which is the UPF on the cloud. These are examples of routes that serve as a demonstration of the worst case scenarios inside the network. This last two paths are expected to have more delay than the ones that use the same APN (Figures 5.2b and 5.2a respectively), but the scenario in Figure 5.3b is expected to have the highest delay.

Table 5.1 has the results of the 250 pings (250s, 1 ping per second) done for each of the paths described above. These results are as expected and as described before.

APN Type	**Route**	**Mean (ms)**
Cloud APN	Cloud to UE	70,966
	UE to Cloud	72,837
	UE to Edge (routed from Cloud)	92,477
Edge APN	Edge to UE	27,47
	UE to Edge	28
	UE to Cloud (routed from Edge)	44,458

Table 5.1 Results of the delay tests using `ping`.

To see the results in an ordered manner the rank of the paths based on their delays, starting from the least to the greatest, is presented below:



(a) UE to cloud using edge APN (routed from the edge)

(b) UE to edge using cloud APN (routed from the cloud)

Fig. 5.3 Worst case scenario experimental paths for edge and cloud APNs

1. Edge to UE using edge APN (27,47 ms).
2. UE to edge using edge APN (28 ms).
3. UE to cloud using edge APN (44,458 ms).
4. Cloud to UE using cloud APN (70,966 ms).
5. UE to cloud using cloud APN (72,837 ms).
6. UE to edge using cloud APN (92,477 ms).

Notice how the cloud APN has more delay in all paths. The paths that are complementary (uplink and downlink) are similar, being the downlink the one with less delay in both APNs. Finally, the path with the most delay is the longest, as expected and explained during the section. Regarding the stability of the communication paths, the next sections give an insight on what is the throughput and the jitter of the channels measured in some of these paths.

5.2 Throughput and jitter metrics discussion

During the evaluation of this thesis, an attempt was made to measure throughput and jitter while simulating delay as discussed in the previous section. However, this approach faced

significant challenges. The use of the `tc` command to simulate delay altered the channel conditions drastically, resulting in the receiver part of the `iperf` command losing approximately 40% of its packets. In some instances, `iperf` ceased functioning altogether, the sender could not establish a connection with the receiver due to the artificial conditions imposed on the channel. This issue was first observed during initial measurements of throughput, and the problematic outcomes are depicted in Figure 5.4.

```
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  5]   0.00-250.00 sec  31.3 MBytes  1.05 Mbits/sec  0.000 ms  0/23440 (0%)  sender
[  5]   0.00-263.88 sec  18.5 MBytes   589 Kbits/sec  5.066 ms  9540/23438 (41%)  receiver

iperf Done.
```

Fig. 5.4 First measurements using `iperf3` with the simulated delay using `tc` applied in the cloud.

Consequently, the delay simulation was removed for subsequent measurements to ensure communication stability between the server and the client. The measurements taken are only for the paths seen in Figure 5.2a and Figure 5.2b since these are the shortest paths that traffic can traverse. The measurements displayed in Table 5.2 reflect scenarios without the simulated delay. It is important to note that `iperf3` sets the maximum bandwidth to 1 Mbps as default. As this setup has all network components were located within the same rack, the tests provided stable results.

APN	**Sender**		**Receiver**		
	Transfer	**Bitrate**	**Transfer**	**Bitrate**	**Jitter**
Cloud	31,3 MBytes	1,05 Mbits/s	31,2 MBytes	1,05 Mbits/s	1,782 ms
Edge	31,3 MBytes	1,05 Mbits/s	31,3 MBytes	1,05 Mbits/s	2,386 ms

Table 5.2 Summary of transfer, bitrate, and jitter values for Sender and Receiver.

In real-world cloud deployments, several factors interact in complex ways to impact network performance metrics. This interaction is especially critical in scenarios where UE communicates with the cloud. The extended path and varied network conditions can introduce greater variability and pose significant challenges. The key factors influencing throughput, jitter, and overall network delay are detailed below:

- **Increased physical distance** leads to longer propagation times, directly influencing delay, which in turn affects throughput and jitter due to time-sensitive data flows.
- **Routing and switching delays** impact latency and can cause jitter when packet arrival times vary.

- **Queueing delays** at network nodes affect both throughput, by slowing down data transmission, and jitter, by introducing variability in packet handling times.
- **Server processing time** influences delay directly and can reduce throughput during high demand periods, affecting the consistency of data transmission (jitter).
- **Protocol overhead** increases the total time taken for data to be secured and transmitted, impacting all three metrics.

These observations underscore the need for future work to conduct measurements in a real scenario with an actual cloud deployment where these factors are naturally present. Such studies would provide more accurate and realistic assessments of network performance, offering valuable insights to further support this thesis hypothesis. Furthermore, the factors listed previously would only exacerbate the delays in a real cloud deployment, thus, the initial delay measurements obtained in this thesis remain valid and something similar should happen to throughput and jitter.

These results correctly demonstrate the inherent advantages of deploying the UPF in the edge over having all the 5G CN in a cloud in terms of reduced latency. This accentuates the thesis's core argument that edge computing offers superior performance in terms of lower latency compared to traditional cloud setups. Thus, even though real-world conditions may introduce additional complexities, they are likely to further validate the thesis's hypothesis by showing an even greater disparity in performance metrics between the cloud and the edge. This expected outcome reinforces the conclusion that edge computing is crucial for applications requiring real-time processing and responsiveness.

Chapter 6

Testbed applications

6.1 Messaging service

This application involves deploying an MQTT broker at the edge. The UE is tasked with transmitting messages to a designated topic in the edge's broker using the 5G connection. The objective of this first application is to prove the viability of running services on the edge, where a UPF is deployed, and the app is hosted.

The edge MQTT broker is defined on the same docker compose as all the other edge NFs. The YAML format code to add to the docker compose file is the following:

```yaml
version: '3'

services:

  mqtt:
    image: eclipse-mosquitto:latest
    container_name: mqttserver
    env_file:
      - .env
    ports:
      - "1883:1883"
    volumes:
      - ./mqtt/mosquitto.conf:/mosquitto/config/mosquitto.conf

    networks:
      default:
        ipv4_address: ${MQTT_IP}
```

This creates a container with the official *eclipse-mosquitto* image and with the MQTT port (1883) mapped. The container named *mqttserver* is inside the docker network that the edge UPF and SMF share. The configuration file specifies the following

```
listener 1883 0.0.0.0
allow_anonymous true
```

These configuration lines set up the broker to listen on port 1883, the default port for unencrypted MQTT communication, and accept connections from any IP address. Additionally, the `allow_anonymous true` setting permits clients to connect without authentication.

To deploy the broker the command used is the same as the used on the core deployment:

```
docker compose -f sa-two-slices-deploy.yaml up mqtt
```

Figure 6.1 shows the response to the `docker ps` command, which lists the running containers on the machine. As seen in the image, after the last command, the MQTT broker is up and running on the edge machine.

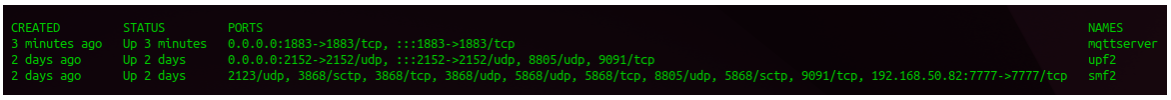


Fig. 6.1 Response to the command `docker ps` after deploying the MQTT broker.

Now, on the UE side, the MQTT is installed using the following set of commands:

```
sudo apt-get update
sudo apt-get install mosquitto-clients
```

Once the client is installed and before sending any message, a route rule has to be added, so the messages sent to the broker with IP 192.168.50.82 (edge Raspberry Pi) are sent through the interface provided by the modem connected to the Raspberry. This is done with the following command:

```
sudo ip route add 192.168.50.82 via 192.168.225.25 dev enx729c4b30039b
```

The last step is to subscribe to a topic on the UE or on the edge and send a message to this topic from the UE so it is seen by the subscriber. This is done with the following commands:

```
# Command to subscribe to the topic. In this case, it is executed on the edge.
mosquitto_sub -h localhost -t "test/topic"
# Command to publish to the topic
mosquitto_pub -h 192.168.50.82 -t "test/topic" -m "Hello world over 5G"
```

Figure 6.2 is a screenshot of the terminals of the UE Raspberry Pi (above) and the edge Raspberry Pi (below). The first shows how the route is written to the routing tables of the UE and how the publish command executes without error. On the other end, the edge device subscribes to the topic and receives the message sent by the UE. This can be set up so the communication is downlink (edge to UE). This demonstrates that the application works.

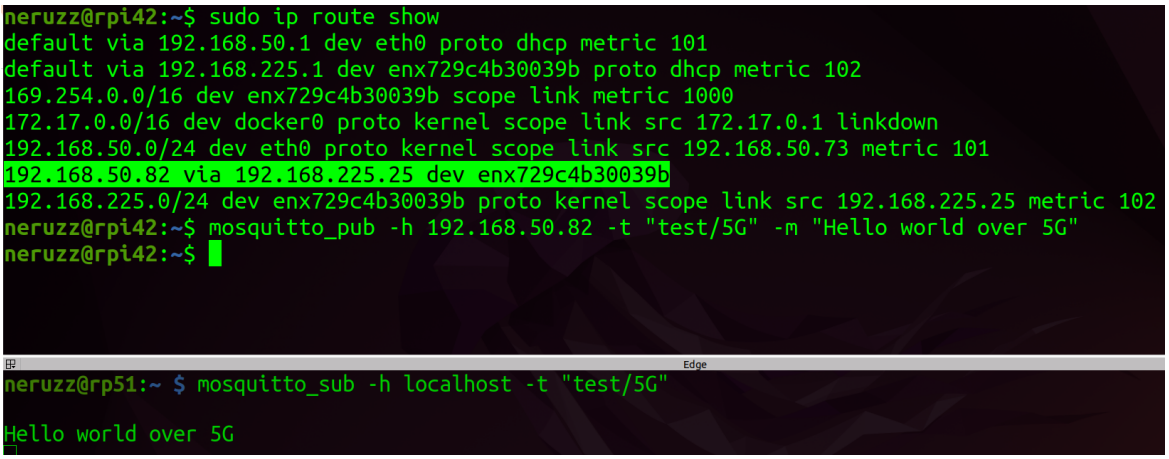


Fig. 6.2 MQTT application working. The message sent by the UE is received on the edge.

Finally, to demonstrate that the message is routed through the 5G modem and thus, it arrives to the desired UPF in the edge, a `tcpdump` is performed inside the UPF. This is done executing the following commands in the edge device:

```
# The following command enters the shell of the UPF container
docker exec -it upf2 bash
# Inside this shell the tcpdump is performed to show the TCP traffic
tcpdump -i ogstun
```

The first command enters the shell of the UPF container, the second one performs the `tcpdump`. Notice that the `tcpdump` is performed on the *ogstun* interface. The *ogstun* interface is the tunnel interface that communicates the registered UEs to the UPF of the data plane they are subscribed to, in this case the edge. This interface has the IP 10.0.101.1 on the UPF side, as shown in Figure 6.3. Figure 6.4 demonstrates that the MQTT packets that use the

port 1883 and are seen on the image, are routed through the UPF in the edge using the *ogstun* interface and thus, they use 5G to go through the network. This test ends successfully as an application has been successfully deployed on the edge of the network and 5G has been used to send a MQTT message from a UE connected to the 5G network to the MQTT broker on the edge of the same network using 5G.

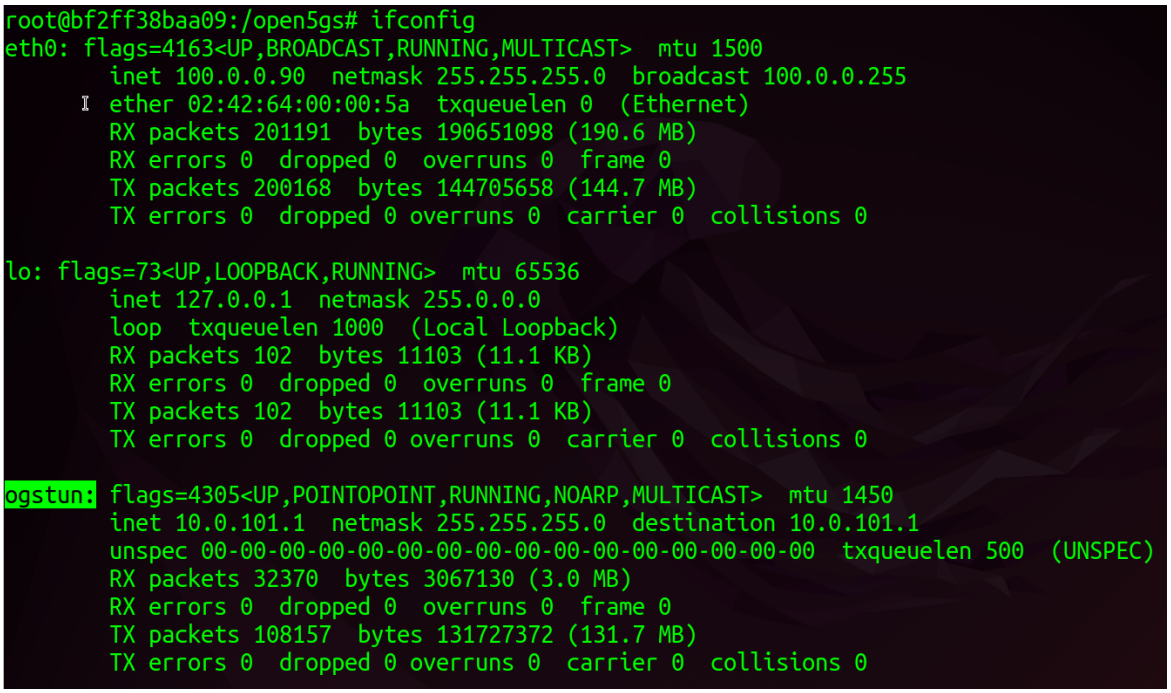


Fig. 6.3 *ogstun* interface IP shown on the edge UPF container shell.

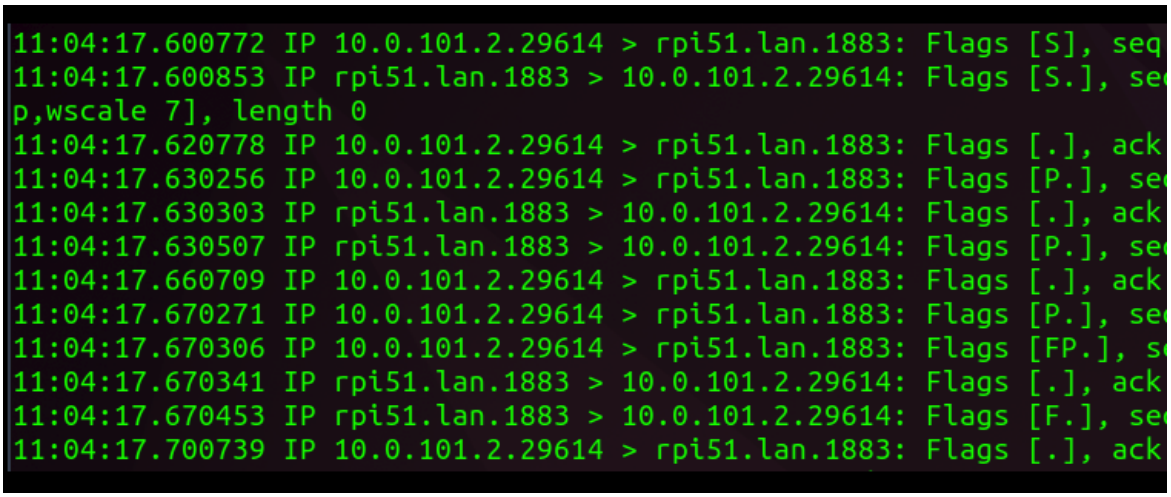


Fig. 6.4 `tcpdump` shows that the *ogstun* interface routes all the MQTT messages. These messages are distinctive because they use port 1883.

6.2 Video streaming service

This application demonstrates the viability of hosting real-time services at the network's edge, ensuring robust QoS. For this purpose, a streaming service using VLC is implemented at the edge. The UE is responsible for receiving this stream and displaying it using the 5G connection.

The routing command used on the first application has to be executed so, again, all the interactions from UE to edge are routed through the 5G modem interface on the UE. Then, the streaming service must be created at the edge so it can be deployed on a container. This is done in two steps. The first step is to create the docker image of the streaming service and download the file that is going to be streamed. In this case, the film that is streamed is *12 angry men*, an incredibly good and famous film from the 50s. The image is created using a Dockerfile with the following code:

```
FROM debian:bullseye-slim

# Update the packages and install VLC without recommended packages
RUN apt-get update && apt-get install -y --no-install-recommends \
    vlc-bin vlc-plugin-base && \
    rm -rf /var/lib/apt/lists/*

# Create a non-root user named 'vlcuser' and set the working directory
RUN useradd -m vlcuser
WORKDIR /home/vlcuser

# Switch to the non-root user
USER vlcuser

# Expose port 8080, which VLC will use to stream the video.
EXPOSE 8080

# Use the interface-less version of VLC to start streaming when the container runs.
CMD ["cvlc", "-vvv", "/mnt/streaming/12AngryMen.mp4", "--sout",
"#transcode{vcodec=h264,acodec=mpga,vb=800,ab=128}:
standard{access=http,mux=ts,dst=:8080}",
"--no-sout-all", "--sout-keep"]
```

As explained in the comments, the Docker image installs VLC player on a Debian distribution and then creates a non-root user to be able to execute the `cvlc` command, which is unusable inside a container as root (containers always execute as root and so a user has to

be created). Then the CMD command runs automatically the stream in *localhost:8080* using HTTP protocol as soon as the container is executed. The second step is to create the service on the compose file, as done in the MQTT example of the last section.

```
version: '3'

services:
  streaming:
    build: ./streaming
    image: streaming
    container_name: streaming
    env_file:
      - .env
    volumes:
      - ./streaming:/mnt/streaming
      # - /etc/timezone:/etc/timezone:ro
      # - /etc/localtime:/etc/localtime:ro
    expose:
      - "8080/tcp"
    ports:
      - "8080:8080/tcp"
    networks:
      default:
        ipv4_address: ${STREAMING_IP}
```

This service builds the Dockerfile, creates a container named *streaming* and exposes and maps the port 8080 to reach the streaming from outside. Once again, the service is deployed using the following command inside the folder of the project on the edge device:

```
docker compose -f sa-two-slices-deploy.yaml up streaming
```

Now, `docker ps` shows what can be seen in Figure 6.5. The service is running and working. The UE has to be able to request it using `curl`. This test's output is shown in Figure 6.6 where the following curl command is performed:

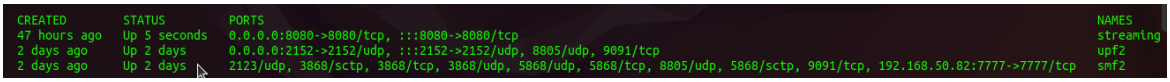


Fig. 6.5 Streaming service is deployed successfully.

```
# The option --output - allows dumping all binary data onto the terminal as text.
# The IP on the command is the edge device IP
curl --interface enx729c4b30039b http://192.168.50.82:8080 --output -
```

On the same image (Figure 6.6) a `tcpdump` of the *ogstun* interface is performed as it was done on the MQTT application to confirm that the data of the streaming that arrives to the user is going through the 5G modem interface and tunnel with the UPF located on the edge.

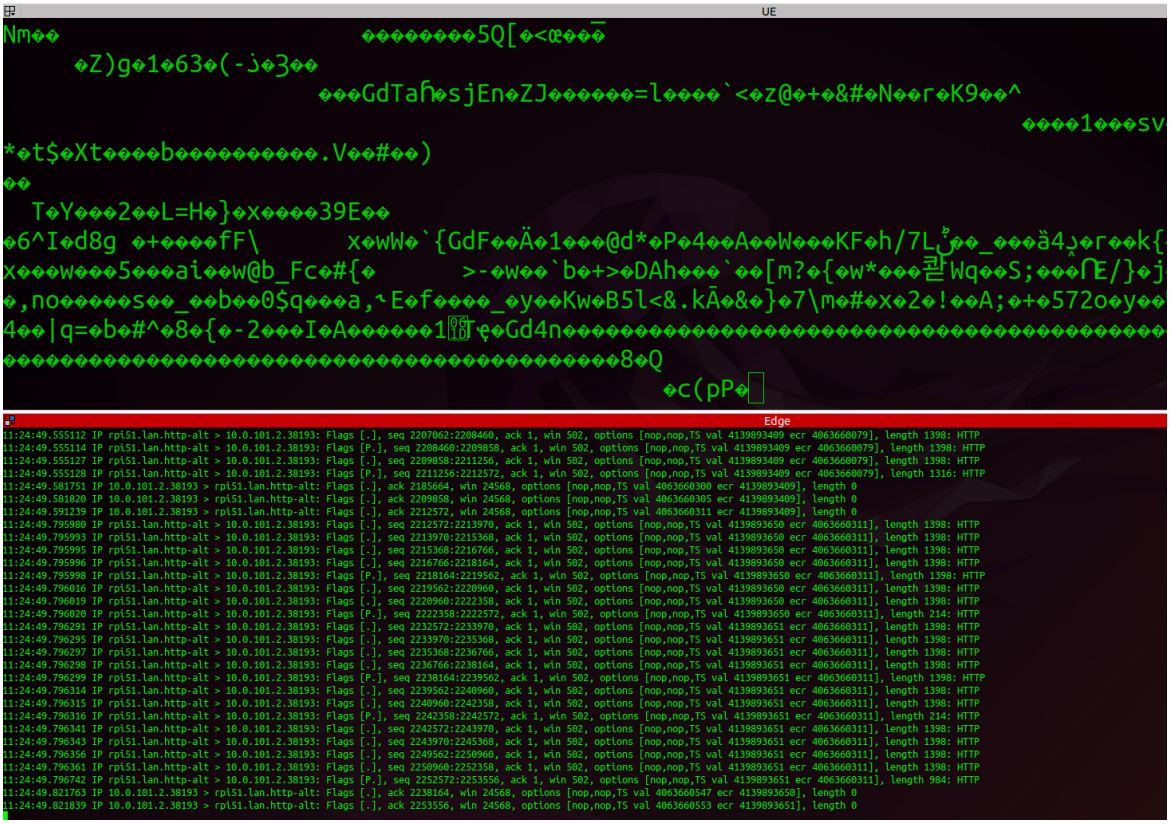


Fig. 6.6 Binary contents from the streaming are outputted using `curl` on the UE (above) and `tcpdump` in the edge UPF shows HTTP traffic from the streaming going to the UE (below).

The last thing to be done is watch the streaming from the UE side using, for example, VLC. The streaming can be played using VLC from the *Open network stream* tab located in *Media > Open network stream*. Then the stream should be visible at *http://192.168.50.82:8080* as shown in Figure 6.7 and Figure 6.8. The streaming will be requested using the interface given by he 5G modem thanks to the `ip route` rule configured at the beginning of the deployment.

This test also finished with a satisfactory outcome, since the application worked on the edge, the stream was visible from the UE and all the data transmission was done using the deployed 5G network. The streaming can be optimized if transmitted using RTP protocol

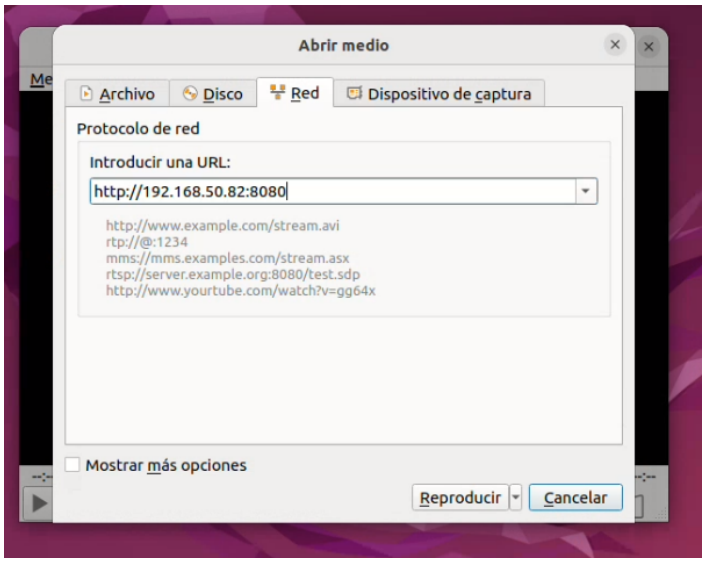


Fig. 6.7 VLC network streaming configuration.

Fig. 6.8 Streaming works on the UE side.

instead of HTTP, but the main purpose of this scenario was to demonstrate that delay-sensible applications such as video streaming can be deployed on the network's edge, and they work with good QoS.

6.3 Real time object detection service

This app implements a use case where a video recorded at the UE is streamed at real-time to the edge, where a High Performance Computing (HPC) object detection algorithm is applied. The annotated video generated as the output of the object detection can then be viewed at the edge and could be sent to other parts of the network. The objectives of this app are:

- Optimize the streaming quality using Real-time Transport Protocol (RTP)[1] protocol instead of HTTP. This will reduce the delay of the streaming.
- Demonstrate how computational heavy services that cannot be supported by the UE can be offloaded to the edge so they can be computed in real-time.
- Demonstrate the capability of the testbed of hosting an application that could be used on a real case scenario, such as traffic object detection.

The application applies object detection and takes advantage of the streaming management capabilities of using the project *jetson-inference*, which can be found in this GitHub repository[2]. The *jetson-inference* GitHub repository is an instructional toolkit for implementing real-time vision with deep learning on NVIDIA Jetson devices.

In this application, a Jetson Nano is deployed as the UE[3]. The Jetson Nano is connected to a Quectel modem responsible for transmitting video streams using RTP to the edge. The edge consists of a Raspberry Pi 5, which handles UPF and SMF services, and a Jetson AGX connected via Ethernet to the Raspberry Pi, tasked with executing AI model inference on the received video stream. All this is depicted in the diagram of Figure 6.9.

Initially, the Jetson Nano was configured to use the Quectel modem by following a detailed tutorial, which is accessible at this link[4]. This application involves establishing a network route using the *usb1* interface, created by the modem on the Jetson Nano, to communicate with the Jetson AGX. The routing command is below, and its output is depicted in Figure 6.10. Figure 6.11 depicts the response to the `lsusb` command, which lists all USB devices connected to the Jetson Nano. The figure shows the Intel RealSense[5] camera and the Quectel modem, which does not appear with any device name.

[1] RTP, defined in RFC 3550, provides end-to-end network transport functions suitable for applications transmitting real-time data, such as audio, video or simulation data, over multicast or unicast network services.

[2] https://github.com/dusty-nv/jetson-inference/tree/master

[3] A Jetson Nano is used as UE to avoid compatibility issues at the streaming endpoints. The *jetson-inference* manages the RTP streaming layer seamlessly and is specifically designed for use on Jetson devices, thus excluding the Raspberry Pi and Samsung Galaxy phone from this application.

[4] https://www.waveshare.com/wiki/USB_TO_M.2_B_KEY#Working_With_Jetson_Nano

[5] https://www.intelrealsense.com/depth-camera-d405/

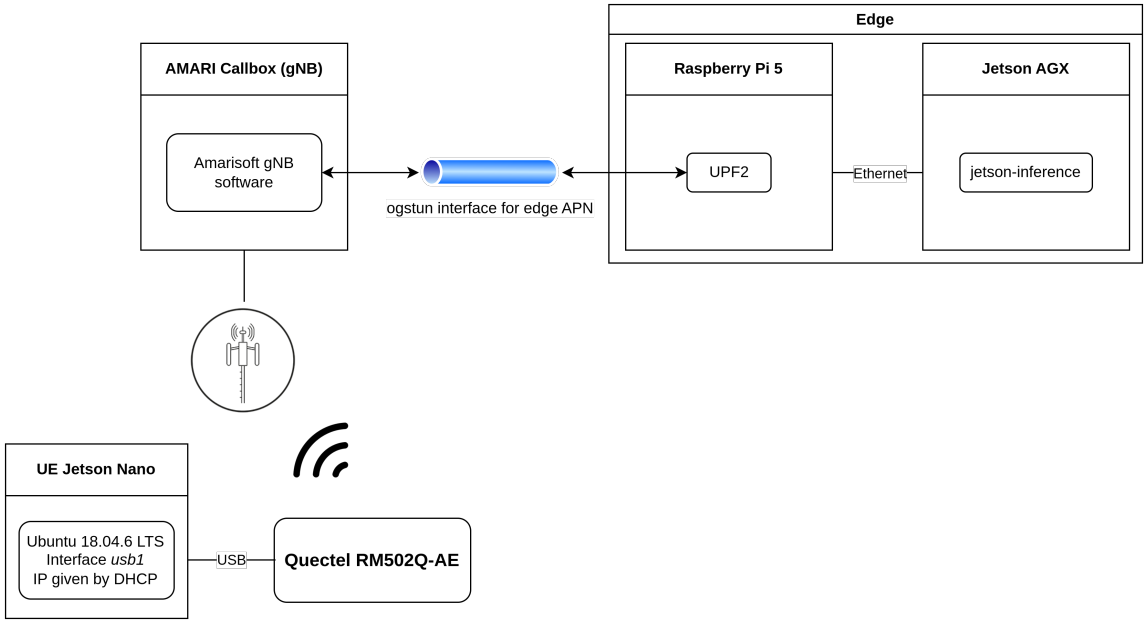


Fig. 6.9 Diagram of the architecture behind the application scenario

```
# Command to add a route from UE to Jetson AGX using usb1 as interface
sudo ip route add 192.168.50.12/32 dev usb1
```

```
neruzz@nano1:~$ ip route
default via 192.168.225.1 dev usb1
default via 192.168.50.1 dev eth0 proto dhcp metric 100
169.254.0.0/16 dev eth0 scope link metric 1000
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1 linkdown
172.18.0.0/16 dev docker_gwbridge proto kernel scope link src 172.18.0.1 linkdown
172.19.0.0/16 dev br-e8c92d16215d proto kernel scope link src 172.19.0.1 linkdown
192.168.50.0/24 dev eth0 proto kernel scope link src 192.168.50.32 metric 100
192.168.50.12 dev usb1 scope link
192.168.225.0/24 dev usb1 proto kernel scope link src 192.168.225.54
```

Fig. 6.10 Routes in Jetson Nano after adding the route to communicate with Jetson AGX using the *usb1* interface

Next, the *jetson-inference* Docker container has to be run in the Jetson Nano following the GitHub guide on this link[6]. The container execution has to show that the camera is detected, or it will not be able to read its output. This is done by showing all the video devices located in the host's */dev/* folder. These devices are mounted as volumes on the container so they can be used and are shown in Figure 6.12. The camera connected to the UE is recognized as

[6]https://github.com/dusty-nv/jetson-inference/blob/master/docs/aux-docker.md

```
neruzz@nano1:~/jetson-inference$ lsusb
Bus 002 Device 004: ID 8086:0b5b Intel Corp.
Bus 002 Device 003: ID 2c7c:0800
Bus 002 Device 002: ID 0bda:0411 Realtek Semiconductor Corp.
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 002: ID 0bda:5411 Realtek Semiconductor Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Fig. 6.11 Output of the `lsusb` command on the Jetson Nano shows that the camera and the modem are connected to the USB ports.

multiple video devices because it supports various capabilities. For example, it can capture standard video streams as well as depth information and other data types, each accessible through different video device entries such as *video0*, *video1*, etc.,

```
neruzz@nano1:~/jetson-inference$ docker/run.sh
ARCH:  aarch64
reading L4T version from /etc/nv_tegra_release
L4T BSP Version:  L4T R32.7.4
CONTAINER_IMAGE:  dustynv/jetson-inference:r32.7.1
DATA_VOLUME:      --volume /home/neruzz/jetson-inference/data:/jetson-inference/data
ta --volume /home/neruzz/jetson-inference/python/training/classification/models:/jet
jetson-inference/python/training/detection/ssd/data --volume /home/neruzz/jetson-inf
-inference/python/www/recognizer/data:/jetson-inference/python/www/recognizer/data
V4L2_DEVICES:     --device /dev/video0  --device /dev/video1  --device /dev/video2
root@nano1:/jetson-inference#
```

Fig. 6.12 Different video devices of the camera connected to the UE are mounted on the *jetson-inference* container

Inside the container, the raw camera footage can be visualized using the `video-viewer` command. The command is below and Figure 6.13 is a screenshot of the Jetson Nano screen after executing the command. This command shows the GStreamer[7] pop up window with the raw video of the camera. GStreamer is a versatile multimedia framework that supports a wide range of applications for handling and streaming audio and video. It is designed as a series of plugins that can be connected in a pipeline to perform complex operations on multimedia data. The *jetson-inference* repository uses GStreamer for all the video and streaming managing and makes the streaming configuration easy as the pipelines are managed by the container automatically, avoiding errors and incompatibilities.

```
1 video-viewer /dev/video2
```

[7] https://gstreamer.freedesktop.org/

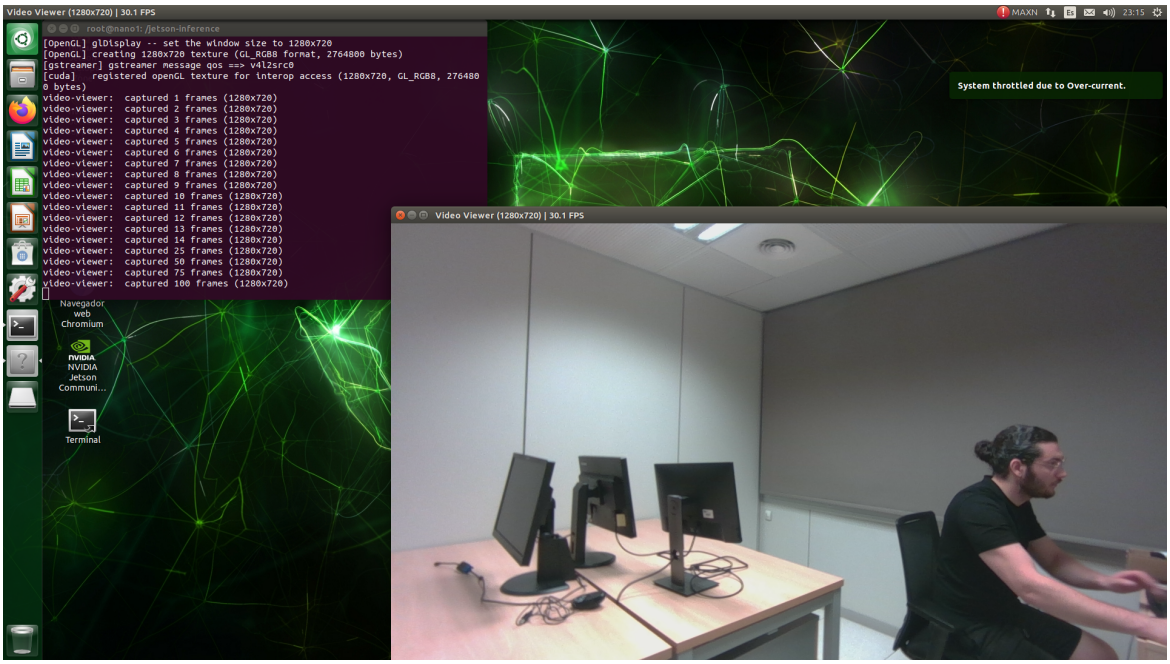


Fig. 6.13 Screenshot of the UE screen that shows the GStreamer pop up window with the raw footage of the video camera connected to it

To send the video from the Jetson Nano to the Jetson AGX, *video-viewer* is used with the output part of the command. The streaming uses RTP protocol and is sent to the port 1234 of the Jetson AGX. The command used is below and the output of the command is depicted in Figure 6.14.

```
# The command gets the camera as input and outputs an RTP streaming to the AGX
video-viewer /dev/video2 rtp://agx11:1234
```

The tool `tcpdump` is used to check that the streaming arrives tho the Jetson AGX in the edge. The command is below, and its output can be seen in Figure 6.15.

```
tcpdump -i any port 1234
```

Another check using `tcpdump` on the UPFs *ogstun* interface, demonstrates that the stream traffic goes through the GTP tunnel (ogstun is the edge end of this tunnel) and thus, the application is using the 5G network to communicate UE to edge. This test is depicted in Figure 6.16.

Finally, the video streaming is computed using *detectnet* in the container. With the *detectnet* deep neural network, objects are located by extracting their bounding boxes inside

```
[gstreamer] gstreamer stream status ENTER ==> src
NvMMLiteBlockCreate : Block : BlockType = 4
video-viewer:  captured 2 frames (1280x720)
H264: Profile = 66, Level = 40
NVMEDIA_ENC: bBlitMode is set to TRUE
video-viewer:  captured 3 frames (1280x720)
[gstreamer] gstreamer message stream-start ==> pipeline1
[gstreamer] gstreamer changed state from READY to PAUSED ==> udpsink0
[gstreamer] gstreamer message async-done ==> pipeline1
[gstreamer] gstreamer changed state from PAUSED to PLAYING ==> udpsink0
[gstreamer] gstreamer changed state from PAUSED to PLAYING ==> pipeline1
video-viewer:  captured 4 frames (1280x720)
video-viewer:  captured 5 frames (1280x720)
video-viewer:  captured 6 frames (1280x720)
video-viewer:  captured 7 frames (1280x720)
video-viewer:  captured 8 frames (1280x720)
video-viewer:  captured 9 frames (1280x720)
video-viewer:  captured 10 frames (1280x720)
video-viewer:  captured 11 frames (1280x720)
video-viewer:  captured 12 frames (1280x720)
video-viewer:  captured 13 frames (1280x720)
video-viewer:  captured 14 frames (1280x720)
video-viewer:  captured 25 frames (1280x720)
video-viewer:  captured 50 frames (1280x720)
video-viewer:  captured 75 frames (1280x720)
video-viewer:  captured 100 frames (1280x720)
```

Fig. 6.14 The video from the camera is sent to the Jetson AGX as an RTP streaming on port 1234

```
neruzz@agx11:~$ sudo tcpdump -i any port 1234
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
20:53:08.829715 IP rpi51.lan.1314 > agx11.lan.1234: UDP, length 961
20:53:08.849144 IP rpi51.lan.1314 > agx11.lan.1234: UDP, length 1400
20:53:08.869101 IP rpi51.lan.1314 > agx11.lan.1234: UDP, length 1400
20:53:08.884561 IP rpi51.lan.1314 > agx11.lan.1234: UDP, length 1400
20:53:08.904088 IP rpi51.lan.1314 > agx11.lan.1234: UDP, length 1400
20:53:08.929795 IP rpi51.lan.1314 > agx11.lan.1234: UDP, length 1400
20:53:08.939757 IP rpi51.lan.1314 > agx11.lan.1234: UDP, length 1400
20:53:08.959746 IP rpi51.lan.1314 > agx11.lan.1234: UDP, length 1400
20:53:08.979740 IP rpi51.lan.1314 > agx11.lan.1234: UDP, length 1400
```

Fig. 6.15 `tcpdump` command in the Jetson AGX shows how the RTP streaming sent by the UE is arriving at port 1234

the frames of the streaming video. The command has the streaming from the UE as input and then the output can be set up as another streaming using RTP, RTSP, WebRTC[8] or saved

[8]WebRTC (Web Real-Time Communication) is an open-source project and technology that enables real-time communication such as voice, video, and data transfer directly in web browsers and mobile applications. It allows peer-to-peer connections without the need for plugins or third-party software, facilitating a wide range of applications like video conferencing, file sharing, and live-streaming. WebRTC is supported by most modern web browsers and is designed to work through NATs and firewalls, making it highly versatile for real-time applications on the internet.

```
root@f950008576d4:/open5gs# tcpdump -i ogstun
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ogstun, link-type RAW (Raw IP), capture size 262144 bytes
20:58:50.374513 IP 10.0.101.17.1314 > agx11.lan.1234: UDP, length 1400
20:58:50.394520 IP 10.0.101.17.1314 > agx11.lan.1234: UDP, length 1400
20:58:50.410009 IP 10.0.101.17.1314 > agx11.lan.1234: UDP, length 1400
20:58:50.429513 IP 10.0.101.17.1314 > agx11.lan.1234: UDP, length 1400
20:58:50.449495 IP 10.0.101.17.1314 > agx11.lan.1234: UDP, length 1400
20:58:50.464945 IP 10.0.101.17.1314 > agx11.lan.1234: UDP, length 1400
20:58:50.485016 IP 10.0.101.17.1314 > agx11.lan.1234: UDP, length 1400
20:58:50.494963 IP 10.0.101.17.1314 > agx11.lan.1234: UDP, length 805
20:58:50.509553 IP 10.0.101.17.1314 > agx11.lan.1234: UDP, length 925
20:58:50.524982 IP 10.0.101.17.1314 > agx11.lan.1234: UDP, length 1400
20:58:50.544970 IP 10.0.101.17.1314 > agx11.lan.1234: UDP, length 1400
20:58:50.564489 IP 10.0.101.17.1314 > agx11.lan.1234: UDP, length 1400
```

Fig. 6.16 `tcpdump` performed over the *ogstun* interface on the UPF shows how the UE traffic arrives to the edge using the 5G GTP tunnel

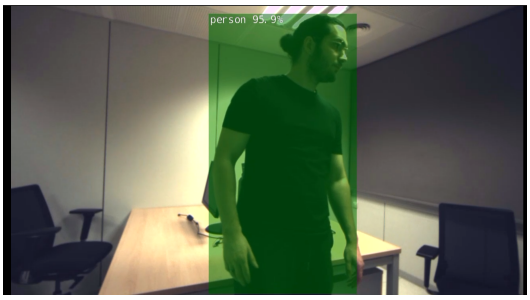
to a file, as in the scope of this project. The command is below, and the output is depicted in Figure 6.17. Finally, some frames of the video file with the captured inference streaming are shown in Figure 6.18 where some objects are detected on the camera streaming that the Jetson AGX receives in port 1234.

```
# The command gets the streaming as input and outputs an MP4 file with the inference
detectnet rtp://agx11:1234 inference.mp4
```

```
[TRT]    ------------------------------------------------
[TRT]    Timing Report /usr/local/bin/networks/SSD-Mobilenet-v2/ssd_mobilenet_v2_coco
.uff
[TRT]    ------------------------------------------------
[TRT]    Pre-Process   CPU   0.02954ms  CUDA   0.07459ms
[TRT]    Network       CPU   7.90619ms  CUDA   7.83411ms
[TRT]    Post-Process  CPU   0.01274ms  CUDA   0.00214ms
[TRT]    Total         CPU   7.94846ms  CUDA   7.91085ms
[TRT]    ------------------------------------------------


[TRT]    ------------------------------------------------
[TRT]    Timing Report /usr/local/bin/networks/SSD-Mobilenet-v2/ssd_mobilenet_v2_coco
.uff
[TRT]    ------------------------------------------------
[TRT]    Pre-Process   CPU   0.02960ms  CUDA   0.07398ms
[TRT]    Network       CPU   7.88158ms  CUDA   7.78058ms
[TRT]    Post-Process  CPU   0.04890ms  CUDA   0.00266ms
[TRT]    Total         CPU   7.96008ms  CUDA   7.85722ms
[TRT]    ------------------------------------------------
```

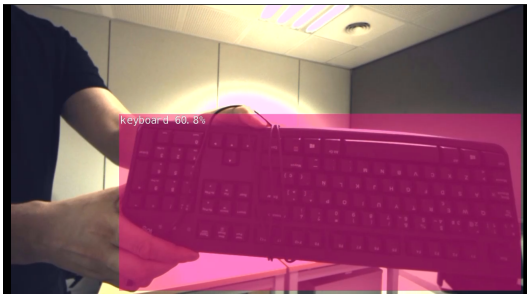
Fig. 6.17 Output of the *detectnet* command that executes the deep neural network inference over the streaming video received



(a) AI detecting a person (me)

(b) AI detecting a water bottle

(c) AI detecting scissors



(d) AI detecting a keyboard

Fig. 6.18 The streaming chain works using 5G and the AI inference is displayed over the raw camera video. These images are captured from the video saved on the Jetson AGX but could be streamed to the edge or other parts of the 5G network.

The functionality of the application has been successfully demonstrated, confirming that edge computing, when integrated with high-performance computing capabilities over a 5G network, can effectively overlay AI inference using deep neural networks on an RTP streaming sent from the UE. This setup is particularly useful in real-world scenarios such as autonomous vehicle navigation, remote healthcare monitoring, and urban surveillance systems, where real-time data processing and decision-making are crucial. In cases where the UE lacks sufficient computing power, data can be offloaded to the edge for processing, thereby leveraging the edge's robust computational resources to handle complex tasks and return processed data or decisions to the UE.

However, a limitation has been identified in the reverse pathway (downlink), from the edge (Jetson AGX and Raspberry Pi 5) back to the UE. Attempts to transmit data back to the UE were unsuccessful. Using the `nmap` command, it was observed that the modem only permits traffic through HTTP, HTTPS, and TCP port 1555, blocking all UDP streaming attempts. This explains why the streaming app in Section 6.3, which uses HTTP streaming, works successfully both in the uplink and in the downlink paths, and the RTP UDP streams of this application fail on the downlink path. Future work could explore the potential of integrating WebRTC in the downlink path (and potentially uplink), which operates over HTTP

and HTTPS protocols, to overcome these limitations. This adjustment could potentially enable seamless two-way communication between the edge devices and the UE, enhancing the system's applicability in interactive applications where real-time feedback is essential.

Chapter 7

Conclusions and future work

7.1 Conclusions

The conclusions of this project are structured according to the objectives described in the introduction section. Each objective has been successfully achieved, and the lists below provide insights into some of the key learnings and conclusions derived from each.

This project has significantly enhanced our understanding of 5G architectural concepts, highlighting the intricate features and advanced functionalities that define modern network systems. The following are some of the broad key concepts that were thoroughly explored:

- Understanding the key differences between 5G and previous generations.
- Exploring the architecture of 5G, its NFs, and their functionalities.
- Grasping the concept of network slicing and programming due to the flexibility of 5G and CUPS.
- Analyzing the components of NG-RAN and how they interact with the 5G CN.
- Examining the significant relationship between edge computing and 5G, and the potential applications when combined with edge computing.

The exploration of various 5G network software and stacks, both open-source and proprietary, has provided valuable insights into the practical aspects of network setup and management. Here are the principal concepts and tools that were mastered:

- Utilization of the Open5GS stack and understanding how each NF and daemon operates within the program.

- Learning how the Amarisoft stack functions across different kinds of deployments.
- Implementing UERANSIM as both UE and RAN, and its application in testing deployments prior to the use of COTS stacks.
- Recognizing that while proprietary software offers strong troubleshooting support, it lacks flexibility. Conversely, open-source software, though highly flexible, often relies on community support, which may not meet all needs.
- Acquiring basic knowledge in the operation of other open-source 5G cores, such as OAI or Free5GC.

The successful deployment of a 5G testbed equipped with edge computing capabilities marks a major achievement of this project. The following items highlight the critical learnings obtained during the deployment process, including the integration of edge routing:

- Gaining practical reverse engineering skills to comprehend the foundational projects and repositories utilized.
- Deepening knowledge on Docker and Docker Compose, including how to create Docker images.
- Enhancing expertise in network management on Linux systems, specifically for 5G networks. This includes command-line skills, network structure reverse engineering, network creation, and testing.
- Learning to integrate an open-source core with Amarisoft gNB, and how to assess connectivity and various testing parameters using a Callbox.
- Developing proficiency in bash scripting and understanding the main core functions' communication and log interpretation.
- Gaining basic knowledge in SIM programming and related configurations and concepts.
- Learning to configure COTS phones to connect to the network and access 5G services, an important contribution given the scarcity of real UEs in the state of the art.
- Configuring a Quectel modem with AT commands to interface with the 5G network and provide internet to devices like a Raspberry Pi and a Jetson Nano, following configuration guides from the seller.

Measuring various performance metrics was crucial for validating our project hypotheses and refining the network setup. Below are the key insights derived from these measurements:

- Acquired skills in utilizing measurement tools such as `iperf3` and `ping`.
- Determined the significance of the UPF location: The further the UPF from the UE, the greater the delay, which adversely affects delay-sensitive applications.
- Observed that `iperf3` experiences functionality issues under network conditions degraded by `tc` commands.
- Found that throughput and jitter remain consistent within a single rack.

The practical tests conducted on the testbed proved instrumental in demonstrating its capabilities and limitations. These tests provided concrete examples of the testbed's real-world applicability:

- Simplified the setup of new services using a Docker Compose file to facilitate deployment.
- Demonstrated the feasibility of establishing a messaging service at the edge, confirming that hosting applications at the edge is viable.
- Implemented a streaming service at the network edge with satisfactory Quality of Service (QoS), proving that delay-sensitive applications can effectively be hosted at the edge.
- Performed object detection and other AI inference tasks on video streams sent from the UE to a High-Performance Computing (HPC) node at the edge, achieving immediate response times.
- Identified *jetson-inference* as the optimal solution for streaming and processing data using HPC on Jetson devices.
- Noted that HTTP streaming is less efficient than using RTP for real-time data transfer.
- Conducted network debugging to identify and resolve data transmission issues, particularly in the downlink path of the object detection application.

Overall, this project has met all its objectives, marking it as a success in advancing both the understanding and practical implementation of 5G technologies. By overcoming initial challenges and engaging in hands-on experimentation, significant strides were made

in demonstrating the potential and limitations of current 5G solutions. This experience has provided not only valuable knowledge but also a practical platform for future research. Furthermore, it has laid the groundwork for ongoing enhancements in 5G network development and deployment.

7.2 Limitations and future work

7.2.1 Limitations

The limitations of the project are listed below:

- Open-source 5G Core platforms like Open5GS, Free5GC, and OAI are experimental and lack stability in some functionalities. These platforms do not fully support the vast array of use cases or the COTS hardware used, such as the phone used on the project. Although developers often address issues quickly, some solutions may be delayed if they are complex or if the bug is not critical. Furthermore, support and error resolution are primarily community-driven due to incomplete documentation and private support team. This reliance on community assistance can complicate troubleshooting efforts. Additionally, the unique use cases or deployments of individual users, such as the one developed in this thesis, mean that experiences and problems can vary widely, leading to numerous unresolved inquiries and issues on forums and GitHub. Consequently, research often becomes a process of trial and error.

- On the other hand, proprietary software supervised by private companies, such as Amarisoft's products, often lacks flexibility. In this project, Amarisoft's core functioned flawlessly, exhibiting no bugs or incompatibility issues with COTS hardware, and the support team quickly and effectively resolved any doubts or configuration errors. However, a significant limitation is that the deployment pursued in this project requires executing the core in containers and across different nodes, which Amarisoft's licensing restricts to operation only within the AMARI Callbox. When inquiries were made about integrating Open5GS with Amarisoft or modifying the software to relocate a UPF from the Callbox, the support team clarified that they could not provide assistance with products not their own, nor is it possible to execute core functions outside the Callbox. This restriction made it impossible to use both Amarisoft's core and gNB, despite them being well-developed and supported, and this was the determining factor to use Amarisoft only as gNB.

- Generally, there is a notable shortage of accessible documentation or overly technical materials, such as 3GPP reports and standards, which can be challenging to navigate or understand. This issue is particularly evident in areas like SIM programming and implementing or understanding 5G specific functionalities or procedures. Additionally, documentation for 5G modems and mobile network modems is scarce and often inadequate. For example, the Quectel RM502Q-AE modem used in the project lacked essential starter documentation, and some of the available AT commands provided by Quectel for internal configuration were not working. During various phases of the project, problem-solving often relied on trial and error and experimenting, as explained before.

- The BSC environment is designed with robust security measures to prevent external infiltrations from outside the office network. This security setup complicates the construction of real-world scenarios involving external components, such as cloud services. Additionally, certain protocols like ICMP, typically utilized by tools like `ping`, are restricted to internal use within the BSC network. As a result, connectivity checks to the internet are conducted using `curl` instead of `ping`, and all `ping` tests within the project are confined to the office network. This stringent network security is the primary reason why all components (gNB, cloud and edge) are housed within the same rack inside the same network.

7.2.2 Future work

The future work can be divided into two main categories: the first concerns the project discussed in this thesis, and the second relates to the current state of the technology environment. Focusing on the project described in this thesis, there are several upgrades and research paths:

- One of the main research paths is to develop a more realistic platform or testbed. This could be achieved by deploying the control plane in a real cloud environment, while situating the UE plane closer to the edge, potentially within the office itself. This configuration would move away from the rigidity of the current rack setup used in the office. Such a setup would naturally incorporate real-world delays to cloud communications, which cannot be effectively simulated in a controlled environment. Metrics gathered in this more authentic setting would be both more realistic and scientifically valuable, providing deeper insights into system performance and behavior under typical operational conditions.

- This realistic platform could be deployed using Kubernetes, since this offers numerous advantages for managing distributed applications across varied environments. Kubernetes excels in automating deployment, scaling, and operations of containerized applications, making it highly suitable for a dynamic testbed like this, which uses VF deployed in docker containers. It can efficiently handle the deployment of the control plane in the cloud and the UPF at the edge. Furthermore, Kubernetes provides robust fault tolerance, ensuring that services can recover quickly from hardware failures or software issues. Its capabilities in load balancing and traffic routing are essential for optimizing performance and maintaining consistent service levels across different network conditions. Additionally, Kubernetes modular approach allows for continuous integration and continuous deployment (CI/CD) practices, simplifying updates and scaling operations without downtime. These features collectively enhance the platform's ability to provide a realistic, robust, automated and scientifically rigorous testing environment.

- Another potential improvement in the research could involve conducting the metrics analysis directly on the deployed applications. For example, integrating a real-time statistics heads-up display (HUD) on streaming video content could provide immediate insights into the QoS. This enhancement would allow monitoring network performance parameters such as latency, packet loss, and throughput in real-time as they affect video streaming quality. Additionally, extending this approach to other applications, like VoIP or real-time gaming, could help in comprehensively evaluating the network's impact on various real-world uses, further refining the development and optimization of the testbed.

- To conclude, it is essential to stay updated with the advancements in the technologies and frameworks utilized in the testbed to facilitate timely adaptations or upgrades. This approach would not only allow the testbed to support a wider variety of applications but also enhance its capability to host more demanding services. Additionally, continuing to test the testbed with applications beyond those discussed in the thesis is crucial. This experimentation will help in fine-tuning the testbed's performance and expanding its utility across more diverse use cases.

The second category focuses on outlining potential contributions that can enhance the overall state of the technologies used in this project, addressing some of the limitations described in the previous section. These contributions aim to improve interoperability, functionality, and user support, thereby advancing the field and mitigating existing challenges.

- The stability and functionality of open-source 5G Core platforms such as Open5GS, Free5GC, and OAI need enhancement to better accommodate diverse use cases and hardware configurations, like those utilized in this project. Rapid response to bugs is commendable, yet complex issues often face delays. Community-driven support and troubleshooting, hampered by gaps in documentation, lead to inconsistent problem resolution. Future efforts should aim to enrich the documentation and establish more robust support frameworks to facilitate smoother development cycles and reduce reliance on trial and error approaches.

- The rigidity of proprietary software licensing, particularly with products from companies like Amarisoft, limits their utility in diverse operational contexts. This project's experience underscores the need for more adaptable licensing models and software that can be customized for varied deployment environments, encouraging future discussions about licensing flexibility and interoperability or, at least, a distributed system solution to allow the possible deployment of edge in the 5G network.

- The lack of clear and accessible documentation for critical 5G technologies, especially in modem configuration and SIM programming, represents a significant barrier. The project's struggles with the Quectel RM502Q-AE modem illustrate the challenges posed by inadequate resources. Future work should focus on closing these informational gaps and providing more reliable, user-friendly guides and technical support to ensure developers have the resources needed to implement 5G technologies effectively.

References

[1] O. Teyeb, et al., "Evolving lte to fit the 5g future.," *ERICSSON*, 2017. Available online: https://www.ericsson.com/4ae09e/assets/local/reports-papers/ericsson-technology-review/docs/2017/etr_evolving_lte_to_fit_the_5g_future.pdf.

[2] P. Marcin Dryjanski, "5g core network – a short overview," 2017. Available online: https://www.grandmetric.com/5g-core-network-a-short-overview/.

[3] O. Özenir, "Redundancy techniques for 5g ultra reliable low latency communications," Master's thesis, Università di Bologna, 2021.

[4] A. Aijaz, "Duplication in dual connectivity enabled 5g wireless networks: Overview and challenges," *IEEE Communications Standards Magazine*, pp. 20–28, Sept. 2019.

[5] S. Lee, "Introduction to open5gs," January 2019. Available online: https://open5gs.org/open5gs/docs/guide/01-quickstart/.

[6] Amarisoft, "Amari callbox classic," 2023. Available Online: https://www.amarisoft.com/test-and-measurement/device-testing/device-products/amari-callbox-classic.

[7] "Architecture enhancements for control and user plane separation of epc nodes," Technical Report 23.214, 3GPP. Available online: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3077.

[8] "System architecture for the 5g system," Technical Report 23.501, 3GPP. Available online: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144.

[9] "Procedures for the 5g system," Technical Report 23.502, 3GPP. Available online: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3145.

[10] "Service requirements for the 5g system," Tech. Rep. 22.261, 3GPP. Available online: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3107.

[11] "Study on New Radio (NR) to support non-terrestrial networks (Release 15)," Technical Report 38.912, 3GPP. Available online: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3059.

[12] 5G PPP Technology Board Working Group and 5G-IA's Trials Working Group, "Edge Computing for 5G Networks," 1 2021. Available online: https://doi.org/10.5281/zenodo.3698117.

[13] N. Hassan, K.-L. A. Yau, and C. Wu, "Edge computing in 5g: A review," *IEEE Access*, Sep 2019. Received August 2, 2019, accepted August 26, 2019, date of publication August 30, 2019, date of current version September 18, 2019.

[14] B. Liu, Z. Luo, H. Chen, and C. Li, "A survey of state-of-the-art on edge computing: Theoretical models, technologies, directions, and development paths," *IEEE Access*, May 2022. Received April 24, 2022, accepted May 15, 2022, date of publication May 18, 2022, date of current version May 25, 2022. Supported in part by the National Natural Science Foundation of China under Grant 61801319, and other grants.

[15] ETSI Industry Specification Group, "Multi-access Edge Computing (MEC): Framework and Reference Architecture," ETSI Group Specification ETSI GS MEC 003 V3.1.1, European Telecommunications Standards Institute, March 2022. Available online: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/03.01.01_60/gs_mec003v030101p.pdf.

[16] Y. Cao, Y. Wu, P. Lv, Z. Zhu, S. Quan, X. Zhou, and S. Gao, "Research on 5gc cloud-native deployment mechanism based on cloud-edge collaboration," 2023.

[17] Á. Vázquez-Rodríguez, C. Giraldo-Rodríguez, and D. Chaves-Diéguez, "A cloud-native platform for 5g experimentation," 2023.

[18] S. Barrachina-Muñoz, M. Payaró, and J. Mangues-Bafalluy, "Cloud-native 5g experimental platform with over-the-air transmissions and end-to-end monitoring," *arXiv preprint arXiv:2207.11936*, Jul 2022.

[19] J. Baranda, S. Barrachina-Muñoz, R. Nikbakht, M. Payaró, and J. Mangues-Bafalluy, "Disaggregating a 5g non-public network via on-demand cloud-native upf deployments," 2023.

[20] A. Güngör, "Ueransim: An open-source 5g ue and ran simulator," 2021. GitHub repository: https://github.com/aligungr/UERANSIM.

[21] R. Pi, "Raspberry pi 5." Available online: https://www.raspberrypi.com/products/raspberry-pi-5/.

[22] NVIDIA, "Embedded systems - overview." Available online: https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/#intro.

[23] NVIDIA, "Jetson nano - product development." Available online: https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-nano/product-development/.

[24] NVIDIA, "Jetson orin." Available online: https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-orin/.

[25] NVIDIA, "Jetpack sdk." Available online: https://developer.nvidia.com/embedded/jetpack.

[26] S. Galaxy, "Samsung galaxy a23 5g awesome blue." Available online: https://www.samsung.com/es/smartphones/galaxy-a/galaxy-a23-5g-awesome-blue-128gb-sm-a236blbveub/.

[27] Quectel, "Quectel rm502q-ae 5g specification v1.2." Available online: https://www.quectel.com/wp-content/uploads/2021/03/Quectel_RM502Q-AE_5G_Specification_V1.2.pdf.

[28] R. Pi, "Raspberry pi 4 model b." Available online: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[29] Sysmocom, "sysmoisim-sja2 programmable sim/usim/isim cards." Available online: https://sysmocom.de/products/sim/sysmousim/index.html#.

[30] Osmocom, "sysmousim-sjs1." Available online: https://osmocom.org/projects/cellular-infrastructure/wiki/SysmoUSIM-SJS1.

[31] J. Hecht, *Understanding Fiber Optics*. 2015. ISBN978-1511445658.