UNIVERSITAT POLITÈCNICA DE CATALUNYA - FACULTAT D'INFORMÀTICA DE BARCELONA

Master's Degree Thesis

Design and Implementation of a 5G Testbed in a Virtualized Environment

Supervisors

Dr. Jordi PAILLISSÉ VILANOVA

Prof. Pere BARLET ROS

Prof. Andrea BIANCO

Candidate

Giovanni COLONNI

July 1, 2022

Summary

The new 5G network is one of the hot-topics of this century and promises great improvements in the world of mobile networks, network applications and the way society interacts with itself and technology. As this new network is a very complex and articulated system, the management and maintenance tools will also have to evolve accordingly. In this thesis, I attempt to introduce the concept of the digital twin, already imagined in the book Mirror Worlds by David Gelernter, to the context of the 5G network. The digital twin is a tool for supervision, control and analysis that combines the use of data, Machine Learning and simulation environments in order to create a virtual parallel system, faithful to the real one, that reacts to the same stimuli as its physical twin. This concept already exists in the engineering world in various forms. Thanks to the BNN (Barcelona Neural Network centre) research group, we have applied this idea to the 5G-core network, a concept that has already been theoretically explored by others but which to date has not found a real implementation. The following work consists of the design and development of a testbed in order to run simulations of plausible 5G network usage scenarios through simulators. The goal is to obtain datasets to train artificial intelligence models to predict certain aspects of network behavior.

Acknowledgements

I would like to thank everyone who stood by me on this journey that led to the development of this project and the end of my course of study. Special thanks go to Jordi Paillissé Villanova who with professionalism and patience has followed me along this path, listened to me and revised this work, a crucial contribution without which nothing would have been possible. Another special thank goes to my family, who believed in me all these years and put their trust in me, a trust that I hope I have reciprocated. And finally thanks to all the friends who have made this journey more fun and lighter, I will not forget.

Table of Contents

List of Figures

List of Abbreviations

DT
Digital Twin

AI
Artificial Intelligence

UE
User Equipment

GNB
gNodeB

PDU
Protocol Data Unit

SDO
Standard Development Organization

KPI
Key Performance Indicator

SDN
Software Defined Network

RTT
Round Trip Time

Gbps
Gigabit per second

IOT

Internet Of Things

BNN

Barcelona Neural Network Center

RAN

Radio Access Network

VoLTE

Voice over LTE

IETF

Internet Engineering Task Force

3GPP

3rd Generation Partnership Project

ML

Machine Learning

NN

Neural Network

GNN

Graph Neural Network

URLLC

Ultra Reliable and Low Latency Communication

C-V2X

Cellular - Vehicle to Everything

ISP

Internet Service Provider

NFV

Network Function Virtualization

Chapter 1

Introduction

The new 5G mobile network is one of the hottest topics of this century so far and it is a real promise. The fifth generation of mobile network addresses the most challenging and promising requirements on performance and use cases. Some of these requirements are listed below [1]:

- 1-10Gbps in end-to-end communication
- 1 ms RTT (Round Trip Time)
- 99.99% of availability (perceived)
- 100% coverage

In fact, the idea is that the fifth generation of mobile networks will be a complete revolution in which the world of IOT, augmented reality and mass communication will be an established reality across the globe. However, such a network needs a great deal of engineering effort, from the installation to the maintenance of the network to its evolution. Engineering for its part has many tools to control, monitor and test complex and articulated systems, but new technologies also require new tools. For this reason, such a dense and complex network - both in terms of network devices and in terms of users (mobile, IOT, machine-to-machine communication, and so on) - needs a monitoring tool that can not only monitor the status of the network in real time, but can also allow new use cases and possible extensions to be tested without having to interact with the real network, i.e., without touching the physical, real system. Such a tool must be a kind of virtual parallel system that follows the course and dynamics of the real system in a faithful and accurate manner, able to capture all the characteristics of the system. Such a system has already been imagined and goes by the name of *Digital Twin*.

1.1 Background

The project proposed here follows the idea of the research group BNN, Barcelona Neural Network Center, to create a Digital Twin for the 5G-core network. In mobile networks we can in fact distinguish two components, the Radio Access Network (RAN) and the Core Network.

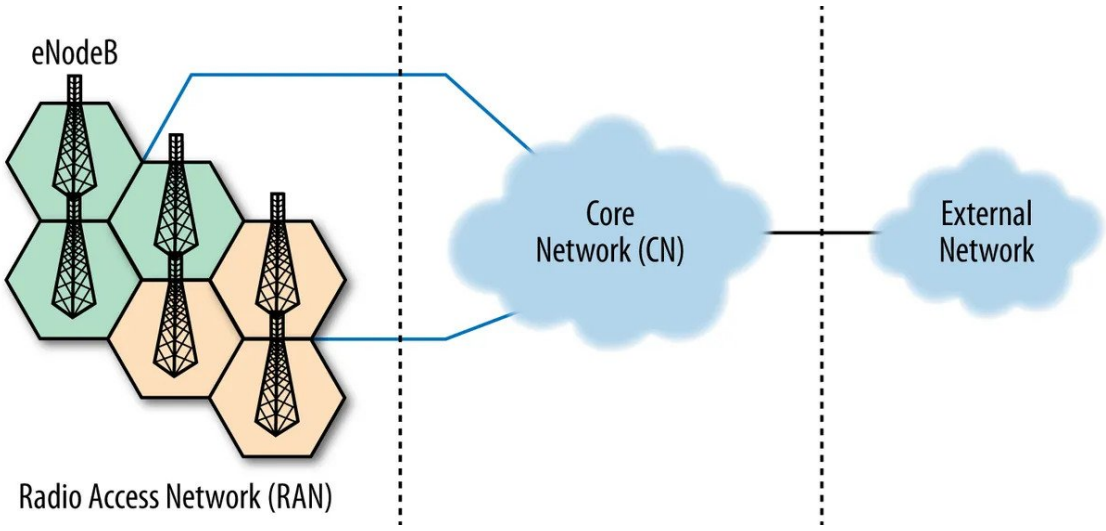

Figure 1.1: Radio Access Network and Core Network, from [2]

The RAN is simply the set of antennas and connections to which a device, such as a mobile phone, connects in order to access the internet or other networks. The core network, on the other hand, is the operator's internal network that performs administrative and control functions, sits between the radio access network and all the other networks, and performs all those functions necessary in order to use services such as VoLTE, SMS, packets exchange and so on. It is also responsible for checking whether those who connect to the RAN are entitled to do so, and this very often coincides with having paid the operator for the required services. The new 5G-core network has a cloud-based (or rather cloud-native) architecture and is rooted in the idea of the service-based architecture (SBA architecture). The idea is that overall the architecture can be made of smaller components that communicate with each other, somewhat like the architectures used in software, Service Oriented Architecture (SOA). This allows not only greater flexibility but also the possibility of combining components purchased from other vendors or third parties. This concept, combined with cloud infrastructure and virtualization technologies, makes it possible to create architectures that are scalable, modular, and relatively inexpensive since they can be implemented in general-purpose hardware. In the end, the 5G-core network is nothing more than a mesh network of interconnected services that

communicate via APIs, but we will discuss the architecture more extensively in the next chapter. The 5G network, as already mentioned, is also full of engineering challenges and stringent requirements that contribute to the implementation of the most ambitious use cases. This is why such a network is complex both to operate and to maintain. A key tool to manage the network effectively and efficiently is to have a digital clone of our physical system (i.e., the 5G network) that updates itself in parallel and in real time, fed with the data collected in the network itself. The idea within the study group is that recent studies and technologies related to the branch of Machine Learning can effectively mimic real-world networks in a precise and accurate manner, and can thus contribute substantially to the management of a large, dense, and complex network such as the one proposed by 3GPP for the fifth generation of mobile networks.

1.2 Motivation

The construction of a Digital Twin for the 5G-core network is interesting for several reasons. In fact, as already mentioned, there are new types of network applications that require very stringent requirements and parameters, and the classical solutions used to date, such as over-provisioning of network resources or old Quality of Service techniques, have serious limitations. New applications, such as holographic telepresence or vehicular networks, need requirements such as ultra-low deterministic latency or a network that constantly changes topology. Similarly, the total number of network devices is constantly increasing, so there is a need for tools for massive communication and therefore for networks that are scalable and above all predictable in terms of their behaviour. The Digital Twin has the potential to be the key to managing networks of such magnitude and complexity. The engineering industry is not new to the Digital Twin concept, which is already used in other applications such as engine design or for Industry 4.0 [3], as an example IBM has an exchange for Digital Twins of physical systems where you can buy and sell this kind of software [4]. Also other entities such as the Standard Development Organization (SDO) and the IETF start to work on the definition of a Digital Twin [5]. The Digital Twin, a concept that will be explored extensively in future chapters, is interesting when applied to 5G networks because it allows the network operator to design and test optimization solutions over the network, perform troubleshooting of real network problems, what-if analysis, network anomaly detection or upgrading (in terms of nodes such as antennas or any other component of interest) without interacting with the real system and thus preserving the state of the network used by the customers. However, in order to create a Digital Twin of the 5G-core network, we need a large amount of data to train statistical models that can mimic the behaviour of the physical system, i.e.,

the real network.

1.3 Goal of the thesis

The aim of this thesis is to create a system capable of simulating 5G network operations using two simulators, one for the radio access network and another for the core network. The purpose of the simulation is to create plausible scenarios in which a certain network topology is created (specified within a file) and a certain traffic pattern is proposed to it, i.e., a certain number of User Equipments (UEs) that are comparable to generic devices. The configuration also makes it possible to specify the manner in which these devices are activated, i.e., how many devices at a time, the time interval between one block of devices and another, and the distribution of devices in the block. The UEs are then activated (or switched on) and connected to the core network in order to register themselves in the system and instantiate the Protocol Data Unit (PDU), that will later be used by the user equipment to exchange data with the network, both that of the operator and the Internet in general. The idea is to capture the traffic generated from the UEs that are connected, and from the traffic extract the sequence of messages related to them. The data collected relates to the time required to instantiate the PDU for each UE that is executed; in particular, we are interested in capturing the exchange of messages within the core network caused by a UE requesting to instantiate a PDU. In fact, the activation of an UE, its registration, and the instantiation of the PDU cause a series of messages that the various components of the core exchange and which, in a concurrent manner, make possible the instantiation of the PDU and thus the possibility for the UE to use all the various network services. Once the sequence has been captured for each UE, the instants at which the messages reached the various components of the core are extracted in order to create the dataset. The concept of PDUs, UEs, gNodeb and the various core components will be explained later.

Chapter 2

Digital Twin - State of the art

The idea of the Digital Twin was initially introduced by David Gelernter with the publication of 'Mirror World', but the term Digital Twin was later coined by John Vickers in one of his publications while working for NASA. Leaving terminology aside, the concept of the Digital Twin is that the set of data and measurements collected, with the appropriate technologies and models, can, if combined correctly, represent an entity in its own right capable of following the life cycle of its physical counterpart as a parallel but digital system. However, this presupposes that between the physical system and its virtual twin there is a connection in which information flows continuously feeding the digital counterpart. The literature around the concept of the Digital Twin is vast and varied, and various definitions are proposed. One of these comes to us from the book Digital Twin Transdisciplinary Systems [6].

"Digital Twin (DT) - the Digital Twin is a set of virtual information constructs that completely describe a potential or actual physical product from the micro atomic level to the macro geometric level. At its optimum, any information that can be obtained from the inspection of a physical artefact can be obtained from its Digital Twin".

Thus, it is something that mimics the nature and dynamics of the system on a moment-by-moment basis, allowing it to be planned and at the same time monitor its inputs and outputs. Another interesting definition is the one proposed by the IETF on the concept of a Digital Twin for networks [7]:

"Digital Twin is the real-time representation of physical entities in the digital

world. It has the characteristics of virtual-reality interrelation and real-time interaction, iterative operation and process optimization, as well as full life-cycle, and full business data-driven."

The overall vision of the Digital Twin is to be a decision support, design and management tool for the system under consideration, a computer network, in this case.

This vision is also the one shared by the research group albeit with some differences, in the above mentioned IETF paper it is assumed that it is a simulation platform that hosts the Digital Twin, while within the research group we believe that it is possible to achieve this by using Machine Learning and Neural Network technologies [8]. In fact, there are several studies that show how ML applied to the network scenario can produce reliable results such as in the case of path delays [9] or optimisation in data-center networks [10]. The use of ML or NN models makes it possible to reduce simulation costs, which are usually prohibitive and, thus, make the simulation not very scalable. The use of ML or NN models makes it possible to reduce simulation costs, which are usually prohibitive and not very scalable. In particular, the research group was able to demonstrate how Neural Networks are particularly effective in modelling and optimising networks. In particular, one study shows how these techniques can surpass current models based on Queueing Theory, which often fail to capture network characteristics and may sometimes make unrealistic assumptions such as assuming a traffic model like Poisson traffic [9].

With the use of these new technologies, it is possible to create a model of the network that can capture network characteristics. Using an optimisation algorithm, the system is able to find the best configuration with respect to reference metrics called Key Performance Indicator (KPIs). As the optimisation algorithm explores the configuration space to find the best one, the network model predicts what the KPI values will be. This is especially useful in SDN networks, that allow applying new configurations from a centralized controller. The resulting model proves to be more accurate [9] than the classical models mentioned above and lighter than models based on Queueing Theory or simulations. Another result of this work is to demonstrate how models based on Graph Neural Networks are more effective than conventional models based on Convolutional Neural Networks, Recurrent Neural Networks etc. [9].

At this point, however, one might still be inclined to think that the Digital Twin is a similar concept to simulation. In this regard it is good to make a distinction because although they both share the ability to perform simulations they have crucial differences. Although they are both models, a Digital Twin needs a realised physical system on which to base itself, in other words a product. In fact, the

Digital Twin begins its function when the product is in place and allows you to see how the product is virtually operating at the moment. Another major difference is that simulation is a static concept, so when a new configuration is made, a new simulation is made. The Digital Twin by contrast combines what is happening in the system at the moment, live, in relation to what is happening in the reality of the system. So it can be used to test new use cases directly on the system without interacting with it and thus without interfering with the real system. Simulators are also used on the design phase of a product and thus for research purposes, in other words they are product-oriented. The Digital Twin, on the other hand, makes it possible to integrate data from the entire product life cycle, making it a data-driven concept and thus oriented towards the product's business model, in other words, it is business-oriented.

2.1 Digital Twin 5G network

It is therefore clear how a Digital Twin can represent a game-change in the management of complex systems, whether purely physical as in the case of Industry 2.0 [8], or in the production of oil and gas [11]. However, no one has yet implemented a Digital Twin for the 5G network, although proposals exist [12].

It is expected that the 5G network and the new possibilities it brings can generate USD 12.3 trillion in economic output (forecast for 2035) and that the 5G network will support over 11 million businesses [12]. Although the possibilities are therefore enormous, realizing them is another matter and involves serious risks in terms of investment. This is because network deployment is complex and initially a hybrid deployment would have to begin, straddling the 4th generation (4G) and 5th generation (5G) mobile networks, and it is worth noting in this regard that there are countries where 4G network deployment is either non-existent or still in its infancy. Moreover, the 5G network is extremely complex in some of its scenarios and use cases, which makes the investment even more risky.

In this case, having a Digital Twin of the network would be crucial to be able to make decisions, plan scenarios, investments and actions in order to reduce the risks related to the complexity of the system, optimize the network and the way it operates and ultimately reduce costs for network operators.
Before talking about the general architecture of a possible Digital Twin based on ML and GNN, it is also necessary to talk about the challenges it poses in terms of engineering. First of all, a Digital Twin product must be able to generalize network topologies that are not included in the testbeds seen by the Digtial Twin during its training phase. This is because it is not possible to train the network based on the target network topology of the Digital Twin because it would be burdensome for the network operator itself, and secondly because that topology is not immutable

and could change for business reasons as well as for tastes and other reasons.

Another problem is the size of the network in which the Digital Twin operates compared to the size of the testbed in which it is tested, the former will certainly be larger than the latter. Because of this and the fact that the training phase is long, the Digital Twin cannot perform it again in the face of topological changes of any nature, and it is therefore necessary for the model to generalize with respect to topology.

Another challenge to the creation of a Digital Twin is posed by network traffic and the way in which this must be captured and processed. Indeed, in order to model the network, it must be able to handle and understand traffic at the level of individual flows. The problem here too is in the size, i.e., the number of flows in real networks is very large and therefore poses scalability problems. So building Machine Learning-Based models that can scale well with respect to the number of flows is still an open challenge.

Another problem is that of the explainability of the resulting model that makes up the Digital Twin. In fact the models in general are black-boxes, therefore not observable, and, hence, difficult to interpret. On the other hand, it is necessary for network operators to be able to interpret the forecasts made by the Digital Twin. There are advances in this field such as explainable AI, or more generally the networking community is investigating methods to interpret models based, for example, on Neural Networks [8].

Ultimately, a DT needs large amounts of data in order to operate, data that must be collected, managed and stored. The problem is that in a large and heterogeneous network, the data extracted are also heterogeneous and in large quantities, so that the labelling and formatting techniques that are usually used can be inefficient and not very scalable. For example, storing traffic flows is a challenge and costs a lot in terms of GB stored, because the flows can be very different in nature [8].

2.1.1 Architecture for the 5G Digital Twin

The Digital Twin is a data-driven paradigm, i.e., powered by the generation of data by the network that is the system we wish to model and mimic. In the particular case of this thesis work, the data were obtained from simulators, executed in virtual machines. Aware of the possible bias that the data will have, due to the nature of the simulation, these nevertheless represent a valid source of information, the reasons for this choice are essentially two:

- For feasibility reasons, the alternative would have been to buy physical antennas and physical devices, but this, besides being very expensive, also represents an obstacle to the size of the simulation and the possible scenarios that can be created.

- It is an exploratory and experimental work, in fact the construction of a Digital Twin for this type of network, the 5G network, is quite ambitious.

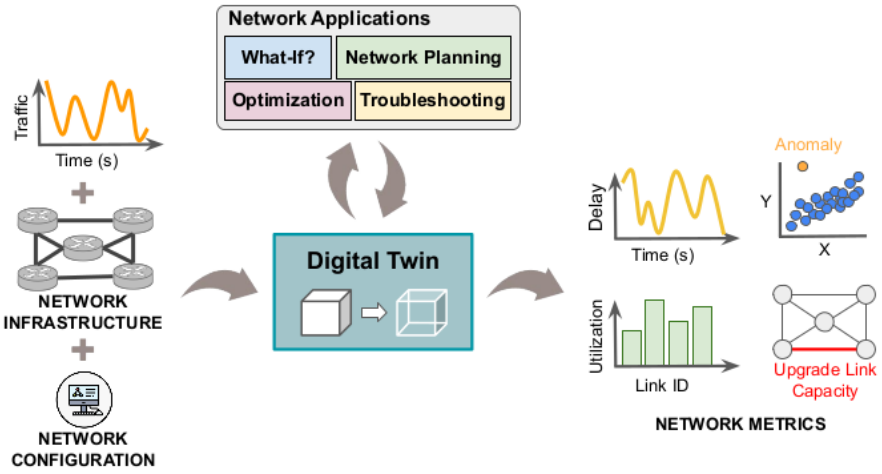

Figure 2.1: Architecture of the Network DT as propose by the BNN research group, from [8]

The architecture proposed by the research group for the realization of the DT is the one shown in Figure 2.1. The model takes as input information about the network, such as the network topology, understood as a graph composed of nodes and edges, in which each node and edge carries information characterizing them (e.g., link capacity, type of technology, etc.). It then takes in the network configuration, i.e., routing, scheduling policy etc., and the network traffic data which as mentioned is at the flows level. The Digital Twin at that point, which is represented here as a black-box, is the set of statistical models trained in the test phase, the applications that can be run on it, and the network optimizer.
The output of the Digital Twin depends on the network application or the task that we want to perform as showed in Figure 2.1 (i.e., what-if analysis, net-planning, optimization, troubleshooting). The advantages for the network operators is that since the Digital Twin is a copy of the network that is operating on, they can test all the possible configuration of the network, even the most disruptive ones that are unfeasible to test in the real network, because they would cause network failures or interruption on the services. The output of the Digital Twin are in general performance metrics (KPI) and they can be customized for the needs of the network operator (from per-path delays to global metrics values). The network model proposed for the Digital Twin is based on a Graph Neural Network (GNN), because, compared with the traditional Deep Learning models, GNNs have superior performance when modelling complex relationships and they are way more

performant in case of changes in the topology [9, 10]. The other application of the Digital Twin architecture is the optimizer Figure 2.2.

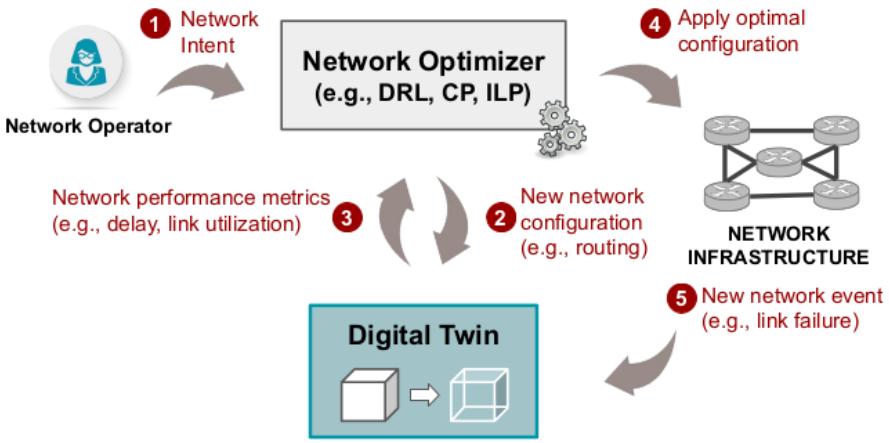

Figure 2.2: Optimizer of the DT, from [8]

The idea is the network operator wants to archive specific requirements in terms of network performance (ex. a certain per-path delay, maximum link utilization and so on). This configuration is given to the optimizer that calculates a new network configuration (so routing, policies, topology etc.), the optimizer calculate a new network configuration that is submitted to the Digital Twin as input. The Digital Twin at this point calculates the network performance metrics of interest (the PKI) and if the performances do not meet the expected ones, the optimizer continues to search for other network configurations until it finds the expected one, at that point the new configuration is injected in the real network. This is the power of the Digital Twin, testing any kind of configuration that is optimal with respect to any kinds of requirements without touching the real network.

Chapter 3

5G Network

3.1 5G Architecture

We are currently at the 18th Release of the 3GPP consortium for the 5G deployment standards and actually some of the requirements of the vision of the 5G network are currently missed such as the cell densification, the support for ultra low latency communication (the goal is to reduce it down to 1ms) and so on. Actually the deployments that are operative are hybrid solutions that use the non-standalone 3x option deployed with the 4G core network. This setup is not able to guarantee the performance that the 5G is able to archive in theory, as it was imaged at the beginning.

The next big improvements on the future standards will support services such as Ultra Reliable and Low Latency Communication (URLLC) that will guarantee a communication with 1 ms of latency, this is intended for all the scenarios that need almost real-time communication such as robotics, autonomous driving and Vehicle to Anything communication (sometimes referred as C-V2X).

Usually in a mobile network we can distinguish two different parts, the *Core Network* and the *Radio Access Network (RAN)*. The RAN is the radio part, the access point of the network, and it is responsible for aggregating the traffic from the end devices called, in general, *User Equipment (UE)*. These could be simple smartphone, IOT devices, robotic devices or in general whatever object has a SIM, so basically whatever is connected to the network. There are major improvements on the RAN side such as the adoption of the millimetre wave or beamforming, also there is the will of improving the density (per km^2) of the antennas. Also, the disaggregation of the RAN that enables the virtualization of some RAN components that will reduce the cost both in terms of hardware and software, in this way also other kind of ISP (of low level) can be created and so new business opportunities.

The other part of the mobile network is the Core Network that aggregates the data traffic from the UEs and performs other functions in order for the network to operate properly such as authentication of the subscribers and the devices, injecting policy for the subscriber, managing the mobility of the devices and routing the traffic from the operator network to the internet. The 5G-core is the one, alongside the RAN, in charge of enabling all the services that the network offers.

The 3GPP defines the 5G-core network as a cloud-aligned Service-Based-Architecture (SBA). The SBA is an architectural concept and provides the possibility of deploying all the components needed for the architecture to operate in a virtualized way. So all the components are virtualized and run on off-shell hardware, the general name for such a components is Network Function Virtualization (NFV) and the idea is to decouples software from hardware (similarly to Software Defined Network, SDN) in a way that they can be deployed in off-shell hardware without buying specific components and so by accelerating the deployment, by so opening the possibility of using cloud-based solution and edge-computing (or fog computing) in order to support even the most challenging scenario. NFV can be for example a Router, a Load Balance, a Firewall or a NAT.

Along with the SBA another key component of the 5G-core network is network slicing. The idea of network slicing is to split the physical network of the operator into multiple logical networks in order to customize the behaviour and the performance of the physical infostructure. This will enable different scenarios and use cases by injecting in the virtual network's policies and constraints such as resource availability, fixed latency paths and so on. So it is basically a virtual layer built at the top of the physical infostructure in order to simplify the management of the network and define specific requirements for each of the slices.

To explain the components of the 5G architecture we have to spend some more words on the SBA, that is the key point of the new 5G-core network. SBA is the set of interconnections between the network function (NF). It's defined by the 3GPP technical specification [13] and its the meeting point between Service-Oriented-Architecture and micro-services.

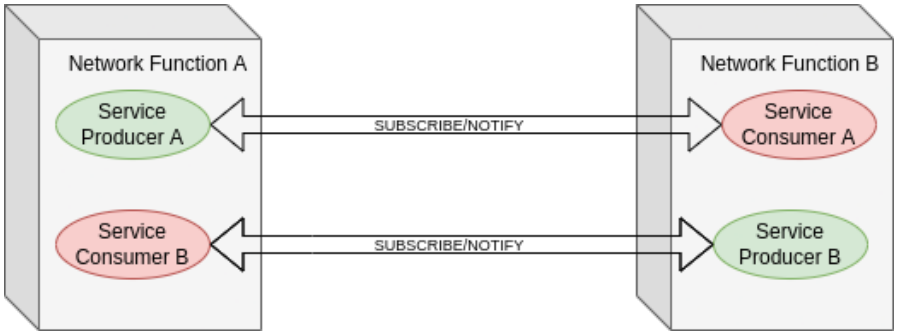

Figure 3.1: Service Base Architecture in Producer-Consumer model

All of this new concepts are used for changing the paradigm form having a very specific telecom-style components (hardware and software) to a web-based approach (then based on HTTP) that is more extensible, reusable and scalable. This approach is able to meet the new challenges and requirements of the 5G network. The summary of the architecture of the 5G network is depicted in Figure 3.2:

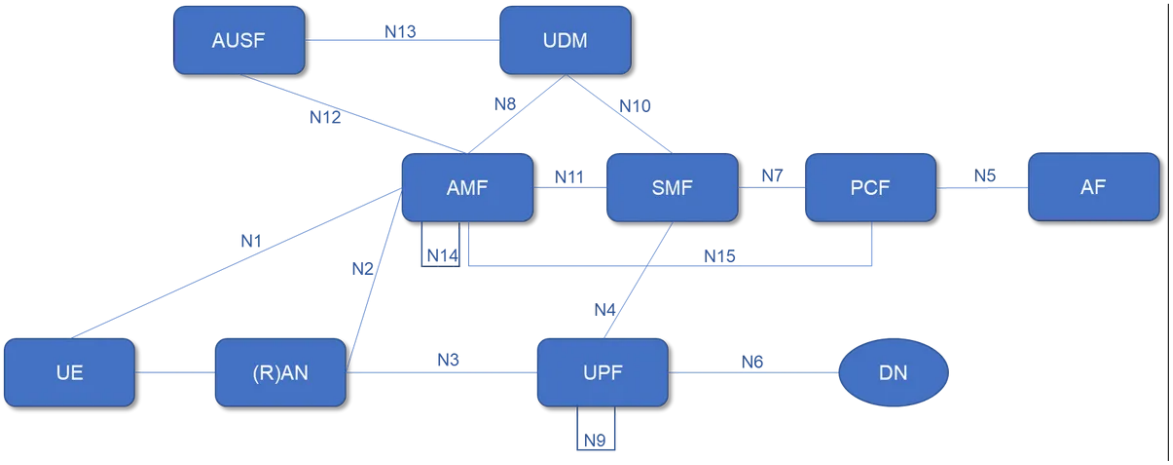

Figure 3.2: 5G-core architecture overview, from [14]

- *AMF*, Access and Mobility Management Function, is where the control plane of different access networks (in our case only the RAN) connects to the 5G-core. It is also the component responsible for authenticate the UE and so it permits to an UE to exchange traffic with a Data Network (there could be more then one data network but in general when we refer to Data Network we are talking about internet). It is also responsible for managing the UE mobility, so when the UE travel from one gNodeb to another one for guarantee the session continuity. Each gNodeb maintain a connection with the AMF.

- *gNodeb* is the radio access point where the device connects for accessing the network, also referred as base-station (it is basically the antenna).

- *SMF* Session Management Function keeps trace of the PDU sessions opened by the UEs for exchange the traffic (we will explain the PDU concept later) and managing the QoS flows inside the PDU. Since it is in charge of managing the sessions it also receives the PCC (Policy and Charging Charging Control) from the PCF, then the SMF will create a template by converting the rules in QoS policies that will be injected in the PDU session.

- *UPF*, User Plane Function, is the one in charge of forwarding the UE traffic between the access network (so the RAN) and the the data network and

enforcing the QoS rules in the template provided by the SMF. The UPF is also the one that is used as anchor point for the PDU [13].

- *UDM*, Unified Data Management is the one that stores the UE data such as the SUPI, the slices, the key for the authentication and so on. (There will be an example of such data below)
- *AUSF*, Authentication Server Function, is the one in charge of offering the services to the other components for the authentication of the UEs.
- *NRF* Network Repository Function, this component maintains an updated version of all the 5G network function, is like the software virtualized copy of the elements of the architecture and is the core component of the SBA.
- *NSSF*, Network Slide Selection Function, it perform the selection of the network slice (based on the UEs requests and settings) along the sliced network. It also allow the AMF to perform the initial registration of a PDU and also supports slide change and update.
- *PCF*, Policy Control Function, is responsible of managing the policy, so modification, deletion and retrieving of the policies of a UE. The policy are stored in the UDM.

All of this components are micro-services (even we can say that each component is composed by more then one micro-services) and they are software components. They can be deployed in the same way as a container is deployed in a cloud info-structure.

3.1.1 PDU Protocol Data Unit

When a UE (so a generic device that can connect to the 5G-core network) is turned on, it is going to connect to a 5G cell (so the gNodeb, the antenna) and it starts to perform the registration procedure. This procedure involves the AMF and the goal is to authenticate the SIM of the UE. The AMF consumes the services of the UDM and the AUSF to make sure that the UE has a valid subscription and if it's intended to join the 5G network. Here is where the predefined keys (k, and opc, more details later) are used for authenticate the UE. Once the UE is registered to the network it starts to instantiate the PDU through the PDU Session establishment procedure. The PDU is essentially a way to instruct the UPF to construct the data path for the UE and is the highest granular view of the connection of an UE to a Data Network. An UE can, in general, establish more then one PDU but each PDU is

always related to a Data Network, inside the PDU there could be more QoS flows with different requirements, those parameters depend on the policy and the kind of traffic. Each PDU can have several parameters:

- PDU session Identifier, the ID of the PDU
- S-NSSAI, is the identifier of the slice where the PDU will be instantiated
- DNN, Data Network Name, is the end point of the PDU since it is always refered to a DN
- PDU Session Type,defines the type of protocol that the end.user want to carry (IPv4, IPv6, Ethernet, Dual Stack or Unstructured)
- SSC Mode, is Session and Service Continuity mode and it's about the life of the PDU Session and the possibility to reallocate it once expired
- User Plane Security Information, indicates the kind of security and protection of the PDU Session, which kind of encryption and the integrity protection

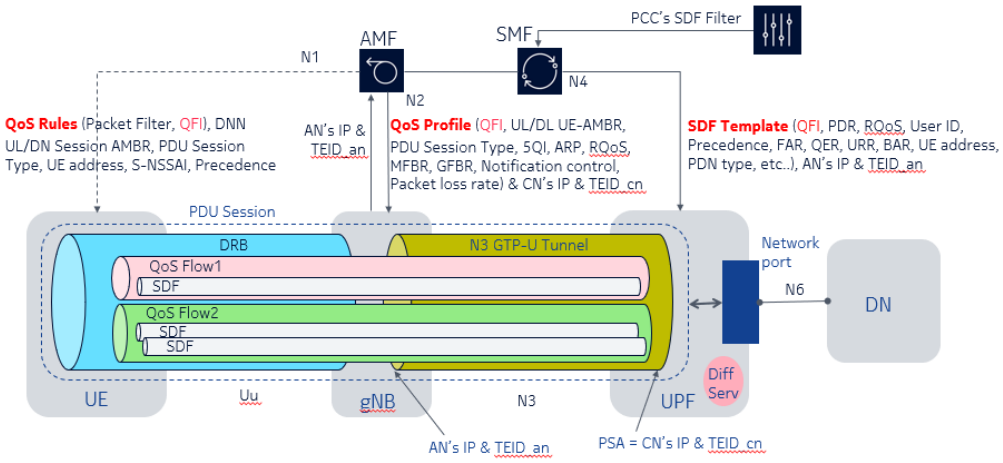

Figure 3.3: PDU with more QoS flows inside,from [15]

After the establishment of the PDU a Non-GBR (Guaranteed Bit Rate) QoS flow is created between the UE and the gNodeb in order to extend the GTP-U tunnel to the UE and so the possibility for the UE to exchange traffic (the tunnel that carry the user data toward the gNodeb and then the Data Network, we'll talk about this detail later). As showed in the Figure 3.3 inside the same PDU there could be more QoS flows with different SDF templates for different Data Network. For example in the first UE we have a non-GBR flow which carries the ordinary

traffic (ex. browsing) and another QoS flow that is GBR, that could be for example a video-conference or a live streaming. This is how the different QoS flows can be used. So basically for each PDU that the UE establish there are 2 kinds of tunnels:

- Bi-direction wireless tunnel (Data Radio Bearer) between the UE and the GNB (it is showed in the first part of the figure)
- 2 uni-directional tunnel between the gNodeb and the UPF and it's where the down-link and up-link traffic flow, so the data of the UE

The message procedure used to establish a PDU is reported in Figure 3.4 and it is explained in [13]. Those are the messages exchanged during the PDU Session Establishment procedure, but actually there is much more when an UE is turned on. We will take a closer look to the overall procedure in the next sessions.

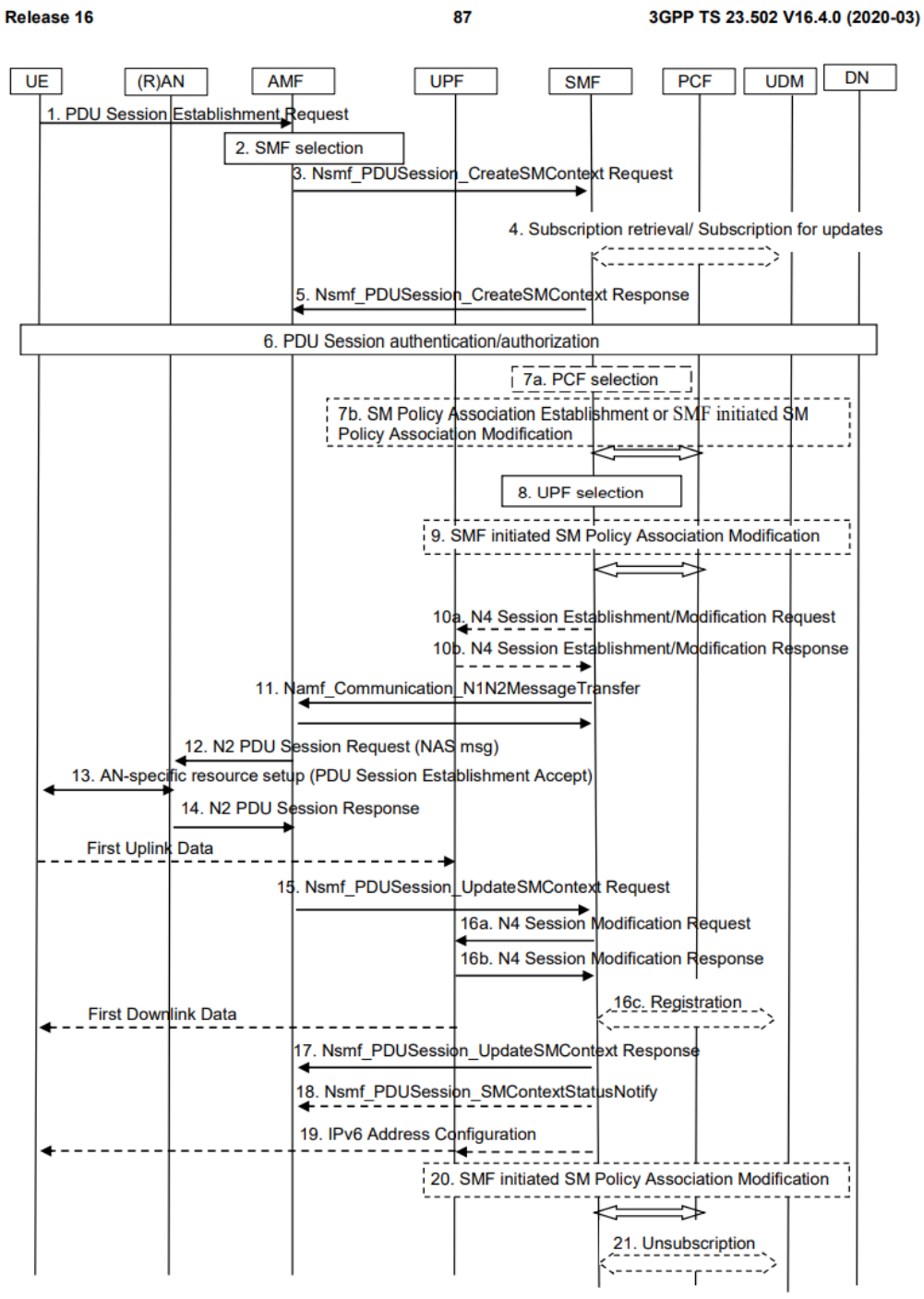

Figure 3.4: Message exchange procedure for UE registration and PDU instantiate, from [13]

Chapter 4

Developed System

The actual developed system is composed of 3 parts, the RAN part, the Core network part and the Analysis part. The system is based on 3 virtual machines that run in the data center of the UPC, in the space dedicated to the BNN research group. The VMs used are Ubuntu x86 with 4 CPUs Intel Xeon wiht 4GB or RAM. They run at the top of a VMWare hypervisor and they are co-located in the same physical server. The *VM1* is in charge of running the core network and perform the capture process. The *VM2* runs the RAN network, so the gNodeb and the UEs and the last machine, *VM3*, is in charge of controlling the two other machines and perform the analysis on the capture file in order to produce the data. The overall system is depicted in Figure 4.1.

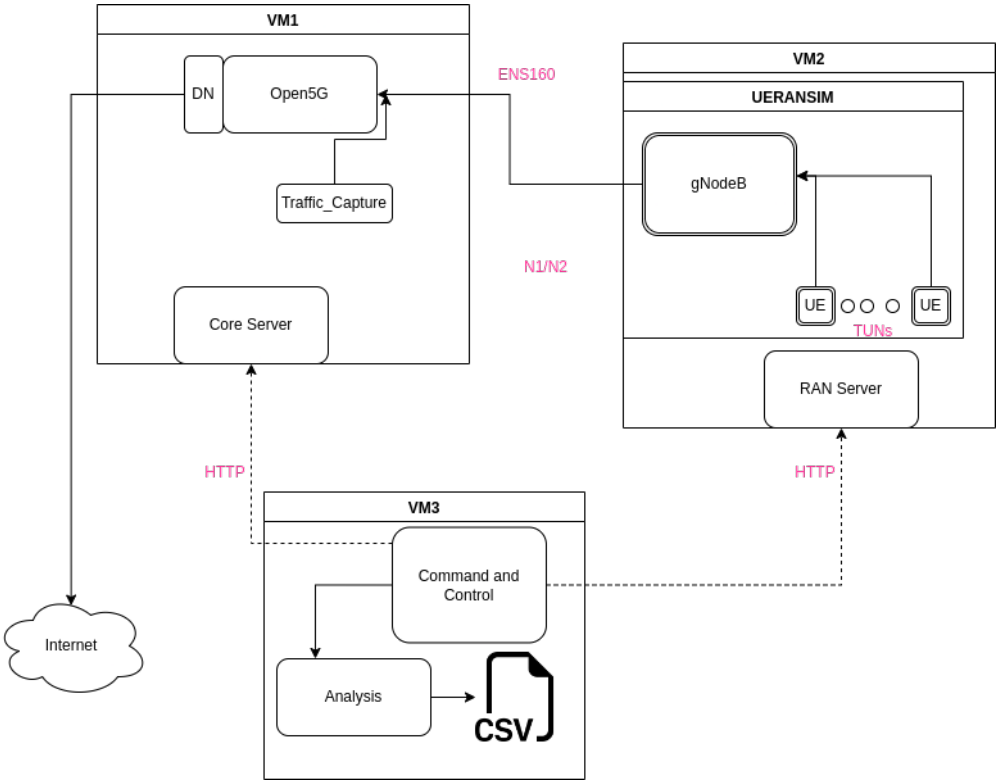

Figure 4.1: Architecture of the test-bed for the simulation

The purpose of the system is to perform simulations and collecting data, in particular we are interested to trace the sequence of the messages exchanged in the core network when an UE ask for authentication and for establishing a PDU. In particular the system is able to create 2 kinds of data-set, the first one is the simpler one and tracks for each of the UEs executed in the simulation, the amount of time for establish the PDU connection along with the number of messages exchanged. The second type of data-set traces the sequence of messages exchanged for establish the PDU session. In this kind of experiments we perform the simulation with a certain number of UEs (we will see how after) and register all the messages exchanged during the registration procedure to the core network and also the PDU session establishment for each UE.

4.1 Core Network

The core network part is the one in charge of simulating the 5Gcore network and all the components of the SB Architecture such as the AMF, PCF and so on. So most of the traffic created by the simulation is located here because each request issued

by an UE generate many more messages in the core side because the entities of the core have to coordinate in order to fulfil the request, so they send more messages.

Open5Gs

Open5Gs is an open source project that provides both 4g and 5G mobile packet core network functionality for building your own LTE/5G network info-structure. Along with Free5Gs, those are the only open source implementation of the 5G core network. The overall pictures of the info-structure is showed inFigure 4.2

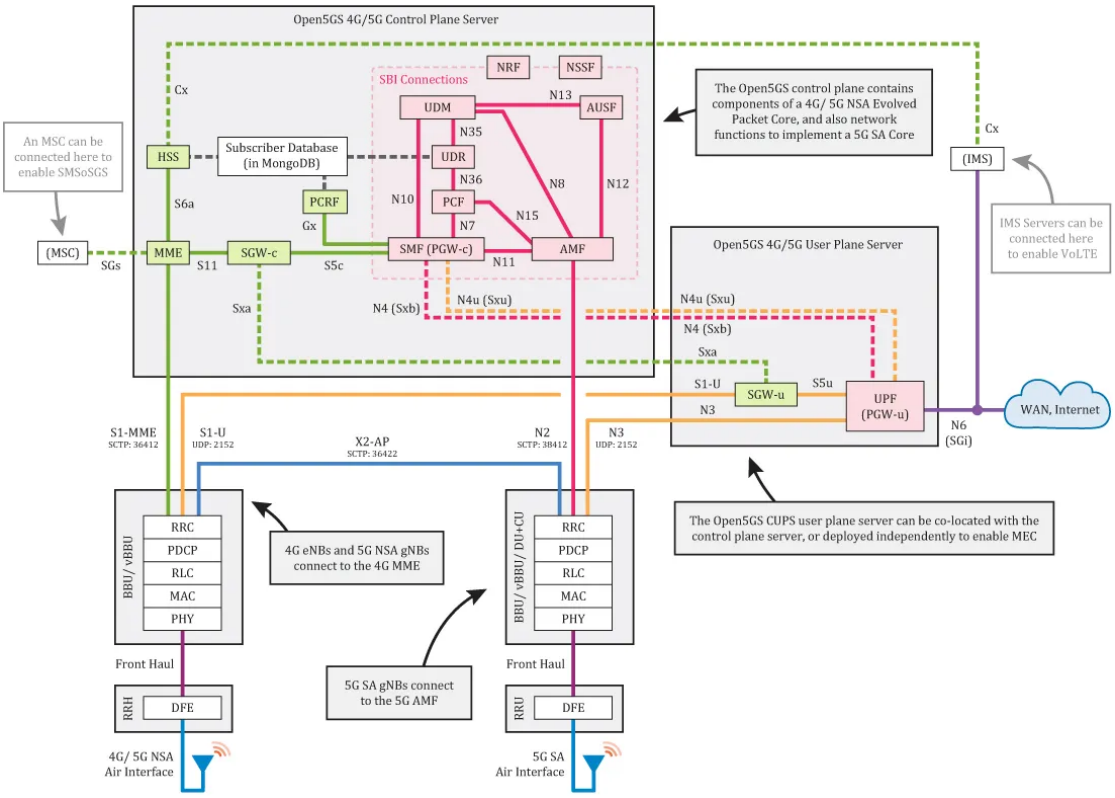

Figure 4.2: Open5Gs architecture [16]

Actually we can distinguish two components of the architecture, the control plane and the user plane. The control plane contains both the 4G-EPC, Evolved Packet Core, so all the functions of the 4G core network and also, mote importantly, the 5G-SA (Stand Alone) Core network. This is the one that we are going to use for our simulation and for tracking and analyzing the traffic generated by the core when the UEs are turned on. Also, in order to better understand what is the part of the core network which we are interested, you should consider only the components interconnected with the interfaces named with an N (so N2,N3,N8 ecc.).

4.1.1 Open5Gs - Database

Another component of the control plane that the simulator needs is the subscriber database, is a MongoDB database used for storing the data of each of the subscribers (so the UE's data), the current open sessions and the accounts.

In *Account* there are the users that can interact with the dashboard, the dashboard can be used for inserting new UEs and let them authenticate to the core network. The *Session* collection is used for storing the data of the UE's session, so all the information related to the current PDU. The last one, the collection *Subscriber*, collects the information related to each of the UEs and those are necessary to let the UE register to the network, instantiate the PDU and use all the services of the core network. The json object below show the information stored for each UE,

```
{ "_id" : "id",
"imsi" : "901700000000020", "__v" : 0,
"access_restriction_data" : 32,
"ambr" : {
    "uplink" : { "value" : 1, "unit" : 3 },
    "downlink" : { "value" : 1, "unit" : 3 } },
"network_access_mode" : 0,
"security" : { "k" : "465B5CE8B199B49FAA5F0A2EE238A6BC",
    "amf" : "8000", "op" : null,
        "opc" : "E8ED289DEBA952E4283B54E88E6183CA",
            "sqn" : NumberLong(40002) },
"slice" : [
            { "sst" : 1, "default_indicator" : true, "_id" : "id",
            "session" : [
                { "name" : "internet",
                "type" : 3,
                "_id" : "..id",
                "pcc_rule" : [ ],
                "ambr" : { "uplink" : { "value" : 1, "unit" : 3 },
                            "downlink" : { "value" : 1, "unit" : 3
   } },
                "qos" : { "index" : 9,
                            "arp" : { "priority_level" : 8,
       "pre_emption_capability" : 1,
                            "pre_emption_vulnerability" : 1 }
                            }
                }
              ]
          }
         ],
```

```
29 "subscribed_rau_tau_timer" : 12,
30 "subscriber_status" : 0
```

The most important ones are:

- **IMSI**, International Mobile Subscriber Identity Number, is the number that characterize a user in the 5G core network and it's unique in the GSM (Global System of Mobile Communication). The identifier is divided in two parts, the first 5/6 digits (depending on the European/American standard) identify the GSM operator's country and the other 9/10 digits are allocated by the network operator for identifying the subscriber inside the network. the format of teh IMSI is *[MCC|MNC|MSISDN]* where:
  - MCC, Mobile Country Code, identify the Country
  - MNC, Mobile National Code, identify the nation
  - MSISDN, this is the part that identify the single user

  This number is also stored in teh SIM card of the UE and so of the user. It's thanks to this number that the components of the 5G core network can recognize and manage the user (example the AMF lookup the data of the subscriber by using this number).
- **AMBR**: The Aggregate Max Bit Rate contains 2 fields related to the uplink and downlink and it gives the information on the guarantee bit-rate both for the downstream (so the download) and the upstream (upload). In fact in subsection 3.1.1 we said that there are two unidirectional tunnels, the purpose of this information is to characterize these two tunnels.
- **Security**: In order to claim the IMSI sent over the air the UE has to prove its identity by using pre-shared secrect. The pre-shared secrets are the Key (k), the operational key (OPc) and the sequence number (SQN). The first two information are pre-shared with the network operator and they are used together in order to obtain, trough the use of a function, the mixed key that is send over the air. Then the core network will mix again the two parameters (k,OPc) and then compare with the one sent by the UE. If they are the same key then the user is authenticated. The problem here is that an attacker can sniff the network and authenticate another sim by simply provided the mixed key (reply attack), in order to avoid such a situation we use a sequence number (SQN) to mix together with the two keys so for every authentication the mixed result is different. The sequence number is stored both in the UE and in the core, and it starts from 0.

- **Slice** In the subscriber collection we have also the corresponding slice to use for a certain UE. The information is stored in the core side and also configured in the sim of the UE. Each slide is identified by the SST Slide/Service Type and reflect the purpose of the slide. Each slide has a related session that specify, along with other things, the data network which the slide is connected (Internet in this case), the set of policy rules and the AMBR, Aggregate Maximum Bit Rate, both for the up-link and for the down-link.

Connecting Core and RAN

In order to reach the network the RAN can use 2 interfaces in the 5G setting, the N2 interface and the N3 interface. The N2 interface is the one used for the connection with the AMF and the protocol used for signaling is NGAP.
NGAP supports the UE related services and operations such as the Initial Ue context setup for the establishment of the context between the UE and the gNodeb. It's also the protocol used by the UE for the setup of the PDU, the modification and the release of the PDU along with the related resources. NGAP is also used for services that are not related to the UE such as the trace of the active devices, emergency services and so on. The protocol also provides the mechanisms required to manage the attachment and the management between the gNodeb and the AMF along with error indication and load balancing. The transport protocol used the transporting the NGAP signals is SCTP, Stream Control Transmission Protocol, is a connection-oriented protocol, full-duplex and support multiple streams of data, it's used for overcome some limitations of TCP. In fact this protocol supports natively multistreaming and multihoming endpoints, along with selective ACKs and for this reason it can address the problem of head-of-blocking that reduce the performance of TCP.

The N3 interface connects the gNodeb with the User Plane Function (UPF), each gNodeb can be connected to one or more UPF. During the PDU Session Establishment Requests the Session Management Function (SMF) selects the UPF for the PDU. Then the connection between the gNodeb and the UPF is based on a UDP tunnel, the GTP-U. This tunnel carries the UE's data into the data network by using the GTP-U protocol and each PDU sessions is related to its own GTP-U tunnel.

Core Server

In order to interact with open5Gs and perform the capture of the traffic for the later analysis the Virtual Machine has a server that executes the commands. The server is based on the Flask framework, it is a lite web-application written in

python that use the WSGI (Web Server Gateway Interface, is a common interface for communication between web-application and web-server), basically is an API server. The server must essentially do three things, firstly execute *Tshark* for a given period of time depending on the number of UEs executed and the way they are executed (more details later), secondly send the capture file to the machine performing the traffic analysis for data-set extraction. Immediately before starting the capture the server have also to restart the open5s services in order to capture the magic numbers exchanged at the beginning of the process, otherwise Wireshark is unable to read the Http2 layer of the packets and the analysis becomes more difficult. The third task it must perform is to write in a file all the sockets belonging to the processes related to open5Gs (i.e., the address where the process listens and the port) so that it can map from the captured traffic the IP and port to the corresponding component. In fact, in this configuration, all components of the open5Gs architecture are processes that communicate with each other using the localhost interface and thus share the address space 127.0.0.1/8. This tasks are performed by the core server by using 2 API endpoints (you can find the code in [17]):

- */getSockets*, this endpoint trigger the command for retrieving the list of sockets related to Open5Gs and in order to obtain such information we use the tool *ss -tpua* and grep the result in order to filter all the output that is not of our interest

- */getCapture/<t>*, this endpoint is used to issue the capture command. The capture command restart the open5Gs process and start the capture process immediately after for t seconds, this value is calculated by the client of the server that will retrieve the file and perform the analysis. After t seconds the file is saved in the *out* directory and then sent to the client.

4.2 RAN - Radio Access Network

The Radio Access Network is the other part of the system and is the one that is in charge of executing the gNodeb and the UEs. The gNodeb is connected, trough the N2 interface to the AMF, and to the PCF with the N3 interface. As we said the N2 interface use SCTP as transport protocol for the traffic, and NGAP as as application protocol. In order to manage this part we use *UERANSIM* as simulator of both the gNodeb and the UEs, and a server for executing commands in the simulator.

4.2.1 UERANSIM

UERANSIM is the only open-source simulator, state-of-art for simulating the UE and the gNodeb, complaint to the 3GPP specification and ready to use in Open5Gs and also Free5Gs. Even if it's not completely developed, some features are still in progress, is more then enough for our needs in terms of simulation capability. It is only available for Linux distributions since Windows does not implement the SCTP protocol at all.

In order to run the software you have to download the git repository and compile it with *make*, along with the installation of the dependencies. After building the project the results are 4 executable, namely: *nrue, nrgnb, nrbinder* and *libdevbnd.so*. The first one, *nrue* is the executable used for simulating the UE, it uses a tunnel interface that is basically a virtual device (mounted in */dev/* at which the process is attached and use it for transmitting the traffic. Each nrue process executed has a configuration file, in this file there are the information that the UE needs in order to authenticate, instantiate the PDU session and so exchanging the traffic with the Data Network which the PDU is attached. The most interesting part of the configuration file is listed below:

```
1
2  # IMSI number of the UE. IMSI = [MCC|MNC|MSISDN] (
       In total 15 digits)
3  supi: 'IMSI'
4  # Mobile Country Code value of HPLMN
5  mcc: '901'
6  # Mobile Network Code value of HPLMN (2 or 3 digits
       )
7  mnc: '70'
8
9  # Permanent subscription key
10 key: '465B5CE8B199B49FAA5F0A2EE238A6BC'
11 # Operator code (OP or OPC) of the UE
12 op: 'E8ED289DEBA952E4283B54E88E6183CA'
13 # This value specifies the OP type and it can be
       either 'OP' or 'OPC'
14 opType: 'OPC'
15 # Authentication Management Field (AMF) value
16 amf: '8000'
17
18 # List of gNB IP addresses for Radio Link
       Simulation
19 gnbSearchList:
```

```
20    - GNB_IP:PORT
21
22  # Initial PDU sessions to be established
23  sessions:
24    - type: 'IPv4'
25      apn: 'internet'
26      slice:
27        sst: 1
28
29  # Configured NSSAI for this UE by HPLMN
30  configured-nssai:
31    - sst: 1
32
33  # Default Configured NSSAI for this UE
34  default-nssai:
35    - sst: 1
36      sd: 1
```

The configuration file for the UE can be imaged as the SIM of the physical device that is going to connect to the core network. Here we have the IMSI in the format specified in the core network session section 4.1.1, the MCC and the MNC that must match the one of the IMSI. Also there are the permanent subscription key and the Operational key, the ones used for the authentication of the UE to the core network, in this case the sequence number always from 0 since the UE does not have any state along with different execution. The *gnbSearchList* is used by the UE in order to discover the available gNodeb by using the IP address. Then we have the information for the establishment of the PDU, in particular we have the type, the apn (so the data network name) and the slice in which the PDU will be established. Along with that there is also the configured slide for that UE, in this case only the default one, the same configured in the database.

The other executable provided by UERANSIM is the gNodeb, so *nr-gnb*, so the antenna. It is the one that start the connection with the AMF and maintains the tunnels to the UPF, in this implementation there is only one UPF but there could be more then one in general and in particular in a real scenario. All the UEs trough the TUN interface forward the data to the gNodeb that communicates with the AMF, is also responsible of dealing with the procedures that don't involve the core network but only the gNodeb and the UE. Also the gNodeb has a configuration file listed below:

```
1  mcc: '901'            # Mobile Country Code value
```

```
2  mnc: '70'              # Mobile Network Code value
3
4  nci: '0x000000010'     # NR Cell Identity (36-bit)
5  idLength: 32           # NR gNB ID length in bits
       [22...32]
6  tac: TAC             # Tracking Area Code
7
8  linkIp: GNB_IP    # gNB's local IP address for Radio
       Link Simulation
9  ngapIp: GNB_IP    # gNB's local IP address for N2
       Interface
10 gtpIp: GNB_IP     # gNB's local IP address for N3
       Interface
11
12 # List of AMF address information
13 amfConfigs:
14   - address: IP_CORE_MACHINE
15     port: 38412 # default
16
17 # List of supported S-NSSAIs by this gNB
18 slices:
19   - sst: 1
```

We can see that also the gNodeb has a mcc and the mnc since also the antenna belongs to a provider and so it must be classified with the mobile country code and mobile network code. Along with this there is also the NCI (NR Cell Identity) that is part of another code used for identifying the cell globally, in particular this code is used with the PLMN ID (Public Land Management Network) in order to create the NR Cell Global Identifier (NCGI), by the way in our context is not meaningful. Then we have the Tracking Area Code (TAC) and the 3 addresses for the link-ip, ngap tunnel and the gtp tunnel (namely *linkIp*, *ngapIp*,*gtpIp*, those are used for performing the connections with the N2 and the N3 interface and to specify the address of the Radio Link Simulator. Then we have *amfConfigs* that is the list of the AMF, in this case since we have only one AMF we have only one entry with the IP address of the core network machine along with the port.

The last two executable provided by UERANSIM are *nr-binder* and *libdevbnd.so*, they are used for binding the UE interface to an arbitrary internet application such as a browser or to a command like *ping*, in this way the UE can communicate to the Data Network which it is attached and so exchanging traffic and since the Data

Network of default is internet (the core network has the NAT configured), it can exchange traffic with arbitrary host on the internet. For example is possible to run a browser such as Firefox and bind the application to the UE by using *nr-binder*.

4.2.2 RAN server

The RAN server is in charge of managing the gNodeb connection to the core network and the execution of the UEs. In particular we need a system that is able to run an arbitrary number of UEs in a certain patter. In particular the system is able to perform two kind of experiments, the first one is with only 1 gNodeb and arbitrary number of UEs, the second one is able to run an arbitrary number of gNodeb and arbitrary number of UEs for each of the gNodeb. Like the core server, the RAN server is a Flask application that expose APIs for the iteration with the RAN simulator, now we are going to explain the different API that control the different settings.

Setting 1

In this setting we have that the gNodeb is deployed directly in the RAN virtual machine, and so the gNodeb is directly attached to the virtual NIC of the VM. Since the gNodeb is attached to the NIC of the VM there could be only one gNodeb per machine in this setting, so all the UEs will be connected to the same gNodeb. The UEs are executed with a certain pattern that try to simulate a certain scenario, it could be one shot of n UEs all together at the same time or it is possible to specify the number of blocks to execute, and n is the size of the block. Along with the number of blocks and the number of UEs per blocks the third parameter is the interval between the blocks, so every time a block of UEs is executed the system sleep for t seconds and then proceed with the next one. In order to perform such experiment we can use the endpoint */runLocal/<n_blocks>/<n_ue>/<t>/<type_run*, the parameters are:

- *n_blocks* is the number of blocks executed in the simulation
- *n_ue* is the number of Ues per block
- t the interval in second between each blocks
- *type_run* define the way in which the block is filled, actually there are 2 type of run. The first run is when the size of the block reflect exactly the n_ue specified, in the other type of run we have that the size of the blocks reflect a Poisson distribution and so the n_blocks is the number of occurrences and n_ue is the mean value. In this case so we have that the size of each of the block depends on the Poisson distribution.

The resulting topology of such a setting is depicted in Figure 4.3.

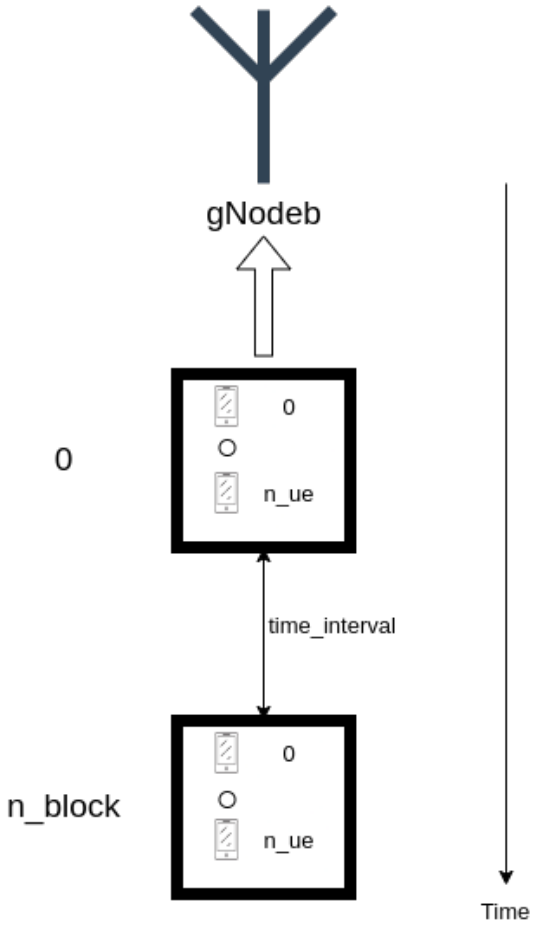

Figure 4.3: Topology of Setting 1

In order to execute the experiments the server first run the gNodeb (so nr-gnb with the configuration file above) and then with a *.sh* file called RunUes.sh for the first type of run or RunUes_poisson.sh in case of the second type it's able to run the UEs with the specified pattern. Each block correspond to run the executable *nr-ue* with the option *-n* equal to the number of UEs, along with this we have to specify the configuration file for that run. Same for the next block but since all the UEs are identified by the IMSI code stored in the configuration file we have to increment that number up to the last run +1 in a way that all the codes are aligned and there is no conflict between subsequent UEs blocks (so there is no UE in two different block that has the same IMSI). The code that perform the execution of the UEs can be found in the repository [17].

So from a topological prospective this setting is a little bit limited by the fact that there could be only one gNodeb and arbitrary number of UEs, actually the number of UEs is limited by the resource assigned by the hypervisor to the machine, in fact sometimes we can experiment some peck of usage when the dimension (in terms of UEs) is big.

Setting 2

Since we would like to overcome the limitation given by the unique gNodeb the second setting is build with the idea of creating more complex topology by deploying more then a single gNodeb. In order to deploy more then antennas we need an environment that provide isolation between each gNodeb. A similar setting can be created by containerize the gNodeb and the Ues, and so by using the Docker's containers we are able to effectively create a topology like the one in Figure 4.4.

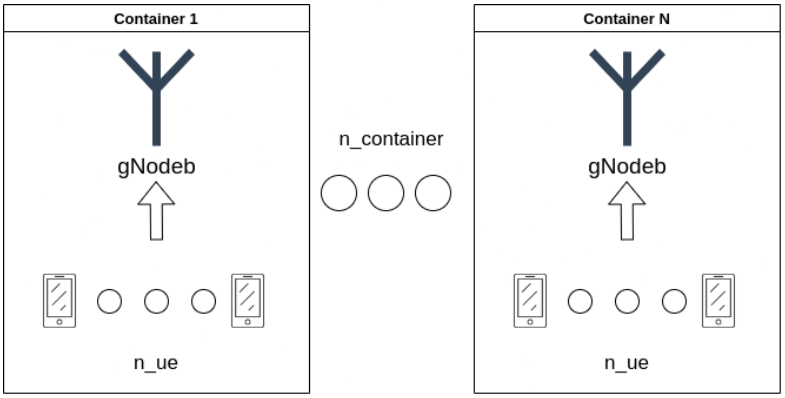

Figure 4.4: Topology of Setting 2

In order to use this setting there is an API endpoint in the RAN server that is */startContainer/<n_container>/<n_ue>* where:

- *n_container* is the number of container that we intend to run
- *n_ue* is the number of UEs per container

In this case there is only one type of run mode, so the same number of UEs in each container and they all start at the same time. In the container case the gNodeb is binder to the network interface of the container and so it's possible, thanks to the isolation between the containers, to deploy more then one gNodeb in the same machine. Obviously the number of containers that you can run depends on the resources assigned to the virtual machine, but in this way we have two degree

of freedom, on the number of UEs per container and the number of containers. The code for executing UERANSIM inside the container and the code to execute the containers can be found in the repository [17]. This other setting provide the simulator of the capability of testing different topology and so to cover a wider range of possibility in which the UEs can be located in different areas and so attached to different gNodeb and as, in a real mobile network, all the gNodeb are attached to the same core network, in this case in particular they are connected to the same AMF but in a real 5G deployment there are more then one AMF and so different gNodeb can be connected to different AMFs.

4.3 Client - Simulation and Analysis

The third virtual machine is the client of the core server and the ran server and it could be seen as a command-and-control machine that executes the gNodeb and the UEs and at the same time start the capture process and retrieve the file. Along with this there is also the analysis part performed on the capture file after each run in order to produce the two data-sets.

4.3.1 Client - Simulation Controller

In order to run the simulator we create a configuration file, the file is a *.json* file and depends on the setting you want to use. The first type of setting has a configuration file like:

```
1    {
2      "mod": 1,
3      "n_blocks": 1,
4      "n_ue": 150,
5      "time_interval": 1,
6      "type_run": 0,
7      "n_runs": 5
8   }
```

mod is used for specify the setting, in this case it is the setting with only one gNodeb, *n_blocks* is the number of blocks to run along with *n_ue* the number of UEs per block, *time_interval* specify the time in second between each block, *type_run* specify how to manage the blocks, and $= 0$ in this case specify to use the exact number of UEs per block, $= 1$ means that we want the size of the blocks to be a Poisson distribution in terms of number of UE per block and last *n_runs* is the number of times to repeat the same experiment. So the client by taking the configuration file is able to run this simulation with this setting *n_runs* times and for each run it perform the analysis.

```
{
  "mod":0,
  "n_containers":5,
  "n_ue":10,
  "n_runs":1
 }
```

Here instead we have the entry of the configuration file in case of setting 2, here we specify the *n_containers* that we want to run, so the number of antenna, the *n_ue* for each container and again *n_runs* so the number of times to repeat the simulation. In general a configuration file can be mixed, so there could be entries related to the two different setting all together in order to facilitate the simulation. The goal of the configuration file is to be able to test a space of simulation by tuning the different parameters (number of blocks, number of UEs and time interval). In this way it is possible to cover a simulation space able to capture the features of the system and so be able to create a model that can effectively predict how the network react in case of a certain load in input, in terms of number of UEs, gNodeb and they way in which they are executed.

The simulation controller use two threads for controlling the core server and the ran server. the core thread use the endpoints */getSockets* and */getCapture/<t>* in order to retrieve the capture file and the socket file, the time t for the capture is calculated, based on empirical experiments, as:

```
    t = min(round((((n_ue) * (n_container)) * 3) + 10),240) # secs
```

In this way the ran server has the time for running all the blocks and so all the UE's packets exchange can be captured by the capturing process, there is an upper bound of 4 minutes that is the maximum time the simulator can spend in running all the simulation.

The ran thread call the */runLocal/<n_blocks>/<n_ue>/<t>/<type_run* or */startContainer/<n_container>/<n_ue>* based on the configuration file and wait in a synchronous way the end of the simulation and then retrieve the capture. The core thread is executed 15 seconds before the ran thread since the core needs to be reloaded in order to make the capture effectively.

4.3.2 Analysis

After the end of the core and RAN threads the analysis starts and the method *run_simulation* is executed automatically and so the post-processing of the data.

Here we have the logic of the analysis that produce the two data-sets, the two procedures are performed by using different methods, here we explain the relevant part of the process.

Dataset 1

The first type of analysis is intended to extract the time that an UE needs to connect to the core network, so first we have to identify which is the first and the last message of the message sequence used by the UE for the connection, the entire sequence of message flows in the N2 interface, the one that connects the gNodeb to the AMF (Access Mobility and management Function).

The first message sent by the UE to the core network is a NGAP/NAS-5S message of name *InitialUeMessage*, in this message there are all the parameters of the UE and of the piece of network where the UE is, such as the PLMN (Public Landlord Management Network), the TAI (Track Area Id). Also in this message is specified the type of registration that the UE is intended to perform, the type of identification that the UE will provide so the format of the SUPI that is the IMSI, the one specified in the configuration file of the UE, along with the MCC MCN that represent all together the 5GS mobile identity. Along with this information there is also the security capability of the UE, so basically the supported encryption protocols. This message is followed by a sequence of other messages in the RAN side and in the core side of the network, but the former will be analyzed in the next type of analysis. In this analysis we are interested in the first and the last message and the time in which they are arrived. The last message that close the sequence and after which the UE is registered to the network with an instantiated PDU is *PDUSessionResourceSetupResponse*, this message contains, along with other information, the data related to the PDU session, and so the information of the GTP-U tunnel, its address,the gtp-TEID (Transport Endpoint ID), and the qosFlowIdentifier that identify the flow inside the PDU.

The first kind of dataset is produced by using the method *ExtractTimeUEsConnection()* listed below:

```
1  def extractTimeUEsConnection(cap,middle_name):
2
3      captured = dict()
4      for packet in cap:
5          handlePkt(packet)
6
7      reg_time = calculate()
8
9      ran_time(reg_time,middle_name)
10
```

```
11      return 0
```

The parameters of the function are *cap*, so the capture object taken from the file and *middle_name*, part of the name of the result file. The method *handlePkt()* is in charge of finding the sequence of messages related to the UE among all the other messages inside the capture file and split all the sequence per UE.

In order to split the traffic per UE we have to find a parameter that is unique for the UE, this is *RAN_UE_NGAP_ID*. This ID is unique among all the UEs, the antenna and the AMF. This code is incremented for each UE starting from 1 up to the number of UE executed in a session of simulation. So this method handles the single NGAP packet and creates a summary of the packet (by using the method *CreatePacket()*) and then it puts the packet in a map in which the key is *RAN_UE_NGAP_ID*, and the value associated to the key is the list of messages (only the NGAP message type) exchanged during the procedure of registration and instantiate of the PDU. The code for the method is listed below:

```
1  def handlePkt(pkt):
2
3      if("NGAP" in pkt):
4          for p in pkt.get_multiple_layers('ngap'):
5              ngap_packet = CreatePacket(p,timestamp = pkt.
       sniff_time,ip_src = pkt.ip.src) # create a summary of the
       packet
6              if(ngap_packet is None):
7                  break
8              captured.setdefault(ngap_packet.ran_ue_ngap_id,[]).
       append(ngap_packet) # list of message per UE
9              if(ngap_packet.ran_ue_ngap_id not in UEs): UEs.append(
       ngap_packet.ran_ue_ngap_id) # list of UE
```

After this procedure we have the dictionary *captured* in which we have all the flows of messages exchanged, so we have to select only the *InitialUeMessage* and the *PDUSessionResourceSetupResponse*, so by iterating the keys of the dictionary and selecting the two message the method produces another list called *reg_time* that contains only the *RAN_UE_NGAP_ID* (for identifying the UE), the sniff time of the *InitialUeMessage* and *PDUSessionResourceSetupResponse* along with the number of packets captured for each UE. Then the method *ran_time* will write this series of time in a *.csv* file, a sample of such dataset:

UE	Connection Time (s)	NumberOfPackets
1	0.09754800	7
2	0.104737	7
3	0.320815	7
4	0.336651	7
5	0.521779	7

Dataset 2

The second type of data-set is intended to capture the messages exchanged inside the components of the core when a UE issue a registration request and ask for a PDU Session. In order to capture this sequence of messages for an arbitrary number of UEs we have to first understand what is the sequence for one UE and then generalize it. For this purpose a capture was performed with a single UE, in order to trace the sequence of messages triggered by the UE during the procedure of registration and PDU creation. The sequence is showed in Figure 4.5.

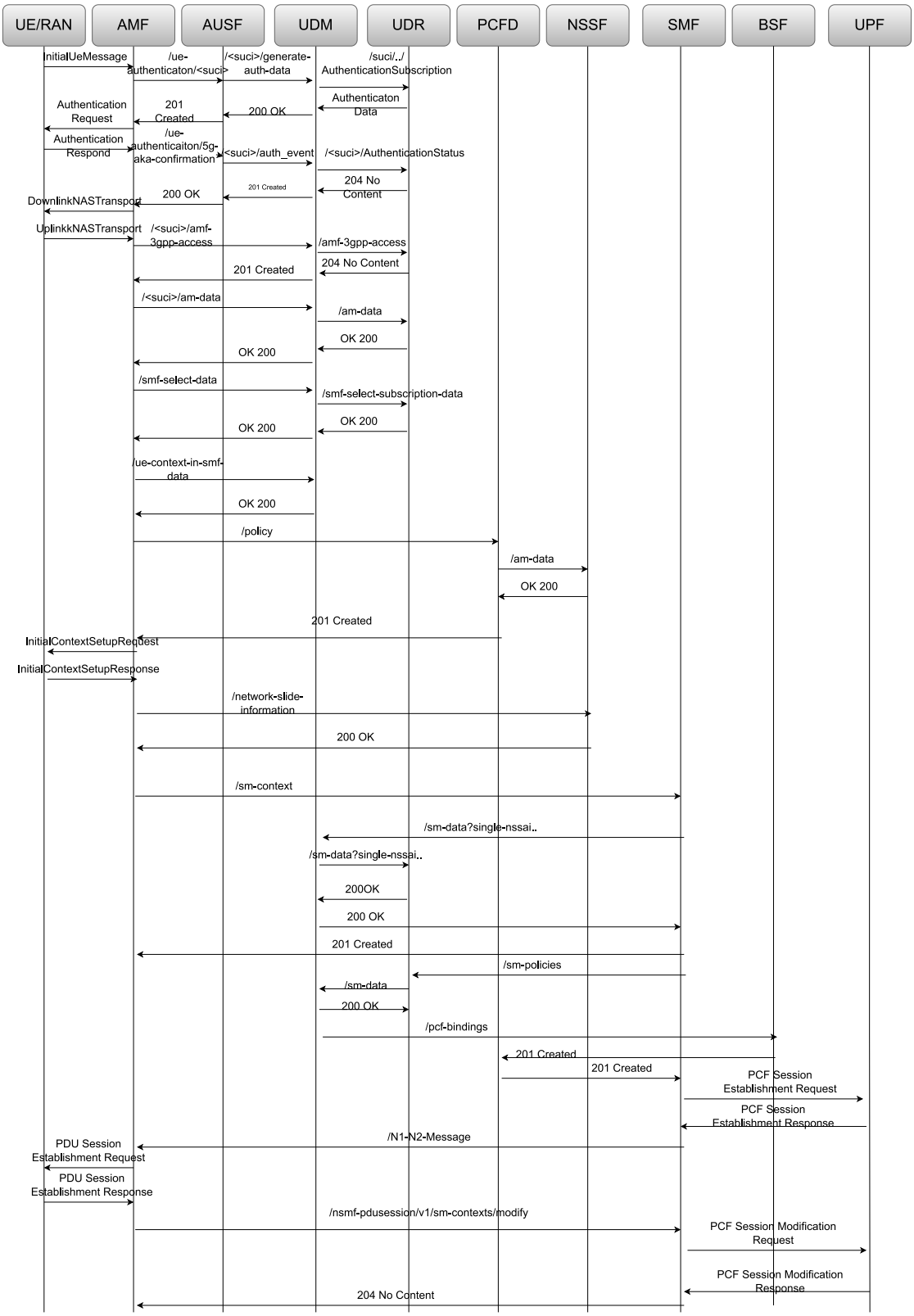

Figure 4.5: Sequence of messages created by the UE that connects to the core network

This sequence of messages is repeated exactly for all the UEs that issue a registration request plus PDU creation, the only exception is for the Http2 request *GET /nnssf-nsselection/* that is issued only one time by the AMF to the NSS by passing the id of the slice which we want the information. In general it could be issued for each of the requested slice, but since here the UEs use only one slice the request is issued only one time. Since this request is a one-time request (and also the response) is it not in our interest to capture it and so it is not included in the data-set generated.

In order to capture such a sequence for an arbitrary number of UEs we need a method able to parse each of the packet and record the sniff time. This method should be able to collects all the times related to a certain type of messages, both for the requests and the responses and in the order in which they are arrived. This is not related to the particular UE, in fact the produced dataset carry a list of all the different messages but without tracing the particular UE, this is because sometimes is not possible to do. For example there are responses that does not carry any information about the particular UE and this is the same for some other request. So the dataset only represent the time which the messages has been recorded into the capture file, independently from the UE. This has been done in order to be able to predict how long each message of the sequence takes for going from a component to another one. By recording the time intervals we are able to model how the single component act and the time that it needs in order to produce the result for a certain request. Consider also that when a component issue a request to another component, the destination component could also issue other requests to other components and so a request can create a chain of requests toward other components.

In order to perform this analysis we are using the method sequence_pdu_establishment, this method call the method *find_start* that find the first message of the first UE that perform the connection. The message that we are looking for here is the first *InitialUeMessage* of the capture file, after that message the capture contains all the messages that we need in order to perform the analysis. Once the starting point for the analysis is found we have to trace all the sequence of packets, here the protocols which we are interested to analyze are 3, Http2, PFCP and NGAP (SCTP). Http2, as mentioned, is used between the components of the core network for the communication and its part of the SB Architecture (SBA). PFCP instead is the Packet Forwarding Control Protocol and is in the N4 interface for the communication between the Control Plane and the User Plane Function (UPF) and it is the protocol that permits the separation of the User Plan with the Control plane. We are also interested in NGAP protocol but only the first message (InitialUeMessage), the reason is explained in the result

section. The code to perform the analysis is listed below:

```
def sequence_pdu_establishment(cap,addr_name):
    s = findStart(cap)
    cap = list(cap)[s:]
    msg_time = dict()
    for pkt in cap:
        sniff_time = pkt.sniff_timestamp
        if("HTTP2" in pkt):
            if("HEADERS" in pkt.http2.stream):
                p = pkt.http2.stream
                method = str(p).split(",")
                method = method[3]
                src = f"{pkt.ip.src}:{pkt.tcp.srcport}"
                dst = f"{pkt.ip.dst}:{pkt.tcp.dstport}"
                c_dst = f"{addr_name.get(dst)}" if addr_name.get(dst) != None else dst # lookup component name
                c_src = f"{addr_name.get(src)}" if addr_name.get(src) != None else src
                p = handle_header_http2(p,pkt) # manage the single header
                k = f"{c_src}:{c_dst}:{p}"
                msg_time.setdefault(k,[]).append((float(sniff_time)))

        if("PFCP" in pkt):
            [pfcp] = pkt.get_multiple_layers("pfcp")
            if(pfcp.msg_type != "1" and pfcp.msg_type != "2"):

                    src = f"{pkt.ip.src}:{pkt.udp.srcport}"
                    dst = f"{pkt.ip.dst}:{pkt.udp.dstport}"
                    c_dst = f"{addr_name.get(dst)}" if addr_name.get(dst) != None else dst # lookup the component name
                    c_src = f"{addr_name.get(src)}" if addr_name.get(src) != None else src

                    method = get_pfcp_procedure(pfcp.msg_type) # method
                    k = f"{c_src}:{c_dst}:{method}" # key for the dict
                    msg_time.setdefault(k,[]).append((sniff_time))
        if("SCTP" in pkt):
            src = "UE"
            dst = "AMF"
            c_dst = f"{addr_name.get(dst)}" if addr_name.get(dst) != None else dst
            for p in pkt.get_multiple_layers("ngap"):
                if(int(p.get_field_value("procedurecode")) == 15):
                    k = f"{src}:{dst}:InitialUeMessage"
                    msg_time.setdefault(k,[]).append(sniff_time)
```

```
40      return msg_time
```

For the Http2 protocol we are interested only in the first packet that carries the the request or response (the Header one), in fact the payload could be big and so the message could also be separated in more then one message since it could not fit in only one. The method *handle_header_http2* handles the single header, in fact sometimes it carries information related to the UE that issue the request and it must therefore be anatomized so that it can be inserted together with other messages of the same type, sometimes this is not necessary but sometimes is it. The obtained dictionary *msg_time* carries a set of keys that represent the sequence of message in the format *source_component:destination_component:message_name* for all the 3 protocols. Each key points to a list of that contains the sniffed time of that time of message. By doing this we are able to trace all the messages and the arrival time of an arbitrary number of UEs, in both settings of the simulator, in local mode, and the containerized mode.

Chapter 5

Results

This section considers the data-sets that were produced during the simulation, the simulation space tested, i.e. how the simulation parameters were adjusted and possible correlations between the data.

5.1 Datasets

The simulation was parameterised in order to be able to create different scenarios and test various use cases, various load levels and how the network load can be distributed. The first parameter we can analyse is the number of UEs per block, i.e. the number of UEs that at a certain instant register on the network and instantiate a PDU. This parameter is equivalent to turning on a number of devices, e.g. mobile phones, all at the same time and having them register to the network. This parameter relates to the instantaneous load on the network, in fact the traffic that these UEs generate by connecting arrives all at once to both the gNodeb and the core network.

The second parameter to take into account is the number of device blocks that are executed. This parameter serves to distribute the load of the devices in the network and regulates, together with the number of UEs per block, the total number of devices that are executed in a simulation.

The third parameter is the time distance, in seconds, between one block of UEs and the next. This parameter serves in a certain way to regulate congestion in the core, since if this time is too short, the core has less time to dispose of the traffic, hence the workload, due to the previous block. So by tuning these 3 parameters the simulation covers a space of parameters that will be used to generate the dataset. The Figure 5.1 is a graphical representation of the simulation space used for generating the data.

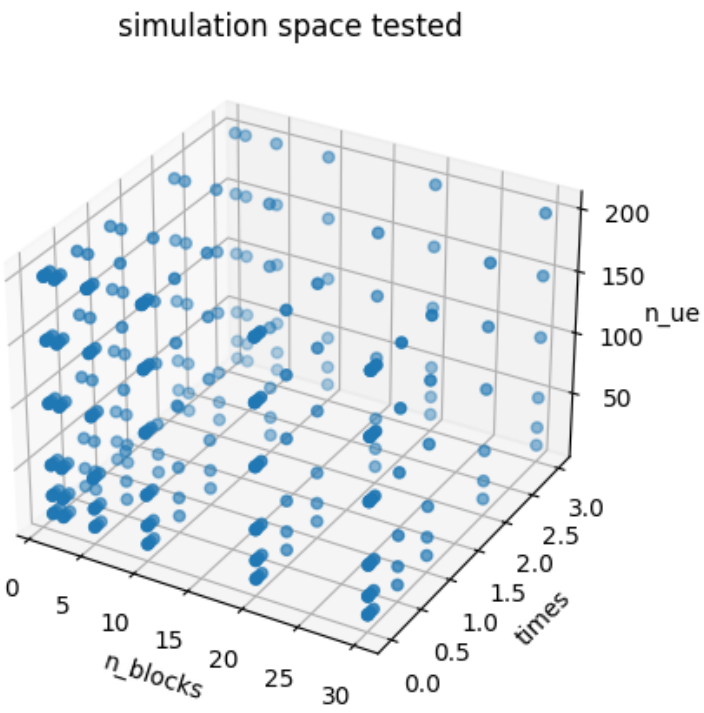

Figure 5.1: Simulation space tested for Setting1

These 3 parameters are the features of the model that we want to build and so the simulation space represents the manifold of the model. Indeed without these parameters, the predictive ability of the model would be very limited if not almost nil and the model would lose its meaning. This is the simulation space tested for the first setting. The second setting tested has only 2 free parameters, the number of container and the number of UEs per container, each container contains a gNodeb as showed in Figure 4.4. In this case the simulation space tested look like Figure 5.2

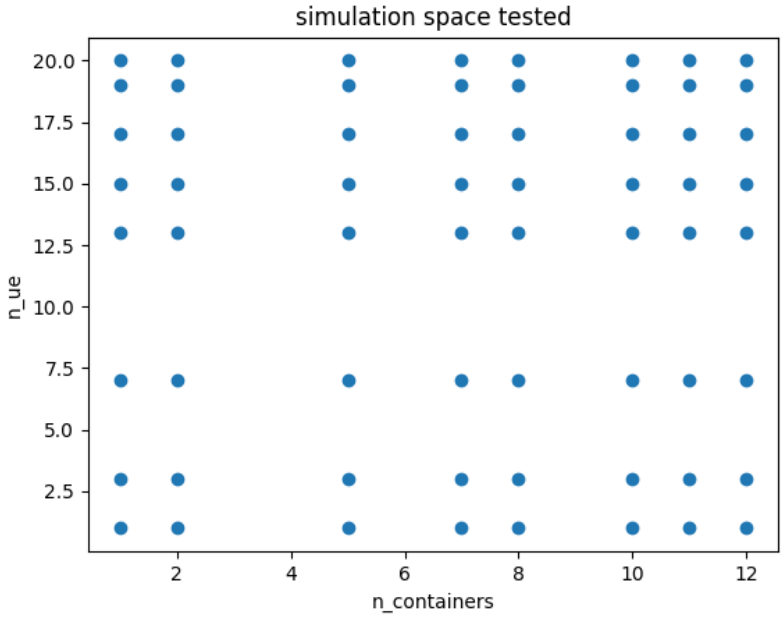

Figure 5.2: Simulation space tested for Setting 2

5.2 Evaluation

The data extracted are aggregated data relating to the network traffic that UEs generate at the time of connection. The idea is to work with a simplified version of the network, in fact at each simulation the core is restarted and thus the previous state of the network is reset, so we do not consider any intermediate state of the network. Data were processed by taking 3 runs of the same configuration and then averaged to avoid outliers and to average between runs.

From the data, we expect to be able to observe a correlation between the total number of UEs executed and the average connection time.

5.2.1 Setting #1

Figure 5.3 shows the average connection time depending on the number of UEs and for different time intervals between the UEs. We can see that the connection time increases when we add more UEs. In addition, increasing the time distance between two blocks is somewhat like adding a delay in the startup of UEs. This correlation is reflected in the data with a higher average connection time. This is evident when the time distance between blocks is 3 seconds, and looking at the graph in Figure 5.3 we can see how the highest average connection time is given

by the 3-second interval. Note that each point corresponds to a single experiment (so the data extracted by a single simulation) performed with a different number of UEs per block, the columns represent the total number of UEs executed in the simulation.

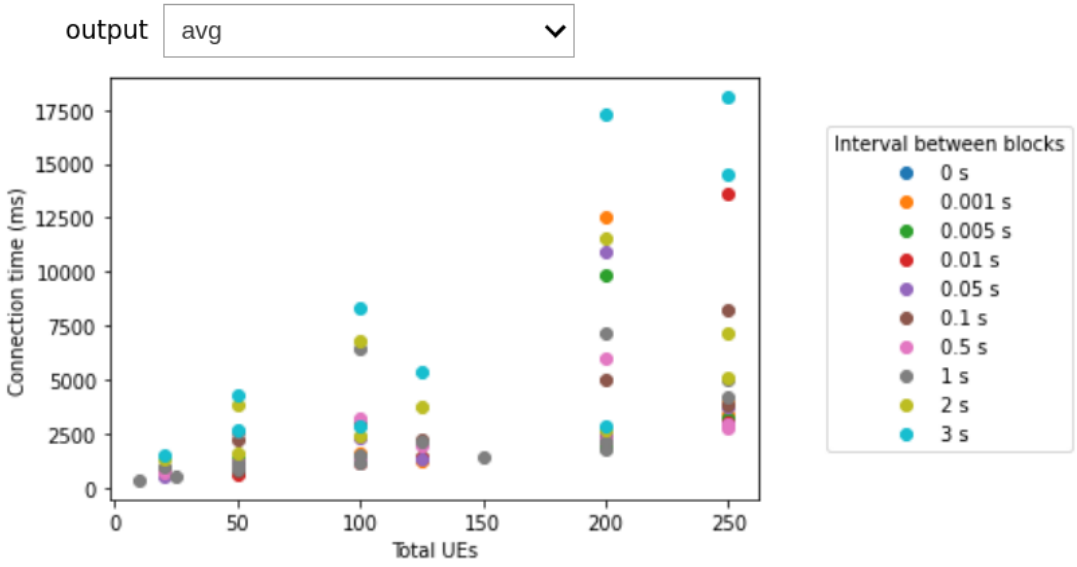

Figure 5.3: Average connection time for setting 1

We can observe a strong correlation between the average connection time and the number of UEs executed during the experiment

We can see, however, that when the interval between blocks is below 1 second this correlation is lost as the number of UEs increases. In fact, the average connection time increases as the number of UEs increases when the interval between blocks decreases. The explanation is that when this interval is lowered the blocks follow each other faster and thus the core is forced to dispose of the same amount of traffic but in less time and thus congestion and thus transmission delays occur. This effect is seen well when the number of total UEs is high (200 to 250) but from below 150 this phenomenon no longer occurs because in any case the core is able to dispose of the traffic if reduced and therefore the time component of the distance between blocks returns to dominate the average connection time.

The graph in Figure 5.4 shows the average connection time for a fixed block of UEs (10 per block) and increasing number of blocks. The same trend is also reflected when we consider the average connection time versus the number of blocks. In fact, the graph in Figure 5.4 shows how the same trend emerges even if we fix the number of UEs per block. We can see that for intervals between block

higher that 1 second, there is a correlation between the number of blocks and the connection time, e.g. for intervals of 3 seconds (light blue dots). However, for block intervals below 1 second, this correlation is lost, for example, the connection time for intervals of 0.001 seconds increases significantly for 20 blocks. Again, we owe this behavior to congestion that increases the total connection time.

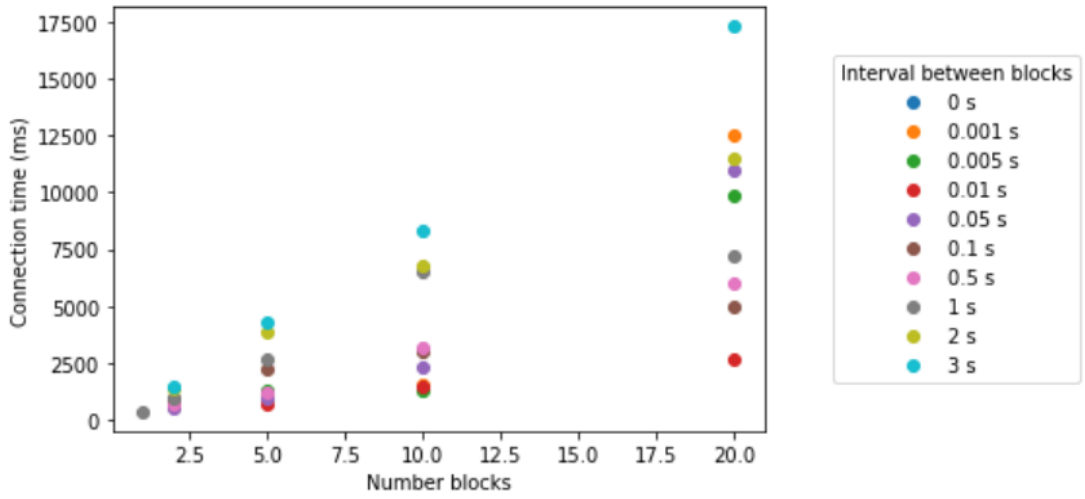

Figure 5.4: Average connection time for setting 1 and 10 UEs per block

5.2.2 Setting #2

On the other hand, as for the data generated with the second mode (setting 2), we have similar trends. In Figure 5.5 we show how the number of containers impact the average connection time of the UEs. In fact from as can be seen from the graph in Figure 5.5 the average connection time for a UE increases as the total number of UEs executed increases. As already mentioned each container executes a certain number of UEs and a gNodeb. The trend we had in the other experiments is preserved here as well, in fact for the same number of UEs by increasing the number of containers we have an increasing average connection time that confirms what we expected. One note to make here is that the number of UEs per block however does not have a clear impact on the average connection time. In fact sometimes a lower number of UEs per block still results in a longer connection time than others that have more UEs. We can clearly see this in the experiment with 5 containers: the connection time is higher for 17 UEs that for 19.

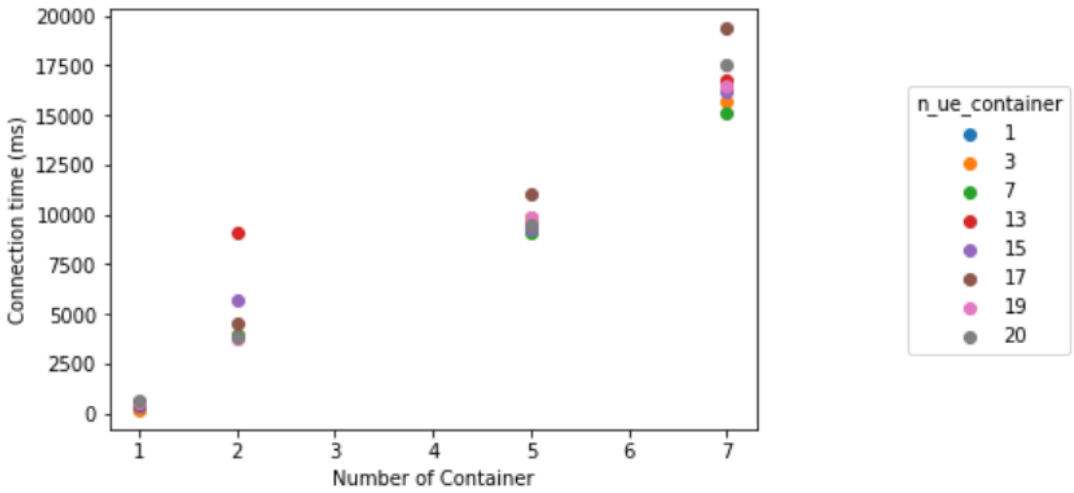

Figure 5.5: Average connection time for setting 2

Chapter 6

Conclusion

The system obtained at the end of this work succeeds in simulating likely scenarios for a mobile network in which various UEs connect to one or more antennas at different times and register to the network in order to be able to surf the Internet. At this point, we can say that the goal of the thesis has been achieved, the obtained system is able to automatically simulate various scenarios and collect data accurately, a testbed for the 5G scenario.

Regardless of the limitations of the developed system, which is easily extendable, the data obtained are relevant and confirm the initial expectations. Obviously this is not sufficient to realize a Digital Twin, which is the problem from which we started. However, it represents a first step, a piece of a much larger and more complex puzzle in which a huge number of scenarios, situations and conditions intertwine.

6.1 Limitation of the current system

The limitations of the current system concern not only the resources involved in its execution, but also the design and features themselves. The first limitation is the core, in fact in this setting the components of the architecture are processes that run in the same virtual machine and therefore compete for the same set of resources. This can be a limitation in the way in which the actual system works. An improvement that has been made is to assign more threads to the core server machine in a way that the CPU's cores where dedicated to it but still we have to take this into account as a possible bottleneck of the network.

In addition, this fact does not allow the core to scale horizontally. In fact some components (such as the AMF) do most of the work and, therefore, it would be better to replicate these processes in order to be able to serve requests, e.g., using more AMFs. In this way, we could increase the total number of antennas connected

at the same time and, also, the number of UEs the core can handle before crashing or having congestion that results in abnormal and meaningless connection times.

Another limitation of the simulation is that the three virtual machines that make up the system are co-located in the same physical machine and thus, again, the three are competing for the same set of resources.

As far as the simulator itself is concerned, one limitation is that of simulation time. In fact, between simulation and analysis of the capture file, the elapsed time can be up to 10 minutes for larger samples. This limits the simulation space in the sense that if you want to simulate a larger region than the one shown in Figure 5.1, the time to do so can become a limitation because it could take days. This also impacts the granularity of the simulation, in fact if one wanted to repeat the simulation starting with a *time_interval* of 0.001 to 3 seconds with a step size of 0.1 it would take a long time. This problem is common for every simulation framework in which we have to wait the natural simulation time, and sometimes it cannot be sped up.

Another limitation of the system is the network topologies that can be created, in fact in the second setting, the antennas are not connected to each other whereas in a real 5G network they theoretically are, this being a matter of host mobility.

6.1.1 Future Work and Improvements

Therefore, given the limitations mentioned above, we can think of several possible improvements to be made in order to make the system more effective, efficient and richer in features, and thus with the possibility of generating better data and models.

As far as the execution of UEs is concerned, an improvement could be achieved by expanding the number of possible distributions of UEs in the blocks and not limiting it to Poisson alone. Also with regard to UEs, given that the 5G network allows more than one PDU to be instantiated at a time for each UE, an enrichment could be to allow UEs to create more than one PDU at the same time and not just the default one.

In addition, in order to speed up the simulation, it would be possible to run the simulations one-by-one and perform the analysis task in parallel to the consecutive simulation. In this way we can run another simulation and in the meantime complete the previous analysis.

The last improvement worth mentioning is that of setting 2. In this setting, all containers are executed at the same time and, therefore, the free parameter *n_interval* is not used. Adding this parameter also to containers would improve the simulation implementation.

Bibliography

[1] *5G Requirements and Key Performance Indicators*. URL: https://www.riverpublishers.com/journal/journal_articles/RP_Journal_2245-800X_612.pdf (cit. on p. 1).

[2] *Source of the image*. URL: https://orm-chimera-prod.s3.amazonaws.com/1230000000545/images/hpbn_0710.png (cit. on p. 2).

[3] *Industry 4.0 and the digital twin*. URL: https://www2.deloitte.com/us/en/insights/focus/industry-4-0/digital-twin-technology-smart-factory.html (cit. on p. 3).

[4] *Digital Twin exchange from IBM*. URL: https://digitaltwinexchange.ibm.com/ (cit. on p. 3).

[5] Cheng Zhou, Hongwei Yang, Xiaodong Duan, Diego Lopez, Antonio Pastor, Qin Wu, Mohamed Boucadair, and Christian Jacquenet. *Digital Twin Network: Concepts and Reference Architecture*. Internet-Draft draft-irtf-nmrg-network-digital-twin-arch-00. Work in Progress. Internet Engineering Task Force, Mar. 2022. 24 pp. URL: https://datatracker.ietf.org/doc/html/draft-irtf-nmrg-network-digital-twin-arch-00 (cit. on p. 3).

[6] *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems* (cit. on p. 5).

[7] X. Duan C. Zhou H. Yang. *Concepts of Digital Twin Network draft-zhou-nmrg-digitaltwin-network-concepts-03* (cit. on p. 5).

[8] Paul Almasan et al. *Digital Twin Network: Opportunities and Challenges*. 2022. DOI: 10.48550/ARXIV.2201.01144. URL: https://arxiv.org/abs/2201.01144 (cit. on pp. 6–10).

[9] *Routenet,Leveraging graph neural net-works for network modeling and optimization in sdn* (cit. on pp. 6, 10).

[10] *Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization* (cit. on pp. 6, 10).

[11] *Digital Twin for the Oil and Gas Industry: Overview, Research Trends, Opportunities, and Challenges* (cit. on p. 7).

[12] Huan X. Nguyen, Ramona Trestian, Duc To, and Mallik Tatipamula. «Digital Twin for 5G and Beyond». In: *IEEE Communications Magazine* 59.2 (2021), pp. 10–15. DOI: 10.1109/MCOM.001.2000343 (cit. on p. 7).

[13] *Technical Specification 23.502*. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3145 (cit. on pp. 12, 14, 16, 17).

[14] *Source of the image* (cit. on p. 13).

[15] *5G Core Part 1 — Architecture Overview* (cit. on p. 15).

[16] *Open5Gs architecture scheme* (cit. on p. 20).

[17] *Repository of the project*. URL: https://github.com/GiovanniColonni/5gDigitalTwin (cit. on pp. 24, 29, 31).