UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

IMPLEMENTATION OF A NON-RT RIC FOR AUTOMATION SERVICE DEPLOYMENT OVER 4G SMALL CELLS BASED ON OPENAIRINTERFACE TECHNOLOGY

A Master's Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Pau Tomàs Culubret

In partial fulfilment

of the requirements for the degree of

MASTER IN TELECOMMUNICATIONS ENGINEERING

Co-advisors: Carlos Herranz and Jordi Casademont

Barcelona, June 2022

Title of the thesis: Implementation of a non-rt RIC for automation service deployment over 4G small cells based on OpenAirInterface technology.

Author: Pau Tomàs Culubret

Co-advisor: Carlos Herranz and Jordi Casademont

Abstract

With the continuous evolution of the mobile communications networks, it is important to have virtualized open-source ecosystems such as OpenAirInterface, that can provide low cost networks to the different operators.

In this project we aim to develop a centralised radio controller for the management of 4G small cells based on OpenAirInterface technology. This radio controller interacts with the specific solution equivalent to OAI's RT RIC called FlexRAN. The objective of the radio controller is to achieve an automated deployment of 4G services over OAI technology cells, where each service is assigned several dedicated radio resources (RAN slice). Specifically, the aim is to manage the radio resources assigned to the different services dynamically and according to the requirements of the service (QoS, latency, etc.).

Acknowledgements

First of all, I would like to thank Carlos Herranz, Ph. D, for giving me the opportunity of developing my master thesis in i2cat Foundation. Moreover, I would like to express my gratitude towards him for being always willing to help me in every matter and for his implication on this project. Also, to my advisor at UPC, Jordi Casademont for his willingness to help.

In addition, I would like to thank to all the people that I met during my master's and bachelor's degree, as we have come this way together. Finally, I would like to thank to my parents and friends who have been always showing their support to me.

Revision history and approval record

Revision	Date	Purpose
0	21/05/2022	Document creation
1	16/06/2022	Document revision
2	20/06/2022	Document revision

Written by:		Reviewed and approved by:	
Date	22/06/2022	Date	24/06/2022
Name	Pau Tomàs	Name	Jordi Casademont
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables

1. Introduction

Since their inception, mobile communication networks have gained prominence as the number of different modern applications have considerably increased during the last decades. All of which, undoubtedly, leading to an enormous demand for more traffic volume, more device connectivity, and greater Quality of Experience (QoE).

In fact, these networks have experienced a tremendous evolution over the decades. Starting with the first generation (1G) launched in 1979 with analog technology. Next, the second generation (2G) in which digital telephony was introduced and followed by the third generation (3G) which was the first generation to incorporate Internet services. Then the forth generation (4G) appeared, increasing the speed and the quality of the data, and now, with the recent fifth generation (5G), it is providing more services than ever, with higher efficiency and capacity.

Consequently, with the introduction in the market of these 5G networks, all the 2G and 3G networks are getting now discharged. However, this is something that will not happen with the 4G [1], these networks will be necessarily in the market for many years, as there are many use services that will not be deployed in 5G. In fact, operators will continue to use 4G as a primary connectivity for users, and 5G will be presented to offer services where it requires a major traffic demand.

With this tremendous evolution speed, it is important to have virtualized open-source ecosystems that allows to develop and manage the software and hardware for the 3GPP cellular network (4G and 5G). These software solutions can be used on radio over general hardware equipment (COST servers), which provides a cost reduction and time optimization for operators. This is a great advantage over the proprietary solutions that works only within the manufacturer's own hardware.

One example of these virtualized open-source ecosystem is OpenAirInterface (OAI). This ecosystem allows to set up a 4G/5G network and inter-operate with commercial equipment, and so, it opens a world full of possibilities to incorporate new technologies and functionalities into a network

In this research line, the aim of this project is to develop a centralized radio controller owned by the i2CAT Foundation (research center) to manage 4G and 5G small cells with OpenAirInterface technology. With this radio controller, it is intended to create an automated deployment of different 4G and 5G services over the different cells. These services will allow to assign a name of dedicated radio resources (RAN slices), and to manage the different PLMN and core networks of the radio. More in detail, this is planned to manage the different radio resources assigned to different services in a dynamic way and taking into account all service requirements (Quality of Service (QoS), latency, etc...).

1.1. Work Plan

The work plan followed in this project is summarized below, as well as a Gantt Diagram with the different tasks performed.

Work Package	Task	ID	Start Date	End Date
OpenAirInterface network 4G study	4G Network Architecture study	1.1	12/01/2022	20/01/2022
	OpenAirInterface 4G study	1.2	18/01/2022	22/01/2022

	OpenAirInterface 4G deployment	1.3	22/01/2022	30/01/2022
OpenAirInterface network 5G study	5G Network Architecture study	2.1	01/02/2022	10/02/2022
	OpenAirInterface 5G study	2.2	10/02/2022	15/02/2022
	OpenAirInterface 5G deployment	2.3	15/02/2022	25/02/2022
Software development	OAI API development	3.1	25/02/2022	10/03/2022
	OAI exporter development	3.2	10/03/2022	25/03/2022
	NetConf-Server development	3.3	25/03/2022	15/04/2022
	Netconf-Manager development	3.4	16/04/2022	25/04/2022
	Racoon-Core development	3.5	26/04/2022	15/05/2022
Software Integration	Integrate all the system elements	4.1	15/05/2022	20/05/2022
Solution Demonstration	Perform different demonstrations to validate the work	5.1	21/05/2022	15/06/2022

Table 1: Work Plan Table

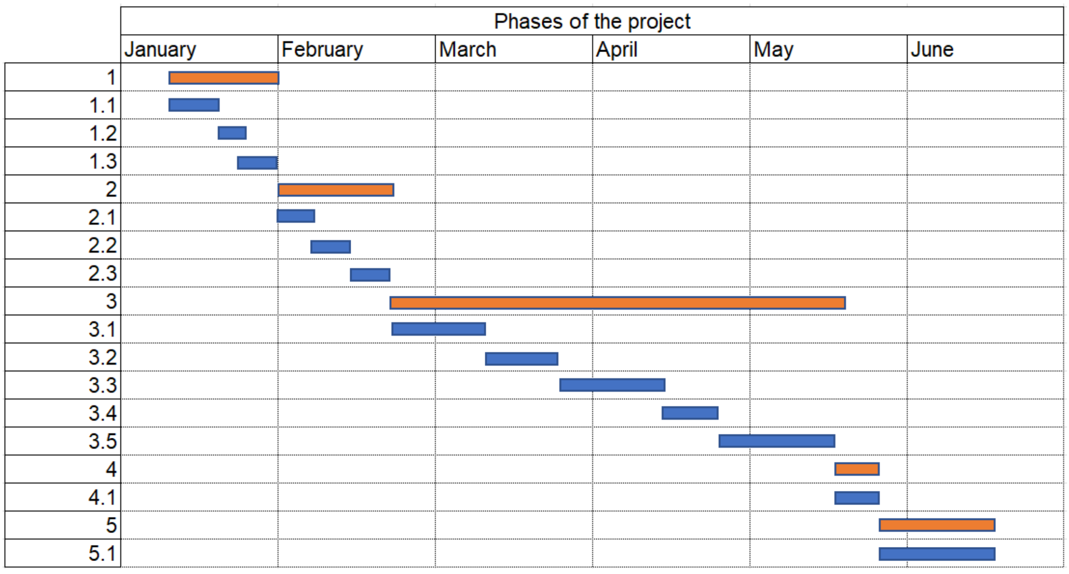


Figure 1: Gantt Diagram

1.1.1. Problems and modifications from initial work plan

Throughout the development of the project some problems in the deployment have occur that have deviated a little the project specifications. In fact, it has not been possible to deploy the 5G network, as OpenAirInterface has some bugs with connectivity of the different elements of the network that made impossible to incorporate 5G in the deployed system. Therefore, this work includes only the deployment of i2CAT's proprietary centralized radio controller for the management of 4G small cells based on OpenAirInterface technology. But, as can be seen in this project, the deployment of the 5G network has also been prepared, to be put in action when OpenAirInterface can fix these software bugs, and we are confident it will be in the near future.

2. State of the art of the technology used or applied in this thesis

This chapter will give an overview of the different 4G and 5G Network Architecture. It will also introduce the OpenAirInterface ecosystem opensource as well as the network management protocol Netconf.

2.1. 4G Network Architecture

Long Term Evolution (LTE) [2] is the fourth generation 4G in the mobile communications networks, it was introduced by the Third Generation Partnership (3GPP) and it is the evolution of the previous 3G (UMTS) and 2G (GSM) generations. Even the related specifications were formally known as evolved UMTS terrestrial radio access (E-UTRA) and evolved UMTS terrestrial radio access network (E-UTRAN). The first version of LTE was documented in 3GPP Release 8 specifications.

The main objective of 4G technology is to provide a high data rate, with low latency and an optimization that allows flexible bandwidth deployment. And its network architecture is designed to support packet-switched traffic with seamless mobility and high quality of service.

LTE main features:

- Data rates of 1Gbps peak in downlink and 500Mbps peak in uplink.
- It uses both Time Division Duplex (TDD) and Frequency Division Duplex (FDD).
- It supports flexible carrier bandwidths, from 1.4 MHz up to 20 MHz as well as both FDD and TDD. LTE designed with a scalable carrier bandwidth from 1.4 MHz up to 20 MHz which bandwidth is used depends on the frequency band and the amount of spectrum available with a network operator.
- MIMO transmissions support.
- In the network architecture, all nodes are IP based, which includ the backhaul connection to the radio base stations.
- Standardization of QoS mechanisms on all interfaces.
- Enhanced capacity and low cost per bit.

Its architecture presents a flat structure, with two main parts in the network, the Core Network (CN) which can be seen as the Evolved Packet Core (EPC) and the Radio Access Network (RAN) which is the Evolved Terrestrial Radio Access Network (E-UTRAN) [3].

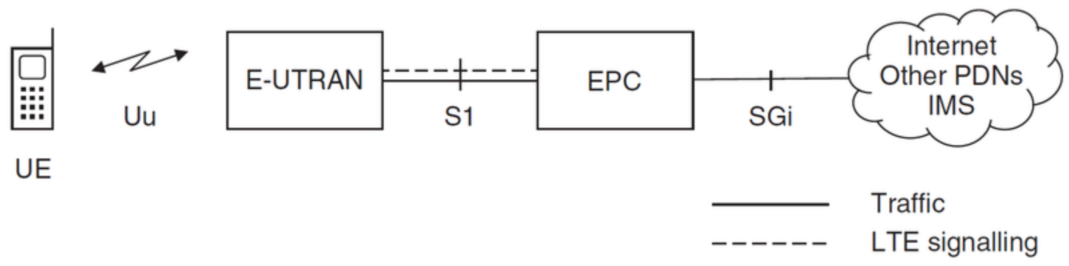


Figure 2 LTE architecture with Core Network and Radio Access Network

In the figure 3, it is detailed which elements are included in this architecture, both in the core network and in the RAN access network, and also all the different interfaces that participate in this technology.

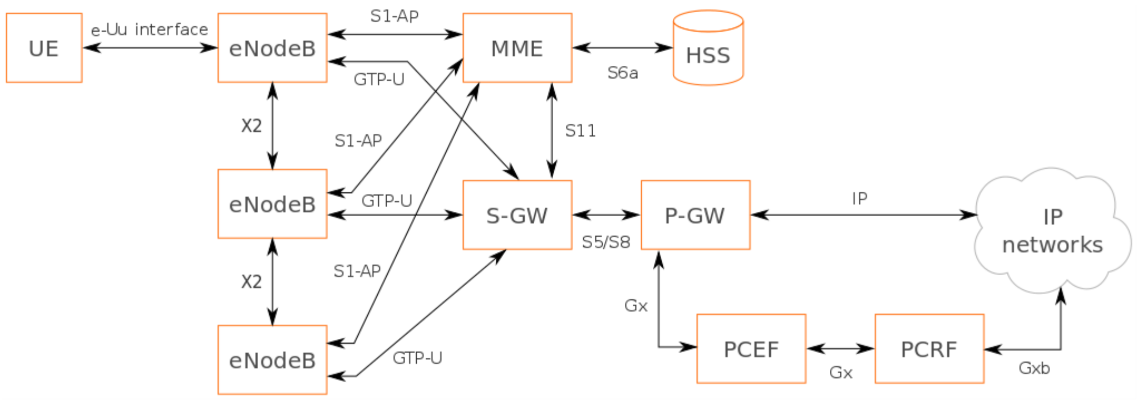


Figure 3 LTE architecture standard [13]

<u>Radio Access Network</u>

The Radio Access Network (RAN) [4] is composed of a single network entity called evolved-NodeB (eNodeB) which constitutes a E-UTRAN base station. The eNodeB integrates all the functionalities of the access networks, very different from previous generations in which the access networks were composed of base stations and colling equipment.

The E-UTRAN access network consists of eNodeBs that provides the connectivity between a user equipment (UE) and the EPC. It communicates with the rest of the LTE system through three different interfaces: E-UTRAN Uu, S1 and X2.

- E-UTRAN Uu interface is also called e-Uu, is the one responsible for the transfer of information over the radio channel between the eNodeB and the UE.
- S1 interface is the one responsible for the connectivity with the core network and in fact, it is actually split into two:
  - S1-AP to support the control plane
  - S1-U to support the user plane
- X2 interface is the one that interconnects the different eNodeB, exchanging both signaling messages intended to enable more efficient management of the use of radio resources (such as information to reduce interference between eNodeBs) as well as traffic from system users when they move from one eNodeB to another during a handover process.

Moreover, the control and user plane are important features in the organization of protocols towers associated with LTE network interfaces. In fact, the control plane refers to the protocol tower required to support the different functions and procedures necessary to manage the operation of such interface or the corresponding entity. And the user plane refers to the protocol tower employed for sending user traffic over such interface.

<u>Core Network</u>

The Core Network [5] conform the EPC and it is composed by four different entities: Mobility Management Entity (MME), Home Subscriber Server (HSS), Serving Gateway (S-GW) and PDN Gateway (P-GW), and all of them are responsible for the IP connectivity of the different UEs that are connected to the Radio Access Network and also to the external networks.

- MME: The MME is the main element of the EPC to manage the access of UEs connected to the E-UTRAN. All terminals that are registered and connected to the 4G network through the RAN have an assigned MME entity. The choice of this MME entity is made in the registration process and is based on a set of aspects such as geographic location as well as load balancing criteria. The MME entity assigned to a user can be changing and depending on the user's mobility within the network service area. It communicates with the eNodeB throw the interface S1-AP.
  Its main functions are:
  - Authentication and authorization of user access through E-UTRAN
  - Management of EPS bearer services
  - Management of user mobility in idle mode
  - Signaling for mobility support between EPS and 3GPP networks
  - Termination of NAS signaling protocols
- HSS: The HSS is the entity that acts as the main database of the 3GPP system and stores network user information. This information is based on the user's subscription data and the information necessary for the network operation itself. This database can be queried and modified by the other network entities in charge of providing connectivity services or end services. In order to access the database, it is necessary to do it from the MME network and through the S6a interface, which its traffic carries subscriber and APN-related data.
- S-GW: The S-GW is the entity that acts as the user plane gateway between E-UTRAN and the EPC backbone throw the GTP-U (also called S1-U) interface. As with the MME entity, users who are registered in the 4G network also have an S-GW entity assigned in the EPC through which their user plane passes. It is also assigned with geographical criteria as well as load balancing. Among its main functions are:
  - S-GW is responsible for providing an anchor point in the EPC backbone with respect to terminal mobility between eNodeBs.
  - The anchor point functionality also applies to mobility management with the legacy 3GPP access networks legacy (3G and 2G RAN).
  - Temporary storage of user IP packets in case the terminals are in idle mode.
  - Routing of user traffic.
- P-GW: The P-GW is the entity in charge of providing connectivity between the 4G network and external networks (which in 3GPP specifications are called Packet Data Network (PDN)), which means that through this entity, users registered and connected in an LTE network become visible in an external network. Therefore, all IP packets generated by users are transported to the external network through this gateway and vice versa.
  Each user is assigned at least one P-GW gateway to transport the packets from its registration in the LTE network, and among its main functions are:
  - Application of the rules of use of the and rate control to the bearer services that the terminal has established.
  - The assignment of the IP address of a terminal used in a specific external network is done from the corresponding P-GW gateway and throw the interface S5/S8.

- The P-GW gateway acts as an anchor point for mobility management between LTE and non-3GPP networks.
- P traffic passing through the P-GW gateway is processed through a set of filters that associate each IP packet with the corresponding user and EPS bearer service.

Nowadays this 4th generation of mobile communications networks is still in use [1], and in fact, it will be around for at least 10 more years, so 5G will not make LTE obsolete in any time soon.

2.2. Evolution towards 5G and its Network Architecture

The fifth generation of mobile communications networks is the last generation of the 3GPP technologies after the 4G. Its standards have been introduced in the 3GPP Realise 15.

Before its release, previous generations wanted to offer fast and reliable communications data services to the different users in the network, and with 5G networks, it is also offer to the end user a wide variety of wireless services provided through multiple access platforms and multi-layer networks.

In terms of 3GPP, it defines two different deployment architectures for 5G as can be seen in the figure 4:

Non-standalone (NSA) network architecture

The NSA 5G Network Architecture [6] takes advantage of the existing LTE infrastructure to deploy a 5G service. 5G Radio Access Network (RAN) and its New Radio (NR) interface is deployed with the 4G RAN and 4G CN infrastructure.

So, by introducing a 5G NR, it allows increase bandwidth capacity and network throughput respect to 4G networks, and to have greater flexibility in the functions of the user plan provided by the gateways (S-GW and P-GW) of the Evolved Packet Core. In contrary, NSA networks are limited to what is offered by the LTE network, in fact, there are several functionalities that are only available for 4G (network slicing, QoS treatment, flexibility in edge computing, and the general extensibility of the 5G core).

Regarding its architecture, the 5G NR is called Next Generation NodeB (en-gNB). This base station is linked to the eNodeB through the interface X2, which it is also used for communications between NodeBs. And the communication between en-gNBs is done through the X2-U interface. Moreover, 5G NR need the control plane of a 4G network for control functions. NSA introduced the separation between the control plans and data in the gateways.

Standalone (SA) network architecture

In the SA Network Architecture [7], the LTE Core is replaced for a proper 5G Core, which together with the gNB, the SA architectures consolidates a 5G network. With the introduction of the 5G Core Network, the data and control plans are totally decoupled, and it allows flexible and stateless positioning of virtual environments in the different network segments.

5G core architecture is exposed in the figure 5, and can be divided into three groups depending where are they running: control plane with an equivalent part in the EPC, control plane without an equivalent part in the EPC and user plane.

Control plane with an equivalent part in the EPC:

- AMF (Access and Mobility Management Function): It oversees all the connection, mobility management, authentication and authorization of access and the different location services. It replaces the EPC's MME, managing the mobility aspects.
- SMF (Session Management Function): It is in charge of the management of the sessions of the different UEs (IP assignment, selection of the associated UP function, QoS control and UP routing control). It replaces a small part of the MME and the SW-C control part of the EPC.
- PCF (Policy Control Function): It oversees controlling the policy rules that the PC functions must comply with. It replaces the PCRF of the EPC.
- UDM (Unified Data Management): It is in charge of the Unified Data Management, that is, the user's identity. It replaces a part of the HSS of the EPC.
- AUSF (Authentication Server Function): The essential part of the authentication server function. It replaces a part of the HSS of the EPC.

Control plane without an equivalent part in the EPC:

- SDSF (Structured Data Storage Network Function): It is used to store the different structured data.
- UDSF (Unstructured Data Storage Network Function): It is used to store the different unstructured data.
- NEF (Network Exposure Function): Used to help expose selected capabilities to third party services.
- NRF (NF Repository Function): Used for the discovery of available services.
- NSSF (Network Slicing Selector Function): It is used to select a different quotes for give a service to a UE.

User plane:

- UPF (User Plane Function): Is responsible for the traffic between the RAN and the internet, moreover, it is also responsible for the policy enforcement, the QoS policing and more. It replaces the P-GW in the EPC.

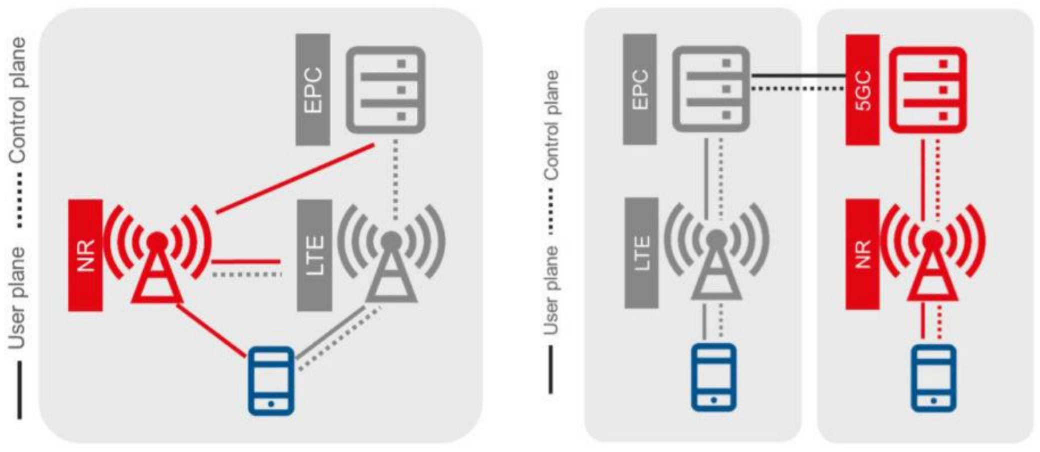


Figure 4: NSA network architecture on the left, SA network architecture on the right

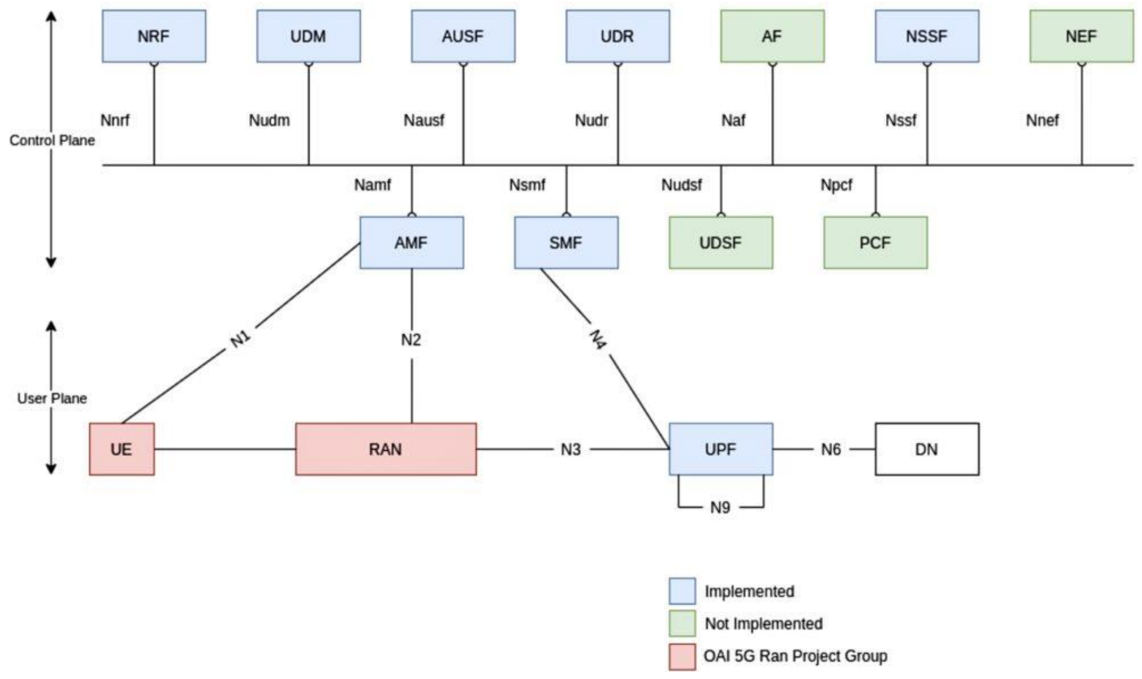


Figure 5 5G Core Network Architecture

2.3. <u>OpenAirIterface</u>

OpenAirInterface (OAI) Software Alliance [8] is a non-profit consortium founded by the French research group EURECOM. It is an ecosystem for open-source software/hardware development for the core network (EPC) and radio access network (RAN) networks.

The OpenAirInterface software implements the 3GPP stack for 4G and 5G, with all elements for deploying a the radio access network (eNB, gNB, 4G UE and 5G UE) and a core network (EPC and 5G-CN), both distributed under separate licenses.

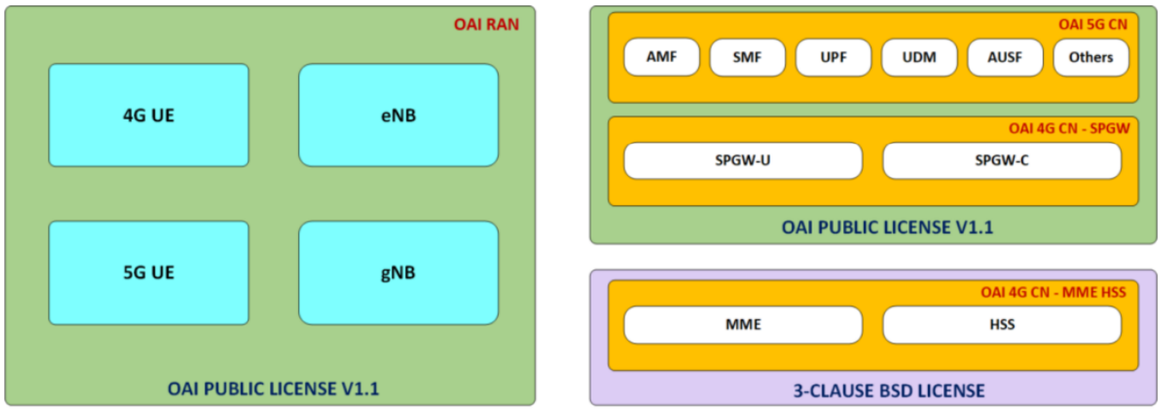


Figure 6: OAI License Model

Currently, the OAI software has a fully functional 4G network, and a 5G network which it is under development. It is written in C and under Linux optimized for x86. For the 4G part, it provides the following features [9]:

- LTE release 8.6 compliant, and implements a subset of release 10.
- FDD and TDD configurations (5, 10, and 20 MHz bandwidth).
- SISO and MIMO transmission.

- DL supported channels: PSS, SSS, PBCH, PCFICH, PHICH, PDCCH, PDSCH, PMCH.
- UL supported channels: : PRACH, PUSCH, PUCCH, SRS, DRS.
- Implements the MAC, RLC, PDCP and RRC layers.
- HARQ support (UL and DL).
- An optimized base band processing (which includes turbo decoder)

And for the 5G, it will provide the following features [9]:

- Static TDD and FDD configurations (10, 20, 40, 80, 100MHz bandwidth).
- Support NSA and SA modes.
- Intermediate downlink and uplink frequencies to interface with IF equipment.
- DL supported channels: NR-PSS, NR-SSS, NR-PBCH, NR-PDCCH, NR-PDSCH.
- UL supported channels: NR-PSS, NR-SSS, NR-PBCH, NR-PDCCH, NR-PDSCH.
- LDPC encoder and decoder (BG1 and BG2 supported).
- Polar encoder and decoder.

Moreover, OpenAirInterface use specific hardware RF modules to deploy a network. The supported modules are:

- USRP B210 over USB3 port, which has been used in this project to deploy the RAN network (eNB/gNB). All its specifications are placed in the annexed part 5, but as a summary:
    - 56 MHz bandwidth
    - Full duplex
    - MIMO 2X2
    - USB3.0.
- USRP X310 over USB3 port.
- BladeRF over USB3 port.
- LimeSDR over USB3 port.
- EURECOM EXPRESSMIMO2 PCIe card requiring a PC with a free 8/16-way PCIe slot.

2.3.1. FlexRAN

FlexRAN [10] is a real-time RAN controller which is also part of the OpenAirInterface Software Alliance, specifically, it is part of the Mosaic5G PROJECT GROUP which it is an ecosystem of opensource platforms and use-cases for 4G and 5G systems.

FlexRAN platform [11] works for 4G networks and is divided into two main components: FlexRAN service and control plane and FlexRAN Application plane. The first one, is composed by a real time RAN controller that connects and manage several underlying RAN runtime, one for each 4G base station. As seen in the figure 7, FlexRAN Control protocol which is in charge of the communication between the Real-Time Controller of the Control Plane and the RAN agent embedded in runtime environment.

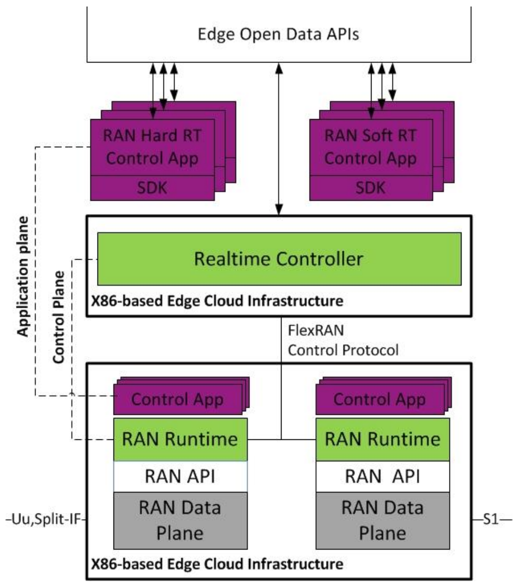


Figure 7: FlexRAN architecture

Between its features, we can highlight 4 main ideas:

- RAN Control & Data Plane Separation:
  Separating the control and data plane brings many benefits. The first one, it reduces the complexity of the system, and the second one, it allows the different third-parties authorized by the different operators to deploy different and innovative applications and services in the RAN.
- Centralized & Real-time Control:
  Having a centralized and real-time controller allows to easily coordinate, monitor, and manage all the different eNB that are connected to the RAN controller.
- Abstraction and Virtualized Control Functions:
  In order to control the RAN infrastructure, FlexRAN introduces a RAN API and a Virtualized Control Functions to perform different control operations in the base station.
- Control Delegation & Policy Reconfiguration:
  Delegating the different mechanisms of the virtualized control functions, such as schedulers or mobility managers, from the master controller to the base stations at runtime, makes the system very flexible to the underlying networking conditions and to the different parameters from the operator.

Moreover, the number of use-cases to implement are huge, but the two more important for this project are:

- RAN Optimization:
  FlexRAN can be used to manage and change the different parameters of a RAN network such as the spectrum sharing, the network slicing or the core parameters.
- Monitor RAN infrastructure:
  FlexRAN can also be used to monitor all the different information from all the eNodeBs and UEs connected to the RAN controller.

2.4. <u>NetConf</u>

NETCONF [12] stands for network configuration, and it is a network management protocol.

It is based on a server-client model on which the device that needs to be configured act as a server and the clients use XML formatted messages to communicate the configuration operations the device must perform. Moreover, these messages use remote procedure calls (RPC), the clients request goes inside an XML <rpc> element, and the server response inside a <rpc-reply>.

Netconf does not offer a generic API to manipulate the devices but instead the device should be modeled and implement the operations that can be performed such as change the IP address of an interface or set the device hostname.

The model is written in YANG, a modeling language like JSON in style but closer to YAML in content. The model defines the different variables and container that the device either write information on (operational data) or the client can modify and set values into (configuration)

One of the main attributes of NETCONF is its ability to use different databases. Most engineers are familiar with running-config and start-up. NETCONF uses a third data store called candidate configuration. The candidate configuration data store contains configuration objects that have not yet been applied to the device.

The NETCONF protocol is divided into four different layers:

- The Content layer: it includes the configuration data and notification data.
- The Operations layer: it defines a set of base protocol operations to retrieve and edit the configuration data, all the operations are listed at the end.
- The Messages layer: is responsible for providing the mechanism for encoding remote procedure calls (RPCs) and notifications.
- The Secure Transport layer: is the one responsible for the security and the reliability of the messages that goes between the client and server.

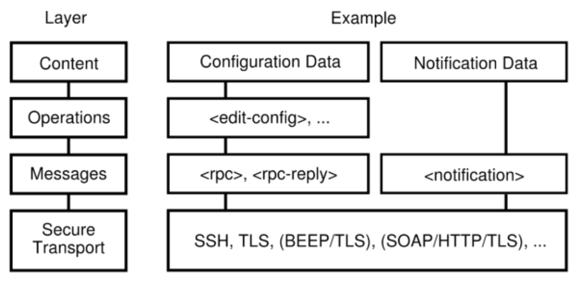


Figure 8: Netconf layers and examples

NETCONF is commonly implemented using SSH as the transport.

Its requirements are summarized next:

- It must be a connection-oriented session and, therefore, there must be a constant connection between a client and a server.

- NETCONF sessions must provide a means of authentication, data integrity, confidentiality.

- Although NETCONF can be implemented with other transport protocols, each implementation must support SSH as a minimum.

The Netconf protocol defines and implements the following operations by default:

- edit-config

- get-config
- get
- rpc
- get-schemas
- lock
- unlock
- commit
- connect
- disconnect
- subscribe
- unsubscribe

3. Methodology

The purpose of this work is to implement a non-RT RAN for automation service deployment over 4G and 5G small cells based on OpenAirInterface technology. The methodology section can be divided into two parts, first of all the OAI LTE and 5G deployment study, and then the description of the developed scenario for the implementation. As said in the introduction, the non-rt RAN has only been deployed for 4G networks, as for 5G networks it hasn't been possible due to many different bugs from OpenAirInterface.

3.1. OpenAirInerface LTE and 5G deployment study

In order to deploy the LTE network and the 5G network, a Wireless technology platform of OpenAirInterface (OAI) has been implemented. With its open-source software-based implementation of the LTE and 5G systems, it is possible to spanning the full protocol stack of 3GPP standard both in E-UTRAN and EPC.

Regarding the LTE system, the OAI open-source software can be used to build and customize base stations (OAI eNB), core networks (OAI EPC) and a user equipment (OAI UE). It is also possible to connect the OAI eNB to commercial UEs, which it is what has been done in this project, as it provides a better implementation to test different configurations and network setups and monitor the network and also the UE in real-time.

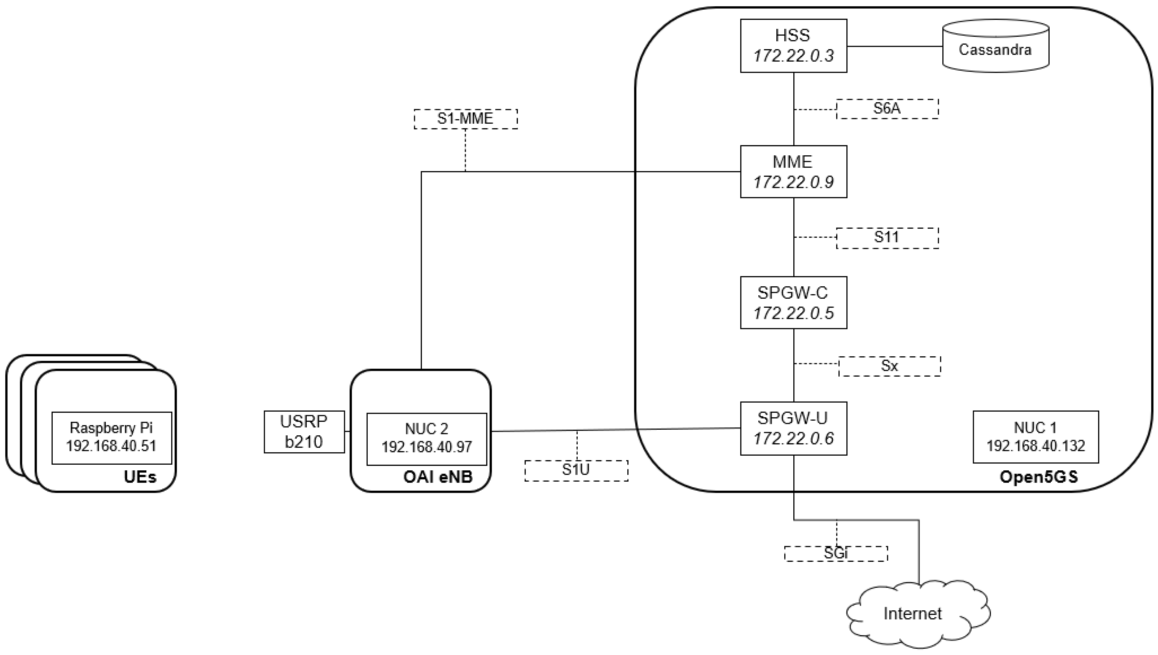


Figure 9: OpenAirInterface LTE software overall scenario

Figure 9 shows the schematic of the implemented LTE scenario in i2cat laboratory. The OpenAirInterface software has been used to deploy *OAI eNB*. The core network has been done using *OAI EPC* and Open5GS [13] including the MME, HSS and SP-GW . And regarding the UE, instead of deploying it with OAI technology, it has been used two different devices, a Quectel and a Simcom modules UEs.

Both OAI eNB and Open5Gs are running on top of NUC servers, and these are connected to a USRP, and the UEs have been running inside a Raspberry Pi and controller with the minicom software.

Regarding the 5G system, the OAI software has been used for the study of the implementation using a OAI gNB and OAI EPC NSA. As in LTE, the UEs UE, instead of deploying it with OAI technology, it has been used a smartphone 5G device. As said, the implementation of 5G in this project has not been possible due to many OpenAirInterface bugs.

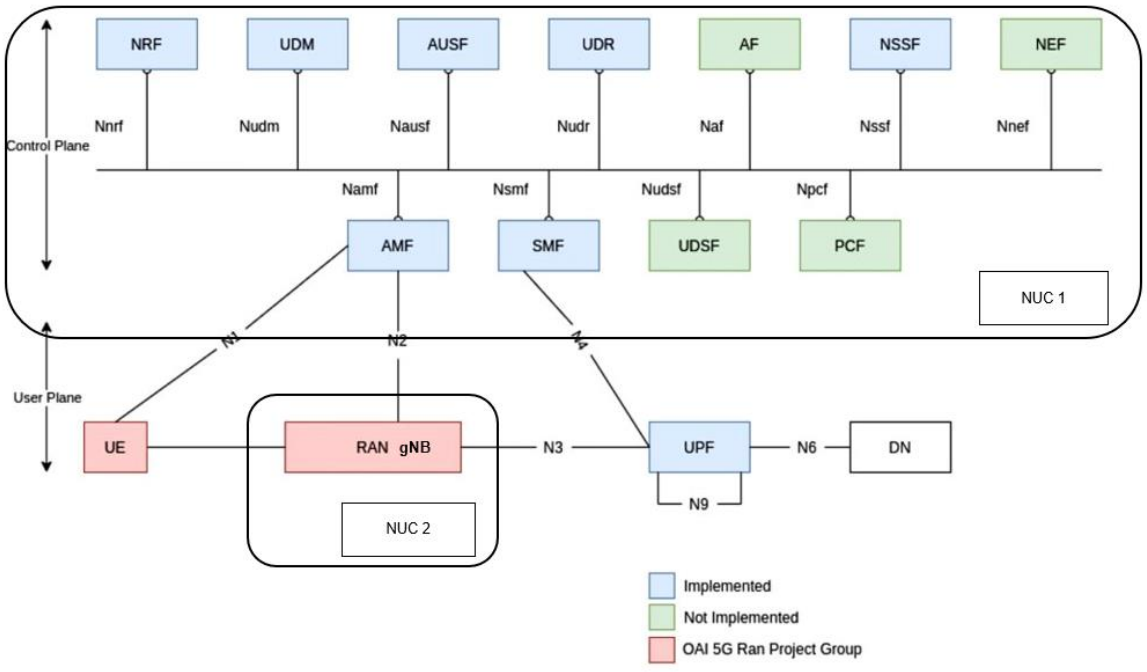


Figure 10: OpenAirInterface 5G software overall scenario

3.2. <u>Scenario description</u>

The main idea of the project, is to automate the deployment and configuration of *N* OAI eNB, including RAN slices. To do so, a full scenario has been developed in order to control and monitor all the relevant parameters.

Three different i2cat tools have implemented for OAI: Racoon, Netconf-Manager and Netconf-Server. These three tools, before doing this project there were implementations for other technologies such as Amarisoft or Accellerant, and now, after this work, for OAI technology. Moreover, it has also been developed new tools, specifically for OpenAirInterface software: oai API and oai exporter.

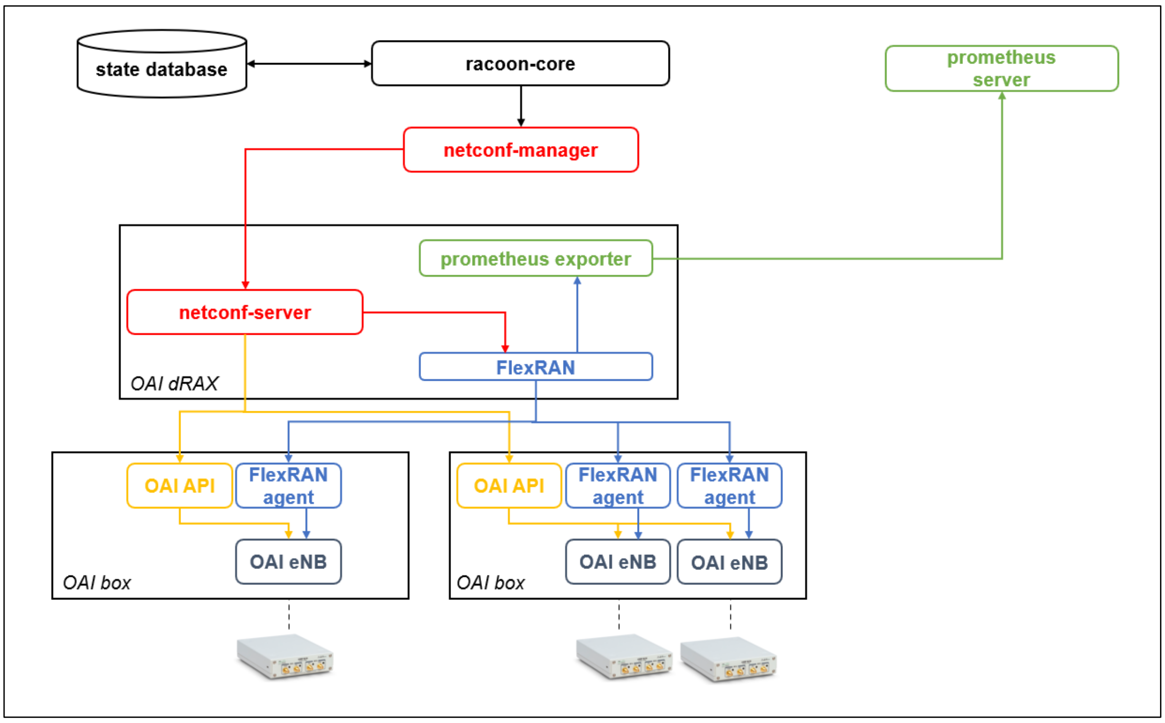


Figure 11: RACOON & OAI deployment scenario

The above figure corresponds to the developed software scenario. In order to understand it, a description of each element is exposed next, as well as some definitions that are important to understand this scenario.

Racoon-core

RACOON (RAN Controller Over OpenDaylight and NETCONF) is i2cat's RAN (Radio Access Network) controller. Before the development of this work, it supported different devices:

- i2cat's customized PC Engines APU4 Wi-Fi devices (dual band 2.4-5MHZ)
- i2cat's customized Gateworks Ventana/Newport Wi-Fi devices (dual band 2.4-5 MHz)
- Accelleran's E100 4G small cells
- Amarisoft's Callbox Pro (Supports 4G, 5G SA and 5G NSA)
- i2cat's customized PC Engines APU 802.11p V2X devices (High spectrum from 5GHz band)

And now, with this work, it has been added to the list the OpenAirInterface 4G small cells and in a near future, when OpenAirInterface have a valid and fully functional 5G network, OAI 5G small cells.

It is programmed in Java from scratch; uses the Spring Framework to create a REST API capable of registering, configuring, and deploying network infrastructure on different technologies.

More in depth, racoon-core is the most important part of the deployment, as from this software all the small cells are controlled. For i2cat software protection, we will only go in deep of the OAI model inside the racoon core internals, as it is what has been developed in this work.

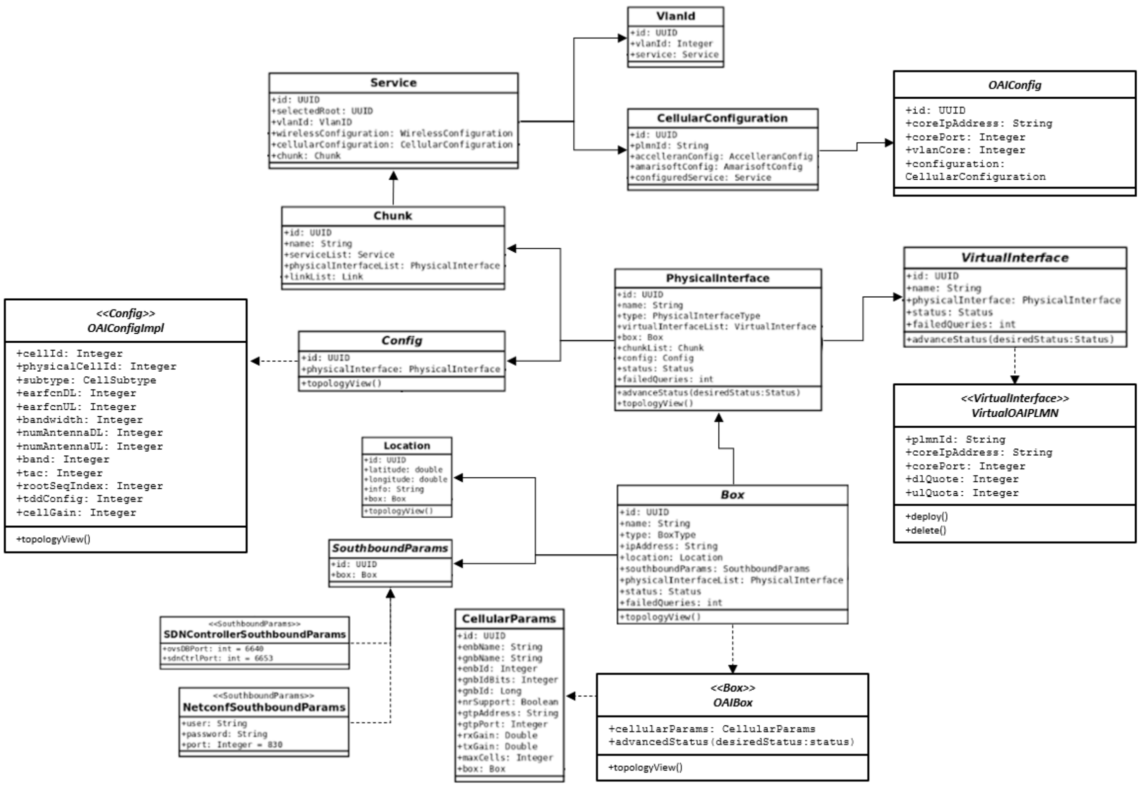


Figure 12: Racoon-core internals: Model overview

In the figure above, it is presented an overview of the racoon-core model. As seen, in this model we can find different internal elements that participate in the creation and management of the different OAI cells. In fact, we can distinguish four different internal elements:

Box:

- The box model represents a physical device that is being managed by Racoon.
- The physical devices are NUCs or servers where the desired network is willing to be deployed.
- To register a device, it has to be reachable from RACOON's southbound and the user has to provide the necessary credentials during the registration process in order to establish a NETCONF connection to the node.
- The registration process automatically discovers the device's present physical interfaces.
- Boxes can be interpreted as a container of physical interfaces.

- Boxes can be deleted; by deleting a box the connection between the device and RACOON's southbounds is not interrupted and any configuration given to the device prior the delete process will still be present after the process finishes.
- OAIBox is an extension of Box, it includes all the Cellular Parameters from the box, and it developed in a NUC where the OAI network is placed.

Physical Interface:

- A Physical interface represents a physical OAI 4G cell; generically it can be anything that can be described as a "network equipment" (Wi-Fi radio card, 4G cell, ethernet port…).
- Usually, physical interfaces can't be created; as mentioned before they are automatically discovered and populated during the box registration process; but with OpenAirInterface technology, the 4G OAI cells can be created on demand (up to the cell limit for the OAI box, in our case, limited to two cells per box).
- Configuration can (it's expected to) be provided before starting to use the physical interface.
- OAI cells physical interfaces can also be deleted.

Chunk:

- A chunk is the definition of a subset of the topology on which a client or tenant will be allowed to deploy services on.
- To create a chunk, the user has to select which physical interfaces, and which backhaul links (if any) the user will be allowed to deploy services onto.
- Chunks can be created and destroyed on demand but it's not possible to delete a chunk that has services running on top of it.
- Creating and deleting a chunk does not impact the devices nor stores any data on them; RACOON is the only responsible for the chunk administration and storing on the database.

Services:

- Racoon services are responsible for deploying the provided configuration on the physical equipment; configure a cell to start radiating a certain PLMN or UL and DL quotes.
- The user specifies on which physical interfaces (from the user's chunk) will the service be deployed into and the configuration parameters for each technology selected.
- Services can be created and deleted on demand.

Netconf-Manager

As explained in the state of art, NetConf is a network management protocol based on a server-client model on which the device that will be configured act as a server and the clients use XML formatted messages to communicate the configuration operations the device has to perform. The model is written in YANG, a modeling language similar to JSON in style but closer to YAML in content. The model defines the different variables and container that the device either write information on (operational data) or the client can modify and set values into (configuration).

Therefore, Netconf-Manager is a spring-boot java application that acts as a Netconf client and translates from REST API calls to Netconf RPC (Remote Procedure Call) operations in order to retrieve or set configurations from a Netconf device.

Netconf-Manager defines the endpoints that racoon will use to either create virtual interfaces or set the channel configuration to a physical interface.

Internally uses a Juniper java module (Licensed as BSD2) that handles the connection and credentials exchange between the server and the client and the main business logic is implemented on a separate java module.

The REST API calls that are defined, communicate with NetConf servers without needing to implement the Netconf protocol on the RAN controller's side.

Netconf server

The Netconf-server is equivalent to a neetopeer 2 agent, it provides an implementation to the Netconf RPCs defined by the Netconf protocol. It uses LibSSH as the server's crypto library to encrypt the messages between the client and the server, which means that the client needs to authenticate with the server before any operations execution.

Moreover, netconf-server act as the transAPI for the model.

This API is writted on C, and before this work, it managed three different physical devices:

- i2cat-box API: Handles very basic information such as the device's dependencies, the hostname, memory / cpu usage,...
- wired API: Capable of providing ip addresses to wired interfaces present on the system, creating new 8021q interfaces, creating GRE interfaces,...
- wireless API: Administers the device's Wi-Fi interfaces present on the system. The API is capable of instantiating virtual access points using Hostapd, creating and joining 80211s mesh networks,...

And with the development of this work, it can manage the OpenAirInterface network. To do so, Netconf-Server communicates with two different elements. Firstly, in order to deploy and undeploy OAI eNB and to configure IP addresses to wired interfaces present on the system (creating new 8021q interfaces for example) it communicates with a OAI API which are present on each box, secondly, Netconf performs the RAN configurations via FlexRAN to configure the different cell parameters. As a summary Netconf server performs the following tasks:

Via OAI API:

- Deploy OAI cell
- Stop OAI cell

Via FlexRAN:

- Add a new PLMN id
- Remove a PLMN id
- Create RAN slice
- Modify an existing RAN slice
- Disable RAN slicing

It is important to understand why the NetConf server communicates with these two elements separately. First of all, the OAI API, as can be seen in the figure 11, it is build inside the box, which means that, for every box, there will be a OAI API that will deploy and stop the eNodeB cells inside these boxes, and to configure the wired interfaces present in the box. Secondly, the FlexRAN software is build together with the Netconf-server and a OAI exporter inside the dRAX, as the real-time controller FlexRAN can manage and communicate with all the deployed eNBs.

Regarding the OAI dRAX element, it is an imaginary and central element, that provides communication with all the boxes and cells. It includes the netconf server, the FlexRAN and the Prometheus exporter.

Prometheus exporter

Apart of performing RAN configurations, FlexRAN also allows to monitor the state of RAN infrastructure (including both cells and UEs). To have a better look in all the FlexRAN monitoring information, it has been developed an exporter written with python and using a Prometheus database and Grafana exporter [14] to present all the relevant information of the RAN infrastructure.

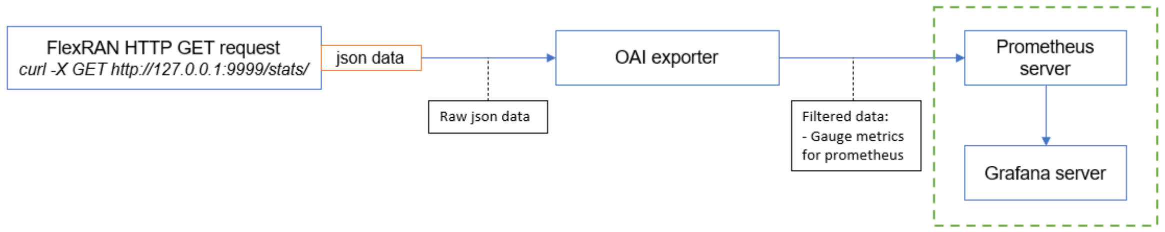


Figure 13: OAI exporter diagram

As can be seen, the information arrives to the OAI exporter with a cURL API call to FlexRAN and respond it is a json file with all the eNBs and UEs information.

The OAI exporter, filters the important information that needs to be monitored, and stores it into a Prometheus server. To store the data into the database, it has been used the python Prometheus client libraries which offer four core metric types. The first type, counter is a cumulative metric that represents a single monotonically increasing counter whose value can only increase or be reset to zero on restart, the second type, gauge is a metric that represents a single numerical value that can arbitrarily go up and down, the third, histogram samples observations (usually things like request durations or response sizes) and counts them in configurable buckets, and the last type, summary, which is similar to a histogram, a summary samples observations (usually things like request durations and response sizes) and it also provides a total count of observations and a sum of all observed values, it calculates configurable quantiles over a sliding time window. As all the monitored metrics can go up and down, the main type used for the metrics have been gauge.

Once the metrics are stored into the prometheus server, they are exposed into a Grafana dashboard, where there is the possibility to filter by eNB or UE. All the relevant metrics and the dashboards are exposed in the results part.

OAI API

The OAI API is a REST API that has been developed specifically for OAI technology. It is written with python, and it is responsible for the deployment and the stop of the different OAI cells, as well as the configuration of the wired (ethernet) interfaces present in the box.

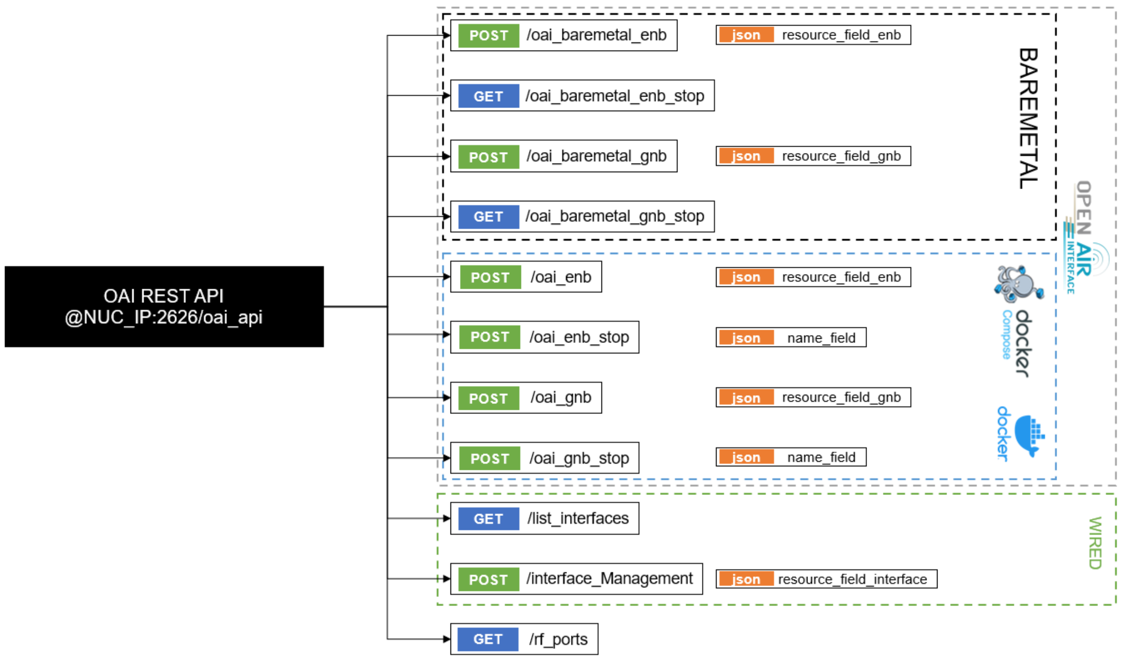


Figure 14: Different OAI API calls

Figure 14 shows all the developed calls in this REST API.

Moreover, in order to deploy OAI cells, there are two different calls to do it. One possibility is using OpenAirInterface binary files, with the meaning that the cell is deployed in the NUC internally, and the second option to deploy an OAI cell is by using a container, concretely a docker container to develop the cell. Moreover, the OAI API can deploy both OAI eNB and also gNB, this is because, even though FlexRAN can still not control 5G networks, in this work it has already been prepared the scenario of 5G, using both BareMetal and Docker Container, for future implementations.

In order to deploy an eNB or gNB using binary files/baremetal, OAI API requests a json file with the cell main parameters for its configuration, and to stop it, it does not need any parameter as a NUC only allows to deploy one eNB or gNB per OAI box. In addition to the baremetal, docker also requests a json file with the cell main parameters for its deployment, and as it is possible to have multiple containers and therefore multiple OAI cells, in fact, in order to stop them, the OAI API requests the name of the cell.

Apart from deploying OAI cells, the REST API also allows to return the different USRPs inside a box.

Finally, there are two other calls to manage the wired (ethernet) interfaces with two different calls, the first one, /list_interfaces returns a list with all the available interfaces in each box, and also, the second one /interface_Management allows to add and delete a vlan, also to set an interface up and down and to set/change the interfaces IP.

This OAI API, is placed inside each OAI box, as said before, this is because it needs superuser permissions to apply configurations to manages its cells and interfaces from that box. In this MSc Thesis, i2CAT provided two NUCs as OAI boxes, an OAI box is a physical machine which can be a NUC (Next Unit of Computing) or a server. In this project i2cat has provided Open-VERSO NUCs, which are Linux machines with a real-time kernel. And each of them contains the OAI API, the FlexRAN agent which is responsible for the configuration of an specific cell inside that box, and finally the OAI eNB deployed together with the USRPs B210 (the limitation in the number of USRPs is the number of USB3 ports in the host, in our case, NUCs only have 2 USB3 ports.)

4. Results

This chapter aims to provide the different analysis and configuration of the OpenAirInterface network as well as the different results and tests performed in the developed platform.

4.1. Lab scenario

First of all, let's present the scenario and the laboratory where the project has been carried out.

The scenario that has been implemented is composed by two different NUCs in which the OAI network is placed, in NUC1, the core network with all its components, and in NUC2, the OAI eNB with the USRP b210. Also a Raspberry Pi which act as a UE and also a Raspberry Pi with two different Quectel and Simcom modules. The communication between these two NUCs is done throw the interface eno1, it is the onboard Ethernet (wired) adapter, which it is the most reliable interface to manage the communication between both NUCs as they are both placed in the same laboratory room.

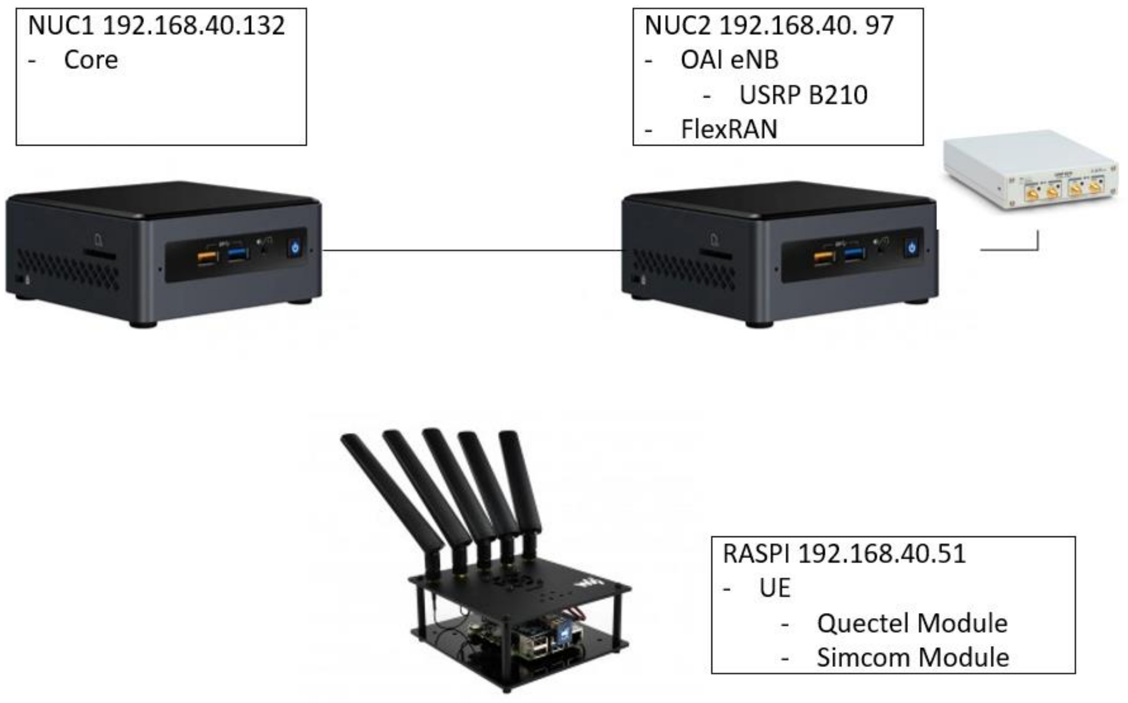


Figure 15: OpenAirInterface scenario

The next image, shows the workspace of this scenario, which is placed in the i2cat laboratory, located in the Nexus building next to the ETSETB school in Barcelona.

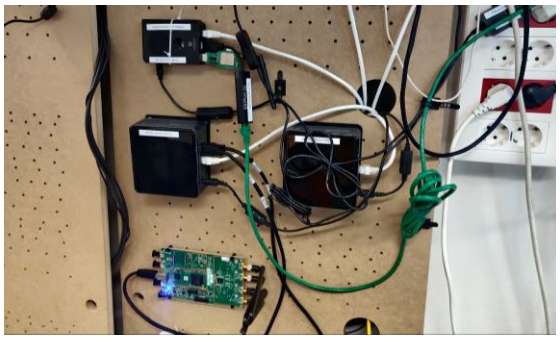


Figure 16: Work Space with the two NUCs and the USRP.

4.2. Network configuration

To understand comprehensively the network architecture and configuration of each machine, in the figure 17 is presented a diagram with all the different details of the IP addresses and interfaces of each entity in NUC1, NUC2 and the Raspberry Pi.

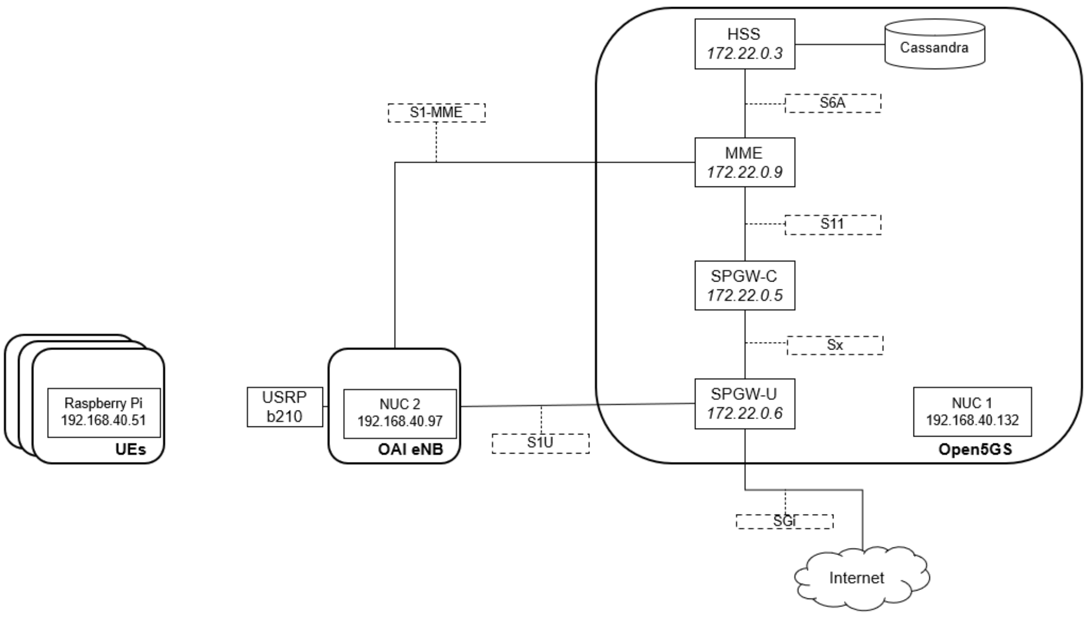


Figure 17: OAI Network Diagram

NUC1 NETWORK

The core presented in NUC1 is deployed using open5gs, a gitlab project [15] that can be used to create a EPC, specifically it can configure a series of software components and network functions that implement the 4G/ 5G NSA and 5G SA core functions.

For the implementation of the LTE network, it has been used the 4G/5G NSA core, which contains the following components:

- MME - Mobility Management Entity
- HSS - Home Subscriber Server
- PCRF - Policy and Charging Rules Function
- SGWC - Serving Gateway Control Plane
- SGWU - Serving Gateway User Plane

- PGWC/SMF - Packet Gateway Control Plane / (component contained in Open5GS SMF)
- PGWU/UPF - Packet Gateway User Plane / (component contained in Open5GS UPF)

As related in section 2.1 of the state of art, the core has two main planes, the control plane, and the user plane.

For the LTE network, and first regarding the control plane, the MME is the main control plane hub of the core, it is linked to the HSS throw the S6a interface and to the SGWC and PGWC throw the S11 interface. Second, the user plain is the one that carriers the user data packets between the eNB and the external WAN, and it composed by the SGWU and the PGWU which connects to the control plane throw the SGWC and the Sxa interface.

All the Open5GS components have config files, that have been used to configure each component's IP local addresses and local interfaces and the IP addresses and DNS names of the other external components.

By default, all the Open5GS components are configured for use inside a single server/PC, in fact, they communicate each other using the local loopback address space 127.0.0.X and lo interface. Therefore, some modifications have been made to stablish the communication with the RAN network in the other NUC. Next are exposed HSS, MME and SGWU config files, which are the one modified, the other config files can be consulted in Open5Gs gitlab repository [15].

MME configuration

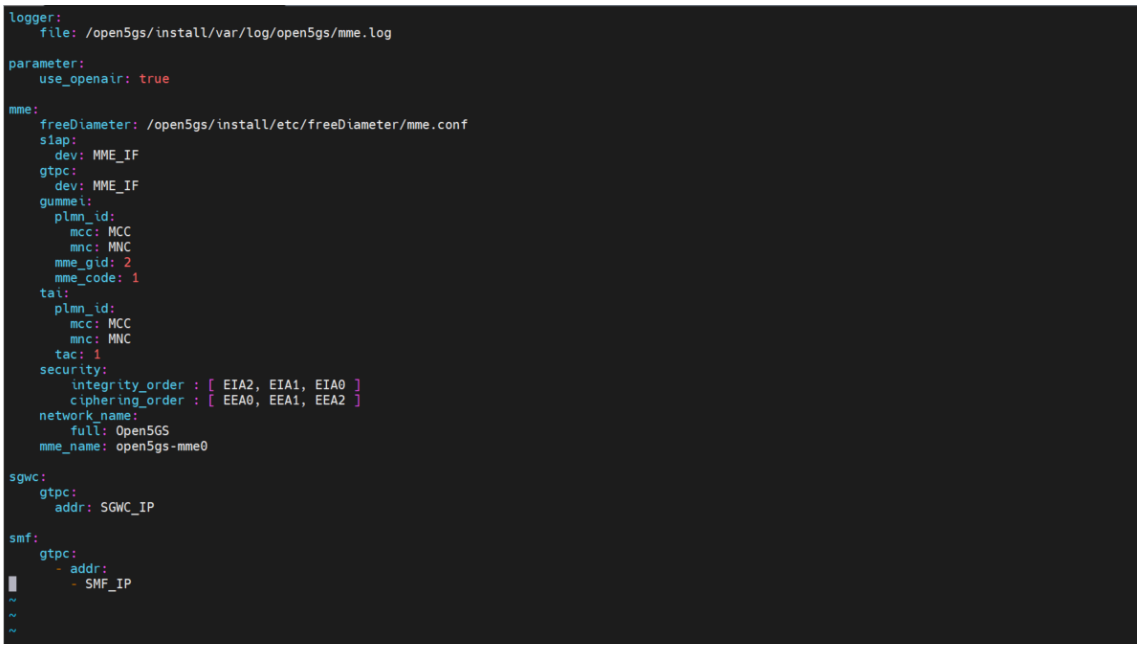
The MME config file includes the different PLMN and TAC information. This needs to be modified to match the UEs.

In the next table can be found the PLMN and TAC information related to both Quectel and Simcom modules that need to be added to the MME configuration file.

	PLMN	**TAC**
Quectel	00102	2
Simcom	00103	2

Table 2: Quectel and Simcom PLMN information

Also, as the eNB is placed in a different NUC, we need to change the S1AP bind address and interface, to 192.168.40.97 and eno1 which is the address where the eNB is placed.



```
logger:
    file: /open5gs/install/var/log/open5gs/mme.log

parameter:
    use_openair: true

mme:
    freeDiameter: /open5gs/install/etc/freeDiameter/mme.conf
    s1ap:
      dev: MME_IF
    gtpc:
      dev: MME_IF
    gummei:
      plmn_id:
        mcc: MCC
        mnc: MNC
      mme_gid: 2
      mme_code: 1
    tai:
      plmn_id:
        mcc: MCC
        mnc: MNC
      tac: 1
    security:
        integrity_order : [ EIA2, EIA1, EIA0 ]
        ciphering_order : [ EEA0, EEA1, EEA2 ]
    network_name:
        full: Open5GS
    mme_name: open5gs-mme0

sgwc:
    gtpc:
      addr: SGWC_IP

smf:
    gtpc:
      - addr:
        - SMF_IP
~
~
~
```

Figure 18: OAI MME configuration file.

In order to have a better control of the .conf files, there is a generic .env file where all the parameters are stored. The important ones for this configuration are:

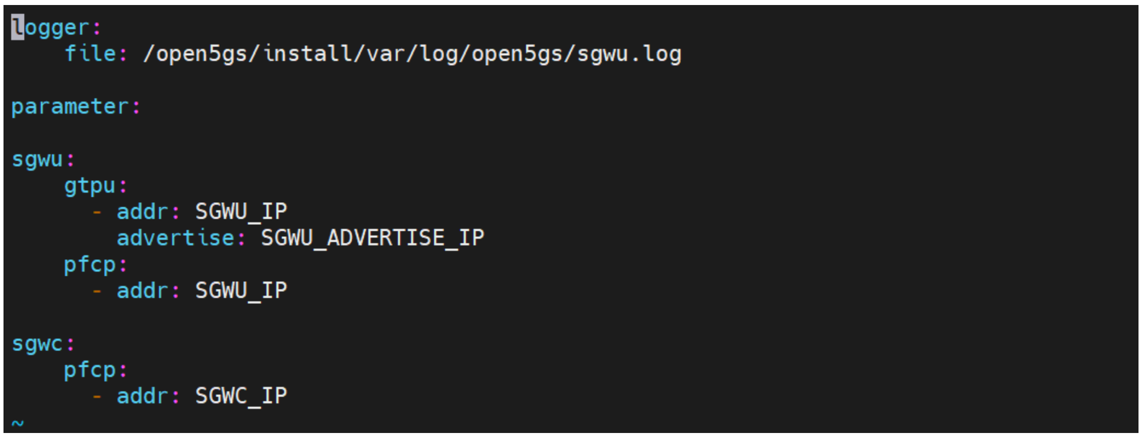
MME_IP: 192.168.40.97 (other's NUCs IP)

MCC: 001 (Quectel PLMN, for example)

MNC: 02

<u>SGWU configuration</u>

The same happens with the SGWU, as the eNB is placed in a different NUC, we need to change the GTP-U bind address, to 192.168.40.97 which is the address where the eNB is placed.



```
logger:
    file: /open5gs/install/var/log/open5gs/sgwu.log

parameter:

sgwu:
    gtpu:
      - addr: SGWU_IP
        advertise: SGWU_ADVERTISE_IP
    pfcp:
      - addr: SGWU_IP

sgwc:
    pfcp:
      - addr: SGWC_IP
~
```

Figure 19: OAI SGWU configuration file with env variables.

SGWU_IP: 192.168.40.97

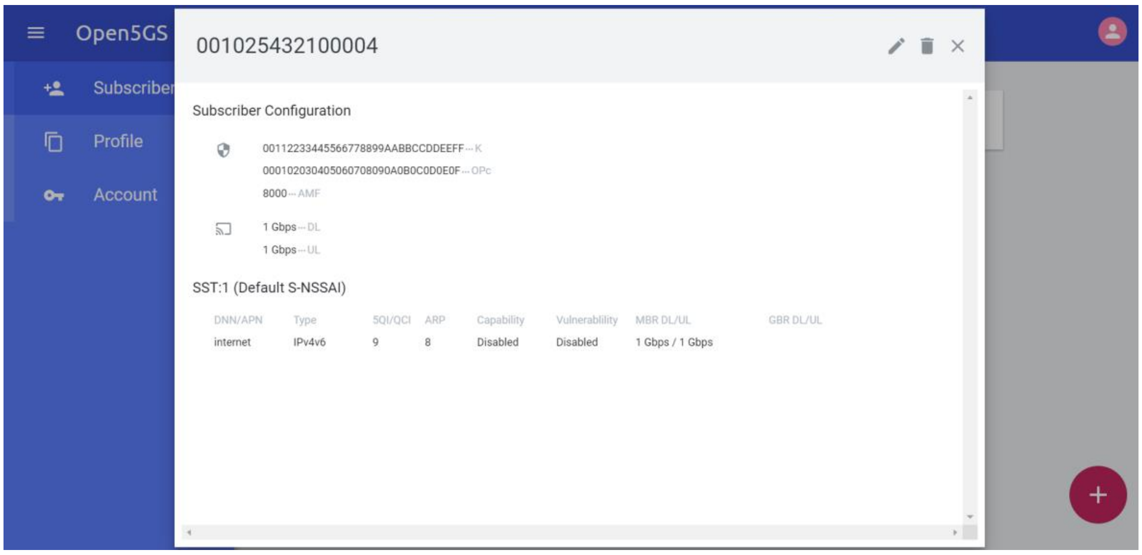
HSS configuration

The last thing to be done is to include the subscriber information into the core network. This information is exclusive for each core network, and it is related to the UEs. The different parameters that are included in the subscriber information are [22]:

- IMSI: International Mobile Subscriber Identity. It is a 15 digit unique identifying number that is used to identify the subscriber to the service. It is usually issued by the operator. IT consists on three parts: MCC Mobile Country Code (geographic region of the SIM), MNC Mobile Network Code (operator) and MSIN Mobile Subscriber Identifier (to identify individual subscribers).
- Key: Subscriber Authentication Key. It is a 128 bit field, and it is part of the Authentication Algorithm. It is placed into the USIM and also to the HSS.
- OPs: Operator Code. It is a 128 bit field and it is also part of the Authentication Algorithm. It is the same for all SIMs from a single operator.
- APN: Access Point Name. Is the gateway name between a 4G mobile network and the public Internet. Usually, each operator has its own APN.

To include this information, we need to do throw a WebUI application, which is an application part of the Open5Gs that allows you to interactively edit subscriber data. To access it, we have to connect to http://@NUC1_IP:3000 login, and follow the steps:

1. Go to Subscriber Menu.
2. Click + Button to add a new subscriber.
3. Fill the IMSI, security context(K, OPc), and APN of the subscriber.
4. Click SAVE Button



001025432100004

Subscriber Configuration

00112233445566778899AABBCCDDEEFF ··· K
000102030405060708090A0B0C0D0E0F ··· OPc
8000 ··· AMF

1 Gbps ··· DL
1 Gbps ··· UL

SST:1 (Default S-NSSAI)

DNN/APN	Type	5QI/QCI	ARP	Capability	Vulnerablility	MBR DL/UL	GBR DL/UL
internet	IPv4v6	9	8	Disabled	Disabled	1 Gbps / 1 Gbps	

Figure 20: Subscriber information from the webUI for the Quectel UE.

In the next table can be found the different Subscriber information for both Quectel and Simcom modules.

	Quectel	**SimCom**
IMSI	001025432100004	001035432000005
OPs	000102030405060708090A0B0C0D0E0F	000102030405060708090A0B0C0D0E0F

K	00112233445566778899AABBCCDDEEFF	00112233445566778899AABBCCDDEEFF
APN	internet	internet

Table 3: Subscriber information for Quectel and SimCom

Moreover, in order to deploy this open5gs core network, a docker single node deployment using docker-compose has been developed [16]. In this docker container, all the IP and interface parameters have been included in an environment file .env to centralize all the configuration, this environment file is presented in the annexed part 1.

<u>NUC2 network</u>

In the NUC 2, it can be found the OAI eNB with the USRP B210 and the FlexRAN that will act as a real-time controller.

The OAI eNB has been placed into NUC2 using two different tools, the first one and finally not useful, using docker to build and run the LTE RAN, and the second one and the one finally used in this project, running it in a binary file. In the last part of this section, it is explained why it is considered the binary/baremetal tool more reliable than the docker tool.

Moreover, in order to deploy an OAI eNB, we need to set a config file, where all the eNB parameters are settled, this .conf file is presented in the annexed part 3. In the following list are summarized all the parameters that are modified every time a 4G RAN is deployed with RACOON. Moreover, all these parameters are actually dynamic, so, they can be modified anytime to match the desired configuration (in brackets are typical values provided to these parameters).

ENB_ID – eNB id, it is part of the identification parameters of the RAN. (123)

ENB_NAME – It is also part of the identification parameters of the RAN. (i2cat)

UTRA_BAND_ID – The E-UTRA BAND id is equivalent to the EUTRA Operating Band, which it is the band where the DL ad UL frequencies are located. (in this work it has been used almost all the time the band 1, in the annexed part 4 are presented all the possible bands)

DL_FREQUENCY_IN_MHZ – As we are working with FDD (Frequecy Division Duplex) spectrum, it requires pair bands, one for downlink and another for uplink. Both frequencies are transmitted simultaneously on different frequencies. (as we are using band 1, the DL frequency used is 2150MHz)

UL_FREQUENCY_OFFSET_IN_MHZ – (the UL frequency used is 1950MHz)

NB_PRB – Number of RB (Physical Resource Blocks). (As we are using band 1 it can be 5, 10, 15 or 20)

MCC – Mobile Country Code, it consists of three decimal digits. (eg Quectel 001)

MNC – Mobile Network Code, it consists of two decimal digits. (eg Quectel 02)

MNC_LENGTH – Length of the Mobile Network Code. (2)

TAC – Tracking area. (1)

NID_CELL - Physical cell id, used to calculate the position of some reference and synchronization signals

CELL_ID – Id of the cell. (eg 1023)

ENB_S1C_IF_NAME - S1C interface name (interface between eNB and MME). (eno1 interface used)

ENB_S1U_IF_NAME - S1U interface name (interface between eNB and SGWU). (eno1 interface used)

ENB_S1C_IP_ADDRESS – S1C IP address. (192.168.40.97)

ENB_S1U_IP_ADDRESS – S1U IP address. (192.168.40.97)

ENB_X2_IF_NAME – X2 interface name (interface between eNBs). (eno1 interface used)

ENB_X2_IP_ADDRESS – X2 IP address. (192.168.40.97)

ENABLE_X2 – Boolean if X2 interface is enabled. (yes, as we expect more than 1 eNB)

FLEXRAN_ENABLED – Boolean if FlexRAN is enabled. (yes, as it is the controller)

FLEXRAN_INTERFACE_NAME – FlexRAN interface name. (eno1 interface used)

FLEXRAN_IPV4_ADDRESS – FlexRAN IP address. (192.168.40.97)

MME_S1C_IP_ADDRESS – MME core network IP address. (192.168.40.132, equivalent to NUC1 IP, where EPC is located).

SERIAL – USRP serial number. (Currently working with two different USRPs 31DB5A3 and 3150321).

N_ANTENNA_DL – Number of antennas for downlink transmission. (Only using SISO, 1)

N_ANTENNA_UL – Number of antennas for uplink transmission. (Only using SISO, 1)

ROOT_SEQ_INDEX – The Root Sequence Index allows the UE to calculate which PRACH preamble it can use to attach to the cell. It is important to have different values for neighbors cells (1)

TZ – Time Zone (Europe/Spain)

An important parameter is the USRP serial number. A USRP is a Radio Frequency fully integrated, single-board, Universal Software Radio Peripheral (USRP™) platform with continuous frequency coverage from 70 MHz – 6 GHz to run the eNB. I2cat has lend two different USRPs.

USRP B210 1	31DB5A3
USRP B210 2	3150321

Table 4: USRP serial number information

Finally, the real-time controller FlexRAN, it is used to configure OpenAirInterface cells, core networks and manage RAN slices. It is composed by two main components: the FlexRAN Service and Control Plane and the FlexRAN Application plane. The FlexRAN Service and Control Plane is based on the of the Real-Time Controller, which is connected to several underlying RAN runtime, one for each RAN module (eg. 4G eNB). The RAN runtime environment provides the separation between the control and data plane, as it acts as an abstraction layer with RAN module on one side and RTC and control apps on the other side. Moreover, the communication between the real-time controller and the RAN agent embedded in runtime environment is managed by the

FlexRAN protocol. RAN control applications can be developed both on the top of the RAN runtime and RTC SDK which allows to not only monitor but also control and coordinate the state of RAN infrastructure, all this can be done as all FlexRAN data and APIs are open to be consumed by 3rd parties. It reduces the complexity of developing new control solutions.

As seen in the figure 15, FlexRAN has been deployed in NUC2 for the purpose of being in the same NUC as the eNB RAN. In order to install it, it has been used a mosaic5g script located in its gitlab [17].

The controller runs in localhost:9999/capabilities and can be checked using a browser. Moreover, to enable the controller, it must be done into the RAN's configuration file setting the correct IP address and interface in the network _controller part:

```
NETWORK_CONTROLLER :
{
    FLEXRAN_ENABLED        = "yes";
    FLEXRAN_INTERFACE_NAME = "eno1";
    FLEXRAN_IPV4_ADDRESS   = "192.168.40.97";
    ...
};
```

In this configuration, FlexRAN is enabled, the interface is eno1 and the IP address is the NUC's IP.

As said, FlexRAN allows to monitor, control, and coordinate the state of the RAN infrastructure throw a RESTful API using simple HTTP requests. Below are summarized all the API calls used in this project.

- Get the eNB and UE configuration and statistics:
  To obtain all the agent's statistics, we use the next command:

  ```
  $ curl -X GET http://FlexRAN_PUBLIC_IPADDR:9999/stats/
  ```

  Where FlexRAN_PUBLIC_IPADDR is the public address of the NUC (192.168.40.97). This should return a json format data with all the information for all eNBs and their connected UEs.
  This API call has been used in the OAI exporter and also to do some checks in the slice, core and PLMN configurations in the netconf-server part.

- Radio Resource Management
  For radio resource management, FlexRAN has been used for slicing purposes.
  In fact, network slicing is considered a very important mechanism that allows to serve to all network clients in a flexible and cost-efficient manner.
  To configure the slicing, we use the next command:

  ```
  $ curl -X POST http://FlexRAN_PUBLIC_IPADDR:9999/slice/enb/-
  1 --data-binary @ran-sharing.json
  ```

In the FlexRAN window, the same JSON file alongside a message indicating the transmission of the configuration to the respective agent should be shown. The RAN part will equally acknowledge the setting for each parameter.
This API call has been used in the netconf-server part.

- PLMN Management
  Regarding the PLMN management, with flexRAN it is also possible to dynamically manage the PLMNs and attached core networks of a base station.
  Before adding a core network, it is needed to add the corresponding PLMN. This is because the eNB informs to the MME in the S1SetupRequest about the broadcasted PLMNs and the MME will only accept this setup request if the PLMNs the MME is serving (as sent back as servedPLMNs in the S1SetupResponse) matches the broadcasted PLMNs of the eNB.
  So, in order to add the PLMN, we use the next command:

```
$ curl -XPOST FlexRAN_PUBLIC_IPADDR:9999/plmn/enb -d
@plmn.json
```

  In the plmn.json file, there is a list of the new PLMNs.
  After adding the PLMN, it is ow possible to add and connect to a core network using the following command:

```
$ curl -XPOST FlexRAN_PUBLIC_IPADDR:9999/mme/enb/ -d
@mme.json
```

  Where in the mme.json is listed the IP address of the MME of the core network to connect to.
  Finally it is possible also to delete the core network:

```
$ curl -XDELETE FlexRAN_PUBLIC_IPADDR:9999/mme/enb/ -d
@mme.json
```

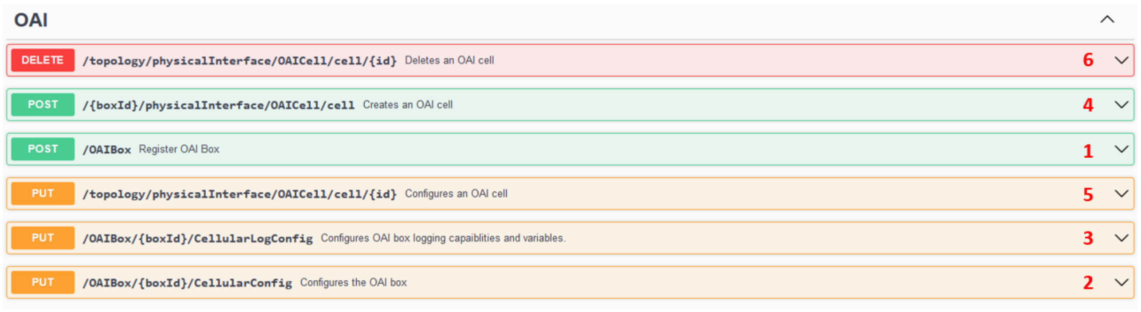
  And also the PLMN can be deleted with the same command used to add the PLMN but with a plmn.json file without the PLMN that wants to be deleted.
  This API call has been used in the netconf-server part.

4.3. RACOON SWAGGER Platform

The automated service is controlled in the RAN controller RACOON with different API calls that allows to deploy and configure the different parameters. Before doing a demo, it will be explained which are the calls that have been developed and that are necessary to deploy a service.

To visualize all the RACOON calls, it has been used a Swagger UI, which it is an open-source tool to visualize and interact with the API's resource without having any of the implementation logic in place, allowing to execute the different calls with its correspondent json configuration file if needed. Moreover, RACOON is organized by technologies, so, the calls that are used for deploying the OpenAirInterface boxs and cells are placed under the same section, and the chunk and service calls as are also used for other technologies are placed separately.

Next can be found the six different calls that corresponds to the OpenAirInterface technology and allows to register and configure an OAI box, and to register, configure and delete an OAI cell.



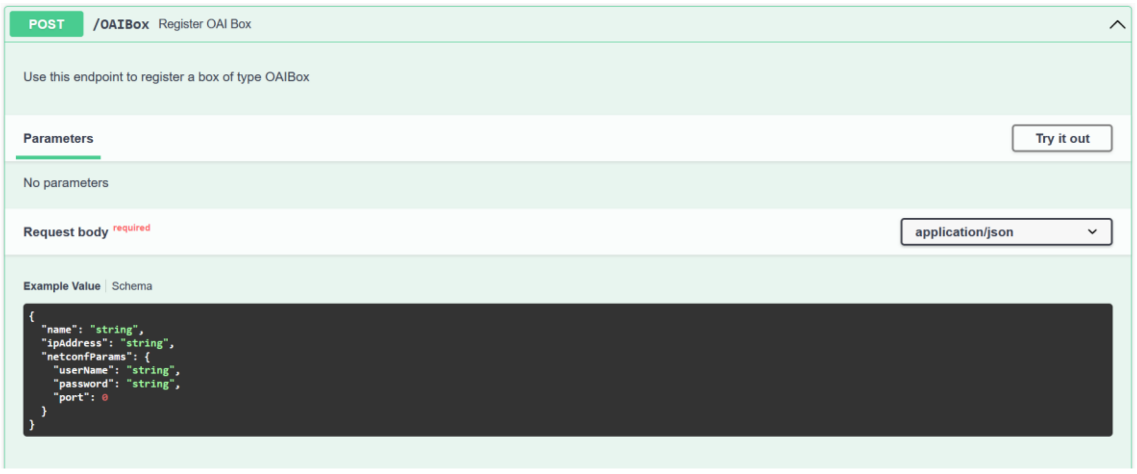
OAI

DELETE	/topology/physicalInterface/OAICell/cell/{id}	Deletes an OAI cell	6
POST	/{boxId}/physicalInterface/OAICell/cell	Creates an OAI cell	4
POST	/OAIBox	Register OAI Box	1
PUT	/topology/physicalInterface/OAICell/cell/{id}	Configures an OAI cell	5
PUT	/OAIBox/{boxId}/CellularLogConfig	Configures OAI box logging capabilities and variables.	3
PUT	/OAIBox/{boxId}/CellularConfig	Configures the OAI box	2

Figure 21: OAI technology calls

The procedure followed to deploy an OAI BOX is:

1. Register OAI Box



POST /OAIBox Register OAI Box

Use this endpoint to register a box of type OAIBox

Parameters

No parameters

Request body required

application/json

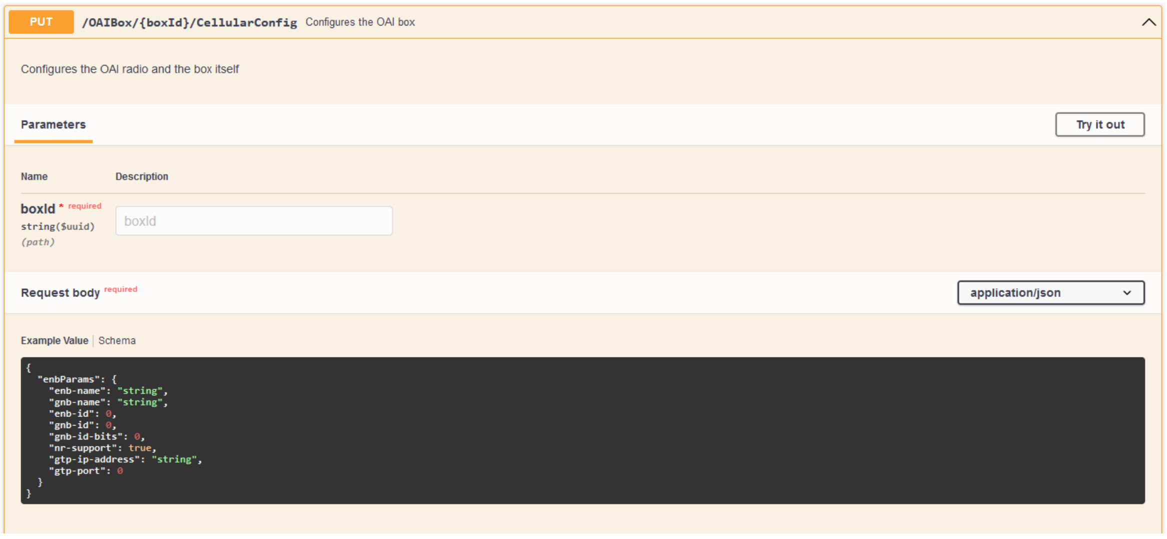
Example Value | Schema

```
{
  "name": "string",
  "ipAddress": "string",
  "netconfParams": {
    "userName": "string",
    "password": "string",
    "port": 0
  }
}
```

Figure 22: OAIBox RACOON call

The first step to deploy an OAI box is register it into RACOON, with this, we will set a json file with the name of the box, the IP address and the NetConf parameters, as Netconf client needs to authenticate with the server before any operations to configure a cell. When this call is executed, if it is correct, it will return a 200 code with a json file that includes a unique box id identifier.

2. Configure the OAI box



/OAIBox/{boxId}/CellularConfig Configures the OAI box

Configures the OAI radio and the box itself

Parameters

Name	Description
boxId * required string($uuid) (path)	boxId

Request body required

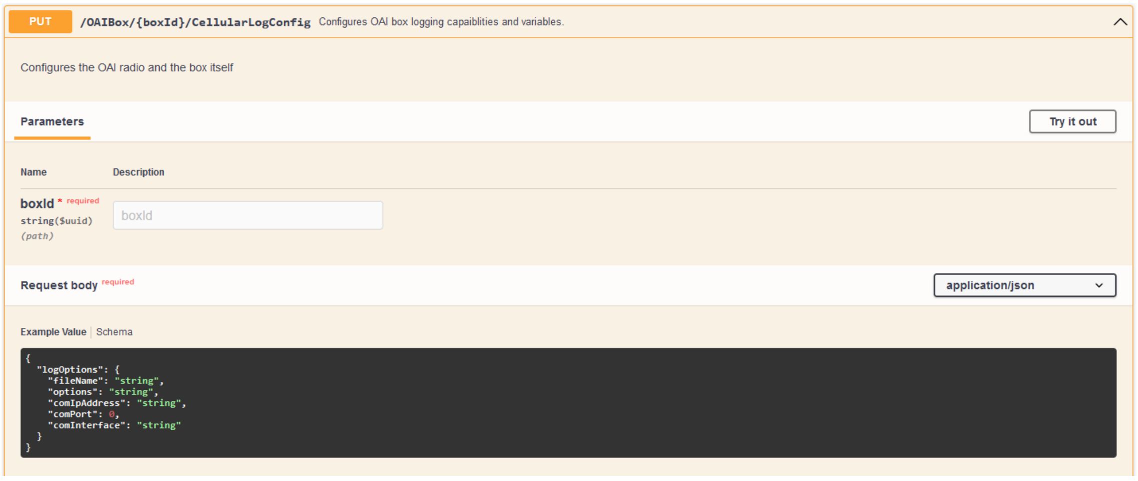
Example Value | Schema

```
{
  "enbParams": {
    "enb-name": "string",
    "gnb-name": "string",
    "enb-id": 0,
    "gnb-id": 0,
    "gnb-id-bits": 0,
    "nr-support": true,
    "gtp-ip-address": "string",
    "gtp-port": 0
  }
}
```

Figure 23: Config box RACOON call

The second step is to configure box's eNB and gNB parameters. It requires a boxid, which it has been returned in the register call, and a json file with the eNB and gNB main parameters and the GTP IP address as the communication will be carried with a tunnel protocol defined by the 3GPP standards.

3. Configure the OAI box logging capabilities and variables



/OAIBox/{boxId}/CellularLogConfig Configures OAI box logging capaiblities and variables.

Configures the OAI radio and the box itself

Parameters

Name	Description
boxId * required string($uuid) (path)	boxId

Request body required

Example Value | Schema

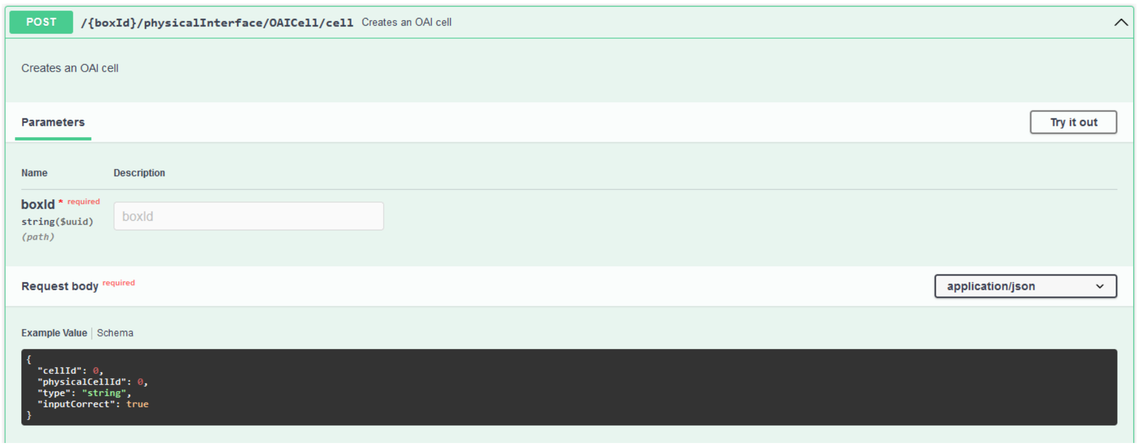
```
{
  "logOptions": {
    "fileName": "string",
    "options": "string",
    "comIpAddress": "string",
    "comPort": 0,
    "comInterface": "string"
  }
}
```

Figure 24: Log Config Box RACOON call

And finally, the last step of the box configuration is to set the logOptions, we need to set the boxId and also a json file with FlexRAN IP address, port and the interface in order to monitor and configure the different eNBs and UEs.

After a box is created in RACOON, it is possible to deploy an OpenAirInterface cell. To do so, a two step procedure needs to be taken into place, first to register the cell and then to configure.

4. Create an OAI cell


POST /{boxId}/physicalInterface/OAICell/cell Creates an OAI cell

Creates an OAI cell

Parameters

Name | Description
boxId * required
string($uuid)
(path)

Request body required

application/json

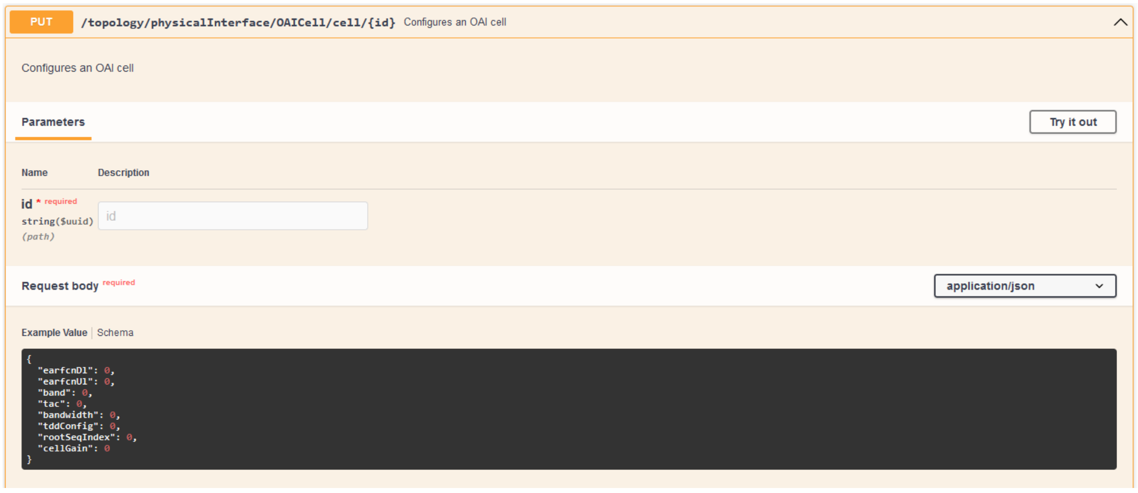
Example Value | Schema

```
{
  "cellId": 0,
  "physicalCellId": 0,
  "type": "string",
  "inputCorrect": true
}
```

Figure 25: OAI cell creation RACOON call

So, the first thing to do, is create and register the OAI cell into RACOON. To do so, the cell runs inside a box, it requires a boxId, and a json file with the cell ID, a physical cell ID and the type of the cell, if it is 4G-FDD, 4G-TDD, 5G-FDD or 5G-TD. This will return a cell id, that will be the cell identifier.

5. Configure the OAI cell


PUT /topology/physicalInterface/OAICell/cell/{id} Configures an OAI cell

Configures an OAI cell

Parameters

Name | Description
id * required
string($uuid)
(path)

Request body required

application/json

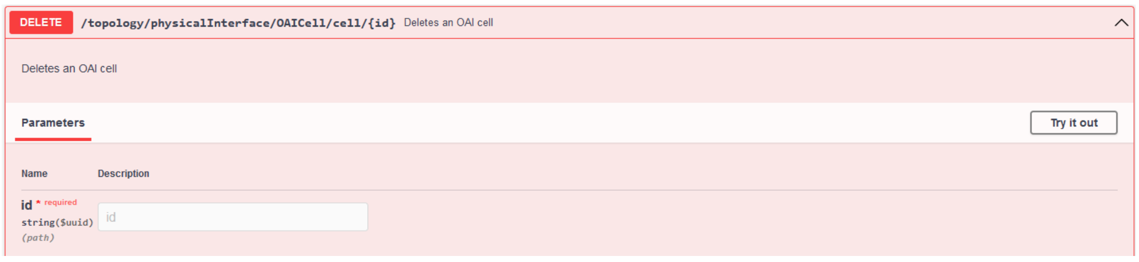
Example Value | Schema

```
{
  "earfcnDl": 0,
  "earfcnUl": 0,
  "band": 0,
  "tac": 0,
  "bandwidth": 0,
  "tddConfig": 0,
  "rootSeqIndex": 0,
  "cellGain": 0
}
```

Figure 26: OAI cell configuration call

And finally, in order to configure the OAI cell, we need to specify its cell id, as well as all the configuration parameters of a specific cell. After this is done, the cell will be up and running.

Apart from these methods of the creation of the box and cell, there is another call in the OAI technology, which it is one to delete the created cell


DELETE /topology/physicalInterface/OAICell/cell/{id} Deletes an OAI cell

Deletes an OAI cell

Parameters

Try it out

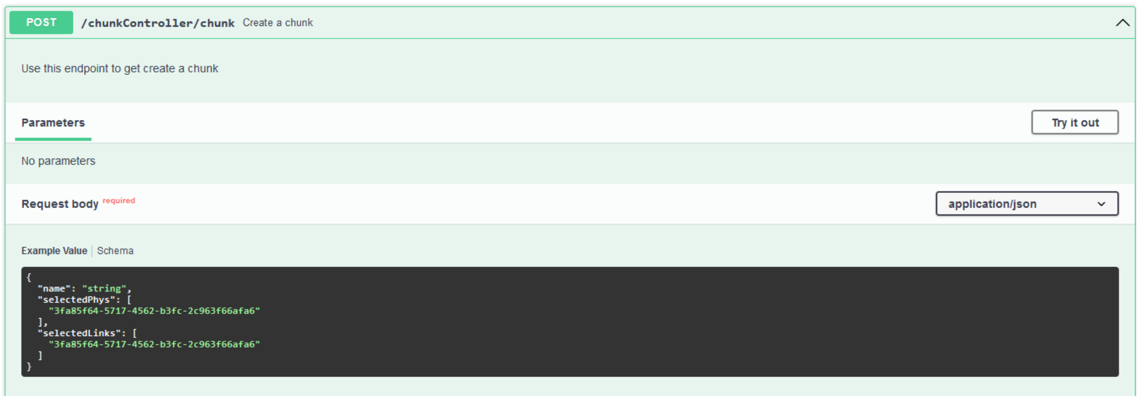
Name	Description
id * required string($uuid) (path)	id

Figure 27: OAI cell deleted call

If we want to delete a cell, it only requires to set the specify the cellId as is its unique identifier.

Moreover, there is the possibility on having a service on a concrete deployed cell to configure a certain PLMN, UL and DL quotes as well as a core network.

This service will be deployed in a chunk, which means, that before registering the service, we need to register the chunk.


POST /chunkController/chunk Create a chunk

Use this endpoint to get create a chunk

Parameters

Try it out

No parameters

Request body required

application/json

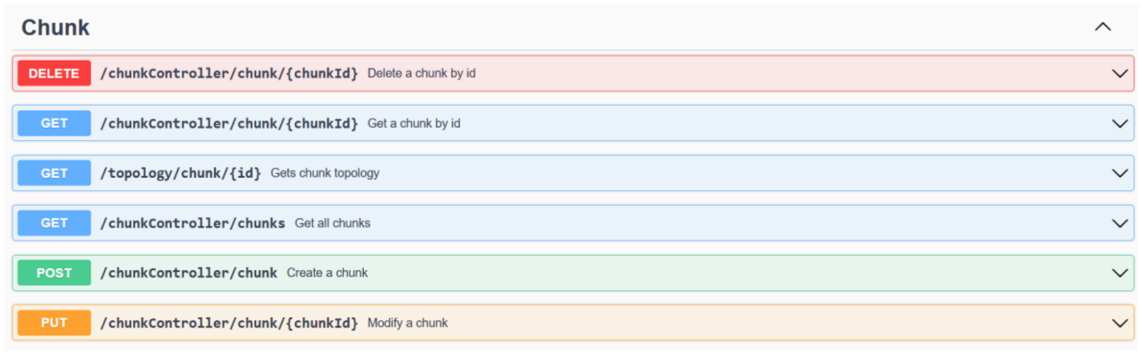
Example Value | Schema

```
{
  "name": "string",
  "selectedPhys": [
    "3fa85f64-5717-4562-b3fc-2c963f66afa6"
  ],
  "selectedLinks": [
    "3fa85f64-5717-4562-b3fc-2c963f66afa6"
  ]
}
```

Figure 28: OAI chunk creation call

In the case of the OAI technology, the chunk only carries physical interfaces (OAI cells and backhaul links). Moreover, each chunk returns an id as a unique identifier of the chunk.

Moreover, there are other calls for managing the chunks.


Chunk

DELETE /chunkController/chunk/{chunkId} Delete a chunk by id

GET /chunkController/chunk/{chunkId} Get a chunk by id

GET /topology/chunk/{id} Gets chunk topology

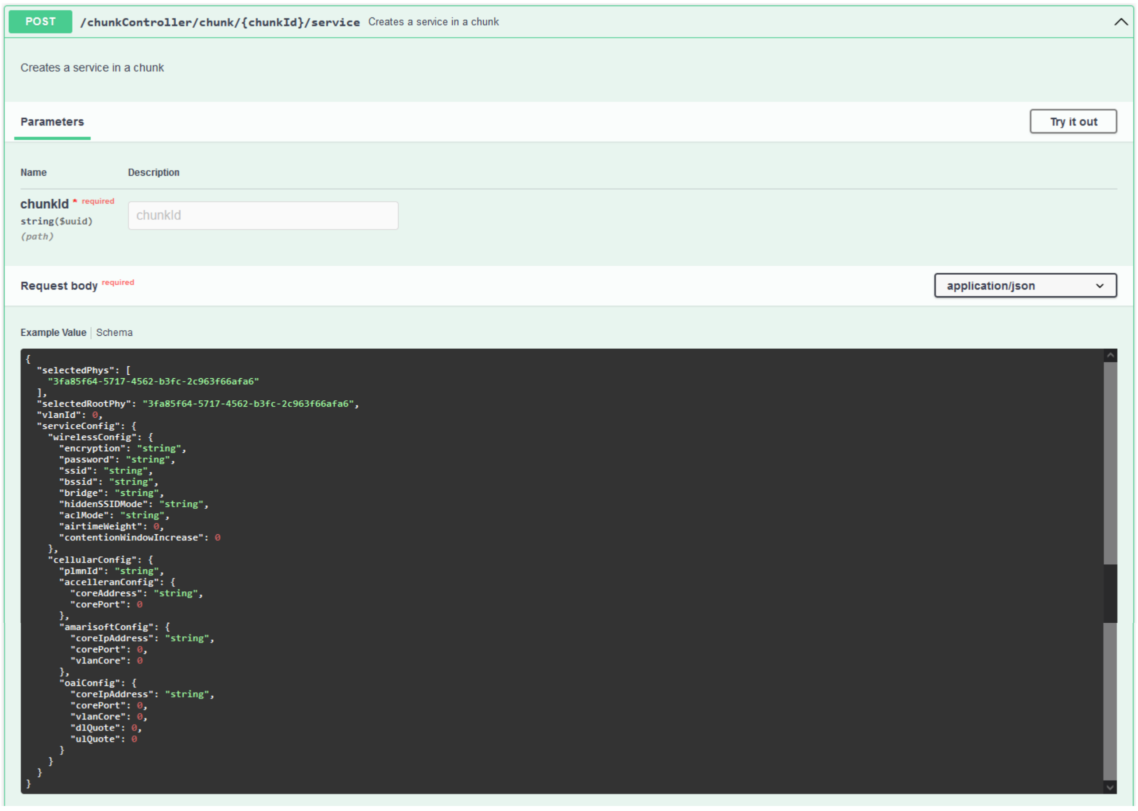
GET /chunkController/chunks Get all chunks

POST /chunkController/chunk Create a chunk

PUT /chunkController/chunk/{chunkId} Modify a chunk

Figure 29: OAI chunk management calls

After registering a chunk, services can be deployed to configure a PLMN, network slicing or to specify a core network, to do so, the service call requires first the chunk ID, and second a json file with the different physical interfaces (cells or backhaul links) and its service configuration. In the case of OAI, it is only necessary the vlanId and the cellConfiguration (plmnId and oaiConfiguration). Moreover, after deploying the service, the call will return another id corresponding to the service.



POST /chunkController/chunk/{chunkId}/service Creates a service in a chunk

Creates a service in a chunk

Parameters

Name | Description

chunkId * required
string($uuid)
(path)

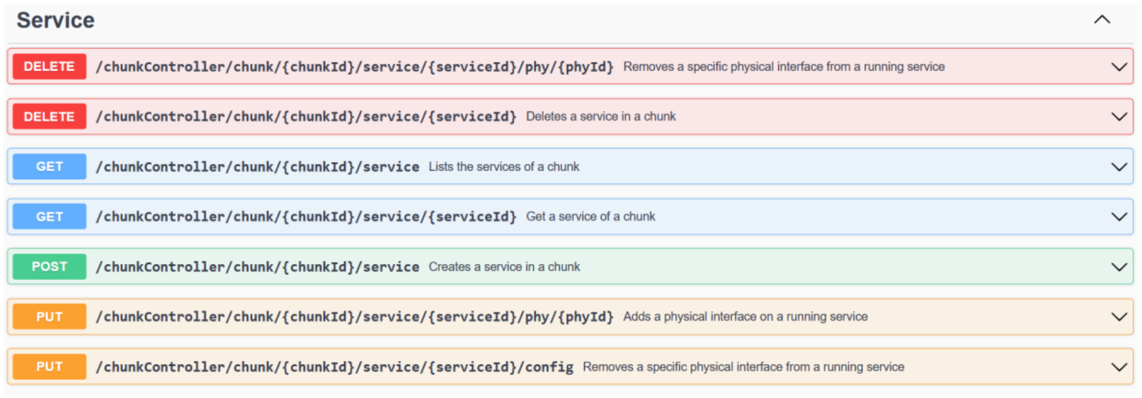
Request body required — application/json

Example Value | Schema

```
{
  "selectedPhys": [
    "3fa85f64-5717-4562-b3fc-2c963f66afa6"
  ],
  "selectedRootPhy": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "vlanId": 0,
  "serviceConfig": {
    "wirelessConfig": {
      "encryption": "string",
      "password": "string",
      "ssid": "string",
      "bssid": "string",
      "bridge": "string",
      "hiddenSSIDMode": "string",
      "aclMode": "string",
      "airtimeWeight": 0,
      "contentionWindowIncrease": 0
    },
    "cellularConfig": {
      "plmnId": "string",
      "accelleranConfig": {
        "coreAddress": "string",
        "corePort": 0
      },
      "amarisoftConfig": {
        "coreIpAddress": "string",
        "corePort": 0,
        "vlanCore": 0
      },
      "oaiConfig": {
        "coreIpAddress": "string",
        "corePort": 0,
        "vlanCore": 0,
        "dlQuote": 0,
        "ulQuote": 0
      }
    }
  }
}
```

Figure 30: OAI service creation call

As well as in the chunks, there are also other calls for manage services.



Service

DELETE /chunkController/chunk/{chunkId}/service/{serviceId}/phy/{phyId} Removes a specific physical interface from a running service

DELETE /chunkController/chunk/{chunkId}/service/{serviceId} Deletes a service in a chunk

GET /chunkController/chunk/{chunkId}/service Lists the services of a chunk

GET /chunkController/chunk/{chunkId}/service/{serviceId} Get a service of a chunk

POST /chunkController/chunk/{chunkId}/service Creates a service in a chunk

PUT /chunkController/chunk/{chunkId}/service/{serviceId}/phy/{phyId} Adds a physical interface on a running service

PUT /chunkController/chunk/{chunkId}/service/{serviceId}/config Removes a specific physical interface from a running service

Figure 31: OAI service management calls

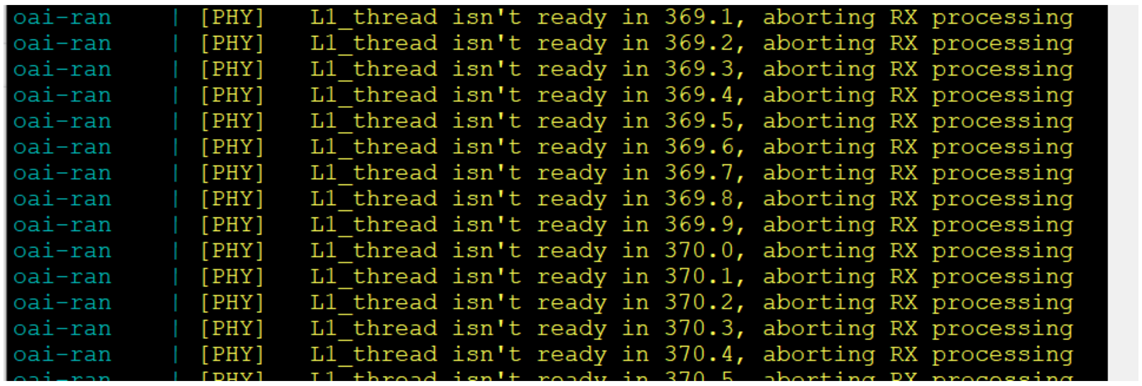
4.4. Solution study

In this section, it will be explained the different studies that have been done in order to test the different solutions. First with the study of the different tools that has been tested to deploy a consistent OAI eNB. And second, an explanation about why it is still not possible to deploy an OAI gNB cell.

4.4.1. OAI eNB: baremetal vs docker

In order to deploy an OAI eNB, two different tools have been tested to see which one is more reliable. Both of them are based in the openairinterface5g [18] gitlab repository, a OpenAirInterface repository, where there is the possibility to run docker and binary images for eNB, gNB, lte-UE and nr-UE.

Regarding the docker tool, after two months of testing, we have reach the conclusion that there is a bug, that cannot be resolved. In a cell deployment inside a docker, there is one problem in which, after some time after the deployment, it enters in an infinite loop where in the OAI RAN logs only appears the error message “L1_thread isn’t ready..”.



```
oai-ran     | [PHY]   L1_thread isn't ready in 369.1, aborting RX processing
oai-ran     | [PHY]   L1_thread isn't ready in 369.2, aborting RX processing
oai-ran     | [PHY]   L1_thread isn't ready in 369.3, aborting RX processing
oai-ran     | [PHY]   L1_thread isn't ready in 369.4, aborting RX processing
oai-ran     | [PHY]   L1_thread isn't ready in 369.5, aborting RX processing
oai-ran     | [PHY]   L1_thread isn't ready in 369.6, aborting RX processing
oai-ran     | [PHY]   L1_thread isn't ready in 369.7, aborting RX processing
oai-ran     | [PHY]   L1_thread isn't ready in 369.8, aborting RX processing
oai-ran     | [PHY]   L1_thread isn't ready in 369.9, aborting RX processing
oai-ran     | [PHY]   L1_thread isn't ready in 370.0, aborting RX processing
oai-ran     | [PHY]   L1_thread isn't ready in 370.1, aborting RX processing
oai-ran     | [PHY]   L1_thread isn't ready in 370.2, aborting RX processing
oai-ran     | [PHY]   L1_thread isn't ready in 370.3, aborting RX processing
oai-ran     | [PHY]   L1_thread isn't ready in 370.4, aborting RX processing
oai-ran     | [PHY]   L1_thread isn't ready in 370.5, aborting RX processing
```

Figure 32: RAN logs errors

After an extensive search, we found out that these errors refer to some memory leaks that we thought that happened in the docker compose container. So, as it is not reliable, we decided to only work with BareMetal, because, even though it limits the number of cells per box, it is more reliable.

4.4.2. Problems with 5G OpenAirInterface networks

The main problem with the 5G networks, come from the OAI gNB. When deploying the OAI-RAN 5G network, we found out that there was a problem with the encryption of msg3 of the CRC (Scheduled PUSCH transmission), such as it is exposed in bold in the next sequence of messages from the gNB SA configuration .

```
oai-ran     | [NR_MAC]   NR band duplex spacing is 0 KHz (nr_bandtable[37].band = 78)
oai-ran     | [NR_MAC]   NR band 78, duplex mode TDD, duplex spacing = 0 KHz
oai-ran     | [NR_MAC]   [gNB] Generate RAR MAC PDU frame 970 slot 7 preamble index 41 TA
command 25
oai-ran     | [NR_MAC]   Random Access 0 Msg3 CRC did not pass)
oai-ran     | [NR_MAC]   [gNB 0][RAPROC] Frame 971, Slot 10 : CC_id 0 Scheduling
retransmission of Msg3 in (971,17)
oai-ran     | [NR_MAC]   Random Access 0 Msg3 CRC did not pass)
oai-ran     | [NR_MAC]   [gNB 0][RAPROC] Frame 972, Slot 10 : CC_id 0 Scheduling
retransmission of Msg3 in (972,17)
oai-ran     | [NR_MAC]   Random Access 0 Msg3 CRC did not pass)
oai-ran     | [NR_MAC]   [gNB 0][RAPROC] Frame 973, Slot 10 : CC_id 0 Scheduling
retransmission of Msg3 in (973,17)
oai-ran     | [NR_MAC]   Random Access 0 failed at state 2 (Reached msg3 max harq rounds)
oai-ran     | [NR_MAC]   to remove in mac rnti_to_remove[0] = 0xdb5f
oai-ran     | [NR_MAC]   handle_nr_ul_harq(): unknown RNTI 0xdb5f in PUSCH
oai-ran     | [NR_PHY]   to remove rnti 0xdb5f
```

```
oai-ran     | [NR_PHY]   to remove rnti_to_remove_count=1, up_removed=1 down_removed=0
pucch_removed=0
oai-ran     | [NR_PHY]   [gNB 0][RAPROC] Frame 985, slot 19 Initiating RA procedure with
preamble 63, energy 37.0 dB (I0 242, thres 120), delay 0 start symbol 0 freq index 0
oai-ran     | [MAC]   UL_info[Frame 985, Slot 19] Calling initiate_ra_proc
RACH:SFN/SLOT:985/19
oai-ran     | [NR_MAC]   [gNB 0][RAPROC] CC_id 0 Frame 985 Activating Msg2 generation in
frame 986, slot 7 using RA rnti 10b SSB index 0 RA index 0
oai-ran     | [NR_MAC]   [gNB 0][RAPROC] CC_id 0 Frame 986, slotP 7: Generating RA-Msg2
DCI, rnti 0x10b, state 1, CoreSetType 2
```

The CRC stands for Cyclic Redundancy Checksum [19], and it is used for reducing the error rate in the data transmission and data storage. During the firsts months we tried to develop a solution for this, but we couldn't manage to do it, because, even though it is an open source, it is not possible to eliminate this part of the original code. At the end, it was decide to do the system for LTE and prepare it for the future implementation of 5G.

4.5. RAN Slicing

An important feature of this project is the fact that it can dynamically assign RAN slices [21].

Slicing allows to assign a slice or percentage of the radio resources blocks (RBs) to the different operators, both for uplink and downlink transmissions. A Resource Block forms the block structure in LTE in the time-frequency domain, as a summary:

- 1 frame is 10ms and consists of 10 sub-frames.
- 1 sub-frame is 1ms and contains 2 slots.
- 1 slot is 0.5ms in the time domain and each 0.5ms allocation can contain N Resource Blocks (where 6 < N < 110) depending on the bandwidth allocation and resource availability.
- **1 Resource Block** is 0.5ms and contains 12 subcarriers per OFDM symbol in the frequency domain.

Moreover, the different UEs connected to the LTE RAN, will be allocated in their operator RAN slice.

In this project, the different RAN slices are created throw the real-time controller FlexRAN [20] as said in the 4.2 section, with the command:

```
$ curl -X POST http://FlexRAN_PUBLIC_IPADDR:9999/slice/enb/-
1 --data-binary @ran-sharing.json
```

As said, to assign the RAN slice, it must be introduced into the RACOON service next to the other parameters (PLMN id and core network).

4.5.1. RAN Slicing dynamically assignation

In the next figure is presented a dynamic RAN slice assignation of two different operators, one with PLMN 00102 and the other one 00103. The current cell of the example is working in the E-UTRA band 7 transmitting with downlink frequency of 2680MHz and uplink frequency of 2560MHz.

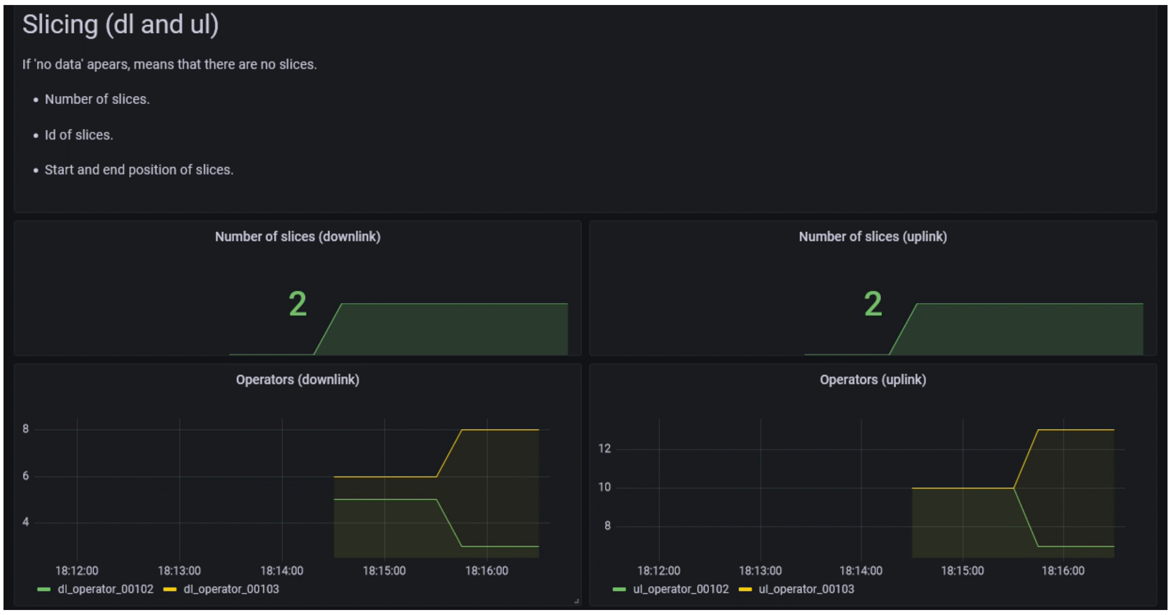
This test consists of the RBs assignation of two different operators over time (time1 and time2). In the next table is presented the dynamically slice assignation for both 00102 and

00103 operators, the downlink and uplink RBs over time, it is also presented the position of this RAN slicing.

Transmission	Operator	Time 1 RBs [slice position]	Time 2 RBs [slice position]
Downlink	00102	5 RBs [0-5]	3 RBs [0-3]
	00103	6 RBs [6-12]	8 RBs [4-12]
Uplink	00102	10 RBs [2-12]	7 RBs [2-9]
	00103	10 RBs [13-23]	13 RBs [10-23]

Table 5: Dynamically slice assignation

In the OAI Exporter is expressed the number of RBs in real-time, for each operator and each transmission.


Slicing (dl and ul)

If 'no data' apears, means that there are no slices.

- Number of slices.
- Id of slices.
- Start and end position of slices.

Figure 33: RAN slice assignation, RBs over time

4.6. <u>Demonstration of a non-rt RAN controller for an automation service deployment over a 4G small cell</u>

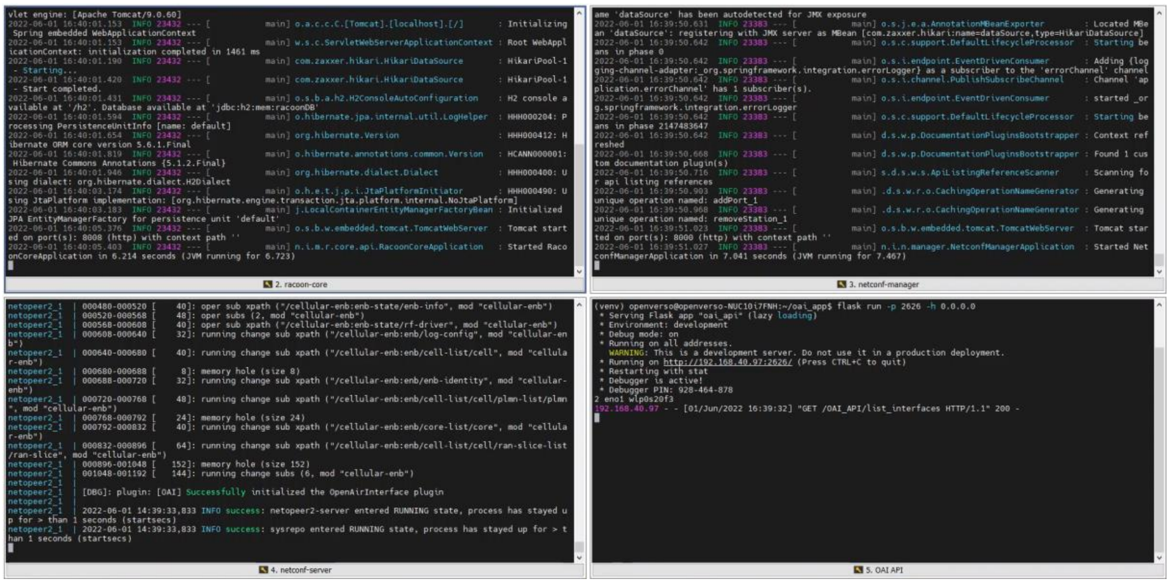
In this last part, it will be presented a demonstration on a 4G service deployment based on the developed RAN controller RACOON and how it communicates with the other software elements as well as with FlexRAN. In this demonstration, it will be created a LTE cell in a box with two different core networks and two UEs, one connected to each core network. At the end it is also exposed the different Grafana dashboards with the cell and UEs information.

These are the steps that have been followed to deploy this service:

- Creation and configuration of a OAI BOX.
- Creation and configuration of a OAI eNB cell with a USRP b210.
- Configuration of two different core networks with its subscriber information.

- Configuration of two different services to set the OAI core networks in the eNB with its correspondents PLMN ids and an assigned uplink and downlink quotes.
- Connection of two different UEs to the eNB, the first one, a Quectel device, and the second one a SimCom.
- Grafana and Prometheus exporter to monitor the different eNB and UEs information.

Before starting with this demo, we need to run all of the five different software tools developed: racoon-core, netconf-manager, netconf-server, oai-API and oai-exporter.


```
vlet engine: [Apache Tomcat/9.0.60]
2022-06-01 16:40:01.153  INFO 23432 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing
 Spring embedded WebApplicationContext
2022-06-01 16:40:01.153  INFO 23432 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebAppl
icationContext: initialization completed in 1461 ms
2022-06-01 16:40:01.190  INFO 23432 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1
 - Starting...
2022-06-01 16:40:01.420  INFO 23432 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1
 - Start completed.
2022-06-01 16:40:01.431  INFO 23432 --- [           main] o.s.b.a.h2.H2ConsoleAutoConfiguration    : H2 console a
vailable at '/h2'. Database available at 'jdbc:h2:mem:racoonDB'
2022-06-01 16:40:01.594  INFO 23432 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: P
rocessing PersistenceUnitInfo [name: default]
2022-06-01 16:40:01.654  INFO 23432 --- [           main] org.hibernate.Version                    : HHH000412: H
ibernate ORM core version 5.6.1.Final
2022-06-01 16:40:01.819  INFO 23432 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001:
 Hibernate Commons Annotations {5.1.2.Final}
2022-06-01 16:40:01.946  INFO 23432 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: U
sing dialect: org.hibernate.dialect.H2Dialect
2022-06-01 16:40:03.174  INFO 23432 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: U
sing JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2022-06-01 16:40:03.183  INFO 23432 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized
JPA EntityManagerFactory for persistence unit 'default'
2022-06-01 16:40:05.376  INFO 23432 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat start
ed on port(s): 8008 (http) with context path ''
2022-06-01 16:40:05.403  INFO 23432 --- [           main] n.i.m.r.core.api.RacoonCoreApplication   : Started Raco
onCoreApplication in 6.214 seconds (JVM running for 6.723)
```

```
ame 'dataSource' has been autodetected for JMX exposure
2022-06-01 16:39:50.631  INFO 23383 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Located MBe
an 'dataSource': registering with JMX server as MBean [com.zaxxer.hikari:name=dataSource,type=HikariDataSource]
2022-06-01 16:39:50.642  INFO 23383 --- [           main] o.s.c.support.DefaultLifecycleProcessor  : Starting be
ans in phase 0
2022-06-01 16:39:50.642  INFO 23383 --- [           main] o.s.i.endpoint.EventDrivenConsumer       : Adding {log
ging-channel-adapter:_org.springframework.integration.errorLogger} as a subscriber to the 'errorChannel' channel
2022-06-01 16:39:50.642  INFO 23383 --- [           main] o.s.i.channel.PublishSubscribeChannel    : Channel 'ap
plication.errorChannel' has 1 subscriber(s).
2022-06-01 16:39:50.642  INFO 23383 --- [           main] o.s.i.endpoint.EventDrivenConsumer       : started _or
g.springframework.integration.errorLogger
2022-06-01 16:39:50.642  INFO 23383 --- [           main] o.s.c.support.DefaultLifecycleProcessor  : Starting be
ans in phase 2147483647
2022-06-01 16:39:50.642  INFO 23383 --- [           main] d.s.w.p.DocumentationPluginsBootstrapper : Context ref
reshed
2022-06-01 16:39:50.668  INFO 23383 --- [           main] d.s.w.p.DocumentationPluginsBootstrapper : Found 1 cus
tom documentation plugin(s)
2022-06-01 16:39:50.716  INFO 23383 --- [           main] s.d.s.w.s.ApiListingReferenceScanner     : Scanning fo
r api listing references
2022-06-01 16:39:50.903  INFO 23383 --- [           main] .d.s.w.r.o.CachingOperationNameGenerator : Generating
unique operation named: addPort_1
2022-06-01 16:39:50.968  INFO 23383 --- [           main] .d.s.w.r.o.CachingOperationNameGenerator : Generating
unique operation named: removeStation_1
2022-06-01 16:39:51.023  INFO 23383 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat star
ted on port(s): 8000 (http) with context path ''
2022-06-01 16:39:51.027  INFO 23383 --- [           main] n.i.n.manager.NetconfManagerApplication  : Started Net
confManagerApplication in 7.041 seconds (JVM running for 7.467)
```

```
netopeer2_1  | 000480-000520 [    40]: oper sub xpath ("/cellular-enb:enb-state/enb-info", mod "cellular-enb")
netopeer2_1  | 000520-000568 [    48]: oper subs (2, mod "cellular-enb")
netopeer2_1  | 000568-000608 [    40]: oper sub xpath ("/cellular-enb:enb-state/rf-driver", mod "cellular-enb")
netopeer2_1  | 000608-000640 [    32]: running change sub xpath ("/cellular-enb:enb/log-config", mod "cellular-en
b")
netopeer2_1  | 000640-000680 [    40]: running change sub xpath ("/cellular-enb:enb/cell-list/cell", mod "cellula
r-enb")
netopeer2_1  | 000680-000688 [     8]: memory hole (size 8)
netopeer2_1  | 000688-000720 [    32]: running change sub xpath ("/cellular-enb:enb/enb-identity", mod "cellular-
enb")
netopeer2_1  | 000720-000768 [    48]: running change sub xpath ("/cellular-enb:enb/cell-list/cell/plmn-list/plmn
", mod "cellular-enb")
netopeer2_1  | 000768-000792 [    24]: memory hole (size 24)
netopeer2_1  | 000792-000832 [    40]: running change sub xpath ("/cellular-enb:enb/core-list/core", mod "cellula
r-enb")
netopeer2_1  | 000832-000896 [    64]: running change sub xpath ("/cellular-enb:enb/cell-list/cell/ran-slice-list
/ran-slice", mod "cellular-enb")
netopeer2_1  | 000896-001048 [   152]: memory hole (size 152)
netopeer2_1  | 001048-001192 [   144]: running change subs (6, mod "cellular-enb")
netopeer2_1  |
netopeer2_1  | [DBG]: plugin: [OAI] Successfully initialized the OpenAirInterface plugin
netopeer2_1  |
netopeer2_1  | 2022-06-01 14:39:33,833 INFO success: netopeer2-server entered RUNNING state, process has stayed u
p for > than 1 seconds (startsecs)
netopeer2_1  | 2022-06-01 14:39:33,833 INFO success: sysrepo entered RUNNING state, process has stayed up for > t
han 1 seconds (startsecs)
```

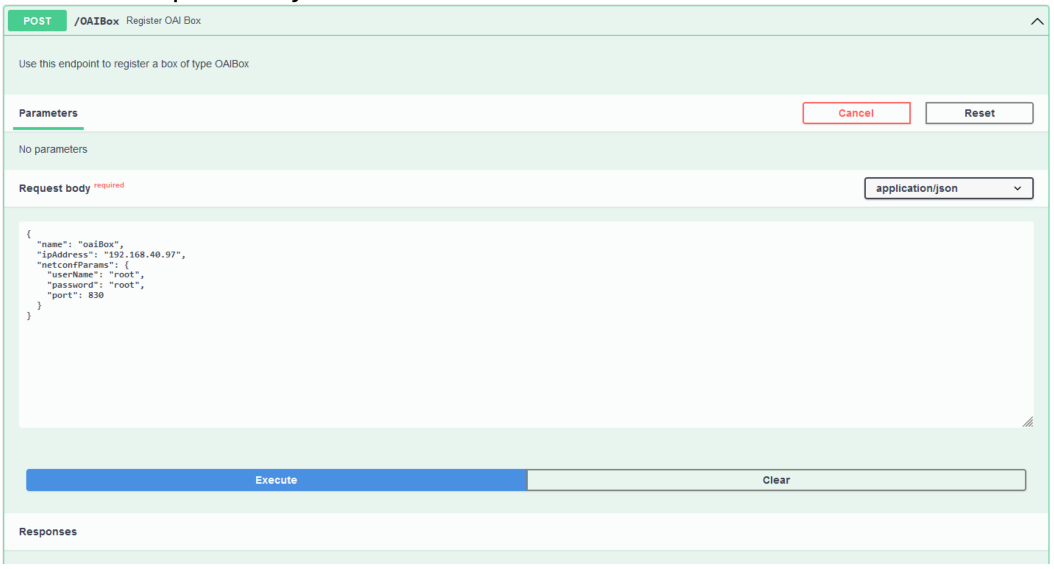
```
(venv) openverso@openverso-NUC10i7FNH:~/oai_app$ flask run -p 2626 -h 0.0.0.0
 * Serving Flask app "oai_api" (lazy loading)
 * Environment: development
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://192.168.40.97:2626/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 928-464-878
2 eno1 wlp0s20f3
192.168.40.97 - - [01/Jun/2022 16:39:32] "GET /OAI_API/list_interfaces HTTP/1.1" 200 -
```

Figure 34: Screenshot of the racoon-core, netconf-manage, netconf-server and oai-api initialized.

To start the deployment of this system, all the registrations and configurations are done throw the racoon-core API, the access to it, is done throw its SWAGGER URL: http://192.168.40.132:8008/swagger-ui/index.html#/

<u>Registration and configuration of a OAI BOX</u>

1. To first register the box, we will execute the call /OAIBox with the parameters inside the request body.

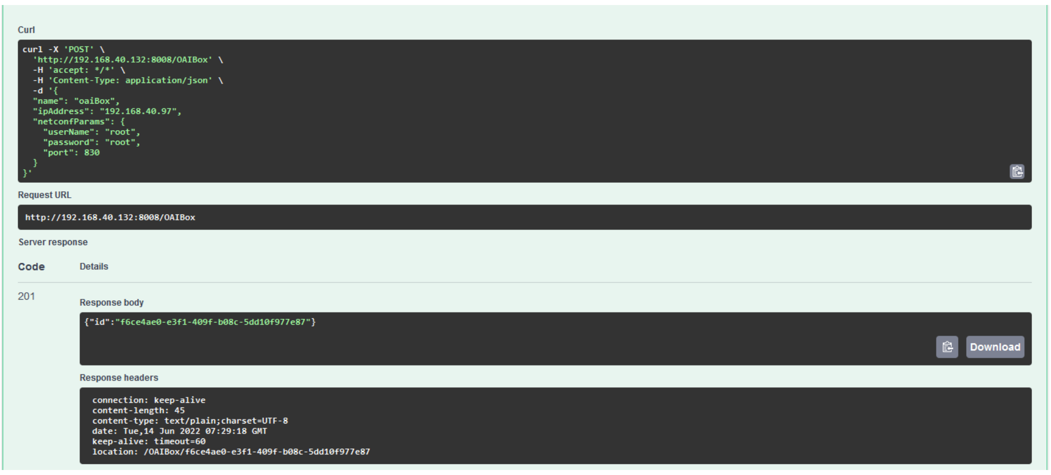


Use this endpoint to register a box of type OAIBox

No parameters

Request body required

```
{
  "name": "oaiBox",
  "ipAddress": "192.168.40.97",
  "netconfParams": {
    "userName": "root",
    "password": "root",
    "port": 830
  }
}
```


Curl

```
curl -X 'POST' \
  'http://192.168.40.132:8008/OAIBox' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "oaiBox",
  "ipAddress": "192.168.40.97",
  "netconfParams": {
    "userName": "root",
    "password": "root",
    "port": 830
  }
}'
```

Request URL

```
http://192.168.40.132:8008/OAIBox
```

Server response

Code Details

201

Response body

```
{"id":"f6ce4ae0-e3f1-409f-b08c-5dd10f977e87"}
```

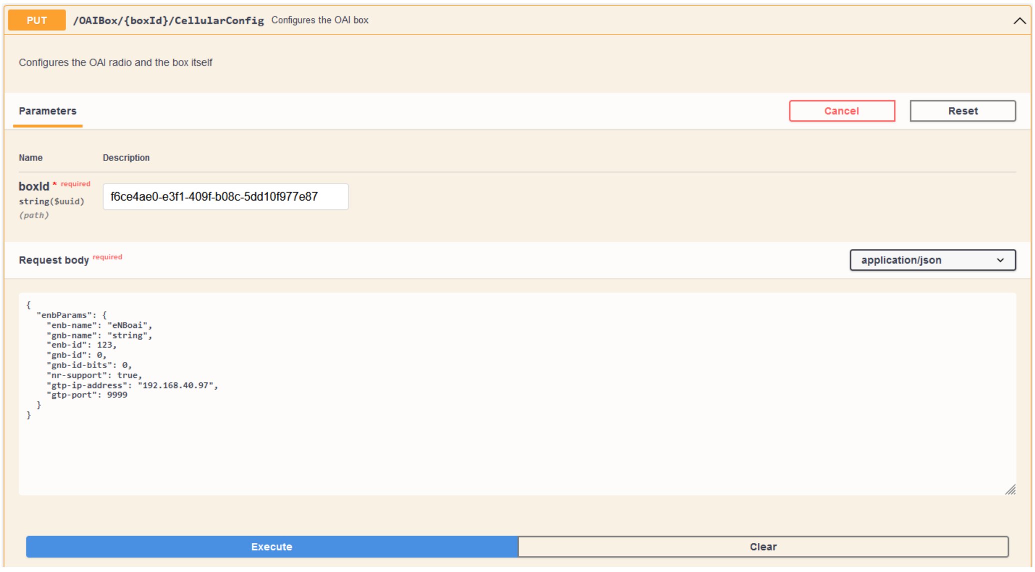
Response headers

```
connection: keep-alive
content-length: 45
content-type: text/plain;charset=UTF-8
date: Tue,14 Jun 2022 07:29:18 GMT
keep-alive: timeout=60
location: /OAIBox/f6ce4ae0-e3f1-409f-b08c-5dd10f977e87
```

Figure 35: Box Registration demo

As can be seen, the ipAddress corresponds to the eNB NUCs IP.
The response returns a 201 code and has a body which includes the *box id*.

2. Second, we need to configure the box parameters.



Configures the OAI radio and the box itself

Name Description

boxId * required string($uuid) (path): f6ce4ae0-e3f1-409f-b08c-5dd10f977e87

Request body required: application/json

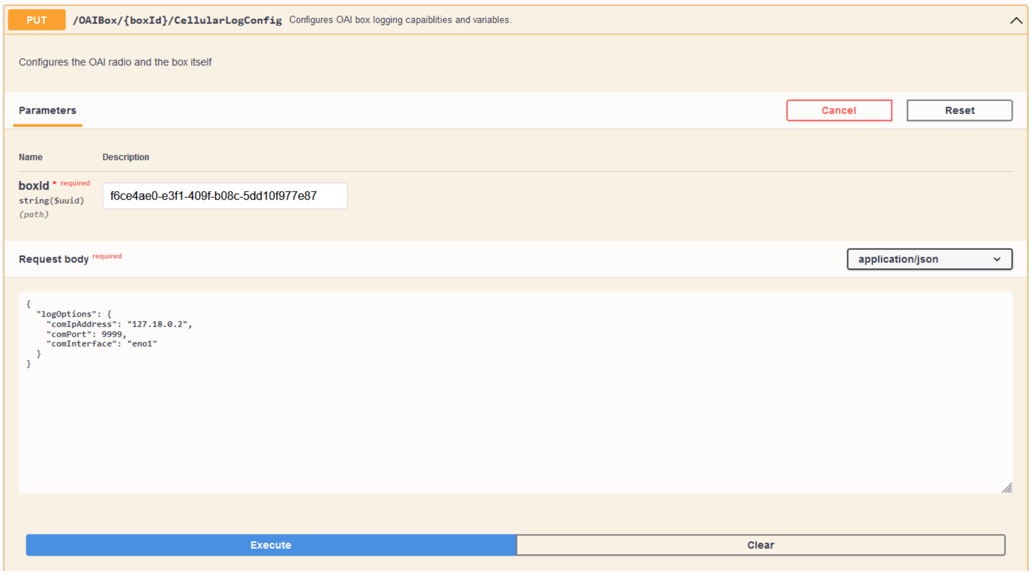
```
{
  "enbParams": {
    "enb-name": "eNBoai",
    "gnb-name": "string",
    "enb-id": 123,
    "gnb-id": 0,
    "gnb-id-bits": 0,
    "nr-support": true,
    "gtp-ip-address": "192.168.40.97",
    "gtp-port": 9999
  }
}
```

Figure 36: Box Config demo

As can be seen, we only set the enb parameters, leaving the gnb to null or 0.

If the configuration is correct, it returns a 204.

3. And third, the configuration of the box logging capabilities and parameters.



PUT /OAIBox/{boxId}/CellularLogConfig Configures OAI box logging capaiblities and variables.

Configures the OAI radio and the box itself

Parameters

Cancel Reset

Name Description

boxId * required
string($uuid)
(path)
f6ce4ae0-e3f1-409f-b08c-5dd10f977e87

Request body required application/json

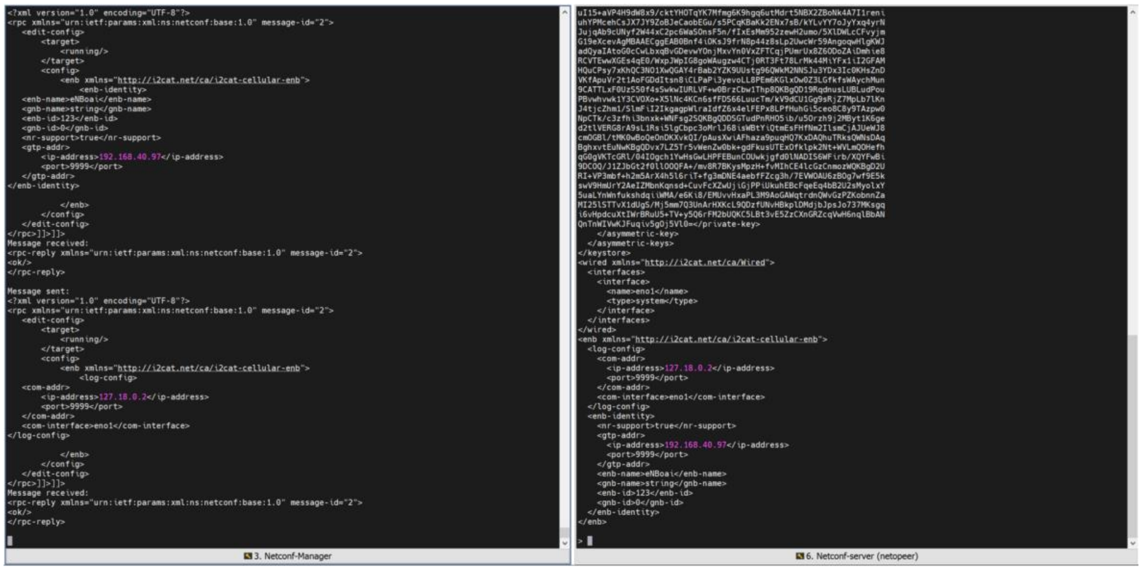
```
{
  "logOptions": {
    "comIpAddress": "127.18.0.2",
    "comPort": 9999,
    "comInterface": "eno1"
  }
}
```

Figure 37: Box Config FlexRAN demo

In the logging capabilities side, we must set the FlexRAN information, as from it, all future configurations will be done. In our case, FlexRAN is located at IP 127.18.0.2, port 9999 and interface eno1.

If the configuration is correct, it returns a 204 code.

After the registration and configuration on the box, we can check out the Netconf-Manager and Netconf-Server (netopeer2) logs to see if the configuration is correct.



```
<?xml version="1.0" encoding="UTF-8"?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
   <edit-config>
       <target>
           <running/>
       </target>
       <config>
           <enb xmlns="http://i2cat.net/ca/i2cat-cellular-enb">
               <enb-identity>
   <enb-name>eNBoai</enb-name>
   <gnb-name>string</gnb-name>
   <enb-id>123</enb-id>
   <gnb-id>0</gnb-id>
   <nr-support>true</nr-support>
   <gtp-addr>
       <ip-address>192.168.40.97</ip-address>
       <port>9999</port>
   </gtp-addr>
</enb-identity>

           </enb>
       </config>
   </edit-config>
</rpc>]]>]]>
Message received:
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
<ok/>
</rpc-reply>

Message sent:
<?xml version="1.0" encoding="UTF-8"?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
   <edit-config>
       <target>
           <running/>
       </target>
       <config>
           <enb xmlns="http://i2cat.net/ca/i2cat-cellular-enb">
               <log-config>
   <com-addr>
       <ip-address>127.18.0.2</ip-address>
       <port>9999</port>
   </com-addr>
   <com-interface>eno1</com-interface>
</log-config>

           </enb>
       </config>
   </edit-config>
</rpc>]]>]]>
Message received:
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
<ok/>
</rpc-reply>

█
```

3. Netconf-Manager

```
uI15+aVP4H9dW8x9/cktYHOTqYK7Mfmg6K9hgq6utMdrt5NBX2ZBoNk4A7I1reni
uhYPMcehCsJX7JY9ZoBJeCaobEGu/s5PCqKBaKk2ENx7sB/kYLvYY7oJyYxq4yrN
JujqAb9cUNyf2W44xC2pc6WaSOnsF5n/fIxEsMm952zewH2umo/5XlDWLcCFvyjm
G19eXcevAgMBAAECggEAB0Bnf4iOKsJ9frN8p44z8sLp2UwcWr59AngoqwHlgKWJ
adQyaIAtoG0cCwLbxqBvGDevwYOnjMxvYn0VxZFTCqjPUmrUx8Z6ODoZAiDmhie8
RCVTEwwXGEs4qE0/WxpJWpIG8goWAugzw4CTj0RT3Ft78LrMk44MiYFx1iI2GFAM
HQuCPsy7xKhQC3NO1XwQGAY4rBab2YZK9UUstg96QWkM2NNSJu3YDx3Ic0KHsZnD
VKfApuVr2t1AoFGDdItsn8iCLPaPi3yevoLL8PEm6KGlxOw0Z3LGfkfsWAychMun
9CATTLxF0Uz550f4sSwkwIURLVF+w0BrzCbw1Thp8QKBgQD19RqdnusLUBLudPou
PBvwhvwk1Y3CVOXo+X5lNc4KCn6sfFDS66LuucTm/kV9dCU1Gg9sRjZ7MpLb7lKn
J4tjcZhm1/SlmFiI2IkgagpWlraIdfZ6x4elFEPx8LPfHuhGi5ceo8C8y9TAzpw0
NpCTk/c3zfhi3bnxk+WNFsg25QKBgQDDSGTudPnRHO5ib/u5Orzh9j2MByt1K6ge
d2tlVERG8rA9sL1Rsi5lgCbpc3oMrlJ68isWBtYiQtmEsFHfNm2IlsmCjAJUeWJ8
cmOGBl/tMK0wBoQeOnDKXvkQI/pAusXwiAFhaza9puqHQ7KxDAQhuTRksQWNsDAq
BghxvtEuNwKBgQDvx7LZ5Tr5vWenZw0bk+gdFkusUTEx0fklpk2Nt+WVLmQOHefh
qG0gVKTcGRl/04IOgch1YwHsGwLHPFEBunCOUwkjgfd0lNADIS6WFirb/XQYFwBi
9DCOQ/J1ZJbGt2f0llO0QFA+/mv8R7BKysMpzH+fvMIhCE4lcGzCnmozWQKBgD2U
RI+VP3mbf+h2m5ArX4h5l6riT+fg3mDNE4aebfFZcg3h/7EVWOAU6zBOg7wf9E5k
swV9HmUrY2AeIZMbnKqnsd+CuvFcXZwUjiGjPPiUkuhEBcFqeEq4bB2U2sMyolxY
5uaLYnWnfukshdqiiWMA/e6Ki8/EMUvvHxaPL3M9AoGAWqtrdnQWvGzPZKobnnZa
MI25lSTTvX1dUg5/Mj5mm7Q3UnArHXKcL9QDzfUNvHBkplDMdjbJpsJo737MKsgq
i6vHpdcuXtIWrBRuU5+TV+y5Q6rFM2bUQKC5LBt3vE5ZzCXnGRZcqVwH6nqlBbAN
QnTnWIVwKJFuqiv5gOj5Vl0=</private-key>
    </asymmetric-key>
  </asymmetric-keys>
</keystore>
<wired xmlns="http://i2cat.net/ca/Wired">
  <interfaces>
    <interface>
      <name>eno1</name>
      <type>system</type>
    </interface>
  </interfaces>
</wired>
<enb xmlns="http://i2cat.net/ca/i2cat-cellular-enb">
  <log-config>
    <com-addr>
      <ip-address>127.18.0.2</ip-address>
      <port>9999</port>
    </com-addr>
    <com-interface>eno1</com-interface>
  </log-config>
  <enb-identity>
    <nr-support>true</nr-support>
    <gtp-addr>
      <ip-address>192.168.40.97</ip-address>
      <port>9999</port>
    </gtp-addr>
    <enb-name>eNBoai</enb-name>
    <gnb-name>string</gnb-name>
    <enb-id>123</enb-id>
    <gnb-id>0</gnb-id>
  </enb-identity>
</enb>

> █
```

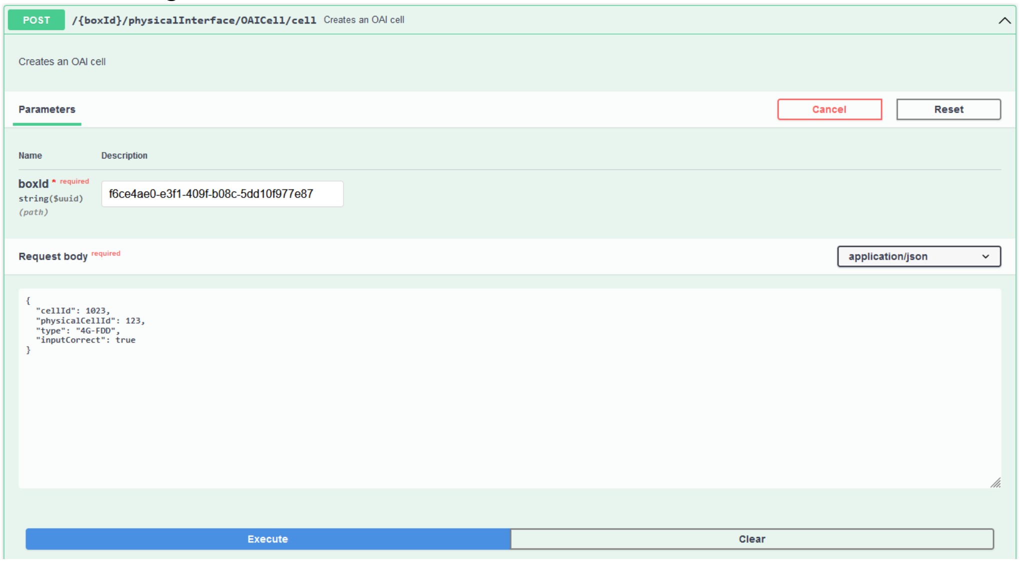
6. Netconf-server (netopeer)

Figure 38: Netconf-Manager and Netconf-server logs

As can be seen in the Netconf-Manager logs (left), it sends two messages, the first one with box configuration parameters, and the second one with the log capabilities parameters, both in a .xml format. Moreover, in the Netconf-server (netopeer-2), when we introduce the command to get the configuration, we find a .xml format file, with the name of the box, and both configurations.

Creation and configuration of a OAI cell

1. The first thing to do is the creation of the cell.



POST /{boxId}/physicalInterface/OAICell/cell Creates an OAI cell

Creates an OAI cell

Parameters

Name Description

boxId * required string($uuid) (path): f6ce4ae0-e3f1-409f-b08c-5dd10f977e87

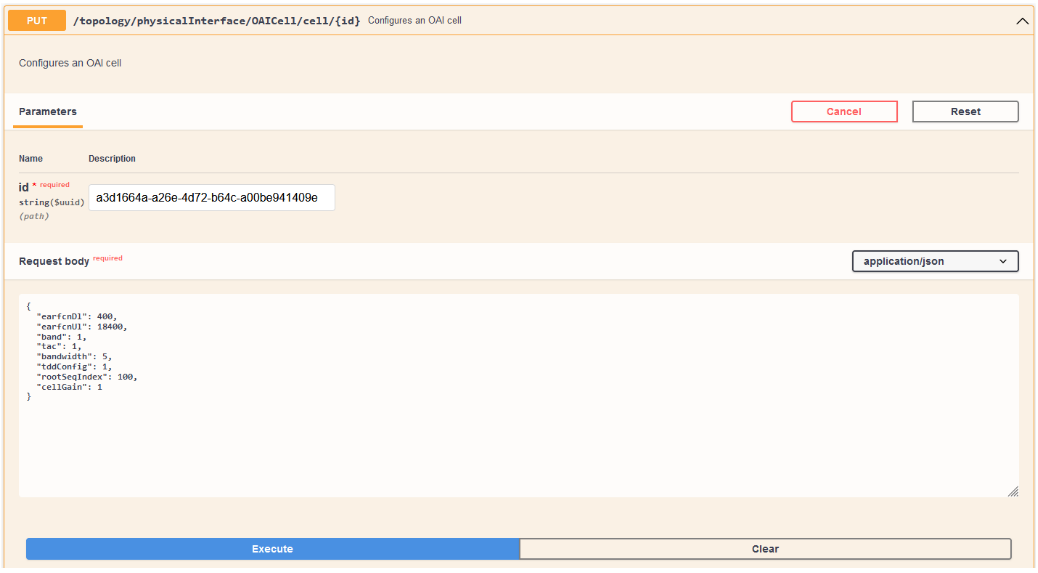
Request body required: application/json

```
{
  "cellId": 1023,
  "physicalCellId": 123,
  "type": "4G-FDD",
  "inputCorrect": true
}
```

Figure 39: Cell Creation demo

In order to create a cell inside the box, the first parameter that needs to be introduced is the boxId, and later, the main elements of the cell (cellId, physicalCellId, type).
With the creation of the cell, it is only registered to racoon-core. To deploy it, we need to configure it with all the parameters.
This first call, returns an id, which is unique for this cell.

2. The second thing to do is to configure and deploy the cell.



PUT /topology/physicalInterface/OAICell/cell/{id} Configures an OAI cell

Configures an OAI cell

Parameters

Name Description

id * required string($uuid) (path): a3d1664a-a26e-4d72-b64c-a00be941409e

Request body required: application/json

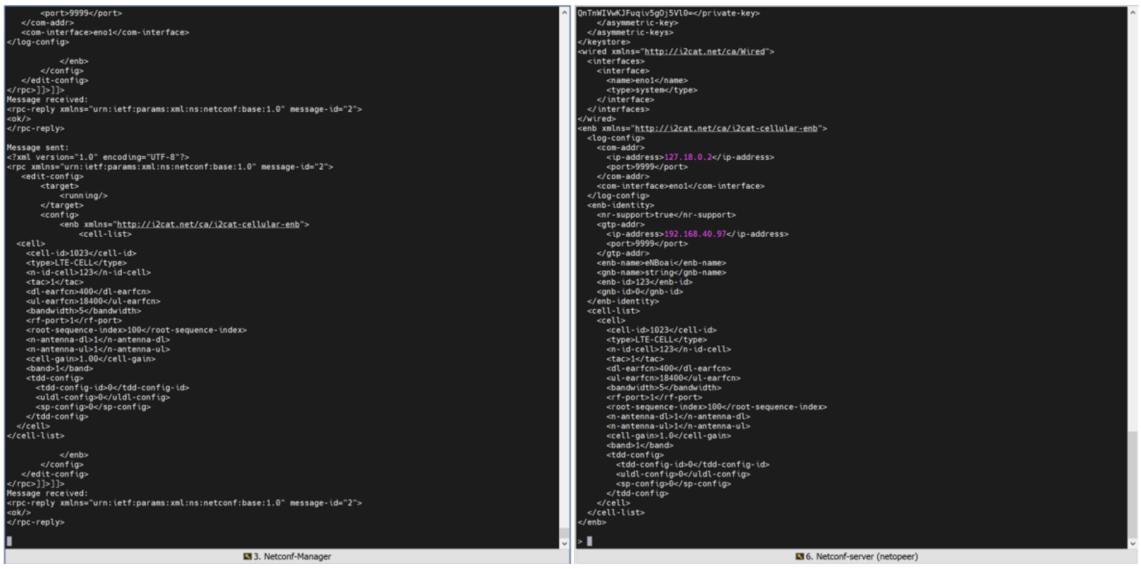
```
{
  "earfcnDl": 400,
  "earfcnUl": 18400,
  "band": 1,
  "tac": 1,
  "bandwidth": 5,
  "tddConfig": 1,
  "rootSeqIndex": 100,
  "cellGain": 1
}
```

Figure 40: Cell Configuration demo

First, it requires the cell id, and then all the main parameters of the cell, that, as can be seen, we have specified the eutra band 1, with a earfcnDl frequency of 400 and earfcnUl frequency of 18400 and a bandwidth of 5MHz. We also set the tac, the tddConfig, the rootSeqInndex and the cellGain. As said in the section 4.2 there are other parameters to be said, but, those are fixed. One important fixed parameter is the serial number of the USRP, that is hardcoded to 31DB5A3 (a USRP b210).

This returns a 204 code if it is correct.

As before, if we now check the configuration in the Netconf-Manager and Netconf-Server (netopeer2) we will now see, that in the Netconf-Manager (left) it appears an xml message with all the cell configuration, and in the Netconf-Server (netopeer) (right) when we get the configuration, it returns a xml with the box and the cell configuration.

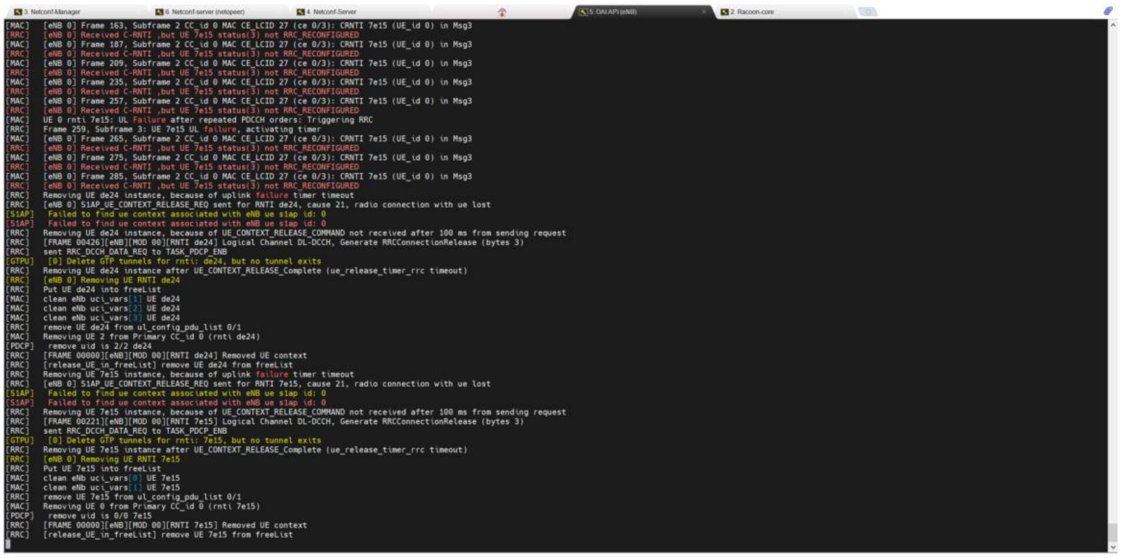


```
        <port>9999</port>
    </com-addr>
    <com-interface>eno1</com-interface>
</log-config>

            </enb>
        </config>
    </edit-config>
</rpc>]]>]]>
Message received:
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
<ok/>
</rpc-reply>

Message sent:
<?xml version="1.0" encoding="UTF-8"?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
    <edit-config>
        <target>
            <running/>
        </target>
        <config>
            <enb xmlns="http://i2cat.net/ca/i2cat-cellular-enb">
                <cell-list>
  <cell>
    <cell-id>1023</cell-id>
    <type>LTE-CELL</type>
    <n-id-cell>123</n-id-cell>
    <tac>1</tac>
    <dl-earfcn>400</dl-earfcn>
    <ul-earfcn>18400</ul-earfcn>
    <bandwidth>5</bandwidth>
    <rf-port>1</rf-port>
    <root-sequence-index>100</root-sequence-index>
    <n-antenna-dl>1</n-antenna-dl>
    <n-antenna-ul>1</n-antenna-ul>
    <cell-gain>1.00</cell-gain>
    <band>1</band>
    <tdd-config>
      <tdd-config-id>0</tdd-config-id>
      <uldl-config>0</uldl-config>
      <sp-config>0</sp-config>
    </tdd-config>
  </cell>
</cell-list>

            </enb>
        </config>
    </edit-config>
</rpc>]]>]]>
Message received:
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
<ok/>
</rpc-reply>
```

```
QnTnWIVwKJFuqiv5gOj5Vl0=</private-key>
    </asymmetric-key>
  </asymmetric-keys>
</keystore>
<wired xmlns="http://i2cat.net/ca/Wired">
  <interfaces>
    <interface>
      <name>eno1</name>
      <type>system</type>
    </interface>
  </interfaces>
</wired>
<enb xmlns="http://i2cat.net/ca/i2cat-cellular-enb">
  <log-config>
    <com-addr>
      <ip-address>127.18.0.2</ip-address>
      <port>9999</port>
    </com-addr>
    <com-interface>eno1</com-interface>
  </log-config>
  <enb-identity>
    <nr-support>true</nr-support>
    <gtp-addr>
      <ip-address>192.168.40.97</ip-address>
      <port>9999</port>
    </gtp-addr>
    <enb-name>eNBoai</enb-name>
    <gnb-name>string</gnb-name>
    <enb-id>123</enb-id>
    <gnb-id>0</gnb-id>
  </enb-identity>
  <cell-list>
    <cell>
      <cell-id>1023</cell-id>
      <type>LTE-CELL</type>
      <n-id-cell>123</n-id-cell>
      <tac>1</tac>
      <dl-earfcn>400</dl-earfcn>
      <ul-earfcn>18400</ul-earfcn>
      <bandwidth>5</bandwidth>
      <rf-port>1</rf-port>
      <root-sequence-index>100</root-sequence-index>
      <n-antenna-dl>1</n-antenna-dl>
      <n-antenna-ul>1</n-antenna-ul>
      <cell-gain>1.0</cell-gain>
      <band>1</band>
      <tdd-config>
        <tdd-config-id>0</tdd-config-id>
        <uldl-config>0</uldl-config>
        <sp-config>0</sp-config>
      </tdd-config>
    </cell>
  </cell-list>
</enb>

>
```

Figure 41: Netconf-Manager and Netconf-server logs 2

Moreover, if we check the OAI API, where the eNB is deployed, we can see the logs of the initialization of the eNB.

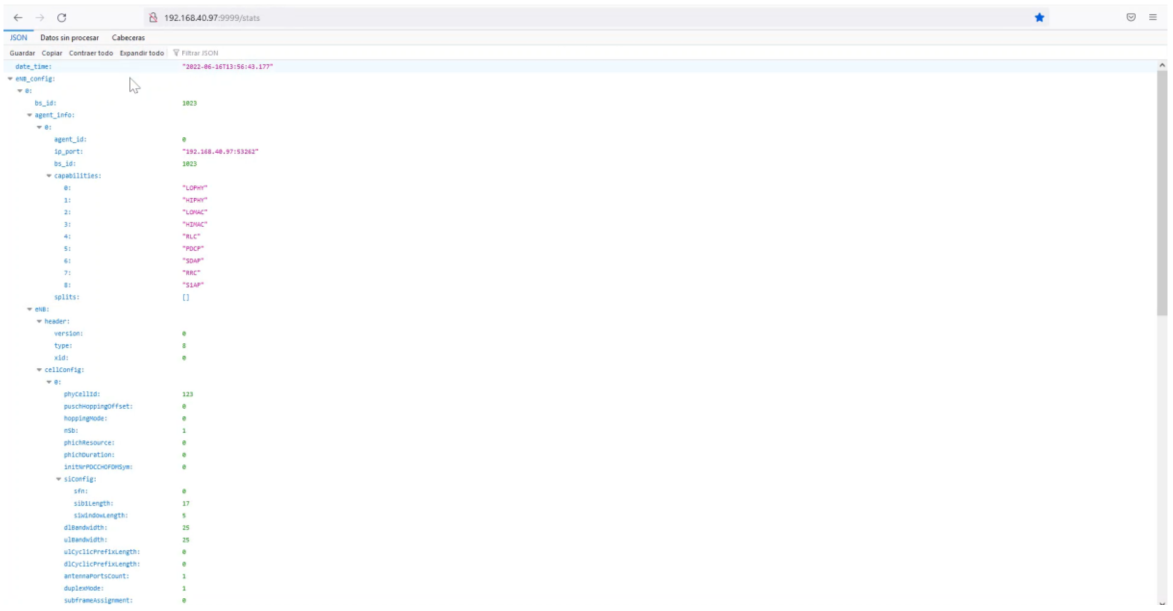


```
[MAC]   [eNB 0] Frame 163, Subframe 2 CC_id 0 MAC CE_LCID 27 (ce 0/3): CRNTI 7e15 (UE_id 0) in Msg3
[RRC]   [eNB 0] Received C-RNTI ,but UE 7e15 status(3) not RRC_RECONFIGURED
[MAC]   [eNB 0] Frame 187, Subframe 2 CC_id 0 MAC CE_LCID 27 (ce 0/3): CRNTI 7e15 (UE_id 0) in Msg3
[RRC]   [eNB 0] Received C-RNTI ,but UE 7e15 status(3) not RRC_RECONFIGURED
[MAC]   [eNB 0] Frame 209, Subframe 2 CC_id 0 MAC CE_LCID 27 (ce 0/3): CRNTI 7e15 (UE_id 0) in Msg3
[RRC]   [eNB 0] Received C-RNTI ,but UE 7e15 status(3) not RRC_RECONFIGURED
[MAC]   [eNB 0] Frame 235, Subframe 2 CC_id 0 MAC CE_LCID 27 (ce 0/3): CRNTI 7e15 (UE_id 0) in Msg3
[RRC]   [eNB 0] Received C-RNTI ,but UE 7e15 status(3) not RRC_RECONFIGURED
[MAC]   [eNB 0] Frame 257, Subframe 2 CC_id 0 MAC CE_LCID 27 (ce 0/3): CRNTI 7e15 (UE_id 0) in Msg3
[RRC]   [eNB 0] Received C-RNTI ,but UE 7e15 status(3) not RRC_RECONFIGURED
[MAC]   UE 0 rnti 7e15: UL Failure after repeated PDCCH orders: Triggering RRC
[RRC]   Frame 259, Subframe 3: UE 7e15 UL failure, activating timer
[MAC]   [eNB 0] Frame 265, Subframe 2 CC_id 0 MAC CE_LCID 27 (ce 0/3): CRNTI 7e15 (UE_id 0) in Msg3
[RRC]   [eNB 0] Received C-RNTI ,but UE 7e15 status(3) not RRC_RECONFIGURED
[MAC]   [eNB 0] Frame 275, Subframe 2 CC_id 0 MAC CE_LCID 27 (ce 0/3): CRNTI 7e15 (UE_id 0) in Msg3
[RRC]   [eNB 0] Received C-RNTI ,but UE 7e15 status(3) not RRC_RECONFIGURED
[MAC]   [eNB 0] Frame 285, Subframe 2 CC_id 0 MAC CE_LCID 27 (ce 0/3): CRNTI 7e15 (UE_id 0) in Msg3
[RRC]   [eNB 0] Received C-RNTI ,but UE 7e15 status(3) not RRC_RECONFIGURED
[RRC]   Removing UE de24 instance, because of uplink failure timer timeout
[RRC]   [eNB 0] S1AP_UE_CONTEXT_RELEASE_REQ sent for RNTI de24, cause 21, radio connection with ue lost
[S1AP]   Failed to find ue context associated with eNB ue s1ap id: 0
[S1AP]   Failed to find ue context associated with eNB ue s1ap id: 0
[RRC]   Removing UE de24 instance, because of UE_CONTEXT_RELEASE_COMMAND not received after 100 ms from sending request
[RRC]   [FRAME 00426][eNB][MOD 00][RNTI de24] Logical Channel DL-DCCH, Generate RRCConnectionRelease (bytes 3)
[RRC]   sent RRC_DCCH_DATA_REQ to TASK_PDCP_ENB
[GTPU]   [0] Delete GTP tunnels for rnti: de24, but no tunnel exits
[RRC]   Removing UE de24 instance after UE_CONTEXT_RELEASE_Complete (ue_release_timer_rrc timeout)
[RRC]   [eNB 0] Removing UE RNTI de24
[RRC]   Put UE de24 into freeList
[MAC]   clean eNb uci_vars[1] UE de24
[MAC]   clean eNb uci_vars[2] UE de24
[MAC]   clean eNb uci_vars[3] UE de24
[RRC]   remove UE de24 from ul_config_pdu_list 0/1
[MAC]   Removing UE 2 from Primary CC_id 0 (rnti de24)
[PDCP]   remove uid is 2/2 de24
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI de24] Removed UE context
[RRC]   [release_UE_in_freeList] remove UE de24 from freeList
[RRC]   Removing UE 7e15 instance, because of uplink failure timer timeout
[RRC]   [eNB 0] S1AP_UE_CONTEXT_RELEASE_REQ sent for RNTI 7e15, cause 21, radio connection with ue lost
[S1AP]   Failed to find ue context associated with eNB ue s1ap id: 0
[S1AP]   Failed to find ue context associated with eNB ue s1ap id: 0
[RRC]   Removing UE 7e15 instance, because of UE_CONTEXT_RELEASE_COMMAND not received after 100 ms from sending request
[RRC]   [FRAME 00221][eNB][MOD 00][RNTI 7e15] Logical Channel DL-DCCH, Generate RRCConnectionRelease (bytes 3)
[RRC]   sent RRC_DCCH_DATA_REQ to TASK_PDCP_ENB
[GTPU]   [0] Delete GTP tunnels for rnti: 7e15, but no tunnel exits
[RRC]   Removing UE 7e15 instance after UE_CONTEXT_RELEASE_Complete (ue_release_timer_rrc timeout)
[RRC]   [eNB 0] Removing UE RNTI 7e15
[RRC]   Put UE 7e15 into freeList
[MAC]   clean eNb uci_vars[0] UE 7e15
[MAC]   clean eNb uci_vars[1] UE 7e15
[RRC]   remove UE 7e15 from ul_config_pdu_list 0/1
[MAC]   Removing UE 0 from Primary CC_id 0 (rnti 7e15)
[PDCP]   remove uid is 0/0 7e15
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 7e15] Removed UE context
[RRC]   [release_UE_in_freeList] remove UE 7e15 from freeList
```

Figure 42: OAI eNB log

With this five steps configuration, a cell is up and running, now it is time to introduce the different services, the core network, PLMN and also the uplink and downlink quotes (if needed).

Moreover, we can also check the eNB information in the real time controller FlexRAN.



```
date_time:                    "2022-06-16T13:56:43.177"
▾ eNB_config:
  ▾ 0:
      bs_id:                  1023
    ▾ agent_info:
      ▾ 0:
          agent_id:           0
          ip_port:            "192.168.40.97:53262"
          bs_id:              1023
        ▾ capabilities:
            0:                "LOPHY"
            1:                "HIPHY"
            2:                "LOMAC"
            3:                "HIMAC"
            4:                "RLC"
            5:                "PDCP"
            6:                "SDAP"
            7:                "RRC"
            8:                "S1AP"
          splits:             []
    ▾ eNB:
      ▾ header:
          version:            0
          type:               8
          xid:                0
      ▾ cellConfig:
        ▾ 0:
            phyCellId:        123
            puschHoppingOffset: 0
            hoppingMode:      0
            nSb:              1
            phichResource:    0
            phichDuration:    0
            initNrPDCCHOFDMSym: 0
          ▾ siConfig:
              sfn:            0
              sib1Length:     17
              siWindowLength: 5
            dlBandwidth:      25
            ulBandwidth:      25
            ulCyclicPrefixLength: 0
            dlCyclicPrefixLength: 0
            antennaPortsCount: 1
            duplexMode:       1
            subframeAssignment: 0
```

Figure 43: FlexRAN information

Before the creation of the services, it is mandatory to have core networks running in order to have connectivity once those are introduced in the eNB.

<u>Core network deployment</u>

Core networks are deployed using Open5Gs, specifically inside docker-compose containers. Even though , in section 4.1 it was said that core network was only deployed in NUC1, for this test, it has been also deployed another core network in NUC2 to have multiple core's running and test the slicing part properly.

So, in order to deploy the core, we do it with the command

```
$ docker-compose -f nsa-deploy.yaml up -d
```

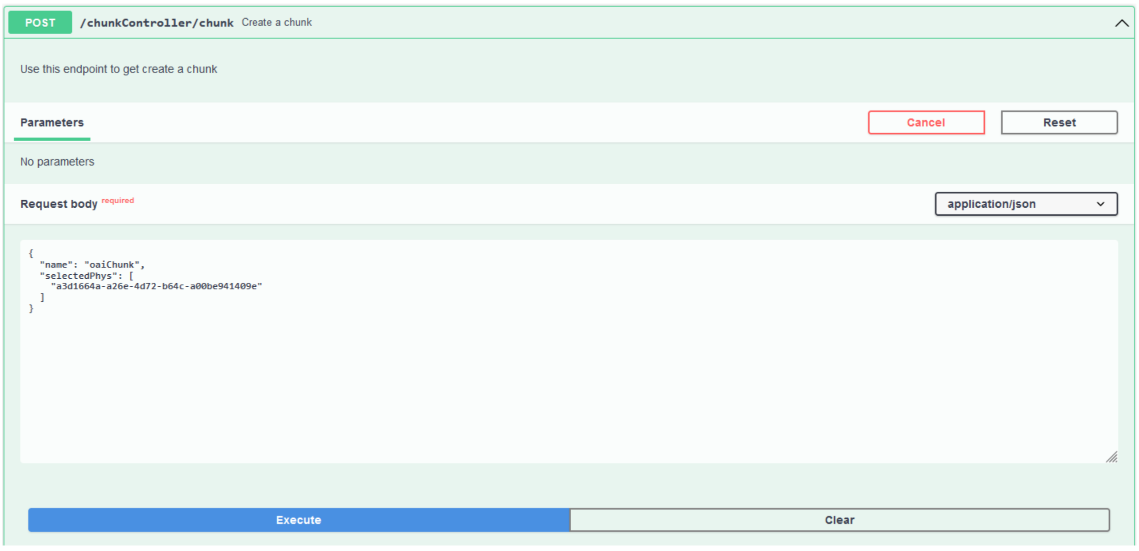
Where the *nsa-deploy.yaml* deploys both a 4G and 5G NSA core. Its content can be found in the annexed part 2. As we need two different cores, we have to run this command in both NUCs, with different nsa-deploy.yaml parameters, one for the Quectel, and the other one for the SimCom, as related in table 2 and 3.

Moreover, once the core is up and running, we need to introduce the subscriber information to stablish connection with UEs, as in figure 20. As we have two different UEs, with two different SIMs and in consequence different information.

Once they are up and running, it is time to deploy services in RACOON.

<u>Chunk and service configuration</u>

But before doing that, as said before, in order to generate a service, we need to first create a chunk.


POST `/chunkController/chunk` Create a chunk

Use this endpoint to get create a chunk

Parameters

No parameters

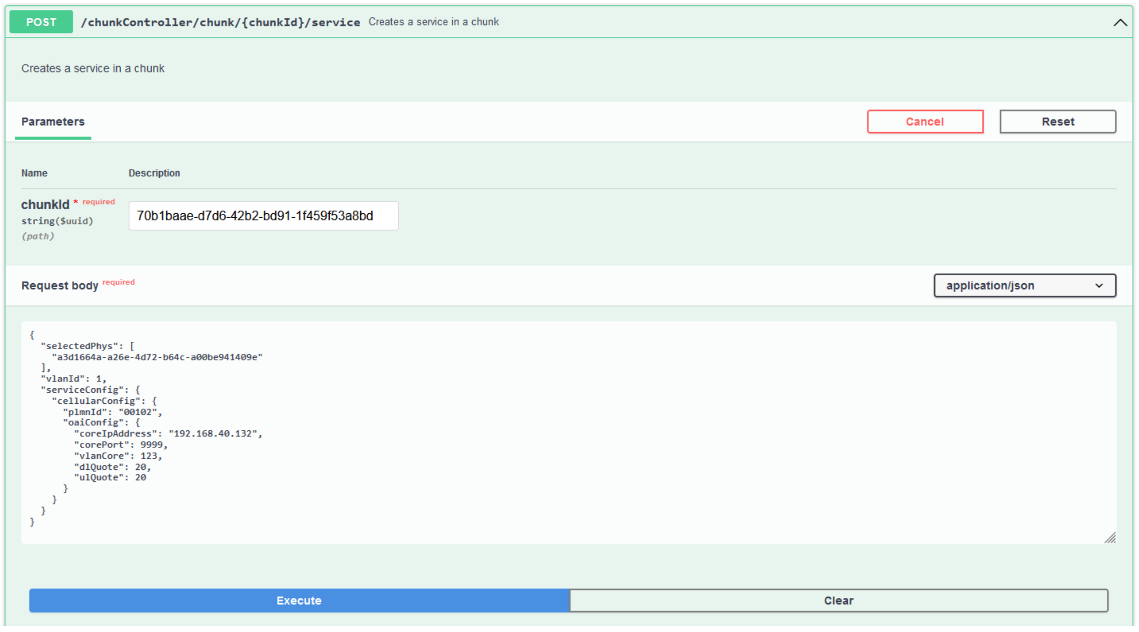
Request body required

```
{
  "name": "oaiChunk",
  "selectedPhys": [
    "a3d1664a-a26e-4d72-b64c-a00be941409e"
  ]
}
```

Figure 44: Chunk demo

To do so, we need to add the physical interface (cell id) inside the chunk call, so it is referenced to the cell we want to add configurations. If the requested body is correct, it will return a code 200 as well as a unique chunk id.

After this, it is possible to start configuring the different services. We will add two services, to add both core networks with the PLMN ID and the dl and ul quotes.


POST `/chunkController/chunk/{chunkId}/service` Creates a service in a chunk

Creates a service in a chunk

Parameters

Name	Description
chunkId * required string($uuid) (path)	70b1baae-d7d6-42b2-bd91-1f459f53a8bd

Request body required

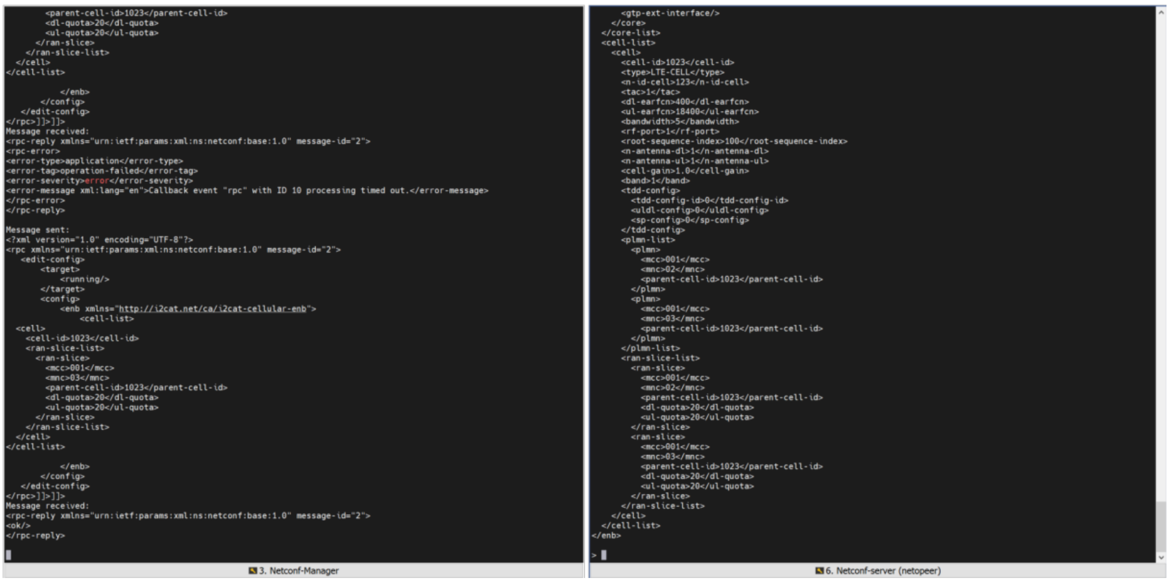
```
{
  "selectedPhys": [
    "a3d1664a-a26e-4d72-b64c-a00be941409e"
  ],
  "vlanId": 1,
  "serviceConfig": {
    "cellularConfig": {
      "plmnId": "00102",
      "oaiConfig": {
        "coreIpAddress": "192.168.40.132",
        "corePort": 9999,
        "vlanCore": 123,
        "dlQuote": 20,
        "ulQuote": 20
      }
    }
  }
}
```

Figure 45: Service demo

As can be seen, in the requested body, it is added a vlanId, and then the configuration of the cell with the PLMN id, core IP address, the core port, vlan core, and also both downlink and uplink quotes. This must be done twice, for both core networks, one placed in NUC1 and the other one in NUC2.

If the service call is correct, it returns a 200 code and a response with a unique service id.

After that, we will have the eNB configuration completed. To see it, let's check again the netconf terminal information.



```
        <parent-cell-id>1023</parent-cell-id>
        <dl-quota>20</dl-quota>
        <ul-quota>20</ul-quota>
      </ran-slice>
    </ran-slice-list>
  </cell>
</cell-list>

           </enb>
       </config>
   </edit-config>
</rpc>]]>]]>
Message received:
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
<rpc-error>
<error-type>application</error-type>
<error-tag>operation-failed</error-tag>
<error-severity>error</error-severity>
<error-message xml:lang="en">Callback event "rpc" with ID 10 processing timed out.</error-message>
</rpc-error>
</rpc-reply>

Message sent:
<?xml version="1.0" encoding="UTF-8"?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
   <edit-config>
       <target>
           <running/>
       </target>
       <config>
           <enb xmlns="http://i2cat.net/ca/i2cat-cellular-enb">
               <cell-list>
  <cell>
    <cell-id>1023</cell-id>
    <ran-slice-list>
      <ran-slice>
        <mcc>001</mcc>
        <mnc>03</mnc>
        <parent-cell-id>1023</parent-cell-id>
        <dl-quota>20</dl-quota>
        <ul-quota>20</ul-quota>
      </ran-slice>
    </ran-slice-list>
  </cell>
</cell-list>

           </enb>
       </config>
   </edit-config>
</rpc>]]>]]>
Message received:
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
<ok/>
</rpc-reply>

▌
```

3. Netconf-Manager

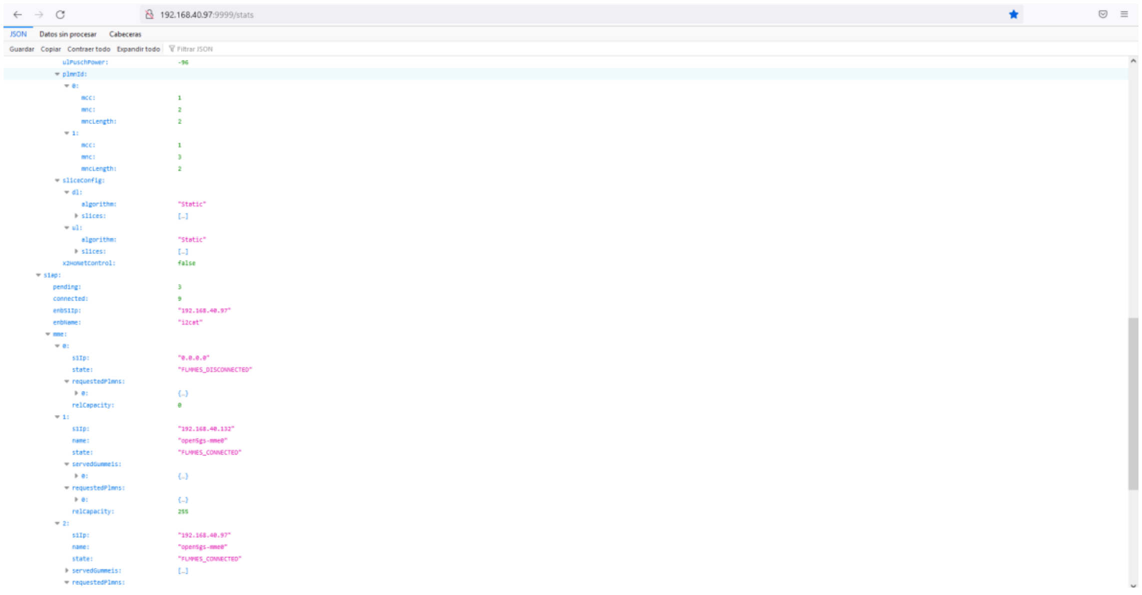
```
      <gtp-ext-interface/>
    </core>
  </core-list>
  <cell-list>
    <cell>
      <cell-id>1023</cell-id>
      <type>LTE-CELL</type>
      <n-id-cell>123</n-id-cell>
      <tac>1</tac>
      <dl-earfcn>400</dl-earfcn>
      <ul-earfcn>18400</ul-earfcn>
      <bandwidth>5</bandwidth>
      <rf-port>1</rf-port>
      <root-sequence-index>100</root-sequence-index>
      <n-antenna-dl>1</n-antenna-dl>
      <n-antenna-ul>1</n-antenna-ul>
      <cell-gain>1.0</cell-gain>
      <band>1</band>
      <tdd-config>
        <tdd-config-id>0</tdd-config-id>
        <uldl-config>0</uldl-config>
        <sp-config>0</sp-config>
      </tdd-config>
      <plmn-list>
        <plmn>
          <mcc>001</mcc>
          <mnc>02</mnc>
          <parent-cell-id>1023</parent-cell-id>
        </plmn>
        <plmn>
          <mcc>001</mcc>
          <mnc>03</mnc>
          <parent-cell-id>1023</parent-cell-id>
        </plmn>
      </plmn-list>
      <ran-slice-list>
        <ran-slice>
          <mcc>001</mcc>
          <mnc>02</mnc>
          <parent-cell-id>1023</parent-cell-id>
          <dl-quota>20</dl-quota>
          <ul-quota>20</ul-quota>
        </ran-slice>
        <ran-slice>
          <mcc>001</mcc>
          <mnc>03</mnc>
          <parent-cell-id>1023</parent-cell-id>
          <dl-quota>20</dl-quota>
          <ul-quota>20</ul-quota>
        </ran-slice>
      </ran-slice-list>
    </cell>
  </cell-list>
</enb>

> ▌
```

6. Netconf-server (netopeer)

Figure 46: Netconf-Manager and Netconf-server logs 2

On the left it is only possible to see the last message of the last service, and on the right, in the netopeer configuration, we can see the xml configuration of the cell with both PLMNs and also the different slicing.

Also, it is interesting to show the information throw FlexRAN to see if the 4G RAN has been configured properly.



```
        ulPuschPower:          -96
      ▾ plmnId:
        ▾ 0:
            mcc:               1
            mnc:               2
            mncLength:         2
        ▾ 1:
            mcc:               1
            mnc:               3
            mncLength:         2
      ▾ sliceConfig:
        ▾ dl:
            algorithm:         "Static"
          ▸ slices:            […]
        ▾ ul:
            algorithm:         "Static"
          ▸ slices:            […]
        x2HoNetControl:        false
  ▾ s1ap:
      pending:                 3
      connected:               9
      enbS1Ip:                 "192.168.40.97"
      enbName:                 "i2cat"
    ▾ mme:
      ▾ 0:
          s1Ip:                "0.0.0.0"
          state:               "FLMMES_DISCONNECTED"
        ▾ requestedPlmns:
          ▸ 0:                 {…}
          relCapacity:         0
      ▾ 1:
          s1Ip:                "192.168.40.132"
          name:                "open5gs-mme0"
          state:               "FLMMES_CONNECTED"
        ▾ servedGummeis:
          ▸ 0:                 {…}
        ▾ requestedPlmns:
          ▸ 0:                 {…}
          relCapacity:         255
      ▾ 2:
          s1Ip:                "192.168.40.97"
          name:                "open5gs-mme0"
          state:               "FLMMES_CONNECTED"
        ▸ servedGummeis:       […]
        ▾ requestedPlmns:
```

Figure 47: FlexRAN information 2

As can be seen both core networks have connected.

Now is time to add the two different UEs

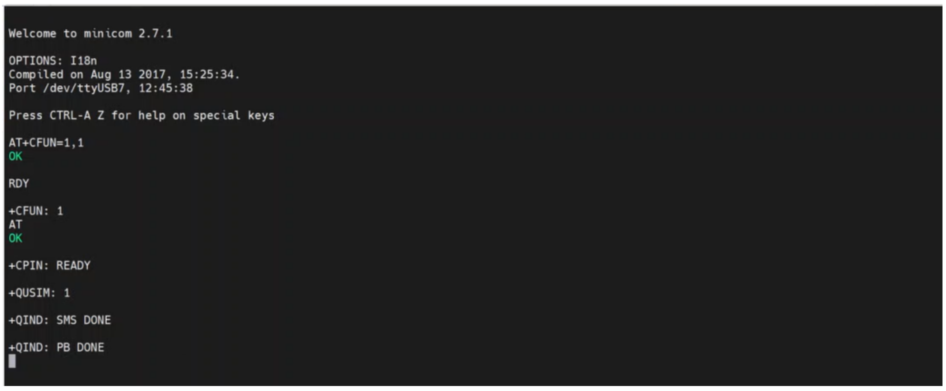
<u>UE connectivity</u>

The UEs are connected into the Raspberry Pi, the Quectel module in the device ttyUSB7 and the SimCom module in the ttyUSB3.

Next are the steps that needs to be done in order to stablish the communication with the eNB cell.

1. Check the profiles to see if the correct APN is there:
   `$ qmicli --device=/dev/cdc-wdm0 --device-open-qmi -p --wds-get-autoconnect-settings` (the device could vary in your case)
   `$ qmicli --device=/dev/cdc-wdm0 --device-open-qmi -p --wds-get-profile-list=3gpp`
   Example to modify the profiles:
   `$ qmicli -d /dev/cdc-wdm0 --wds-modify-profile=3gpp,1,'apn=apntest'`
2. After that, just reboot ther radio of the modem (disconnect it manually) or use `AT+CFUN=1,1` using "minicom -d /dev/ttyUSB3" (or /dev/ttyUSB7)
3. Put the modem in raw_ip mode:
   `$ echo Y > /sys/class/net/wwan0/qmi/raw_ip`
4. If echo command fails, do:
   ```
   $ sudo ifconfig wwan0 down
   $ echo Y > /sys/class/net/wwan0/qmi/raw_ip
   $ sudo ifconfig wwan0 up
   ```

Next is an screenshot of the reboot of the quectel, and its connection to the eNB.



```
Welcome to minicom 2.7.1

OPTIONS: I18n
Compiled on Aug 13 2017, 15:25:34.
Port /dev/ttyUSB7, 12:45:38

Press CTRL-A Z for help on special keys

AT+CFUN=1,1
OK

RDY

+CFUN: 1
AT
OK

+CPIN: READY

+QUSIM: 1

+QIND: SMS DONE

+QIND: PB DONE
```

Figure 48: Minicom Quectel Information

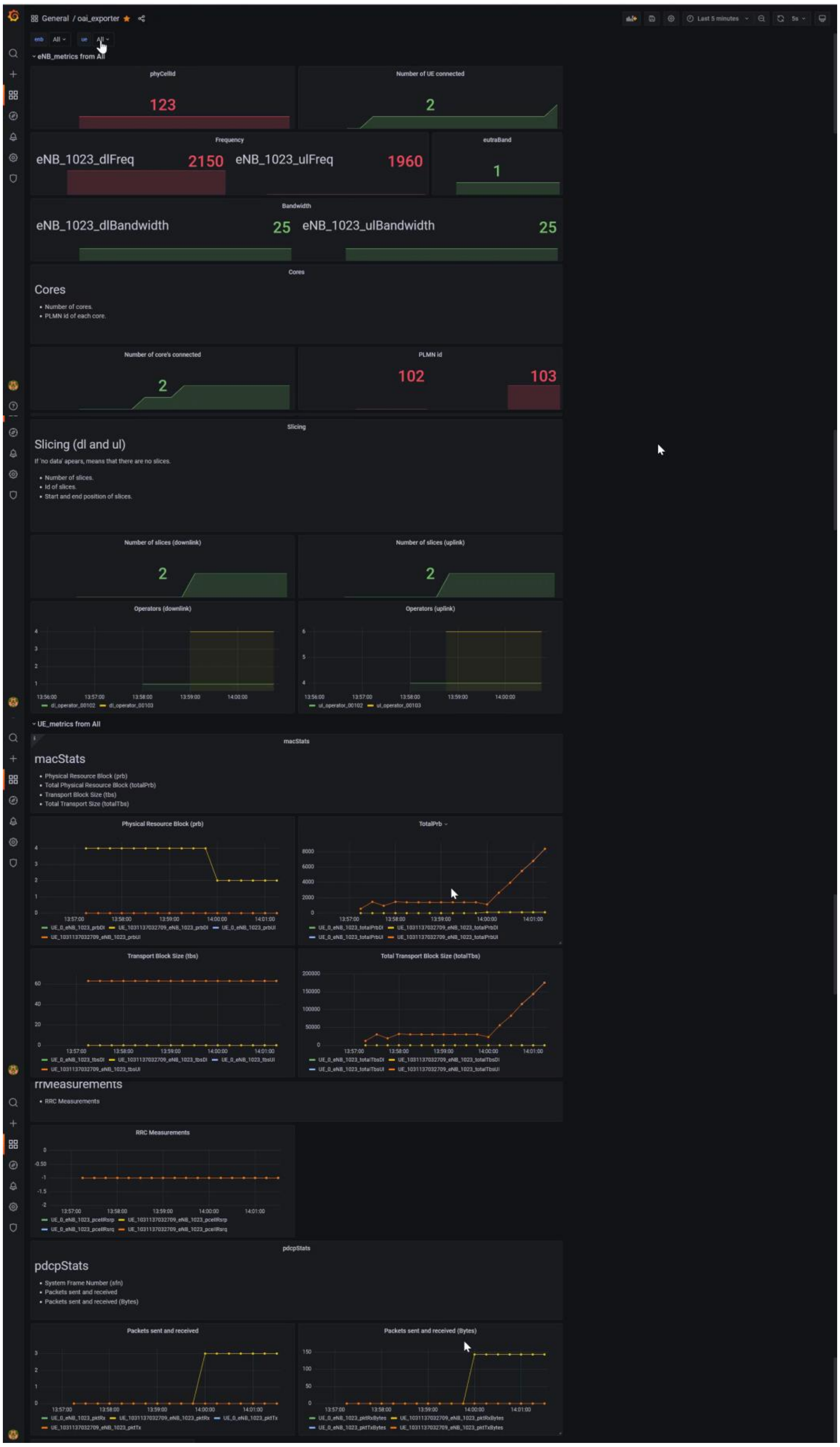
Moreover, and finally, in order to expose all these metrics, it has been developed a Grafana dashboard, with the next parameters.

In eNB_metrics:

PhyCellId, Number of UE connected, Downlink and uplink frequencies, Downlink and uplink bandwidth and Slicing information.

In UE_metrics:

MacStats, Physical Resource Block (prb), Transport Block Size (tbs), Total Transport Size (totalTbs), rrMeasurements, pcellRsrp, pcellRsrq, pdcpStats, System Frame Number (sfn), Packets sent and received and Packets sent and received (Bytes).



eNB_metrics from All

phyCellId: 123

Number of UE connected: 2

Frequency: eNB_1023_dlFreq 2150, eNB_1023_ulFreq 1960

eutraBand: 1

Bandwidth: eNB_1023_dlBandwidth 25, eNB_1023_ulBandwidth 25

Cores

- Number of cores.
- PLMN id of each core.

Number of core's connected: 2

PLMN id: 102, 103

Slicing

Slicing (dl and ul)

If 'no data' apears, means that there are no slices.

- Number of slices.
- Id of slices.
- Start and end position of slices.

Number of slices (downlink): 2

Number of slices (uplink): 2

Operators (downlink)

Operators (uplink)

UE_metrics from All

macStats

- Physical Resource Block (prb)
- Total Physical Resource Block (totalPrb)
- Transport Block Size (tbs)
- Total Transport Size (totalTbs)

Physical Resource Block (prb)

UE_0_eNB_1023_prbDl
UE_1031137032709_eNB_1023_prbDl
UE_0_eNB_1023_prbUl
UE_1031137032709_eNB_1023_prbUl

TotalPrb

UE_0_eNB_1023_totalPrbDl
UE_1031137032709_eNB_1023_totalPrbDl
UE_0_eNB_1023_totalPrbUl
UE_1031137032709_eNB_1023_totalPrbUl

Transport Block Size (tbs)

UE_0_eNB_1023_tbsDl
UE_1031137032709_eNB_1023_tbsDl
UE_0_eNB_1023_tbsUl
UE_1031137032709_eNB_1023_tbsUl

Total Transport Block Size (totalTbs)

UE_0_eNB_1023_totalTbsDl
UE_1031137032709_eNB_1023_totalTbsDl
UE_0_eNB_1023_totalTbsUl
UE_1031137032709_eNB_1023_totalTbsUl

rrMeasurements

- RRC Measurements

RRC Measurements

UE_0_eNB_1023_pcellRsrp
UE_1031137032709_eNB_1023_pcellRsrp
UE_0_eNB_1023_pcellRsrq
UE_1031137032709_eNB_1023_pcellRsrq

pdcpStats

- System Frame Number (sfn)
- Packets sent and received
- Packets sent and received (Bytes)

Packets sent and received

UE_0_eNB_1023_pktRx
UE_1031137032709_eNB_1023_pktRx
UE_0_eNB_1023_pktTx
UE_1031137032709_eNB_1023_pktTx

Packets sent and received (Bytes)

UE_0_eNB_1023_pktRxBytes
UE_1031137032709_eNB_1023_pktRxBytes
UE_0_eNB_1023_pktTxBytes
UE_1031137032709_eNB_1023_pktTxBytes

Figure 49: OAI Exporter demo

As can be seen, OAI exporter, is in real time, and shows the different information of both the eNB and UEs.
In the dashboard the information can be filtered by two different elements, first of all by eNB id, and second of all, by UEs IMSI id.
Moreover, the eNB has been built with the name *eNB_ENBid*, where *ENBid* is the id of the cell, and the UE with the name *UE_IMSInum_eNB_ENBid, where IMSInum* is the UE's IMSI number and *ENBid* is the id of the cell where it is connected.
In the figure above, appears one eNB (eNB_1023) and two different UEs (UE_0_eNB_1023 and UE_001031136032709_eNB_1023). The first UE, corresponds to the Quectel UE, and it appears 0 as its IMSI, because it needs a reboot to appear the correct number, the second UE does not need this reboot.

5. Budget

The different costs of this work can be divided into the main components that are needed to deploy the service, the salary and the different amortization of the computer. Table 6 shows the cost different components and table 7 shows the salary and amortization costs. In this project everything that has been used is open-source so there is no need to include it.

Regarding the salary, it has been taking into consideration an i2cat research intern engineer performing 800 hours job with a salary of 10€ for hour worked, taxes included. Moreover, about the amortization of the computer, it has been considered a 700€ PC with a residual value of 70€ and life span of 5 year

Product	Price / Unit (€)	Number of units	Total cost
Raspberry Pi 2	40	1	40
SimCom Module	23	1	23
Quectel Module	30	1	39
USRP b210	1720	2	3440
Total cost			3542

Table 6: Components cost

Concept	Cost (€)
Salary (including social charges)	8000
Amortization	70
Total cost	8070

Table 7: Salary and amortization costs

The total cost of the project for 7 months has been **11612€**

6. Conclusions and future development

This project has been developed with the intention of implement a centralized radio controller to manage different 4G and 5G small cells with OpenAirInterface technology for the creation of an automated deployment of the different network services over the different cells. To this end, a 4G OAI small cells management has been implemented in the i2cat radio controller RACOON, and 5G OAI small cells will be implemented in a near future when the different bugs are fixed.

This implementation has shown that FlexRAN is a powerful tool that allows to perform many operations. Firstly, it allows to monitor the different RAN infrastructure, with the different eNBs and UEs, both in a high level, for example, with the monitoring of the different number of packets or bytes sent by each UE, and in a low level, for example, with the monitoring of the different core networks, PLMN ids or slicing quotes from an eNB. And secondly, it allows to configure the different eNB parameters, for example, the core network, PLMN or network slicing. Moreover, in order to visualize this FlexRAN data interactively, it has been possible to store the data in a Prometheus server and visualize it in a Grafana Dashboard in real-time.

Moreover, it has shown that it is possible to automate the OpenAirInterface technology throw the RAN controller tool RACOON, which was already developed for other devices such as Amarisoft or Accellerant.

And finally, RAN configurations, can be largely automated using both Netconf and an OAI REST API. This separation allows to have a centralized and distributed system at the same time, as RAN deployments are managed separately from its service management.

6.1. Future Work

There is a lot of effort to have a fully functional 5G network with no bugs from part of OpenAirInterface Software Alliance and finally have the opportunity to integrate it into the centralized ran controller.

Also, next generation FlexRAN, called FlexRIC, should be implemented, because it is a software suite that contains two components, first, a RAN agent that allows for interfacing with the radio stack, and also a real-time controller for 5G.

Moreover, with the OAI network scenario completed, it opens a lot of new opportunities to develop and test with 4G and 5G networks, such as, add different types of frameworks of a virtualized network, or the possibility of separate the Centralized Unit (CU) and Distributed Unit (DU) to reduce different parameters such as latency.

Bibliography

[1] Juan Pedro Tomás, *Carrier aggregation es clave para coexistencia entre 4G y 5G: Ericsson*, 2020

[2] Jim Zyren, *Overview of the 3GPP Long Term Evolution Physical Layer*

[3] Advantages of 4G-LTE. 4G LTE <*https://about4glte.wordpress.com/advantages/*>

[4] LTE Network Infrastructure and Elements <*https://sites.google.com/site/lteencyclopedia/lte-network-infrastructure-and-elements*> [consulted: 12 of June 2022]

[5] Marcin Dryjanski, Ph.D., *Mobile network architecture – 4G design issues*, 2015

[6] 5G Non Standalone Solution Overview <*https://www.cisco.com/c/en/us/td/docs/wireless/asr_5000/21-10_6-4/5G-NSA-Solution/21-10-5G-NSA-Solution-Guide/21-10-5G-NSA-Solution-Guide_chapter_01.pdf*> [consulted: 12 of June 2022]

[7] Olaonipekun Oluwafemi Erunkulu et al, *5G Mobile Communication Applications: A Survey and Comparison of Use Cases*, 2021

[8] OAI website <*https://openairinterface.org/*> [consulted: 10 of June 2022]

[9] Functional Split Architecture repository <*https://gitlab.eurecom.fr/oai/openairinterface5g/blob/develop/doc/FEATURE_SET.md#openairinterface-4g-lte-enb-feature-set*> [consulted: 12 of June 2022]

[10] Xenofon Foukas et al. *FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks*, 2016

[11] FlexRAN web page <*https://mosaic5g.io/flexran/*> [consulted: 10 of June 2022]

[12] Stefan Vallin, *Automating network and service configuration using NETCONF and YANG*, 2011

[13] Open5GS Quick Start <https://open5gs.org/open5gs/docs/guide/01-quickstart/> [consulted: 10 of March 2022]

[14] Grafana Support for Prometheus website <*https://prometheus.io/docs/visualization/grafana/*> [consulted: 2 of June 2022]

[15] Github Open5Gs <*https://github.com/open5gs*> [consulted: 2 of June 2022]

[16] GitHub docker Open5Gs <*https://github.com/herlesupreeth/docker_open5gs*> [consulted: 2 of June 2022]

[17] FlexRAN Mosaic5G Github repository <*https://gitlab.eurecom.fr/mosaic5g/mosaic5g/-/wikis/tutorials/flexran*>

[18] OpenAirInterface5G Github repository <*https://gitlab.eurecom.fr/oai/openairinterface5g*>

[19] Saleh Alrkiyan, *Cyclic Redundancy Check CRC*, 2017

[20] Lucas Nóvoa, Virgínia Tavares, et al. *RAN Slicing using OpenAirInterface and FlexRAN in a Virtualized Scenario,* 2020

[21] Yu Abiko; Takato Saito, et al. *Flexible Resource Block Allocation to Multiple Slices for Radio Access Network Slicing Using Deep Reinforcement Learning*

[22] Cristina-Elena Vintilă Victor-Valeriu Patriciu, Ion Bica. *Security Analysis of LTE Access Network*. 2011

Annex 1 – Open5Gs environment files

In this part, it is presented the two different environment files with all the relevant parameters for both core's deployed. One with the Quectel information, and the other one with the SimCom information. In bold is highlighted the different mobile subscriber information.

.env core1 (NUC1 192.168.40.132)

```
#TZ=Europe/Berlin

MCC=001
MNC=02

TEST_NETWORK=172.22.0.0/24
DOCKER_HOST_IP=192.168.41.68

# MONGODB
MONGO_IP=172.22.0.2

# HSS - open5gs
HSS_IP=172.22.0.3

# PCRF
PCRF_IP=172.22.0.4

# SGW
SGWC_IP=172.22.0.5
SGWU_IP=172.22.0.6
SGWU_ADVERTISE_IP=172.22.0.6

# SMF
SMF_IP=172.22.0.7

# UPF
UPF_IP=172.22.0.8
UPF_ADVERTISE_IP=172.22.0.8
```

```
# MME
MME_IP=172.22.0.9

# AMF
AMF_IP=172.22.0.10

# AUSF
AUSF_IP=172.22.0.11

# NRF
NRF_IP=172.22.0.12

# UDM
UDM_IP=172.22.0.13

# UDR
UDR_IP=172.22.0.14

# IMS DNS
DNS_IP=172.22.0.15

# RTPENGINE
RTPENGINE_IP=172.22.0.16

# MYSQL
MYSQL_IP=172.22.0.17

# FHOSS
FHOSS_IP=172.22.0.18
# ICSCF
ICSCF_IP=172.22.0.19

# SCSCF
SCSCF_IP=172.22.0.20
```

```
# PCSCF
PCSCF_IP=172.22.0.21

# SRSLTE ENB
SRS_ENB_IP=172.22.0.22

# UERANSIM
NR_GNB_IP=172.22.0.23
NR_UE_IP=172.22.0.24

UE1_IMEI=356938035643803
UE1_IMEISV=4370816125816151
UE1_IMSI=001025432100004
UE1_KI=00112233445566778899AABBCCDDEEFF
UE1_OP=000102030405060708090A0B0C0D0E0F
UE1_AMF=8000

# OAI ENB
OAI_ENB_IP=172.22.0.25

# OPEN5GS WEBUI
WEBUI_IP=172.22.0.26

# PCF
PCF_IP=172.22.0.27

# NSSF
NSSF_IP=172.22.0.28

# BSF
BSF_IP=172.22.0.29
```

.env core2 (NUC1 192.168.40.97)

```
#TZ=Europe/Berlin

MCC=001
MNC=03

TEST_NETWORK=172.22.0.0/24
DOCKER_HOST_IP=192.168.40.97

# MONGODB
MONGO_IP=172.22.0.2

# HSS - open5gs
HSS_IP=172.22.0.3

# PCRF
PCRF_IP=172.22.0.4

# SGW
SGWC_IP=172.22.0.5
SGWU_IP=172.22.0.6
SGWU_ADVERTISE_IP=192.168.40.97

# SMF
SMF_IP=172.22.0.7

# UPF
UPF_IP=172.22.0.8
UPF_ADVERTISE_IP=192.168.40.97

# MME
MME_IP=172.22.0.9

# AMF
AMF_IP=172.22.0.10
```

```
# AUSF
AUSF_IP=172.22.0.11

# NRF
NRF_IP=172.22.0.12

# UDM
UDM_IP=172.22.0.13

# UDR
UDR_IP=172.22.0.14

# IMS DNS
DNS_IP=172.22.0.15

# RTPENGINE
RTPENGINE_IP=172.22.0.16

# MYSQL
MYSQL_IP=172.22.0.17

# FHOSS
FHOSS_IP=172.22.0.18

# ICSCF
ICSCF_IP=172.22.0.19

# SCSCF
SCSCF_IP=172.22.0.20

# PCSCF
PCSCF_IP=172.22.0.21

# SRSLTE ENB
```

```
SRS_ENB_IP=172.22.0.22

# UERANSIM
NR_GNB_IP=172.22.0.23
NR_UE_IP=172.22.0.24

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

# OAI ENB
OAI_ENB_IP=172.22.0.25

# OPEN5GS WEBUI
WEBUI_IP=172.22.0.26

# PCF
PCF_IP=172.22.0.27

# NSSF
NSSF_IP=172.22.0.28

# BSF
BSF_IP=172.22.0.29

# ENTITLEMENT SERVER
ENTITLEMENT_SERVER_IP=172.22.0.30

# OSMOMSC
OSMOMSC_IP=172.22.0.31

# OSMOHLR
```

```
OSMOHLR_IP=172.22.0.32


# SMSC
SMSC_IP=172.22.0.33
```

Annex 2 – Open5Gs nsa-deploy.yaml

Next it is exposed a part of the .yaml file in order to deploy a nsa network with open5gs. Moreover, it is only exposed the part for the mme, hss, sgw, pgw and webui as are the main components used and modified from open5gs for the 4G Core Network deployment.

```
version: '3'

services:

  webui:

    image: docker_open5gs

    container_name: webui

    depends_on:

      - mongo

    env_file:

      - .env

    environment:

      - COMPONENT_NAME=webui

    volumes:

      - ./webui:/mnt/webui

      - /etc/timezone:/etc/timezone:ro

      - /etc/localtime:/etc/localtime:ro

    expose:

      - "3000/tcp"

    ports:

      - "3000:3000/tcp"

    networks:

      default:

        ipv4_address: ${WEBUI_IP}

  hss:

    image: docker_open5gs

    container_name: hss

    env_file:

      - .env
    environment:
      - COMPONENT_NAME=hss-1
    volumes:
      - ./hss:/mnt/hss
      - ./log:/open5gs/install/var/log/open5gs
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
    depends_on:
      - mongo
    expose:
      - "3868/udp"
      - "3868/tcp"
      - "3868/sctp"
      - "5868/udp"
      - "5868/tcp"
      - "5868/sctp"
    networks:
      default:
        ipv4_address: ${HSS_IP}
  sgwc:
    image: docker_open5gs
    depends_on:
      - smf
      - upf
    container_name: sgwc
    env_file:
      - .env
    environment:
      - COMPONENT_NAME=sgwc-1
    volumes:
      - ./sgwc:/mnt/sgwc
      - ./log:/open5gs/install/var/log/open5gs
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
    expose:
      - "2123/udp"
      - "8805/udp"
    networks:
      default:
        ipv4_address: ${SGWC_IP}
  sgwu:
    image: docker_open5gs
    depends_on:
      - smf
      - upf
    container_name: sgwu
    env_file:
      - .env
    environment:
      - COMPONENT_NAME=sgwu-1
    volumes:
      - ./sgwu:/mnt/sgwu
      - ./log:/open5gs/install/var/log/open5gs
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
    expose:
      - "8805/udp"
      - "2152/udp"
    ports:
      - "2152:2152/udp"
    networks:
      default:
        ipv4_address: ${SGWU_IP}
  mme:
    image: docker_open5gs
    depends_on:
      - hss
      - sgwc
      - sgwu
      - smf
      - upf
    container_name: mme
    env_file:
      - .env
    environment:
      - COMPONENT_NAME=mme-1
    volumes:
      - ./mme:/mnt/mme
      - ./log:/open5gs/install/var/log/open5gs
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
    expose:
      - "3868/udp"
      - "3868/tcp"
      - "3868/sctp"
      - "5868/udp"
      - "5868/tcp"
      - "5868/sctp"
      - "36412/sctp"
      - "2123/udp"
```

```
ports:
  - "36412:36412/sctp"
networks:
  default:
    ipv4_address: ${MME_IP}
```

Annex 3 – OAI eNB configuration file

In this annexed part, it is exposed the *enb.fdd.conf* file with all the relevant parameters for a eNB deployment. Moreover, most of the parameters are dynamic, using an environment file with the variables presented in the section 4, but in the next configuration, the different environment variables have already been replaced.

```
Active_eNBs = ( "i2cat");
# Asn1_verbosity, choice in: none, info, annoying
Asn1_verbosity = "none";

eNBs =
(
 {
    ////////// Identification parameters:
    eNB_ID    =  555; #123;
    cell_type =  "CELL_MACRO_ENB";
    eNB_name  =  "i2cat";

    // Tracking area code, 0x0000 and 0xfffe are reserved values
    tracking_area_code = 1;
    plmn_list = ( { mcc = 0; mnc = 0; mnc_length = 2; } );

    tr_s_preference      = "local_mac"
    nr_cellid            = 1023;

    // In seconds
    rrc_inactivity_threshold = 0;

    ////////// Physical parameters:

    component_carriers = (
      {
      node_function             = "3GPP_eNODEB";
      node_timing               = "synch_to_ext_device";
      node_synch_ref            = 0;
      frame_type                = "FDD";
      tdd_config                = 3;
      tdd_config_s              = 0;
      prefix_type               = "NORMAL";
      eutra_band                = 1;
      downlink_frequency        = 2150000000L;
      uplink_frequency_offset   = -190000000;
      Nid_cell                  = 123;
      N_RB_DL                   = 25;
      Nid_cell_mbsfn            = 500;#123;
      nb_antenna_ports          = 1;
      nb_antennas_tx            = 1;
      nb_antennas_rx            = 1;
      tx_gain                   = 90;
      rx_gain                   = 125;
      pbch_repetition           = "FALSE";
      prach_root                = 0;
      prach_config_index        = 0; #100;
```

```
         prach_high_speed          = "DISABLE";
         prach_zero_correlation    = 1;
         prach_freq_offset         = 2;
         pucch_delta_shift         = 1;
         pucch_nRB_CQI             = 0;
         pucch_nCS_AN              = 0;
         pucch_n1_AN               = 0;
         pdsch_referenceSignalPower= -25;
         pdsch_p_b                 = 0;
         pusch_n_SB                = 1;
         pusch_enable64QAM         = "DISABLE";
         pusch_hoppingMode         = "interSubFrame";
         pusch_hoppingOffset       = 0;
         pusch_groupHoppingEnabled = "ENABLE";
         pusch_groupAssignment     = 0;
         pusch_sequenceHoppingEnabled = "DISABLE";
         pusch_nDMRS1              = 1;
         phich_duration            = "NORMAL";
         phich_resource            = "ONESIXTH";
         srs_enable                = "DISABLE";
   /*
         srs_BandwidthConfig       =;
         srs_SubframeConfig        =;
         srs_ackNackST             =;
         srs_MaxUpPts              =;
   */

         pusch_p0_Nominal          = -96;
         pusch_alpha               = "AL1";
         pucch_p0_Nominal          = -104;
         msg3_delta_Preamble       = 6;
         pucch_deltaF_Format1      = "deltaF2";
         pucch_deltaF_Format1b     = "deltaF3";
         pucch_deltaF_Format2      = "deltaF0";
         pucch_deltaF_Format2a     = "deltaF0";
         pucch_deltaF_Format2b     = "deltaF0";

         rach_numberOfRA_Preambles                 = 64;
         rach_preamblesGroupAConfig                = "DISABLE";
         rach_powerRampingStep                     = 4;
         rach_preambleInitialReceivedTargetPower   = -108;
         rach_preambleTransMax                     = 10;
         rach_raResponseWindowSize                 = 10;
         rach_macContentionResolutionTimer         = 48;
         rach_maxHARQ_Msg3Tx                       = 4;

         pcch_default_PagingCycle                  = 128;
         pcch_nB                                   = "oneT";
         bcch_modificationPeriodCoeff              = 2;
         ue_TimersAndConstants_t300                = 1000;
         ue_TimersAndConstants_t301                = 1000;
         ue_TimersAndConstants_t310                = 1000;
         ue_TimersAndConstants_t311                = 10000;
         ue_TimersAndConstants_n310                = 20;
```

```
        ue_TimersAndConstants_n311                  = 1;
        ue_TransmissionMode                         = 1;

        //Parameters for SIB18
        rxPool_sc_CP_Len                                          =
"normal";
        rxPool_sc_Period                                          =
"sf40";
        rxPool_data_CP_Len                                        =
"normal";
        rxPool_ResourceConfig_prb_Num                             = 20;
        rxPool_ResourceConfig_prb_Start                           = 5;
        rxPool_ResourceConfig_prb_End                             = 44;
        rxPool_ResourceConfig_offsetIndicator_present             =
"prSmall";
        rxPool_ResourceConfig_offsetIndicator_choice              = 0;
        rxPool_ResourceConfig_subframeBitmap_present              =
"prBs40";
        rxPool_ResourceConfig_subframeBitmap_choice_bs_buf        =
"00000000000000000000";
        rxPool_ResourceConfig_subframeBitmap_choice_bs_size       = 5;
        rxPool_ResourceConfig_subframeBitmap_choice_bs_bits_unused =
0;
/*
        rxPool_dataHoppingConfig_hoppingParameter                 = 0;
        rxPool_dataHoppingConfig_numSubbands                      =
"ns1";
        rxPool_dataHoppingConfig_rbOffset                         = 0;
        rxPool_commTxResourceUC-ReqAllowed                        =
"TRUE";
*/
        // Parameters for SIB19
        discRxPool_cp_Len
= "normal"
        discRxPool_discPeriod
= "rf32"
        discRxPool_numRetx
= 1;
        discRxPool_numRepetition
= 2;
        discRxPool_ResourceConfig_prb_Num
= 5;
        discRxPool_ResourceConfig_prb_Start
= 3;
        discRxPool_ResourceConfig_prb_End
= 21;
        discRxPool_ResourceConfig_offsetIndicator_present
= "prSmall";
        discRxPool_ResourceConfig_offsetIndicator_choice
= 0;
        discRxPool_ResourceConfig_subframeBitmap_present
= "prBs40";
        discRxPool_ResourceConfig_subframeBitmap_choice_bs_buf
= "f0ffffffff";
```

```
      discRxPool_ResourceConfig_subframeBitmap_choice_bs_size
= 5;

discRxPool_ResourceConfig_subframeBitmap_choice_bs_bits_unused  =
0;

      //SSB central frequency of NR secondary cell group (for ENDC
NSA)
      nr_scg_ssb_freq = 641272;
    }
  );

    srb1_parameters :
    {
        # timer_poll_retransmit = (ms) [5, 10, 15, 20,... 250, 300,
350, ... 500]
        timer_poll_retransmit    = 80;

        # timer_reordering = (ms) [0,5, ... 100, 110,
120, ... ,200]
        timer_reordering         = 35;

        # timer_reordering = (ms) [0,5, ... 250, 300,
350, ... ,500]
        timer_status_prohibit    = 0;

        # poll_pdu = [4, 8, 16, 32 , 64, 128, 256,
infinity(>10000)]
        poll_pdu                 =  4;

        # poll_byte = (kB)
[25,50,75,100,125,250,375,500,750,1000,1250,1500,2000,3000,infinit
y(>10000)]
        poll_byte                =  99999;

        # max_retx_threshold = [1, 2, 3, 4 , 6, 8, 16, 32]
        max_retx_threshold       =  4;
    }

    # ------- SCTP definitions
    SCTP :
    {
        # Number of streams to use in input/output
        SCTP_INSTREAMS  = 2;
        SCTP_OUTSTREAMS = 2;
    };

    enable_measurement_reports = "no";

    ////////// MME parameters:
    mme_ip_address      = ( { ipv4       = "0.0.0.0";
                              ipv6       = "192:168:30::17";
                              port       = 36412 ;
                              active     = "yes";
                              preference = "ipv4";
                            }
                          );

    ///X2
    enable_x2           = "no";
    t_reloc_prep        = 1000;      /* unit: millisecond */
    tx2_reloc_overall = 2000;      /* unit: millisecond */
    t_dc_prep           = 1000;      /* unit: millisecond */
    t_dc_overall        = 2000;      /* unit: millisecond */

    NETWORK_INTERFACES :
    {
        ENB_INTERFACE_NAME_FOR_S1_MME            = "eno1";
        ENB_IPV4_ADDRESS_FOR_S1_MME              =
"192.168.40.97";
        ENB_INTERFACE_NAME_FOR_S1U               = "eno1";
        ENB_IPV4_ADDRESS_FOR_S1U                 =
"192.168.40.97";
        ENB_PORT_FOR_S1U                         = 2152; # Spec
2152
        ENB_IPV4_ADDRESS_FOR_X2C                 =
"192.168.40.97";
        ENB_PORT_FOR_X2C                         = 36422; # Spec
36422
    };
  }
);

MACRLCs =
(
  {
    num_cc             = 1;
    tr_s_preference = "local_L1";
    tr_n_preference = "local_RRC";
    phy_test_mode    = 0;
    puSch10xSnr       =  160;
    puCch10xSnr       =  160;
  }
);

L1s =
(
  {
    num_cc = 1;
    tr_n_preference = "local_mac";
  }
);

RUs =
(
  {
    local_rf                      = "yes"
    nb_tx                         = 1
    nb_rx                          = 1
    att_tx                         = 0
    att_rx                         = 0;
    bands                          = [7];
    max_pdschReferenceSignalPower = -27;
    max_rxgain                     = 95;
    eNB_instances                  = [0];
#    clock_src                      = "external";
    sdr_addrs = "serial=31DB5A3"
  }
);

THREAD_STRUCT =
(
  {
    #three config for level of parallelism
"PARALLEL_SINGLE_THREAD", "PARALLEL_RU_L1_SPLIT", or
"PARALLEL_RU_L1_TRX_SPLIT"
    parallel_config    = "PARALLEL_SINGLE_THREAD";
    #two option for worker "WORKER_DISABLE" or "WORKER_ENABLE"
    worker_config      = "WORKER_ENABLE";
  }
);

NETWORK_CONTROLLER :
{
    FLEXRAN_ENABLED        = "yes";
    FLEXRAN_INTERFACE_NAME = "eno1";
    FLEXRAN_IPV4_ADDRESS   = "192.168.40.97";
    FLEXRAN_PORT           = 2210;
    FLEXRAN_CACHE          = "/mnt/oai_agent_cache";
    FLEXRAN_AWAIT_RECONF   = "no";
};

log_config :
  {
     global_log_level                    ="info";
     global_log_verbosity                ="high";
     hw_log_level                        ="info";
     hw_log_verbosity                    ="medium";
     phy_log_level                       ="info";
     phy_log_verbosity                   ="medium";
     mac_log_level                       ="info";
     mac_log_verbosity                   ="high";
     rlc_log_level                       ="info";
     rlc_log_verbosity                   ="high";
     pdcp_log_level                      ="info";
     pdcp_log_verbosity                  ="high";
     rrc_log_level                       ="info";
     rrc_log_verbosity                   ="medium";
};
```

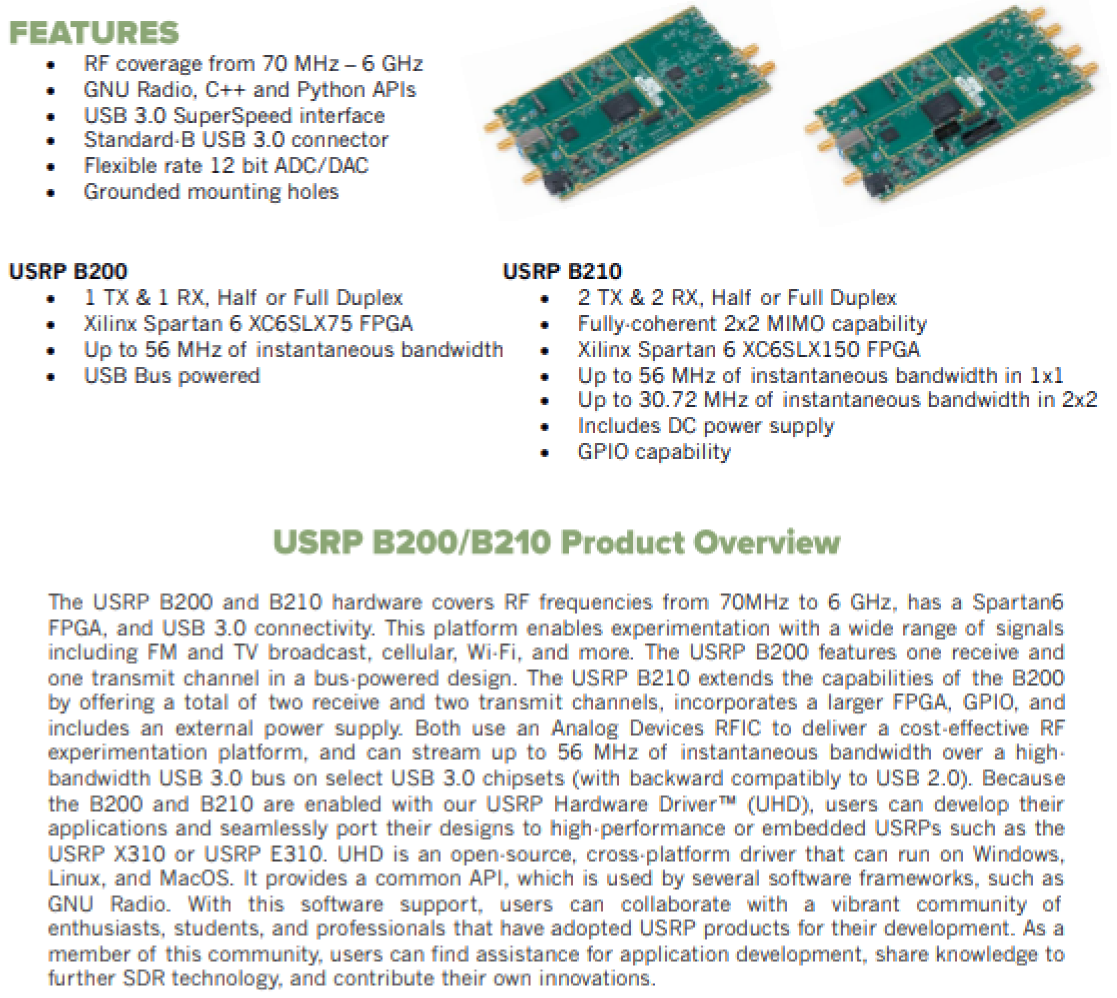
Annex 4 – E-UTRA band

Next can be found a list with the different 4G LTE frequency band. There are only the bands with the geographical area: Global, EMEA (Europe, Middle East, and Africa) and EU, as if not, the list would be too large. Moreover, the list does not include the channel bandwidth, which include frequency bands of 1.4, 3, 5, 10, 15, 20 MHz. In fact, the two different bands that have been used in this project are 1 and 7, and both includes 5, 10, 15, 20 MHz.

and	Name	Mode	Downlink (MHz)			Bandwidth	Uplink (MHz)			Duplex spacing	Geographical	3GPP
			Low	Middle	High	DL/UL (MHz)	Low	Middle	High	(MHz)	area	release
			Earfcn				Earfcn					
1	2100	FDD	2110	2140	2170	60	1920	1950	1980	190	Global	8
			0	300	599		18000	18300	18599			
3	1800+	FDD	1805	1842.5	1880	75	1710	1747.5	1785	95	Global	8
			1200	1575	1949		19200	19575	19949			
7	2600	FDD	2620	2655	2690	70	2500	2535	2570	120	EMEA	8
			2750	3100	3449		20750	21100	21449			
8	900 GSM	FDD	925	942.5	960	35	880	897.5	915	45	Global	8
			3450	3625	3799		21450	21625	21799			
20	800 DD	FDD	791	806	821	30	832	847	862	-41	EMEA	9
			6150	6300	6449		24150	24300	24449			
22	3500	FDD	3510	3550	3590	80	3410	3450	3490	100	EMEA	10.4
			6600	7000	7399		24600	25000	25399			
28	700 APT	FDD	758	780.5	803	45	703	725.5	748	55	APAC,EU	11.1
			9210	9435	9659		27210	27435	27659			
31	450	FDD	462.5	465	467.5	5	452.5	455	457.5	10	Global	12.0
			9870	9895	9919		27760	27785	27809			
32	1500 L-band	SDL	1452	1474	1496	44	Downlink only				EMEA	12.4
			9920	10140	10359							
33	TD 1900	TDD	1900	1910	1920	20					EMEA	8
			36000	36100	36199							
34	TD 2000	TDD	2010	2017.5	2025	15					EMEA	8
			36200	36275	36349							
38	TD 2600	TDD	2570	2595	2620	50					EMEA	8
			37750	38000	38249							
41	TD 2600+	TDD	2496	2593	2690	194					Global	10
			39650	40620	41589							
42	TD 3500	TDD	3400	3500	3600	200						10
			41590	42590	43589							
43	TD 3700	TDD	3600	3700	3800	200						10
			43590	44590	45589							
46	TD Unlicensed	TDD	5150	5537.5	5925	775					Global	13.2
			46790	50665	54539							
47	TD V2X	TDD	5855	5890	5925	70					Global	14.1
			54540	54890	55239							
48	TD 3600	TDD	3550	3625	3700	150					Global	14.2
			55240	55990	56739							
49	TD 3600r	TDD	3550	3625	3700	150					Global	15.1
			56740	57490	58239							
50	TD 1500+	TDD	1432	1474.5	1517	85						15.0
			58240	58665	59089							
51	TD 1500-	TDD	1427	1429.5	1432	5						15.0
			59090	59115	59139							
52	TD 3300	TDD	3300	3350	3400	100						15.2
			59140	59640	60139							
53	TD 2500	TDD	2483.5	2489.5	2495	11.5						16.0
			60140	60197	60254							
65	2100+	FDD	2110	2155	2200	90	1920	1965	2010	190	Global	13.2
			65536	65986	66435		131072	131522	131971			
67	700 EU	SDL	738	748	758	20	Downlink only				EMEA	13.2
			67336	67436	67535							
68	700 ME	FDD	753	768	783	30	698	713	728	55	EMEA	13.3
			67536	67686	67835		132672	132822	132971			
72	450 PMR/PAMR	FDD	461	463.5	466	5	451	453.5	456	10	EMEA	15.0
			68936	68961	68985		133472	133497	133521			
87	410	FDD	420	422.5	425	5	410	412.5	415	10	EMEA	16.2
			70546	70571	70595		134182	134207	134231			
88	410+	FDD	422	424.5	427	5	412	414.5	417	10	EMEA	16.2
			70596	70621	70645		134232	134257	134281			
103	NB-IoT	FDD	757	757.5	758	1	787	787.5	788	-30		17.5
			70646	70651	70655		134282	134287	134291			

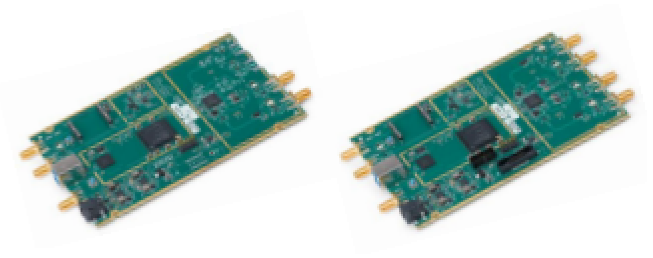
Annex 5 – USRP b210 specifications

In the next figures, it is presented the different features, the product overview, specifications, and a diagram with the different components of the USRP b210 used in this project.



FEATURES

- RF coverage from 70 MHz – 6 GHz
- GNU Radio, C++ and Python APIs
- USB 3.0 SuperSpeed interface
- Standard-B USB 3.0 connector
- Flexible rate 12 bit ADC/DAC
- Grounded mounting holes



USRP B200

- 1 TX & 1 RX, Half or Full Duplex
- Xilinx Spartan 6 XC6SLX75 FPGA
- Up to 56 MHz of instantaneous bandwidth
- USB Bus powered

USRP B210

- 2 TX & 2 RX, Half or Full Duplex
- Fully-coherent 2x2 MIMO capability
- Xilinx Spartan 6 XC6SLX150 FPGA
- Up to 56 MHz of instantaneous bandwidth in 1x1
- Up to 30.72 MHz of instantaneous bandwidth in 2x2
- Includes DC power supply
- GPIO capability

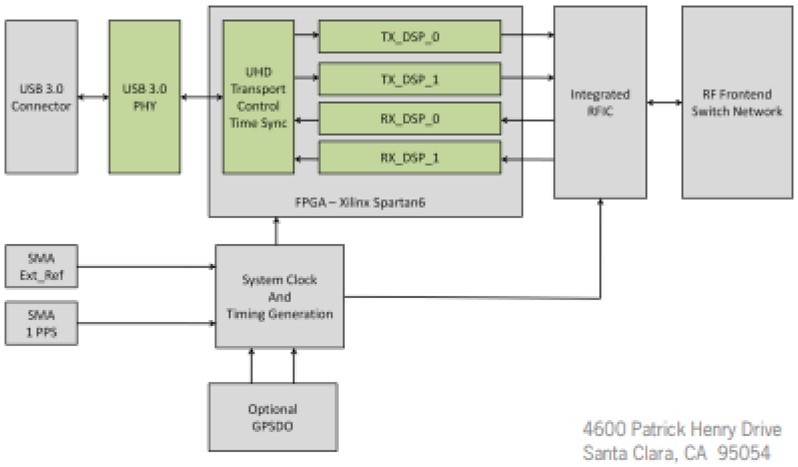
USRP B200/B210 Product Overview

The USRP B200 and B210 hardware covers RF frequencies from 70MHz to 6 GHz, has a Spartan6 FPGA, and USB 3.0 connectivity. This platform enables experimentation with a wide range of signals including FM and TV broadcast, cellular, Wi-Fi, and more. The USRP B200 features one receive and one transmit channel in a bus-powered design. The USRP B210 extends the capabilities of the B200 by offering a total of two receive and two transmit channels, incorporates a larger FPGA, GPIO, and includes an external power supply. Both use an Analog Devices RFIC to deliver a cost-effective RF experimentation platform, and can stream up to 56 MHz of instantaneous bandwidth over a high-bandwidth USB 3.0 bus on select USB 3.0 chipsets (with backward compatibly to USB 2.0). Because the B200 and B210 are enabled with our USRP Hardware Driver™ (UHD), users can develop their applications and seamlessly port their designs to high-performance or embedded USRPs such as the USRP X310 or USRP E310. UHD is an open-source, cross-platform driver that can run on Windows, Linux, and MacOS. It provides a common API, which is used by several software frameworks, such as GNU Radio. With this software support, users can collaborate with a vibrant community of enthusiasts, students, and professionals that have adopted USRP products for their development. As a member of this community, users can find assistance for application development, share knowledge to further SDR technology, and contribute their own innovations.

Spec	Typ.	Unit
Power		
DC Input	6	V
Conversion Performance and Clocks		
ADC Sample Rate (max)	61.44	MS/s
ADC Resolution	12	bits
ADC Wideband SFDR	78	dBc
DAC Sample Rate (max)	61.44	MS/s
DAC Resolution	12	bits
Host Sample Rate (16b) **	61.44	MS/s
Frequency Accuracy	±2.0	ppm
W/ GPS Unlocked TCXO Reference	±75	ppb
W/ GPS Locked TCXO Reference	< 1	ppb

Spec	Typ.	Unit
RF Performance (single channel)		
SSB/LO Suppression	-35/50	dBc
3.5 GHz	1.0	deg RMS
6 GHz	1.5	deg RMS
Power Output	>10	dBm
IIP3 (@ typ NF)	-20	dBm
Receive Noise Figure	<8	dB
Physical		
Dimensions	9.7x15.5x1.5	cm
Weight	350	g

*All specifications are subject to change without notice.
** See benchmark results for sample rates in various configurations.



4600 Patrick Henry Drive
Santa Clara, CA 95054

Glossary

CN	Core Network
eNB	evolved NodeBs
EPC	Evolved Packet Core
E-UTRAN	Evolved Terrestrial Radio Access Network
FDD	Frequency División Duplexing
gNB	gNodeB
GSM	Global System for Mobile Communications
GUMMEI	Globally Unique MME Identity
HSS	Home Subscriber Server
IMSI	International Mobile Subscriber Identity
LTE	Long Term Evolution
MCC	Mobile Country Code
MIMO	Multiple Input Multiple Output
MME	Mobility Management Entity
MNC	Mobile Network Code
OAI	OpenAirInterface
OPc	Operator key or code
OSA	OpenAirInterface Software Alliance
P-GW	PDN Gateway
PLMN Id	Public Land Mobile Network Identifier
QoS	Quality of Service
RAN	Radio Access Network
RPC	Remote Procedure Calls
S-GW	Serving Gateway
TAI List	Tracking Area Identity List
TCP	Transmission Control Protocol
TDD	Time División Duplexing
UE	User Equipment
UMTS	Universal Mobile Telecommunications System
USRP	Universal Software Radio Peripheral