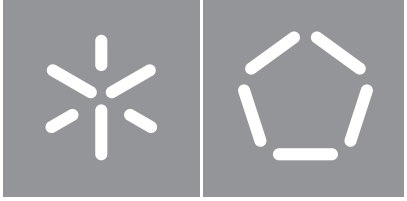
Universidade do Minho
Escola de Engenharia

Tiago Magalhães

Implementation of a virtualized 5G network

June, 2023



Universidade do Minho
Escola de Engenharia

Tiago Magalhães

Implementation of a virtualized 5G network

Master Dissertation
Integrated Master Degree in Informatics Engineering

Work supervised by
António Luís Duarte Costa
Helena Fernández López

June, 2023

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the Universidade do Minho.

_____, _____

(Location) (Date)

_____

(Tiago Magalhães)

Acknowledgements

I would like to express my gratitude to my supervisors for their support and guidance and for allowing me to take this dissertation. I would also like to thank my friends whom I have met during my university course. I am grateful for the great memories and experiences shared and the support and help they provided me. Furthermore, I would like to thank my family, especially my parents and brother, for allowing me to pursue my studies at the university and for their support throughout these years.

Abstract

Implementation of a virtualized 5G network

Many organizations have developed open software components for 5G (Fifth Generation) networks and recognize the importance of new technologies based on virtualization and softwarization.

With these solutions, it is possible to implement a 5G virtualized network without having access to a mobile network, which has many restrictions.

Implementing a 5G testbed is essential because it allows the creation of a framework that can enable the development and research of new solutions related to 5G.

This dissertation proposes a solution that uses open-source software to emulate the access network and deploys software modules that implement core network functionalities. Moreover, network capabilities, as well as interoperability, are described.

Keywords: 5G Simulation, 5G Testbed, 5G Virtualization, Softwarization, Emulator

Resumo

Implementação de uma rede 5G virtualizada

Muitas organizações têm desenvolvido soluções de *software* aberto para componentes da rede 5G (*Fifth Generation*) e reconhecido a importância de novas tecnologias baseadas em virtualização e em princípios de *software*.

É possível implementar uma rede virtualizada 5G com base nestas soluções, sem necessidade de ter acesso a uma rede móvel, o que possui muitas restrições.

A criação de um ambiente de testes 5G é importante, uma vez que permite criar uma estrutura que pode possibilitar o desenvolvimento e o estudo de novas soluções relacionadas com o 5G.

Nesta dissertação, é proposta uma solução emulando a rede de acesso e implementando um core recorrendo a *software de código aberto*. As capacidades da rede são descritas, bem como a interoperabilidade entre as diferentes soluções.

Palavras-chave:

5G, Ambiente de testes, Virtualização, *Software*, Emulador

Contents

List of Figures

Listings

Glossary

eNB	eNB (sam as eNodeB) is a 3GPP-compliant implementation of the 4G base station.
eNodeB	eNodeB is a 3GPP-compliant implementation of the 4G base station.
gNB	gNB (same as gNodeB) is a 3GPP-compliant implementation of the 5G base station.
gNodeB	gNodeB is a 3GPP-compliant implementation of the 5G base station.
Non Standalone	5G RAN is used with the existing LTE RAN and Core.
Standalone	The standalone (SA) mode of 5G uses 5G RAN with 5G Core instead LTE Evolved Packet Core.

Acronyms

3D	Three dimensional
3GPP	3rd Generation Partnership Project
5G	Fifth Generation
5GC	5G Core Network
AF	Application Function
AI	Artificial Intelligence
AMF	Access and Mobility Management Function
API	Application Programming Interface
AUSF	Authentication Server Function
BBU	Baseband Unit
BS	Base Station
CBR	Constant Bit Rate
CLI	Command Line Interface
CN	Core Network
COTS	commercial off-the-shelf
CPU	Central Processing Unit
CU	Central Unit
CUPS	Control and User Plane Separation
D2D	Device to Device
DL	Downlink
DN	Data Network
DU	Distributed Unit

e2e	End-to-End
EPC	Envolved Packet Core
ETSI	European Telecommunications Standards Institute
FCAPS	Fault, Configuration, Accounting, Performance, and Security
FDD	Frequency Division Duplex
FES	Future Event Set
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IDE	Integrated Development Environment
IoT	Internet of things
LTE	Long Term Evolution
M2M	Machine to Machine
MAC	Media Access Control Layer
MANO	Management and Orchestration
MEC	Multi Access Edge Computing
MIMO	Multiple-Input Multiple-Output
ML	Machine Learning
mm-wave	Millimeter Wave
NEF	Network Exposure Function
NF	Network Functions
NFV	Network Function Virtualization
NFVI	NFV Infrastructure
NFVO	NFV Orchestrator
NG	New Generation
NRF	Network Repository Function

NS Network Service
NSF Network Exposure Function
NSSF Network Slice Selection Function

OAI Open Air Interface

PCF Policy Control Function
PDU Packet Data Unit
PFC Policy Control Function
PHY Physical Layer
PNF Physical Network Functions

RAN Radio Access Network
REST Representational State Transfer
RF Radio Frequency
RRC Radio Resource Control
RRH Remote Radio Head
RU Radio Unit

SBI Service-based Interfaces
SDN Software Defined Network
SMF Session Management Function
SON Self-Organizing Network
SSH Secure Socket Shell

TLS Transport Layer Security
TN Transport Network

UDM Unified Data Management
UDP User Datagram Protocol
UDR Unified Data Repository
UDSF Unstructured Data Storage Function

UE	User Equipment
UL	Uplink
UPF	User Plane Function
veth	Virtual Ethernet
VIM	Virtualized Infrastructure Manager
VL	Virtual Link
VM	Virtual Machine
VNF	Virtual Network Function
VNFM	VNF Manager
VoD	Trace-based Video-on-demands
VoIP	Voice-over-IP

Chapter 1

Introduction

To make the Fifth Generation (5G) networks more accessible, cost-effective, and efficient, virtualization and softwarization techniques like Software Defined Network (SDN) and Network Function Virtualization (NFV) are being used. These technologies allow for breaking down network functions into smaller blocks, increasing flexibility and reducing costs.

This dissertation discusses the problem of testing and developing new 5G solutions due to vendor constraints, licenses, and specialized equipment, especially with the high cost of 5G Base Stations (BSs). The motivation for this research is to create a virtualized 5G network using open-source software that can be used for future experimentation.

The objectives of this research are to identify and use free software modules to implement the core functionalities of the 5G network and create scenarios of increasing complexity to test these functionalities. The document is structured into six additional chapters that cover the research methodology, literature review, simulation and emulation of 5G networks, evaluation of scenarios, and finally, conclusion.

1.1 Context

With the increase in the use of smartphones, and the emergence of Internet of things (IoT) and new applications such as self-driving cars, Three dimensional (3D) video, and ultra-high definition screens, it is essential that the network can respond to pressing challenges, such as scalability, and the need for higher data rates. Thus, the 5G cellular network will be essential to meet these demands [1, 2].

There was a gradual evolution until this generation, starting in 1980 and having experienced a new generation every ten years, this decade being the fifth. To meet the demands identified above, 5G architecture and new technologies, such as using Millimeter Wave (mm-wave) and small cells, which allow higher

data rates with reduced latency, and massive Multiple-Input Multiple-Output (MIMO) that improve energy efficiency will play a crucial role.

Although traffic is growing exponentially, investing in more specialized infrastructure is no longer sustainable. Therefore, the use of solutions based on new technologies such as SDN and NFV can reduce constraints and hardware costs and increase management, flexibility, energy efficiency, and End-to-End (e2e) latency [3]. In addition, many emerging verticals such as eHealth, energy, and city management will benefit from 5G enhancements.

It can be seen that many benefits to 5G come from softwarization (e.g., increased programmability), virtualization, and disaggregation of network functions (i.e., separation of control and data planes). Consequently, standards bodies have embraced these technologies, as have other organizations promoting open-source solutions to stimulate innovation, competitiveness, research, and interoperability. These technologies contribute to removing barriers to 5G adoption. Furthermore, these increase the speed and security of 5G development because they can be broken into small blocks and reviewed frequently, leading to advanced use cases sooner and at lower costs.

1.2 Problem

Developing and testing new solutions and use cases requires access to a cellular network in operation. However, vendor constraints, licenses, and specialized equipment exist, especially with 5G. For example, a BS costs four times higher than a Long Term Evolution (LTE) BS [4].

With a virtualized 5G network using open-source publicly available software, it is possible to lower expenses and additional features can be implemented and adapted. However, despite the emergence of such solutions, it is essential to evaluate and validate how the different modules from different organizations can be used interoperably and what functionalities and 3rd Generation Partnership Project (3GPP) specifications can be achieved.

1.3 Motivation

Many industry associations, mobile operators, and researchers have driven the use of solutions based on software and virtualization [5, 6]. In addition, it should be possible to implement a virtualized 5G network from emerging solutions. Among these solutions are simulation/emulation frameworks such as Simu5G [7], and publicly available open source software modules for Radio Access Network (RAN) and 5G Core Network (5GC) such as srsLTE [8], and free5GC [9], in addition to others provided by organizations such as Open Air Interface (OAI).

A virtualized 5G testbed can be used to research and develop 5G solutions. For instance, novel algorithms for New Generation (NG) protocol stack, orchestration techniques, better use of resource allocation

[10], network slicing [11], and simulation of an urban environment. As a result, all these use cases can be tested, measured, and verified before real usage with reduced costs and constraints.

1.4 Objectives

This dissertation aims to implement a 5G virtualized network to be used for future experimentation. This work chooses to emulate the user devices and the access network and deploy software modules that implement the core functionalities of the 5G network. Therefore, in the first phase, it is necessary to identify a simulator/emulator of 5G mobile networks and then software modules whose use, for research purposes, is free of charge and implement the 5GC network. After this identification, scenarios with increasing complexity will be defined and implemented. The first scenario aims to simulate an e2e 5G network, with only one BS and a reduced number of user devices. The second scenario extends this network, providing several user devices with different mobility standards. Finally, in the last scenario, the user devices and the access network are emulated and connected to a set of software modules that implement core network functionality. The limitations and features offered will be identified in each of these scenarios.

1.5 Document Structure

The dissertation has six additional chapters. Chapter 1 begins with an introduction, in which a contextualization and problem formulation is made, as well as the motivation and goals to achieve. In Chapter 2, concepts and technologies related to the dissertation are addressed. Chapter 3 introduces the role of virtualization and softwarization in 5G networks. Then, 5G testbeds architecture and capabilities related to the problem are described. In Chapter 4, the solution to solve the problem is presented. Then, Chapter 5 describes the implementation of the proposed solution, and in Chapter, 6 the results are discussed. Finally, Chapter 7 presents the conclusion and identifies potential future enhancements.

Chapter 2

Concepts and technologies

This chapter aims to provide a comprehensive background on the concepts and technologies related to the dissertation. Firstly, a brief overview of the different generations of mobile networks is presented, focusing on the latest 5G network and its key features and enabling technologies. The chapter then delves deeper into the concepts of softwarization and virtualization in 5G networks, specifically the use of NFV and SDN, which form the foundation of the 5G architecture. Furthermore, the chapter highlights emulators/simulators of RAN and software modules whose use for research purposes is free of charge that implement the core of the 5G network are identified.

2.1 5G

Since its first generation, mobile networks have been mainly composed of RAN, which is constituted by the BS, the User Equipment (UE), and the core.

The RAN connects UEs to BSs and then BSs to the core network, which is linked to the global internet. The core network provides services to users connected through the RAN, such as aggregation that combines the links from RAN to increase throughput, authentication, and gateways to access other networks. These components, over the generations, have been enhanced to improve data rate, mobility, coverage, and spectral efficiency [1].

Nowadays, with the growth of mobile traffic, the need for networks to support much more devices as a result of the emergence of IoT paradigm and new user-oriented mobile multimedia applications like mobile video conferencing, streaming video, augmented reality, and online gaming, new technologies are needed to meet these challenges [12, 13].

Therefore, 5G architecture and technologies will have to count these new applications and the need to handle more traffic per area and energy efficiency. It follows that the 5G technology enablers are:

- The use of mm-wave frequencies from 3-300 GHz will increase capacity because it depends on spectral efficiency and bandwidth [14].
- Massive MIMO, due to the use of higher frequencies, the use of arrays of hundred antennas are possible. Moreover, these antennas are positioned in order to be directed to devices through spatial multiplexing. Hence, this technology improves energy efficiency and capacity [14].
- Small Cells are BSs with compact size and lower power consumption. Because mm-wave have higher path loss, diffraction, and blockage, the use of these cells in urban areas and indoor environments will provide better coverage and low latency [14].
- Device to Device (D2D) will enable low latency and scalability because devices will communicate directly without going through the network infrastructure, decreasing control signal and latency [1].

These enhancements will benefit new emerging verticals such as:

- IoT paradigm, which involves millions of different devices. Data collected by these devices will benefit from high bandwidth and capacity with 5G.
- Vehicular and healthcare applications will benefit from low latency, since these kinds of applications require fast feedback.
- Smart Cities and public safety systems will need higher bandwidth and reduced latency.
- Augmented reality applications are becoming popular in education, gaming, and simulations. Thus, 5G low latency can enable real-time usage [15].

Emerging software technologies like SDN and NFV will also have a role in 5G network.

2.1.1 5G Architecture

In order to implement the 5G testbed, it is necessary to know what are the necessary components. Figure 1 shows compliant 3GPP 5G architecture.

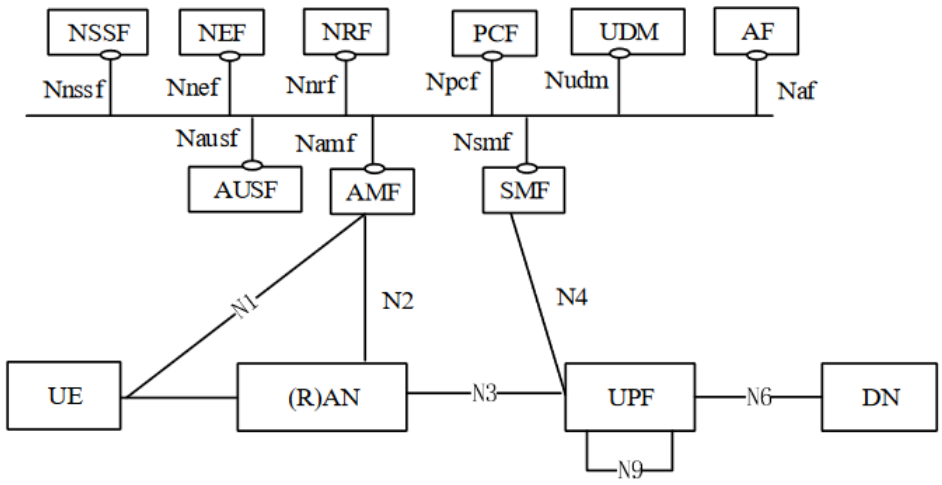


Figure 1: 5G High Level System architecture (Extracted from [16]).

This architecture shows two significant components: RAN and 5GC. The RAN is constituted by UE and (R)AN, and these two communicate over a radio interface and are responsible for network selection, identification and authentication, authorization, access control and barring, policy control, and lawful interception [16].

The architecture's main goal is to enable the use of virtualization and softwarization. As a result of decoupled Network Functions (NF) and the separation of user and control plane functions assuming a service-based architecture, where each NF interacts with each other through the exposed interfaces, towards to provide scalability, distributed deployment, and network slicing.

The NFs presented in a 5G system are the following:

- Authentication Server Function (AUSF) responsible for providing authentication service related to the UE when other NF requests;
- Access and Mobility Management Function (AMF) is responsible for mobility management and security, and it is the termination that connects the control plane of the RAN, by establishing the connection between the UE and 5GC;
- Data Network (DN), e.g. operator services, Internet access or 3rd party services;
- Unstructured Data Storage Function (UDSF) is responsible for storing and managing unstructured data;
- Network Exposure Function (NEF) handles the access of third-party applications to the 5GC. This application are usually related to monitoring, partners and policies;

- Network Repository Function (NRF) is a repository that handles the discovery process of other 5G network functions;
- Network Slice Selection Function (NSSF) is responsible to select the set of Network slices that will serve the UE;
- Policy Control Function (PCF) is responsible for managing the network policy with regard to control plane functions and to enforce them;
- Session Management Function (SMF), is responsible for session management, for instance to handle UE IP address allocation, session establishment, modify and release;
- Unified Data Management (UDM) is responsible to store user data such as identification, subscription and authorization;
- Unified Data Repository (UDR) stores data related to the subscription, policy and structured data for exposure;
- User Plane Function (UPF) handles Quality of Service, forwarding and routing between the RAN and the DN;

The interfaces connecting the control plane interfaces are denominated Service-based Interfaces (SBI) and are connected to a bus, which allows communication among NFs through open Application Programming Interfaces (APIs).

- *Namf*: interface exposed by AMF.
- *Nsmf*: interface exposed by SMF.
- *Nnef*: interface exposed by NEF.
- *Npcf*: interface exposed by PCF.
- *Nudm*: interface exposed by UDM.
- *Naf*: interface exposed by Application Function (AF).
- *Nnrf*: interface exposed by NRF.
- *Nnssf*: interface exposed by NSSF.
- *Nausf*: interface exposed by AUSF.
- *Nudr*: interface exposed by UDR.

- *Nudsf*: interface exposed by UDSF.

The reference point interfaces are referenced by starting with an "N"followed by a number, and on the contrary with SBI it uses point-to-point connection.

- *N1*, connects the UE and the AMF using NAS protocol.
- *N2*, connects the (R)AN and the AMF using NGAP protocol.
- *N3*, connects the (R)AN and the UPF using GTP protocol.
- *N4*, connects the SMF and the UPF using Packet Forwarding Control Protocol (PFCP).
- *N6*, connects the UPF and the DN.
- *N9*, connection between two UPFs.

2.2 ETSI NFV MANO

In Europe, NFVs standardization initiatives are done by European Telecommunications Standards Institute (ETSI) to provide a standardized approach for the deployment and management of virtualized network functions, which helps reduce operational costs and improve flexibility and agility in network infrastructure.

2.2.1 NFV

Usually, network functions are executed in the vendor's hardware. With NFV paradigm, these functions are virtualized and decoupled from the hardware. These functions can be encapsulated as virtual machines or containers. As a result, providers can run these functions in off-the-shelf hardware instead of a proprietary one. In addition, Virtual Network Functions (VNFs) can be connected between each other and with different solution providers if open interfaces are employed, which allows the building of complex network services. Moreover, if more capacity is needed, instead of deploying specialized hardware, a new VNF can be deployed in a fast and automated way. For these reasons, operators can build more scalable and flexible networks while reducing the costs of expensive hardware devices [17].

NFV can be deployed in data centers, network nodes, and end-user premises. However, to achieve provisioning, instantiation, energy efficiency, and interconnection between VNFs, ETSI designed the NFV Management and Orchestration (MANO) architectural framework [17].

Figure 2 depicts the functional blocks of the architecture and the reference points that connect each block. The main building blocks present in the architecture are:

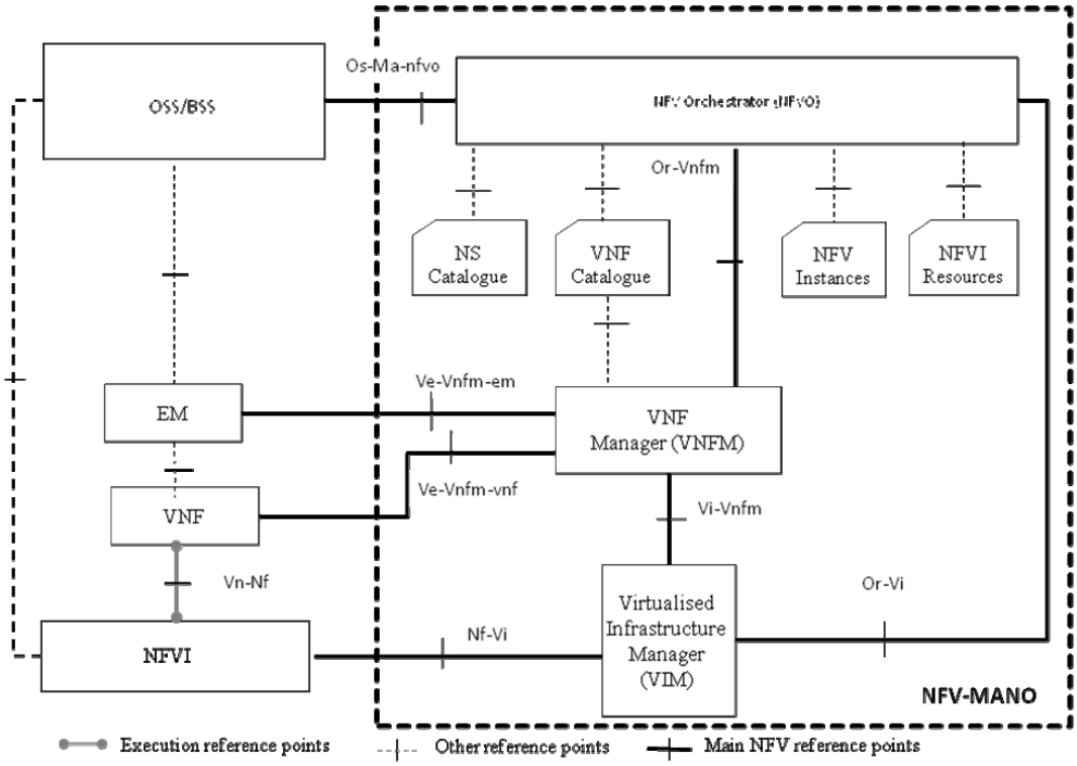


Figure 2: NFV MANO framework architecture (Extracted from [17]).

- Virtualized Infrastructure Manager (VIM) is responsible for controlling the NFV Infrastructure (NFVI), which is in control of physical resources like computing, storage, and networking and software resources such as the hypervisor. Available functionalities are the connection between the virtualized resources with the physical ones.

- VNF Manager (VNFM) manages the lifecycle of VNF instances, such as instantiation, modification, scaling, termination, and upgrading.

- NFV Orchestrator (NFVO) is responsible for the orchestration of resources and Network Services (NSs). The NS orchestrations include VNF deployment, interconnections, and lifecycle management. The network orchestration includes monitoring the allocated resources. Moreover, it will provide management of multiple VIMs and NFVOs.

The other functional blocks are:

- OSS/BSS, which are the combination of applications and systems that a service provider uses, these applications and systems can implement new functionalities to the MANO framework by communicating through the reference point (Or-Ma-NFVO).

- Element Management has the role for Fault, Configuration, Accounting, Performance, and Security (FCAPS) for a VNF.

- Repositories store information that is accessible through reference points to other functional blocks.
  - VNF catalog is a repository that supports the management of VNF Packages.
  - NS catalog provides a repository of all Network Services through the management of NS deployment templates, which describes the services and deployment services. This is used by the NFVO to instantiate a NS, that can be formed by many NFVs, Physical Network Functionss (PNFs), Virtual Links (VLs), and VNF Forwarding Graphs.
  - NFV instances is a repository with records about Network Services instances and VNF instances.
  - NFVI resources keep information about available, reserved, and allocated resources in NFVI environment.

2.3 SDN

SDN is a paradigm to make networks programmable through a controller, which is programmable software that can be deployed on servers and has a global vision of the network topology. Based on this centralized vision, it diffuses traffic rules or updates policies. In these networks, the control plane and data plane are separated, transforming the data plane devices into simple forwarding devices that are able to run in commercial off-the-shelf (COTS) hardware, while the forwarding decisions are programmed by the controller rules. Because traditional networks are built with specialized routers and switches with closed and proprietary code running on them, it is, therefore, difficult to test/develop new protocols and solutions. As a result, network management becomes difficult since changes need to be made by manual reconfiguration of individual elements. Moreover, automation and dynamic adaptation can provide failure solving and load balancing. The key element of this paradigm is the separation of control and data plane, which provides programmability through open APIs. A possible SDN architecture is shown in Figure 3. The northbound uses Representational State Transfers (RESTs) APIs while southbound communication uses OpenFlow protocol, which is usually built on top of Transport Layer Security (TLS) to provide a secure OpenFlow channel [18–20].

The SDN approach enables the following requirements [19]:

- Adaptability - Networks must adjust dynamically according to the application's needs, policies, and network conditions.
- Automation - Changes such as configurations and provisioning must be done automatically in order to reduce manual work and errors.
- Maintainability - Deployment of new applications is transparent.

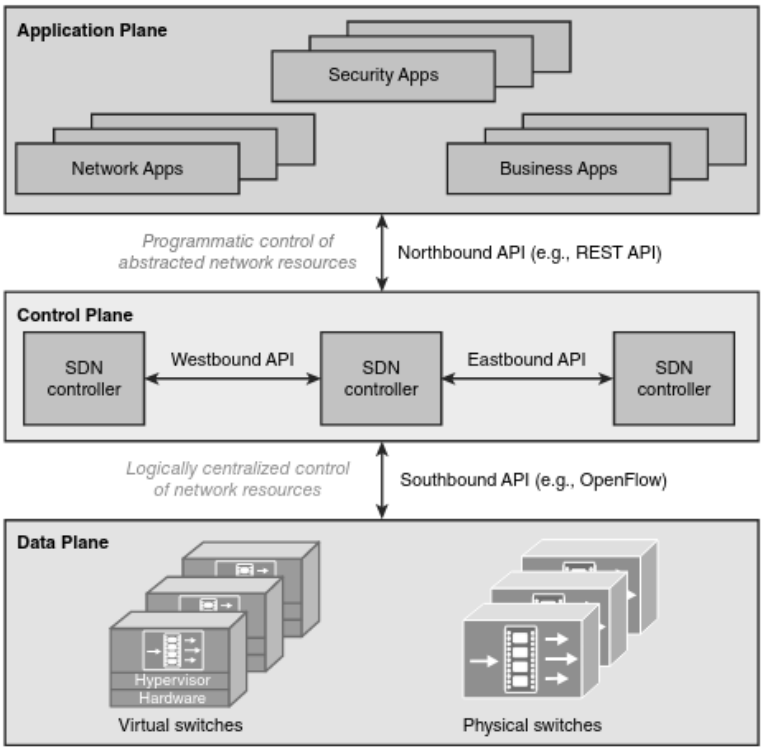


Figure 3: SDN architecture (Extracted from [19]).

- Model Management - Network management software must allow management of the network at a higher level rather than implementing changes through individual network elements.
- Mobility - The control plane must take into account mobile user devices and virtual servers.
- Security - Applications should have security integrated.
- On-demand Scaling - The network must have the ability to scale up and scale down.

2.4 Emulators and Simulators

Network testbeds can be implemented using simulators and emulators. Whereas the first one allows engineers to replicate the behavior of a network, an emulator allows them to introduce real devices and applications into a simulated network to mimic the behavior of a live network.

2.4.1 Simulators

In simulators, the operations of real devices and their interactions are modeled based on mathematical formulas, providing the fundamental behavior of a system without simulating all the details. Simulators must be validated in order to be accurate and results are not different from real-world usage [21]. Usually, a

network simulator is a closed environment that does not allow communication with external devices. On the other hand, However, simulators are slower than real-world environments. On the other hand, simulators can be slower than real-world environments. For instance, 1 millisecond ($1ms$) in simulator time can last much longer in real time.

Simulators are divided into two categories:

1. Physical layer simulators - Usually, simulators related to 5G are physical layer simulators. In other words, they are interested in measuring physical-layer quantities/design (spectral efficiency, antenna layout, transmission schemes, etc.), and layers above the MAC layer are usually not modeled [7, 22].

2. End-to-end simulators - End-to-end simulators are interested in the performance of application-level quantities (e2e delay, the throughput of user transactions, etc.) and include models of application logic and layer-2 above protocols, as well network equipment running those protocols [7, 22].

2.4.2 Emulators

Emulators are used in real-time and real devices and applications can communicate with simulated devices. Therefore, it is capable of retrieving real-time statistics.

Simu5G can run as an emulator, thus it is able to communicate with external devices and be interoperable with other frameworks. For example, the core network can be emulated by other software, and mobile edge computing frameworks like Intel Smart Edge (previously, OpenNESS project) can communicate with the 5G network emulated by Simu5G.

2.5 Simu5G

Whereas there are tools to simulate the physical layer (Vienna 5G, Matlab, etc.), a novice tool that provides 5G end-to-end simulation is Simu5G, which is based on the OMNeT++ event simulator. OMNeT++ is a simulation library and framework, firstly for building network simulators. Due to OMNeT++ providing a component architecture for models, reusability and easier integration with different OMNeT++ base frameworks can be reached. For instance, Veins (Vehicular Network simulation framework) can be integrated with Simu5G.

Alternatively, there are other e2e simulators for 5G, such as 5G-LENA, and 5G-air-simulator. Nevertheless, these simulators do not provide some functionalities. 5G-LENA does not provide Frequency Division Duplex (FDD) and both do not enable D2D applications and Non Standalone 5G deployment. On the contrary, 5G-air-simulator provides Massive MIMO and broadcast functionalities.

A significant annotation of end-to-end simulators is the lack of scalability due to simulating the whole protocol stack in all nodes. Simu5G solved scalability issues by including heterogeneous GNodeB. However,

the GNodeB light-cell mode does not simulate the 5G NG protocol stack. As a result, overhead is reduced but provides interference. Simu5G was validated according to scenarios presented in 3GPP evaluation document [7, 22].

Simu5G is a discrete-event simulation library for 4G/5G New Radio networks based on the OMNeT++ framework. The simulation library in OMNeT++ is a collection of C++ classes that provide the basic building blocks for creating simulation models. It includes classes for creating modules, messages, gates, and connections, as well as classes for managing the simulation itself, such as the simulation kernel and event scheduler.

One of the key features of the simulation library is its support for hierarchical modules. This allows developers to create complex models by combining simple modules into compound modules, and then combining those compound modules into even larger structures. The library also supports message passing between modules, which allows modules to communicate with each other and exchange data.

Another important feature of the simulation library is its support for parameterization. Modules can be defined with parameters that can be set at runtime, allowing developers to create flexible models that can be easily customized for different scenarios. The library also includes support for topology description, which allows developers to define the structure of their models using a simple text-based language.

The simulation library also includes support for result recording and analysis. Simulation results can be written to output files in a variety of formats, including line-oriented text files, output vector files, and output scalar files. These files can then be analyzed using a variety of tools and programming languages, including Matlab, GNU R, Perl, Python.

2.6 Discrete Event Simulation

A discrete event simulation is characterized by having the following components:

- State - Set of variables that represent properties/variables needed to describe the modeled system.
- Event - An occurrence in a specific time that modifies the state of the system being modeled.
- Clock - Variable responsible for keeping the current value of the simulated time in the model.
- Events list - A list containing information about events and the time that they will occur.

In a discrete event simulation, the system and its behavior are represented as a sequence of events, and time advances in discrete jumps between instances of event occurrence. In other words, the clock is set to the instant of the beginning of the next event, and no changes or time progression are considered between events. Therefore, time progresses only when a change occurs in the system. As a result, the simulation time is highly dependent on the occurrence of events. In comparison to continuous systems, state changes occur continuously over time.

Events in a discrete event simulation are instantaneous and can be scheduled for specific times, and a list of upcoming events is maintained. When an event occurs, it triggers a change in the simulation's state.

2.6.1 Discrete Event Simulation in OMNeT++

In OMNeT++, the event loop is the core of the simulation engine. It is responsible for scheduling and executing events in the simulation from the Future Event Set (FES). Events are scheduled to occur at a specific simulation time, and the event loop makes sure they run in the right order. [23].

Events are processed in strict timestamp order to maintain causality, which ensures that no current event may have an effect on earlier events. Processing an event involves calls to user-supplied code.

In summary, the event loop in OMNeT++ uses schedulers to manage the scheduling of events, and the FES to keep track of scheduled events and ensure they are executed in the correct order.

The simulation works according to the following order:

1. Initialize: Set up the network, initialize state and statistic variables, and insert initial events to the FES.
2. While the FES is not empty and the simulation not yet complete.
    a) Fetch the initial event from FES.
    b) Update the clock with event time.
    c) Process the event according to user logic, such as updating the state and inserting/deleting new events.
3. Finish simulation: Record statistics information and enables to perform of any necessary finalization after the simulation has finished.

2.7 Core Network

This section presents projects that are developing 5GC in order to be fully compliant with 3GPP releases.

2.7.1 OAI

The OAI Core Network (CN) development has as objective to be fully compliant with 3GPP. At this point, the OAI CN has the following CN entities functional AMF, SMF, NRF, AUSF, UDM, UDR, and UPF. Additionally, OAI CN supports registration, deregistration and session establishment, modification, and release procedures. Moreover, network function registration and discovery are also supported. Therefore, the entities

can be deployed, scalable, and adapted to the network needs according to Network Slices parameters and data network name. Some limitations are location services, IPv6 support, and slice functions that are only available with basic functionalities. Some components such as AF, UDSF, PCF, and NEF are not implemented yet [24, 25].

2.7.2 free5GC

Free5GC is written in GO language, and it is compatible with Ubuntu. The objective is to implement the core defined in 3GPP Release 15 (R15) and beyond. Currently, it is fully functional. The project implements all the CN entities except UDSF, AF, and NEF and are not expected to be implemented near future. Moreover, it has 5GC Orchestrator integrated, supports applications services, and has network services and slicing like OAI CN. [9, 24].

2.7.3 Open5GS

Open5GS is an open implementation of 5GC and Envolved Packet Core (EPC), written in C language, and it is 3GPP Release-16 compliant. Open5GS includes the core entities like OAI CN, IPv6 support, and on the contrary, with OAI CN implements PCF component, which operators can create and deploy network policies in real-time and prioritizing types of traffic. Although advanced 5G Standalone core functionalities, such as slicing, are currently under development [24, 26].

2.7.4 Magma

Magma is an open-source solution that provides a mobile core network (5GC and EPC) developed by Facebook Connectivity in order to simplify the deployment of cellular networks in rural areas [24, 26]. Magma has three components:

- Access Gateways - Includes core network, which provides network services. Can be deployed on the cloud or next to radio equipment
- Orchestrator - It is a cloud service to monitor and apply configuration changes.
- Federation Gateway - It is a proxy between the Magma core running in the access gateway and the network operator infrastructure.

All these projects implement real cores, since no elements are modeled or abstracted.

2.8 Summary

The fifth generation of mobile networks, known as 5G, aims to address the growing demands of mobile traffic and new user-oriented mobile multimedia applications, such as mobile video conferencing, streaming video, augmented reality, and online gaming. This is accomplished through various technology enablers, including mm-wave, Massive MIMO, Small Cells, and D2D communication. These advancements are expected to increase capacity, improve energy efficiency, and reduce latency, thus enhancing the overall user experience. Furthermore, 5G has the potential to benefit emerging technologies, such as the IoT and Augmented Reality.

5G mobile networks use technologies like SDN and NFV to improve connectivity and reduce latency. NFV is a paradigm in which network functions are virtualized and decoupled from the hardware, allowing them to run on off-the-shelf hardware instead of proprietary hardware. This makes networks more scalable and flexible while reducing costs. Additionally, VNFs can be connected between each other and with different solution providers using open interfaces, allowing for the building of complex network services. This enables faster creation of new services.

On the other hand, SDN is a network architecture in which the control plane and the data plane are decoupled, allowing for network control to be directly programmable and the network's global state to be visible. This allows for more flexibility and agility in managing the network and implementing new services.

The combined use of NFV and SDN in 5G networks can provide significant advantages, such as cost savings, increased agility, and improved scalability. In addition, these technologies enable more flexible and efficient use of network resources, making it easier to add new services and adapt to changing network conditions. Furthermore, using virtualized and software-defined network functions allows for faster deployment and more accessible network management, resulting in faster and more efficient service delivery.

Adopting these technologies has also led to creating a more open environment for innovation in the 5G industry. They separate proprietary hardware and software, allowing for the emergence of several open-source solutions for 5G core implementation, such as the OAI, free5GC, Open5GS, and Magma projects. The flexibility and configurability offered by these open-source solutions allow for greater experimentation and customization in deploying 5G networks, fostering innovation and driving the development of new technologies.

When implementing 5G RAN, one of the solutions available are emulators and simulators, which are used to test and evaluate the performance of 5G networks. These tools mimic real-world network conditions and environments, offering a controlled and repeatable testing environment for 5G systems. This allows developers to test the 5G network's performance, capacity, and scalability before deploying it in the real world. Furthermore, emulators and simulators are also used to test new features and services and identify and resolve potential issues before they arise in the live network.

Chapter 3

Literature Review

The literature review in this chapter initially focuses on the role of virtualization and softwarization in 5G networks, highlighting the trends, technologies, and frameworks that have been developed. The objective is to gather information about examples of 5G testbeds architectures that have already been implemented and their capabilities.

Softwarization is the use of a given capability that is implemented through software rather than traditional hardware. This provides flexibility, since deployment is automatic and dynamic configuration is possible, as it is only necessary to update software that is not dependent on vendor's hardware. Virtualization, on the other hand, is the process of abstracting resources that were previously given in hardware and transferring them to software. This allows efficient resource allocation, traffic adaptation, and easier management due to abstraction. As a result, Capital Expenditure (CAPEX) and Operational Expenditure (OPEX) can be reduced.

New principles based on softwarization and virtualization, such as SDN, Multi Access Edge Computing (MEC), network virtualization, and cloud computing, are being proposed in 5G architecture to bring enhancements. According to the literature, diverse open-source software, frameworks, and libraries have emerged based on these principles.

The chapter then goes on to discuss how softwarization technologies are used in the different 5G components. It covers the Radio Access Network, where the main differences in the protocol stack occur in the Media Access Control Layer (MAC) and Physical Layer (PHY) layers, and the need for cloud computing and SDN principles in RAN architecture. The chapter also covers the Core Network, where a service oriented design with well-defined APIs is the main innovation, and Transport Networks, where SDN technology will be critical to adapt the network according to the RAN needs.

Finally, the chapter discusses previously established 5G testbeds and their capabilities, including the

various architectural designs implemented.

3.1 Towards softwarization and virtualization of 5G networks

Softwarization is the use of a given capability that is implemented through software rather than traditional hardware. This provides flexibility, since deployment is automatic and dynamic configuration is possible because it is just necessary to update software that is not dependent on the vendor's hardware. Virtualization is the process of abstracting resources that were previously given in hardware and transferring them to software. This allows efficient resource allocation, traffic adaptation, and easier management due to abstraction. As a result, Capital Expenditure (CAPEX) and Operational Expenditure (OPEX) can be reduced. For these reasons, virtualization and softwarization are key enablers for 5G networks in order to provide agility, flexibility, fine-grain control, and reliability [27].

New principles based on softwarization and virtualization, such as SDN, MEC, network virtualization, and cloud computing, are being proposed in 5G architecture to bring enhancements. According to [24], diverse open source software, frameworks, and libraries emerged based on these principles.

Next, it is discussed how softwarization technologies are used in the different 5G components.

3.1.1 Radio Access Network

The main differences in the protocol stack in RAN, by contrast with the previous generations, occur in the MAC and PHY layers since the use of mm-wave. Although the main differences were driven by mm-wave, 5G specifies a family of waveforms. At the PHY layer, the flexibility introduced by the setting of waveform parameters (numerology) allows the network to adapt conforming to the requirements of various users and services [28]. Alternatively, in LTE, in which there is only one waveform, therefore single-numerology.

Despite the improvements at the protocol stack level, there is a need for cloud computing and SDN principles in RAN architecture. Furthermore, with the increase in traffic, operators need to put more resources and invest in more specialized infrastructure (small cells and MIMO). In addition, to address the energy efficiency challenge, it is important to note that the base station is the equipment in the 5G architecture that consumes more power. Moreover, placing more small cells to extend coverage and capacity can lead to overhead in the network due to frequent handovers [29].

For these reasons, the RAN architecture needs to be redesigned in order to reduce hardware constraints and costs while improving flexibility for upgrades in the infrastructure, coverage, and energy efficiency [12]. A new innovation in 5G was the Central Unit (CU)/Distributed Unit (DU) split, where the BS is divided into two logical units, the CU with higher layer stack functionalities and DU with lower layer stack functionalities, which can be distributed. In addition, the lower part of the physical layer can be separated from the DU in a standalone Radio Unit (RU). However, in the early generations, these components were located together.

Therefore, it can be seen that the SDN principles are in this split, where RU operates as transceiver and all the control and processing operations are executed through open interfaces and API. Network virtualization is present in RAN since there are frameworks that provide these functionalities virtualized decoupled from hardware. With this virtualization, cloudification of the RAN can be obtained by deploying in the cloud a Baseband Unit (BBU)(CU/DU) pool that is responsible for allocating resources to RUs. Examples of open source frameworks that allow the deployment of a virtualized RAN are OAI and *srsRAN*. These frameworks implement the UE and GNodeB, although OAI allows deploying these components in Standalone mode and BS deployment in split mode [24].

3.1.2 Core Network

The 5G core has, as the main innovation, a service oriented design with well-defined APIs, being the network services deployed as VNFs and control and user plane separation into different network functions following the SDN principles. This enables the deployment of VNFs in different locations in the network architecture based on different requirements such as latency, storage capacity, and processing [24, 30]. Another principle presented in 5GC is Control and User Plane Separation (CUPS) from the SDN principle in order to improve efficiency, user plane operations are distributed closer to users while still allowing centralized control plane functions [31]. Examples of open source frameworks that allow the deployment of a virtualized core are OAI, Open5GS, and free5GC.

3.1.3 Transport Networks

In addition to the radio access and core networks, the transport network will be critical in 5G in order to achieve flexibility and adaptability. Here, SDN technology will be fundamental to adapt the network according to the RAN needs. Moreover, with RAN coordination, mobility, and load balancing can be efficiently coordinated [30, 32].

3.1.4 Open virtualization and management frameworks

Another new technology is an NFV MANO framework, whose main goals are management, orchestration, and deployment of network resources like computing, storage, and virtual machines/containers resources on the cloud. Consequently, the complexity of the underlying resources is abstracted and easier to manage functionalities such as network configuration, monitoring, maintenance, fault management, and security. Therefore, this framework has a crucial role in the management and deployment of e2e networks [24, 30].

Some popular MANO frameworks are ONAP, OSM, and Open Baton. All these frameworks support virtual machines and containers as infrastructure, VNFs as network services, and have external APIs to communicate. The differences are in the technologies. OSM supports containers with Kubernetes, Open

Baton with Docker, and ONAP with both. All are compliant with ETSI MANO, except ONAP. OSM and ONAP support communication with REST APIs. On the other hand, Open Baton supports JAVA SDK. All these frameworks provide network slicing, which is a multi-tenancy virtualization approach in which logical networks are isolated from hardware and software components and distributed as slices to tenants. This is possible because the orchestrator can allocate the virtualized resources to different slices, according to different requirements [24].

3.1.5 Mobile Edge Computing

Because of the intention to virtualize 5G network components, another enabler is MEC which brings critical network architecture components closer to users. For instance, virtualized applications can be deployed at the RAN, closer to the user. Therefore, MEC can enhance 5G latency and throughput. Some frameworks that enable MEC are LL-MEC and LightEdge. LightEdge is interoperable with frameworks like Open5GS, while LL-MEC can provide network slicing [24].

3.2 5G testbeds

This section describes testbeds presented in the literature and their capabilities.

3.2.1 Open and Programmable 5G Network-in-a-Box

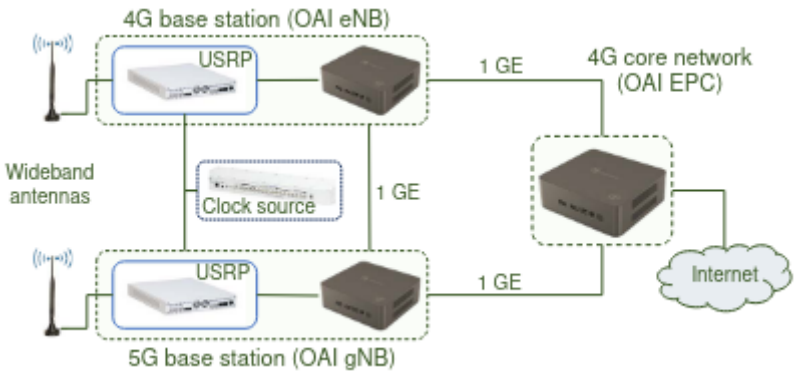


Figure 4: Architecture of the testbed "Open and Programmable 5G Network-in-a-Box"(Extracted from [33]).

The testbed proposed in [33] is a network-in-box solution and shows an architecture based on open software stack and general proposed hardware to demonstrate end-to-end connectivity in 5G Non Standalone mode and 4G LTE mode. The eNodeB, GNodeB and the EPC are based on OAI. This testbed operates at sub-6 GHz frequency and is able to evaluate based on latency and throughput performance in different frequency bands.

3.2.2 5G Testbed Development for Network Slicing

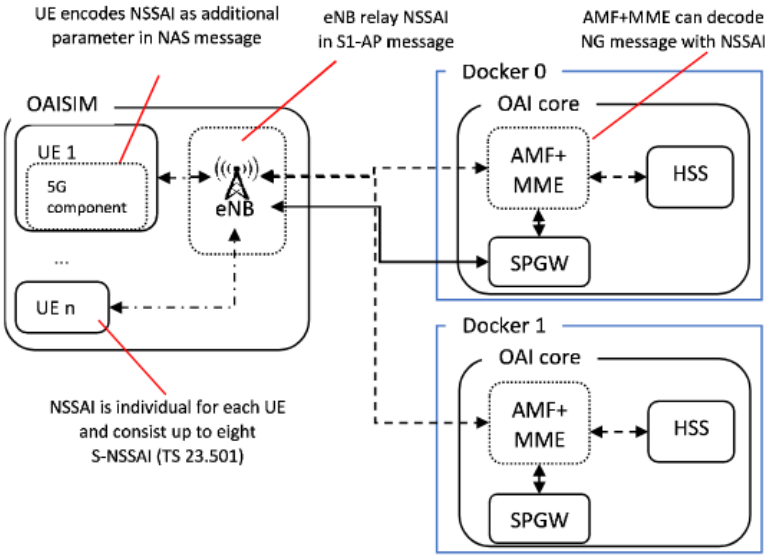


Figure 5: Architecture of the testbed "5G Testbed Development for Network Slicing"(Extracted from [34]).

This testbed proposed in [34], by contrast with the previous one, uses cloud computing and enables network slicing, also RAN and Core uses OAI modules. However, instead of direct deployment on general purpose hardware, the RAN and Core are deployed in Docker containers, and there are two instances of core to provide slicing. Network slicing is achieved through virtual networks by Docker Containers, with modified UE, core and gNodeB by the authors to provide configuration and management of network slices. The objective was to study and test performance issues, security and how slice selection works in RAN and core protocols.

3.2.3 Testbed for 5G Connected Artificial Intelligence on Virtualized Networks

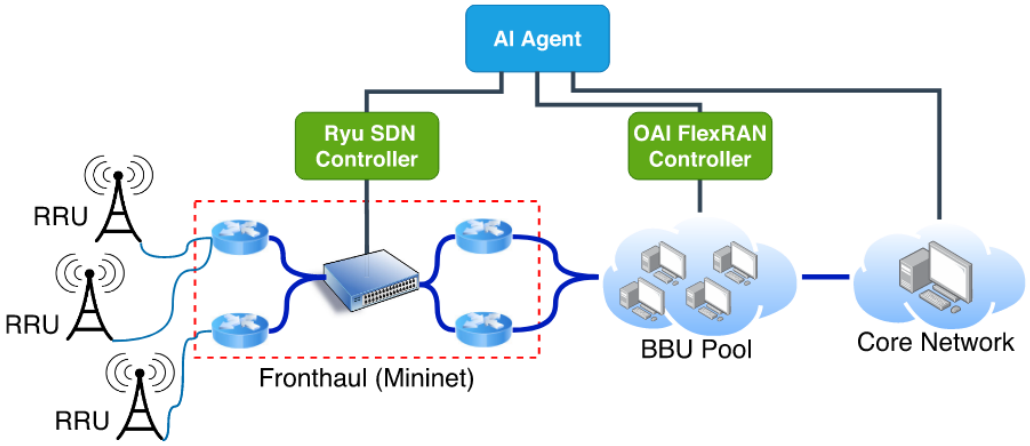


Figure 6: Architecture of the testbed "5G Connected Artificial Intelligence on Virtualized Networks"(Extracted from [35]).

The [35] testbed, in comparison with the previous ones, the main goal is to enable Machine Learning (ML) applications with the aim of based on Artificial Intelligence (AI) decisions to optimize the network. Likewise, this testbed uses OAI modules for RAN but with split architecture with docker containers instead of monolithic implementation and Free5GC to implement core. Moreover, it has a virtualized Transport Network (TN) emulated using Mininet [36]. This testbed provides network slicing at RAN level due to the usage of a controller in RAN (FlexRAN [37]), which will set up the RAN functions and policy configuration that will be included in the slice. The information retrieved by this controller and TN controller (Ryu SDN controller) is used by the AI agent ML models to deduce various slice configurations. The slicing process is accomplished by communicating through FlexRAN API to indicate, for example, what resources to allocate, what algorithms should the slice use, etc. The other application tested was how it should VNFs be positioned to meet the specifications of a network slice. The architecture does not follow MANO implementation in order to provide a lightweight and simple architecture, for this reasons Kubernetes is used as orchestrator. On the contrary, this testbed did not need to modify modules to enable slicing. Alternatively, the FlexRAN platform that allowed the slice management was used. Yet, end-to-end slicing is not provided due to lacking at core and TN.

3.2.4 Virtualized C-RAN with Mininet and OAI Supporting Flexible Network Topologies

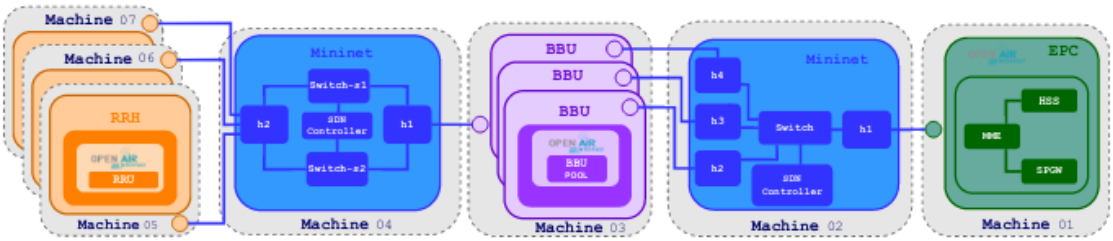


Figure 7: Architecture of the testbed "Virtualized C-RAN with Mininet and OAI Supporting Flexible Network Topologies"(Extracted from [38]).

The testbed [38] shown in Figure 7 provides a virtualized cloud RAN with a BBU pool and Remote Radio Head (RRH) pool. These modules are implemented by OAI in two separated virtual machines, and likewise with the previous uses Mininet to emulate transport networks (backhaul and fronthaul) with SDN in virtual machines. The core network is in another virtual machine using OAI to emulate EPC Core. This testbed enables experimentation of different configurations for fronthaul to test congestion control, quality of service metrics, and resource management under different network topologies and conditions.

3.2.5 A Cloud-based SDN / NFV Testbed for End-to-End Network Slicing in 4G / 5G.

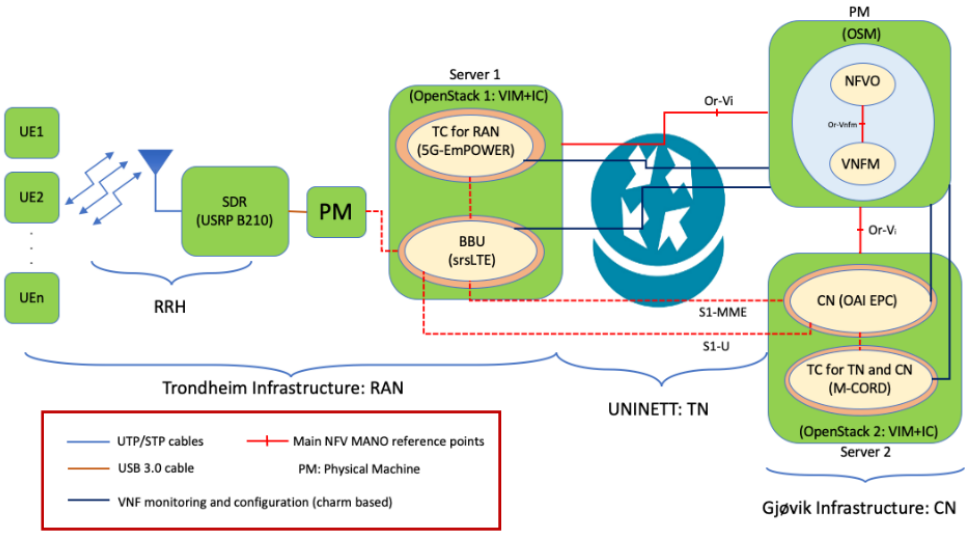


Figure 8: Architecture of the testbed "A Cloud-based SDN / NFV Testbed for End-to-End Network Slicing in 4G / 5G"(Extracted from [39]).

In the testbed [39] displayed in Figure 8, OSM is used as NFV orchestrator, since it allows several VIMs which enables cross-location, OAI as core, srsLTE as RAN and SDN Controllers in RAN, core and TN to enable end to end slicing. In addition, this testbed has ML toolkits integrated because of using 5G-EmPOWER as SDN Controller, which has these tools integrated.

3.3 Summary

In summary, 5G is becoming more virtualized and software-based to increase flexibility and programmability, which is leading to more significant disruptions than in previous generations. This presents both challenges and opportunities for innovation. As a result, many testbeds are being developed to study various use cases, improvements, and architecture decisions for 5G, such as deployment scenarios, such as the location of NFs and MEC, new algorithms, network slicing, and the use of AI.

The first two testbeds mentioned in the previous text are among the earliest in the literature and do not have the ability to scale. The first testbed relies on hardware and does not utilize software-based approaches, while the second testbed focuses on the concept of network slicing using docker technology, but does not achieve a high level of complexity or scalability. However, it is possible to achieve more scalable network slicing using current technologies.

The testbeds 3, 4, and 5, since they use OAI RAN, present some limitations in representing a large network because there are scalability limitations in the use of several BSs and UEs. Moreover, both testbeds 3 and 5 need the use of Software Defined Radio, which increases the costs and hardware requirements.

This dissertation proposes to study the interoperability of a new tool called Simu5G, which allows for the integration of various 5G enablers, such as MEC, D2D, and vehicular networks, without the need to connect to other frameworks. In addition, Simu5G allows for the study of how network conditions can impact the development of applications by enabling the possibility of mobility, support for different numbers of UEs and GNB, and customization of radio layer functionalities. As a result, it is expected the study of new use cases would be more accessible and extensible using this testbed.

Chapter 4

Conceptual Solution

This chapter proposes an architecture to build a virtualized 5G network using open-source available components deployed in virtual machines. The chapter starts by explaining the decisions leading to the selected components, followed by a description of those components. The chapter ends with a specific proposal on how to connect them for tests.

4.1 RAN and user terminal emulation/simulation

The work aims to implement a fully virtualized 5G network. It can be seen that the RAN is a core element of the system. Emulators and simulators were sought since they do not require physical Radio Frequency (RF) hardware or a COTS terminal. This would increase costs, and the study of hardware compatibility would be necessary. Moreover, the objective is to implement a fully virtualized testbed. Thus, to simulate the RAN, RAN products were studied:

- *my5G-RANTester*[40], it is a tool used to evaluate and understand 5G functionalities by providing different tests and emulating control and data planes for the UE and GNodeB.

  This emulator provides conformance tests for evaluating the following system procedures:

  - UE Registration
  - GNodeB Registration
  - Packet Data Unit (PDU) session request

  After successful registration, it is possible to check the connectivity with the internet through the tunnel created between RAN and 5GC (data plane). Moreover, the performance tests available

measure the latency of UE registration and AMF responses per second. Additionally, a stress test with queued UEs one at a time, which simulates a GNodeB with multiple UEs is available to test the scalability. The necessary configurations to allow the connection between interfaces and network elements is done through a *yaml* file.

Overall, this framework is more concerned with the 5GC and the linking between this and the access network. Therefore, a simulation of the wireless channel is not provided. Because of this, *my5G-RANTester* main cornerstone is allowing the study of 5GC functionalities by observing packets through *wireshark*, black-boxing testing the functionality and the compliance with the 3GPP Release 15, and evaluating the performance. Despite focusing on 5GC functionalities, the network functions mainly tested are AMF, SMF, and UPF since those are directly related to the RAN through the NAS and NGAP protocols. Therefore, the RAN level only offers a few applications [41].

- *gNBsim* [42], it is another tool that simulates GNodeB and UE by generating NAS and NGAP messages and supports the following procedures to be tested:
  - UE Registration;
  - UE Initiated PDU Session Establishment;
  - UE Initiated De-registration;
  - AN Release;
  - UE Initiated Service Request;
  - Network triggered PDU Session Release;
  - UE Requested PDU Session Release;
  - Network triggered UE Deregistration.

  Moreover, it is capable of sending generated Internet Control Message Protocol (ICMP) requests over the existing data plane.

  Altogether, this tool is similar to the previous, being the primary objective testing 5GC functionalities. Alternatively, each test can be configured to run in parallel or sequential order, and the number of data packets to be sent can be set up. Moreover, provides a profile feature that provides automation to specify and release the tests and customize the tests by compounding and making use of the standard tests provided.

- *UERANSIM* [43] simulates a 5G Standalone UE and RAN. It implements NAS and NGAP protocol and system procedures such as UE and Network initiated De-registration, UE initiated PDU session establishment, etc.

At the user plane, it implements GTP protocol, with ipv4 and ipv6 supported. The radio protocols below the Radio Resource Control (RRC) layer are not implemented, being partially simulated over User Datagram Protocol (UDP). Nevertheless, the primary RRC methods are available.

The necessary configurations to allow the connection between interfaces and network elements are done through a *yaml* file.

Conversely, *UERANSIM* is less focused on testing than the previous tools, even though it can serve that purpose. The main difference is the data plane usage with several types of applications, thus it is possible to use the N3 connection to access the internet using applications such as *ping*, *tcpdump*, Hypertext Transfer Protocol (HTTP), Secure Socket Shell (SSH), and *wget*, whereas the other tools only allow using ICMP.

- *OpenAirInterface* is an open platform for implementing the core and RAN of a communication system. Its RAN project has a real-time approach, where elements are real, requires hardware to implement the software-defined radio frontend, and is compatible with commercial terminals.

  It also offers a deprecated emulation platform (*OAISim*) for emulating UEs and GNodeB, with communication between them based on an emulated physical channel. The environment can be configured with various parameters, including path loss, channel model, antenna description, system bandwidth, and frequency. Also, the network topology can be configured with mobility models. Traffic load types such as Voice-over-IP (VoIP), Machine to Machine (M2M), and gaming are available, with packet trace in *Wireshark* at layer 3. Configurations can be programmed in OAI scenario descriptors (OSDs).

  The platform offers parallel emulation, enabling several GNodeB and UEs to be virtualized on the same or different machines, improving scalability. In addition, it displays low-level stack metrics on a dashboard and traffic and application metrics through standard output logging.

  Scalability is achieved through parallelism and dynamic allocation of PHY abstraction and channel modeling on multiple threads, with a PHY abstraction mode to make emulation faster.

  Two simulators are available: the RF Simulator for testing without an RF board and the L2 nFAPI.

  The RF Simulator runs faster/slower than real-time based on CPU and supports ”noS1”mode where generated IP traffic loads are sent and received between gNB and UE in tunnel interfaces (”oaitun”) by applications like *ping* and *iperf*. The ”phy-test,”where random Uplink (UL) and Downlink (DL) traffic is generated at every scheduling opportunity. Some exceptions can occur since OAI UE are not thread-safe. Thus, it is necessary to run with multi-threading restrictions.

  The L2 nFAPI Simulator uses an NFAPI interface to connect the ENodeB and UEs. NFAPI is a standard interface that defines a set of messages, commands, and parameters for exchanging information between the PHY and MAC layers, such as scheduling and channel state information.

For example, with NFAPI, the MAC layer can request data from the PHY layer, and the PHY layer can send data to the MAC layer and request control information.

This direct communication through NFAPI instead of the PHY layer offers greater efficiency and flexibility. In addition, the simulator aims to support multi-UE simulation with many UE (ideally up to 255), though currently, it only supports 16.

- *Simu5G* [44] it is a simulator based on the OMNeT++ framework, and it is able to run as an emulator.

  By contrast with the previous emulators/simulators, it provides mobile network applications such as VoIP, Constant Bit Rate (CBR), and Trace-based Video-on-demands (VoD).

  In terms of extensibility, due to being integrated into the OMNeT++ framework, other models from the INET framework can be used to extend, such as VEINs, which allow using vehicular networks. Furthermore, the emulation mode can integrate other frameworks and applications outside INET and use actual-world conditions. Besides, it uses a modular architecture concept from OMNeT++, which turns easier to extend, allowing the development of new modules, algorithms, and protocols.

  Simu5G is based on two types of files, NED and INI. These are the high-level files that configure the network. The low-level logic related to the behavior of the different components in a network is done through C++ modules.

  The NED file is declarative and describes the topology that represents the network. An example of a NED file is shown in Listing 4.1. In this file Simu5G models are imported to represent network elements, such as the UE, background cells and the channel control, which models the transmission medium between the connections where proprieties such as propagation models, path loss and signal attenuation are defined, while the connections are established using the ++ operator, which selects the first available unconnected interface for the element.

```
1    import simu5g.nodes.NR.NRUe;
2    import simu5g.nodes.backgroundCell.BackgroundCell;
3    import simu5g.world.radio.LteChannelControl;
4    import inet.node.ethernet.Eth10G;
5    import inet.node.inet.Router;
6
7
8    network NetworkName
9    {
10       parameters:
```

```
            int numUe = default(1);
            int numBgCells = default(0);
        submodules:
            channelControl: LteChannelControl {
                @display("p=50,25;is=s");
    }
    ...
     connections:
    //# Data Network connections
    server.pppg++ <--> Eth10G <--> router.pppg++;
    router.pppg++ <--> Eth10G <--> upf.filterGate;
    upf.pppg++ <--> Eth10G <--> gnb.ppp;

```

Listing 4.1: NED file excerpt example of Simu5G usage.

The INI file is where various parameters related to simulation and network elements are configured, as shown in Listing 4.2. The primary parameters that define the Simu5G network include UE mobility, numerology, bandwidth, which is determined according to the number of resource blocks. In addition other essential parameters include carrier frequency, transmission power, simulation time, and other parameters can be defined based on the specifications of Simu5G modules.

```
# Config name
[General]
# Simulation configuration such as images folder to use
# during simulation, time of the simulation and
# directories where the statistics will be saved
 image-path=../../../images
 output-scalar-file = ${resultdir}/${configname}/${repetition}.sca
 output-vector-file = ${resultdir}/${configname}/${repetition}.vec
 # Simulation Time
 sim-time-limit=20s
 **.routingRecorder.enabled = false

 # UE element definition of initial position and mobility
 *.ue[0].mobility.initialX = 450m
 *.ue[0].mobility.initialY = 350m
 *.ue[0].mobility.speed = 0mps
```

```
17      *.ue[0].mobility.initialMovementHeading = 0deg
18      *.ue[0].mobility.typename = "LinearMobility"
19
20      # Resource blocks
21      **.numBands = 100
22      # Transmission Power
23      **.ueTxPower = 26 # dBm
24      **.eNodeBTxPower = 40 # dBm
25
26      # Carrier frequency *.carrierAggregation.componentCarrier[0].numerologyIndex
        ↪  = 0
27      # Numerologia
28      *.carrierAggregation.componentCarrier[0].carrierFrequency = 2GHz
29
```

Listing 4.2: INI file excerpt example of Simu5G usage.

Additionally, INET modules define a number of widely used statistics in the NED file. Some of these statistics are recorded into scalars, histograms or vectors by default, offering a set of metrics.

The interaction with the previous emulators mentioned is done via Command Line Interface (CLI). Although *Simu5G* can be used through a terminal, the OMNeT++ provides a graphical runtime environment, which allows a way of visualizing and animating network components and the flow of messages, making it easier for beginners to understand a system at a higher level [45].

Simu5G provides some 5G enablers such as D2D communication, small cells, and MEC. As mentioned in Chapter 2 Simu5G is an e2e simulator but allows to specify low-level parameters such as different types of scheduling algorithms, carrier aggregation and other parameters at the MAC and PHY layer that are the layer with changes in relation with the previous generations as mentioned in Chapter 3.

With this framework, it is possible to study network planning by building a representation of real access networks by introducing mobility and interference and testing different applications.

The two most appropriate solutions for the work requirements are *OAISim* and *Simu5G*. *Simu5G* is based on 3GPP specifications and uses the OMNeT++ stack, while OAI SimuLTE uses the 3GPP stack. However, *OAISim* is outdated, and its documentation is incomplete, lacking detail and clarity. Thus not the most updated version of the RAN would be used. *Simu5G* emulator capabilities, on the other hand, have been thoroughly documented and tested, making it more reliable, including support for hundreds of UEs.

The use of the same code for both emulation and real-time mode in OAI is not practical for the current work, as it does not involve the use of real devices. *Simu5G*, on the other hand, provides flexibility in both emulated and simulated environments.

In OAI, OSDs do not support variable parameters, so when running an emulation, a different OSD must be prepared for each combination of parameter values. In *Simu5G*, there is support for variable parameters and automation to run different scenarios.

This work aims to build a e2e system to be the basis for 5G experimentation. In terms of ease of use, *Simu5G* provides more utilities and a more effortless ability to integrate new systems, test new applications, plan and visualize real scenarios and with the ability to provide metrics as the other alternatives. The out-of-the-box implementation of MEC is a key factor in *Simu5G*'s suitability as a system-level network solution. Furthermore, there is no literature on *Simu5G* being used as a RAN emulator connected to a 5GC.

Given these considerations, the choice of *Simu5G* as the most suitable solution for building an end-to-end 5G experimentation system is recommended.

4.1.1 5G Core

With regard to 5GC software modules mentioned in Chapter 2, all the publicly available modules studied, with the exception of *Magma*, follow 3GPP architecture. This is essential to follow open standards because it will facilitate interoperability with other solutions which follow the same. All software modules provide good documentation and active communities, although *Magma* has fewer examples of interoperability with other software. All have the main features implemented to deploy, so this testbed can be agnostic to the 5GC. At this moment, *free5GC* is in a more stable and advanced development maturity. For that reason, this was the software package of choice.

It is important to consider how the connection between the Simu5G can be made to connect to the N3 interface, since Simu5G only offers data plane support. Simu5G allows communication externally through a Virtual Ethernet (veth) device, which is created in pairs to connect a network namespace to another physical network device in another namespace, although it can be as standalone network devices [46]. A network namespace uses a copy of the network stack from the host in an isolated way by defining new routes, interfaces, and rules that only will apply to traffic inside the namespace without using the host rules. As a result, this is useful not to collide rules and offer logical division. However, a veth interface is not configured to use GTP-U protocol. Nonetheless, from the selection of RAN software previously described, all those solutions provide a TUN interface, a virtual interface to be used by user space programs without the need for a physical interface that connects through N3. So using the provided TUN by UERANSIM, whose usage can be manually utilized and offers more support than the alternatives, it is possible to connect Simu5G with the selected 5GC.

4.2 Virtualized 5G Network design

Figure 9 shows a block diagram of the 5G network to be implemented, there is a logical division with virtual machines to separate the RAN and 5GC, which makes sense since the core is stepping into cloudification in data centers. However, RAN can have some of its elements in the cloud. Moreover, the UE and Base stations will have to exist in urban areas with specific planning and fixed. Another reason for different VMs is the recommendation from the UERANSIM documentation to use RANs and 5GC in different machines.

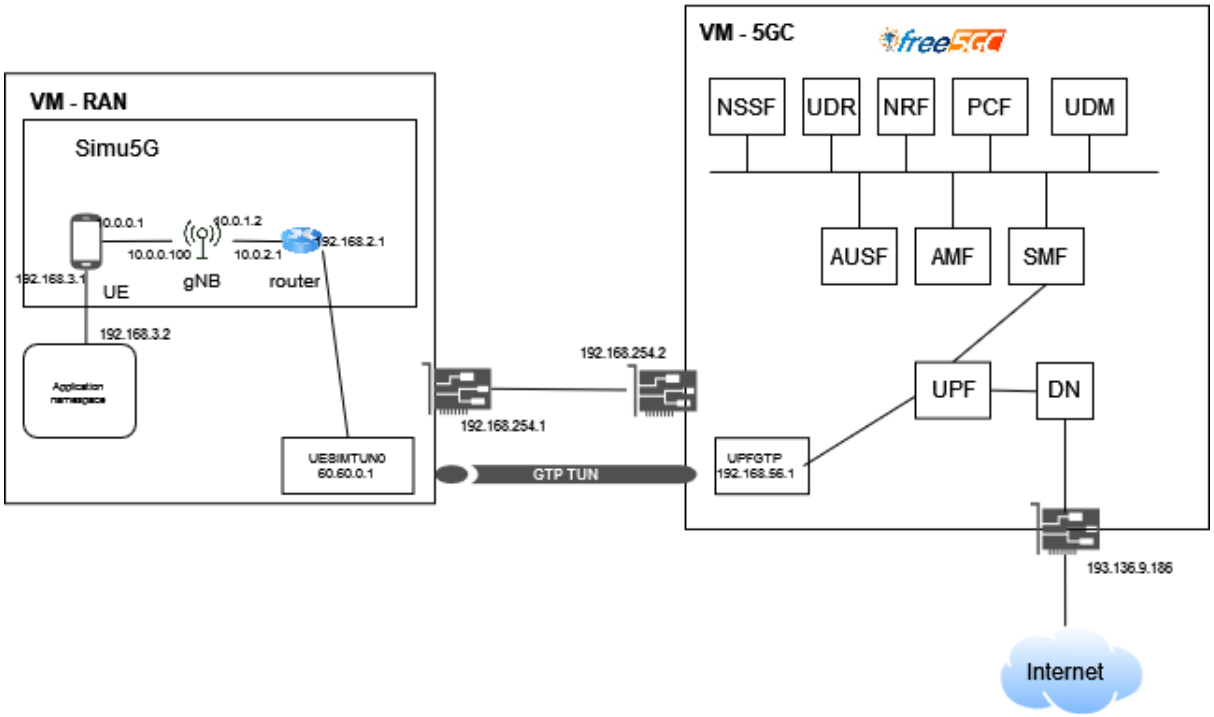
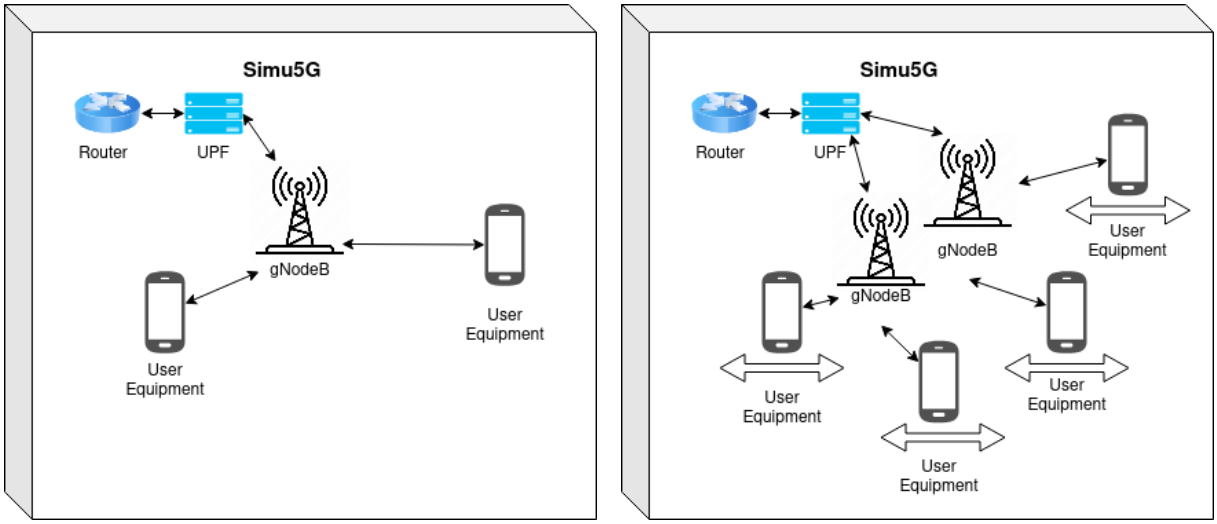


Figure 9: 5G System Design.

As shown, technologies such as MANO, SDN controller, and multi-split design alternatives, e.g. NF deployed in different locations, were not used. This option does not affect the objective of achieving an end-to-end connectivity solution and allows reducing complexity.

As mentioned, to implement the virtualized 5G network, sequential steps have been defined. According to [47], even the most simple testbed needs a considerable complexity of deployment, configuring, and monitoring due to the fast progress of related software. Moreover, interoperability with the different solutions is unclear [24]. Therefore, it is essential to set clear iterative goals. Firstly, two scenarios were developed to simulate an end-to-end 5G network using Simu5G, as depicted in Figure 10.

In Scenario 1, the network is simulated with two UEs and one GNodeB. The GNodeB is connected to the UPF, which acts as a gateway to other networks that are represented by the Router. As shown, the Simu5G simulates the data plane, and that is why only the User Plane Function (UPF) is represented as part of the 5GC. This scenario is used to explore the functionalities of the simulator. Wireshark will be used



(a) Scenario 1.

(b) Scenario 2.

Figure 10: Simu5G simulation scenarios.

to sniff packets and understand how the handover process is executed. In the second scenario, 10b more UEs are added, and different mobility patterns will be explored.

Chapter 5

Implementation

In this chapter, a specific implementation of a testbed supported by two virtual machines is presented. The goal is to provide all the steps required to instantiate and test the scenario proposed in the previous chapter (Figure 9). Two virtual machines were deployed on a server at the Department of Informatics at the University of Minho, which uses VMware ESXi as the hypervisor to host 5GC and Simu5G running as an emulator. The following procedure is based on [48–50].

5.1 Simu5G Virtual Machine installation

One VM was used to install Simu5G, with enough resources to meet the installation requirements. The next sections provide a detailed description of the procedure that was followed.

5.1.1 Simu5G Virtual Machine hardware

The Simu5G VM was deployed with 25 GB of storage, 4 GB of RAM, 2 vCPUs, using the operational system Ubuntu x86 18.04 LTS.

5.1.2 Simu5G prerequisites

Since Simu5G is supported by OMNET++ and INET framework, it is required to install OMNeT++ v6.0 pre11 version and INET v3.4.2.

1. To install OMNeT++ prerequisites the following commands are necessary:

```
$ sudo apt-get update
```

```
$ sudo apt-get install build-essential clang lld gdb bison \
  flex perl \
  python3 python3-pip qt5-default libqt5opengl5-dev \
  libxml2-dev zlib1g-dev doxygen graphviz libwebkit2gtk-4.0-37
$ python3 -m pip install --user --upgrade numpy pandas \
   matplotlib scipy seaborn posix_ipc
$ sudo apt-get install openscenegraph-plugin-osgearth libosgearth-dev
$ sudo apt-get install openmpi-bin libopenmpi-dev
```

2. With the prerequisites installed, the next step is to download OMNET++, unpack the archive and set the environment variables to run the OMNET++ using the terminal:

```
$ tar xvfz omnetpp-6.0pre11-src-linux.tgz
$ cd omnetpp-6.0pre11
$ source setenv
$ gedit ~/.bashrc
  ...
  export PATH=$HOME/omnetpp-6.0pre11/bin:$PATH
$ source ~/.bashrc
```

3. Finally, to configure and build, the commands below are used:

```
$ ./configure
$ make
```

4. To verify that everything operates properly, the following tests are executed:

```
$ cd samples/aloha
$ ./aloha
```

Once the binary is launched, the graphical environment with a set of simulation examples similar to the University of Hawaii radio network will appear, allowing the user to run simulations and verify that no errors have occurred.

5. With OMNeT++ working, the INETv4.3.2 package was downloaded and then extracted in the OMNeT++ workspace. Then with the Integrated Development Environment (IDE), the unpacked INET project was imported via *File -> Import -> Existing Projects to the Workspace* and built (*Project -> Build*).

```
$ tar xvfz inet-4.3.2.tgz
$ cp -r inet-4.3.2 omenetpp-6.0pre11/{workspace-name}/
```

6. Finally, the last step is to download Simu5G (`Simu5G-1.2.1.tar.gz`) and extract it next to the INET directory. Build and set Simu5G environment variables in Simu5G directory:

```
$ . setenv
$ make makefiles
$ make
```

An error occurred during the building phase because some Python modules required by the IDE were not found. To fix this error the line `#include <limits>` needs to be added to the top of these two files: `src/sim/chistogramstrategy.cc` and `src/sim/simtime.cc`

5.2 Free5GC Virtual Machine installation

A second virtual machine(VM2) was created to install Free5GC, meeting minimal hardware requirements. The following sections detail the procedure followed.

Free5GC stage 3 version was deployed on the VM 2, where each core network function was containerized and interconnected to emulate the core network. This setup used 160 GB of storage, 4 GB of RAM, and 2 vCPUs, while running the Ubuntu x86 18.04 LTS operational system with kernel version 5.0.0-23-generic.

5.2.1 Free5GC prerequisites

1. It is necessary to update the Linux kernel to 5.0.0-23-generic.

   a) Check kernel version and update repositories:

```
$ uname -r
$ sudo apt update && sudo apt upgrade
```

b) Search and install kernel version:

```
$ apt search linux-headers-5.0.0.23
$ ls -l /usr/linux-headers-5.0.0.23
$ sudo apt install linux-image-5.0.0-23-generic
$ sudo apt install linux-headers-5.0.0-23-generic
```

c) Update grub and initramfs:

```
$ sudo update-initramfs -u -k all
$ sudo update-grub
$ reboot
```

2. Free5GC uses a customized 5G GTP-U kernel module tested with the kernel version installed before. This module handles PFCP packets.

```
$ git clone https://github.com/free5gc/gtp5g.git
$ cd gtp5g
$ make
$ sudo make install
```

3. Free5GC was developed using Go 1.14.4. First, it is necessary to check the Golang version:

```
$ go version
```

4. If the Golang version mismatches the version supported by free5GC, the necessary steps to follow are:

```
$ sudo rm -rf /usr/local/go
$ wget https://dl.google.com/go/go1.14.4.linux-amd64.tar.gz
$ sudo tar -C /usr/local -zxvf go1.14.4.linux-amd64.tar.gz
```

Otherwise, if Golang is not present in the system:

```
$ wget https://dl.google.com/go/go1.14.4.linux-amd64.tar.gz
$ sudo tar -C /usr/local -zxvf go1.14.4.linux-amd64.tar.gz
```

In both cases, update `.bashrc` to include Go in the PATH:

```
$ mkdir -p ~/go/{bin,pkg,src}
$ echo 'export GOPATH=$HOME/go' >> ~/.bashrc
$ echo 'export GOROOT=/usr/local/go' >> ~/.bashrc
$ echo 'export PATH=$PATH:$GOPATH/bin:$GOROOT/bin' >> ~/.bashrc
$ source ~/.bashrc
```

5. Subsequently, install the packages needed to run the control and user plane elements:

```
$ sudo apt -y update
$ sudo apt -y install mongodb wget git
$ sudo systemctl start mongodb
```

```
$ sudo apt -y update
$ sudo apt -y install git gcc cmake autoconf libtool pkg-config \
libmnl-dev libyaml-dev
$ go get -u github.com/sirupsen/logrus
```

6. To install the control plane, it is necessary the free5GC code repository and install the Go dependencies:

```
$ cd ~
$ git clone --recursive -b v3.0.4 -j \
     `nproc` https://github.com/free5gc/free5gc.git
$ cd free5gc
$ go mod download
```

7. To compile the network functions services:

```
$ cd ~/free5gc
$ make all
```

8. Install the UPF using the following commands:

```
$ cd ~/free5gc
$ make upf
```

9. Install WebConsole to register UEs:

   a) Install Node.js and packages with Yarn:

```
$ sudo apt remove cmdtest
$ sudo apt remove yarn
$ curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | \
sudo apt-key add -
$ echo ”deb https://dl.yarnpkg.com/debian/ stable main” \
| sudo tee /etc/apt/sources.list.d/yarn.list
$ curl -sL https://deb.nodesource.com/setup_12.x | \
sudo -E bash -
$ sudo apt-get update
$ sudo apt-get install -y nodejs yarn
```

   b) Build:

```
$ cd ~/free5gc
$ make webconsole
```

   c) Run tests related to procedures such as Registration, ServiceRequest, Handover, Deregistration, PDUSessionReleaseRequest, N2Handover, and Non3GPP. If the tests succeed, no errors should appear in the log.

```
$ chmod +x ./test.sh
$ ./test.sh <procedure_name>
```

5.3 Network Setup

This section addresses all the necessary steps to configure the network level of the system.

5.3.1 Virtual Machines Connection

In order to connect the two VMss, an isolated network between them was created.

1. Create a new vSwitch and give it the appropriate name without attaching any uplink adapters.
2. Add to each VM of the virtual network adapter the new vSwitch via *Edit Settings -> Add -> Network Adapter* and select from the presented list.
3. Make sure both virtual machines have two vNICs (one for the internet connection and one for the private connection)
4. Assign a static IP address on the virtual machines to the new network adapter.
   a) Assign static IP with netplan:

```
$ cd /etc/netplan
$ ls
01-network-manager-all.yaml
$ cat 01-network-manager-all.yaml
```

   b) Edit the Netplan configuration file in the following way:

```
network:
  version: 2
  renderer: NetworkManager
  ethernets:
    ens192:
      dhcp4: no
      addresses: [192.168.254.1/24]

```

Listing 5.1: 01-network-manager-all.yaml Simu5G VM.

   c) Check for errors in the configuration:

```
$ sudo netplan try
```

   d) Apply the new changes:

```
$ sudo netplan apply
```

e) Verify that the interface has the given IP:

```
$ ifconfig ens192 | grep inet
```

5.3.2 Simu5G Network

Enabling IP forwarding is required to Simu5G VM reroute packets from the veth to VM2.

1. Check if IPv4 forwarding is enabled:

```
$ sysctl net.ipv4.ip_forward
```

2. If forward equals 0 (false), enable it:

```
$ sysctl -w net.ipv4.ip_forward=1
```

Using NAT to connect Simu5G packets to TUN interface *uesimtun0*:

```
$ iptables -A FORWARD -o uesimtun0 -i veth1 -j ACCEPT
```

And setting up the NAT to translate IP on packets from *veth-netns0* to the IP of *uesimtun0* as they are about to go out (POSTROUTING):

```
iptables -t nat -A POSTROUTING -s 10.0.3.2/24 -o uesimtun0 -j MASQUERADE
```

Check out the routing table within Simu5G router and add a gateway to forward packets to the *veth* as presented in listing 5.2.

```
1  ifconfig:
2
3  # interface to the external server
4  name: eth0
5      inet_addr: 192.168.2.1
6      Mask: 255.255.255.0
7      MTU: 1500
8      Metric: 1
9      POINTTOPOINT MULTICAST
10
11 # interface to the gnb
```

```
name: ppp0
    inet_addr: 10.0.2.1
    Mask: 255.255.255.0
    MTU: 1500
    POINTTOPOINT MULTICAST
ifconfigend.

route:

#Destination     Gateway          Genmask          Flags  Metric  Iface
192.168.2.0      *                255.255.255.0    H      0       eth0
192.168.3.0      *                255.255.255.0    H      0       ppp0
0.0.0.0          192.168.2.2      0.0.0.0          G      0       eth0

routeend.
```

Listing 5.2: Router mrt file Simu5G emulation.

With the software configured and deployed, it is necessary to configure Simu5G as emulator and to make packets traverse Simu5G to reach VM2.

Figure 11 shows a detailed view of the emulation setup, where VM1 includes the UEs and RAN components emulated by running the Simu5G in emulator mode. VM2 contains the core network components, such as 5GC and OAI, which are connected to the UE and RAN components through the (*uesimtun0*) tunnel. These VMs are deployed in a virtualization server, using VMware ESXi as Bare Metal Hypervisor, and are connected with Bridged Networking mode, which allows seeing Simu5G VM and 5GC VMs as nodes in the network as if they were physically connected. VMware ESXi enables this abstraction.

To enable packet traversal through Simu5G, a veth (virtual Ethernet) interface was configured, allowing Simu5G to capture packets. UE and Router will receive packets destined to veths by INET's *ExtLowerEthernetInterface* modules. These modules can receive real packets through network interface cards coupled to the modules, with the interfaces being created using veth. The router sends packets to the veth connecting the Simu5G router to the *useimtun0* interface. The tunnel then forwards the packets to the core VM and vice-versa [51, 52].

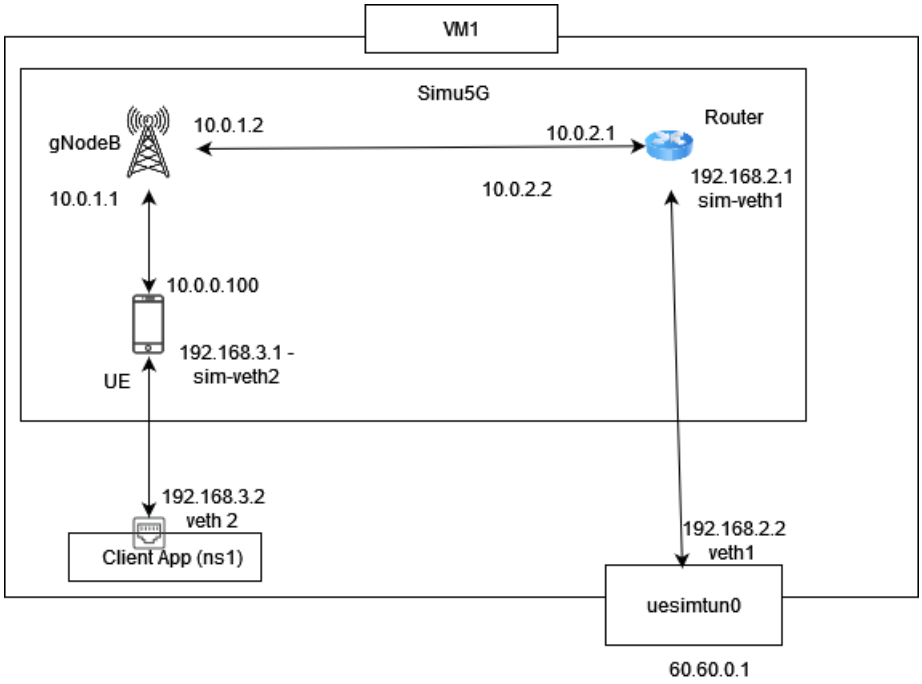


Figure 11: VM1 architecture

To set up the virtual devices and the necessary routing, the following script was used:

```
sudo ip netns add ns1

sudo ip link add sim-veth1 type veth peer name veth1
sudo ip link add veth2 netns ns1 type veth peer name sim-veth2

sudo ip addr add 192.168.2.2/24 dev veth1
sudo ip netns exec ns1 ip addr add 192.168.3.2/24 dev veth2

sudo ip netns exec ns1 ip link set veth2 up
sudo ip link set sim-veth1 up
sudo ip link set sim-veth2 up
sudo ip link set veth1 up

sudo route add -net 192.168.3.0 netmask 255.255.255.0 dev veth1
sudo ip netns exec ns1 route add default dev veth2

sudo ethtool --offload veth1 rx off tx off
sudo ip netns exec ns1 ethtool --offload veth1 rx off tx off
```

Enabling the emulation feature requires accessing the IDE by right-clicking on the INET folder, selecting Properties, then choosing Project Features, and finally selecting the radio button relative to Network Emulation Support.

5.4 Simu5G MEC framework

As shown in Figure 12, the MEC system involves many components. Thus, Simu5G provides models for these components. The core element of this technology is the MEC orchestrator, with a view of the system as a whole. It will manage the requests for instantiation of MEC Apps and determining which MEC host is the most appropriate for the app based on a policy defined by the user according to the following parameters: CPU, memory, disk, and MEC services. This policy can be changed by overriding the findbestmechost method of the MEC Orchestrator module. Moreover, it has a list of MEC hosts, and it will request the MEC platform which is responsible for the lifecycle of the MEC apps, such as instantiation, relocation, and termination. The UALCMP which is an OMNeT++ component that acts as a proxy and receives requests from a Device application usually embedded in UE. These received requests are related to lifecycle procedures, such as the instantiation and termination of the MEC app through an API [53, 54].

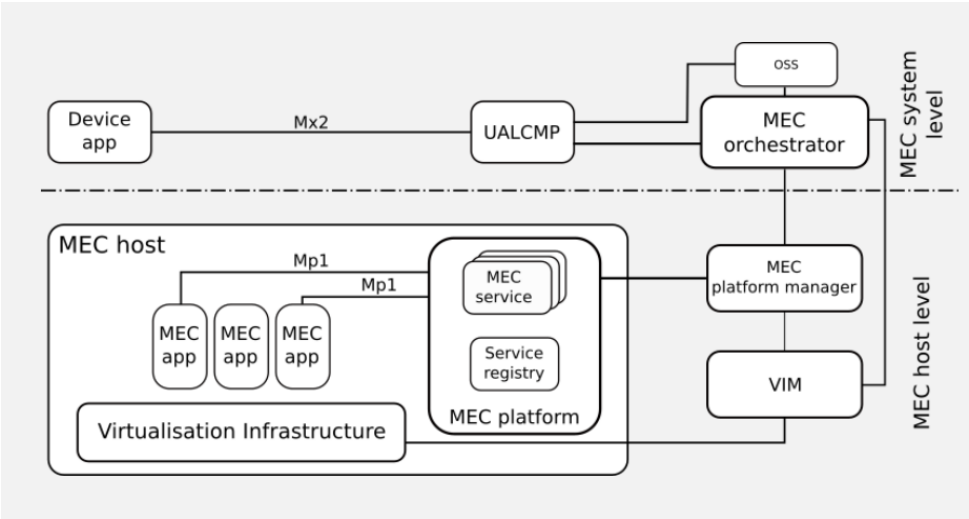


Figure 12: MEC architecture (Extracted from [53])

In Simu5G simulation, the UALCMP model provides the API to be consumed by the Device App that will initiate the creation and termination of MEC Apps, if the UALCMP is connected to an INET external interface, it will be possible to have a Device App running outside the simulator. The MEC orchestrator is a simple module connected to the UALCMP through an OMNeT++ module that manages the MEC hosts. Those can be configured through the mecHostList parameter [53].

After a MEC App creation request is sent from the UALCMP connected to a Device app, the MEC orchestrator selects the most suitable MEC host from those related with the MEC system based on a policy defined by the user according to requirements. The requirements of the MEC apps are specified in a JSON file called App Descriptor, as exemplified in Listing 5.3. The required MEC services and computing resources, such as RAM, CPU and disk are specified in this file.

```
{
"appDid" : "MECAPP",
"appName" : "AppName",
"appProvider" : "simu5g.apps.mec.appName.AppName",
"appInfoName" : "appInfoName_",
"appDescription" : "appDescription_",
"virtualComputeDescriptor" :{
    "virtualDisk": 60,
    "virtualCpu" : 2000,
    "virtualMemory":60
    },
"appServiceRequired": [
    {
        "ServiceDependency" :{
            "serName" : "LocationService",
            "version" : "v2",
            "serCategory": "Location"
        }
    }
],
}
```

Listing 5.3: Application Descriptor example.

If the MEC App is simulated, the appProvider field is used to supply the module name required to create the module that implements the MEC App. Otherwise, if it is emulated it is necessary to add the ip and the port where the application is hosted, as show in Listing 5.4.

```
{
"emulatedMecApplication" :{
    "ipAddress": "192.168.1.2",
    "port": 2015
```

```
    }
}
```

Listing 5.4: Application Descriptor with emulated MEC App.

All these requests are not standard compliant because they use OMNeT++ messages through module communications. This still allows having a standard behavior without losing functionality

When the Orchestrator selects the MEC Host responsible to allocate the MEC app, it will be instantiated by creating an OMNeT++ model running the application logic. The Device app, embedded in UE, manages the instantiation/termination of MEC apps. The Device App communicates through an API with the UALCMP and establishes a UDP socket connection shared with the UE app. The communication over the UDP socket is done through messages with code that can determine the start, termination, and acknowledgements of the previous requests. The basic structure of these messages is composed of code related to the message type, the length of the subsequent package size, and the data. The following codes are implemented in Simu5G:

0. Instantiates the creation of an MEC App, by receiving its name as a payload.
1. Termination request of an MEC App, and with the App name as payload;
2. Acknowledgment related to the Start(responds with the Ip address and Port of the instantiated MEC App);
3. Acknowledgment related to the Termination.
4. Code relative to negative acknowledgement related to the Start of an MEC App.
5. Code relative to negative acknowledgement related to the termination of an MEC App.

The configuration of components, such as VIM and Orchestrator are done via NED/INI files, for instance, the maximum resources possible to be allocated.

The Simu5G MEC Platform is integrated with MEC Services such as a Location Service and a Radio Network Service. The latter provides network information related to radio network conditions. This way, MEC Apps can use this information to optimize or provide new services according to these conditions. In addition, the former service offers information about the location of devices, which may be helpful for location-based recommendations and vehicular networks [55, 56].

5.4.1 MEC Apps implementation

To develop an internal application, it is necessary to create the UE App using the Device module provided by the Simu5G framework, which are modeled as depicted in Figures 14 and 13. The Device module then

sends requests for the instantiation or termination of MEC apps to the UALCMP (User Application-Layer Control and Management Plane) via a REST API. Therefore, the logic of a UE App should be initiated by requesting the instantiation or termination of the MEC App from the Device App.

Firstly, it is necessary to define the *initialize* function, which executes this method prior to the start of the simulation and is responsible for configuring the simulation clock, initialize state, and statistical counters.

To initiate the installation or termination of a MEC app, it is necessary to establish a connection with the Device App, which will then request the MEC app from the UALCMP.

The *par* struct reads parameter values from the module present in the ini file, which is used to customize the behavior.

The following code in on the Listing 5.5 demonstrates how to establish this connection:

```
void UEApp::initialize(int stage)
{
    cSimpleModule::initialize(stage);
    // avoid multiple initializations
    if (stage!=inet::INITSTAGE_APPLICATION_LAYER)
        return;

    localPort_ = par("localPort");
    deviceAppPort_ = par("deviceAppPort");

    char* deviceAppAddressStr = (char*)par("deviceAppAddress").stringValue();
    deviceAppAddress_ = inet::L3AddressResolver().resolve(deviceAppAddressStr);

    //binding socket
    socket.setOutputGate(gate("socketOut"));
    socket.bind(localPort_);

    mecAppName = par("mecAppName").stringValue();
```

Listing 5.5: Initialized method with device app.

Next, it is necessary to define the initial events that need to be added to the FES (Future Event Set), as shown in Listing 5.6.

```
    mecAppStart_ = new cMessage("mecAppStart");
    sendRequest_ = new cMessage("sendRequest");
```

Listing 5.6: Defining initial events.

In the INET Framework, there are various methods available to schedule the first event in a simulation. The "send()"method is used to transmit messages between modules, while the *scheduleAt* method is employed to set up events that occur at a specific time, including self-messaging events. Furthermore, *cancelEvent* method allows for the removal of events that have been scheduled using *scheduleAt*.

By utilizing these methods, it is possible to efficiently schedule events and control the flow of the simulation.

Within this method, it is essential to schedule the first event that will initiate the execution of the *handleMessage* method. As exemplified in listing 5.7, this event should be a MEC Start Request to the Device App, which will retrieve the port and address to communicate with the MEC App.

```
1     simtime_t startTime = par("startTime");
2     scheduleAt(simTime() + startTime, mecAppStart);
```

Listing 5.7: Schedule first event.

Finally, the statistical counters are initialized, and an example of such a counter is e the response time, which will be recorded. To initialize a statistic, it is necessary to register it using the *registerSignal* method and save it as a signal variable to enable its recall later, as depicted in 5.8. To calculate the response time requires timestamps for when a packet is sent and received, and the process for obtaining the timestamps will be further explained.

```
1     simsignal_t responseTime_ = registerSignal("responseTime");
2 }
```

Listing 5.8: Register signal.

The *handleMessage* method receives and dispatches the events to the appropriate processing function. It is not recommended to store state variables in local variables within this method because they will be destroyed once the method finishes executing.

As a UE App, it will be necessary to handle events from other modules, including the Device App and the MEC App, as well as process its own events.

The method can be split into three blocks, with the first one including a condition that determines how it handles messages addressed to itself, as illustrated in listing 5.9.

```
1 void UeApp::handleMessage(cMessage *msg)
```

```
    if (msg->isSelfMessage())
    {
        if(!strcmp(msg->getName(), "mecAppStart"))
            processStart();
        else if(!strcmp(msg->getName(), "event2"))
            processEvent2();
        else
            throw cRuntimeError("Error");
    }
```

Listing 5.9: handleMessage first block

Then the second block, as shown in listing 5.10 is responsible for handling the upcoming events received from the Device App, which will consist of the acknowledgment messages indicating the initialization/termination status of MEC Apps.

```
else
    {
        inet::Packet* packet = check_and_cast<inet::Packet*>(msg);
        inet::L3Address ipAdd = packet->getTag<L3AddressInd>()->getSrcAddress();

        if(ipAdd == deviceAppAddress_ || ipAdd == inet::L3Address("127.0.0.1"))
        {
            auto mePkt = packet->peekAtFront<DeviceAppPacket>();

            if (mePkt == 0)
                throw cRuntimeError("UERequestApp::handleMessage - \tFATAL! Error when
    ↪  casting to DeviceAppPacket");
            if( !strcmp(mePkt->getType(), ACK_START_MECAPP) )
                processAckStart(msg);
            else if(!strcmp(mePkt->getType(), ACK_STOP_MECAPP))
                processAckStop(msg);
            else
                throw cRuntimeError("UERequestApp::handleMessage - \tFATAL! Error,
    ↪ DeviceAppPacket type %s not recognized", mePkt->getType());
        }
```

Listing 5.10: handleMessage second block

Finally, the listing 5.11 shows how the third is responsible for handling the messages from the MEC App.

```
    else
        {
            auto mePkt = packet->peekAtFront<RequestResponseAppPacket>();
            if (mePkt == 0)
                throw cRuntimeError("UERequestApp::handleMessage - \tFATAL! Error when
    ↪  casting to RequestAppPacket");

            if(mePkt->getType() == Event3)
                handleEvent3(msg);
            else if(mePkt->getType() == Event4)
                handleEvent4(msg);
            else
                throw cRuntimeError("UERequestApp::handleMessage - \tFATAL! Error,
    ↪ RequestAppPacket type %d not recognized", mePkt->getType());
        }
    }
}
```

Listing 5.11: handleMessage third block

After implementing the previous methods, the next step is to create functions that can process the events.

After the first scheduled event, *mecAppStart*, is triggered, it is directed to the module as shown in listing 5.9. The handler then will call *processStart* method to send the *Start* request to the DeviceApp. Upon receiving the Start ACK from the DeviceApp, it will be necessary defining the appropriate function to process it. For instance, a process function can be used to create a packet to send to the UE and modify the state of the start_ variable accordingly.

In most use cases related to networks, packets are used as the primary data structure. To demonstrate how to use the packet API and implement a method that processes an event, let us consider as an example the code in Listing 5.13 with the previous scenario, where a packet is sent after receiving the ACK message as an approach for dealing with the message.

A packet is composed of different data types called chunks. These chunks can represent bits, slices, and protocols, and users can define their custom chunks in a message file as displayed in listing 5.12.

To construct the packet chunks, it is recommended to use the *makeShared* method provided by the INET framework, instead of the new operator. This is due to chunks being shared between packets with

shared pointers. Therefore, when creating a packet with its data, such as timestamp, length, and request type, the *makeShared* method will allocate and manage memory deallocation when the packet pointer goes out of scope.

Listing 5.12 demonstrated that the packet will include a *sentTime* field that will be used to calculate the response time, and the type of the request that identifies the event.

```
import nodes.mec.MECPlatform.MECPackets;
import inet.networklayer.common.L3Address;
import inet.common.INETDefs;
import inet.common.packet.chunk.Chunk;

class MyPacket extends inet::FieldsChunk
{
    int type;
    simtime_t sentTime;
}
```

Listing 5.12: Costum Packet definition in mypacket.msg file

```
void UEApp::processAckStart(cMessage* msg)
{
    inet::Packet* packet = check_and_cast<inet::Packet*>(msg);
    auto pkt = packet->peekAtFront<DeviceAppStartAckPacket>();

    if(pkt->getResult() == true)
    {
            // Get UEAPP Adress and Port.
            mecAppAddress_ = L3AddressResolver().resolve(pkt->getIpAddress());
            mecAppPort_ = pkt->getPort();
            // Remove first event.
            cancelEvent(selfStart_);

            inet::Packet* pkt = new inet::Packet("MyPacket");
            auto req = inet::makeShared<MyPacket>();

            // Set packet attributes
            req->setType(UEAPP_REQUEST);
            req->setSentTime(simTime());
```

```
20              // Bit size packet in inherited field
21              req->setChunkLength(inet::B(requestPacketSize_));
22              // Append chunck to packet
23              pkt->insertAtBack(req);
24              start_ = simTime();
25              socket.sendTo(pkt, mecAppAddress_ , mecAppPort_);
26      }
27  }
```

Listing 5.13: Process Ack Start.

Once this module receives the response, another processing function must be defined. In this case, the available information enables the response time calculation, as the packet was sent and the response subsequently received. To do this, the *sentTime* field is retrieved, and the difference between the current simulated time and the *sentTime* is calculated. After this, the signal is emitted to record the time statistic. Listing 5.14 shows the code with the previous logic.

```
1  void UEApp::handleEvent3(cMessage* msg)
2  {
3      inet::Packet* packet = check_and_cast<inet::Packet*>(msg);
4      auto res = packet->peekAtFront<RequestResponseAppPacket>();
5      res->getSentTime();
6      simtime_t respTime = simTime()- res->getRequestSentTimestamp();
7      emit(responseTime_, respTime);
8      delete packet;
9  }
```

Listing 5.14: Handle Event.

Developing a MEC App follows a similar logic to that of a UE application. However, instead of building an application from scratch, the *MecAppBase* module is utilized, which provides several pre-defined functions that need to be overridden. These functions include the handling of messages, scheduling of messages, management of socket connections, establishing contact with the Mp1 interface, and using the MEC services.

By utilizing the *MecAppBase* module, developers do not need to worry about the underlying details of the MEC host's processing power or the scheduling of messages and socket connections. Instead, they can focus on implementing the core functionality of the application. The module handles the handling of messages, while the developer needs to implement the *handleProceedMessage*, *handleSelfMessage*, and *handleHttp* methods.

This approach provides an abstraction layer that simplifies the development process and allows developers to build MEC applications more efficiently. By leveraging the *MecAppBase* module, developers can focus on implementing the key features of the application, while the module handles the low-level details of communication and scheduling.

Therefore, as illustrated in Listing 5.15, the initialize method follows the same logic as before, but there is no need to schedule a first event since it will wait for a request from the UE App. Instead, it is necessary to bind the socket to receive and send requests to the UE App and connect to the MP1 interface through the inherited fields from the parent class.

```
void MECApp::initialize(int stage)
{
    MecAppBase::initialize(stage);

    // avoid multiple initializations
    if (stage != inet::INITSTAGE_APPLICATION_LAYER)
        return;

    localUePort_ = par("localUePort");
    ueAppSocket_.setOutputGate(gate("socketOut"));
    ueAppSocket_.bind(localUePort_);

    mp1Socket_ = addNewSocket();
    connect(mp1Socket_, mp1Address, mp1Port);
}
```

Listing 5.15: InitializeMecApp.

Instead of coding the entire logic of the *handleMessage* method, it is necessary to write separate methods that compose the handle message. These methods include *handleProcessedMessage*, which handles events from the UE App, and *handleHttpMessage*, which is typically composed of *handleMp1Message* and *handleServiceMessage*. The former is responsible for discovering MEC Services, while the latter parses and manages the responses or notifications from the MEC Services. The *handleProcessedMessage* function, in the context of the previous example, would be responsible for dealing with the *UEAPP_REQUEST* event, as depicted in Listing 5.16.

```
void MECApp::handleProcessedMessage(cMessage *msg)
{
if(ueSocket.belongsToSocket(msg)) {
```

```
    inet::Packet* packet = check_and_cast<inet::Packet*>(msg);
    auto req = packet->peekAtFront<RequestResponseAppPacket>();
    if(req->getType() == UEAPP_REQUEST)
        handleEvent(msg);

    }
    MecAppBase::handleProcessedMessage(msg);
```

Listing 5.16: handleProcessedMessage.

As displayed in Listing 5.17, the *handleEvent* method logic is similar to the UEAPP_REQUEST event, as a packet is sent. However, the *sentTime* field is updated to calculate the correct request time.

```
void MECApp::handleEvent(msg)
{
    inet::Packet* packet = check_and_cast<inet::Packet*>(msg);
    ueAppAddress = packet->getTag<L3AddressInd>()->getSrcAddress();
    ueAppPort  = packet->getTag<L4PortInd>()->getSrcPort();

    auto req = packet->removeAtFront<MyPacket>();
    req->setType(MECAPP_RESPONSE);
    req->setSentTime(simTime());
    req->setChunkLength(B(packetSize_));
    inet::Packet* pkt = new inet::Packet("MyPacket");
    pkt->insertAtBack(req);

    ueAppSocket_.sendTo(pkt, ueAppAddress, ueAppPort);

}
```

Listing 5.17: handleEvent in MECApp.

The following diagrams in Figure 13 and Figure 14 illustrate the concepts explained in this section and demonstrate how the different methods interact with each other with Simu5G and OMNeT++ event loop.

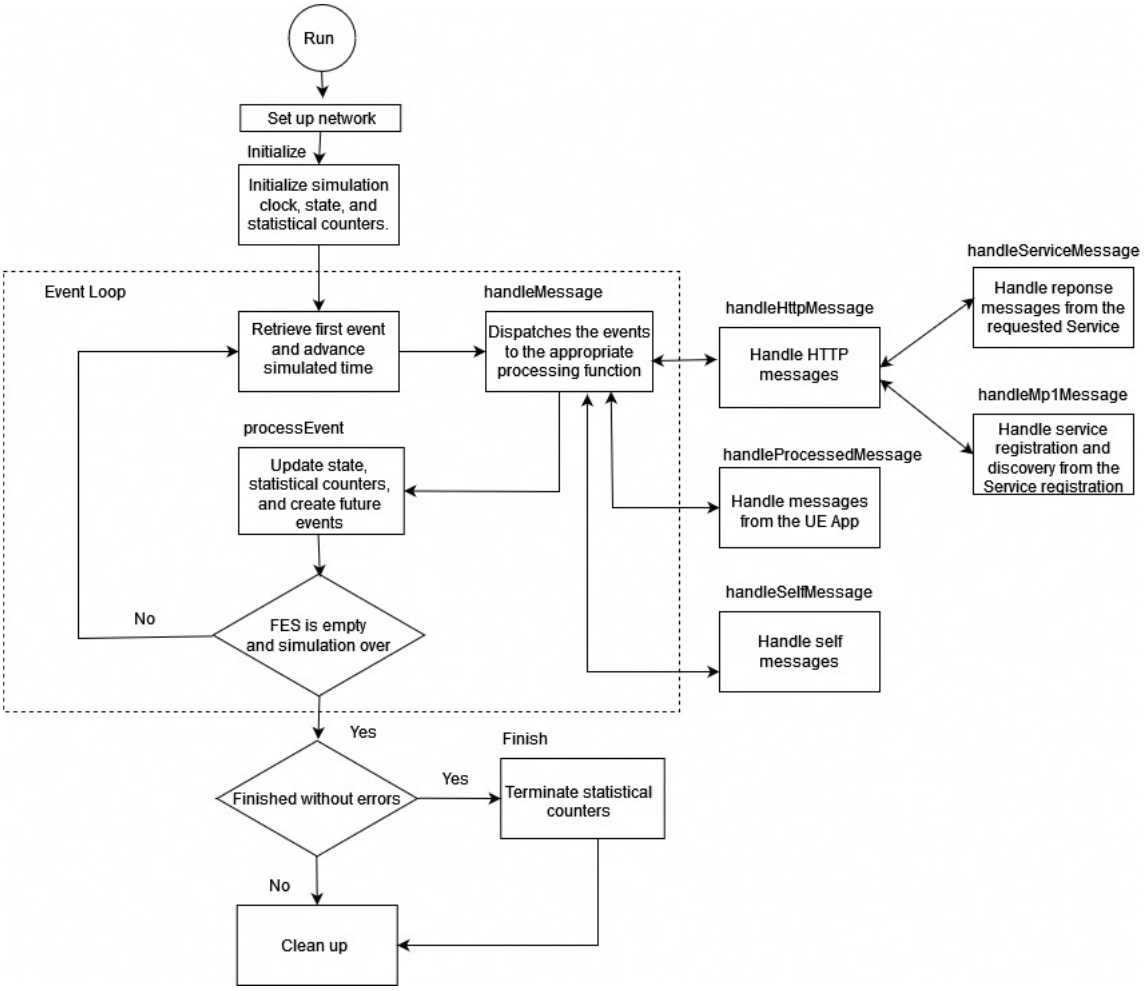


Figure 13: MEC App model.

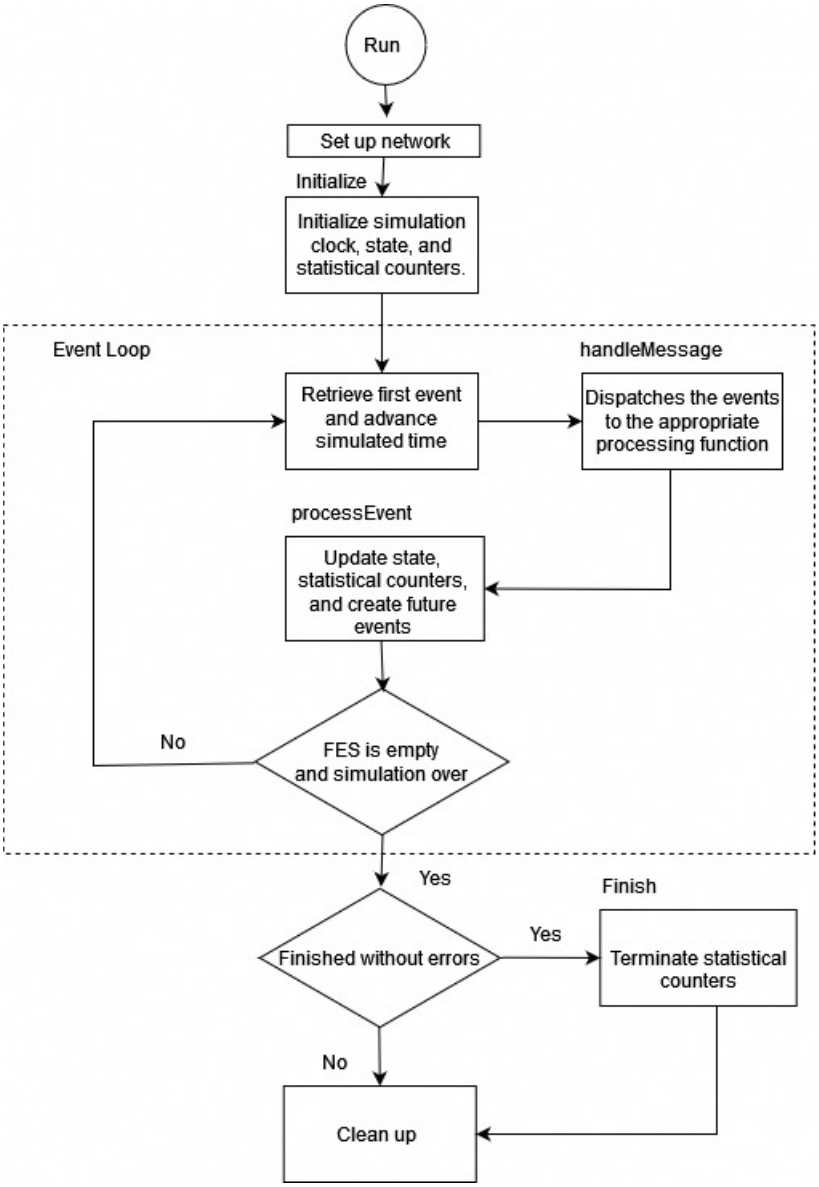


Figure 14: UE App model.

Chapter 6

Results

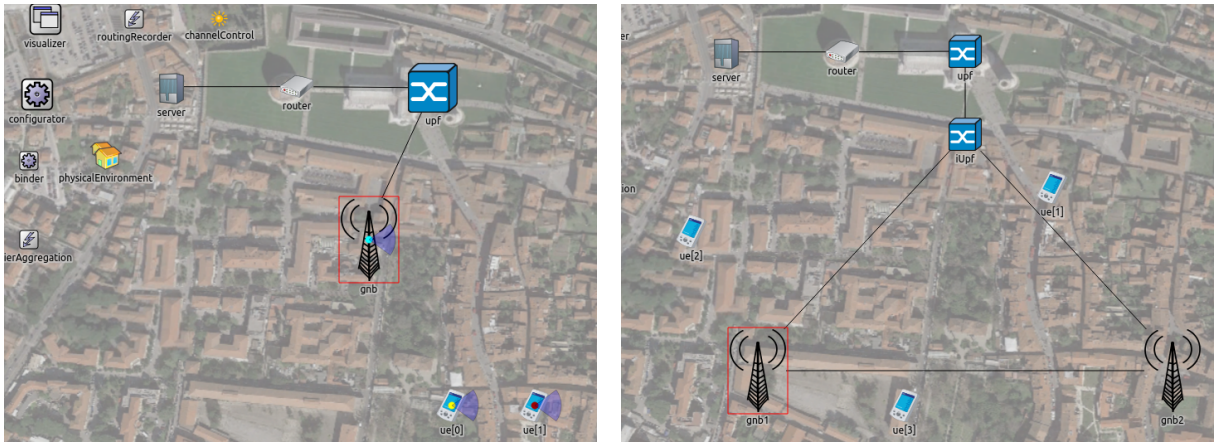
This chapter presents a set of results obtained using the virtualized 5G network deployed based on the proposed architecture and components. Firstly, a set of tests were conducted to test end-to-end connectivity between a UE and a host in an external network. Connectivity is observed by capturing packets in several points of the network, while packets move from UE through RAN to UPF and the data network. Finally, a test planned to show a MEC use case is presented.

6.1 Simulation test

The first scenario is implemented as depicted in Figure 15. However, Simu5G does not support capturing packets at layer 2 and its applications, the packets were captured at layer3, simulated through INET application and not Simu5G application as shown in Figure 16. The behavior was similar between scenarios, as the applications transmitted packets at a constant rate. However, with introduced mobility, packet loss occurred.

Only the UPF component from the core is deployed in this setup because Simu5G only supports the user plane. In an emulation scenario where the core is provided by another software, components related to the control plane would also be presented.

In these scenarios, the user devices start by sending a video streaming request to the server and then the server starts streaming and sending UDP packets at a constant rate to the clients. The mobility shown in Figure 15b is linear, which has constant speed or acceleration or bonnMotion mobility, which can represent a mobility dataset.



(a) Simu5G simulation scenario 1(a).

(b) Simu5G simulation scenario 1(b).

Figure 15: Simu5G simulation scenarios.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.0.1	10.0.3.2	UDP	1028	3088 → 9999 Len=1000
2	0.007000	10.0.0.1	10.0.3.2	UDP	1028	3088 → 9999 Len=1000
3	0.008000	10.0.0.1	10.0.3.2	UDP	1028	3088 → 9999 Len=1000
4	0.008000	10.0.0.1	10.0.3.2	UDP	1028	3088 → 9999 Len=1000
5	0.008000	10.0.0.1	10.0.3.2	UDP	1028	3088 → 9999 Len=1000
6	0.008000	10.0.0.1	10.0.3.2	UDP	1028	3088 → 9999 Len=1000
7	0.012000	10.0.0.1	10.0.3.2	UDP	1028	3088 → 9999 Len=1000
8	0.012000	10.0.0.1	10.0.3.2	UDP	1028	3088 → 9999 Len=1000
9	0.012000	10.0.0.1	10.0.3.2	UDP	1028	3088 → 9999 Len=1000
10	0.012000	10.0.0.1	10.0.3.2	UDP	1028	3088 → 9999 Len=1000
11	0.012000	10.0.0.1	10.0.3.2	UDP	1028	3088 → 9999 Len=1000
12	0.015000	10.0.0.1	10.0.3.2	UDP	1028	3088 → 9999 Len=1000
13	0.015000	10.0.0.1	10.0.3.2	UDP	1028	3088 → 9999 Len=1000
14	0.015000	10.0.0.1	10.0.3.2	UDP	1028	3088 → 9999 Len=1000

Figure 16: Wireshark packets captured from scenario 1.

6.2 End-to-end system test

The first result obtained is a test to verify if the end-to-end network connectivity is successful. In other terms, if a packet can traverse the implemented emulated network, reaching all its components. This test guarantees that all components of the emulated network are accessible and that the routing ensure that the packets arrive at those same components.

To perform this test the *ping* command was employed, which is a command used to validate connectivity and reachability in a network. Since we can specify an IP address, it will offer information such as the duration of the data transmission and if the host can get a response from the specific address. The other tool used is *Wireshark* to ensure that the packets reach the desired interfaces.

The first step is to run the *ping* command in the application namespace as in Figure 17 it ended to isolate applications that have as objective cross the simulation network. The namespace is isolated, and it owns separate network interfaces and routing tables. Therefore, all the application traffic must follow the rules presented in Figure 18 for the application namespace, not the host.

```
simu5g@simu5g:~$ sudo ip netns exec ns2 ping 8.8.8.8
```

Figure 17: Ping command.

```
simu5g@simu5g:~$ sudo ip netns exec ns2 ip route
default dev veth2 scope link
192.168.3.0/24 dev veth2 proto kernel scope link src 192.168.3.2
```

Figure 18: IP routes in application namespace.

These rules guarantee that the ingress traffic goes to the emulated environment via *veth2* as we can see in Figure 19.

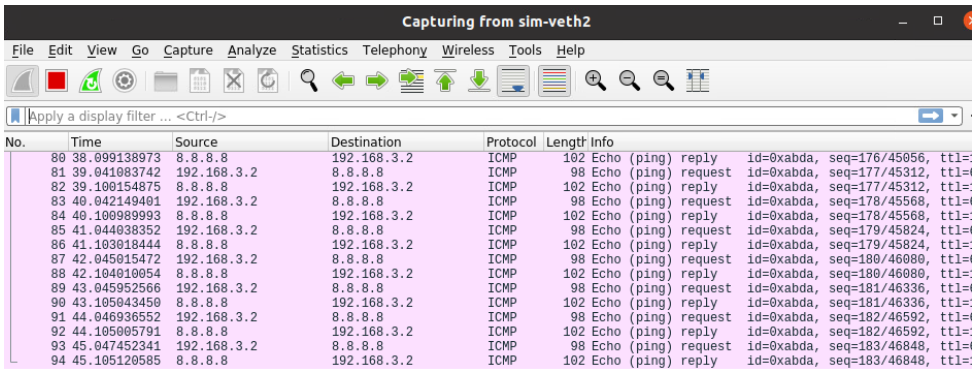


Figure 19: Sim-veth2 traffic.

Once in the simulation, the traffic will be governed by the rules configured in the *.mrt* files, as shown in Listing 5.2, which configures the routing tables of the simulated elements in Simu5G. If the emulation stops, traffic will no longer flow through *veth1* between the router and the tunnel, and a *ping* command, for instance, to 10.0.2.1(router) will not reach the destination. To confirm that traffic is reaching the simulation a *ping* command to the simulated router is executed (Figure 21).

```
simu5g@simu5g:~/omnetpp-6.0/sampless/simu5g/emulation/extclientserver$ ./run.sh
OMNeT++ Discrete Event Simulation  (C) 1992-2022 Andras Varga, OpenSim Ltd.
Version: 6.0, build: 220413-71d8fab425, edition: Academic Public License -- NOT FOR COMMERCIAL USE
See the license for distribution terms and warranty disclaimer

Setting up Cmdenv...

Loading NED files from /home/simu5g/omnetpp-6.0/sampless/simu5g/simulations:  18
Loading NED files from /home/simu5g/omnetpp-6.0/sampless/simu5g/emulation:  7
Loading NED files from /home/simu5g/omnetpp-6.0/sampless/simu5g/src:  85
Loading NED files from /home/simu5g/omnetpp-6.0/sampless/inet4.4/src:  1166

Preparing for running configuration ExtClientServer, run #0...
Assigned runID=ExtClientServer-0-20221027-15:45:43-634165
Setting up network "simu5g.emulation.extclientserver.ExtClientServerExample"...
Initializing...

Running simulation...
** Event #0   t=0   Elapsed: 2.5e-05s (0m 00s)  0% completed  (0% total)
     Speed:     ev/sec=0   simsec/sec=0   ev/simsec=0
     Messages:  created: 70   present: 70   in FES: 10
** Event #8704   t=2.081   Elapsed: 2.02754s (0m 02s)  0% completed  (0% total)
     Speed:     ev/sec=4293.42   simsec/sec=1.02638   ev/simsec=4183.09
     Messages:  created: 2515   present: 67   in FES: 6
```

Figure 20: Emulation running.

```
simu5g@simu5g:~$ sudo ip netns exec ns2 ping 10.0.2.1
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.
64 bytes from 10.0.2.1: icmp_seq=1 ttl=31 time=18.9 ms
64 bytes from 10.0.2.1: icmp_seq=2 ttl=31 time=14.1 ms
64 bytes from 10.0.2.1: icmp_seq=3 ttl=31 time=13.6 ms
64 bytes from 10.0.2.1: icmp_seq=4 ttl=31 time=13.7 ms
```

Figure 21: Ping to the simulated router.

After that, the connection between the router and tunnel is made through the *veth* pair *veth1 sim-veth1*. And it is possible to see the traffic reaching the pair as shown in Figures 22 and 23 .

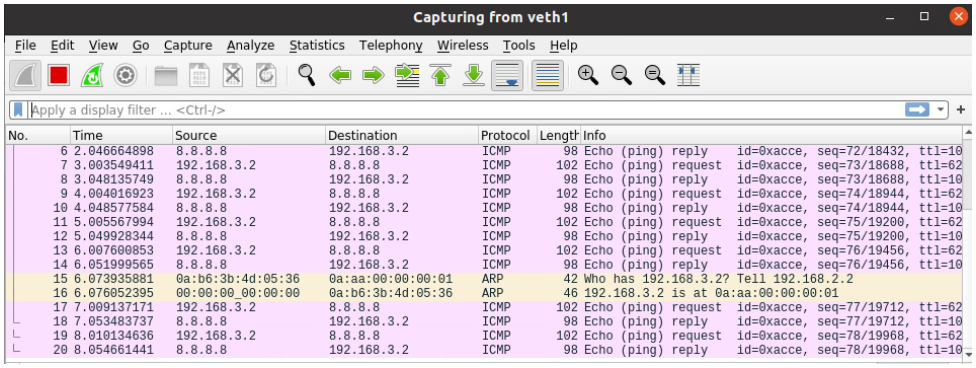


Figure 22: veth1 traffic

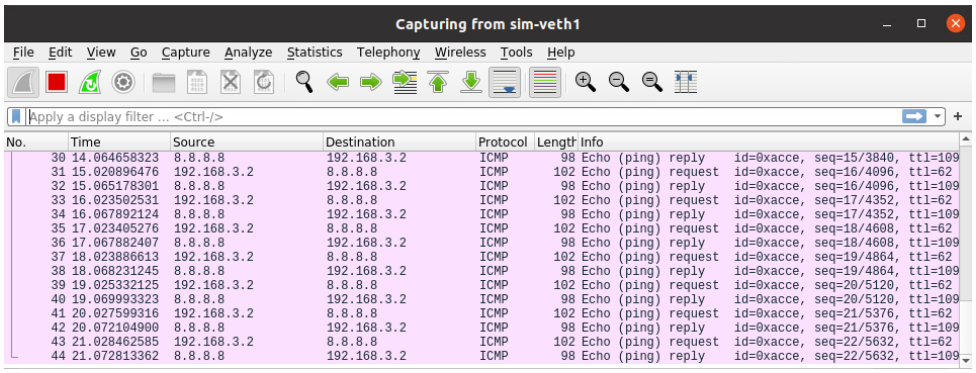


Figure 23: sim-veth1 traffic

Subsequently, the packets are expected to reach the *TUN* interface due to the NAT and packet forwarding rules introduced as shown in Figure 24.

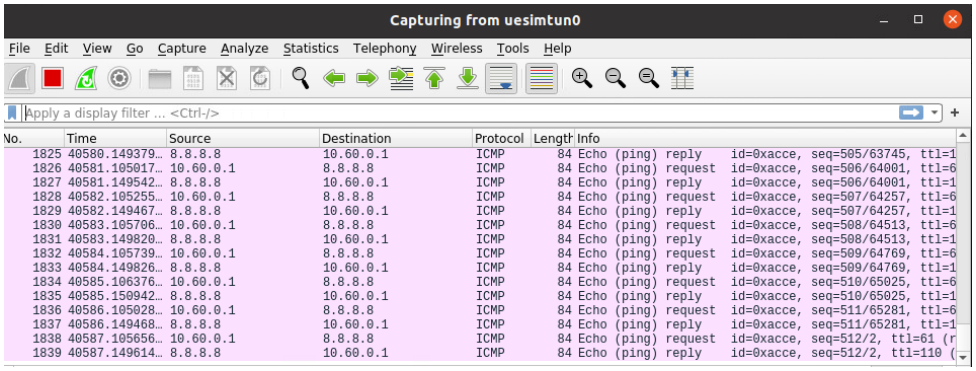


Figure 24: uesimtun0 traffic.

Then, as seen in Figure 25, packets arrive at the 5GC interface.

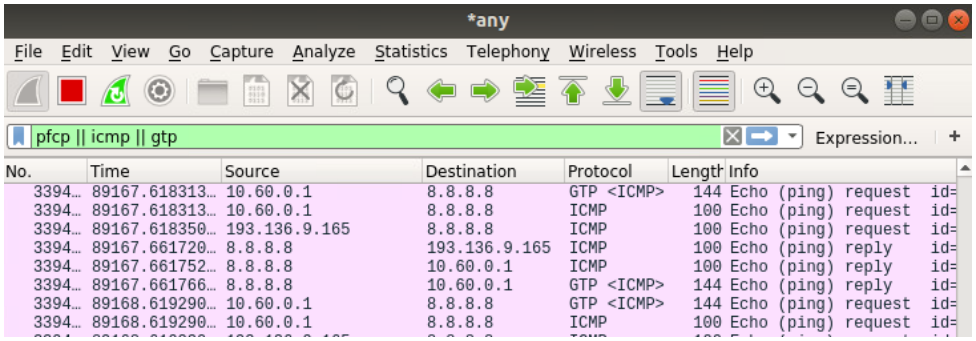


Figure 25: upfgtp traffic.

Traffic reaches all the expected interfaces, and the ping is successful, as illustrated in Figure 26. The *ping* command was configured to send 100 packets with a $1s$ delay between each packet.

```
simu5g@simu5g:~$ sudo ip netns exec ns2 ping google.com -c 100 -i 1
PING google.com (142.250.200.110) 56(84) bytes of data.
64 bytes from mad41s13-in-f14.1e100.net (142.250.200.110): icmp_seq=1 ttl=110 time=60.8 ms
64 bytes from mad41s13-in-f14.1e100.net (142.250.200.110): icmp_seq=2 ttl=110 time=62.0 ms
64 bytes from mad41s13-in-f14.1e100.net (142.250.200.110): icmp_seq=3 ttl=110 time=62.0 ms
64 bytes from mad41s13-in-f14.1e100.net (142.250.200.110): icmp_seq=4 ttl=110 time=62.6 ms
64 bytes from mad41s13-in-f14.1e100.net (142.250.200.110): icmp_seq=5 ttl=110 time=61.2 ms
64 bytes from mad41s13-in-f14.1e100.net (142.250.200.110): icmp_seq=6 ttl=110 time=63.2 ms
64 bytes from mad41s13-in-f14.1e100.net (142.250.200.110): icmp_seq=7 ttl=110 time=60.6 ms
64 bytes from mad41s13-in-f14.1e100.net (142.250.200.110): icmp_seq=8 ttl=110 time=61.6 ms
```

Figure 26: Ping response.

The metrics collected using this test were the average latency of $61.869ms$, and a jitter of $6.752ms$ as shown in Figure 27.

```
64 bytes from mad41s13-in-f14.1e100.net (142.250.200.110): icmp_seq=97 ttl=110 time=61.9 ms
64 bytes from mad41s13-in-f14.1e100.net (142.250.200.110): icmp_seq=98 ttl=110 time=61.4 ms
64 bytes from mad41s13-in-f14.1e100.net (142.250.200.110): icmp_seq=99 ttl=110 time=65.8 ms
64 bytes from mad41s13-in-f14.1e100.net (142.250.200.110): icmp_seq=100 ttl=110 time=59.9 ms

--- google.com ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99098ms
rtt min/avg/max/mdev = 55.235/61.869/127.125/6.752 ms
```

Figure 27: Ping response.

6.3 Simu5G MEC Framework test

This test has, as its primary objective, to test the Simu5G emulation making use of one of its features, an ETSI MEC scenario, as Simu5G is distinguishable from other solutions for integrating this technology. In contrast, the solutions that provide MEC capability are more focused on this technology, being necessary to integrate with other components to have a whole system. Furthermore, Simu5G expects to offer a quick method for testing some features of MEC applications in emulated or simulated deployments.

This test uses the emulated mode to test a MEC scenario. It was created, a MEC App that collects an image from the UE. In this case, it could be an image of traffic in a street to represent a real-world use case, and the MEC app will take advantage of AI, specifically the YOLOv5 deep learning framework built on the PyTorch platform, to identify obstacles in the image and send it as a response to the UE, which in this example is used the It can be seen how this use case will benefit from the MEC technology by reducing the latency by being close to the RAN, which is crucial in a vehicular network. Moreover, the AI models add overhead, taking a long time to process. The service location is another feature that could be used related to car mobility through an API request. Nonetheless, it was not the case in this test.

There were realized three tests the first with The MEC app requiring 10 MB of RAM, 10 MB of storage, 1500 instructions per second of CPU and the second MEC app requiring 2 GB of RAM, 4 GB of storage, 3000 instructions per second of CPU. Additionally, the third test was performed without a MEC solution.

The Figure 29 shows the UE App initiating the MEC App, then receives as ACK the IP Address to connect to the MEC App. Subsequently, it sends the image depicted in Figure 30. The MEC App receives the image as demonstrated in Figure 31 and runs the detector to find obstacles presented in the image received. After that, the detection is sent to the UE App. The UE App receives the images and sends to the device app the request to terminate the MEC App. Finally, The device app sends a termination ACK to confirm. The network is composed by 1 UE, 1 gNodeB, 1 MEC Host, 1 UPF, 1 MEC orchestrator, and 1 MEC App.

To obtain the results, the traffic was captured at the UE component's NIC interface up to the RLC layer, which is responsible for data transmission over the radio link. The device with the App was used for this purpose. The higher the throughput and the lower the delay parameters, the faster the application will perform.

Accordingly, to the results, the transfer of images takes minutes to complete. This is due to packets being sent with $1s$ intervals in order for the emulation can maintain real-time speed. Although this represents a long time it does not cause substantial issues, as the real packets are processed as they were generated inside the simulation distortions, besides the duration of the capture and injection into the simulation [57, 58].

```
simu5g@simu5g:~/5gapps/apps$ python3 server.py
Receiving image file.: 100%|                    | 300k/300k [06:00<00:00, 85
Running detector.
Transfering detection result.
```

Figure 28: MEC App execution.

```
simu5g@simu5g:~/5gapps/apps$ python3 client.py
Start message to device application sent
Message from device application: code 2, data length 16, endpoint 19
2.168.2.2:4022
connect
Client message: cars.jpg
Transfering image to the server.
Receiving detection file: 100%|     | 640k/640k [12:49<00:00, 852B/s]
Sendind delete to terminate the MEC App.
Termination ack received
You have been disconnected from the MECApp
```

Figure 29: UE App execution.

Figure 30: Traffic image sent to MEC App.

From the results presented in Table 1, Table 2 and Table 3, it can be seen that the increased using a MEC solution improves the performance since the throughput is higher. When specifying the same application with the need for more resources in a network without other MEC Apps competing, for resources there is a higher Throughput and lower Delay since it will have more computation power.

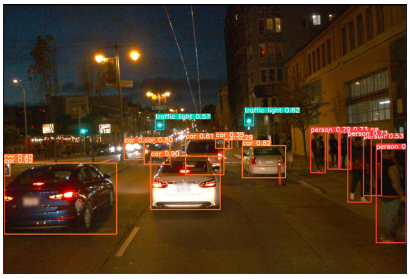


Figure 31: Image resulting from detection sent to UE App.

Table 1: Test results with lower resources requirements.

Parameter	**Value**
ThroughputUL mean	$599.395649Bps$
DealyUL mean	$0.012697s$
ThroughputDL mean	$310.240893Bps$
DelayDL mean	$0.0006115s$

Table 2: Test results with higher resources requirements.

Parameter	**Value**
ThroughputUL mean	$601.988267Bps$
DealyUL mean	$0.012629s$
ThroughputDL mean	$315.911434Bps$
DelayDL mean	$0.0005857s$

Table 3: Test without MEC solution.

Parameter	**Value**
ThroughputUL mean	$462.836514Bps$
DealyUL mean	$0.013407s$
ThroughputDL mean	$284.505164Bps$
DelayDL mean	$0.0007983s$

Chapter 7

Conclusions and future work

In this chapter, conclusions and future work are presented.

7.1 Conclusion

This work aims the deployment of a fully virtualized 5G using open-source software that is publicly available, allowing for research and experimentation without requiring access to an operational cellular network or equipment.

To achieve this goal, initially, different generations of mobile networks were studied, with a focus on the 5G network and its key features. Also, concepts of softwarization and virtualization were considered. Softwarization refers to the implementation of capabilities through software, providing flexibility in deployment and dynamic configuration. Virtualization, on the other hand, is the abstraction of hardware resources and their transfer to software, enabling efficient resource allocation, traffic adaptation, and easier management. Both concepts help to reduce capital and operational expenses in 5G networks. Moreover, the use of NFV and SDN was also considered since they are present in the 5G architecture. Then, different 5G testbeds and their capabilities were discussed in order to understand the various architectural designs. There were studied five testbeds proposed in research papers. The testbed proposed on [33], called "Open and Programmable 5G Network-in-a-Box"is a network-in-box solution that tests 5G Non Standalone mode and 4G LTE mode. The second one, "5G Testbed Development for Network Slicing"proposed in [34] uses cloud computing to enable network slicing. The [35] testbed, "Testbed for 5G Connected Artificial Intelligence on Virtualized Networks,"enables machine learning applications to optimize the network based on AI decisions. The testbed presented in [38], "Virtualized C-RAN with Mininet and OAI Supporting Flexible Network Topologies"provides a virtualized cloud RAN with a BBU pool and RRH pool to experiment with

different configurations. The last testbed [39], "A Cloud-based SDN / NFV Testbed for End-to-End Network Slicing in 4G / 5G", uses OSM as NFV orchestrator, OAI as core, srsLTE as RAN, and SDN controllers in RAN, core, and TN to enable end-to-end slicing. Each testbed has unique features and objectives for evaluating the performance, security, and functionality of 5G networks.

After that, emulators/simulators for the RAN and software modules that implement the 5GC functionalities were identified and the most suitable ones, based on their functionalities, were chosen.

A conceptual design utilizing Simu5G and Free5GC was proposed and deployed on two virtual machines using VMWare as the hypervisor. The server was located at the University of Minho.

Simu5G is a simulation tool that provides end-to-end simulation of 5G networks using the OMNeT++ event simulator. Unlike other simulation tools that only simulate the physical layer, Simu5G provides a simulation of the entire 5G protocol stack, from the application layer down to the physical layer. One of the advantages of Simu5G is that it is based on the OMNeT++ framework, which provides a component architecture for models, making it easy to reuse and integrate different OMNeT++ base frameworks. For instance, Simu5G can be integrated with Veins, a vehicular network simulation framework.

Free5GC is an open-source project written in GO language that aims to implement the core functionalities defined in 3GPP Release 15 and beyond. The project includes all the CN entities except UDSF, AF, and NEF, which are not expected to be implemented in the near future. It also has 5GC Orchestrator integrated, supports application services, and has network services and slicing like OAI CN.

Finally, the network was tested to evaluate using simulation and emulation modes. Moreover, a test involving MEC applications was performed.

The proposed virtualized 5G network was implemented and tested. Simu5G was employed to emulate UEs and RAN, and was connected to free5GC, a 5G core. Due to the limitations of Simu5G, the network does not implement a control plane and does provide MEC features that have been tested.

5G is becoming more software-based and virtualized, leading to significant disruptions and opportunities for innovation. The previously discussed testbeds were developed to study various aspects of 5G, including deployment scenarios, network slicing, and the use of AI, being limited in scalability. The testbed proposed here with Simu5G is a fully virtualized solution, which offers a more cost-effective approach than using physical equipment. However, the hardware used in the simulation and emulation still has an impact. In particular, when emulating a network, it is important to ensure that the simulation maintains a real-time requirement. Additionally, there are recommended hardware requirements for the 5G core implementation. This testbed is most concentrated on offering an environment for developers to test applications and different scenarios where 5G will be important at the application level. That's why not having the control plane is not a big disadvantage since most of the studied testbeds are more interested in studying how the 5G network can improve by integrating orchestrators, adding how network functions can be deployed in different architectural designs or specific algorithms at the protocol level. Instead, here what matters is the application level and how the application will behave in a 5G scenario by being able to customize

topologies and adding features such as D2D and MEC.

As mentioned in [59], several enablers, such as end-to-end slicing, Self-Organizing Network (SON), and Cloud RAN are under investigation with no standard recommendations. The proposed testbed allows testing at the application level with stable enablers and implementations. Therefore, having a control plane available will be more important for these new enablers.

Simu5G offers scenarios with integration with both 5G and 4G (non-standalone), which is important since some transition from the previous generation is being made with this hybrid approach. This network is a valuable tool for researchers and developers who will be able, among others, to test mobility scenarios as well as to evaluate MEC applications.

7.2 Future Work

Considering that 5G has many use cases, there are many possibilities for future work. In terms of infrastructure, for example, monitoring and availability of the resources used by the core machine could be necessary if the test bed starts to have much use. The possibility of guaranteeing the availability of the network could be implemented by replicating some NFs such as SMF DNN and UPD or selecting the UPF according to the gNodeB, for instance, to distribute traffic. This could be done by setting up new virtual machines with the according NFS and configuring the configuration files available to support multiple NFS and know their addresses.

Regarding integration with other simulation tools, VEINs would be a good framework for vehicular network research, giving a prominent feature to the testbed. Moreover, it should be possible to integrate Simu5G and Veins, since it is an open-source framework for running vehicular network simulations based on OMNeT++ and SUMO.

Another feature that can be integrated in the future is support for network slicing, a trending 5G technology, which would require much more technical work. It would require developing the procedures in Simu5G at the RAN level to select the Slice and control plane communication with the core. The Integration of AI, which brings innovation to 5G Networks, would be another feature. There is some information on integration with OMNeT++, but the interoperability with Simu5G would be necessary to study.

Concerning automation, RAN usage is IDEs-centric, which is suitable for beginners. In this sense, another possible improvement would be to develop a framework that converts NED files, which are declarative files that define the network topology. Simu5G models such as UE, background cells, and channel control are imported in the NED file to represent network elements. The channel control defines the properties of the transmission medium, while connections between the network elements are established using the elements available gates, to XML and updates them in the Simu5G VM. In that way, it would be possible to run the emulation without an IDE. Moreover, it is necessary to repeatedly open manually new terminals and connections to test several applications, which could be improved.

Bibliography

[1] A. Gupta and R. K. Jha. "A Survey of 5G Network: Architecture and Emerging Technologies." In: *IEEE Access* 3 (2015), pp. 1206–1232. issn: 21693536. doi: 10.1109/ACCESS.2015.2461602 (cit. on pp. 1, 4, 5).

[2] G. Pujolle. *Software Networks: Virtualization, SDN, 5G and Security*. Wiley-ISTE, 2015. isbn: 2019950464 (cit. on p. 1).

[3] I. Parvez, A. Rahmati, I. Guvenc, A. I. Sarwat, and H. Dai. "A survey on low latency towards 5G: RAN, core network and caching solutions." In: *IEEE Communications Surveys and Tutorials* 20.4 (2018), pp. 3098–3130. issn: 1553877X. doi: 10.1109/COMST.2018.2841349. arXiv: 1708.02562 (cit. on p. 2).

[4] T. Lynn, J. G. Mooney, B. Lee, · Patricia, and T. Endo. *The Cloud-to-Thing Continuum: Opportunities and Challenges in Cloud, Fog and Edge Computing*. July. 2020, p. 163. isbn: 9783030411091. doi: 10.1007/978-3-030-41110-7. url: http://www.palgrave.com/gp/series/16004 (cit. on p. 2).

[5] *Intel and Ericsson Pioneer Cloud RAN Acceleration - Technology@Intel*. url: https://blogs.intel.com/technology/2021/06/intel-ericsson-collab-on-cloud-ran/ (visited on 12/17/2021) (cit. on p. 2).

[6] I. Chih-Lin, J. Huang, R. Duan, C. Cui, J. Jiang, and L. Li. "Recent progress on C-RAN centralization and cloudification." In: *IEEE Access* 2 (2014), pp. 1030–1039. issn: 21693536. doi: 10.1109/ACCESS.2014.2351411 (cit. on p. 2).

[7] G. Nardini, D. Sabella, G. Stea, P. Thakkar, and A. Virdis. "SiMu5G–An OMNeT++ library for end-to-end performance evaluation of 5G networks." In: *IEEE Access* 8 (2020), pp. 181176–181191. issn: 21693536. doi: 10.1109/ACCESS.2020.3028550 (cit. on pp. 2, 12, 13).

[8] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith. "srsLTE: An open-source platform for LTE evolution and experimentation." In: *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM* 03-07-Octo.October (2016), pp. 25–32. doi: 10.1145/2980159.2980163. arXiv: 1602.04629 (cit. on p. 2).

[9] *free5GC*. url: https://www.free5gc.org/ (visited on 02/03/2022) (cit. on pp. 2, 15).

[10] H. Halabian. "Distributed Resource Allocation Optimization in 5G Virtualized Networks." In: *IEEE Journal on Selected Areas in Communications* 37.3 (2019), pp. 627–642. issn: 15580008. doi: 10.1109/JSAC.2019.2894305 (cit. on p. 3).

[11] A. Esmaeily and K. Kralevska. "Small-Scale 5G Testbeds for Network Slicing Deployment: A Systematic Review." In: *Wireless Communications and Mobile Computing* 2021 (2021). issn: 15308677. doi: 10.1155/2021/6655216. arXiv: 2104.08834 (cit. on p. 3).

[12] M. Agiwal, A. Roy, and N. Saxena. "Next generation 5G wireless networks: A comprehensive survey." In: *IEEE Communications Surveys and Tutorials* 18.3 (2016), pp. 1617–1655. issn: 1553877X. doi: 10.1109/COMST.2016.2532458 (cit. on pp. 4, 18).

[13] A. Gohil, H. Modi, and S. K. Patel. "5G technology of mobile communication: A survey." In: *2013 International Conference on Intelligent Systems and Signal Processing, ISSP 2013* (2013), pp. 288–292. doi: 10.1109/ISSP.2013.6526920 (cit. on p. 4).

[14] S. Mumtaz, Rodriguez , Jonathan, Dai , Linglong. *mmWave Massive MIMO 1st Edition A Paradigm for 5G*. 2016, p. 10. isbn: 9780128044186 (cit. on p. 5).

[15] "Ieee 5G and Beyond Technology R oadmap." In: (2017) (cit. on p. 5).

[16] TSGS. *TS 123 501 - V15.3.0 - 5G; System Architecture for the 5G System (3GPP TS 23.501 version 15.3.0 Release 15)*. 2018. url: https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx (cit. on p. 6).

[17] NFV. "GS NFV-MAN 001 - V1.1.1 - Network Functions Virtualisation (NFV); Management and Orchestration." In: (2014). url: http://portal.etsi.org/chaircor/ETSI_support.asp (cit. on pp. 8, 9).

[18] B. Blanco, J. O. Fajardo, I. Giannoulakis, E. Kafetzakis, S. Peng, J. Pérez-Romero, I. Trajkovska, P. S. Khodashenas, L. Goratti, M. Paolino, E. Sfakianakis, F. Liberal, and G. Xilouris. "Technology pillars in the architecture of future 5G mobile networks: NFV, MEC and SDN." In: *Computer Standards and Interfaces* 54 (2017), pp. 216–228. issn: 09205489. doi: 10.1016/j.csi.2016.12.007 (cit. on p. 10).

[19] E. Edition, O. Systems, S. Edition, and B. D. Communications. *the William Stallings Books on Computer Data and Computer Communications , Eighth Edition*. Vol. 139. 3. 2011, pp. 523–535. isbn: 9780136073734. doi: 10.1007/11935070 (cit. on pp. 10, 11).

[20] M. S. Bonfim, K. L. Dias, and S. F. Fernandes. "Integrated NFV/SDN architectures: A systematic literature review." In: *ACM Computing Surveys* 51.6 (2019). issn: 15577341. doi: 10.1145/3172866. arXiv: 1801.01516 (cit. on p. 10).

[21] S. Y. Wang, C. L. Chou, and C. M. Yang. "EstiNet openflow network simulator and emulator." In: *IEEE Communications Magazine* 51.9 (2013), pp. 110–117. issn: 01636804. doi: 10.1109/MCOM.2013.6588659 (cit. on p. 11).

[22] G. Nardini, G. Stea, A. Virdis, and D. Sabella. "Simu5G: A system-level simulator for 5G networks." In: *SIMULTECH 2020 - Proceedings of the 10th International Conference on Simulation and Modeling Methodologies, Technologies and Applications* Simultech (2020), pp. 68–80. doi: 10.5220/0009826400680080 (cit. on pp. 12, 13).

[23] *OMNeT++ - Simulation Manual*. url: https://doc.omnetpp.org/omnetpp5/manual/ (visited on 02/03/2022) (cit. on p. 14).

[24] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia. "Open, Programmable, and Virtualized 5G Networks: State-of-the-Art and the Road Ahead." In: *Computer Networks* 182.May (2020), p. 107516. issn: 13891286. doi: 10.1016/j.comnet.2020.107516. arXiv: 2005.10027. url: https://doi.org/10.1016/j.comnet.2020.107516 (cit. on pp. 15, 18–20, 32).

[25] *5G CORE NETWORK – OpenAirInterface*. url: https://openairinterface.org/oai-5g-core-network-project/ (visited on 02/03/2022) (cit. on p. 15).

[26] *Open5GS | Open5GS is a C-language implementation of 5G Core and EPC, i.e. the core network of NR/LTE network (Release-16)*. url: https://open5gs.org/open5gs/ (visited on 02/03/2022) (cit. on p. 15).

[27] M. Condoluci and T. Mahmoodi. "Softwarization and virtualization in 5G mobile networks: Benefits, trends and challenges." In: *Computer Networks* 146 (2018), pp. 65–84. issn: 13891286. doi: 10.1016/j.comnet.2018.09.005. url: https://doi.org/10.1016/j.comnet.2018.09.005 (cit. on p. 18).

[28] A. Yazar, B. Peköz, and H. Arslan. "Fundamentals of Multi-Numerology 5G New Radio." In: (2018). arXiv: 1805.02842v2. url: http://arxiv.org/abs/1805.02842 (cit. on p. 18).

[29] W. S. Afifi, A. A. El-Moursy, M. Saad, S. M. Nassar, and H. M. El-Hennawy. "Importance of Cloud Computing in 5G Radio Access Networks." In: January (2019), pp. 255–268. doi: 10.4018/978-1-7998-1152-7.ch010 (cit. on p. 18).

[30] G. P. A. W. Group. "View on 5G Architecture." In: *Version 3.0, June 2019* June (2019), pp. 21–470. url: https://5g-ppp.eu/wp-content/uploads/2019/07/5G-PPP-5G-Architecture-White-Paper_v3.0_PublicConsultation.pdf (cit. on p. 19).

[31] "cups-white-paper." In: () (cit. on p. 19).

[32] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines. "5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges." In: *Computer Networks* 167.2020 (2020). issn: 13891286. doi: 10.1016/j.comnet.2019.106984 (cit. on p. 19).

[33] A. Aijaz, B. Holden, and F. Meng. “Open and Programmable 5G Network-in-a-Box : Technology Demonstration and Evaluation Results.” In: (), pp. 6–8. arXiv: arXiv:2104.11074v1 (cit. on pp. 20, 65).

[34] A. Shorov. “5G testbed development for network slicing evaluation.” In: *Proceedings of the 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering, EIConRus 2019* (2019), pp. 39–44. doi: 10.1109/EIConRus.2019.8656861 (cit. on pp. 21, 65).

[35] C. V. Nahum, L. De Novoa Martins Pinto, V. B. Tavares, P. Batista, S. Lins, N. Linder, and A. Klautau. “Testbed for 5G Connected Artificial Intelligence on Virtualized Networks.” In: *IEEE Access* 8.MI (2020), pp. 223202–223213. issn: 21693536. doi: 10.1109/ACCESS.2020.3043876 (cit. on pp. 22, 65).

[36] *Mininet: An Instant Virtual Network on Your Laptop (or Other PC) - Mininet*. url: http://mininet.org/ (visited on 02/03/2022) (cit. on p. 22).

[37] *Flexran · Mosaic5G*. url: https://mosaic5g.io/flexran/ (visited on 02/03/2022) (cit. on p. 22).

[38] G. Couto, C. Nahum, V. Tavares, J. Pereira, P. Batista, and A. Klautau. “Virtualized C-RAN with Mininet and OAI Supporting Flexible Network Topologies.” In: (2020). doi: 10.14209/sbrt.2020.1570657866 (cit. on pp. 23, 65).

[39] A. Esmaeily, K. Kralevska, and D. Gligoroski. “A Cloud-based SDN / NFV Testbed for End-to-End Network Slicing in 4G / 5G.” In: (2020), pp. 29–35 (cit. on pp. 23, 24, 66).

[40] *my5G/my5G-RANTester: my5G-RANTester is a gNB/UE simulator for studying 3GPP standards and stressing a 5G core.* url: https://github.com/my5G/my5G-RANTester/ (cit. on p. 25).

[41] L. B. Silveira, H. C. de Resende, C. B. Both, J. M. Marquez-Barja, B. Silvestre, and K. V. Cardoso. *Tutorial on communication between access networks and the 5G core*. Oct. 2022. doi: 10.1016/j.comnet.2022.109301 (cit. on p. 26).

[42] *omec-project/gnbsim: gNB simulator*. url: https://github.com/omec-project/gnbsim (cit. on p. 26).

[43] *Home · aligungr/UERANSIM Wiki*. url: https://github.com/aligungr/UERANSIM/wiki (cit. on p. 26).

[44] *Simu5G - 5G New Radio User Plane Simulator for OMNeT++ and INET*. url: http://simu5g.org/ (cit. on p. 28).

[45] A. Varga and R. Hornig. *AN OVERVIEW OF THE OMNeT++ SIMULATION ENVIRONMENT*. isbn: 9789639799202 (cit. on p. 30).

[46] *veth(4) - Linux manual page*. url: https://man7.org/linux/man-pages/man4/veth.4.html (visited on 02/03/2022) (cit. on p. 31).

[47] T. Dreibholz. "Flexible 4G/5G Testbed Setup for Mobile Edge Computing Using OpenAirInterface and Open Source MANO." In: *Advances in Intelligent Systems and Computing* 1150 AISC (2020), pp. 1143–1153. issn: 21945365. doi: 10.1007/978-3-030-44038-1_105 (cit. on p. 32).

[48] *Simu5G - 5G New Radio User Plane Simulation Model for INET*. url: http://simu5g.org/install.html (cit. on p. 34).

[49] *Home · free5gc/free5gc Wiki*. url: https://github.com/free5gc/free5gc/wiki (cit. on p. 34).

[50] *OMNeT++ Installation Guide*. 1992 (cit. on p. 34).

[51] G. Nardini, G. Stea, and A. Virdis. "Scalable real-time emulation of 5G networks with Simu5G." In: *IEEE Access* 9 (2021), pp. 148504–148520. issn: 21693536. doi: 10.1109/ACCESS.2021.3123873 (cit. on p. 42).

[52] *Simu5G - 5G New Radio User Plane Simulator for OMNeT++ and INET*. url: http://www.simu5g.org/emulation.html (visited on 01/24/2022) (cit. on p. 42).

[53] A. Noferi, G. Nardini, G. Stea, and A. Virdis. "Deployment and configuration of MEC apps with Simu5G." In: (Sept. 2021). url: http://arxiv.org/abs/2109.12048 (cit. on p. 44).

[54] S. Kekki, W. Featherstone, Y. Fang, P. Kuure, A. Li, A. Ranjan, D. Purkayastha, F. Jiangping, D. Frydman, G. Verin, K.-W. Wen, K. Kim, R. Arora, A. Odgers, L. M. Contreras, and S. Scarpina. *ETSI White Paper No. 28 MEC in 5G networks*. 2018. url: www.etsi.org (cit. on p. 44).

[55] Mec. *GS MEC 013 - V2.1.1 - Multi-access Edge Computing (MEC); Location API*. 2019. url: https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx (cit. on p. 46).

[56] Mec. *GS MEC 012 - V1.1.1 - Mobile Edge Computing (MEC); Radio Network Information API*. 2017. url: https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx (cit. on p. 46).

[57] I. of Electrical, E. Engineers, and I. C. Society. *2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*. isbn: 9781728144900 (cit. on p. 62).

[58] A. Noferi, G. Nardini, G. Stea, and A. Virdis. *Rapid prototyping and performance evaluation of MEC-based applications* (cit. on p. 62).

[59] L. B. D. Silveira, H. C. D. Resende, C. B. Both, J. M. Marquez-Barja, B. Silvestre, and V Cardoso. *Tutorial on communication between access networks and the 5G core*. url: https://www.3gpp.org/ (cit. on p. 67).