UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dept. of Communications

Deployment and testing of an Open RAN 5G Network Prototype based on Software Defined Radio

Master's Thesis

Master of Science in Telecommunication Technologies, Systems and Networks

AUTHOR: Pan, Jimmy-Antoine

Tutor: Gómez Barquero, David

ACADEMIC YEAR: 2022/2023

Resumen

La quinta generación de redes móviles cambió la forma en que nos comunicamos. El futuro de 5G está establecido, especialmente con su alta velocidad, baja latencia y gran ancho de banda. Las empresas desarrollan y proponen soluciones para dar respuesta a la creciente demanda, haciendo que los usuarios sean cada vez más dependientes de estas empresas.
Open RAN ha surgido para acelerar la innovación en este campo, haciendo abiertas las redes de acceso por radio (RAN), donde cualquier organización, empresa o desarrollador puede desarrollar nuevos componentes y funcionalidades. Organizaciones como Open Networking Foundation (ONF) están desarrollando controladores de red RAN (SD-RAN) para optimizar y gestionar los diversos recursos de radio de forma óptima, donde terceros pueden construir sus aplicaciones.
Se utiliza OAI para emular los componentes radio tradicionales. El código OAI se virtualiza en los servicios informáticos y se utilizan radios definidas por software (SDR) para transmitir las señales de radio.
El objetivo de este trabajo es poder utilizar la red 5G con código abierto y desvincularse de las grandes empresas. Este trabajo tiene como objetivo implementar y probar la RAN de una red 5G basada en las especificaciones de Open RAN, utilizando SD-RAN y Open Air Interface.

Resum

La cinquena generació de xarxes mòbils va canviar la forma en què ens comuniquem. El futur de 5G està establit, especialment amb la seua alta velocitat, baixa latència i gran amplada de banda. Les empreses desenvolupen i proposen solucions per a donar resposta a la creixent demanda, fent que els usuaris siguen cada vegada més dependents d'aquestes empreses.
Open RAN ha sorgit per a accelerar la innovació en aquest camp, fent obertes les xarxes d'accés per ràdio (RAN), on qualsevol organització, empresa o desenvolupador pot desenvolupar nous components i funcionalitats. Organitzacions com Open Networking Foundation (ONF) estan desenvolupant controladors de xarxa RAN (SD-RAN) per a optimitzar i gestionar els diversos recursos de radi de manera òptima, on tercers poden construir les seues aplicacions.
S'utilitza Open Air Interface (OAI) per a verificar el comportament d'aquest programari. El codi OAI es virtualiza en els serveis informàtics i s'utilitzen radis definides per programari (SDR) per a transmetre els senyals de ràdio.
L'objectiu d'aquest treball és poder utilitzar la xarxa 5G amb codi obert i desvincular-se de les grans empreses. Aquest treball té com a objectiu implementar i provar la RAN d'una xarxa 5G basada en les especificacions de Open RAN, utilitzant SD-RAN i Open Air Interface.

Abstract

The fifth Generation of mobile networks changed the way we communicate. The future of 5G is established, especially with its high-speed, low latency, and large bandwidth. Companies develop and propose solutions to respond to growing demand, making users more and more dependent on these companies.
Open RAN has emerged to accelerate innovation in this field, making radio access networks (RAN) open, where any organization, company or developer can develop new components and function-

alities. Organizations such as Open Networking Foundation (ONF) are developing RAN network controllers (SD-RAN) to optimize and manage the various radio resources in an optimal way, where third parties can build their applications.
Open Air Interface (OAI) is used to verify the behavior of such software. The OAI code is virtualized in computer services and software-defined radios (SDR) are used to transmit the radio signals. The objective of this work is to be able to use 5G network with open-source code and break away from big companies. This work aims to deploy and test the RAN of a 5G network based on Open RAN specifications, using SD-RAN and Open Air Interface.

Acknowledgements

To begin with, I want to truly thank GOMEZ BARQUERO and CURIESES SANZ for their continuous support throughout the internship. I also want to point the quality of their mentorship.

I am deeply grateful for all iTEAM's collaborators for integrating me in the team with events as meetings.

I also want to thank Professor ZAOUIA, mentor of ENSEA, for her advices to write and to defend my Master thesis.

Finally, many thanks to ENSEA's International Relations Office for helping me find this great opportunity in Spain.

Contents

Chapter 1

Introduction

1.1 Context

Today, communication has entered our everyday life. According to the German company Statista [9], there are more than 7 billions mobile phone in the world for 85% of the worldwide population. Telecommunication sector is more and more developed by companies in field of cloud, websites, banking, etc. Due to these huge demands, new technologies are continuously developed to optimize the efficiency of the use of the network. Today, 5G is the changing the way we communicate with high speed and low latency. However, today the 3 biggest 5G equipment supplier companies are Huawei, Nokia, and Ericsson and their equipments are considered as black box, i.e., their content is unknown. To remedy the situation, Open RAN technologies are developed in open source : the researcher makes their work public to let other research use it, test it, and sometime improve it with feedback. The aim is to develop and make 5G technology accessible to everyone with fewer cost. In our case, the aim consists in testing the the SD-RAN part of the 5G network.

1.2 Motivation

Today, the 5G network is new, and the market is led by few companies. The research in Open RAN allow anyone to manage the 5G network. Understanding the RIC should help on managing the 5G network and understanding the xApps should help on observe or using the 5G network. The particularity of Open RAN is that the main code is shared with the community. At the end of this project, we should understand more the main principles of the Open RAN. We would be able to use the existing xApps and create our own one, privately or publicly.

1.3 Objectives

The aim of this MSc Thesis is to deploy a 5G RAN using Open Air Interface and O-RAN technology open-source code and understand the main principles. The project will be carried out in UPV, in the iTEAM center, Mobile Communication Group (MCG).

To achieve this objective, some tracks are proposed :

- Study the Open RAN networks with special interest on the RAN Intelligent Controller (RIC) from Open Networking Foundation (ONF) and O-RAN Alliance initiatives
- Research on SD-RAN software and how it works
- Study the use of Software Defined Radio (SDR).
- Deploy SD-RAN software and interconnect it with Open Air Interface, through virtual computation and SDR.
- Research the different xApps available in the SD-RAN project
- Perform tests with ran-simulator deployed on SD-RAN and simulate several network scenarios

Chapter 2

Framework

2.1 The principle of cellular network

2.1.1 The architecture

A cellular network is wireless network which enable for both stationary and in movement mobile devices.

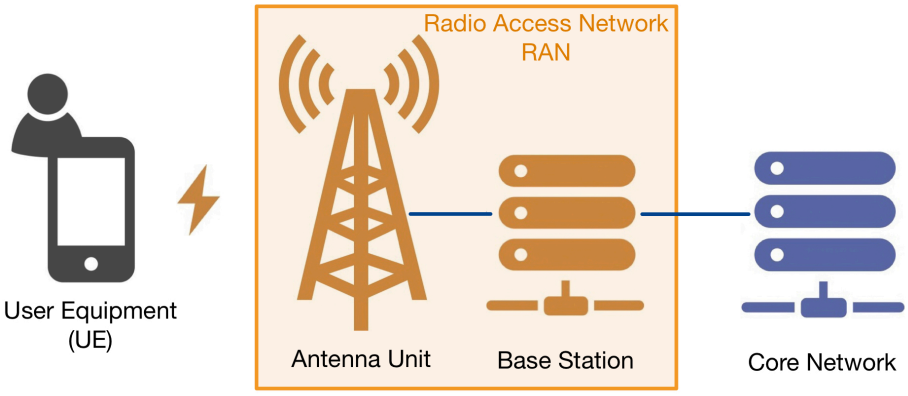

Figure 2.1: Mobile network architecture

The architecture for any mobile network is composed of 3 part :

- The User Equipment (UE) : it represents any device which is used by the end-user to communicate. It can be a mobile phone, a laptop computer etc.
- The Core Network (CN) : also called Backbone Network, it interconnects several networks, providing a path for the exchange of information between different LAN or subnetworks/internet.
- The Radio Access Network (RAN) : it connects several UEs to the core Network. The RAN is made of antennas which cover a specific region which are called cells.

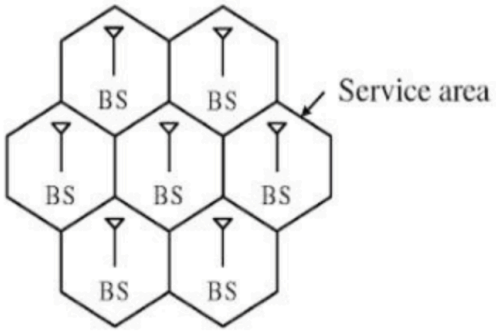

Figure 2.2: Cellular network architecture

2.1.2 3GPP

In the world of telecommunication, there are international agreement which set standards : 3rd Generation Partnership Project or 3GPP. Created in 1998, the 3GPP gathers partners from Asia, Europe, and North America. The aim of the standards organization is to determine technical specifications on mobile network to ensure efficiency, security, and compatibility in all the world.

Generation	Standard name	Maximum Speed	Latency
2G	Global System for Mobile Communications (GSM)	9.6 kbps	300 ms
3G	Universal Mobile Telecommunications System (UMTS)	1,920 Mbps	120 ms
4G	Long Term Evolution (LTE)	300 Mbps	60 ms

Tabla 2.1: 3GPP standards

The 3GPP standards are used by the majority of the countries in the world. However, the maximum speed and latency they set is never reach at the beginning. For example, for LTE, the maximum speed reached is around 150 Mbps. With LTE-advanced, we increased the speed (>1 Gbps).

The 3GPP standards are made by 3 groups, called Technical Specification Groups (TSG) :

- TSG RAN : they work on the radio part (GERAN, UTRAN, E-UTRAN, NG-RAN, etc.)
- TSG CT Core : they work on the Core and the users' terminal
- TSG SA : they work on the system architecture. They are responsible for coordination between all TSG

2.2 From 1G to 4G

From 1980 until today, the wireless network technologies evolve, new features are developed to meet our need. There are international standards, called "Generation" [1] :

- 1981 – 1G : The main goal was to introduce voice system. The first generation wireless transmission was analog. The inconvenient were the big size of the equipments, no security on the communications.
- 1990 – 2G (GSM): The new feature was the exchanges of data as SMS and MMS. The voice system was more secure and the quality was improved. The size of the devices are smaller. However, the transmission of data was really slow. With GPRS (2.5G) and EDGE (2.75G), the mobile equipment can connect to internet even if the network was saturated with digital transmission. The RAN is composed of a Base Transceiver Station (BTS) and a Base Station Controller (BSC). The CN is composed of Packet Switching Domain which transmit the data and the Circuit Switch Domain which transmit voice and SMS.
- 2000 – 3G (UMTS) : Equipments can have video calls or mobile TV and the internet data are in high-speed. However, the costs of these equipments are high. The RAN is composed of Node B. The CN is the same as for GSM.
- 2010 – 4G (LTE) : There are no new features. The improvements lie in the huge increase of the data rate (x75 compared to the 3G). Videos can be viewed in High Definition. The RAN is composed of eNB (Evolved Node B). The CN is now the Evolved Packet Network. The Mobile Management Entity (MME) supports and controls subscribers authentication with the Home Subscriber Server (HSS) which manages security. The Serving Gateway (S-GW) routes the incoming and the outgoing packets. It makes the link to Packet Gateway (P-GW) which works with the IP network.

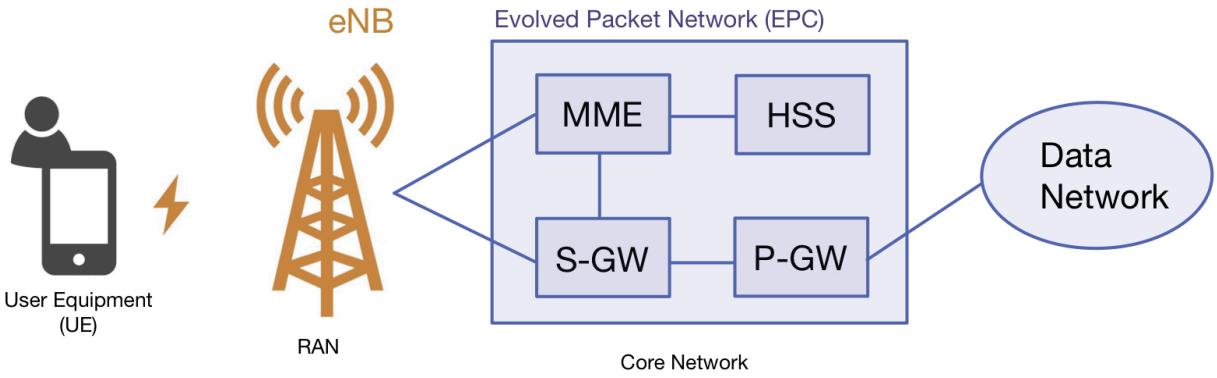

Figure 2.3: 4G Network Architecture

For each “Generation”, we use different installations :

Generation	RAN	Core Network (CN)	Air Interface
2G	BTS/BSC	Packet and Circuit Switch Domain	GSM Edge RAN
3G	Node B	Packet and Circuit Switch Domain	Universal Terrestrial RAN
4G	eNB	Evolved Packet Core	Evolved Universal Terrestrial RAN

Tabla 2.2: Comparison between Mobile Network Generations

2.3 The rise of 5G

In 2018, the 3GPP Release 15 highlighted 5G for the first time. The 5G doesn't replace 4G, it improves the current technologies with :

- Speed : With higher data rate (20 Gbps), we increase the quantity of data that we can load from the network. Example : we can stream or downloading videos with high resolution in seconds.
- Low latency : When 4G's latency is around 60 ms, 5G is around 1ms. The applications who need reactivity can provide better experience. Example : virtual reality, online gaming, video calls …
- Enhanced Capacity : The network can handle more devices at the same time with good performances. It leads to develop field like IoT.
- Network slicing : We can divide the current network in smaller virtual networks. Then, we can allocate resources (as bandwidth, latency, QoS, etc.) and optimize the performances of the different application or services.

With 5G, the industry of Internet of Things can finally shine. With low latency and high data rate, connected devices as smart home, autonomous car or life-saving devices can be developed.

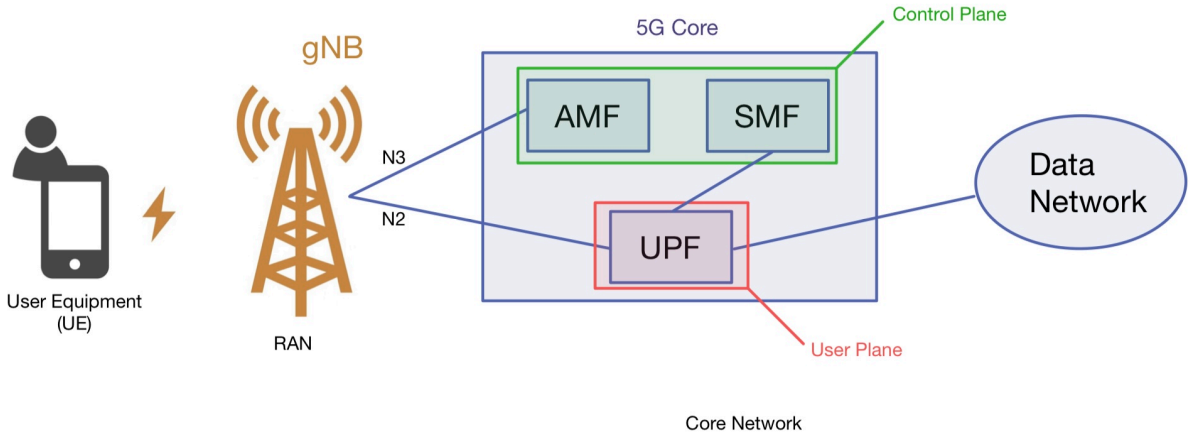

Figure 2.4: 5G Network Architecture

The architecture of 5G is totally different compared to former generations :

- The RAN is composed of a new component : the gNB or gNode B (next generation node B).
- The CN is now the 5G Core and its composition changed. The Core is divided in two part :
  - The user plane which manages the transmission of data between the RAN and the Data Network. The only component is the User Plane Function which forward traffic between RAN and Data Network

– The control plan which ensure safety of the communication between the RAN and the Data Network. The Access Mobility Function (AMF) control the access of the UE to the network. The Session Management Function (SMF) manages the session of the UE (allocate IP, check QoS, etc.)

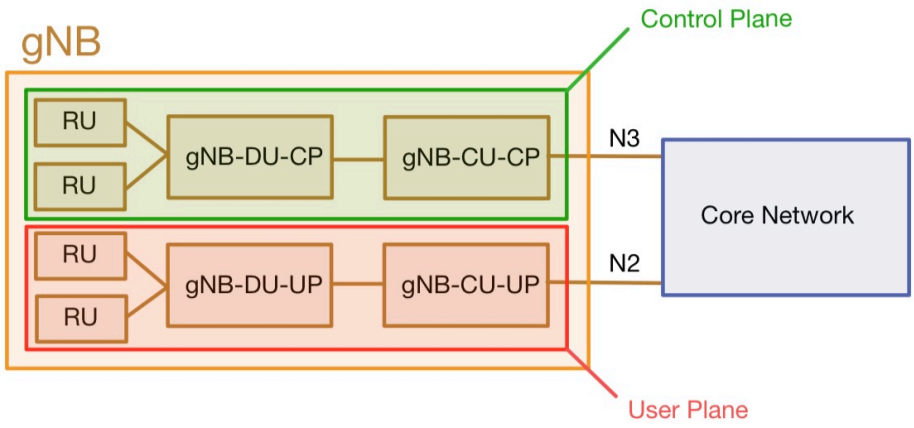

Figure 2.5: 5G gNB Architecture

The gNB is built according to Figure 2.5 architecture : We have components which work with control plane component and others with user plan. The gNB is composed of 3 different types of component :

- Radio Unit (RU) : it manages the connectivity between the UEs and the network. In the gNB, the RU is connected to only one DU.
- Distributed Unit (DU) : it transmits the packets between the RU and the CU. In the DU can serve many RU but only one CU.
- Centralized Unit (CU) : the CU is a center of the gNB, it transmits packet with the CN. A CU can manage many DU.

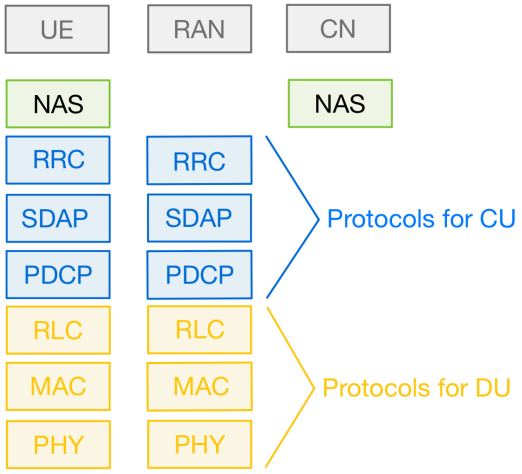

Figure 2.6: 5G Layer protocols

The 5G network works with 7 protocols as follows :

- NAS (Non access stratum): it establishes and maintain connection sessions between RAN and UEs.
- RRC (Radio resource control): it manages the establishment/modification of the connection of UEs, the handover between cells.
- SDAP (Service data adaptation) : it manages the mapping of QoS flows for radio bearers. This protocol establish the order of priority of the packets.
- PDCP (Packet data convergence protocol): it manages the data by ciphering and compressing the headers. It also controls the data rate to avoid congestion.
- RLC (Radio link control) : it manages the packets segmentation and error correction during the transmission with ARQ protocol.
- MAC (Medium access control) : it maps logical channels, dealing with the type of the information, and transport channels, dealing to how the information is carried.
- PHY (Physical) : Lowest layer of the OSI model. It manages modulation, data rate management, channels coding, multiplexing etc.

The centralized units use higher protocols as RCC, SDAP and PDCP and the distributed units uses lower protocols as RLC, MAC and PHY. The radio units use only the PHY protocol. In the other hand, the control plane uses all protocols except SDAP and the user plan uses all protocols except NAS and RRC.

2.4 Open Network Foundation

The Open Network Foundation (ONF) is a group of organization (mainly companies and universities) promoting the development of open source telecommunication technologies. Created in 2011, their goal is to speed innovation with the development of Soft Defined Network.

ONF community gather universities from all the world as the UPV (Spain), the University of UTAH (North America), the National Chiao Tung University (China) or Sorbonne Université (France) but also worldwide companies as Google, Microsoft, Dell, Cisco, or Vodafone.

ONF has currently 2 main areas of projects :

- Mobile Network : the area develops open source solution to improve mobile infrastructure and performances. There are currently 4 projects : SMaRT-5G, Aether, SD-CORE and SD-RAN
- Programming Networks : the area provides network with high flexibility, security and easily configurable. There are currently 5 projects : SD-Fabric, ONOS, P4, PINS and Stratum.

The project consist on deploying the SD-RAN on ONOS environment.

2.5 Open RAN

In 2019, AT&T, China Mobile, Deutsche Telekom, NTT DoComo and Orange created the O-RAN alliance. The objective is to foster innovation and reduce the cost of the equipments. The O-RAN projects can virtually separate the different function of the network. Any supplier can propose solution on these smaller function of the network. When the current leaders (Nokia, Ericsson, Huawei) propose “black box” network, we can understand and configure our “white box” network completely. The different component of the “white box” can be commercialized by smaller companies which can propose multiple version. The creation of the concurrency can reduce the price of the 5G network installation. On the other side, Open RAN allows us to know completely how our network works and add new application (using AI for example) to our network, what is not possible with current “black box” network from the current 5G vendors leaders. The Open RAN has a RAN intelligent controller (RIC) and the Service Management Orchestration. The SMO and the RIC are the new component of this 5G network architecture. The SMO deals with the non-RT part of the 5G network. The RIC is composed of the non-RT RIC, which is in the SMO and the near-RT time RIC which is located in the SD-RAN. Between the SMO and the SD-RAN, the A1 interface manages the communication between the non-RT part and the non-RT part. Between the SD-RAN and the gNB, the E2 interface deals the communication between them. In the gNB part, the component are

ONF community are working on projects based on O-RAN architecture.

Chapter 3

Features of the SD-RAN

SD-RAN is a software that builds open source components for the mobile RAN space, based on O-RAN architecture. They aimed at developing the creation of multivendor RAN solutions (through xApps) and help innovation around the RAN ecosystem. The main goal is to develop is to develop a near-RT RIC called µONOS-RIC and some xApps.

3.1 Architecture of SD-RAN

The SD-RAN simulates the near-real time part of the 5G network. This environment allows near-RT components to work.

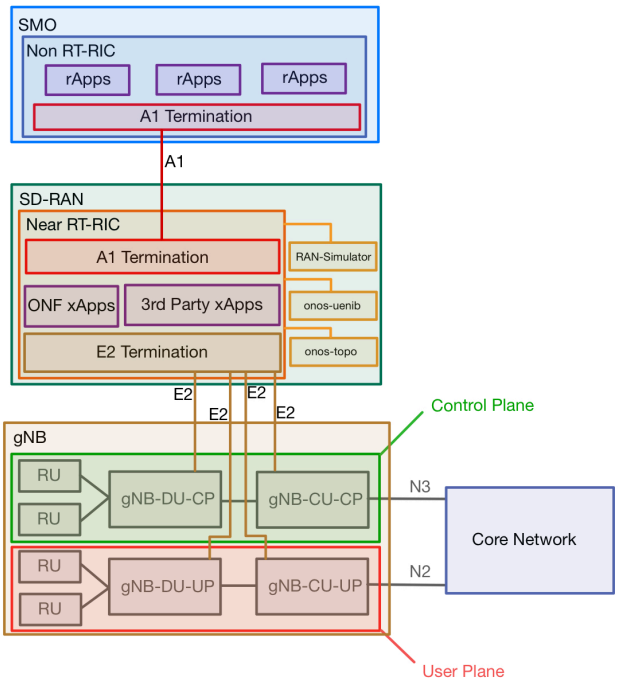

Figure 3.1: SD-RAN Architecture and its environment

The O-RAN architecture includes a RIC, which is divided in two parts :

- non-RT RIC : this RIC controls the network which do not require immediate responses (timescale > 1s) as policy optimization management. This RIC is in the Service Management Orchestration (SMO), which is responsible for managing and orchestrating the services and resources in the O-RAN architecture. The application working in the non-RT RIC are rApps.
- near-RT RIC : this RIC controls the network which require low-latency responses (timescale between 10 ms and 1s) through application called xApps. This RIC is in the SD-RAN and is guided by policies provided by non-RT RIC.

3.2 Components

The SD-RAN cluster contains components which simulate the network. The components can control, manage, and stored the data of the network (UEs, cells, nodes). The most important are :

- onos-e2t : it is E2 termination in the SD-RAN which manages the communication between the RAN and the near-RT RIC.
- onos-a1t : it is A1 termination in the SD-RAN which manages the communication between the non-RT RIC and the near-RT RIC. The A1 interface can receive policies, enrichment information and machine learning model management from the non-RT RIC.
- onos-uenib : it is a tracker for RAN user equipment. It stored the UEs information as their unique identifier (ID) and their aspects. All models set 5 aspects of the UEs as the following :
  - RRCState (Radio Resource Control Status) describe if the mode of the UE. It can be in mode IDLE (standby), CONNECTED or DISCONNECTED.
  - RSRP-Serving (Reference Signal Received Power) is the signal level received by the UE from the serving cell, i.e, the cell to which the UE is connected.
  - The RSRP-neighbor is the level of signal received from the neighbor cell. The UE is connected to a specific cell, but then you receive signals from different cells (with less power than the serving cell).
  - 5QI (5G QoS Indicator) is a pointer to a set of QoS characteristics such as priority level, packet delay or packet error rate, etc. Most of the 5QI are standardized according to 3GPP norms.
  - CGI (Cell Global Identifier) is the unique ID of the cell.
- onos-topo : it provides information about the nodes and the relations between them.

The four previous components are delivered with the O-RAN SD-RAN by default. However, we can add more pieces of software as RAN-simulator or xApps.

3.3 RAN-simulator

To simulate the 5G network, we will use the RAN-simulator. The RAN-simulator emulates the RAN Intelligent Controller which control the xApps and the gNB.

3.3.1 The architecture of the RAN-simulator

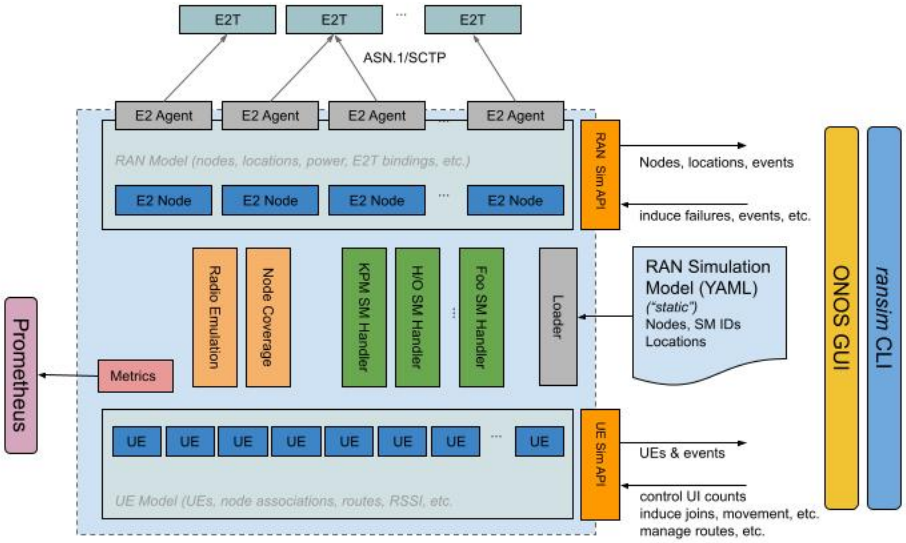

Figure 3.2: RAN-simulator Architecture from https://docs.sd-ran.org/master/ran-simulator/README.html

The RAN-simulator is a software which emulate the Air Interface (UEs and RAN) in the 5G network. It simulates UEs moving between a certain idea where cells and nodes are located. The RAN-Simulator is composed of :

- UEs: the RAN-simulator can simulate up to 100 000 UEs in theory. Each UE has a different identity through a different UE identifier.
- E2 nodes : a node allows the communication of the RAN-simulator with the E2 interface.
- Cells : a cell can cover a virtual to let UE connect to the network.

3.3.2 Models

3.3.2.1 The architecture of a Model

The SD-RAN doesn't provide precise To configure the RAN-simulator and define the different parameters of the RAN-simulator, we have to load a model. The O-RAN alliance provides default

model in their SD-RAN-helm-chart in their GitHub. The most important parameters set are the following :

- E2 nodes : node name, gnbid, the termination controller ID, the service models which can check the node information and the cells that the nodes controls
- cells : cell name, ncgi (ID of a cell), the ncgi of the neighbor cells, the initial PCI.
- controllers : termination control name, ID, and port.
- ueCount: the number of UEs virtualized by the RAN-simulator

The different models are located in ***/sdran-helm-charts/ran-simulator/files/model***. An example of a model is available in Appendix Listing 1.

3.3.2.2 Available models

The ONF's SD-RAN Helm chart provides 8 models with different parameters :

Model	UEs	Cells	Nodes	Max UEs/cell
model.yaml (default)	10	6	2	99999
two-cell-two-node.yaml	1	2	2	99999
three-cell.yaml	5	3	1	99999
three-cell-n-node.yaml	10	3	3	99999
scale-50-150.yaml	100	150	50	99999
scale-model.yaml	100	72	12	99999
model-5cell-100ue.yaml	100	5	3	20
model-7cell-140ue.yaml	140	7	4	99999

Tabla 3.1: Comparison between models

3.4 xApps

The xApps are the application working in the near-RT time (between 10ms and 1s). These applications need quick response. The ONF alliance has released xApps that anyone can test :

- onos-kpimon (KPI MONitoring) : it stored statistics on the different cells as the nodes which the cell is connected, the number of connection (average and maximum).
- onos-pci (Physical Cell Identifier) : it attributes the PCI for all cells.
- onos-mho (Mobile Handover) : it manages the UE handover from a cell to another one. it also stores the number of handover in and out a cell. This xApps works with the RIC.
- onos-mlb (Management Load Balancer) : it is a load indicator. It will adjust the oCn, which measure the load, depending on the handover in or out of UEs

- rimedo-ts (traffic steering): developed by Rimedo Labs, it can apply policy to allow or not the handover of UEs.
- fb-ah (Facebook AirHop) : developed by Facebook and AirHop, it is a PCI conflict manager which can modify cells PCI to avoid conflicts.

These xApps must be installed on the SD-RAN. All of these xApps have been tested with the two-cell-two-node model.

We can resume the architecture of the SD-RAN with a focus on the RAN-simulator and the xApps :

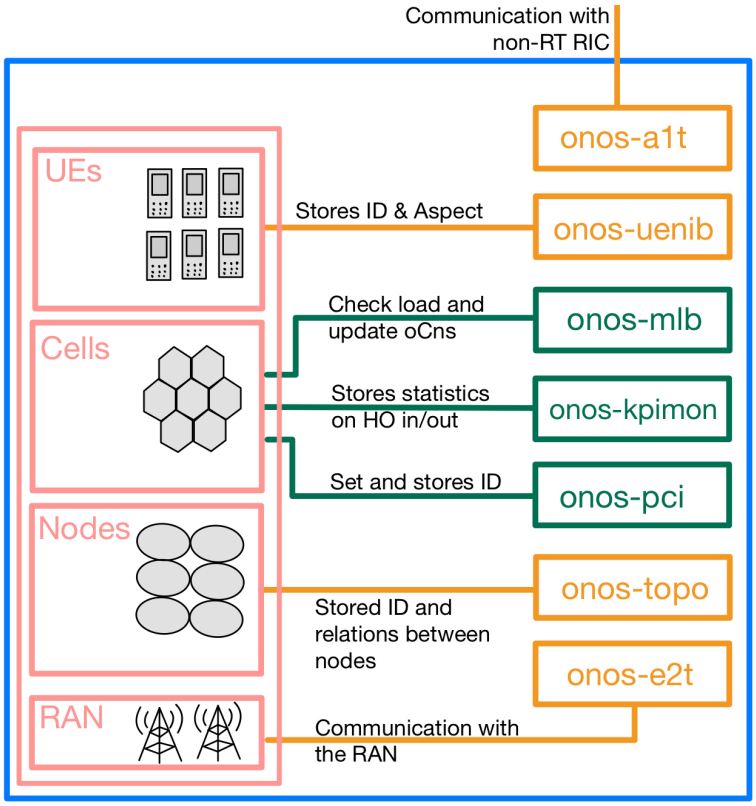

Figure 3.3: Focus on the RAN-simulator and its environment

3.5 Open Air Interface

Open Air Interface (OAI) is a platform developed by the French company Eurecom. This software is aimed at developing soft defined network projects on the RAN and the Core Network with 3GPP standards. The Open Air interface can reproduce 4G and 5G environment.

Chapter 4

Set up

4.1 Environment of work

All the assignments were done on VMs on the server of the Proxmox Virtual Environment. The NAS which contains the Proxmox server memory is located in the last floor of the iTEAM lab. To connect to this server and reach our VMs physically, we can connect ourselves on the network server, especially for USRP we will see later.

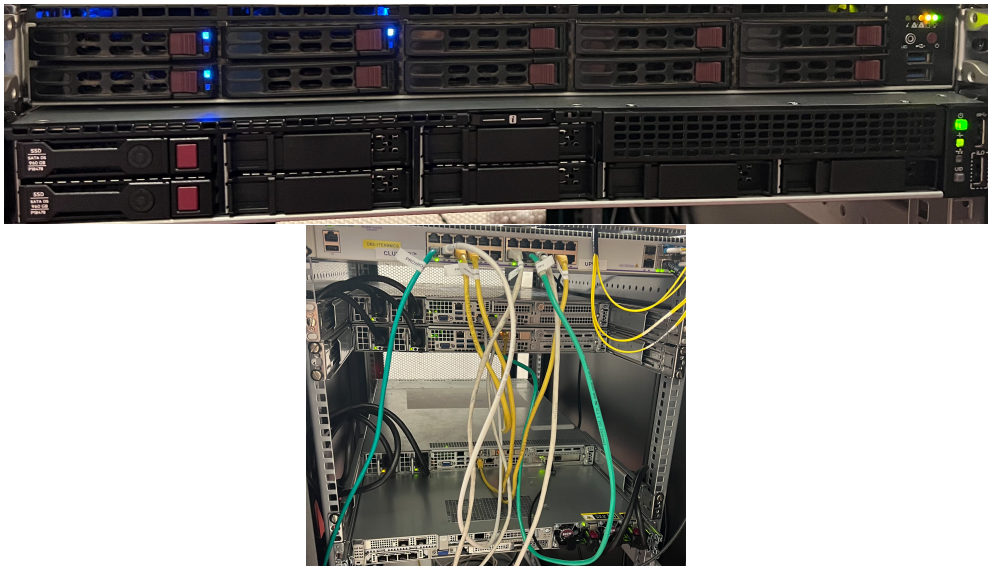

Figure 4.1: Proxmox server (top) and network server (bottom)

4.2 SD-RAN set up

To deploy the SD-RAN, we need some requirements on the environment of the VM :

- Ubuntu v20.04.LTS, 8 cores, 16 GB of memory, 80 GB of disk size.

- Kubernetes v1.27.0 (kubectl, kubeadm, kubelet)
- Docker v23.0.5
- Helm v3.2.1

Docker and Kubernetes are pieces of software which allows us to run containers and deploy microservices. The different components and xApps will be deployed under Kubernetes. Helm is a tool which install these pods easily.

The tutorial of the installation are available in Appendix 2.

Before deploying the SD-RAN, we need to add the different repositories :

Listing 4.1: Adding repositories with helm

```
helm repo add cord https://charts.opencord.org
helm repo add atomix https://charts.atomix.io
helm repo add onos https://charts.onosproject.org
helm repo add sdran https://sdrancharts.onosproject.org
helm repo update
```

We have currently only one namespace ***kube-system***. We need to create a second namespace ***sdran*** where we can deploy our components and xApps :

Listing 4.2: Creating a namespace

```
kubectl create ns sdran
```

To deploy a component, a xApps or the SD-RAN, we need to use helm tool as following :

Listing 4.3: Deploying object using helm

```
helm -n NAMESPACE install COMPONENT_NAME PATH_COMPONENT
```

We deployed in the namespace NAMESPACE, a component located in $PATH\_COMPONENT$ which name is $COMPONENT\_NAME$.

Thus, the command line to deploy in the namespace sdran, a component located in sdran/sd-ran which name is sd-ran is :

Listing 4.4: Deploying SD-RAN using helm

```
cd sdran-helm-charts #All deployments must be done inside this repository
helm -n sdran install sd-ran sdran/sd-ran
```

We can also deploy xApps like this :

Listing 4.5: Deploying xApps using helm chart (Master branch)

```
helm -n sdran install onos-mho onos-mho/onos-mho
helm -n sdran install onos-kpimon onos-kpimon/onos-kpimon
helm -n sdran install onos-mlb onos-mlb/onos-mlb
```

Troubleshooting In the previous instructions, we uploaded the repository from https://sdrancharts.onosproject.org, which is the master branch. Thus, when we use the content of this repository, we are using the files from the internet. We are not able to modify it. If we want to edit files or creating new one, we need to load a copy of the helm chart :

Listing 4.6: Deploying helm chart

```
1 git clone https://github.com/onosproject/sdran-helm-charts.git -b rel-1.4
2 #Cloning the latest version of the helm chart
3 helm dependency build
```

With this cloning, we can skip deploying the previous SD-RAN helm chart (Listing 4.1, line 4). However, the repositories of the components are a bit different : to deploy a component, we need to check first its location.

Listing 4.7: Deploying xApps using helm (Rel-1.4)

```
1 helm -n sdran install onos-mho onos-mho
2 helm -n sdran install onos-kpimon onos-kpimon
3 helm -n sdran install onos-mlb onos-mlb
```

With the previous command lines, we can deploy SD-RAN and xApps individually. The xApps are installed one by one. To uninstall them, we also need to remove them one by one. Uninstalling the SD-RAN will not install the different xApps.

Listing 4.8: Uninstalling SD-RAN and xApps manually using helm (Rel-1.4)

```
1 helm -n sdran uninstall sd-ran
2
3 helm -n sdran uninstall onos-mho
4 helm -n sdran uninstall onos-kpimon
5 helm -n sdran uninstall onos-mlb
```

This can be exhaustive if we want many xApps. Another way to install SD-RAN and xApps is to deploy SD-RAN and add the xApps we want to install as options

Listing 4.9: Deploying xApps with xApps as options using helm (Rel-1.4)

```
1 helm -n sdran install sd-ran sd-ran
2     --set import.onos-mho.enabled=true
3     --set import.onos-kpimon.enabled=true
4     --set import.onos-mlb.enabled=true
```

Thus, we can deploy the xApps with want, but we can also remove the component we do not want as a1t for example. By using this way, we install SD-RAN and xApps in a single time. We cannot remove an xApp individually and if we uninstall the SD-RAN, all the xApps deployed as options will also be removed. (Listing 4.8, Line 1).

We can mix both method : we can install component which needn't be removed during the tests as options and install others individually.

4.2.1 RAN-simulator implementation

4.2.1.1 The deployment

The deployment of the RAN-simulator is similar to xApps deployment, as options with the installation of the SD-RAN or individually. As our work consist in testing the RAN-simulator with different deployment schemes, we will favor deploying SD-RAN with xApps together and install RAN-simulator alone.

Listing 4.10: Deploying SD-RAN and RAN-simulator

```
helm -n sdran install sd-ran sd-ran
    --set import.onos-mho.enabled=true
    --set import.onos-kpimon.enabled=true
    --set import.onos-mlb.enabled=true

helm -n sdran install ran-simulator ran-simulator
```

The RAN-simulator has different models which define the parameters of the simulator (number and ID of UEs, cells, nodes etc.). When the the RAN-simulator is installed as previously, the default model will be loaded if no model is set. To use specific a model, it has to be added as option during the deployment of the RAN-simulator :

Listing 4.11: Deploying SD-RAN and RAN-simulator

```
helm -n sdran install ran-simulator ran-simulator
#Default model deployed

helm -n sdran install ran-simulator ran-simulator
    --set ran-simulator.pci.modelName=two-cell-two-node-model
#two-cell-two-node model deployed
```

To change a model, the RAN-simulator has to be uninstalled and deployed again with the new model.

4.2.1.2 The important command lines

To observe the performance of the SD-RAN, xApps and RAN-simulator, some command lines on the cluster are available :

Listing 4.12: Useful Command Lines

```
kubectl get pods -A
#Display all components available (component name, status, age)

kubectl -n sdran describe pod POD_NAME
#Display information on the component POD_ID
    The POD_ID can be found with the previous command line

kubectl -n sdran get services
#Display the services information about component (IP, port)

kubectl logs -f POD_ID -n sdran
#Check logs of the POD_ID

```

```
kubectl -n sdran rollout restart deploy
#Restart all the components in the namespace SD-RAN
```

4.2.1.3 The onos-cli

The ONOS components can be configured and observe with **onos** command. There are two-way to perform onos command : with the ONOS CLI client or entering the SD-RAN terminal.

The advantages of CLI client are multiple :

- We can perform onos command from our local computer without a different terminal
- We can have access to other repositories. Previously, we saw that RAN-simulator (which use onos command) need model files located in our local computer. There is onos command which allows the RAN-simulator to change model without being removed.

To install the ONOS CLI, the ONOS website (https://docs.onosproject.org/onos-cli/docs/setup) indicates to run the following command :

Listing 4.13: Command lines to install the ONOS CLI client

```
export GO111MODULE=on
go get github.com/onosproject/onos-cli/cmd/onos
```

However, when try this command, the version of **go** (for Golang, which is used to install packages and dependencies) required is 1.17. The default version on our VM is 1.13. With the following commands, we can upgrade the right version of Golang :

Listing 4.14: Command lines to install Golang version 1.17

```
sudorm-rvf/usr/local/go
sudo apt-get -y upgrade
wget https://dl.google.com/go/go1.17.7.linux-amd64.tar.gz
sudo tar -xvf go1.17.7.linux-amd64.tar.gz
sudo mv go /usr/local
export PATH=$PATH:/usr/local/go/bin
```

Listing 4.15: Error during ONOS CLI client installation

```
go get: installing executables with 'go get' in module mode is
deprecated.
```

To avoid this problem and be able to use **onos** command, we execute bash command from the sd-ran pod with :

Listing 4.16: ONOS command line

```
#Enter SD-RAN CLI

kubectl -n sdran exec -it deployment/onos-cli -- (onos cmd)
#Execute one onos command
```

We will be able to check and modify the configuration of our ONOS components. However, we have no access to our local repository because we are inside the SD-RAN pod terminal. With onos command, we can check the property of different components.

4.3 Open Air Interface set up

After testing the RAN-simulator and the different xApps, we will try to use the SD-RAN which will simulate the RAN intelligent controller (RIC) and the Universal Software Radio Peripherals (USRP) which will simulate the gNB. The USRP will be connected on the Open Air Interface with a fiber connection and the OAI and the RiaB VM will be connected via the network server. The version of the USRP is N310, provided by Ettus Research.

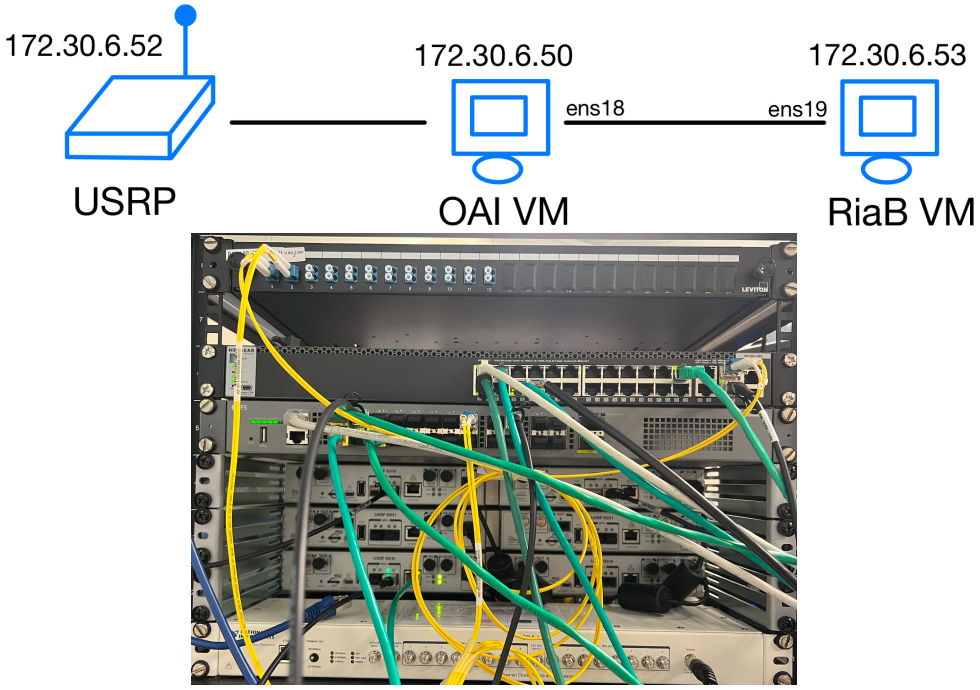

Figure 4.2: RAN-simulator Architecture and the USRP

On the bottom figure, we have 6 USRP connected to the network server.

4.3.0.1 SD-RAN-in-a-box VM

SD-RAN-in-a-box (RiaB) is a software which simplifies the installation of SD-RAN and its environment. The RiaB can and must be deployed without Kubernetes, Docker and Helm. The deployment of the RiaB will install these tools, create the cluster and the differents namespaces that we need to deploy xApps.

Listing 4.17: Install RiaB command lines

```
1 git clone https://github.com/onosproject/sdran-in-a-box
2 cd sdran-in-a-box
```

```
3
4 sudo make clean-all
5 #Remove all components, clusters, Kubernetes, Docker helm etc.
6
7 sudo make omec
8 #Install, Kubernetes, Docker, deploy a cluster and components for Core Network
9
10 sudo make OPT=ric VER=1.4
11 #Install, Kubernetes, Docker, deploy a cluster and components for RIC
```

The main problem in deploying RiaB is that there were incompatibility with version of the different software installed :

- Multus version : during the installation of the RIC, an error on the version of multus occured. Multus is a container which allows pods to have multiple network interfaces.

Listing 4.18: Multus error

```
1 error: timed out waiting for the condition on pods/kube-multus-ds-amd64-7llcf
2 mk/infra.nk:29: recipe for target "../workspace/milestones/k8s-ready' failed
3 make: ***[/home/riab/sdran-in-a-box//workspace/milestones/k8s-ready] Error 1
```

If we check the repository sdran-in-a-box/mk/infra.mk, the version of multus is stable. Editing the version with v3.9.3 solves the problem.

- RaftStorage version : during the installation of the RIC, the version of RaftStorage (which deals with the databases of the SD-RAN) cannot be recognize. In fact, the version of RaftStorage is not compatible with Kubespray (a Kubernetes installer). However, RKE installer can handle RaftStorage installation.

Listing 4.19: Raftstorage error

```
1 Error: unable to build kubernetes objects from release manifest: [unable to recognize "":
2 no matches for kind "RaftCluster" in version
```

The installer will be replaced later in a new version of the SD-RAN helm chart.

By deploying RiaB's Core Network or/and RIC, we can install many components/xApps :

4.4 Open Air Interface VM

We will use the Open Air Interface to emulate the RAN part (gNB CU and gNB DU). The OAI will be the intermediate between the USRP and the RiaB VM. A new VM is created to deploy the OAI. The OAI VM has special prerequisites :

- Ubuntu : 18.04 LTS
- Processor : 8 cores, Haswell-noTSX-IBRS
- Kernel : low-latency (cf Appendix Listing 3)

Listing 4.20: Install RiaB command lines

```
#getting OAI repository
git clone https://github.com/onosproject/openairinterface5g
cd openairinterface5g/
source oaienv
cd cmake_targets/
./build_oai -I -w USRP --eNB --UE

#Building UE and eNodeB Executables
cd openairinterface5g/
source oaienv
cd cmake_targets/
./build_oai -I -w USRP --eNB --UE
```

The building may last up to 30 mn. To simulate the gNB, the USRP must be connect physically and the UHD packages is needed on the OAI VM :

Listing 4.21: Install RiaB command lines

```
#Installing UHD
sudo apt-get install libboost-all-dev libusb-1.0-0-dev
python-mako doxygen python-docutils cmake build-essential
git clone https://github.com/EttusResearch/uhd.git -b UHD-3.15.LTS
cd uhd/host
mkdir build
cd build
cmake ../

make
make test
sudo make install

sudo ldconfig

#Display the USRP available
uhd_find_devices
```

If no error occured, we should get the following output :

Listing 4.22: Install RiaB command lines

```
oaiVM@oai:~/uhd$ uhd_find_devices
[INFO] [UHD] linux; GNU C++ version 9.4.0; Boost_107100; UHD_3.15.0.0-74-ge35f66e8
--------------------------------------------------
-- UHD Device 0
--------------------------------------------------
Device Address:
    serial: 323F0CC
    addr:172.30.6.52
    claimed: False
    mgmt_addr:172.30.6.52
    product: n310
    type: n3xx
```

First, the USRP was connected to a VM and deployed the UHD packages to check the connection. On the other hand, the OAI platform was deployed on another VM. Both installation was successful.

However, we were not able to perform the connection of the USRP on the OAI VM. The USRP is detected successfully but the OAI UE and eNB building cannot be build correctly on this VM. We were trying to perform the integration between the OAI and the SD-RAN.

No.	Time	Source	Destination	Protocol	Length	Info
3044	11.646520	172.30.6.50	172.30.6.51	SCTP	84	INIT
3045	11.646616	172.30.6.51	172.30.6.50	SCTP	52	ABORT
4645	16.646823	172.30.6.50	172.30.6.51	SCTP	84	INIT
4646	16.646861	172.30.6.51	172.30.6.50	SCTP	52	ABORT

Figure 4.3: Wireshark's capture : connection between OAI and RiaB VM

The ping is successfully initiated from the RiaB VM but never reach the OAI VM.

Due to an error with the code, the integration was impossible. We were in touch with the developers and the issue come from the code. No update was released since then.

Chapter 5

Results

5.1 Methodology

The RAN-Simulator works with config files called models. It is interesting to create model our own model files and deploy it to test the performances of the RAN-simulator and the xApps.

To test them, we will set a different number of UEs, cells and nodes, and we will observe the limits of the different components.

5.2 Results with the ran-simulator

5.2.0.1 Creation of new models

To simulate different scenarios, we will create models with different numbers of UEs, cells and nodes.

The name of these models are in the form nUEs_nCells&Nodes. For example, the model 10_2 set 10 UEs, 2 cells and 2 nodes. The number of cells and nodes will be identical due to the specifications of some xApps. In total, 18 models have been created to perform the different tests.

After the creation of a model, we can observe that the model repository is still empty

Listing 5.1: Repository of Models

```
sdran@sdran:~$ ls sdran-helm-charts/ran-simulator/files/model/
model-5cell-100ue.yaml model-7cell-140ue.yaml model.yaml
new.yaml scale-50-150.yaml scale-model.yaml three-cell-model.yaml
three-cell-n-node-model.yaml two-cell-two-node-model.yaml
```

Actually, to apply changes, the helm chart must be updated and the SD-RAN upload again :

Listing 5.2: Helm chart updating

```
helm -n sdran uninstall sd-ran
cd sdran-helm-charts/sd-ran/
helm dependency build
```

After the SD-RAN is deployed again, the new models should be presents

Listing 5.3: Repository with new models

```
sdran@sdran:~$ ls sdran-helm-charts/ran-simulator/files/model/
10000_12.yaml 10_12.yaml 10000_2.yaml 10_2.yaml 10000_3.yaml 10_3.yaml
1000_12.yaml 5000_2.yaml 1000_2.yaml 5000_3.yaml 1000_3.yaml 500_2.yaml
100_12.yaml 500_3.yaml 100_2.yaml 50_2.yaml 100_3.yaml 50_3.yaml
...
```

5.2.0.2 Test with ransim onos command

We can observe the different model parameters through ransim onos command :

Listing 5.4: Ransim onos command

```
kubectl -n sdran exec -it $(kubectl -n sdran get pods -l type=cli -o name) -- /bin/sh
onos ransim get ues
onos ransim get cells
onos ransim get nodes
```

After deployed, the 10_3 model, we can check if the model works :

Listing 5.5: Ransim onos command

```
kubectl -n sdran exec -it $(kubectl -n sdran get pods -l type=cli -o name) -- /bin/sh
$onos ransim get ues
1650607 13842601454c001 90130 false RRCSTATUS_IDLE
1371536 13842601454c003 90129 false RRCSTATUS_IDLE
... (6 more) ...
5070872 13842601454c001 90132 false RRCSTATUS_CONNECTED
7118836 13842601454c001 90134 false RRCSTATUS_IDLE

$onos ransim get cells
NCGI ... Neighbors(NCellOffset)
13842601454c001 ... 13842601454c003(0),13842601454c002(0)
13842601454c003 ... 13842601454c001(0),13842601454c002(0)
13842601454c002 ... 13842601454c001(0),13842601454c003(0)

~ $ onos ransim get nodes
GnbID Status Service Models E2T Controllers Cell NCGIs
5153 Running rcpre2,kpm2,mho e2t-1 13842601454c001
5154 Running rcpre2,kpm2,mho e2t-1 13842601454c002
5155 Running rcpre2,kpm2,mho e2t-1 13842601454c003
```

The RAN-simulator has well created 10 UEs, 2 cells and 2 nodes and attribute them different identifier.

5.2.1 The different tests with xApps

5.2.1.1 onos-uenib

Onos-uenib is a component of the SD-RAN which stores the data of the UEs he detects. The main command lines are the following :

Listing 5.6: onos-uenib command lines

```
#Display the ID and the Aspect types of all ues
onos uenib get ues

#Display the aspect values of the UE with UE_ID as identifier
onos uenib get ue UE_ID --verbose
```

After deployed the RAN-simulator, we observe that onos-uenib doesn't show all UEs set by the RAN-simulator.

```
~ $ onos ransim get ues
IMSI            Serving Cell     CRNTI      Admitted   RRC
1224416         13842601454c001  90130      false      RRCSTATUS_CONNECTED
6723955         13842601454c001  90131      false      RRCSTATUS_CONNECTED
5953100         13842601454c001  90132      false      RRCSTATUS_CONNECTED
1553940         13842601454c003  90133      false      RRCSTATUS_CONNECTED
```

Figure 5.1: onos-uenib command lines

Only the data of 4 UEs are stored by onos-uenib. Actually, these 4 UEs are the those which RRC status is connected. If we wait a bit longer, onos-uenib will get the data of all UE.

When the RAN-simulator is deployed, the different UE's status can be idle or connected with same probability (50%). The onos-uenib component can detect UEs which have made a connected once. So, after a moment, all UEs will make a connection at least once and onos-uenib will be able to store their data even if they switch to idle mode.

We can wonder how much time does onos-uenib need to detect all the UEs. We will use models from 10_3 to 10000_3 and observe onos-uenib performances during 30 minutes.

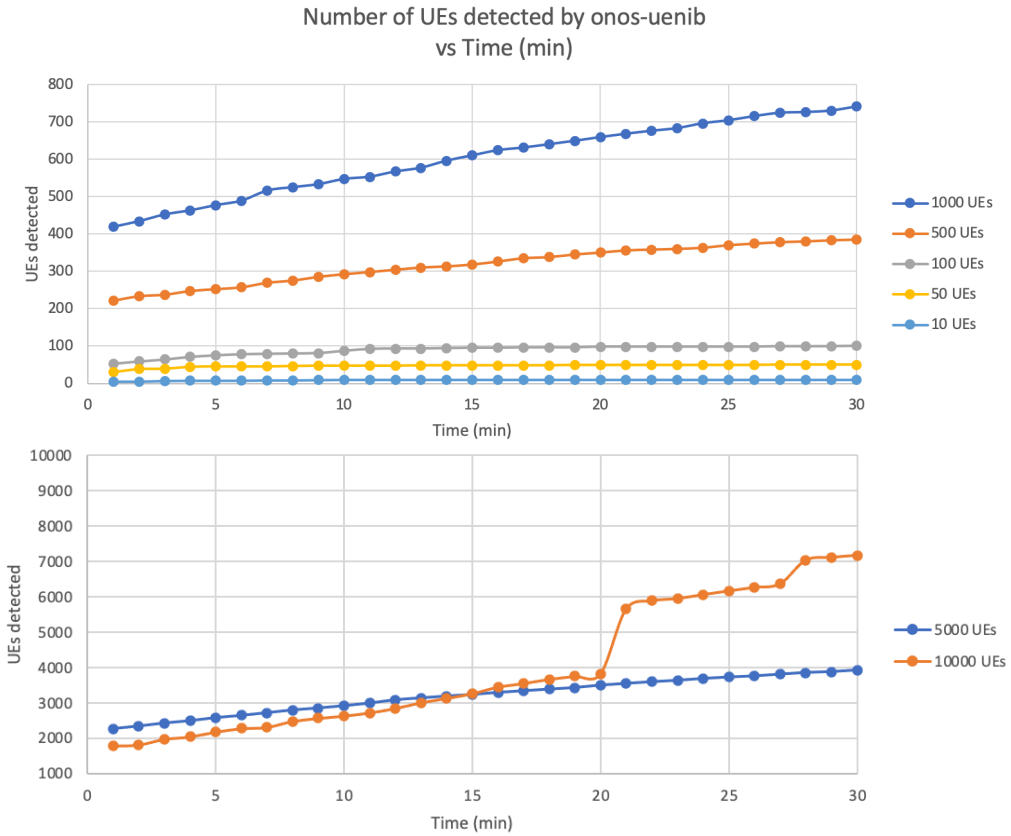

Figure 5.2: Number of UEs detected by onos-uenib vs. time (min) and the number of UEs deployed by the RAN-simulator

In 30 minutes, onos-uenib can detect all the UEs if the RAN-simulator deploys less than 100 UEs. Between 500 and 5000 UEs, onos-uenib needs more time and more handovers to be able to detect all of them. However, if the RAN-simulator deploys more than 10 000 UEs, onos-uenib works not well with our set-up : indeed, for the other models, onos-uenib detect at the beginning around 50% of the UEs deployed by the RAN-simulator. For 10 000 UEs, onos-uenib works less good than with 5 000 UEs : we need more requirements on resources. If we want to increase the UEs deployed by the RAN-simulator (which can deployed up to 100 000 UEs), the SD-RAN must have more RAM storage.

5.2.1.2 onos-mho

The onos-mho xApps is an application which stored the data of the handovers (HO) of UEs in a cell or out a cell. The xApps gives 2 kinds of information :

- cells : it gives information about the cells as CGI (identifier), the number of UEs connected this each cell, the total number of HO in and out the cell.
- UEs : it gives information about the UEs as their ID, the CGI of cell which the UE is connected (or the last cell which made a connection) and the RRC status. The onos-mho xApps will update the RRCState of the onos-uenib.

```
~ $ onos mho get cells
CGI              Num UEs          Handovers-in     Handovers-out
13842602c054140  2                0                0
13842601c054140  2                0                0
~ $ onos mho get ues
UeID     CellGlobalID     RrcState
178450   13842602c054140  CONNECTED
57d7f1   13842602c054140  CONNECTED
378ef7   13842601c054140  CONNECTED
1a35a1   13842601c054140  CONNECTED
```

Figure 5.3: Output of onos-mho command line with 10_3 model after 1 min

We can see that the number of cells and the number of UEs are different from the numbers deployed by the RAN-simulator.

Indeed, onos-mho will store the data of UEs that have been detected once and cells which have made a connection with a cell.

```
~ $ onos mho get cells
CGI              Num UEs          Handovers-in     Handovers-out
13842602c054140  1                12               9
13842601c054140  2                27               28
13842603c054140  1                18               20
~ $ onos mho get ues
UeID     CellGlobalID     RrcState
684dad   13842601c054140  CONNECTED
2088b3   13842601c054140  IDLE
7654ee   13842602c054140  CONNECTED
9088c8   13842603c054140  CONNECTED
178450   13842602c054140  IDLE
57d7f1   13842603c054140  IDLE
378ef7   13842601c054140  CONNECTED
1a35a1   13842601c054140  IDLE
8d0163   13842601c054140  IDLE
6068dd   13842601c054140  IDLE
```

Figure 5.4: Output of onos-mho command line with 10_3 model after 10 min

To test the performances of the onos-mho xApps, we can test the limits with cells as the limits on UEs are the same as with onos-uenib. Unlike adding UEs which just setting a number, adding cells is more difficult. For each cell we want to create, we need to set their ID, their neighbor, the location (latitude, longitude, azimuth, arc, height) and other information.

The experiment will consist in working 12 nodes and 12 cells and see how the number of cells will affect the cells' detection by the onos-mho xApps.

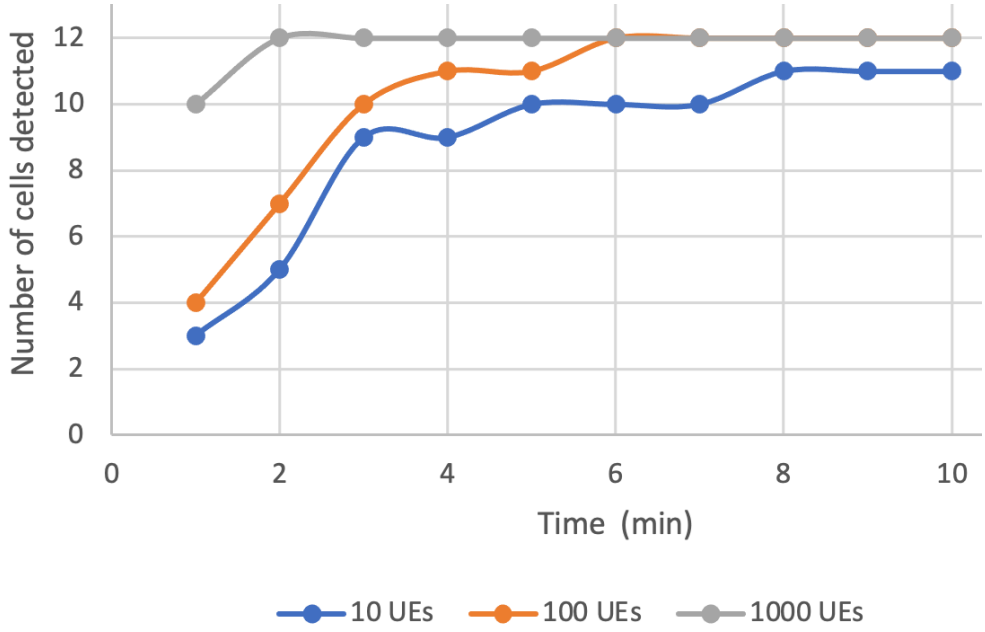

Figure 5.5: Number of cells detected by onos-mho vs. time (min) and the number of UEs deployed by the RAN-simulator

As we can observe, the more UEs are deployed by the RAN-simulator, the more cells will be detected : indeed, all UEs has 1/12 chance to connect a precise cell when they make a HO in. For a precise cell, more UEs means to perform a connection to be detected. For 10 UEs, we never reach 12 cells. In reality, the number of UEs is much higher than the number of cells. So this case will rarely happen.

So, in real conditions, the onos-mho xApps can detect the cells quickly.

5.2.1.3 onos-kpimon

The onos-kpimon xApps is the application which monitor the KPI through the SD-RAN. First, it makes a subscription with all nodes of the SD-RAN. Then, all nodes will report KPI to the xApps as :

- the average number of connection
- the maximum number of connection
- the number of HO which failed
- the number of reconfiguration of the RRC status which failed

The onos-kpimon xApps will store the data and share it with onos-uenib.

```
~ $ onos kpimon list metrics
Node ID          Cell Object ID       Cell Global ID           Time    RRC.Conn.Avg    RRC.Conn.Max
e2:1/5153       13842601454c001            1454c001      16:17:06.0               4               5
e2:1/5154       13842601454c002            1454c002      16:17:06.0               2               4
e2:1/5155       13842601454c003            1454c003      16:17:06.0               4               5
```

Figure 5.6: Data stored by onos-kpimon

We deployed 10_3 model, we have 3 cells represented. The cell object ID (COI) is defined with PLMD (which is defined by the network) and cell global ID (CGI) : COI = PLMD+CGI.

For this xApps, we can modify the interval of the reports with **onos-kpimon set report-interval**

5.2.1.4 rimedo-ts

The xApps developed by Rimedo Labs, rimedo-ts, is a traffic steering application. It manages the mobile handover when UEs are moving physically between cells. The objective is to optimize the network by setting different policies. These policies can be set manually or set with AI models to predict what cell is better for a UE. The objective is to be able to modify the way the network perform the handover by sending policy which indicate how to make the handover. The policies will be set manually.

The RAN-simulator will load the two-node-two-cell node. This model deploys a unique UE with two cells. The particularity of this UE is that it is always connected and never under idle mode. The model simulates a unique UE moving between 2 cells.

To work, the rimedo-ts xApps required information from E2 interface as :

- User Equipment identity IMSI or UE ID
- Cell Global Identity CGI
- RRC state of the UEs
- 5G QoS Identifier 5QI
- The received power by each UEs and cell's RSRP

RRC state, 5QI and RSRP are available with onos-uenib with Aspect command. This information can be set in the policy :

Listing 5.7: onos-uenib command lines

```
{
    "scope":{
        "ueId":"00000000024857552"
        "qosId": {"5qI": 1}
    },
    "tspResources":[
        {
            "cellIdList":[
                {
                    "plmnId":{
                        "mcc":"138",
```

```
                    "mnc":"426"
                },
                "cId":{
                    "ncI":3179675
                }
            }
        ],
        "preference":"FORBID"
    }
]
}
```

The policy send to a1t must be in JSON and requires information as :

- ueID or/and qosID :
  - ueID : the ID of UEs concerned by the policy. It must be a 16 digits value (we will the value with 0 until there is 16 digits). Group of UEs can be selected
  - qosId : the value of 5QI of the UE concerned by the policy. Group of 5QI can be selected
- plmnID and cID :
  - plmnID : header ID of the cells concerned
  - cID : ID of the cells concerned by the policy. Groups of cells can be selected
- preference: it defines how the UEs should do with the cells. There are 5 types :
  - DEFAULT : the UEs will connect to the closest cells around.
  - SHALL : the UEs will always connect to the concerned cells
  - FORBID : the UEs will never connect to the concerned cells
  - PREFER : the UEs will prefer to connect to concerned cells and if other cells are too far away
  - AVOID : the UEs will avoid connecting to concerned cells and if other cells are not to far away

To apply a policy, the a1t external as if it was sent by the non-RT RIC. The IP of the different components can be found in the following command :

```
sdran@sdran:~/sdran-helm-charts$ kubectl -n sdran get services
NAME                      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                                  AGE
onos-a1t                  ClusterIP   10.103.213.215   <none>        7001/TCP,5150/TCP,9339/TCP,9639/TCP      31s
onos-a1t-external         NodePort    10.105.105.129   <none>        9639:31963/TCP                           31s
onos-config               ClusterIP   10.107.41.95     <none>        5150/TCP,5151/TCP                        31s
onos-consensus-store      ClusterIP   10.108.85.20     <none>        5678/TCP,5679/TCP                        31s
onos-consensus-store-hs   ClusterIP   None             <none>        5678/TCP,5679/TCP                        31s
onos-e2t                  ClusterIP   10.110.139.197   <none>        36421/SCTP,5150/TCP                      31s
onos-e2t-hs               ClusterIP   None             <none>        5150/TCP                                 31s
onos-topo                 ClusterIP   10.111.12.117    <none>        5150/TCP                                 31s
onos-uenib                ClusterIP   10.100.155.51    <none>        5150/TCP                                 31s
onos-uenib-hs             ClusterIP   None             <none>        5150/TCP                                 31s
ran-simulator             ClusterIP   10.105.167.49    <none>        5150/TCP,9090/TCP                        31s
sd-ran-rimedo-ts          NodePort    10.103.206.199   <none>        5150:30142/TCP                           31s
```

Figure 5.7: SD-RAN services information

However, we were not able to reach the a1t-terminal component. So we were not able to apply the policy to test the rimedo-ts xApps performances.

Chapter 6

Conclusion

6.1 The project

This master thesis project has focused on testing the Open RAN technologies especially SD-RAN. The SD-RAN environment is a major breakthrough in the 5G technologies. With 3GPP standards, the SD-RAN can run application with 5G standards, especially low latency and high data rate. In the coming year, the Open RAN technology can replace the current 5G technology and develop new markets.

First, the project consisted in understanding the 5G technology, especially the RAN part and how Open RAN can contribute.

Next, to implement the SD-RAN, we need to set the appropriate configurations to run properly the environment of the Open RAN. The set-up took the major time of the work as the SD-RAN project is still under development and the different code is always being improved by the community. The different error could be from a problem of physical connection, an incompatibility between the version of different component or an obsolete code. The job was to try to solve the different errors, to report it to the community and test the new version if available. The SD-RAN was successfully deployed while Open Air Interface was not. For the OAI, the developers were focused on the development SD-RAN part.

Finally, the most interesting part of the project was the tests of the performances of the xApps on the SD-RAN. The SD-RAN can work without SD-RAN because there are already some components which can run and manage it (uenib, a1t, e2t, topo). However, xApps as onos-kpimon can report KPI and onos-mho which can update data stored by uenib quicker. xApps as rimedo-ts can use the a1t to send policy to optimize HO inside the SD-RAN. The onos-mlb is able to manage and balance the load, to optimize the energy allocation for the different cells.

During the project, we saw that the different xApps improve the 5G network. Other vendor can propose xApps to improve the network but also to use this network for other uses as virtual reality, internet of thing or online gaming.

6.2 Future work

As future work, I would recommend to focus on :

- Testing other xApps which control/observe the network developed by other companies as Facebook-AirHop
- Creating new xApps which use the network for internet of thing for example.
- Test the future code release by ONF on Open Air Interface to be able to test the connection between the USRP and the RIC

6.3 My experience

During this Master Thesis, I learned how does the 5G network work and its features. I developed my knowledge around Network and Telecom field, specialities I took during my ENSEA cursus. The most difficult part of the internship was my capacity to integrate all this knowledge.

For the implementation part, I developed deeper skills on pieces of software as Kubernetes and Docker with containerization of microservices. I learned how to use it and what place it has in the project. The most difficult part was the configuration of the virtual machines, the software, and the connection between them.

On the personal level, I developed a good relationship with the iTEAM lab collaborators, especially with my mentor and the Mobile Communication Group. I do truly thank them for integrating me in the project and my mentor for supervising my internship.

Bibliography

[1] VIAVI SOLUTIONS, *Understanding 5G: A Practical Guide to Deploying and Operating 5G Networks*, Authority Publishing, ISBN : 978-1949642216, pages: 290, 2019

[2] Michele Polese, Leonardo Bonati, Salvatore D'Oro, Stefano Basagni, Tommaso Melodia,*Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges*, URL : https://arxiv.org/pdf/2202.01032.pdf, 2022

[3] Open Network Foundation, *SD-RAN Documentation*, URL: https://docs.sd-ran.org/, 2020

[4] *Kubernetes Documentation*, URL: https://kubernetes.io/docs/tasks/tools/, 2022

[5] *Docker Documentation*, URL: https://docs.docker.com, 2023

[6] *Helm Documentation*, URL: https://helm.sh/docs/, 2023

[7] Leyva Pupo, Alejandro Santoyo-Gonzalez, Cristina Cervelló-Pastor, *A Framework for the Joint Placement of Edge Service Infrastructure and User Plane Functions for 5G*, URL: https://www.researchgate.net/publication/335819993, 2019

[8] Eurecom, *Open Air Interface Documentation*, URL: https://gitlab.eurecom.fr/oai/openairinterface5g/-/wikis/home

[9] Mobile phone statistics, *More Phones Than People*, URL: https://www.statista.com/chart/4022/mobile-subscriptions-and-world-population/, 2023

[10] Larry Peterson, Oguz Sunay, and Bruce Davie, *Private 5G: A Systems Approach*, URL: https://5g.systemsapproach.org, 2022

[11] Marcin Dryjańsk,*Rimedo Labs to Integrate and Open Source TS xApp with ONF's SD-RAN*, https://rimedolabs.com/blog/rimedo-labs-to-integrate-and-open-source-ts-xapp-with-onfs-sd-ran/, 2022

[12] Adam Samorzewski,*Policy-based Traffic Steering xApp implementation within O-RAN*, https://rimedolabs.com/blog/policy-based-traffic-steering-xapp-implementation-within-o-ran/, 2022

[13] Techplayon ,*Open RAN – Service Management and Orchestration (SMO)*, https://www.techplayon.com/open-ran-service-management-and-orchestration-smo/, 2021

[14] ETSI ,*THIRD GENERATION PARTNERSHIP PROJECT (3GPP)*, URL: https://www.overleaf.com/project/648b0a970a2e4204e881a069, 2022

[15] I V Ryabov, A I Melnikova4, R I Gorokhova, *Physical principles of work and basic features of Universal Software Radio Peripheral*, URL: https://iopscience.iop.org/article/10.1088/1742-6596/1728/1/012013/pdf, 2020

[16] Ettus Research,*USRP N300/N310/N320/N321 Getting Started Guide*, URL: https://kb.ettus.com/USRP_N300/N310/N320/N321_Getting_Started_Guide, 2022

[17] Alain Sultan, *https://www.3gpp.org/technologies/5g-system-overview*, URL: https://www.3gpp.org/technologies/5g-system-overview, 2022

[18] Nikhil Bhandari, Shivinder Devra, Karamdeep Singh, *Evolution of Cellular Network: From 1G to 5G*, URL: https://oaji.net/articles/2017/1992-1515158039.pdf, 2017

Part I

Appendix

Listing 1: Structure of a model of the RAN-simulation

```
nodes:
  node1:
    gnbid: 20819
    controllers:
    - e2t-1
    servicemodels:
    - mho
    - rcpre2
    - kpm2
    cells:
    - 87893173159116801
    - 87893173159133185
cells:
  cell1:
    ncgi: 87893173159116801
    neighbors:
    - 87893173159133185
    pci: 218
  cell2:
    ncgi: 87893173159133185
    neighbors:
    - 87893173159116801
    pci: 115
controllers:
  e2t-1:
    id: e2t-1
    address: onos-e2t
    port: 36421
ueCount: 10
```

Listing 2: Kubernetes, Docker and Helm installation

```
# Installation of Kubernetes :

sudo apt update && sudo apt -y full-upgrade
[ -f /var/run/reboot-required ] && sudo reboot -f

sudo apt -y install curl apt-transport-https
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg \
| sudo apt-key add -
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" \
 | sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo apt update
sudo apt -y install vim git curl wget kubelet cubed kubectl

# Installation of kubelet, kubeadm and kubectl
sudo apt-mark hold kubelet kubeadm kubectl

# verify if the Kubernetes is running
kubectl version --client

```

```
sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab

sudo swapoff -a
sudo mount -a
free -h

sudo modprobe overlay
sudo modprobe br_netfilter

sudo tee /etc/sysctl.d/kubernetes.conf<<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
EOF

sudo sysctl --system

# Set up container run time. Only set one at a time
Set up run time with containerd

sudo tee /etc/modules-load.d/containerd.conf <<EOF
overlay
br_netfilter
EOF

# Load at runtime
sudo modprobe overlay
sudo modprobe br_netfilter

# Ensure sysctl parameters are set
sudo tee /etc/sysctl.d/kubernetes.conf<<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
EOF

# Reload configs
sudo sysctl --system

# Install required packages
sudo apt install -y curl gnupg2 software-properties-common \
apt-transport-https ca-certificates

# Installation of the Docker
curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
\sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] \
 https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

# Install containerd
sudo apt update
sudo apt install -y containerd.io

```

```
# Configure containerd and start service
sudo su -
mkdir -p /etc/containerd
containerd config default>/etc/containerd/config.toml

# restart containerd
sudo systemctl restart containerd
sudo systemctl enable containerd
systemctl status containerd

sudo systemctl enable kubelet
sudo kubeadm config images pull
sudo systemctl restart kubelet
sudo sysctl -p

# Initialisation of the cluster
sudo kubeadm init \
  --pod-network-cidr=172.24.0.0/16 \
  --cri-socket unix:///run/containerd/containerd.sock \
  --upload-certs \
  --control-plane-endpoint=k8cluster \

sudo nano /etc/hosts
                    append : 172.29.20.5 k8cluster

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

kubectl taint nodes --all node-role.kubernetes.io/control-plane-

# Helm installation
curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | \
sudo tee /usr/share/keyrings/helm.gpg > /dev/null
sudo apt-get install apt-transport-https --yes
echo "deb [arch=$(dpkg --print-architecture) \
signed-by=/usr/share/keyrings/helm.gpg] \
https://baltocdn.com/helm/stable/debian/ all main" | \
sudo tee /etc/apt/sources.list.d/helm-stable-debian.list

sudo apt-get update
sudo apt-get install helm
```

Listing 3: Low

```
sudo -aptget install --linuximagelowlatency --linuxheaderslowlatency
#To run on OAI, we must disable -pstate and -cstate in Linux :
#Go to nano /etc/default/grub file and add set : GRUB_CMDLINE_LINUX_DEFAULT=
"quiet intel_pstate=disable processor.max_cstate=1 intel_idle.max_cstate=0 idle=poll"

sudo -updategrub2

#Go to /etc/modprobe.d/blacklist.conf file and append at the end : blacklist intel_powerclamp

```

```
#Booting on the right Kernel : reboot then shift/esc and choose low latency mode
    (shift ->advanced mode -> low latency)

sudo -aptget install cpufrequtils
#Go to nano /etc/default/cpufrequtils and append :
GOVERNOR="performance"
#create the file if it 'doesnt exist

sudo systemctl disable ondemand.service
sudo /etc/init.d/cpufrequtils restart


#Check the kernel :
uname -r
```