Rui Marcelo
Rodrigues Pereira

Configuração, Implementação e Teste de Redes 5G
Configuration, Deployment and Test of 5G Networks

Rui Marcelo
Rodrigues Pereira

Configuração, Implementação e Teste de Redes 5G
Configuration, Deployment and Test of 5G Networks

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requesitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Doutor Arnaldo Oliveira, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / president

Professor Doutor Adão Paulo Soares da Silva
Professor Associado, Universidade de Aveiro

vogais / examiners committee

Doutor Luís Manuel de Sousa Pessoa
Investigador Sénior, Instituto de Engenharia e Sistemas de Computadores, Tecnologia e Ciência (arguente)

Professor Doutor Arnaldo Silva Rodrigues de Oliveira
Professor Auxiliar, Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Em primeiro lugar gostava de agradecer a toda a minha família, em especial ao meu pai pelo apoio e força que me tem dado ao longo dos anos, sem ele não teria conseguido concluir esta etapa da vida.
Queria também agradecer a todos os colegas que ao longo dos anos me têm acompanhado e proporcionado momentos de boa disposição, em especial ao Daniel que acompanhou ao longo destes meses na aventura que foi esta dissertação.
Queria agradecer ao professor Arnaldo pela paciência e o apoio prestado ao longo do trabalho desta dissertação. Um agradecimento também ao Samuel Madail, pela confiança depositada em mim ao longo dos últimos meses.

Palavras Chave Redes móveis, 5G, StandAlone, OpenAirInterface, USRP

Resumo

As redes móveis têm assumido um papel cada vez mais central ao longo das últimas décadas, com uma importância crescente a cada nova geração. Atualmente encontra-se a ser implementada a quinta geração de redes móveis. Esta nova geração tem o objetivo de, não só, oferecer melhores serviços ao utilizador, mas também a assumir um papel central em áreas da sociedade em que anteriormente a sua presença era menos notória. Assim sendo, prevê-se que o seu impacto mudará de forma profunda a vida das pessoas, mudando a maneira como realizamos algumas das atividades do nosso quotidiano. Também na indústria, irá provocar grandes alterações, permitindo tornar o conceito de Indústria 4.0 uma realidade. Estes são alguns exemplos de alterações que se esperam nas próximas décadas. No âmbito desta dissertação será feito um estudo sobre redes móveis 5G, seguindo uma arquitetura StandAlone (SA). Para tal serão identificados os diferentes elementos que constituem a rede, analisando as funções realizadas por cada um deles, e a forma como eles se interligam. Foi também realizada uma implementação de uma rede 5G com uma arquitetura SA. Para tal recorreu-se ao software disponibilizado pela OpenAirInterface (OAI). Para a implementação da rede utilizou-se 2 PC's, sendo que um ficou responsável por executar o Core, e o outro por executar o gNodeB (gNB). O PC que executa o gNB tem ligado a si, via USB, um USRP b210 que é responsável pelas funções de RF, permitindo fazer implementações com uma Largura de Banda de 40MHz. Para terminal de teste foram utilizados 2 User's Equipment (UE), o primeiro foi o OAI UE, que é um UE emulado, cujo software é disponibilizado pela OAI, para utilizar este UE foi necessário um terceiro PC, e permite a realização de testes simples à rede. O segundo terminal de teste, utilizou-se o Modem da Quectel RM500Q, que ao se ligar a um PC permite realizar testes à rede. Utilizando o Modem, foram realizados testes de desempenho à rede, para uma largura de banda de 40MHz, obtendo-se velocidades de Downlink próximas de 100Mbps e velocidades de Uplink próximas de 8Mbps. Na segunda parte desta dissertação foi desenvolvida uma plataforma para trabalhar em conjunto com a OAI, com o objetivo de melhorar a sua usabilidade. A utilização da plataforma permite que, quem queira utilizar a OAI possa realizar tarefas com várias etapas, como a instalação do software da OAI, o possa fazer indicando apenas as informações sobre o setup que deseja instalar, deixando o processo de instalação à responsabilidade da plataforma, reduzindo o tempo de instalação e o aparecimento de problemas durante esse processo. A plataforma permite a configuração e execução da rede de uma maneira mais amigável, ao contrário da maneira tradicional onde o utilizador era obrigado a editar diretamente os ficheiros de configuração, tornando estas tarefas mais rápidas e reduzindo o aparecimento de erros associados à configuração da rede. Por fim, a plataforma oferece um conjunto de testes à rede, que permite ao utilizador saber o estado da rede, e detetar problemas que possam existir.

Keywords Mobile networks, 5G, StandAlone, OpenAirInterface, USRP

Abstract

Mobile networks have assumed an increasingly central role over the last few decades, with increasing importance with each new generation. Currently, the fifth generation of mobile networks is being implemented. This new generation aims not only to offer better services to the user, but also to assume a central role in areas of society where their presence was previously less noticeable. Therefore, it is predicted that its impact will profoundly change people's lives, changing the way we carry out some of our daily activities. Also in the industry, it will cause major changes, allowing the concept of Industry 4.0 to become a reality. These are some examples of changes that are expected in the coming decades. In the scope of this dissertation, a study will be carried out on 5G mobile networks, following a StandAlone (SA) architecture. To this end, the different elements that make up the network will be identified, analyzing the functions performed by each of them, and how they are interconnected. An implementation of a 5G network with an SA architecture was also carried out. For this, the software provided by OpenAirInterface (OAI) was used. For the implementation of the network, 2 PCs were used, one of which was responsible for running the Core, and the other for running the gNodeB (gNB). The PC running gNB has a USRP b210 connected to it via USB, which is responsible for the RF functions, allowing implementations with a Bandwidth of 40MHz. For the test terminal, 2 User's Equipment (UE) were used, the first was the OAI UE, which is an emulated UE, whose software is provided by the OAI, to use this UE a third PC was needed, and allows for simple tests the net. The second test terminal used the Quectel RM500Q Modem, which, when connected to a PC, allows testing the network. Using the Modem, performance tests were carried out on the network, for a bandwidth of 40MHz, obtaining Downlink speeds close to 100Mbps and Uplink speeds close to 8Mbps. In the second part of this dissertation, a platform was developed to work together with the OAI, to improve its usability. Using the platform allows anyone who wants to use the OAI to perform tasks with several steps, such as installing the OAI software, can do so by simply indicating the information about the setup they want to install, leaving the installation process to the platform's responsibility. , reducing installation time and the appearance of problems during this process. The platform allows the configuration and execution of the network in a more friendly way, unlike the traditional way where the user was forced to directly edit the configuration files, making these tasks faster and reducing the appearance of errors associated with the configuration of the network. Finally, the platform offers a set of tests to the network, which allows the user to know the status of the network, and to detect problems that may exist.

Contents

List of Figures

List of Tables

List of Acronyms

3GPP	Third Generation Partnership Program
5GC	5G Core
AAA	Authentication, Authorization, and Accounting
AF	Application Function
AKA	Authentication and Key Agreement
AM	Acknowledged Mode
AMF	Access and Mobility Management Function
ARQ	Automatic Repeat Request
AS	Access Stratum
AUSF	Authentication Server Function
BCCH	Broadcast Control Channel
BCH	Broadcast Channel
CCCH	Common Control Channel
CESC	Cloud Enabled Small Cell
COTS	Commercial Off-The-Shelf
CP	Cyclic Prefix
CPRI	Common Public Radio Interface
CU	Centralised Unit
DCCH	Dedicated Control Channel
DCI	Downlink Control Information
DL-SCH	Downlink Shared Channel
DN	Distributed Network
DNN	Data Network Name
DRB	Data Radio Bearer
DTCH	Dedicated Traffic Channel
DU	Distributed Unit
eCPRI	Evolved Common Public Radio Interface
eNB	evolved Node B
EPC	Evolved Packet Core
FFT	Fast Fourier transform
gNB	Next Generation Node B
HARQ	Hybrid Automatic Repeat Request
iFFT	Inverse FFT
IMS	IP Multimedia Subsystem

IMSI International Mobile Subscriber Identity
MAC Medium Access Control
MCC Mobile Country Code
MNC Mobile Network Code
MIMO Multiple-Input and Multiple-Output
NAS Non-Access Stratum
NEF Network Exposure Function
nFAPI network Functional Application Platform Interface
NF Network Function
NFV Network Functions Virtualization
NIC Network Interface Card
NRF Network Repository Function
NSA Non-Standalone
NSSAAF Network Slice Specific Authentication and Authorization Function
NSSAI Network Slice Selection Assistance Information
NSSF Network Slice Selection Function
OAI OpenAirInterface
ONF Open Networking Foundation
OMEC Open Mobile Evolved Core
O-RAN Open Radio Access Network
PBCH Physical Broadcast Channel
PCH Paging Channel
PCCH Paging Control Channel
PCF Policy Control Function
PDSCH Physical Downlink Shared Channel
PDCCH Physical Downlink Control Channel
PDCP Packet Data Convergence Protocol
PDU Protocol Data Unit
PLMN Public Land Mobile Network
PON Passive Optical Network
PRACH Physical Random Access Channel
PSBCH Physical Sidelink Broadcast Channel
PSCCH Physical Sidelink Control Channel
PSFCH Physical Sidelink Feedback Channel
PSSCH Physical Sidelink Shared Channel
PUCCH Physical Uplink Control Channel
PUSCH Physical Uplink Shared Channel
QFI QoS flow ID
QoS Quality of Service
RACH Random Access Channel
RAN Radio Access Network
RF Radio Frequency
RLC Radio Link Control

ROC	Runtime Operation Control
RRC	Radio Resource Control
RU	Radio Unit
SA	Standalone
SBCCH	Sidelink Broadcast Control Channel
SC	Small Cell
SCCH	Sidelink Control Channel
SCF	Small Cell Forum
SCI	Sidelink Channel Information
SCP	Service Communication Proxy
SDAP	Service Data Adaptation Protocol
SDN	Software-Defined Networking
SDU	Service Data Unit
SFCI	Sidelink Feedback Control Information
SL-BCH	Sidelink Broadcast Channel
SL-SCH	Sidelink Shared Channel
SMF	Session Management Function
SRB	Signaling Radio Bearer
SST	Slice Service Type
STCH	Sidelink Traffic Channel
TB	Transport Blocks
TM	Transparent Mode
UCI	Uplink Control Information
UDM	Unified Data Management
UDR	Unified Data Repository
UE	User Equipment
UL-SCH	Uplink Shared Channel
UM	Unacknowledged Mode
UPF	User Plane Functions
USRP	Universal Software Radio Peripheral
VNF	Virtualized Network Function
vRAN	Virtualized Radio Access Network

Chapter 1

Introduction

1.1 Framework

The evolution of mobile networks over the last few decades has had a huge impact on people's lives, assuming a fundamental role in our society. This evolution has completely changed the way people live, communicate with each other, the way they work, and even led to the emergence of new habits that previously did not exist.

From the first generation of mobile networks to the present, successive generations have been improving existing services, but also responding to new needs, such as the growing number of users, the technological advancement of the devices itself, which allows them to have new functionalities and services associated with it. However, these new benefits result in higher volume of information exchanged, demanding connections with better quality and more capacity. Figure 1.1 shows the various generations of mobile networks and the services for each new generation.

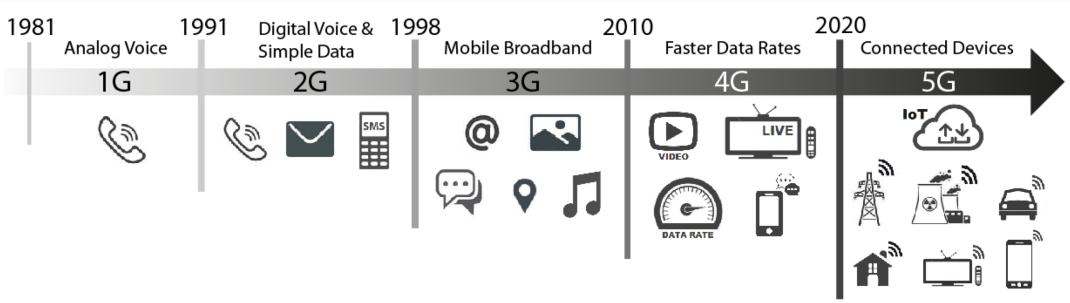

Figure 1.1: Evolution of the mobile network (adapted from [1]).

This new generation of mobile networks (5G) will allow the massive connection of various types of devices, allowing data transmission at higher speeds than the previous generation and with even lower latency times. 5G will allow devices to connect and communicate with each other in contexts that in previous generations was not possible or safe in some cases (critical communications), but now with a very efficient connection both in terms of latency time and amount of data they that can transmit, making it possible for multiple devices to work together to perform tasks autonomously or with minimal human presence /intervention. Therefore, this new generation of mobile networks will have a greater and deeper impact than previous generations, as it will not only allow a better experience for common users, but

will also be fundamental in other areas of society, such as industry, allowing a higher level of automation that will be reflected in the increase of efficiency in the productive processes, and greater safety in the workplaces [2]. In the automotive sector, 5G will allow vehicles to exchange data with each other in real time, which will contribute to an increase in driving safety and efficiency, and also enable autonomous driving to become a reality [3]. These are just some of the many areas where 5G will play a key role. All the changes that will be possible thanks to 5G will establish a new paradigm in the way we work and live in society.

The 5G implementation can have 2 possible architectures, a Non-Standalone (NSA) or a Standalone (SA) [4], as shown in figure 1.2.

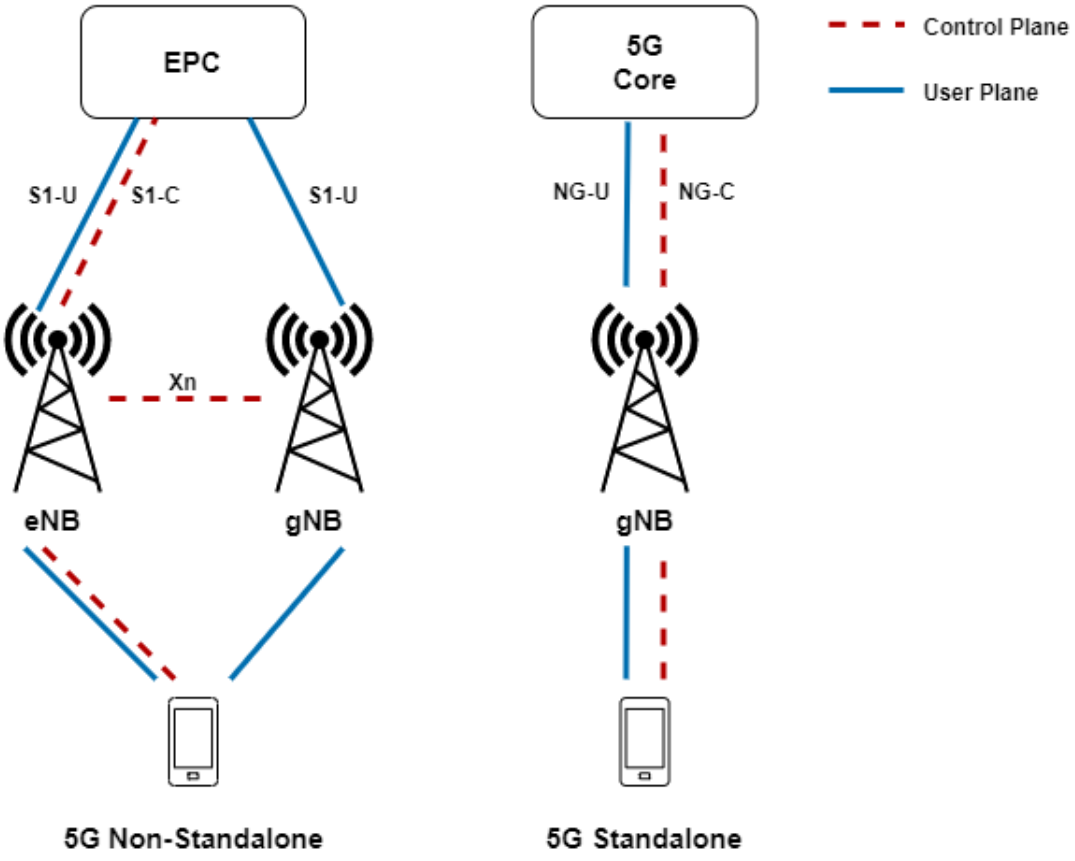

Figure 1.2: Example of NSA and SA architectures.

The NSA architecture is supported by the existing infrastructure of the previous generation, where the gNodeB (gNB) works together with the eNodeB (eNB) from the 4G, where the gNB is responsible for the user plane data and the eNB is responsible for the control plane [5]. In addition, the NSA architecture still uses Evolved Packet Core (EPC), the same core as the previous generation. Despite the NSA allows operators to provide a 5G network more quickly to their customers, this architecture is very limited, not allowing all the benefits that the 5G network can offer, such as network slicing [4]. The SA architecture does not depend on the infrastructure of the previous generation, relying only on the presence of gNB on the Radio Access Network (RAN) and in a new 5G Core [4].The SA makes it possible to use the

maximum potential that the 5G network has to offer, enabling network slicing, thus being able to provide a service with network resources adapted to the characteristics and needs of the connected device, being ideal for devices that perform critical functions and are very sensitive to some criteria such as latency [4].

1.2 Motivation

For the massive connection of different types of devices (smartphones, cars, appliances, etc.), the network must be prepared, being necessary an increase in the densification of the 5G RAN.

In a 5G RAN, there are several types of radio access nodes such as macro cells and small cells, as shown in figure 1.3. Macro cells are large base stations that provide network coverage for several kilometers. Small cells are small base stations capable of providing coverage in smaller and more specific areas [6].

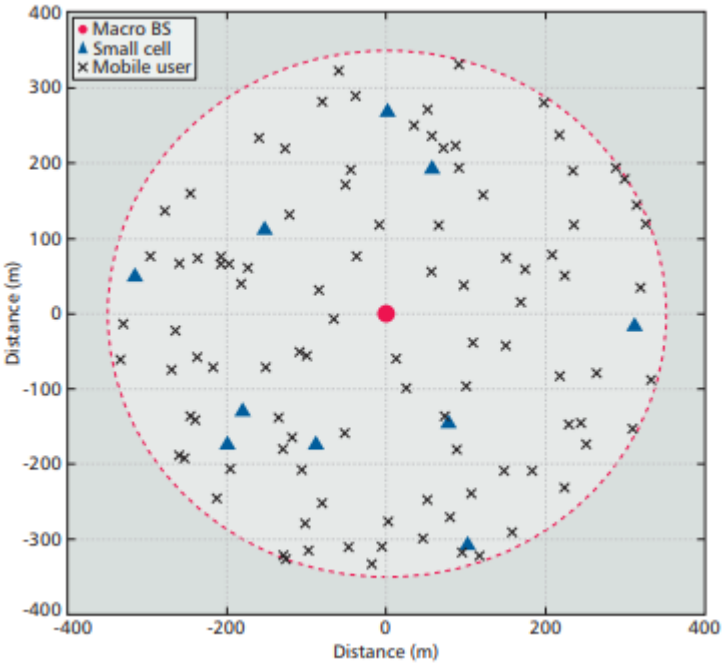

Figure 1.3: Example of RAN, with Base Stations and Small Cells (reproduced from [7]).

Small cells are very important to increase network densification, as they are the easiest and most cost-effective [8] solution for operators, allowing increased flexibility with simultaneous reduction of network complexity, making it possible to rapidly increase coverage. This solution also offers flexibility in the application in different types of scenarios where the signal from macro cells has more penetration difficulties, such as shopping centers, hospitals and others [7] [9]. The use of small cells in these scenarios improves not only the coverage problems but also helps macro cells with the huge traffic demand and allows an increase in the efficiency of spectrum use since the number of users connected to each small cell is smaller than the number that is connected to a macro cell. Thus, the radio resources are shared by a smaller

number of devices, allowing a greater debt for each equipment [7] [10].Another important aspect is spectrum reuse by different small cells in distinctive smaller areas [11].

The gNB, both in an SA architecture and in an NSA architecture, consists of 3 types of components, which are Radio Unit (RU), Distributed Unit (DU) and Centralised Unit (CU). Each small cell can function as a RU, being connected to a DU, which is connected to a CU [12], as shown in figure 1.4. The network functions executed by the DU and CU can

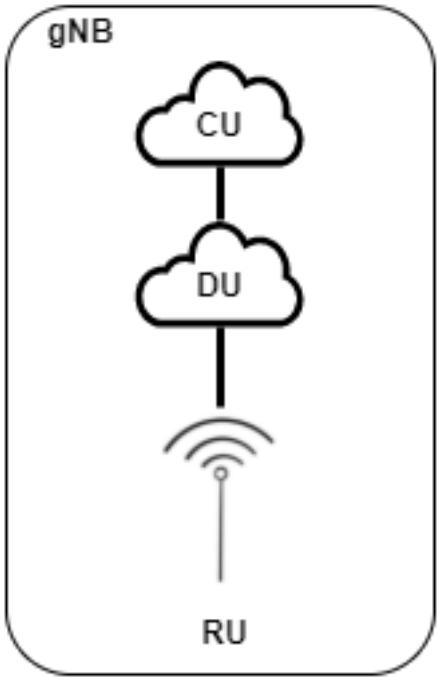

Figure 1.4: gNB elements, with CU, DU and RU.

be executed by software, not requiring dedicated hardware, being performed as Virtualized Network Function (VNF) on Commercial Off-The-Shelf (COTS), allowing an increase in the scalability and flexibility of the network; improvement in the use of network resources; increase in energy efficiency; and a reduction in the operational costs of the network [13].

Finally, the network architecture based on Open Radio Access Network (O-RAN) has gained the interest of operators, as there is the benefit that the interfaces used for communication between the different layers of the network are open, which makes it possible that different components of the network can be provided by different entities/suppliers that contribute to a more innovative development environment. In this way, the O-RAN architecture allows a greater diversity of solutions on the market, since the operators do not depend on a particular supplier, unlike what happens when using proprietary technology where the solutions/products of one supplier do not work with a product from a different supplier [14].

1.3 Objectives

The objective of this dissertation is to implement a 5G network with an SA architecture, using as a work base the open-source code provided by the OpenAirInterface (OAI) platform created by EURECOM, and open source components for RAN provided by Open Networking Foundation (ONF). For this, the following specific objectives are traced for this dissertation:

- Study of the different elements of a 5G network following an SA architecture.
- Implementation and testing, in the laboratory, of a 5G SA network, using band78, with a bandwidth of 40MHZ, using open source software provided by OpenAirInterface (OAI).
- Development of a platform to work together with OAI, to help the use of OAI software, through the automation of installation, configuration, execution, and network testing processes.
- Testing of the developed platform.

1.4 Document Structure

In addition to this introductory chapter, this document has 5 more chapters with the following contents:

- Chapter 2 - **"State of the Art":** provides an overview of a 5G network with a StandAlone architecture, providing the necessary background to carry out the dissertation.
- Chapter 3 - **"OpenAirInterface Deployment":** provides information about the implementation process of the 5G SA network based on OAI, identifying the characteristics of the hosts used, what was necessary to configure and install, and how to run the network.
- Chapter 4 - **"Platform Implementation":** describes the platform development process, identifies the various features available on the platform, and how each one works.
- Chapter 5 – **"Results":** describes the general functioning of the platform, showing the interfaces, and showing how the user can use the platform.
- Chapter 6 - **"Conclusions and Future Work":** an analysis is made of the work carried out, identifying the objectives that have been met, and indicating what can be improved in the future.

Chapter 2

Fundamental Concepts

This chapter provides an overview of the 5G network with a SA architecture. The different components of the network are analyzed, being described how they are interconnected and the functions performed by each of them. A discussion related to the virtualization of network functions is still covered. In the end, open source platforms are analyzed that can be used to implement a network.

2.1 Introduction

A mobile network aims to provide a wireless connection to User Equipments (UE's) that are, most of the time, moving.

A mobile network can be divided into two structures: Core and RAN, as shown in Figure 2.1. As mentioned in chapter 1, there are different possible architectures to implement a 5G network, being able to adopt an SA or NSA architecture and, depending on the architecture that is chosen, the components that constitute the Core and the RAN vary.

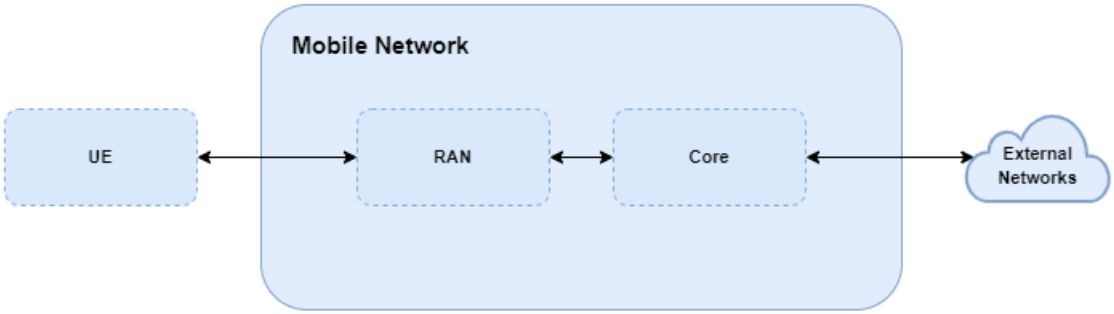

Figure 2.1: Structure of a mobile network.

As previously described in chapter 1, the application of an SA architecture enables the use of the maximum potential that the 5G network has to offer. Therefore, the studies developed in this dissertation will be focused in the SA architecture. A mobile network based on an SA architecture, as can be seen in Figure 2.2, is characterized by the use of the new 5G Core (5GC), and the RAN consists only of gNB's interconnected through the Xn interface. The gNB's communicate with the 5GC using the NG interface and the communication between

the UE and the gNB uses the Uu radio interface [5].

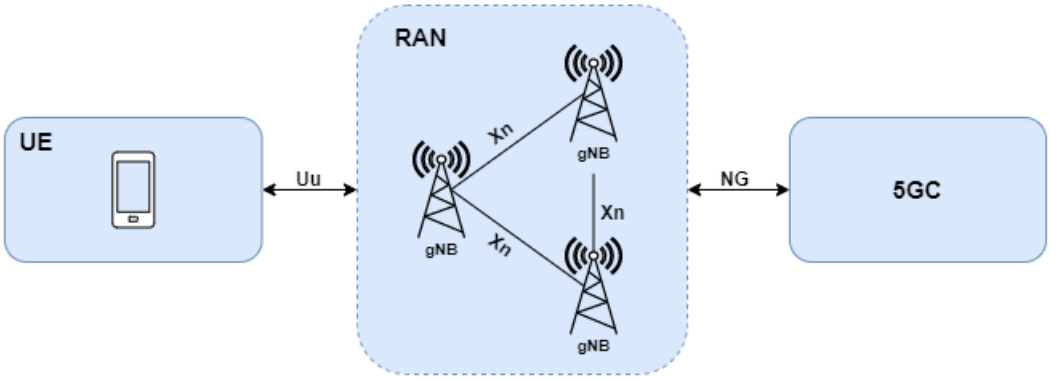

Figure 2.2: Representation of the SA architecture.

In the next sections, the main elements of an SA architecture will be presented.

2.2 5G Core

The core network is the structure that allows to establish a connection between the RAN of a given geographic area and external networks. The 5GC has several functions such as providing a connection to provide services to UE's; users authentication; ensuring that the connection meets the Quality of Service (QoS) requirements; tracking the mobility of the UE to ensure continuous service.

In this new generation, 5GC adopts a service-based architecture, allowing its network functions (NFs) to be deployed using Network Functions Virtualization (NFV) techniques and run them in the cloud. The 5GC consists of a set of elements that can be divided into two groups, those that perform user plane functions and those that perform control plane functions. Figure 2.3 shows the 5GC elements, identifying which ones are related to the control plane and user plane, as well as the interfaces used to communicate [15].

The 5GC elements identified in Figure 2.3 are described in the following.

For the control plane, the below elements are highlighted:

- **Access and Mobility Management Function (AMF):** The AMF has several responsibilities, including records management, connection management, mobility management, authentication, and access authorization.
- **Session Management Function (SMF):** The SMF is responsible for many functions, such as the management of each UE session and IP address allocation; selection and control of UPF functions; controlling part of QoS policy enforcement and downlink data notification.
- **Authentication Server Function (AUSF):** AUSF supports authentication for Third Generation Partnership Program (3GPP) access and untrusted non-3GPP access.

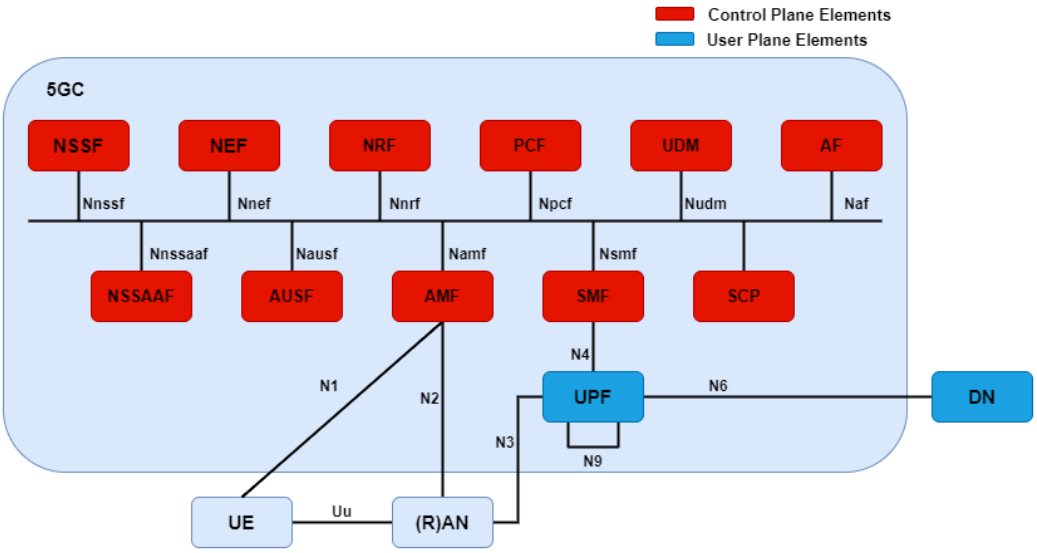

Figure 2.3: 5GC elements, with separation between control plane and user plane elements, and interfaces used.

- **Network Slice Selection Function (NSSF):** NSSF is responsible for selecting the Network Slice instances to serve the UE, determining the Network Slice Selection Assistance Information (NSSAI) and the AMF to be used to serve the UE based on a list of available AMF's provided by the Network Repository Function (NRF).
- **Network Repository Function (NRF):** NRF works as a repository for NF's, supporting discovery mechanisms and allowing to know which services are available.
- **Network Exposure Function (NEF):** NEF is responsible for exposing events and resources securely to third-party services, translating internal-external information. Whenever an external application demands internal information, it has to go through the NEF.
- **Policy Control Function (PCF):** PCF is responsible for providing the policy rules applied to control plane functions, supporting a unified policy framework to control network behavior, and accessing signature information relevant to policy decisions in the unified data repository.
- **Unified Data Management (UDM):** UDM is responsible for generating 3GPP Authentication and Key Agreement (AKA) credentials; user identity management; signature and management and; access authorization based on signature data.
- **Application Function (AF):** AF is responsible for providing support for application influence on traffic routing; accessing Network Exposure Function; interacting with the Policy framework for policy control; and IP Multimedia Subsystem (IMS) interactions with 5GC.
- **Service Communication Proxy (SCP):** SCP has the functionalities of message forwarding and routing to destination NF/NF service; message forwarding and routing

to a next hop SCP and; indirect communication.

- **Network Slice Specific Authentication and Authorization Function (NSSAAF):** NSSAAF is responsible for providing support Network Slice-Specific Authentication and Authorization with an Authentication, Authorization, and Accounting (AAA) Server (AAA-S).

For the user plane, the **User Plane Functions (UPF)** are highlighted, being responsible for packet routing and forwarding, serving as a point of interconnection to the data network (DN). The UPF can be considered as the anchor point of radio access technology.

2.3 5G RAN

The RAN is responsible for connecting the UE's through radio connections, managing the spectrum in order to be used efficiently and meeting the QoS of each UE. As mentioned before, a 5G RAN in an SA architecture is constituted only by gNB. 3GPP defined that gNB is divided into 3 logical nodes, which are CU, DU, and RU. Each of these units will be able to host the different functions of the protocol stack. 3GPP has defined 8 functional split options, as shown in figure 2.4.

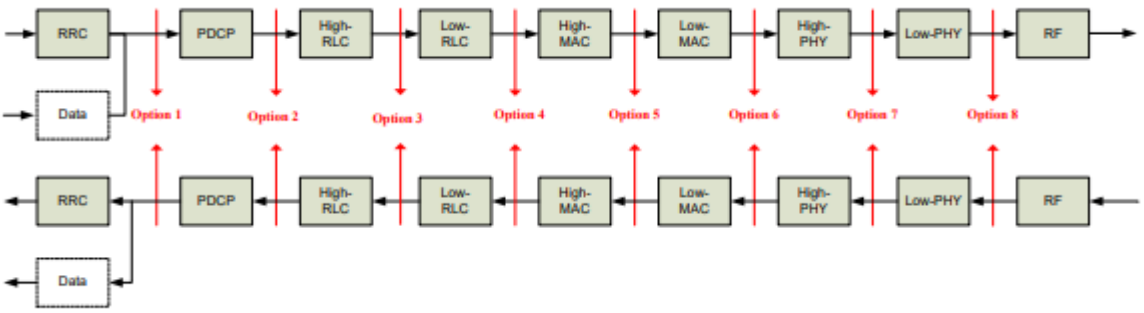

Figure 2.4: Possible network splits (reproduced from [5]).

In figure 2.5 it is possible to observe the connections of the different units of the gNB, and the interfaces used. Backhaul corresponds to the connection of the 5GC to one or more CU's, and uses the N2 (or NG-C) interface for control plane related traffic and the N3 (or NG-U) interface for user plane related traffic. Midhaul corresponds to the connection of the CU to one or more DU's, and uses the F1 interface for communication between the units. The fronthaul corresponds to the connection between the DU and one or more RU's, and uses the Evolved Common Public Radio Interface (eCPRI) for communication between the units. Each of these connections has its own requirements and characteristics. In Backhaul,

the bandwidth requirements are high and the latency requirements are less stringent than in Midhaul, presenting a latency time in the order of tens of milliseconds. In midhaul latency requirements are more stringent than in Backhaul and less stringent than in Fronthaul, presenting a latency time is around 1-2 ms. The fronthaul can be implemented in a Passive Optical Network (PON) and uses the eCPRI interface for communication between the units. The use of eCPRI has the advantage of being an open interface, more flexible and efficient than the Common Public Radio Interface (CPRI), allowing the reduction of latency time and more efficient use of bandwidth. In fronthaul, the communication latency has to be low, in the order of microseconds [16],[17],[18].

Each of these links, the interfaces used, and the latency and bandwidth requirements they have to meet, are critical to achieving 5G goals.

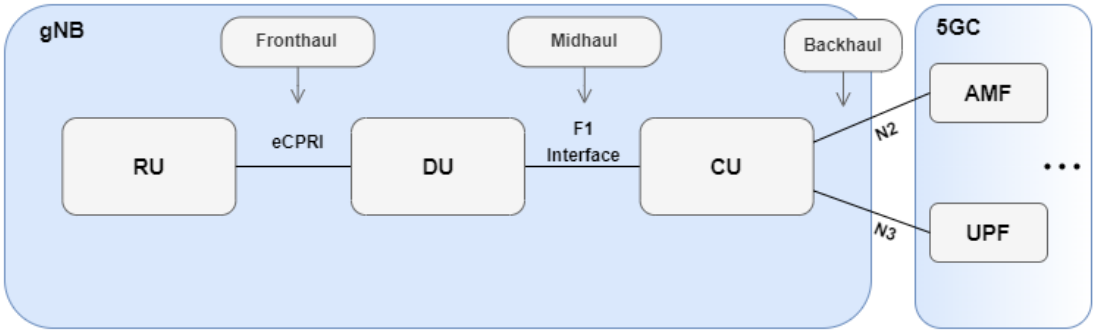

Figure 2.5: gNB Units, and the interfaces used in Fronthaul, Midhaul, and Backhaul.

The gNB elements identified in Figure 2.5 are described below.

2.3.1 Centralised Unit (CU)

The CU is responsible for the upper layers of the protocol stack. In figure 2.6, it is presented a general representation of the CU, where the protocols Packet Data Convergence Protocol (PDCP), Service Data Adaptation Protocol (SDAP) and Radio Resource Control (RRC)) that are executed by the unit are identified, as well the division between the control and user planes and the interface used to communication between planes. The CU can also be divided into CU-C and CU-U, in which one part is responsible for the control plane and the other for the user plane, respectively, and the communication between them is established through the E1 interface [5] [19].

The protocols mentioned above are described in the following.

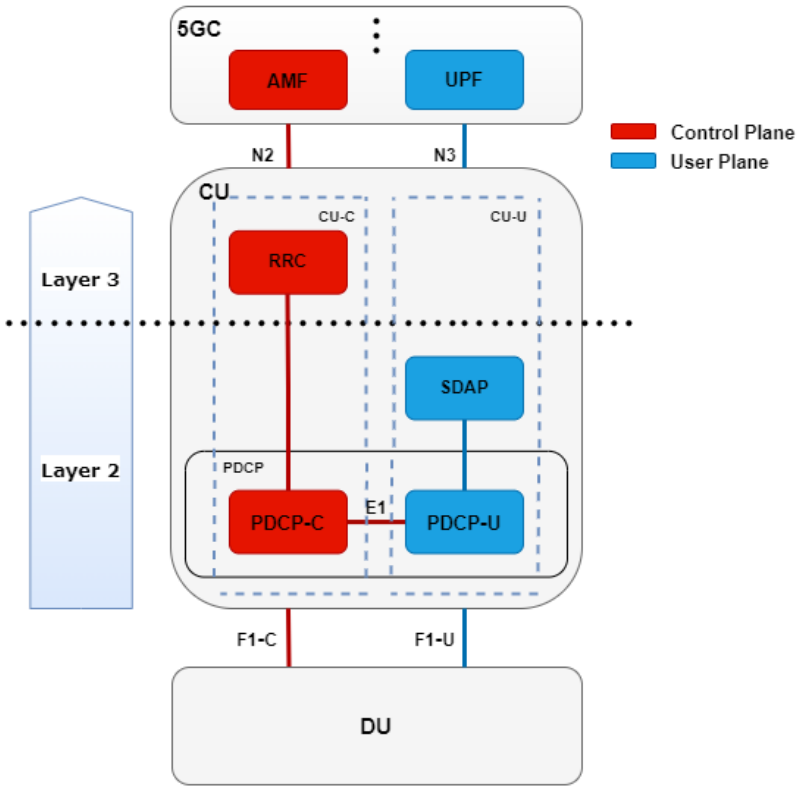

Figure 2.6: CU protocols, layers, and divisions.

- **Radio Resource Control (RRC):** RRC belongs to the control plane and has the functions of: transmitting system information related to Access Stratum (AS) and Non-Access Stratum (NAS); maintaining and releasing the RRC connection between UE and RAN; security; establishing Signaling Radio Bearer (SRB) and Data Radio Bearer (DRB); mobility; QoS management; detection and recovery radio link failure; NAS message transfer and; sidelink related functions such as, resource allocation, reporting, measurements and traffic information [20].

  The way the RRC works and the functions it performs depends on its state, the RRC states can be RRC_IDLE, RRC_INACTIVE, RRC_CONNECTED.

- **Service Data Adaptation Protocol (SDAP):** The SDAP is part of the user plane, and its function is to map between the QoS flow and the DRB channels, also marking the uplink and downlink packets with the QoS flow ID (QFI). For sidelink communication, the SDAP makes mapping between Qos flow and the sidelink data bearer [20].

- **Packet Data Convergence Protocol (PDCP):** PDCP is part of the user plane and the control plane. It communicates with the higher protocols (RRC and SDAP) through DRB channels, and with the lower protocol (RLC) through RLC channels. The PDCP can be divided into PDCP-C and PDCP-U. Some of the functions of the PDCP are: uplink and downlink data transfer; handle header compression and decompression;

ciphering and deciphering; delivery ordering services; integrity related services; Service Data Unit (SDU) discard. For sidelink communications, some of the functions performed by PDCP have restrictions, such as out-of-order delivery [20].

2.3.2 Distributed Unit (DU)

The DU is responsible for hosting layer 1 (PHY-High) and layer 2 protocols (Radio Link Control (RLC) and Medium Access Control (MAC)), represented in figure 2.7 and described in the following.

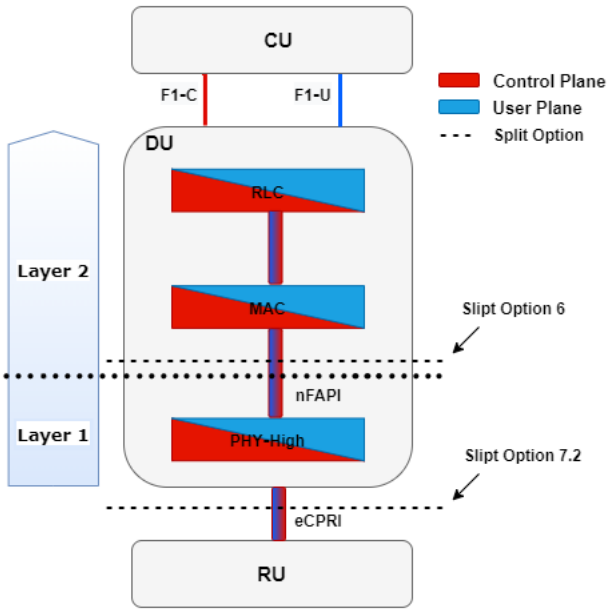

Figure 2.7: DU protocols, layers, and divisions.

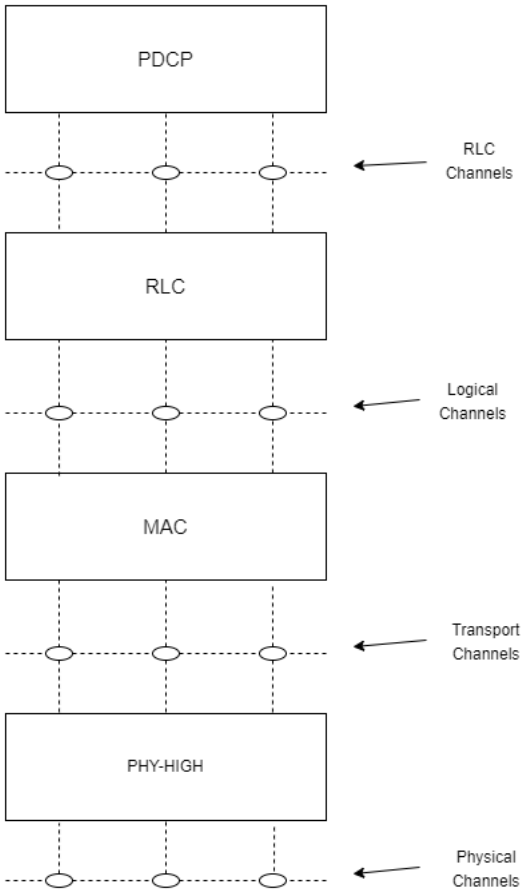

Figure 2.8: Protocols and communication channels.

- **Radio Link Control (RLC):** The RLC performs related functions both in the data plane and in the control plane and is located between PDCP and the MAC in the protocol stack. The RLC uses the RLC channels to communicate with the PDCP and uses the logical channels to communicate with the MAC, as shown in figure 2.8. RLC can support three transmission modes: Transparent Mode (TM), Unacknowledged Mode (UM) and Acknowledged Mode (AM). The functions performed by the RLC vary with the transmission mode being used. The RLC has the function of: transferring Protocol Data Units (PDU's) from the upper layers, numbering them sequentially independently of the PDCP (AM and UM); correcting errors through the Automatic Repeat Request (ARQ) (AM); segmenting the SDU's (AM and UM); re-segmenting the SDU's (AM); reassembly of SDU's (AM and UM); SDU discard (AM and UM) and; protocol error detection (AM) [20] [21].

- **Medium Access Control (MAC):** MAC communicates with RLC using logical channels and communicates with PHY using transport channels, as shown in figure 2.8. The MAC has the function of mapping between logical channels and transport channels (table 2.1, table 2.2 and table 2.3 shows the mapping between channels), multiplexing SDU's of one or more logical channels into Transport Blocks (TB) and delivering these

TB's to the transport channels, it also performs the opposite process. The MAC is also responsible for handling the priorities of each UE through dynamic scheduling, with priority between logical channels of a UE through channel prioritization. MAC also performs information reporting scheduling and error correction through Hybrid Automatic Repeat Request (HARQ). In sidelink communications, the MAC has functions for radio selection and filtering, dealing with priority issues. A logical channel can be defined by the type of information it transfers and can be classified as control and traffic channels. A control channel is only used for information related to the control plane. The control channels are Broadcast Control Channel (BCCH), Paging Control Channel (PCCH), Common Control Channel (CCCH), Dedicated Control Channel (DCCH) [20]. The traffic channel only carries information related to the user plane, which is the Dedicated Traffic Channel (DTCH) [20],[22].

Downlink	
Logical channels	Transport channels
BCCH	BCH
BCCH	DL-SCH
PCCH	PCH
CCCH	DL-SCH
DCCH	DL-SCH
DTCH	DL-SCH

Table 2.1: Mapping table between logical channels and transport channels for downlink[20].

Uplink	
Logical channels	Transport channels
CCCH	UL-SCH
DCCH	UL-SCH
DTCH	UL-SCH

Table 2.2: Mapping table between logical channels and transport channels for uplink [20].

Sidelink	
Logical channels	Transport channels
SCCH	SL-SCH
STCH	SL-SCH
SBCCH	SL-SCH

Table 2.3: Mapping table between logical channels and transport channels for sidelink [20].

- **PHY-High:** PHY-High is responsible for receiving the TB's of the transport channels from MAC, as shown in figure 2.8. Among the various functions performed by this layer, there are the transport channel encoding/decoding operations; the mapping between the transport channels and the physical channels(table 2.4 shows the mapping between channels). It also performs modulation/demodulation operations of the physical channels, being the QPSK, 16 QAM, 64 QAM and 256 QAM the types of modulation supported for both uplink and downlink. For the uplink, there is still one more modulation type that is $\pi/2$ BPSK. The behavior of this layer is controlled by the MAC scheduler, and its functions/operations depend on the type of transport channel. PHY-High also performs functions related to error detection and scrambling [23],[24].

  The mappings performed in this layer are shown in tables 2.4 and 2.5

Downlink	
Transport channels	Physical channels
DL-SCH	PDSCH
BCH	PBCH
PCH	PDSCH
Uplink	
Transport channels	Physical channels
UL-SCH	PUSCH
RACH	PRACH
Sidelink	
Transport channels	Physical channels
SL-SCH	PSSCH
SL-BCH	PSBCH

Table 2.4: Mapping table between transport channels and physical channels[25].

2.3.3 Radio Unit (RU)

The RU is the unit closest to the UE's, establishing the connection between the UE and the rest of the network. This unit is responsible for having the lowest layer of the protocol stack and for the radio functions, as shown in Figure 2.9. This unit is implemented using dedicated hardware. In PHY-Low, resource element mapping (RE Mapping) processes are performed, which consists of: converting the symbols received from the highest layer of the PHY into subcarriers; mapping the symbols into resource elements; performing beamforming processes and; execution of Inverse Fast Fourier transform (iFFT)/Fast Fourier transform

Downlink	
Control Information	Physical channels
DCI	PDCCH
Uplink	
Control Information	Physical channels
UCI UCI	PUCCH PUSCH
Sidelink	
Control Information	Physical channels
1st-stage SCI 2nd-stage SCI SFCI	PSCCH PSSCH PSFCH

Table 2.5: Mapping table between control channels information and physical channels [25].

(FFT) processes that serve to convert the symbols between the frequency domain and the time domain. In this layer is also added and removed the Cyclic Prefix (CP) that serves to distinguish the frames from each other [24]. In the Radio Frequency (RF) part, the conversion processes between digital and analog signals are performed, as well as the transmission and reception of signals, using frequency bands FR1 that operates from a frequency of 410MHz to 7.125GHz, or FR2 that operates from a frequency of 24.25GHz to 52.6GHz (mmWave).

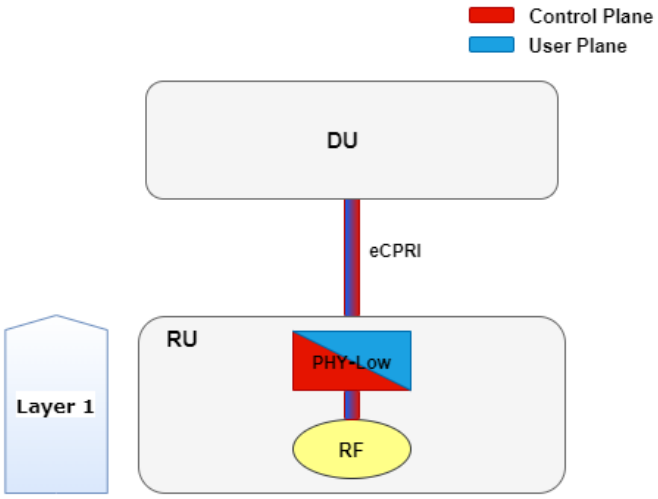

Figure 2.9: RU protocols, and layers.

2.4 Virtualization of RAN Elements

Currently, there is an increasing trend towards virtualization of RAN elements or vRAN. In the RAN, the NF's related to DU and CU can be executed using VNF. These VNF are

executed by COTS servers in Edge DC that can still be split into two layers, as shown in figure 2.10. The first layer, known as Light data center (Light DC), and like the physical unit (RU or Small Cell (SC)) is part of the Cloud Enabled Small Cell (CESC) and aims to provide low latency services by running VNF's that require low processing power. The second layer is the Main data center (Main DC) and aims to provide services that require high computational processing power for the implementation, for example, of Software-Defined Networking (SDN) controllers, security-related VNF, and mobility management [5].

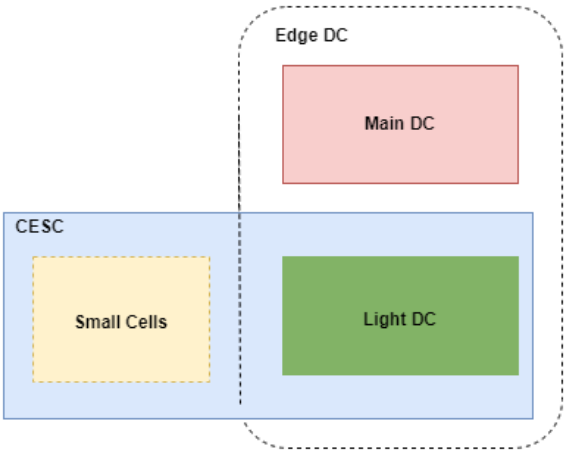

Figure 2.10: Network diagrams with Light DC and Main DC.

2.5 5G Deployment Platforms Overview

One of the objectives of this dissertation is to implement a 5G network. To carry out the implementation there are some open source platforms that provide software that can be used. Some of these platforms are described below.

2.5.1 OpenAirInterface (OAI)

OAI is an open source platform that allows you to emulate 4G/5G networks following the standards defined by 3GPP. OAI has two project groups, the OAI 5G CN, that aims to develop the software associated with the 5GC, and the OAI 5G RAN project, that aims to develop the software associated with the RAN. The source codes are available on EURECOM's Gitlab, and the OAI 5GC source code can be found in the *cn5g* repository [26] while the OAI RAN source code can be found in the *openairinterface5g* repository [27].

The code available in the *cn5g* repository does not yet have all the elements of a 5GC implemented, however, in the master branch of the repository, it is already possible to find a stable version of the core with the ability to support several procedures, having already been implemented with different gNB (including commercial gNB) and RAN simulators [28]. The OAI 5GC has two different implementation options, a simple one with the basic elements, and a more complete one with all the elements of the core developed by OAI until now [28].

The code available in *openairinterface5g* allows to do the emulation of eNB, gNB (for an NSA architecture and for an SA architecture), and also allows the emulation of a UE (SA and

NSA). However, to implement a 5G SA or NSA network, the develop branch of the repository must be used, as the master branch only offers the possibility to emulate the eNB. The code available, until the moment of this work, is not fully developed, resulting in lacking support for some functions for both the NSA and SA architectures. However, it is possible to carry out network tests for both architectures. In each of the repositories, it is possible to find a lot of information related to the code, recommended hardware, and installation tutorials [29],[30].

The highly flexible software provided by OAI, allows the modification of the source code to adjust to the needs of each implementation, allowing highly customized implementation solutions. The figure 2.11 illustrates the features offered by OAI.

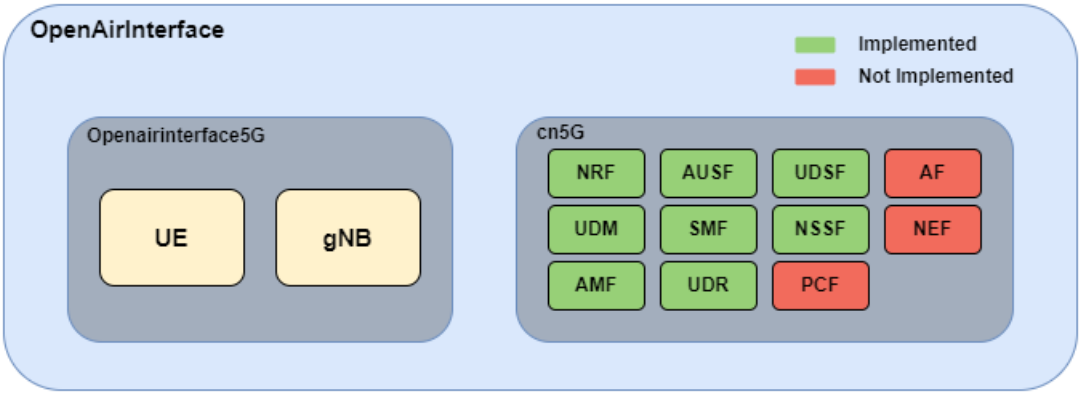

Figure 2.11: Representation of the OAI project, and their different projects.

2.5.2 Open Networking Foundation (ONF)

ONF is a non-profit consortium which has dozens of partners, members, and collaborators that aims to bring together different projects related to the development of open source solutions, taking advantage of network disaggregation, and SDN/NFV/Cloud technologies.

ONF has projects focused on mobile and 5G solutions, and broadband solutions. In mobile and 5G solutions it is important to highlight the *Aether* platform, which is an open source platform for multi-cloud deployments to provide wireless connectivity. The Aether platform is divided into two sub-platforms, which are SD-CORE and SD-RAN [31].

SD-CORE allows 4G and 5G NSA/SA deployments. This versatility and flexibility is due to the fact that the platform has integrated the Open Mobile Evolved Core (OMEC), for 4G and 5G NSA deployments, and also has integrated free5GC, for SA deployments. SD-CORE also provides a rich set of APIs for Runtime Operation Control (ROC) [32].

SD-RAN is responsible for the development, testing, and availability of open source components for the RAN, such as CU, DU, SDN controller, and even a RAN simulator [33],[34].

On the ONF website it is also possible to find information related to the components available and some tutorials related to the installation of the various components. ONF has a large community that includes operators, research institutes and companies, which allows the constant development of its platforms. The figure 2.12 illustrates the resources offered by ONF.

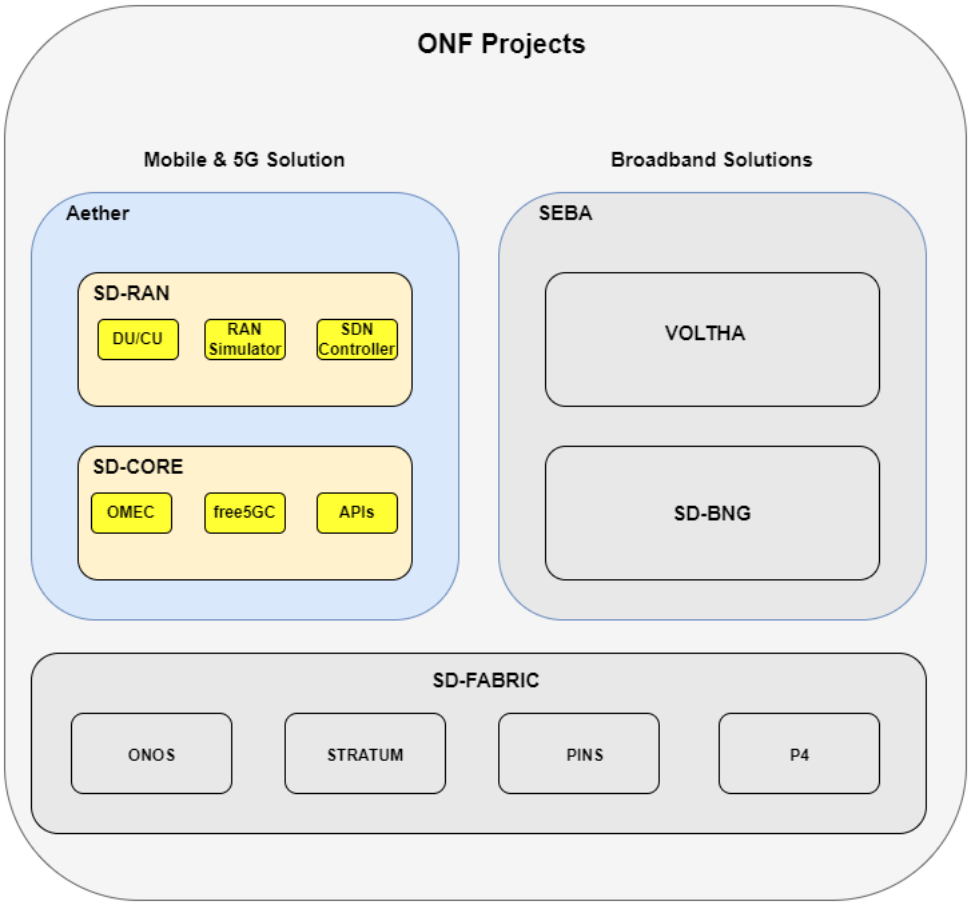

Figure 2.12: Representation of the ONF Platform, and their different projects.

2.6 Summary and Next Steps

In these first two chapters of the dissertation, several aspects related to 5G mobile networks were addressed. The first chapter provided an overview of how mobile networks have played a key role in society over the last few decades, and about their futute impact expected on society. The possible architectures that can be used were also discussed. In addition, the importance of small cells in RAN was analyzed, which elements of RAN can be virtualized and the advantages of doing the virtualization, besides the importance of architecture based on O-RAN. Finally, the objectives of this dissertation were defined.

With the information gathered in this chapter 2, the various elements that form part a 5G SA network were known, analyzing the functions performed by each element of the network. Finally, the different platforms that can be used to implement a network in the laboratory were analyzed.

The following chapter will be dedicated to the implementation of the network, giving a more detailed view of the software that will be used to implement it.

Chapter 3

Openairinterface Features and Deployment

This chapter presents the implementation of a 5G SA network based on OAI, where how it was carried out is described, analyzing the function of each host, its characteristics, and the changes that had to be made. It also identifies the UE's used to test the network and what configurations are required on each one.

3.1 Introduction

As explained in previous chapters, the implementation of a 5G SA network needs a set of fundamental elements. These elements are the Core and the gNB, and these two elements establish the connection between the UE's and the internet, as shown in Figure 3.1.

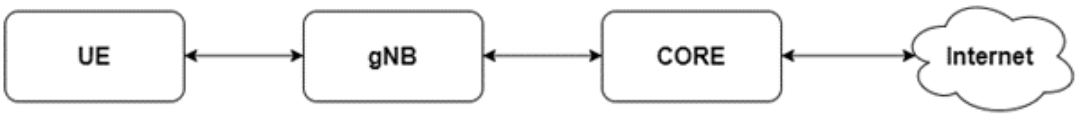

Figure 3.1: Essential elements of a 5G SA network.

Remembering that each of these elements can be further divided into smaller elements. For the core, it can be divided into a set of services, known as AMF, SMF, SCP, AUSF, NSSAAF, NSSF, NEF, NRF, PCF, UDM, UPF, and AF. In the case of gNB, it can be divided into 3 units, which are CU, DU, and RU.

The OAI platform, in the 5G RAN and 5G CORE NETWORK projects, provides all the software necessary to implement a 5G SA network through the virtualization of the different elements of the network. The great flexibility offered by the software developed by OAI allows a set of different possible implementations [26],[27]. The software for the Core, allows the realization of 3 possible deployments. The software provided for the RAN allows two types of

deployment, one using split 8, in which the gNB has a monolithic configuration, and another using split 2, which allows the division between the CU and the DU.

The possible implementations of greatest interest can be summarized as follows:

UE commercial $\leftrightarrow$ gNB(*monolithic*) $\leftrightarrow$ 5GC
OAI UE $\leftrightarrow$ gNB(*monolithic*) $\leftrightarrow$ 5GC
UE commercial $\leftrightarrow$ DU $\leftrightarrow$ CU $\leftrightarrow$ 5GC
OAI UE $\leftrightarrow$ DU $\leftrightarrow$ CU $\leftrightarrow$ 5GC

Within the scope of this dissertation, special focus will be given to implementations with monolithic gNB, varying the test terminal between OAI UE and COTS UE, according to needs. In the next sections of this chapter, the deployment architecture will be explained in detail, as how to use OAI software to implement a 5G SA network.

3.2 Implementation Overview

For the implementation of the network, two PC were needed, a PC that was responsible for hosting and running the network Core, and another PC where gNB was installed and executed. Finally, a Universal Software Radio Peripheral (USRP) b210 was used, connected via USB to the PC that runs the gNB, which was responsible for ensuring RF communication with the test terminal. For the terminals, two types were used. The first terminal was the OAI UE (emulated UE), which was installed and executed on a third machine, to use this terminal it was still necessary to use a second USRP b210. The second terminal consisted of using the Quectel RM500Q modem connected to a personal computer. As exemplified in figure 3.2.

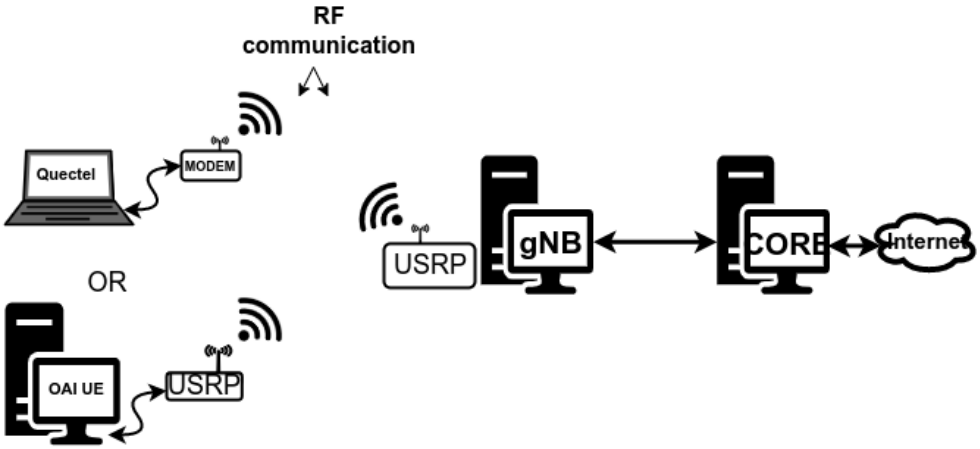

Figure 3.2: Overview of the setup implemented in the lab.

In figure 3.3 it is possible to see the photos of the different components used in the implementation in the laboratory.

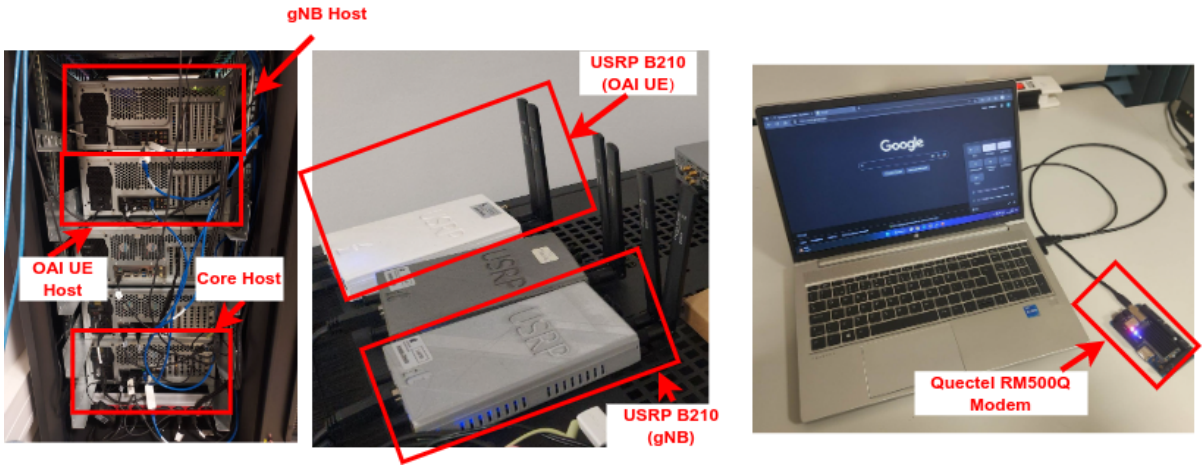

Figure 3.3: Photos of the setup components implemented in the laboratory.

The characteristics of each of the host, the way they communicate with each other, and what configurations each one of them needs will be explained in detail in the following points of this chapter.

3.2.1 5GC Host

The first PC to be analyzed is the PC that was responsible for hosting and running the Core, which must have a connection with the PC that runs the gNB, and must also have an internet connection, to guarantee a connection between the RAN and the Internet. To make this possible, the computer had two network cards, one of the cards was responsible for connecting to the gNB, and the other network card is connected to the outside, to allow the implementation to have a connection to the outside/internet, as shown in figure 3.4.

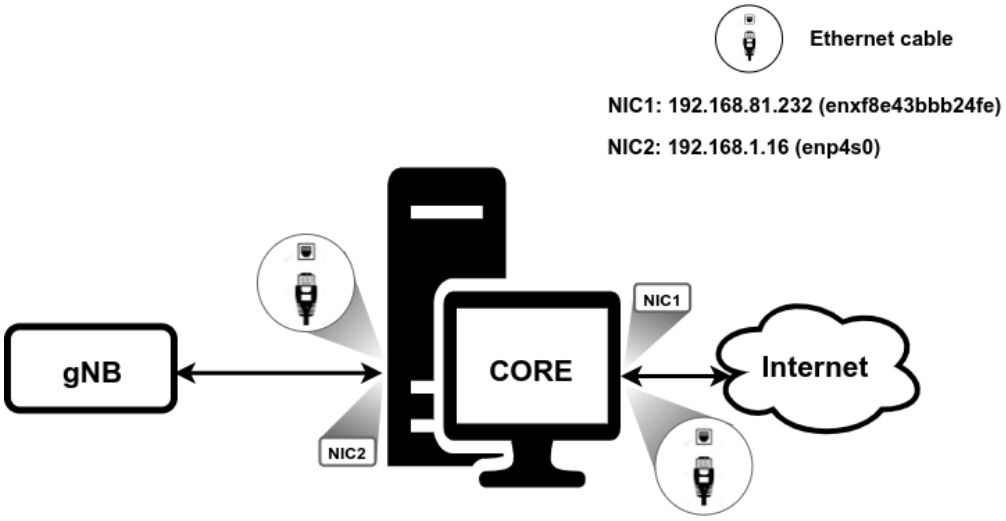

Figure 3.4: Core Host connections, and IP addresses used on NICs.

In the setup that was implemented in this dissertation, the network card responsible for communicating with the outside, identified in figure 3.4 by Network Interface Card (NIC) 1, with the name enxf8e43bbb24fe, was assigned the IP address 192.168.81.232. For the connection with the gNB, the network card, identified in figure 3.4 by NIC2, with the name enp4s0 was manually configured, with a static IP address of 192.168.1.16.

The PC where Core runs must also meet some requirements that are recommended by the OAI, such as operating system and hardware requirements. In the setup that was implemented, the host has the following specifications, listed in the following table 3.1:

Specifications	
CPU	Intel Core i9-12900K
RAM	64 GB
Disk	1 TB
Operational System	Ubuntu 18.04.6 LTS
Kernel	5.4.0-126-generic
NIC1	Intel Ethernet 1000 Mbps
NIC2	Intel Ethernet 1000 Mbps

Table 3.1: Core computer specifications.

As previously mentioned, the software provided by OAI for the core has 3 possible implementations, and in the scope of this dissertation, the Core version that was chosen was the basic version, which has the following elements: AMF, SMF, NRF, UPF(oai-spgwu-tiny), UDM, AUSF, and UDR.

Core works with docker containers, so to get Core working it is necessary to install Docker on the host that will run it.

Docker is a platform that allows the virtualized execution of applications/services in isolation within containers. In Docker, each Container runs an application/service, containing all the necessary dependencies for the correct operation of the application/service, each container is isolated at disk, memory, processing and network level. Can run on same host several containers. The configuration of several Containers can be done using a docker compose file, which allows configuring and initializing several Containers with a single command. Each Container runs from an image. Container images can be found in repositories such as Docker-Hub, where developers can place their projects, thus making their images available for other people to use.

Each of the elements listed above corresponds to a container, for that, it is necessary to pull the official images from the Docker-Hub.

In addition to the images of each of the elements, for the Core to work it is still necessary to pull the ubuntu:bionic and mysql/mysql5.7 images. An extra image is still available from the OAI for testing purposes.

To extract all the necessary images for the Docker-Hub Core, it was necessary to use the commands indicated below.

```
$ docker pull ubuntu:bionic
$ docker pull mysql:5.7
$ docker pull oaisoftwarealliance/oai-amf:develop
$ docker pull oaisoftwarealliance/oai-nrf:develop
```

```
$ docker pull oaisoftwarealliance/oai-smf:develop
$ docker pull oaisoftwarealliance/oai-udr:develop
$ docker pull oaisoftwarealliance/oai-udm:develop
$ docker pull oaisoftwarealliance/oai-ausf:develop
$ docker pull oaisoftwarealliance/oai-spgwu-tiny:develop
$ docker pull oaisoftwarealliance/trf-gen-cn5g:latest
```

After this process, it was necessary to tag the images, the names must match the names that were in the docker-compose. The commands below show how it was done.

```
$ docker image tag oaisoftwarealliance/oai-amf:develop oai-amf:develop
$ docker image tag oaisoftwarealliance/oai-nrf:develop oai-nrf:develop
$ docker image tag oaisoftwarealliance/oai-smf:develop oai-smf:develop
$ docker image tag oaisoftwarealliance/oai-udr:develop oai-udr:develop
$ docker image tag oaisoftwarealliance/oai-udm:develop oai-udm:develop
$ docker image tag oaisoftwarealliance/oai-ausf:develop oai-ausf:develop
$ docker image tag oaisoftwarealliance/oai-spgwu-tiny:develop oai-spgwu-tiny:
develop
$ docker image tag oaisoftwarealliance/trf-gen-cn5g:latest trf-gen-cn5g:latest
```

It was also necessary to clone the cn5g repository from openairinterface, where the rest of the software and core configuration files are available, such as docker-compose configuration files, and the database, which are necessary for the Core to work. The following commands show how it was done.

```
$ git clone https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed.git ~/oai-cn5g-fed
```

The bridge for communication between the different containers is called “demo-oai” created with the address 192.168.70.128/26, and all the core containers communicate through it, as shown in figure 3.5.

Each container has its own associated IP address, and each of these IP addresses is defined in the Docker compose file. Table 3.2 shows the IP addresses used by containers in the Core implementation.

Container IP addresses	
UDR	192.168.70.136
AMF	192.168.70.132
AUSF	192.168.70.138
UDM	192.168.70.137
SMF	192.168.70.133
UPF	192.168.70.134
NRF	192.168.70.130

Table 3.2: Core element addresses.

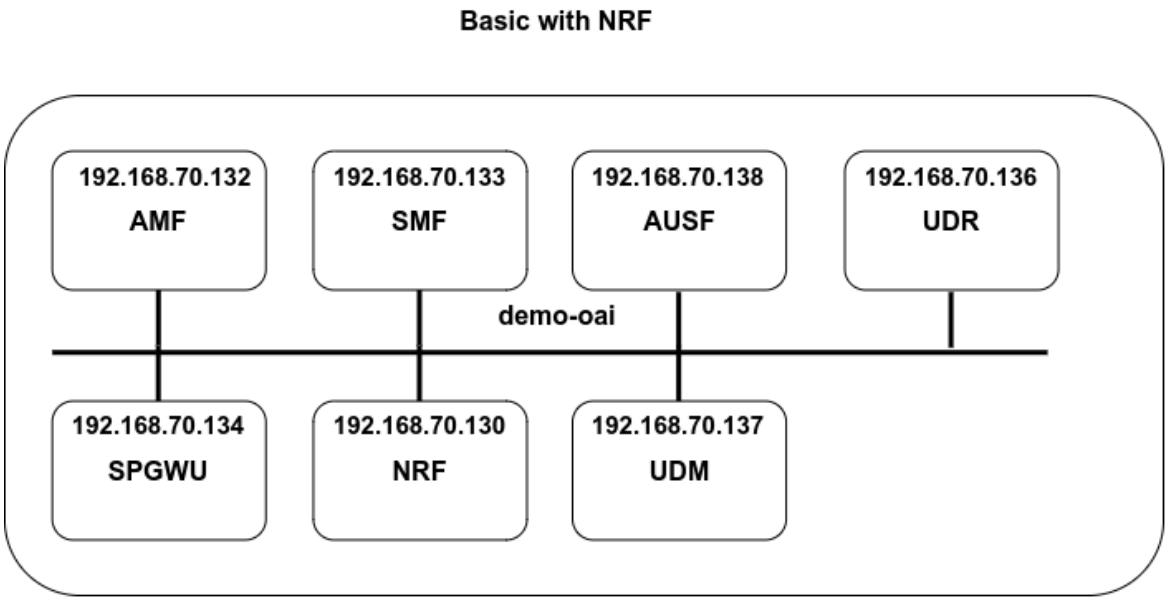

Figure 3.5: 5G Core Continers, and their IP addresses.

3.2.2 gNB Host

The PC responsible for hosting and running the gNB has to communicate with the Core, and it needs to have connected itself, via USB 3.0, to a USRP b210. As shown in figure 3.6.

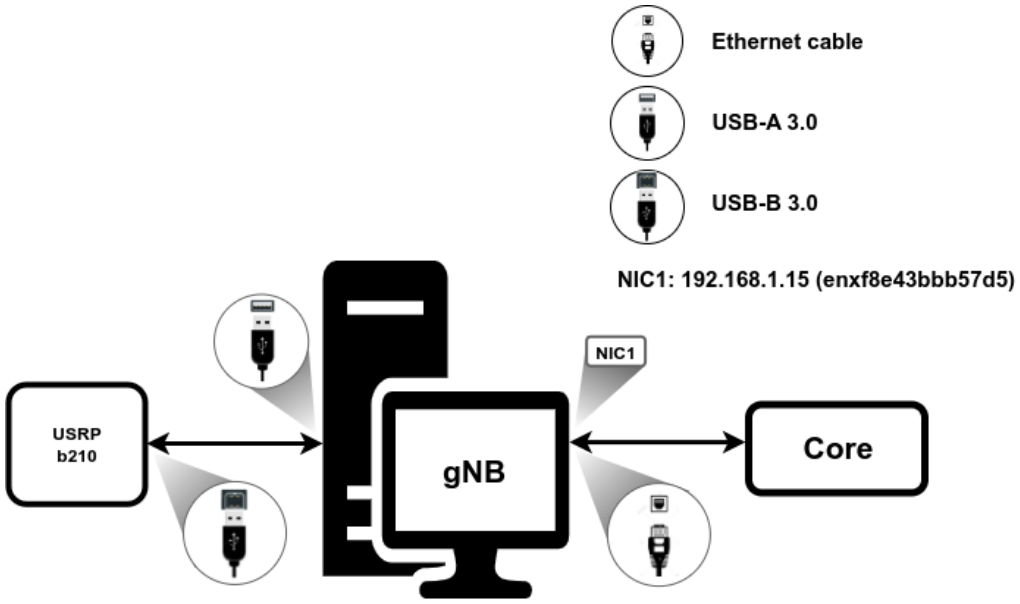

Figure 3.6: gNB Host connections, and IP address used on NIC.

In the setup that was implemented for this dissertation, the network card, identified by NIC1 in figure 3.6, with the name enxf8e43bbb57d5, was configured with the static IP address

192.168.1.15.

The host on which gNB runs has the following specifications, listed in the following table 3.3:

Specifications	
CPU	Intel Core i9-12900K
RAM	64 GB
Disk	1 TB
Operational System	Ubuntu 18.04.6 LTS
Kernel	4.15.0.192-lowlatency
NIC1	USB Type-C to RJ45 Gigabit Ethernet Network Adapter

Table 3.3: gNB computer specifications.

In the scope of this dissertation, as mentioned above, the implementation for the RAN part was a monolithic gNB.

On the computer where gNB runs, it needs to be configured due to the functions it is performing. It is necessary to use a low latency kernel, and disable c-states in the BIOS, also disabling c-states, and p-states in the kernel, and the governor flag needs to be in performance, to have all CPU cores at 100%, to have the best possible system response time. Then it is necessary to install the USRP on the machine that runs gNB, for it is necessary to download and install the software from the EttusRearch repository, the software version used was 4.2.0.0.

Finally, it was necessary to clone the openairinterface5g repository from OAI, and for 5G SA implementations, the develop branch must be used. By making a clone of the repository, all the software and configuration files necessary to configure and start gNB are made available. To obtain and install the OAI software, the following steps were followed. To get the software and choose the branch:

```
$ git clone https://gitlab.eurecom.fr/oai/openairinterface5g.git
$ cd openairinterface5g
$ git checkout develop
```

To compile and install all the software:

```
$ cd openairinterface5g
$ source oaienv
$ cd cmake_targets
$ ./build_oai -I
$ ./build_oai -w USRP --gNB -build-lib all -c
```

After carrying out all the process described above, the gNB was ready to work.

3.2.3 UE

Regarding the terminals used for testing, the operation of each one will be explained in more detail below.

3.2.3.1 OAI UE Emulator

The PC responsible for hosting and running the OAI UE needs to have a USRP connected via USB, in the scope of this dissertation for the OAI UE a b210 was used, as shown in figure 3.7, and through it, it is possible to establish a connection with the network.

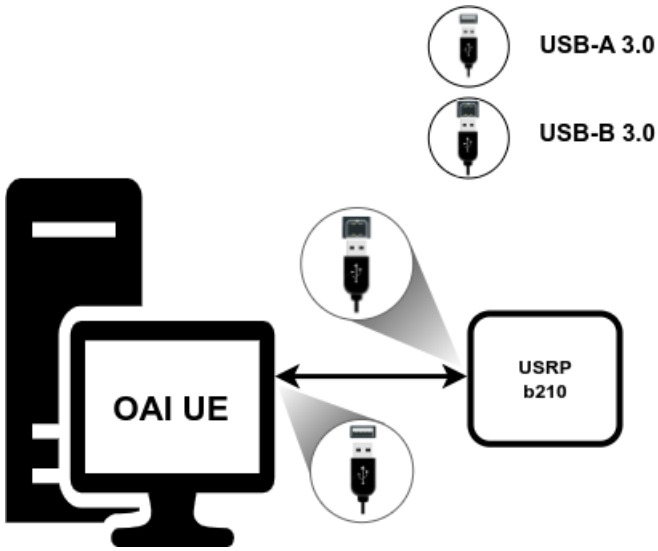

Figure 3.7: Connections between OAI UE and the USRP B210.

The pc on which the OAI UE runs has the specifications listed in the following table 3.4:

Specifications	
CPU	Intel Core i9-12900k
RAM	64 GB
Disk	1 TB
Operational System	Ubuntu 18.04.6 LTS
Kernel	5.4.0-122-lowlatency

Table 3.4: Core computer specifications.

To obtain the necessary software for the OAI UE, the process is similar to that described for the gNB. For the USRP, since the model used is the same, the process is the same as the one done for the gNB. Regarding the OAI software, the obtaining process is the same, making a git clone of the openairinterface5g repository, the only difference being in the installation process, which will install UE instead of gNB.

To install OAI UE, the following steps were performed:

```
$ cd openairinterface5g
$ source oaienv
$ cd cmake_targets
$ ./build_oai -I
$ ./build_oai -w USRP --nrUE -build-lib all -c
```

3.2.3.2 Quectel RM500Q UE

Using the Quectel modem, it is necessary to connect to a computer, via USB, as shown in figure 3.8.

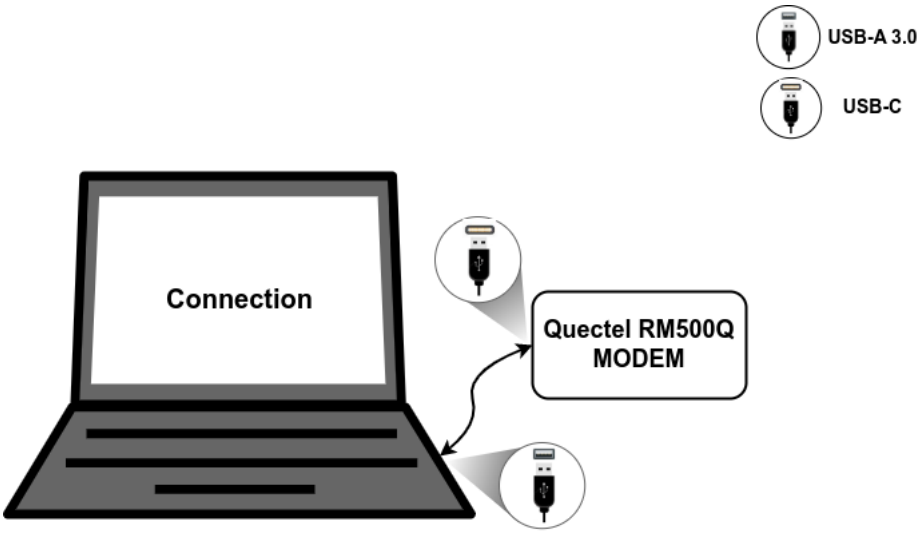

Figure 3.8: Connection between PC and Quectel RM500Q Modem.

The modem allows the computer to which it is connected to establish a connection with the network that has been implemented. To do so, it is necessary to program a SIM card, according to the data of the network to be connected, and configure a Data Network Name (DNN) on the PC.

3.2.4 USRP B210

The network implemented to be able to work needs to use a USRP, which is responsible for performing the RF functions of the setup.

The USRP model used in this dissertation was a b210, in which some of its characteristics are continuous RF coverage from 70 MHz to 6 GHz, with maximum bandwidths up to 56 MHz, full duplex, and Multiple-Input and Multiple-Output (MIMO) 2x2.

The USRP has a USB 3.0 type-B output, and it is through this output that it communicates with the pc running gNB.

3.3 Setup and Run

3.3.1 Core

Before running the core it is necessary to ensure that the International Mobile Subscriber Identity (IMSI), KEY and OPC of each user present in the database, located in the `/oai-cn5g-fed/docker-compose/database` directory, are equal to the values that are present in the configuration file of the UE, in case of using the OAI UE, or equal to the values

used to program the SIM Card, if using the Quectel Modem. It is also necessary to check the docker-compose configuration file, located in the `/oai-cn5g-fed/dokcer-compose` directory, because you must ensure that the Mobile Country Code (MCC) values are equal to the first 3 digits of the IMSI, and the Mobile Network Code (MNC) values are equal to the fourth and fifth digits of the IMSI. Also in the docker-compose configuration file, it is necessary to ensure that the Slice Service Type (SST) and SD values are the same as those used in the gNB configuration file. Finally, ensure that the DNN is the same as the one in the OAI UE configuration file, or the one in the Quectel Modem settings.

To run core, use the command

```
"python3 core-network.py --type stop-basic --scenario 1"
```

in the

```
~/oai-cn5g-fed/dokcer-compose directory.
```

3.3.2 gNB

To configure gNB correctly it is necessary to access the configuration file, and put the values according to what is intended for the setup.

To run gNB, in the `/openairinterface5g/cmake_targets/ran_build/build` directory, run the following command:

```
sudo ./nr-softmodem -O config_file_directory --sa -continous-tx -E
```

3.3.3 OAI UE Emulator

To run OAI UE it is necessary to edit the IMSI, KEY, OPC, and DNN values of the configuration file to match the desired ones. Once this process is done to run the OAI UE, in the `/openairinterface5g/cmake_targets/ran_build/build` directory, execute the following command:

```
sudo -S ./nr-uesoftmodem -r num_prb --numerology 1 --band num_band -C freq
--ue-fo-compensation --sa --nokrnmod -config_file_directory -E
```

3.3.4 Quectel RM500Q UE

To use the terminal with the modem, after programming the card and configuring the DNN, just connect to the usb port and wait for it to connect to the network.

3.4 Summary and Next Steps

In this third chapter, it was analyzed how to implement a 5G SA network based on the software provided by OAI, describing in detail the specifications of each of the hosts used for the different elements of the network, which must be configured on the different hosts, the software that is necessary to put in each of these host, and the way they communicate with each other. In the next chapter, the work carried out will be presented, which aims to automate much of the work carried out in this chapter as much as possible and make the OAI implementation process easier.

Chapter 4

Platform Implementation

In this chapter, the platform developed within the scope of this dissertation will be presented, which consists of a platform capable of assisting in the use of the OAI, for the implementation of a 5G SA network, through the automation of many the processes described in the previous chapter. It will be shown how the platform was built, and how each of the features that the platform offers to the user was developed.

4.1 Introduction

As can be seen in the previous chapter, the process required to implement a 5G SA network based on OAI software, and to get this network up and running, is a multi-step and time-consuming process, which can lead to the emergence of errors. The fact that confirms the previous statement is that when consulting the Maling List on the OAI website, and when searching for doubts and difficulties presented by people who are trying to implement a network, it appears that a substantial part of the problems reported are not related to problems with the available software, but due to the fact that the usability of the software is quite complex at some points, and also not very intuitive.

These problems that were enumerated before, constitute one of the biggest problems of OAI, which makes it a project with great potential, but its difficult usability takes away its value.

The platform developed within the scope of this dissertation aims to facilitate the installation and use of the OAI, especially for those who still have little experience, thus making the use of the software provided by OAI easier, more intuitive, and automated, allowing to work more efficiently.

The process of developing the different functionalities of the platform will be explained in the following topics of this chapter.

4.2 Platform Description

The platform was developed with the python language, the choice to use python for the development of the platform is due to the fact that it is a very rich language in libraries, allowing faster development of the platform, and facilitating the implementation of the various features of the platform. The platform was developed to run on a machine outside the OAI setup, communicating and controlling the setup via ssh connections, as shown in figure 4.1.

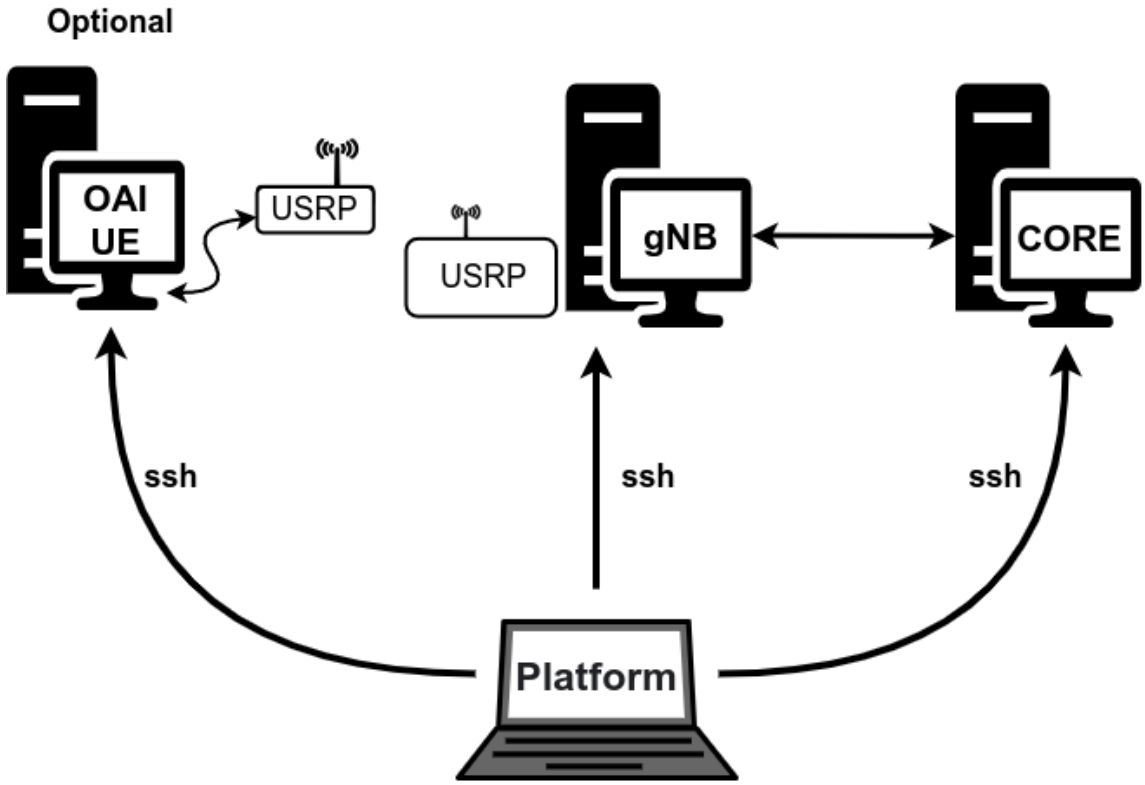

Figure 4.1: Schematic representing the way of integration of the platform with the implemented network.

To implement SSH communication, between the platform and the setup where the network is implemented, the "paramiko" library was used, this library uses the SSHv2 communication protocol, thus allowing the execution of commands on a remote host, the transfer of files between machines is done in a safe way[35].

The developed platform has a graphical interface to allow the user to have an easier, more comfortable, and intuitive user experience. The "PySimpleGUI" library was used for the construction of the graphical interface, this library provides a large number of resources, in a simple way, for the construction of the different interfaces necessary for the platform, thus constituting a good option to be used in this work [36].

The features that were developed for the platform can be grouped into 3 groups, features that help the user in the OAI software installation process, automating the process, features that facilitate the network configuration process, and features that help the user to test the Setup, as shown in the figure 4.2.

The way in which each of the features was implemented will be described below.

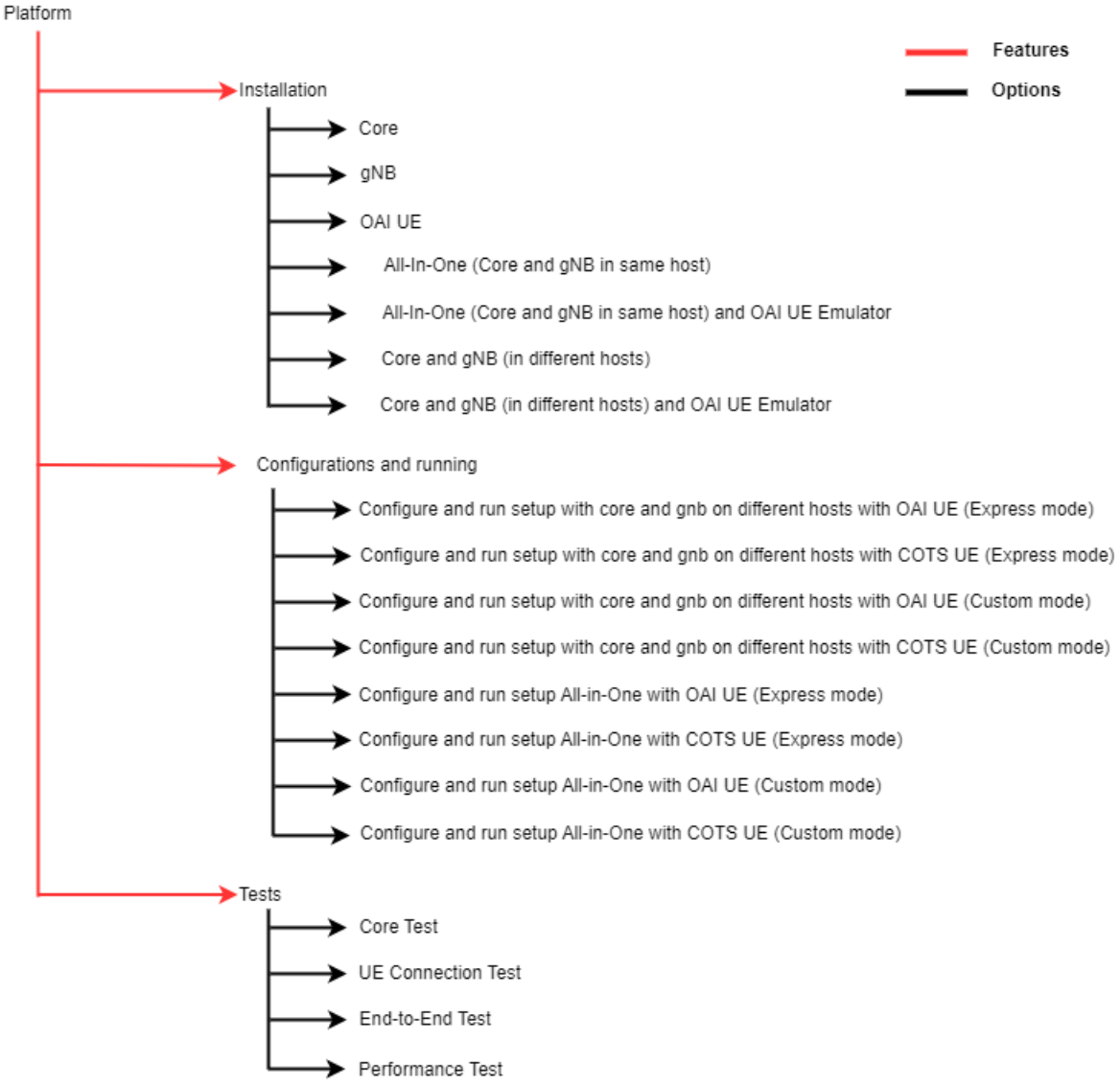

Figure 4.2: Platform features and options.

4.2.1 Installation Features and Procedures

The first functionality developed was the functionality that is responsible for assisting and automating the OAI software installation process. As described in the previous chapter, for the correct installation of the OAI software it is necessary to ensure that a set of other software is installed, both for obtaining the code and for its correct operation.

The platform allows the user to define the type of setup he wants to install, with options such as allowing each of the elements to be installed individually, that is, installing only the Core, gNB, or OAI UE.

It also allows the user to combine the installation of these elements at once, so that he does not need to install one at a time, thus allowing the user in a single installation process to have the setup completely installed. Since the possible types of setup are: All-in-One, which consists of installing Core and gNB on the same host, another possible setup would be to install Core and gNB on different hosts, and for both setups, the user can still define whether or not you want to install OAI UE on another host.

Regardless of the setup chosen, the installation processes for each of the components always follow the same steps. These steps will be listed below, in detail, for each of the elements.

4.2.1.1 Core

To install the Core, the platform works as follows, based on the information provided by the user, the platform can access it through an SSH connection with the remote host.

The first process performed by the platform is to transfer a shell script to the remote host. This shell script contains the instructions for various software and packages, which are as follows, the shell script will install git, which is necessary to obtain the software from the OAI repository, it also has instructions to install the packages net-tools which provide a set of tools for network configurations, is also responsible for installing docker, which is essential for running the images of the different Core elements.

After transferring the shell script to the host, the platform sends the command that will increase the script's permissions on the target host, in order to make that script executable. Having the script the permissions to be able to be executed, the platform sends a command that executes the shell script.

At the end of the script execution, the PC will restart automatically.

After the machine restarts, the platform sends a second shell script. This second shell script, after being transferred and increasing its permissions, when it is executed it will pull the official images of the different elements of the OAI Docker-Hub, namely AMF, NRF, SMF, UDR, UDM, AUSF, UPF(spgwu-tiny), in addition to these images, the images of ubuntu:bionic, mysql:bionic, and TRF-GEN-CN5G are also pulled, which are used to perform network traffic tests. Finally, when all images are available, the script will tag the images. The processes described above are shown in figure 4.3.

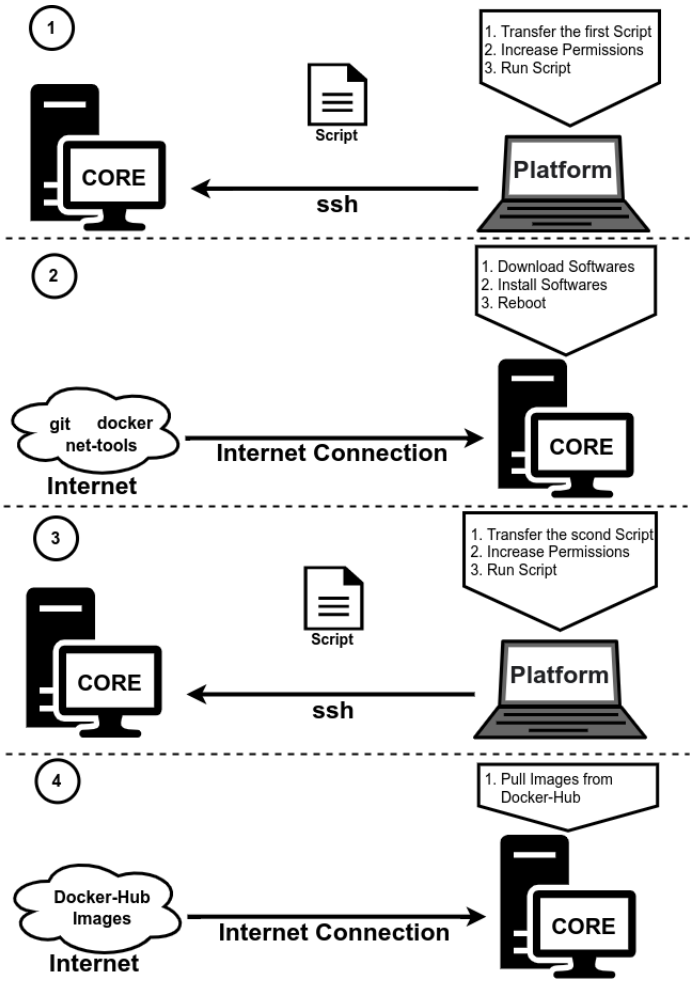

Figure 4.3: Core installation process, performed by the platform.

After this process is finished, the core is ready to be used.

4.2.1.2 gNB

For the installation of gNB, the platform works as follows, based on the data provided by the user, the platform can access the host through an SSH connection. The first step carried out by the platform is to send a shell script to the host where gNB will be installed. This shell script is responsible for installing git on the target host, in order to obtain the EttusReach software for installing the USRP on the host, and for obtaining the OAI software to be installed on the target host.

Once the shell script transfer is complete, the platform will send a command to increase the permissions on the target host, after which the platform will send a command to execute the shell script.

The installation process on the machine running gNB is as follows, after obtaining the software from the EttusRearch repository, the drivers and software necessary for the USRP to work on the host will be installed, after this process is finished and the software is obtained

from the repository of the OAI, the gNB related software will be compiled and installed on the host. After this process, the gNB is ready to be used.
The gNB installation process is described in figure 4.4.

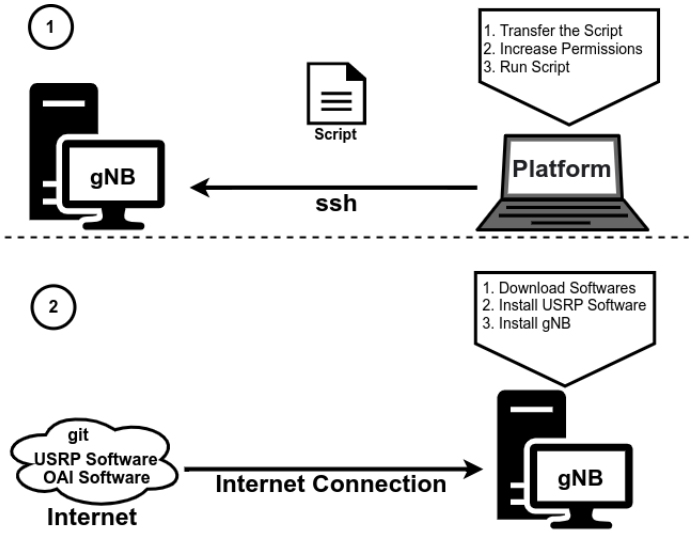

Figure 4.4: gNB installation process, performed by the platform.

4.2.1.3 OAI UE Emulator

For OAI UE, the process is similar to what is done for installing gNB, in which a shell script is also transferred to the target host, which also installs git, software, and driver to run USRP. The difference with gNB is that now the code that is compiled and installed on the host is what refers to the OAI UE. The OAI UE installation process is described in figure 4.5.

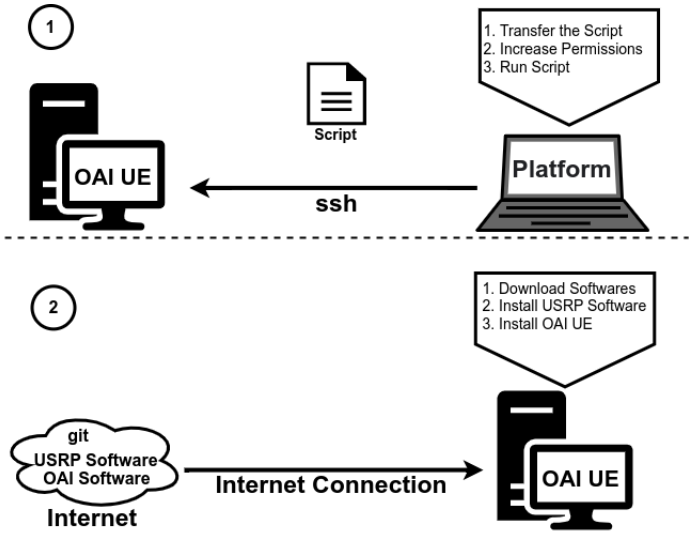

Figure 4.5: OAI UE installation process, performed by the platform.

4.2.2 Configuration Features and Procedures

The second functionality that was implemented for the platform is the functionality that helps the user in configuring the network and putting the network into operation. This functionality was developed so that the user has two options to configure the network and to run the network.

The first option corresponds to the network configuration automatically, where the platform uses a pre-defined configuration, and consists of the following steps, the platform has in its architecture a folder that contains several Templates, in the `/Platform/Template` directory, in this directory, several configuration files are stored, and when this option is selected, the platform transfers two files to the Core, one relating to the database that is transferred to the directory `/oai-cn5g-fed/docker-compose/database`, and a docker-compose file to the `/oai-cn5g-fed/docker-compose` directory. After these files are transferred, the platform sends a command to increase the database permissions. Finally, the platform puts the Core up and running.

For gNB, the platform transfers from the Template folder to the host where gNB is running, a configuration file, this configuration file is transferred to the directory `/openairinterface5g/targets/PROJECTS/GENERIC-NR-5G/CONF` . After the file has been downloaded, the platform sends the command to run gNB.

For the UE, there are two possibilities for the platform to function, depending on the options chosen by the user. If the user has selected the option indicating that the terminal is the OAI UE, the platform transfers the preconfigured configuration file, for the machine that runs the OAI UE, to the directory `/openairinterface5g/targets/PROJECTS/GENERIC-NR-5G/CONF`. Once the transfer is complete, the platform executes the command to run the OAI UE. If the user has indicated that he is using a COTS UE, the platform automatically programs a SIM card, using the software provided by OpenCells, which comes with the platform.

The second operating option is when the user wants to define the parameters of the different elements of the network, and then put the network into operation. The process is carried out as follows: For Core, as described above, the platform also goes to the Template folder to get the files for the database and the docker-compose file, but before transferring these two files to the machine where the Core is, updates these files with the configuration information that the user has introduced, after the platform updates these two files, transfer the files, and puts the Core in operation, as described above.

For gNB the process is similar, the configuration file is also updated with the information entered by the user, and then it is transferred to gNB. Finally, the gNB is put into operation.

For the terminal, when the OAI UE is being used, the configuration file is updated according to the information given by the user, then it is transferred to the machine that is running the OAI UE. When the terminal is a COTS UE, the SIM card is programmed according to the information given by the user. The process of this functionality is described in figure 4.6.

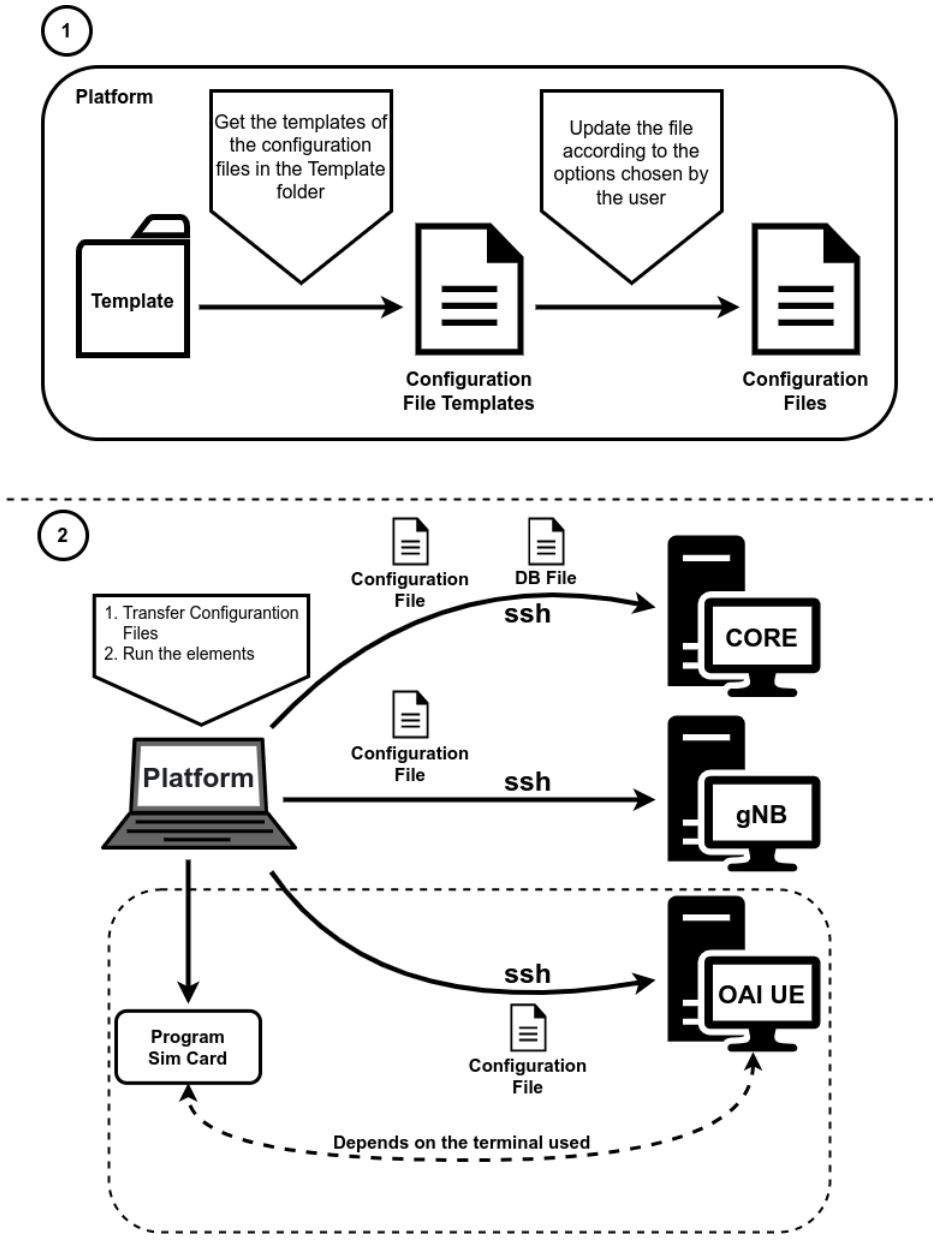

Figure 4.6: Configuration process and running the network, performed by the platform.

4.2.3 Test Features and Procedures

The third functionality that was implemented corresponds to the automation of network tests. The platform has implemented 4 different tests that can be done to the network. The realization of these tests is all with the OAI UE.

The first test implemented is a test to verify if the Core is working, this test consists of, the platform sending through the machine that runs gNB, a set of ping's to the IP address of the AMF, saving the result in a file. Then the platform will fetch that file, and analyze the result, and depending on the result, the platform transmits the test results to the user. The process is described in figure 4.7.

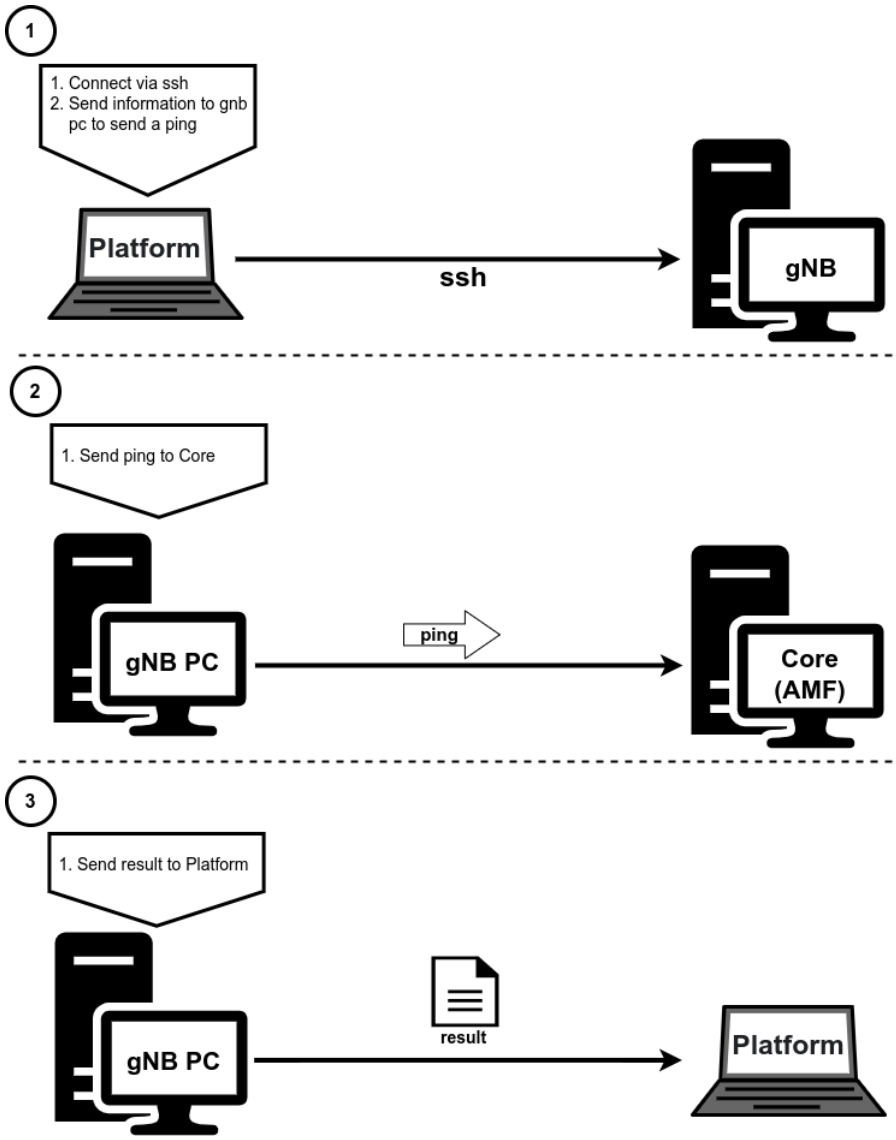

Figure 4.7: How the testing process is performed to verify that the Core is working.

The second test implemented is a test to verify that the OAI UE connects to the network, this test consists after all elements of the network are running, the platform executes an ifconfig command on the machine on which the OAI is running UE, saving the results to a file. Then that file is transferred to the platform, and the result is analyzed, checking if any IP address has been assigned to the OAI UE. Depending on the result, the platform transmits the test result to the user. The process is described in figure 4.8.

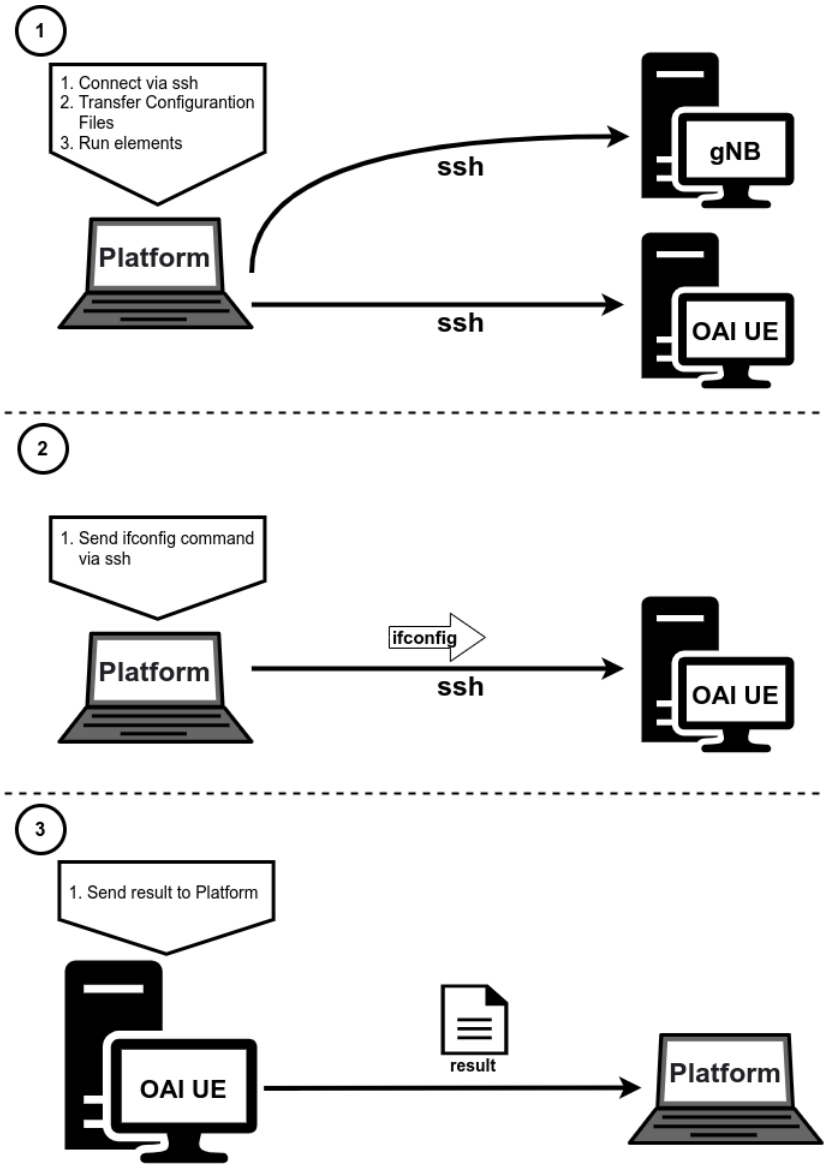

Figure 4.8: How the testing process is performed to verify that the OAI UE connects to the network.

The third test implemented is a test that verifies the end-to-end connection. The process for implementing this test consists of the following after all elements of the network are working, the platform sends a command to the OAI UE, forcing it to send a ping to the address 8.8.8.8 (google) through the created connection over the network, saving that result in a file, and depending on the result, the platform transmits the test results to the user. The process is described in figure 4.9.

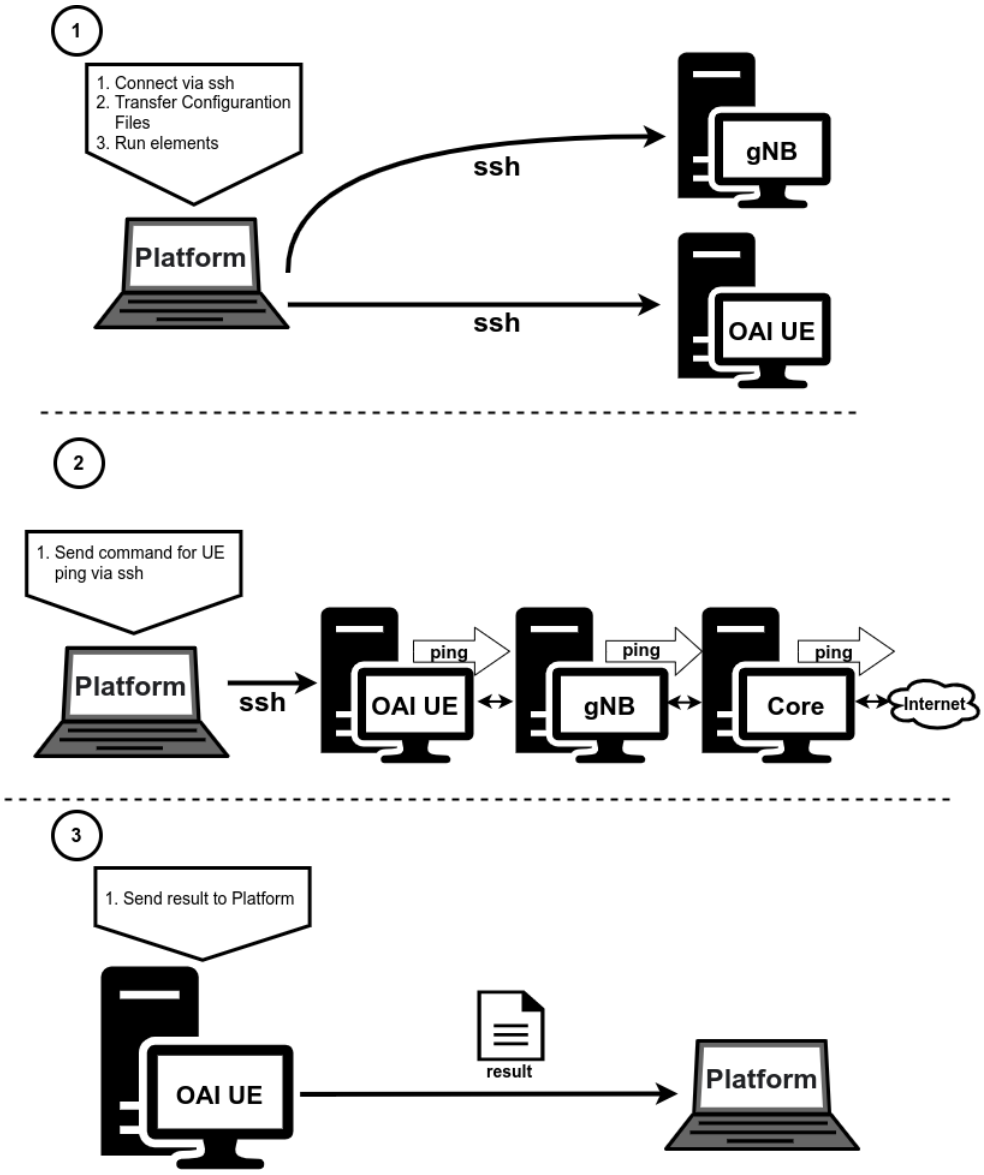

Figure 4.9: How the testing process is performed to verify the end-to-end connection.

The fourth test implemented is a network performance test implemented using iperf, where after the Core, the gNB, and the OAI UE are up and running, the platform sends a command to the UE so that it is ready to receive traffic, and a command is simultaneously sent to the Core to generate traffic in the trf-gen-5gcn container, in the end, the UE results are saved and transferred to the platform. The process is described in figure 4.10.

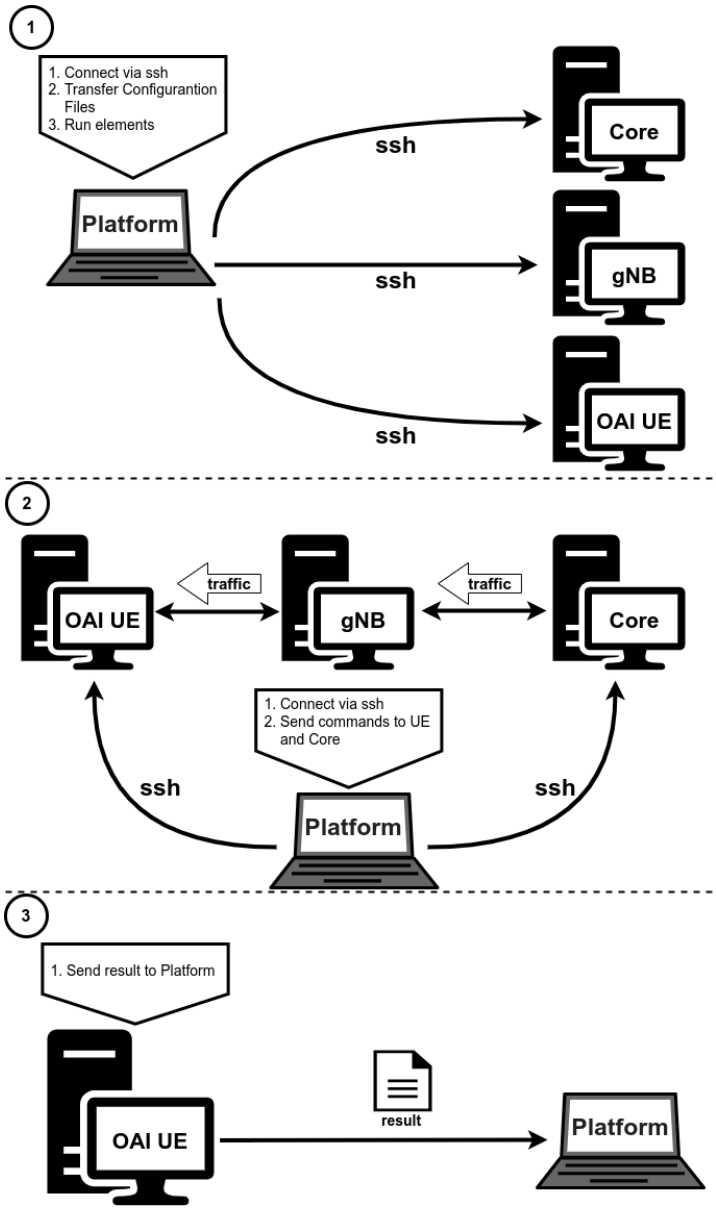

Figure 4.10: Implementation of the network performance test.

To see the implementation details in python and obtain the platform code that was described in this chapter, access the github repository at `https://github.com/RuiMRPereira/OAIPlatform.git`

4.3 Summary and Next Steps

This chapter it was explained how the platform integrates and interacts with the OAI software, how the platform was developed, explaining how each of the features made available by the platform was implemented. In the next chapter, it will be discussed how to use the platform and the results of its use.

Chapter 5

Results

In this chapter, the analysis of the results will be made, first, the results of the implementation carried out in the first part of the work will be shown. Finally, the developed platform will be presented, showing the functioning of different functionalities, and their interfaces.

5.1 Introduction

The work done in this dissertation resulted in a first phase in the implementation of a 5G SA network, and a second phase in the development of a platform to assist the use of the OAI software. In the first phase, which was the implementation of the network, it was necessary to carry out the whole process of preparing the different hosts and installing the necessary software, and the configuration of the different elements that make up the network.

Tests were conducted with two UEs, the OAI UE Emulator, which is provided by OAI and allows for simpler tests, and the Quectel RM500Q modem which allows for more complete tests, allowing to replicate a usage closer to reality.

In the second phase of this dissertation's work, the platform that facilitates the use of the OAI software was developed. The developed platform can offer the user assistance in the installation process of the OAI software, automating this process, and is also able to automate the entire configuration process of all the network elements, or facilitate the configuration of the same, depending on what the user wants. And finally, it also offers a test suite, which allows to identify possible problems in the implemented network and test its performance.

The results of the work are divided into two parts. In the first part the results are shown, where it is possible to see that the network was successfully implemented, and with the result in the performance test performed on the network. The results obtained in the network performance tests are similar regardless of whether or not the platform developed in the second part of the work was used since the focus of the platform is on improving the usability of the software, and not on improving performance. In the results of the second part of the work, the different interfaces and the order in which they are presented to the user are shown, where the comparison between using the platform and not working with the OAI software will be made.

5.2 Test Setup

In the first part of the work done in this dissertation, as shown in previous chapters, the test setup consisted of a machine where the Core was installed and executed, and this machine communicated with another machine, via Ethernet cable, where the gNB was installed and executed. The machine where the gNB is located has a USRP B210 connected to it, which is responsible for ensuring RF communications. To test the implemented network two UEs, referred to above, were used. For the OAI UE Emulator a third machine was used where the software available from OAI to Emulate the UE was installed and executed, this machine also needed, like the gNB, a USRP B210 to communicate via RF with the network. For the tests using the Quectel RM500Q modem, an ordinary PC was used, where it was enough to connect the Modem and configure the DNN, thus allowing the use of the implemented network. The scenarios described above are shown in Figure 3.2 in Chapter 3.

The implementation used to perform the tests has the following characteristics, Band 78 was used, which corresponds to the frequency range from 3.3GHz to 3.8GHz, using a Subcarrier Spacing of 30kHz, and a bandwidth of 40MHz.

For the second part of the work, which was the development of the platform, the platform runs on a machine outside the Setup, which controls the different hosts via SSH. As shown in Figure 4.1 in Chapter 4.

5.3 Analysis

The following sections will analyze the results obtained in this dissertation.

5.3.1 Implementation Testing

The logs obtained in the Core, and in the gNB are the equal regardless of the UE type used in the test. Both the OAI UE Emulator and the SIM Card used in the modem have the same IMSI value, which for this test was 2089900000000000001, and use the same DNN, called "`oai`". The gNB used for testing was assigned the name "`gNB`" and an ID with a value of "`0xc000`", and these values were used for both UEs. The network was also configured with an MCC of 208 and an MNC of 99.

In the following figures, it is possible to observe the logs that the system presents when it is running, with a connected UE.

In the figure 5.1, it is possible to observe the AMF logs when a UE is connected to the system. It is possible to see that in the information regarding the gNB, the state is "connected", presenting the name, ID, MCC, and MNC according to the configurations that were made. The information about the UE it is possible to see through the state that there is a UE registered in the Core, presenting Public Land Mobile Network (PLMN) values equal to the gNB, and by analyzing the first 5 digits of the IMSI, which corresponds to the concatenation of the MCC and MNC values, which also verifies that they are following what was expected.

```
|-----------------------------------------------------------------------------------------------------------|
|-----------------------------------------------------UEs' information-------------------------------------|
| Index |      5GMM state      |      IMSI        |     GUTI      | RAN UE NGAP ID | AMF UE ID |  PLMN   |Cell ID|
|     1|       5GMM-REGISTERED|   208990000000001|               |      2604350524|          1| 208, 99 |14680064|
|-----------------------------------------------------------------------------------------------------------|

|-----------------------------------------------------------------------------------------------------------|
|-----------------------------------------------------gNBs' information-------------------------------------|
|    Index    |      Status      |       Global ID       |       gNB Name       |              PLMN            |
|      1      |    Connected     |         0xe000        |         gNB          |           208, 99            |
|-----------------------------------------------------------------------------------------------------------|
```

Figure 5.1: Core AMF logs with a registered user.

In the figure 5.2 the SMF logs are shown, where is possible to see that the IMSI value is the same as shown in the AMF logs, the DNN that was configured on the UE agrees with the Core and the IP address that the Core assigned to the UE, which in this test was 12.1.1.2.

```
SMF CONTEXT:
SUPI:                           208990000000001
PDU SESSION:
        PDU Session ID:         10
        DNN:                    oai
        SNSSAI:                 SST=1, SD=1
        PDN type:               IPV4
        PAA IPv4:               12.1.1.2
        Default QFI:            6
        SEID:                   2
        Default QoS Flow:
                QFI:            6
                UL FTEID:       TEID=1, IPv4=192.168.70.134
                DL FTEID:       TEID=721966697, IPv4=192.168.1.15
                PDR ID UL:      1
                PDR ID DL:      2
                Precedence:     0
                FAR ID UL:      1
                FAR ID DL:      2
```

Figure 5.2: Core SMF logs with a registered user.

From the information presented by the AMF and SMF logs, it can be concluded that the UE is correctly connected to the network.

In figure 5.3, it's possible to see the logs that the gNB shows when it has a UE connected to the network, from observing the gNB logs is possible to see that data packets are being exchanged between the UE and the Core.

```
[NR_MAC]   Frame.Slot 256.0
UE RNTI f6d8 (1) PH 0 dB PCMAX 0 dBm, average RSRP -110 (8 meas), UL-SNR 29 dB
UE f6d8: CQI 0, RI 1, PMI (0,0)
UE f6d8: dlsch_rounds 2344/2/0/0, dlsch_errors 0, pucch0_DTX 0, BLER 0.00000 MCS 9
UE f6d8: dlsch_total_bytes 288907
UE f6d8: ulsch_rounds 69713/296/0/0, ulsch_DTX 0, ulsch_errors 0, BLER 0.00000 MCS 9
UE f6d8: ulsch_total_bytes_scheduled 6488838, ulsch_total_bytes_received 6488580
UE f6d8: LCID 1: 931 bytes TX
UE f6d8: LCID 4: 18 bytes TX
UE f6d8: LCID 4: 334 bytes RX
```

Figure 5.3: gNB logs with a UE connected.

5.3.1.1 Test with OAI UE Emulator

When using the OAI UE Emulator as a terminal, when connecting to the network the UE shows the logs that are shown in figure 5.4

```
[NR_PHY]   ============================================
[NR_PHY]   Harq round stats for Downlink: 2808/2/0/0 DLSCH errors: 0
[NR_PHY]   ============================================
[NR_MAC]   [768.0] Received TA_COMMAND 31 TAGID 0 CC_id 0
[NR_PHY]   Adjusting frame in time by -2 samples
[NR_PHY]   Adjusting frame in time by -3 samples
[NR_PHY]   Adjusting frame in time by -2 samples
[NR_PHY]   Adjusting frame in time by -2 samples
[NR_PHY]   Adjusting frame in time by -2 samples
[NR_MAC]   [778.0] Received TA_COMMAND 31 TAGID 0 CC_id 0
[NR_PHY]   Adjusting frame in time by -2 samples
[NR_PHY]   Adjusting frame in time by -3 samples
[NR_PHY]   Adjusting frame in time by -2 samples
[NR_PHY]   Adjusting frame in time by -2 samples
[NR_PHY]   Adjusting frame in time by -2 samples
```

Figure 5.4: UE logs when connected.

The tests performed by the OAI UE Emulator served essentially to verify if the network was well configured and working correctly, in figure 5.5 it is possible to observe one of the tests performed with this UE, where it is also possible to verify that the UE has the IP address 12.1.1.2, which is in accord with the information shown by the SMF log. The test consisted in sending a ping to the address 8.8.8.8 (google) through the implemented network, the test being successful, it can be concluded that the implementation was done correctly.

```
PING 8.8.8.8 (8.8.8.8) from 12.1.1.2 oaitun_ue1: 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=113 time=51.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=113 time=56.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=113 time=32.0 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=113 time=31.0 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=113 time=29.9 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=113 time=27.0 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=113 time=59.9 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=113 time=56.0 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=113 time=68.0 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=113 time=107 ms
64 bytes from 8.8.8.8: icmp_seq=11 ttl=113 time=33.0 ms
```

Figure 5.5: Test with OAI UE Emulator.

5.3.1.2 Test with Quectel RM500Q Modem

When the test terminal used was the Quectel RM500Q Modem, and the network is working properly, what is observed on the PC where the Modem is connected should be similar to what is shown in figure 5.6. Where is possible to see that the PC is connected to a mobile network, with the name "`OpenAirInterface`", which is the name of the network implemented.

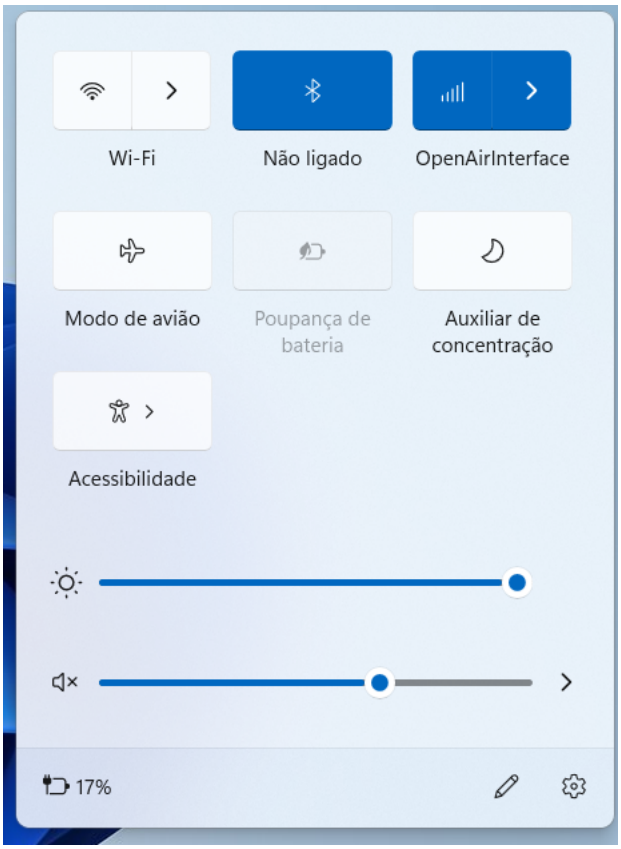

Figure 5.6: Quectel RM500Q modem connected to PC.

It was with the Quectel RM500Q Modem that the network performance tests were carried out, in order to test how it is a real use, it was tried to use in various situations, such as watching videos varying their quality, to which the network managed to perform satisfactorily.

In the various performance tests performed, the results obtained were DownLink speeds close to 100 Mbps and UpLink speeds close to 8 Mbps, in figure 5.7 its possible to see a screenshot of one of these tests. The results obtained for the Downlink and Uplink speeds are in line with what was expected, according to the information available on the OAI website, for an implementation with a bandwidth of 40 MHz, and using hardware similar to the one used in this dissertation.[37][38]

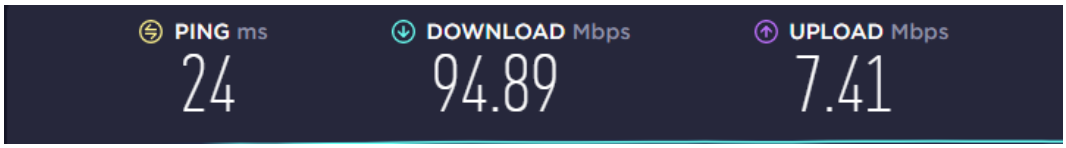

Figure 5.7: Performance test, using speed test.

5.3.2 Platform Results

The source code of the developed platform is available in the github repository at `https://github.com/RuiMRPereira/OAIPlatform.git`. The development of the platform had the objective of getting the platform to do the work carried out in the first part of the work in the most automated way possible, and thus being able to understand the advantage of its use. The way the platform works is possible to observe the results in the attached images. The attached images appear in the order in which the interfaces are presented to the user.

The OAI installation functionalities in an automated way, it is possible to see the 7 possible installation options, in annexes A.1.1, A.1.2, A.1.3, A.1.4, A.1.5, A.1.6, A.1.7. Comparing the installation time with the traditional/manual way, the decrease in installation time is noticeable.

The configuration features and running the network, in annexes A.2.1, A.2.2, A.2.3, A.2.4, A.2.5, A.2.6, A.2.7, A.2.8, it is possible to observe their operation, and the order in which each interface is presented to the user. Compared to manually configuring and running the network, using the platform this process became much faster, and the errors associated with the configuration process were reduced to zero.

The test features that were developed and their functioning can be seen in annexes A3.1, A3.2, A3.3, A3.4, and A3.5. This set of tests developed, the information that is transmitted when using, and the suggestions given by the results of each one, allow the user to be able to verify the correct functioning of the network that has been implemented. And when the implemented network presents a problem, the platform, with the information that it shows to the user, allows him to quickly correct the problem, for that the platform transmits the possible origin of this problem, giving suggestions for the user to execute other tests to identify the problem more precisely.

Comparing the use of OAI in a traditional way with the use of OAI together with the platform, it was clear that the platform allows the user to perform more tasks, tests, and solve problems in a much more efficient and simple way.

5.4 Summary and Next Steps

In this chapter the results of the different phases of the work were shown, this included the tests for the implementation and the interfaces of the developed platform. In the next chapter the conclusions will be made, and an analysis of what can be done in the future.

Chapter 6

Conclusions and Future Work

In this chapter, conclusions will be drawn about the work carried out, and the future work that may result from this dissertation. In the conclusions, an analysis will be made of the work carried out, analyzing its value. In future work, it will be indicated what can still be done in the work carried out, the aspects that can be improved, and the work that can still be done.

6.1 Conclusions

This dissertation had as some of the objectives, to make an in-depth study on 5G SA networks, first studying its architecture, then analyzing the different open-source projects that make it possible to implement the network.

In the initial phase of this dissertation, a study was made on the architecture of a 5G SA network, analyzing the function of the Core, and the gNB. For the Core, the function of each of the services it contains was also studied and described. For the gNB, a study was carried out in each of the functional units that constitute it, CU, DU, and RU, showing how the protocol stack was divided by each of these units.

After that, a 5G network with an SA architecture was deployed and tested. From the analysis of some options available to implement the network, it was decided to use the software provided by OAI. The choice for the OAI was because it is an open-source project of reference, and in which more information could be found. Despite that, in the initial phase of the deployment, the information related to implementations with the SA architecture was still limited, due to the OAI itself that had only recently enabled implementation of an end-to-end SA network.

In the last phase of the work of this dissertation, the platform was developed to assist in the use of the OAI software, the need to develop the platform is because, throughout the work carried out in this dissertation, it was verified that many of the problems and difficulties was due to the way of working with the OAI. These problems ranged from misconfigured parameters, as in some cases the configuration files are quite confusing, to problems with missing routes configuration on some machines whenever the deployment was changed. In addition, the time spent whenever it was necessary to change something in the deployment was a constraint. Due to all this, the platform was developed to solve these problems, automating as many processes as possible.

The goals regarding features to automate installation, configuration and testing processes

have been achieved. However, the functioning of the platform must be improved, as the platform would need greater robustness in its functioning, and it should be able to transmit more information to the user. Despite these limitations, the platform is already able to offer a substantial improvement in the use of OAI.

6.2 Future Work

The developed platform still has several aspects to be improved. In terms of robustness, the platform is still not in good condition, in case of failures during the execution of some features, the platform still cannot transmit this information to the user, and it is still necessary to work to improve that. Another aspect to be improved is the information that the platform can show the user, mainly information related to the Core, it is necessary to develop the platform so that the user has access to this information, as in the case of the logs of each element of Core. Finally, visually there is still work to be done, improving the appearance of the interfaces that are presented to the user.

Bibliography

[1] L. Guevara and F. A. Cheein, "The role of 5G technologies: Challenges in smart cities and intelligent transportation systems," *Sustainability (Switzerland)*, vol. 12, no. 16, Aug. 2020, ISSN: 20711050. DOI: 10.3390/SU12166469. [Online]. Available: https://www.researchgate.net/publication/343577759_The_Role_of_5G_Technologies_Challenges_in_Smart_Cities_and_Intelligent_Transportation_Systems.

[2] S. K. Rao and R. Prasad, "Impact of 5G Technologies on Industry 4.0," *Wireless Personal Communications*, vol. 100, no. 1, pp. 145–159, 2018, ISSN: 1572-834X. DOI: 10.1007/s11277-018-5615-7. [Online]. Available: https://doi.org/10.1007/s11277-018-5615-7.

[3] S. A. Abdel Hakeem, A. A. Hady, and H. Kim, "5G-V2X: standardization, architecture, use cases, network-slicing, and edge-computing," *Wireless Networks*, vol. 26, no. 8, pp. 6015–6041, 2020, ISSN: 1572-8196. DOI: 10.1007/s11276-020-02419-8. [Online]. Available: https://doi.org/10.1007/s11276-020-02419-8.

[4] *Non-standalone and Standalone: two paths to 5G - Ericsson*, https://www.ericsson.com/en/blog/2019/7/standalone-and-non-standalone-5g-nr-two-5g-tracks [Online; accessed 20-Jan-2022].

[5] Ethem Alpaydın, "5G PPP Architecture Working Group: View on 5G Architecture," *Version 3.0, June 2019*, no. June, pp. 21–470, 2019. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2019/07/5G-PPP-5G-Architecture-White-Paper_v3.0_PublicConsultation.pdf.

[6] *Macrocell vs. small cell vs. femtocell: A 5G introduction*, https://www.techtarget.com/searchnetworking/feature/Macrocell-vs-small-cell-vs-femtocell-A-5G-introduction[Online; accessed 10-Jan-2022].

[7] N. Wang, E. Hossain, and V. K. Bhargava, "Backhauling 5G small cells: A radio resource management perspective," *IEEE Wireless Communications*, vol. 22, no. 5, pp. 41–49, Oct. 2015, ISSN: 15361284. DOI: 10.1109/MWC.2015.7306536.

[8] *Tips and Trends: Small Cell 5G Systems - Qorvo*, https://www.qorvo.com/design-hub/blog/tips-and-trends-small-cell-5g-systems [Online; accessed 10-Jan-2022].

[9] *An Introduction to the 5G Small Cell - LitePoint*, https://www.litepoint.com/blog/an-introduction-to-the-5g-small-cell/ [Online; accessed 10-Jan-2022].

[10] *What is a small cell?* https://www.techtarget.com/searchnetworking/definition/small-cell [Online; accessed 10-Jan-2022].

[11] A. Pratap, R. Misra, and S. K. Das, "Resource Allocation to Maximize Fairness and Minimize Interference for Maximum Spectrum Reuse in 5G Cellular Networks," *19th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM 2018*, Aug. 2018. DOI: 10.1109/WOWMOM.2018.8449760.

[12] TSGR, "TS 138 401 - V16.5.0 - 5G; NG-RAN; Architecture description (3GPP TS 38.401 version 16.5.0 Release 16)," 2021. [Online]. Available: https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx.

[13] *What are Virtual Network Functions (VNF)? Management Orchestration*, https://www.thousandeyes.com/learning/glossary/vnf-virtual-network-functions[Online; accessed 13-Jan-2022].

[14] *Open RAN explained: innovation and flexibility - Ericsson*, https://www.ericsson.com/en/openness-innovation/open-ran-explained [Online; accessed 13-Jan-2022].

[15] TSGS, "TS 123 501 - V16.6.0 - 5G; System architecture for the 5G System (5GS) (3GPP TS 23.501 version 16.6.0 Release 16)," 2020. [Online]. Available: https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx.

[16] *"5G is here!" How do mobile transport providers get ready?* https://www.nokia.com/blog/5g-here-how-do-mobile-transport-providers-get-ready [Online; accessed 16-Feb-2022].

[17] *Why the eCPRI Interface is Critical to 5G — Keysight Blogs*, https://blogs.keysight.com/blogs/inds.entry.html/2020/09/30/why_the_ecpri_interf-TC33.html [Online; accessed 16-Feb-2022].

[18] E. Harstead, D. Van Veen, V. Houtsma, and P. Dom, "Technology roadmap for time-division multiplexed passive optical networks (TDM PONs)," *Journal of Lightwave Technology*, vol. 37, no. 2, pp. 657–664, Jan. 2019, ISSN: 07338724. DOI: 10.1109/JLT.2018.2881933.

[19] TSGR, "TS 138 401 - V16.3.0 - 5G; NG-RAN; Architecture description (3GPP TS 38.401 version 16.3.0 Release 16)," 2020. [Online]. Available: https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx.

[20] ——, "TS 138 300 - V16.2.0 - 5G; NR; NR and NG-RAN Overall description; Stage-2 (3GPP TS 38.300 version 16.2.0 Release 16)," 2020. [Online]. Available: https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx.

[21] *5G NR RLC*, https://devopedia.org/5g-nr-rlc [Online; accessed 14-Feb-2022].

[22] *5G NR MAC*, https://devopedia.org/5g-nr-mac [Online; accessed 08-Feb-2022].

[23] ——, "TS 138 202 - V16.1.0 - 5G; NR; Services provided by the physical layer (3GPP TS 38.202 version 16.1.0 Release 16)," 2020. [Online]. Available: https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx.

[24] *5G NR PHY*, https://devopedia.org/5g-nr-phy [Online; accessed 15-Feb-2022].

[25] ——, "TS 138 212 - V16.2.0 - 5G; NR; Multiplexing and channel coding (3GPP TS 38.212 version 16.2.0 Release 16)," 2020. [Online]. Available: https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx.

[26] *cn5g · GitLab*, https://gitlab.eurecom.fr/oai/cn5g [Online; accessed 15-Mar-2022].

[27] *oai / openairinterface5G · GitLab*, https://gitlab.eurecom.fr/oai/openairinterface5g/ [Online; accessed 15-Mar-2022].

[28] *5G CORE NETWORK – OpenAirInterface*, https://openairinterface.org/oai-5g-core-network-project/ [Online; accessed 11-Mar-2022].

[29] *5G RAN – OpenAirInterface*, https://openairinterface.org/oai-5g-ran-project/ [Online; accessed 11-Mar-2022].

[30] *OpenAirFeatures · Wiki · oai / openairinterface5G · GitLab*, https://gitlab.eurecom.fr/oai/openairinterface5g/-/wikis/OpenAirFeatures [Online; accessed 11-Mar-2022].

[31] *ONF Mobile Projects - Open Networking Foundation*, https://opennetworking.org/onf-mobile-projects/ [Online; accessed 11-Mar-2022].

[32] *SD-Core - Open Networking Foundation*, https://opennetworking.org/sd-core/ [Online; accessed 11-Mar-2022].

[33] *SD-RAN - Open Networking Foundation*, https://opennetworking.org/sd-ran/ [Online; accessed 11-Mar-2022].

[34] *SD-RAN - Community - Confluence*, https://wiki.opennetworking.org/display/COM/SD-RAN [Online; accessed 11-Mar-2022].

[35] *Welcome to Paramiko! — Paramiko documentation*, https://www.paramiko.org/ [Online; accessed 3-Sep-2022].

[36] *PySimpleGUI*, https://www.pysimplegui.org/en/latest/[Online; accessed 3-Sep-2022].

[37] F. Kaltenberger, "OpenAirInterface RAN Roadmap," 2020. [Online]. Available: https://openairinterface.org/wp-content/uploads/2021/12/Roadmap-5G-end-to-end-demo.pptx.

[38] ——, "OpenAirInterface RAN Roadmap," 2020. [Online]. Available: https://openairinterface.org/wp-content/uploads/2022/07/2022-07-12-EURECOM-RAN-SLIDES.pdf.

Appendix A

Platform operation

A.1 Installation Features

A.1.1 Core Installation

Figure A.1: Core installation process 1ª interface.

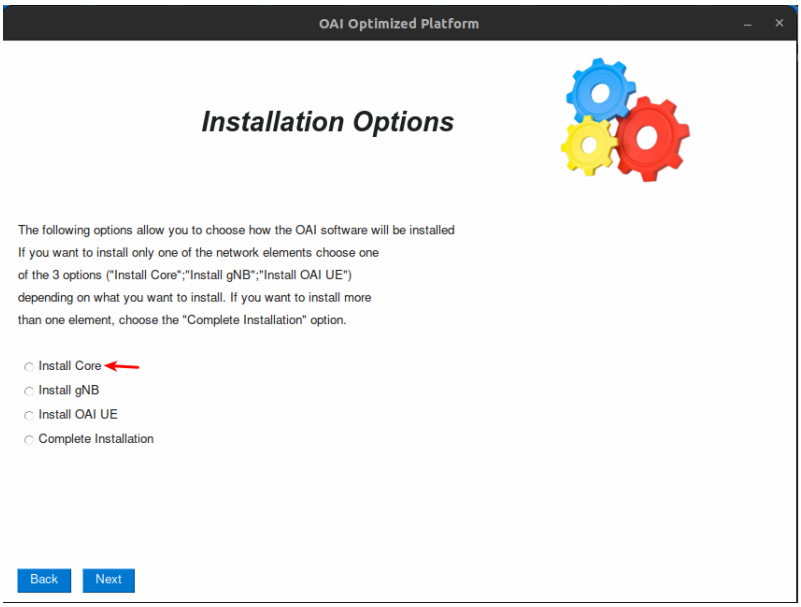

Figure A.2: Core installation process 2ª interface.

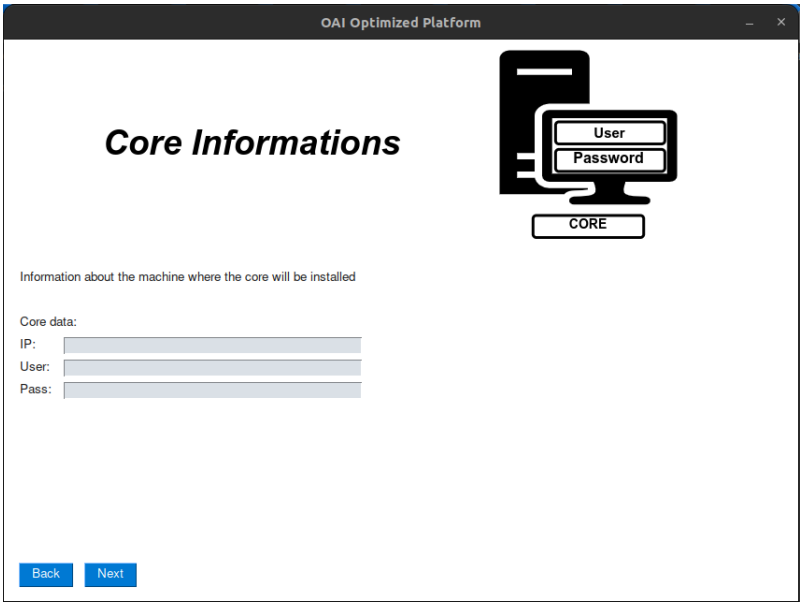

Figure A.3: Core installation process 3ª interface.

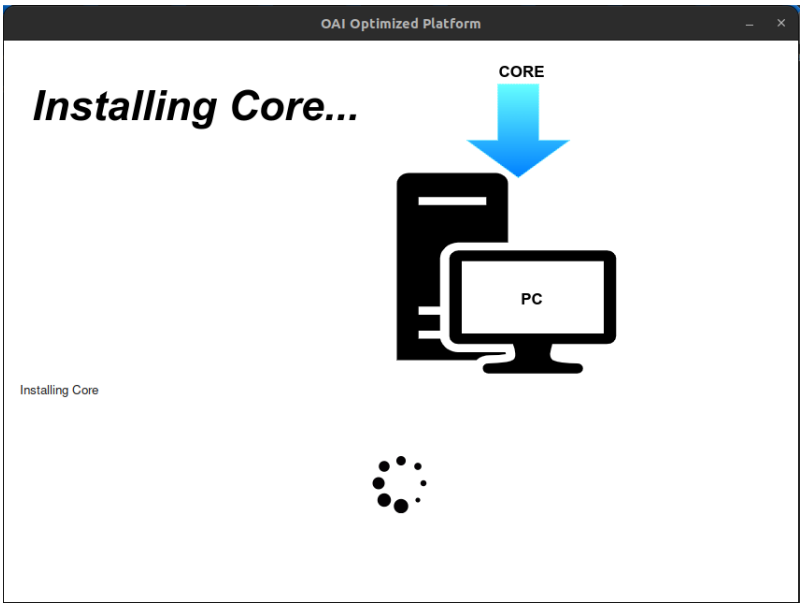

Figure A.4: Core installation process 4ª interface.

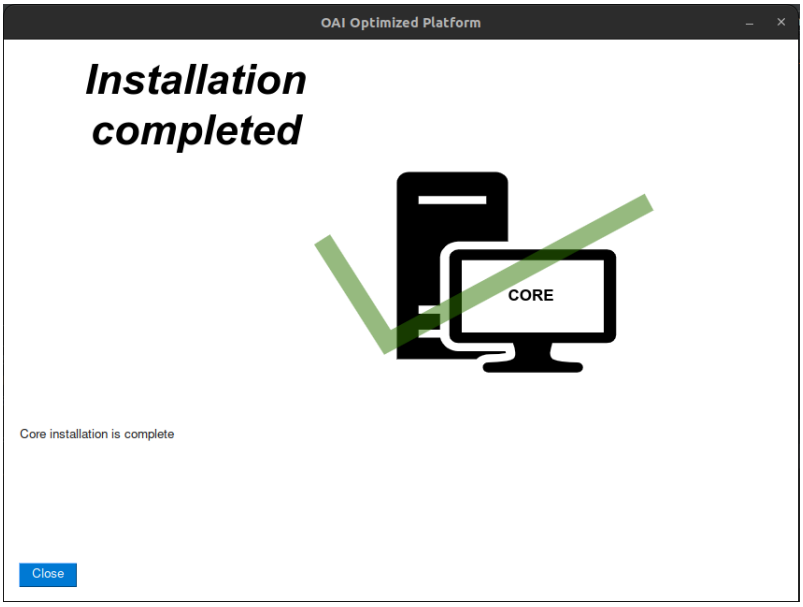

Figure A.5: Core installation process 5ª interface.

A.1.2 gNB Installation

Figure A.6: gNB installation process 1ª interface.

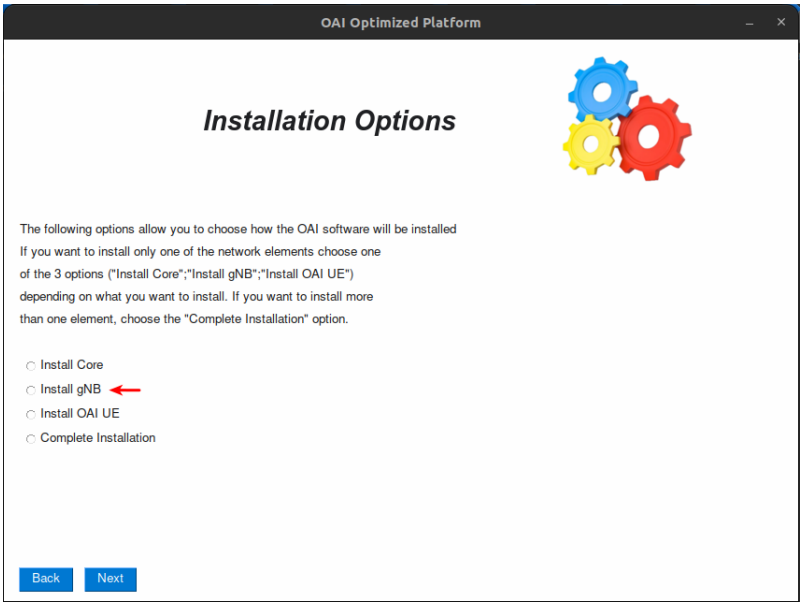

Figure A.7: gNB installation process 2ª interface.

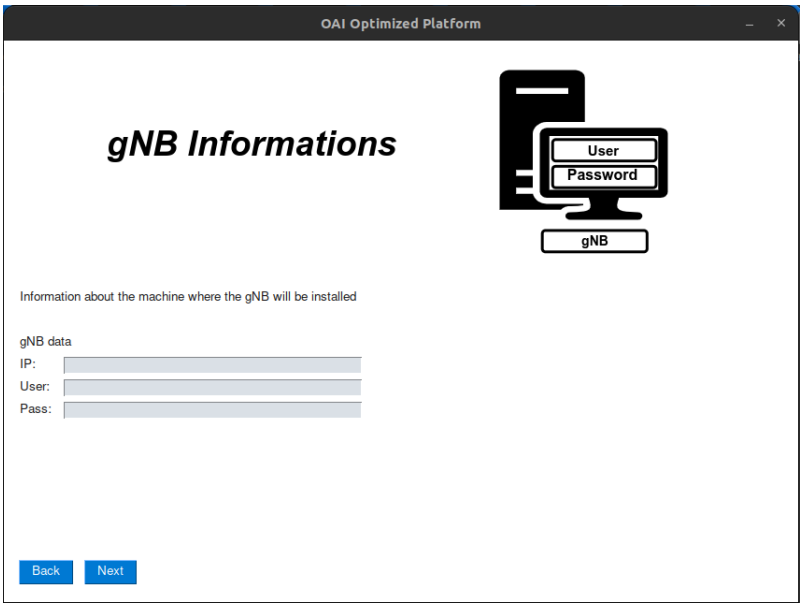

Figure A.8: gNB installation process 3ª interface.

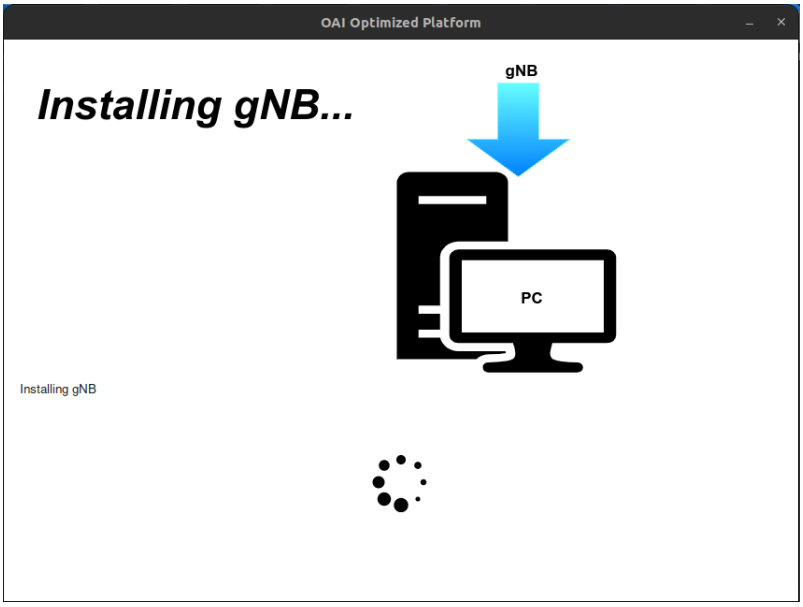

Figure A.9: gNB installation process 4ª interface.

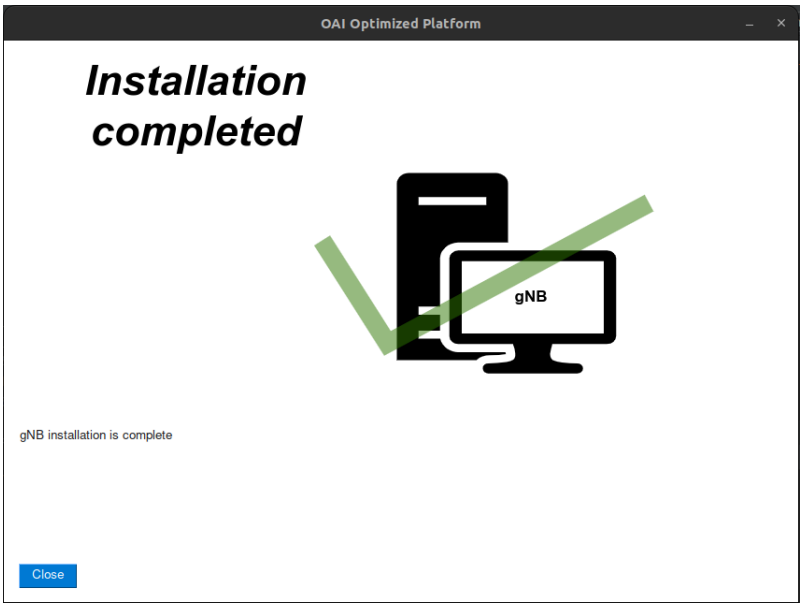

Figure A.10: gNB installation process 5ª interface.

A.1.3 OAI UE Installation

Figure A.11: OAI UE installation process 1ª interface.

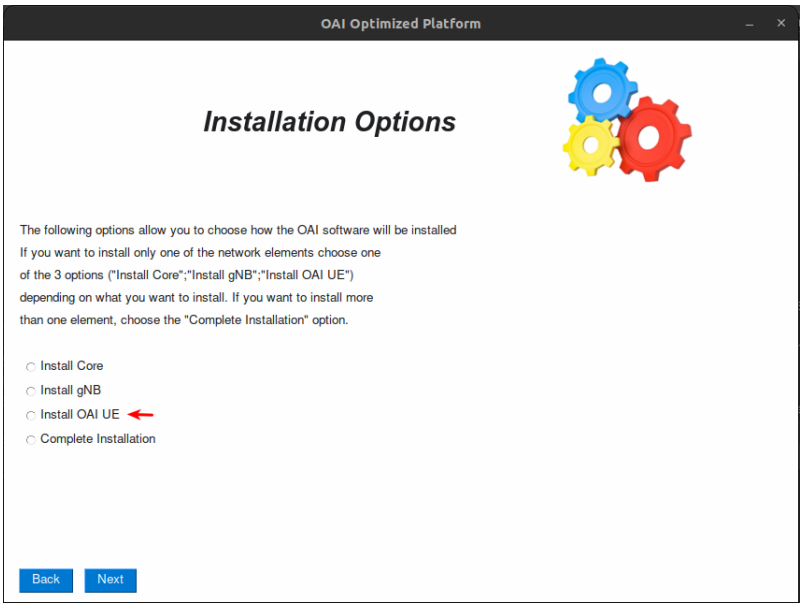

Figure A.12: OAI UE installation process 2ª interface.

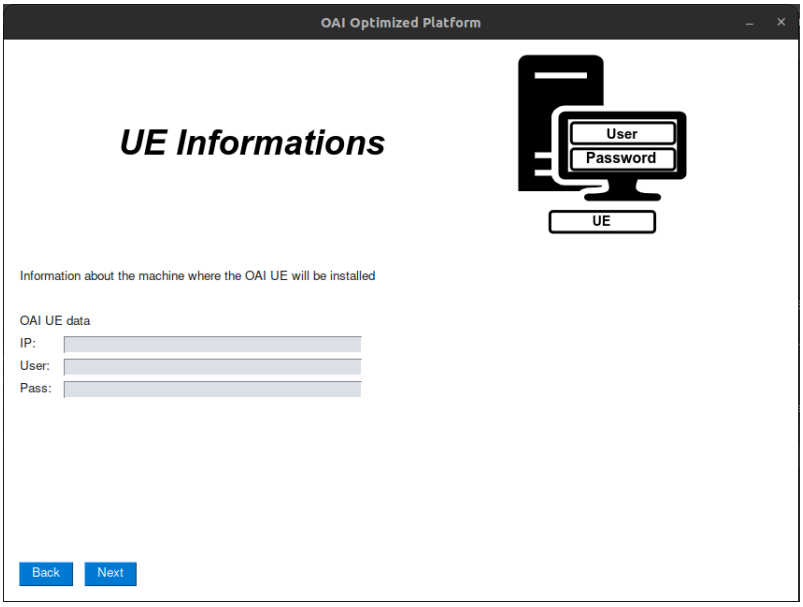

Figure A.13: OAI UE installation process 3ª interface.

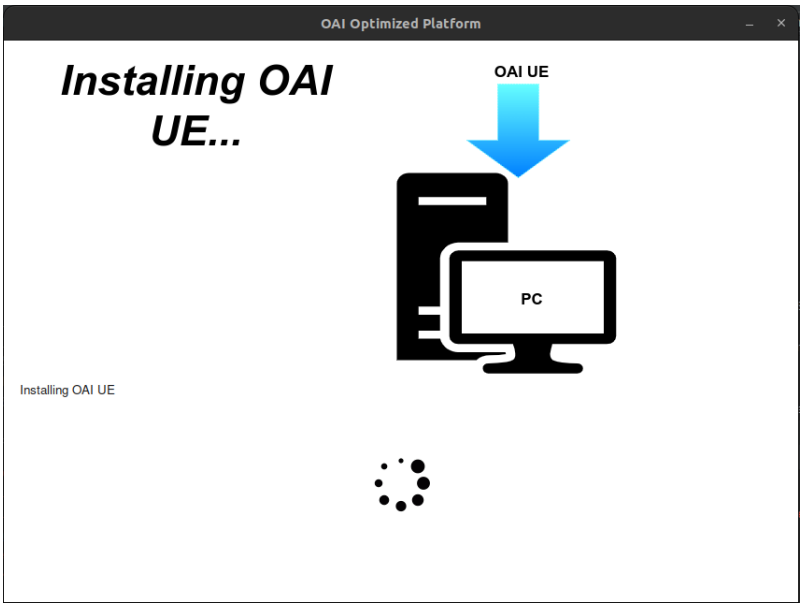

Figure A.14: OAI UE installation process 4ª interface.

Figure A.15: OAI UE installation process 5ª interface.

A.1.4 Install Core and gNB in different hosts with OAI UE

Figure A.16: Core and gNB installation process on different hosts and OAI UE installation process 1ª interface.

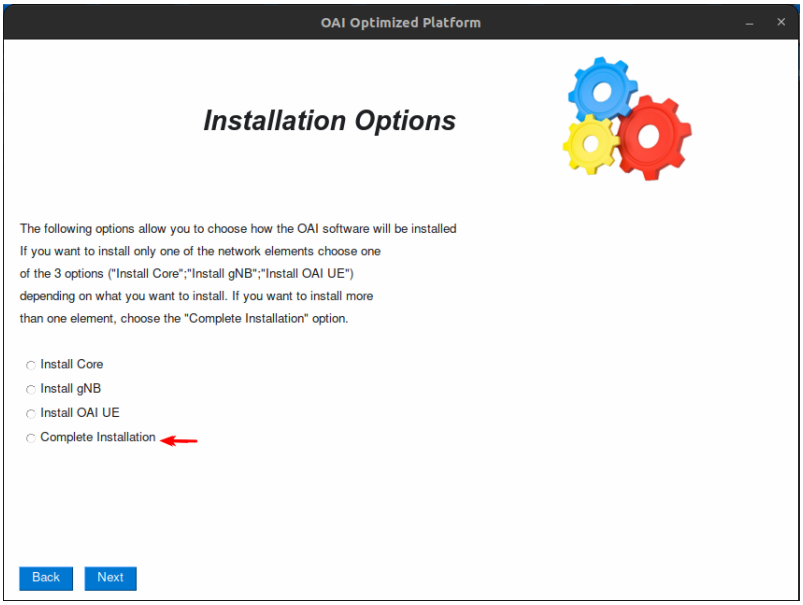

Figure A.17: Core and gNB installation process on different hosts and OAI UE installation process 2ª interface.

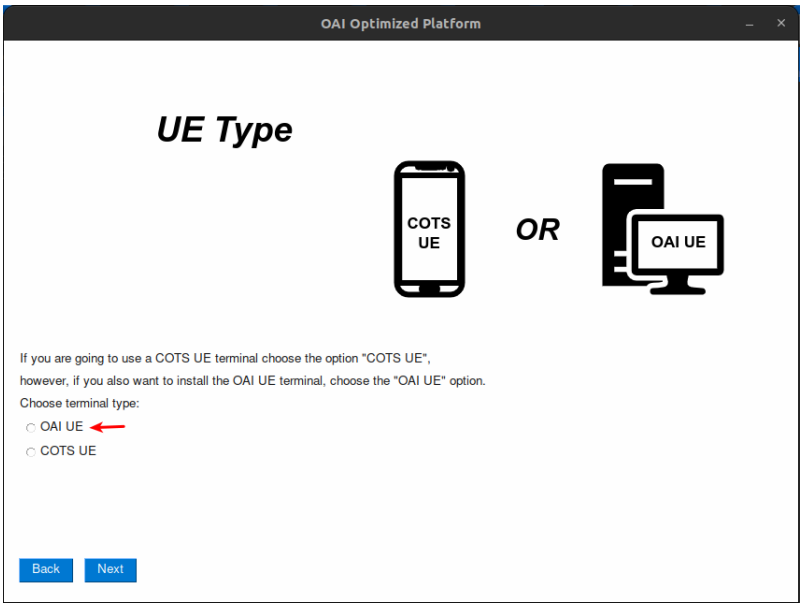

Figure A.18: Core and gNB installation process on different hosts and OAI UE installation process 3ª interface.

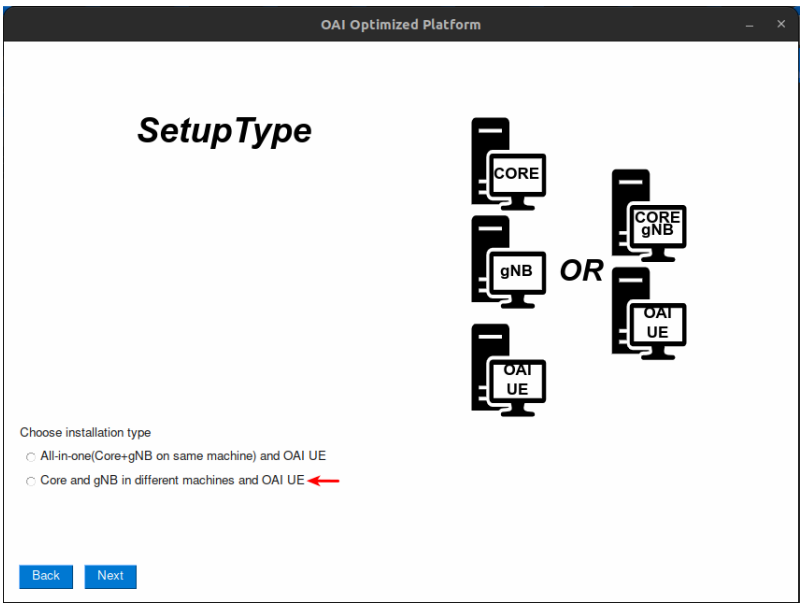

Figure A.19: Core and gNB installation process on different hosts and OAI UE installation process 4ª interface.

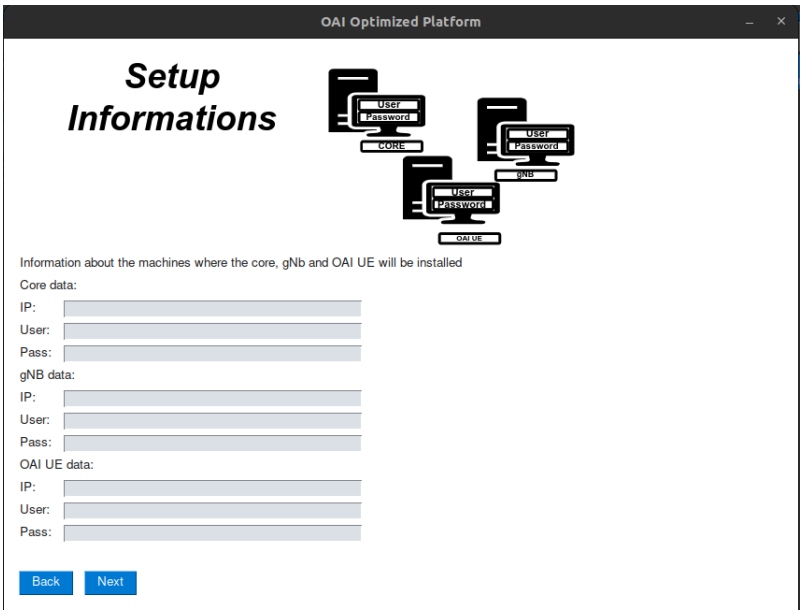

Figure A.20: Core and gNB installation process on different hosts and OAI UE installation process 5ª interface.

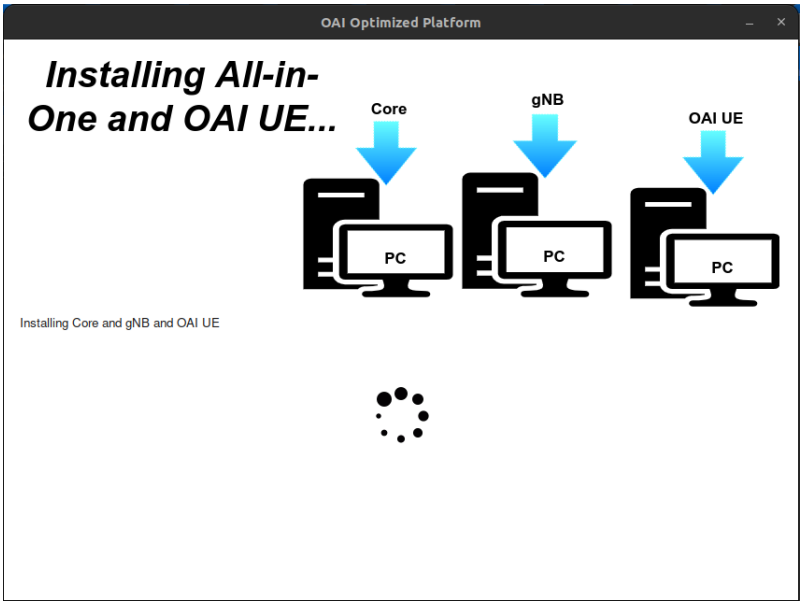

Figure A.21: Core and gNB installation process on different hosts and OAI UE installation process 6ª interface.

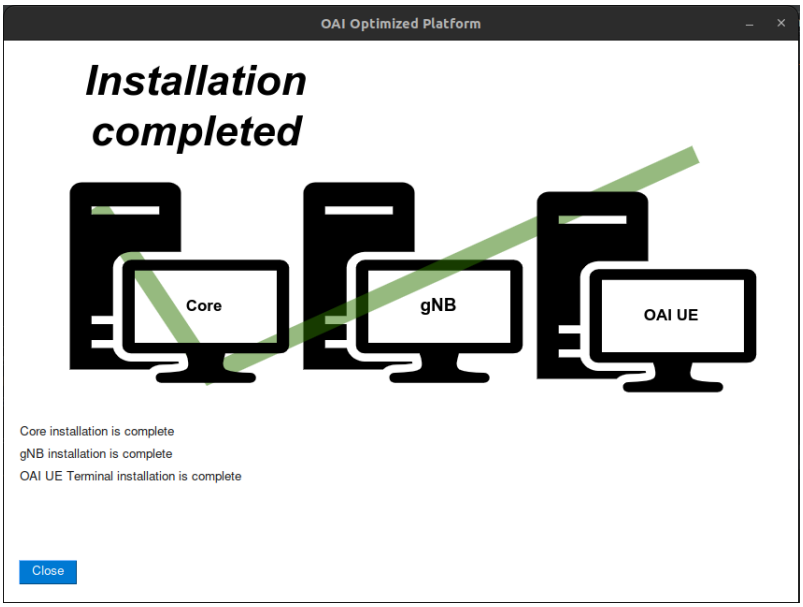

Figure A.22: Core and gNB installation process on different hosts and OAI UE installation process 7ª interface.

A.1.5 Install Core and gNB in different hosts with COTS UE

Figure A.23: Core and gNB installation process on different hosts 1ª interface.

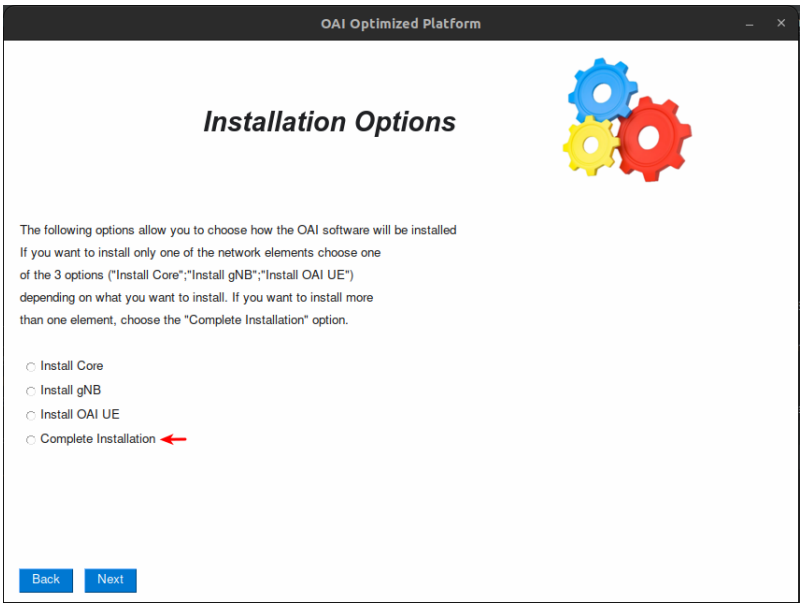

Figure A.24: Core and gNB installation process on different hosts 2ª interface.

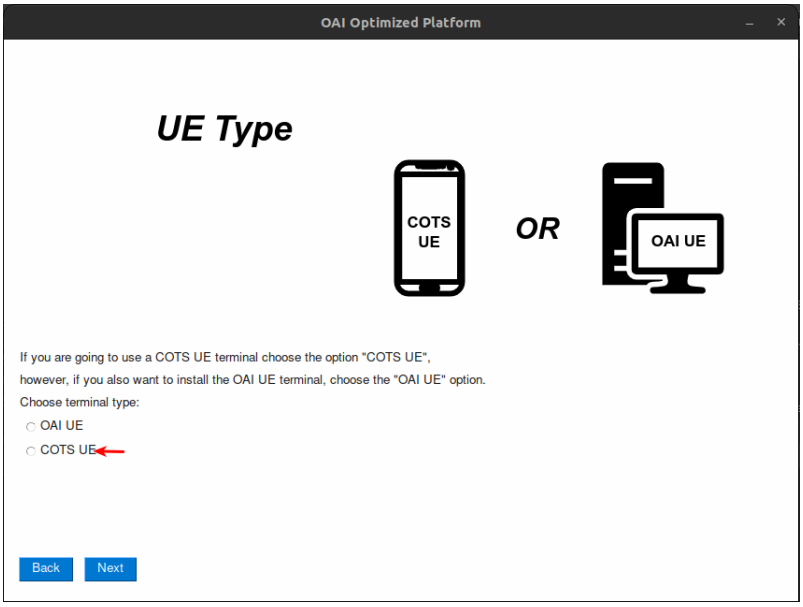

Figure A.25: Core and gNB installation process on different hosts 3ª interface.

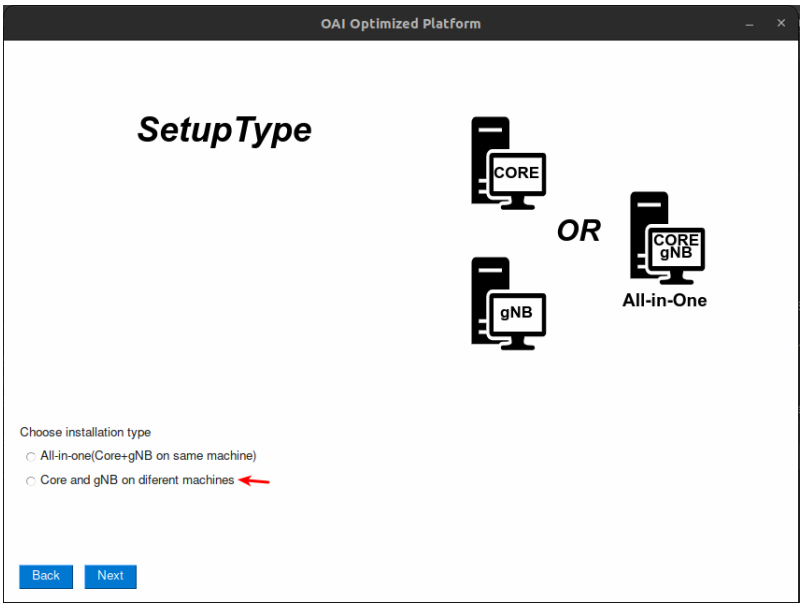

Figure A.26: Core and gNB installation process on different hosts 4ª interface.

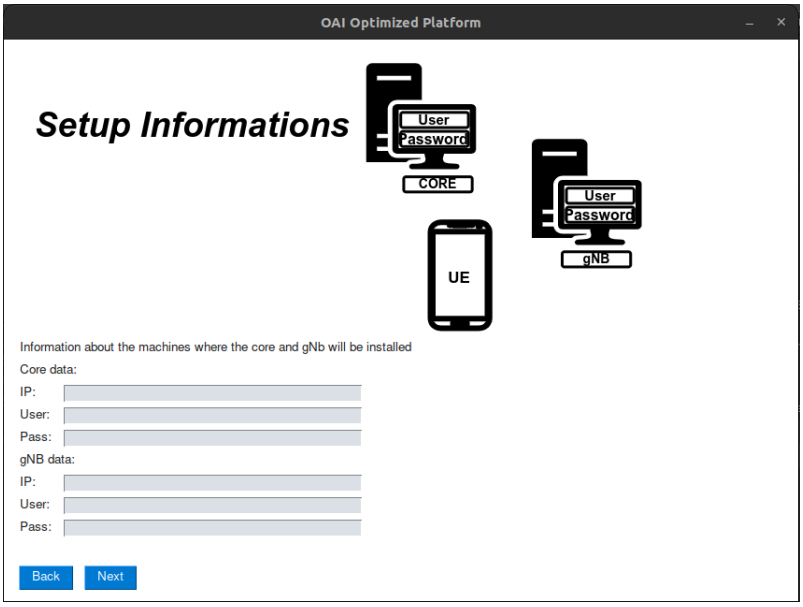

Figure A.27: Core and gNB installation process on different hosts 5ª interface.

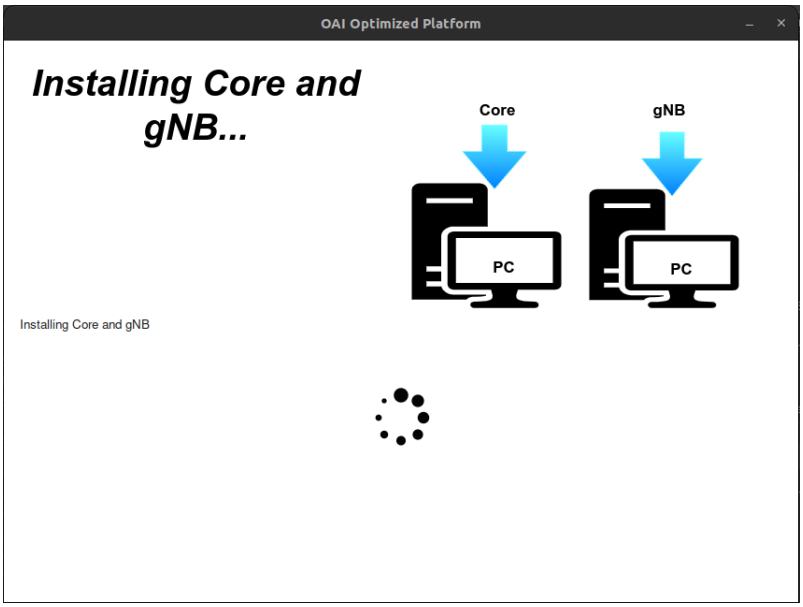

Figure A.28: Core and gNB installation process on different hosts 6ª interface.

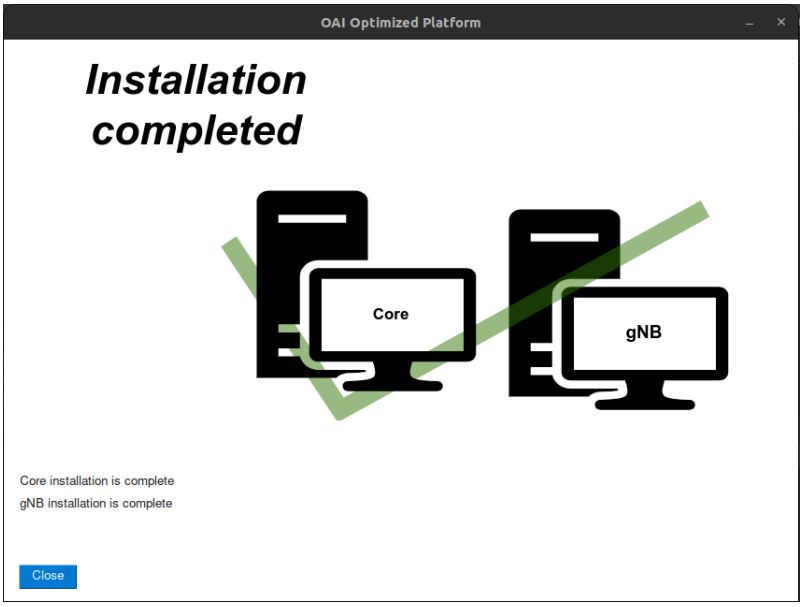

Figure A.29: Core and gNB installation process on different hosts 7ª interface.

A.1.6 Install All-in-One (Core and gNB in same host) with OAI UE

Figure A.30: All-in-One installation process (Core and gNB on the same host) and OAI UE installation process 1ª interface.

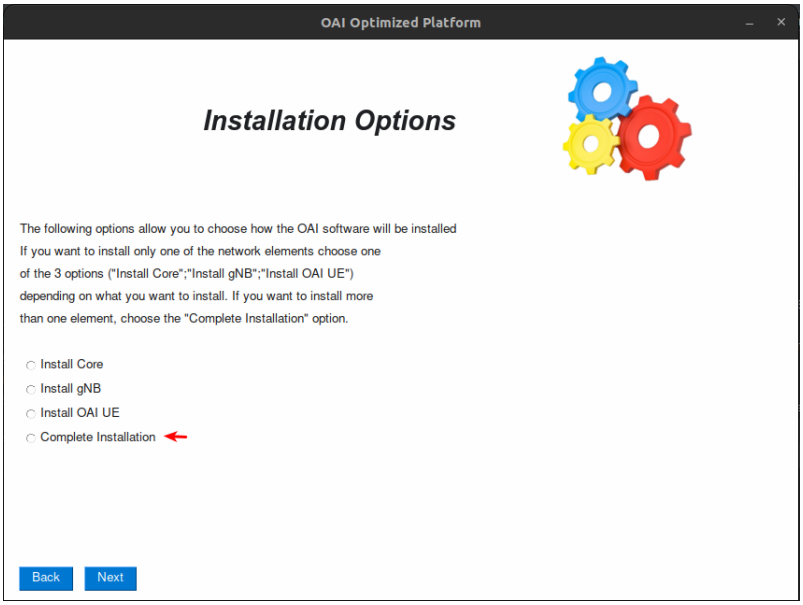

Figure A.31: All-in-One installation process (Core and gNB on the same host) and OAI UE installation process 2ª interface.

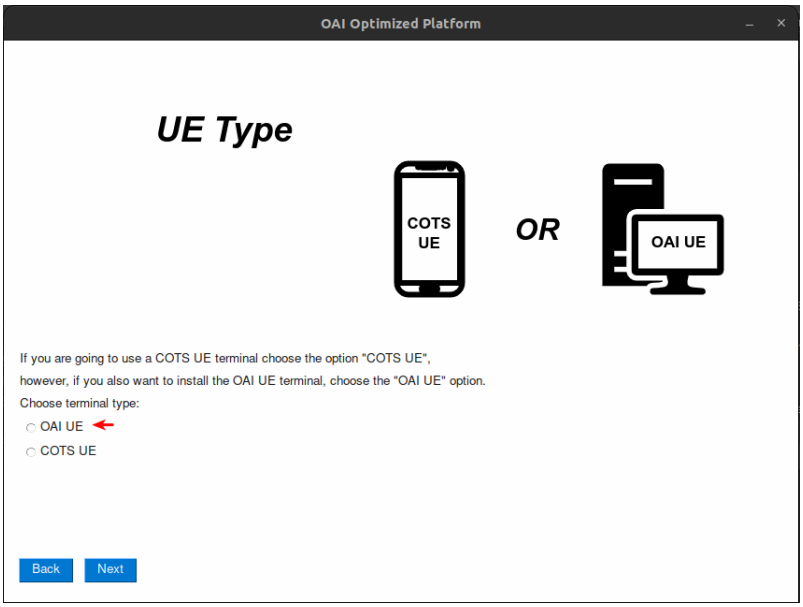

Figure A.32: All-in-One installation process (Core and gNB on the same host) and OAI UE installation process 3ª interface.

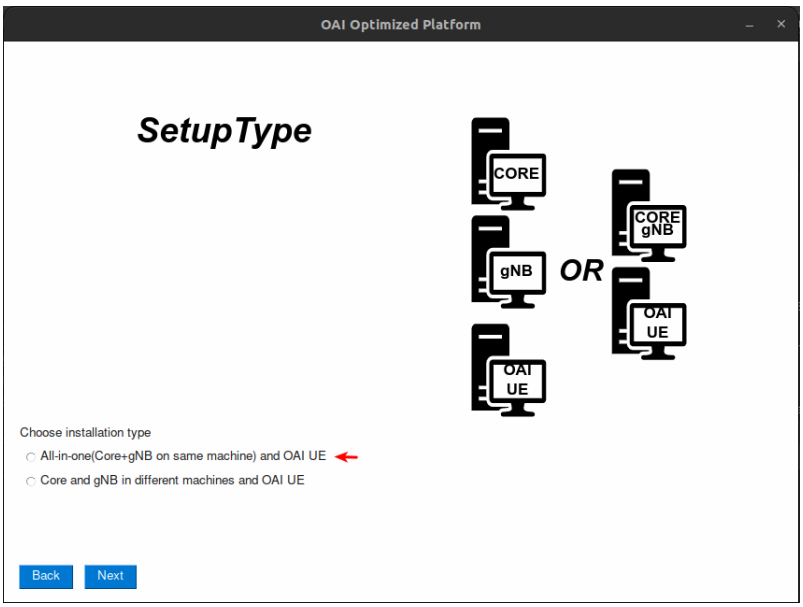

Figure A.33: All-in-One installation process (Core and gNB on the same host) and OAI UE installation process 4ª interface.

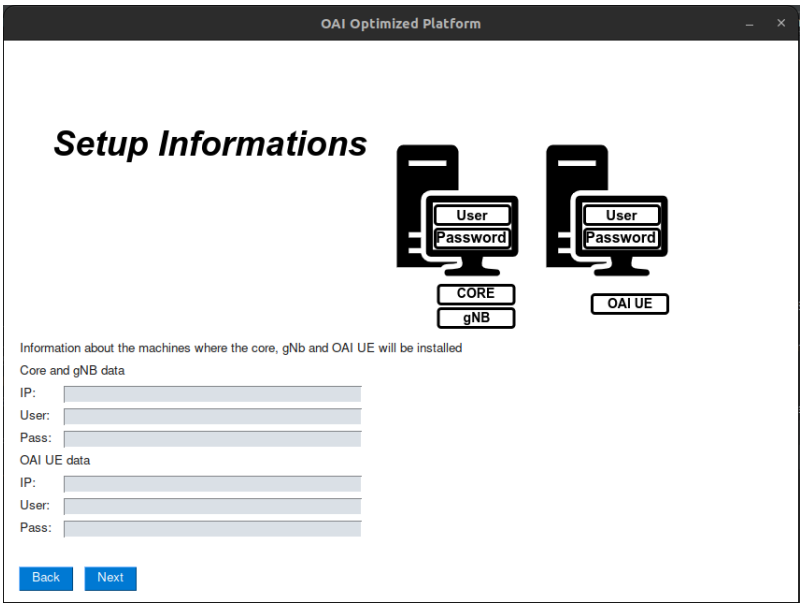

Figure A.34: All-in-One installation process (Core and gNB on the same host) and OAI UE installation process 5ª interface.

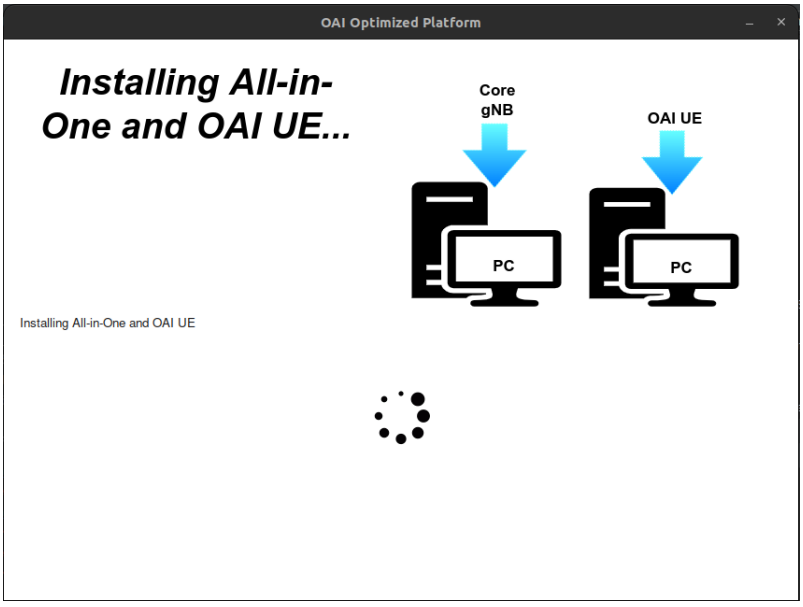

Figure A.35: All-in-One installation process (Core and gNB on the same host) and OAI UE installation process 6ª interface.

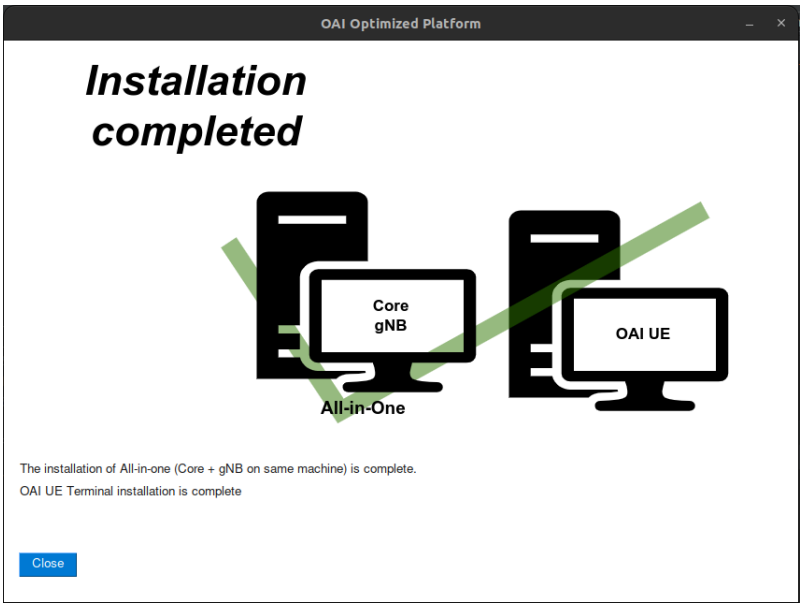

Figure A.36: All-in-One installation process (Core and gNB on the same host) and OAI UE installation process 7ª interface.

A.1.7 Install All-in-One (Core and gNB in same host) with COST UE

Figure A.37: All-in-One installation process (Core and gNB on the same host) 1ª interface.

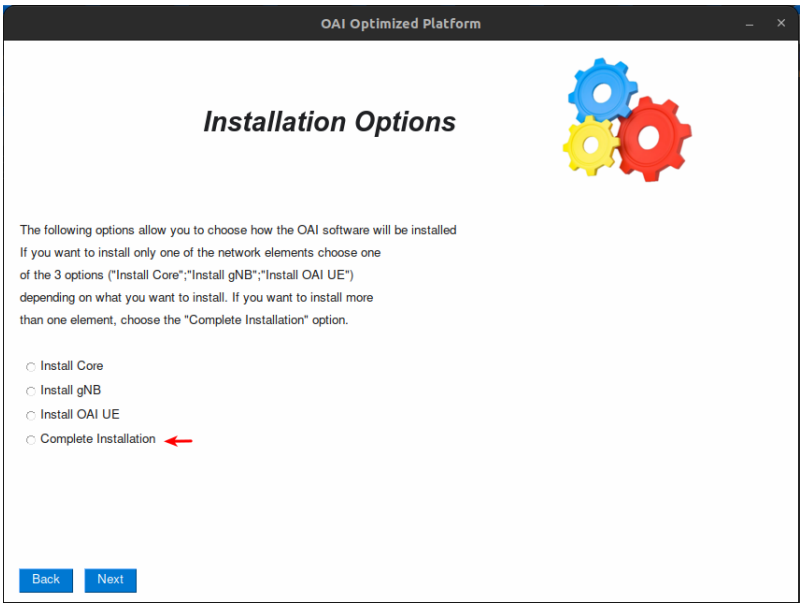

Figure A.38: All-in-One installation process (Core and gNB on the same host) 2ª interface.

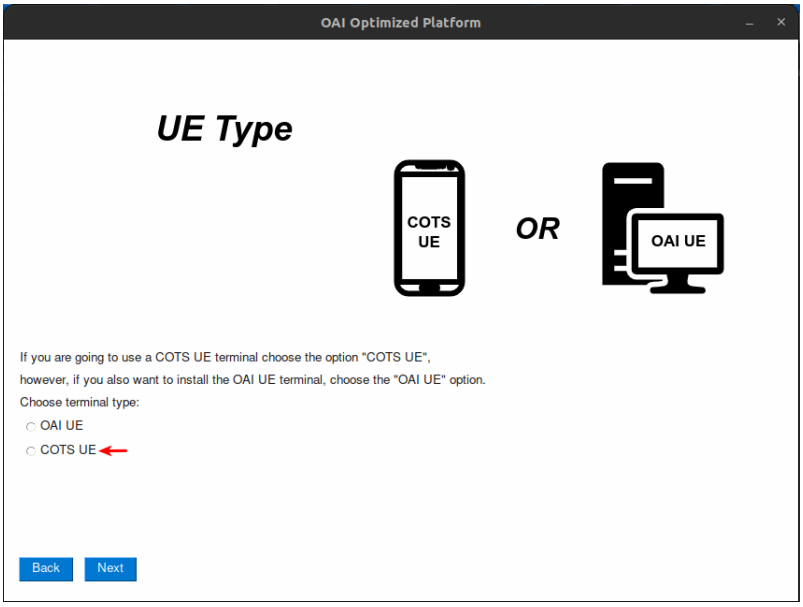

Figure A.39: All-in-One installation process (Core and gNB on the same host) 3ª interface.

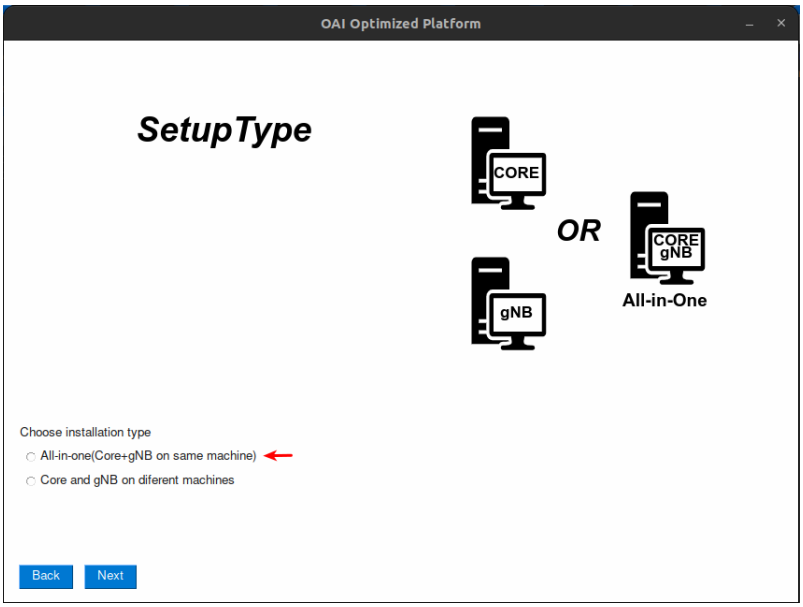

Figure A.40: All-in-One installation process (Core and gNB on the same host) 4ª interface.

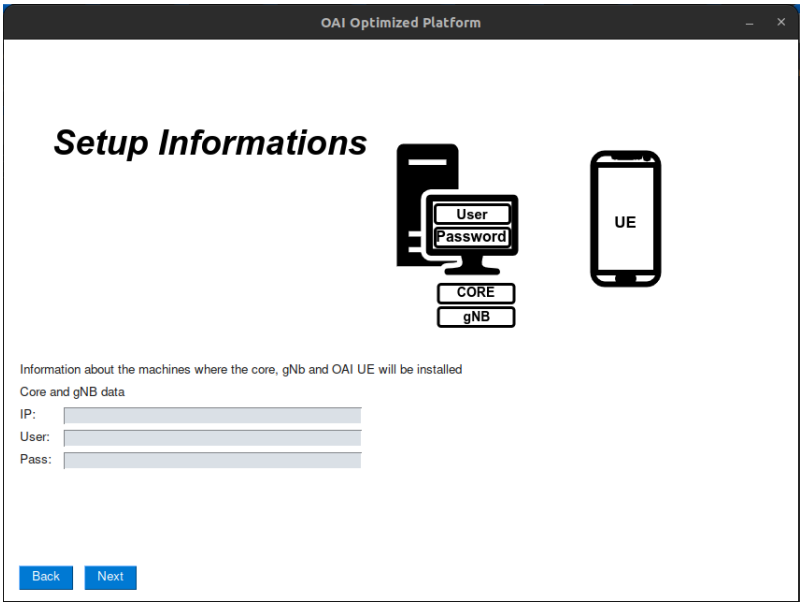

Figure A.41: All-in-One installation process (Core and gNB on the same host) 5ª interface.

Figure A.42: All-in-One installation process (Core and gNB on the same host) 6ª interface.

Figure A.43: All-in-One installation process (Core and gNB on the same host) 7ª interface.

A.2 Setup and execution features

A.2.1 Configure and run setup with core and gnb on different hosts with OAI UE (Express mode)

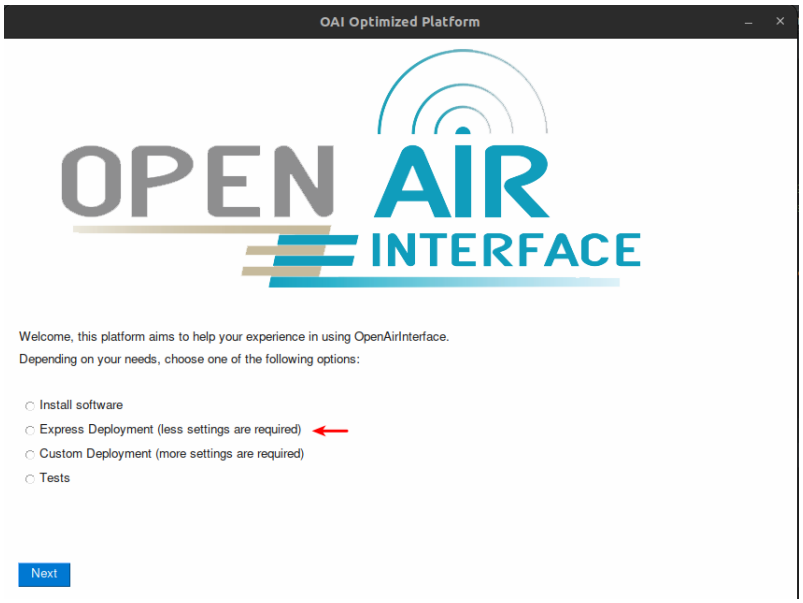

Figure A.44: Configure and run setup with core and gnb on different hosts with OAI UE (Express mode) 1[a] interface.

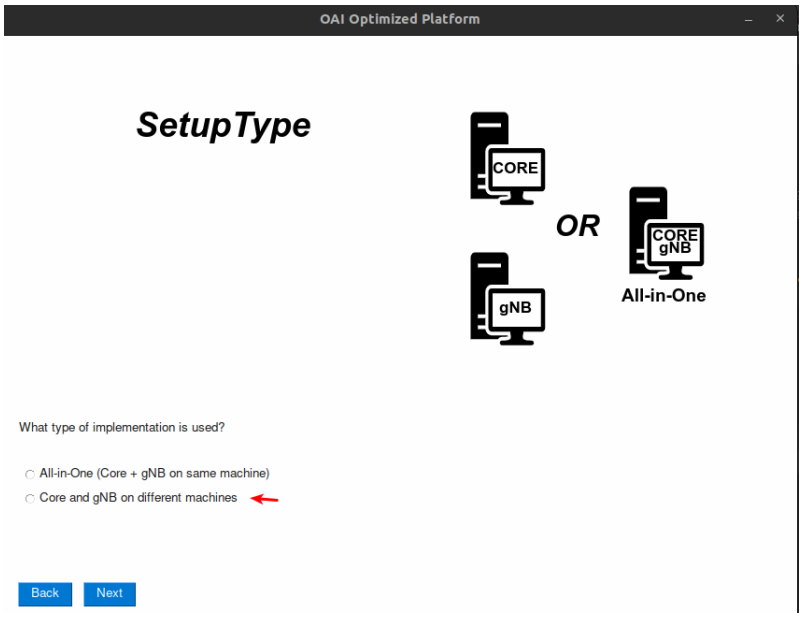

Figure A.45: Configure and run setup with core and gnb on different hosts with OAI UE (Express mode) 2[a] interface.

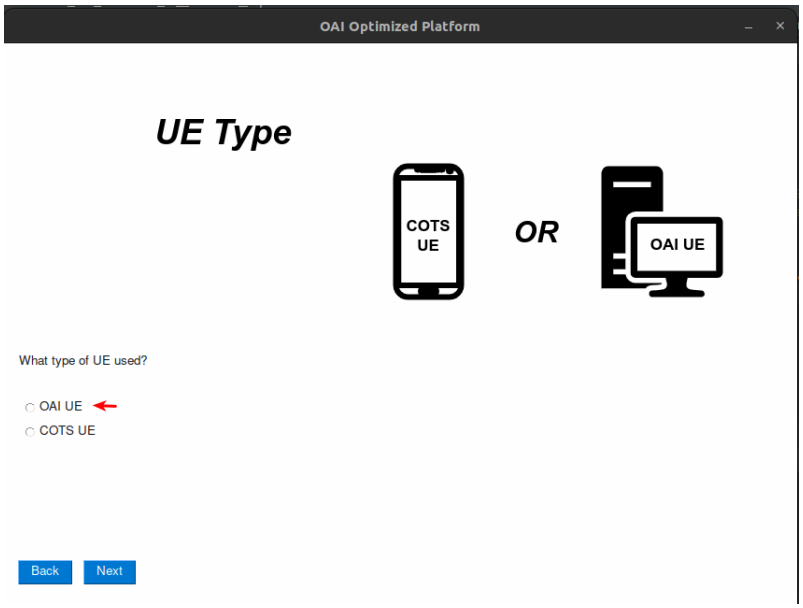

Figure A.46: Configure and run setup with core and gnb on different hosts with OAI UE (Express mode) 3ª interface.

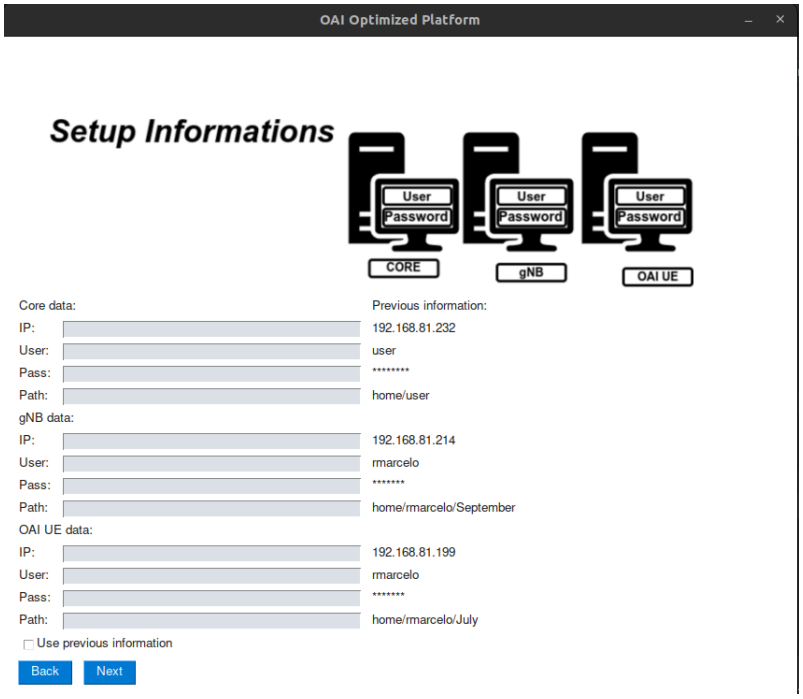

Figure A.47: Configure and run setup with core and gnb on different hosts with OAI UE (Express mode) 4ª interface.

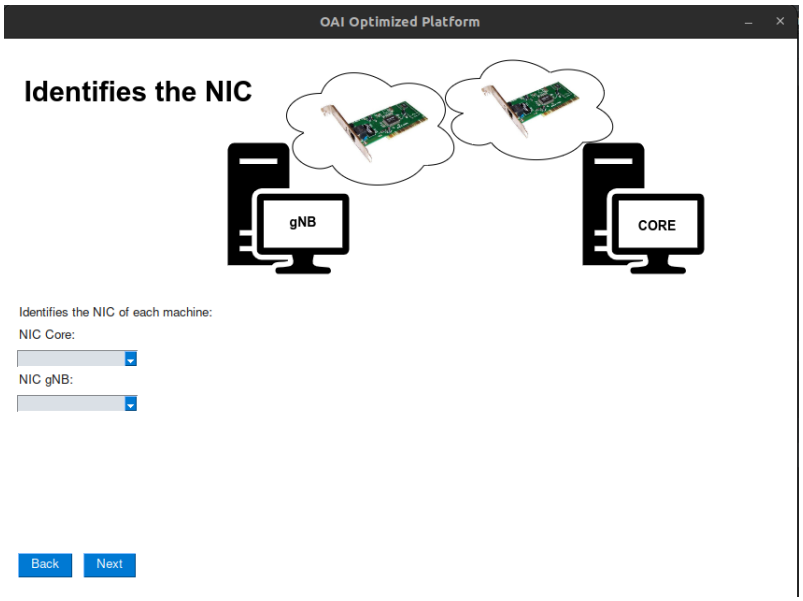

Figure A.48: Configure and run setup with core and gnb on different hosts with OAI UE (Express mode) 5ª interface.

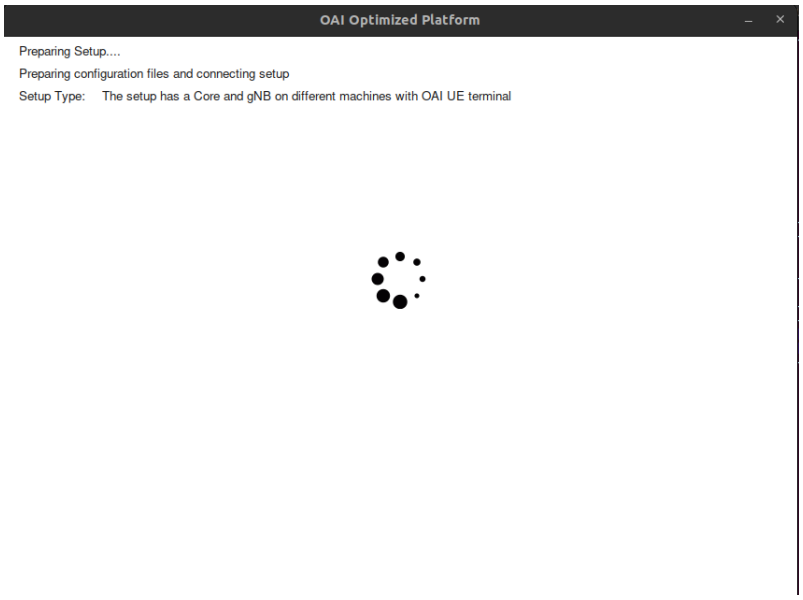

Figure A.49: Configure and run setup with core and gnb on different hosts with OAI UE (Express mode) 6ª interface.

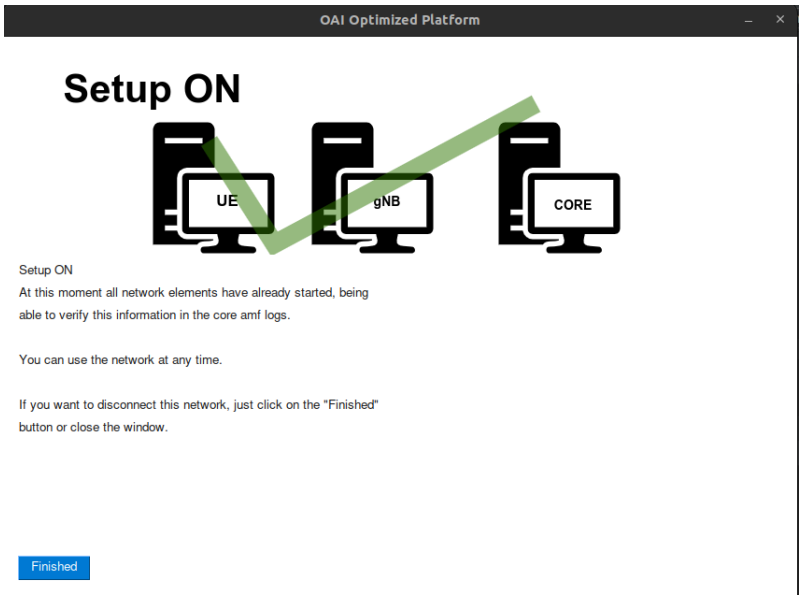

Figure A.50: Configure and run setup with core and gnb on different hosts with OAI UE (Express mode) 7[a] interface.

A.2.2 Configure and run setup with core and gnb on different hosts with COTS UE (Express mode)

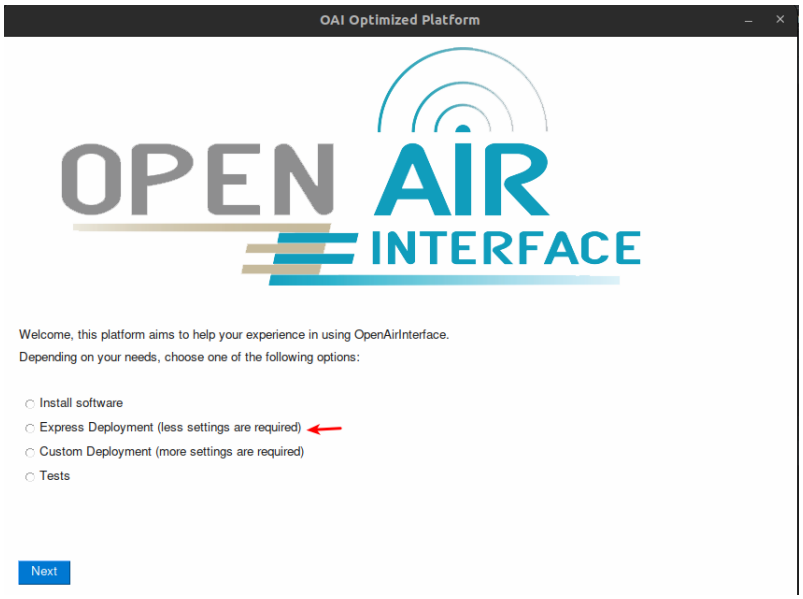

Figure A.51: Configure and run setup with core and gnb on different hosts with COTS UE (Express mode) 1[a] interface.

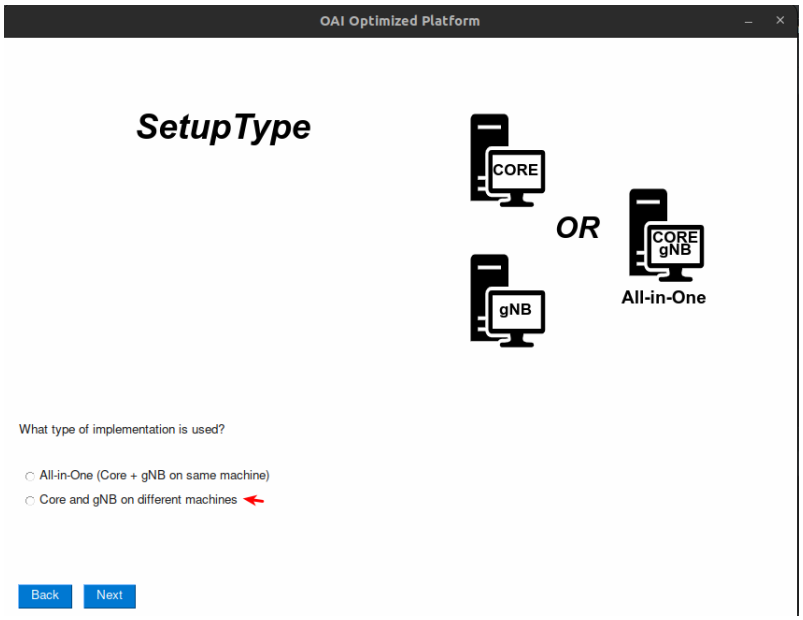

Figure A.52: Configure and run setup with core and gnb on different hosts with COTS UE (Express mode) 2ª interface.

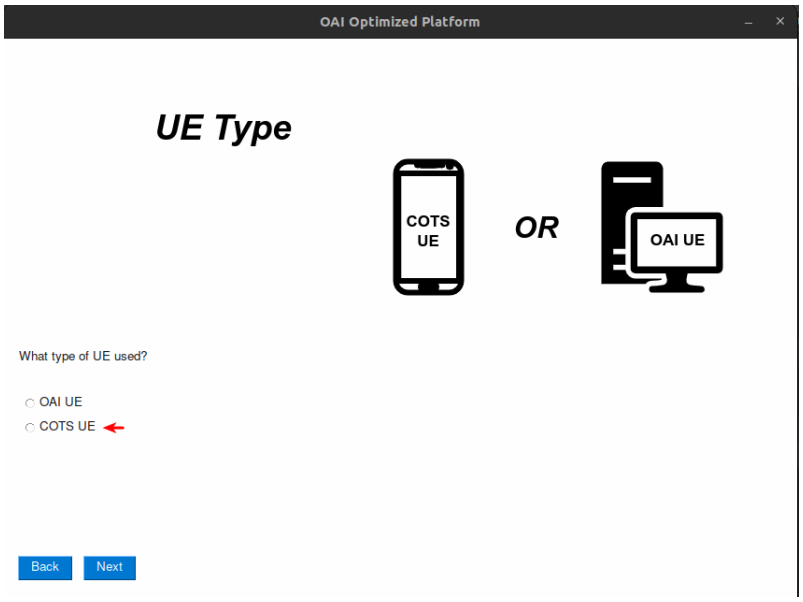

Figure A.53: Configure and run setup with core and gnb on different hosts with COTS UE (Express mode) 3ª interface.

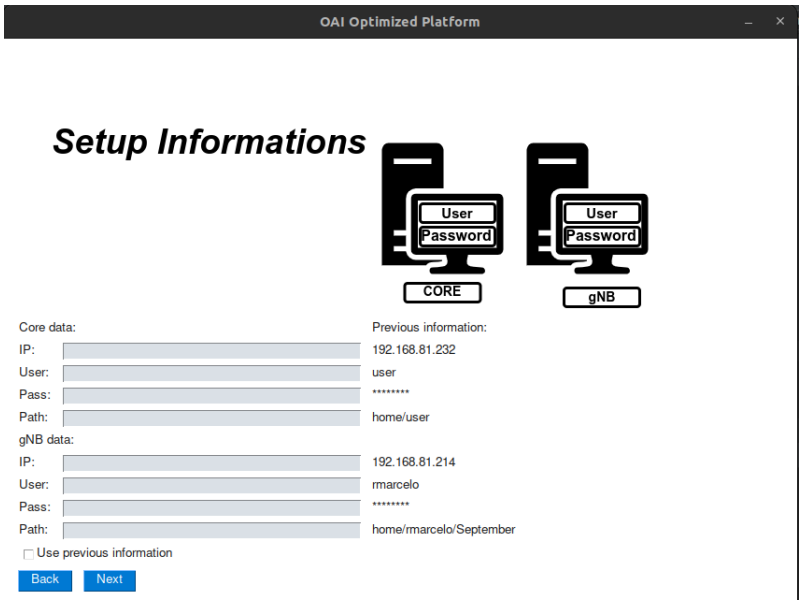

Figure A.54: Configure and run setup with core and gnb on different hosts with COTS UE (Express mode) 4ª interface.

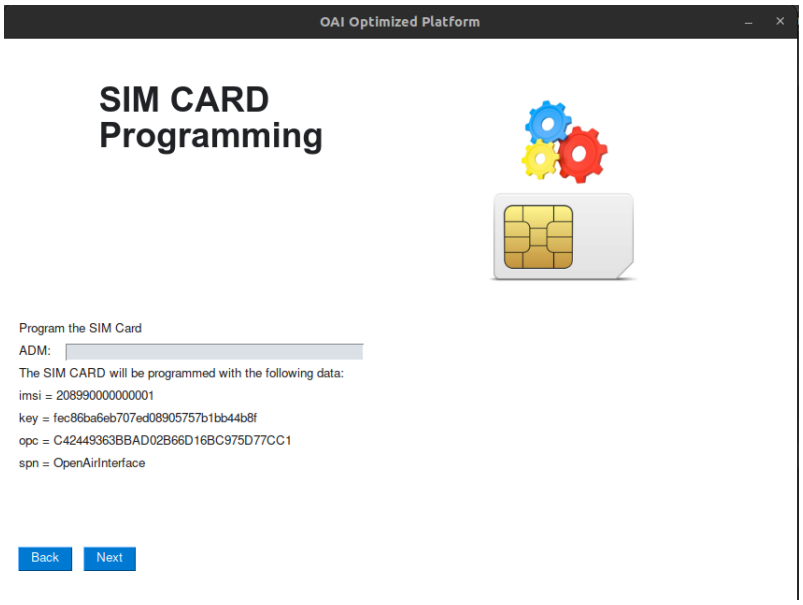

Figure A.55: Configure and run setup with core and gnb on different hosts with COTS UE (Express mode) 5ª interface.

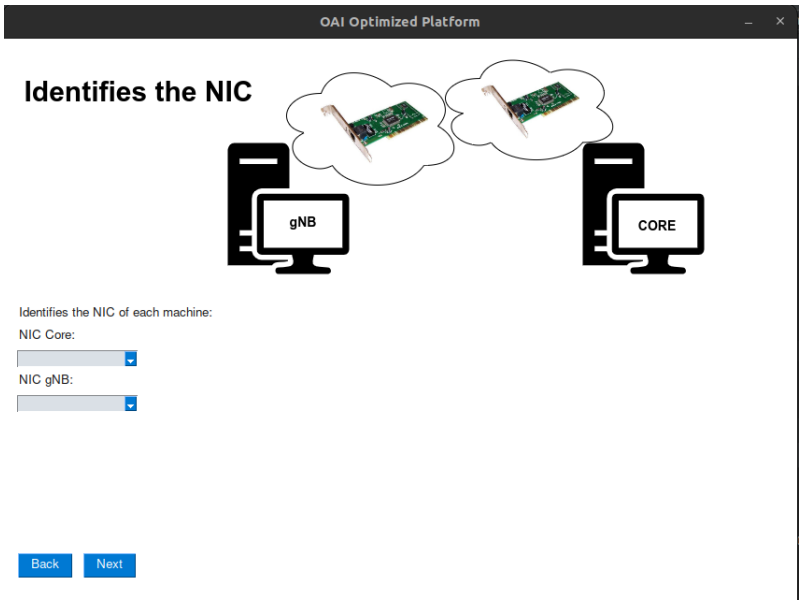

Figure A.56: Configure and run setup with core and gnb on different hosts with COTS UE (Express mode) 6ª interface.

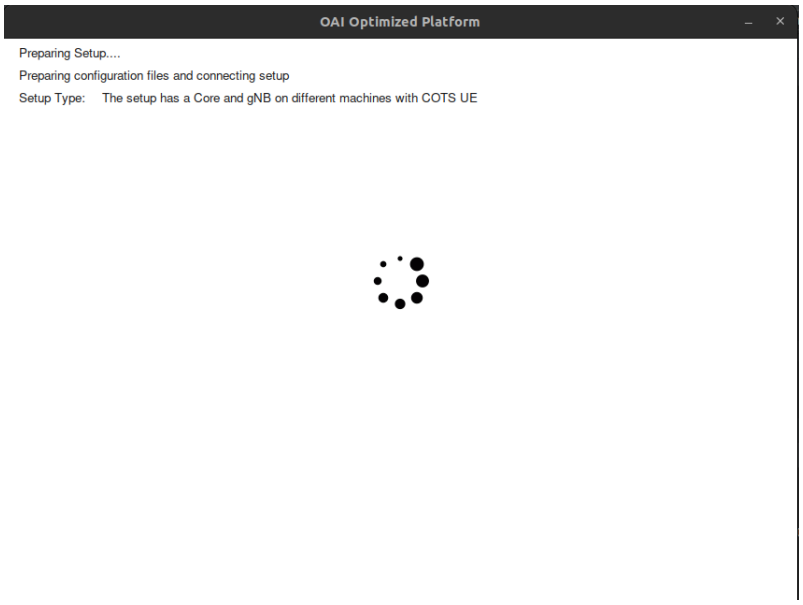

Figure A.57: Configure and run setup with core and gnb on different hosts with COTS UE (Express mode) 7ª interface.

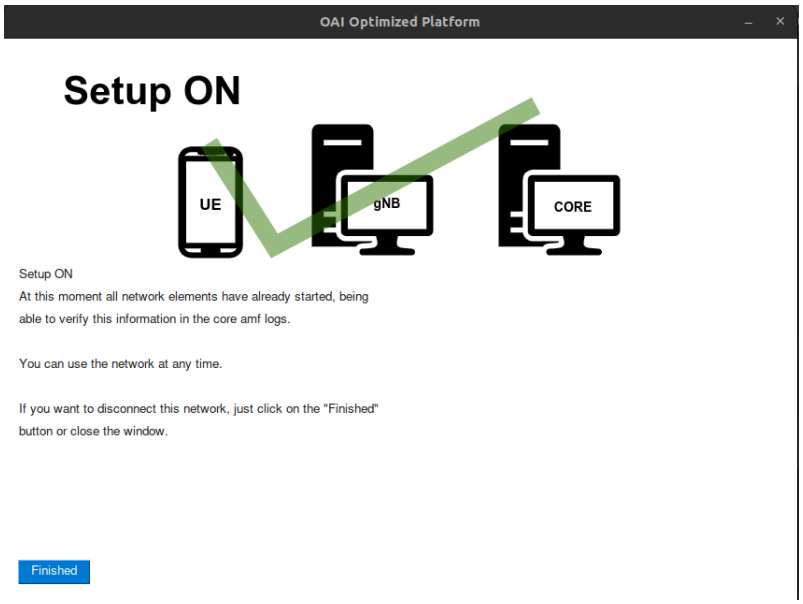

Figure A.58: Configure and run setup with core and gnb on different hosts with COTS UE (Express mode) 8ª interface.

A.2.3 Configure and run setup All-in-One with OAI UE (Express mode)

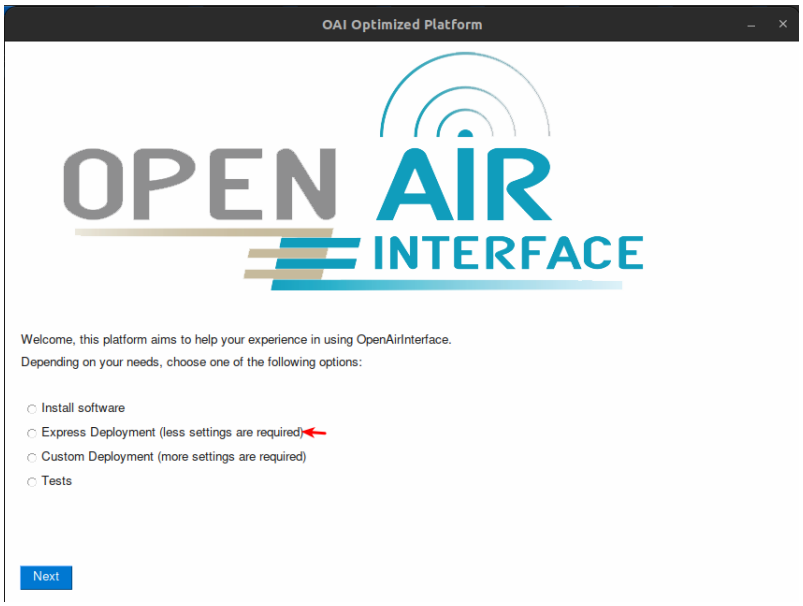

Figure A.59: Configure and run setup All-in-One with OAI UE (Express mode) 1ª interface.

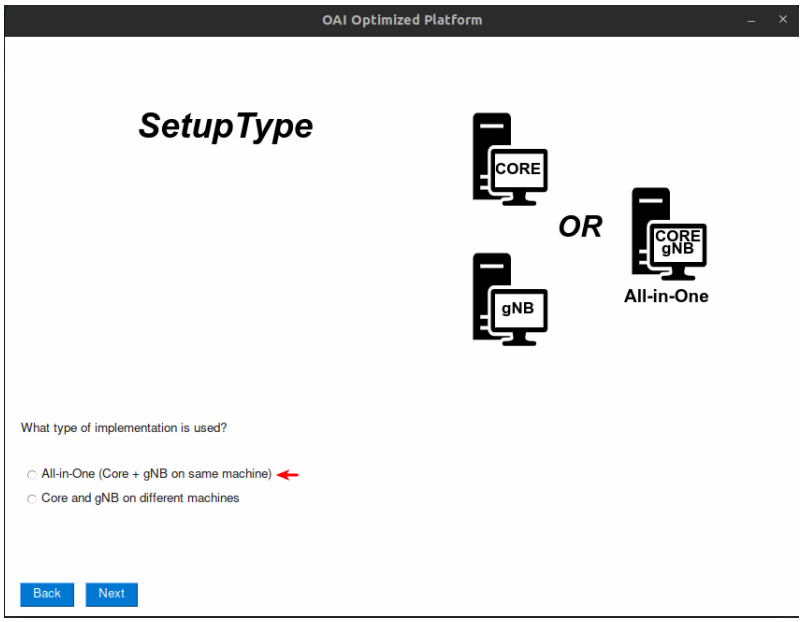

Figure A.60: Configure and run setup All-in-One with OAI UE (Express mode) 2ª interface.

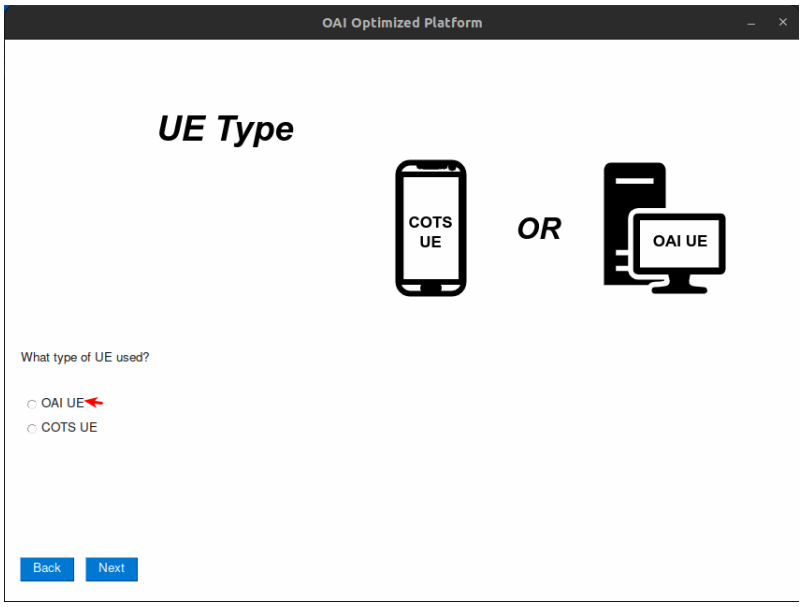

Figure A.61: Configure and run setup All-in-One with OAI UE (Express mode) 3ª interface.

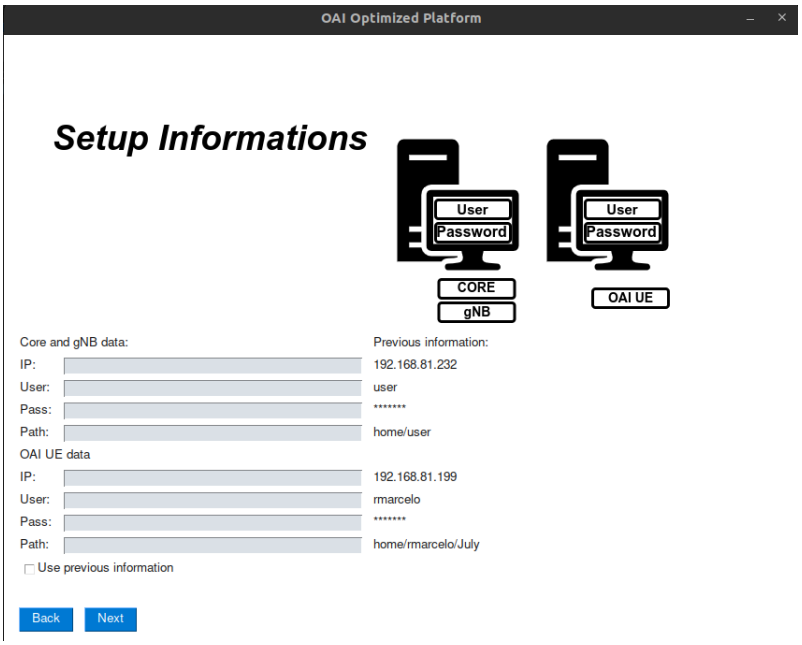

Figure A.62: Configure and run setup All-in-One with OAI UE (Express mode) 4ª interface.

Figure A.63: Configure and run setup All-in-One with OAI UE (Express mode) 5ª interface.

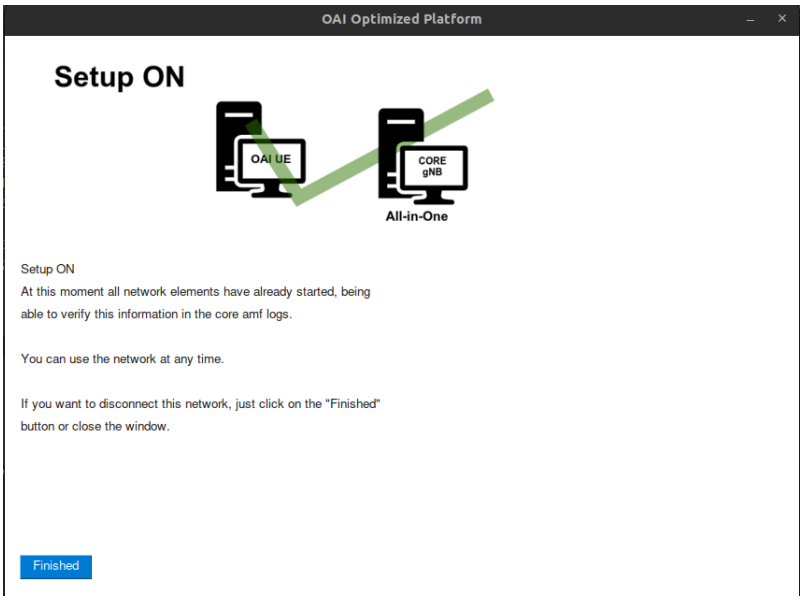

Figure A.64: Configure and run setup All-in-One with OAI UE (Express mode) 6ª interface.

A.2.4 Configure and run setup All-in-One with COTS UE (Express mode)

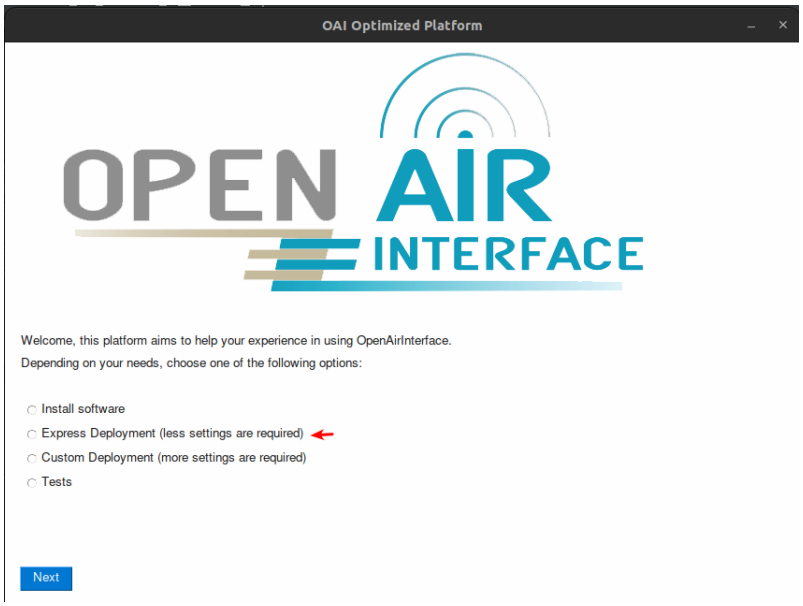

Figure A.65: Configure and run setup All-in-One with COTS UE (Express mode) 1ª interface.

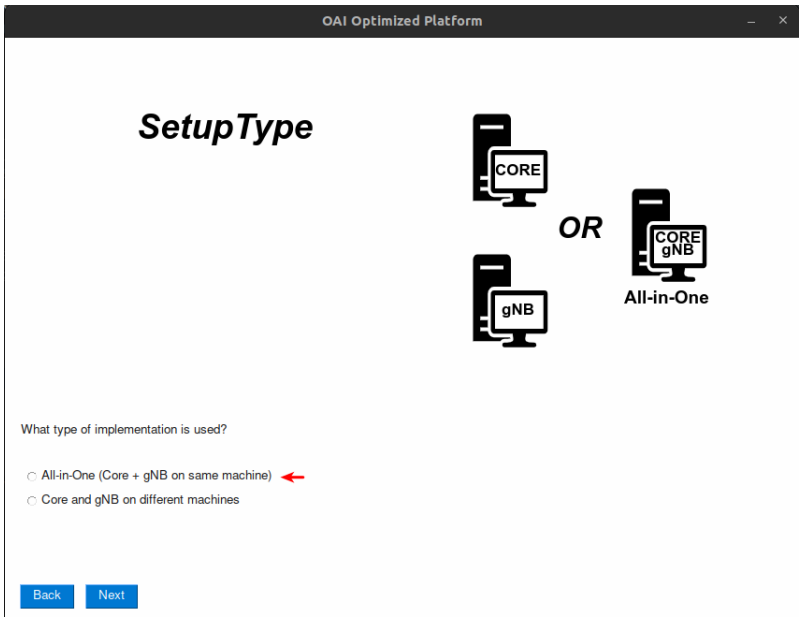

Figure A.66: Configure and run setup All-in-One with COTS UE (Express mode) 2ª interface.

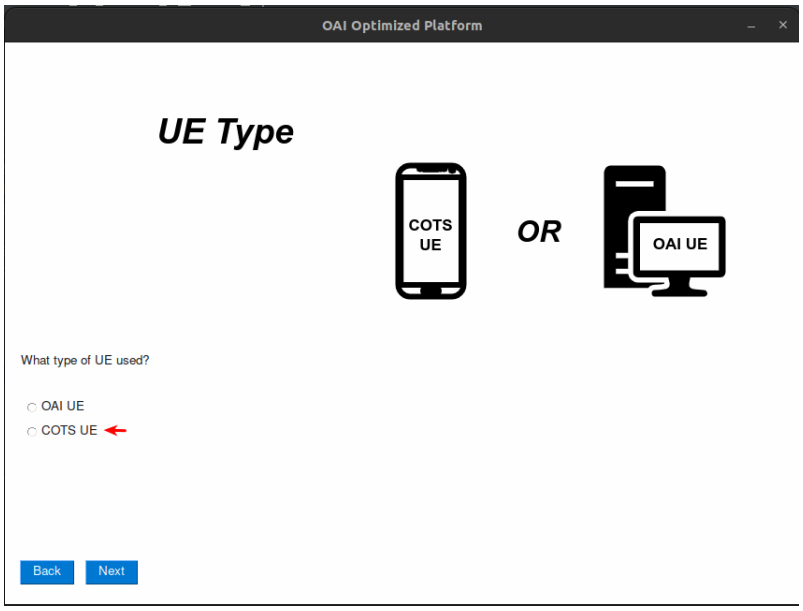

Figure A.67: Configure and run setup All-in-One with COTS UE (Express mode) 3ª interface.

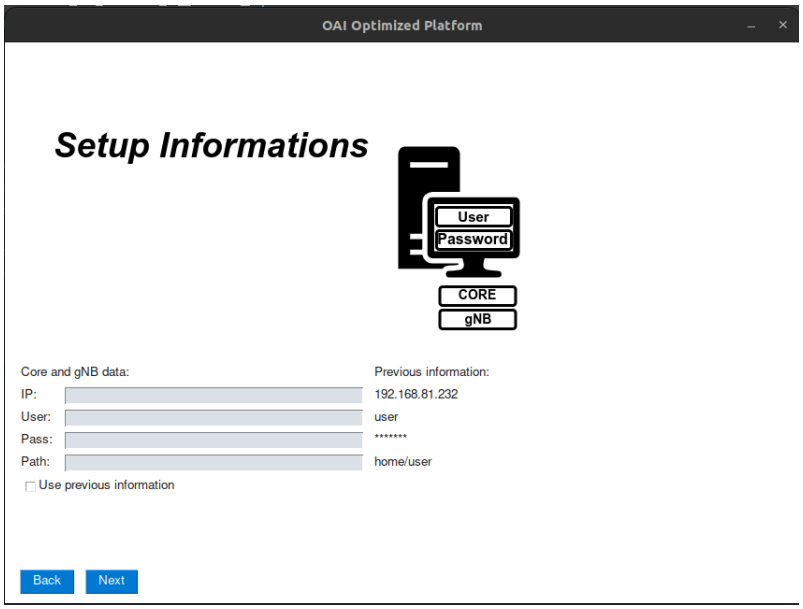

Figure A.68: Configure and run setup All-in-One with COTS UE (Express mode) 4ª interface.

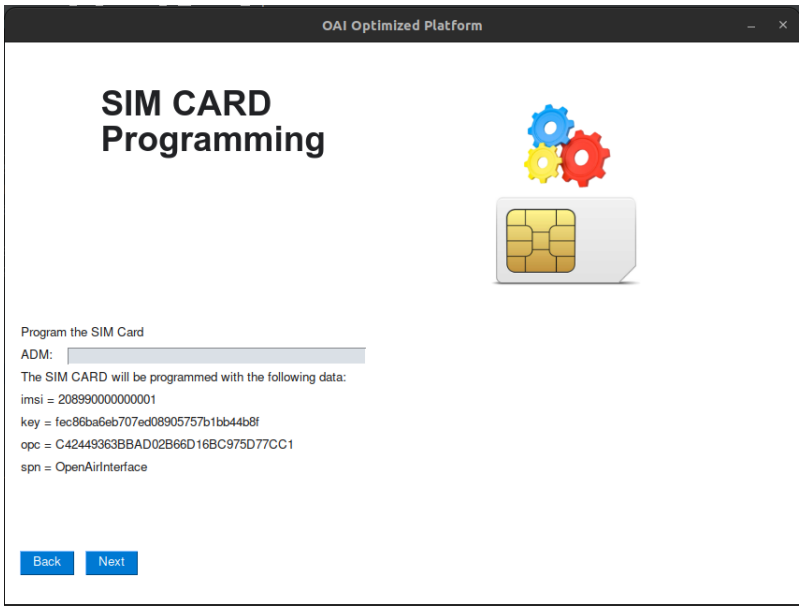

Figure A.69: Configure and run setup All-in-One with COTS UE (Express mode) 5ª interface.

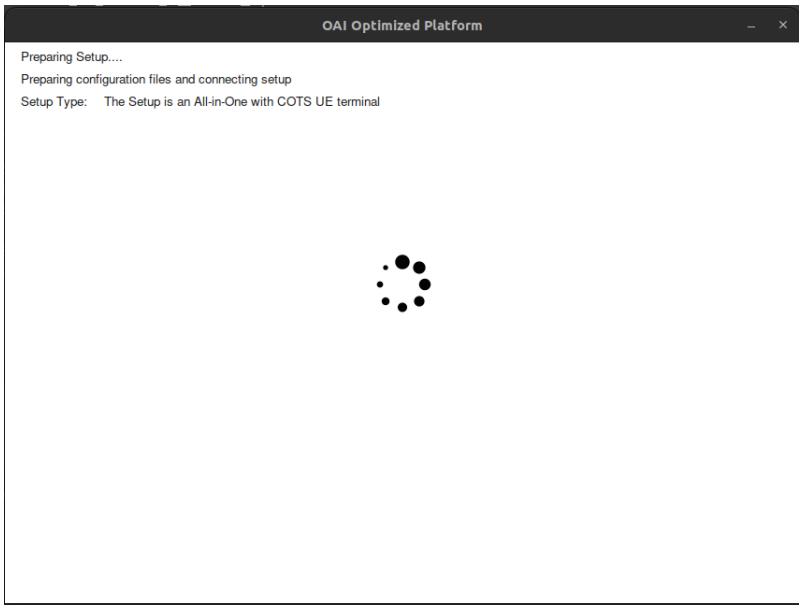

Figure A.70: Configure and run setup All-in-One with COTS UE (Express mode) 6ª interface.

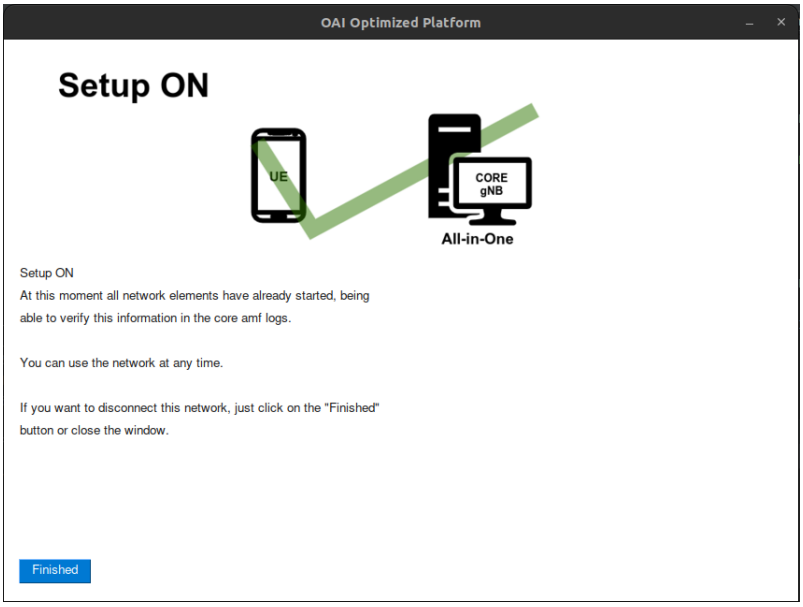

Figure A.71: Configure and run setup All-in-One with COTS UE (Express mode) 7ª interface.

A.2.5 Configure and run setup with core and gnb on different hosts with OAI UE (Custom mode)

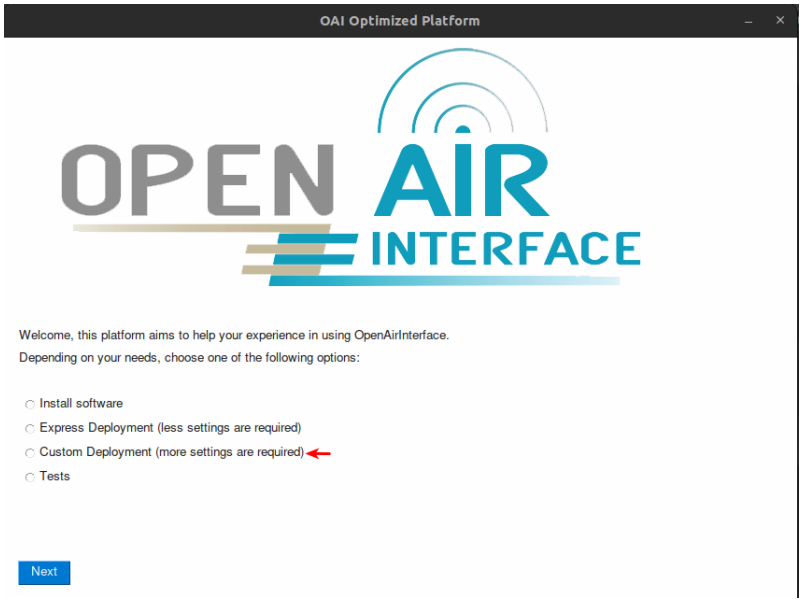

Figure A.72: Configure and run setup with core and gnb on different hosts with OAI UE (Custom mode) 1ª interface.

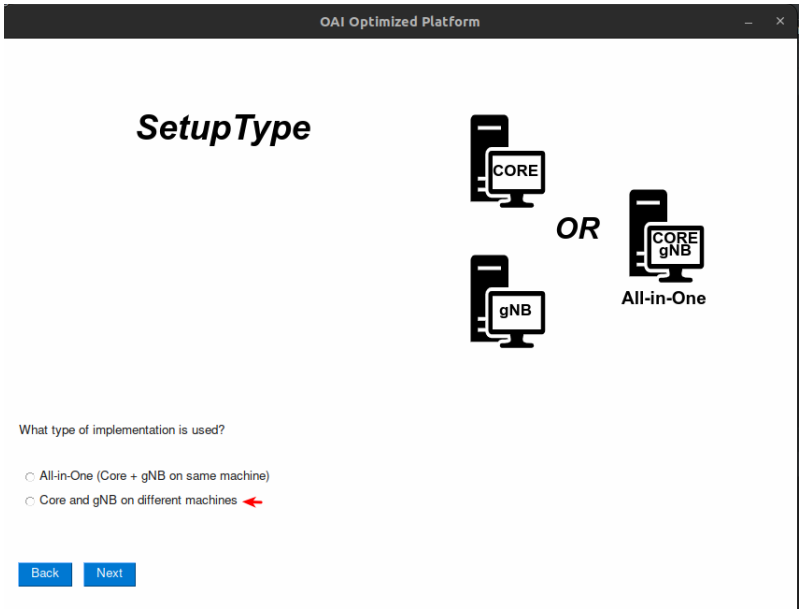

Figure A.73: Configure and run setup with core and gnb on different hosts with OAI UE (Custom mode) 2ª interface.

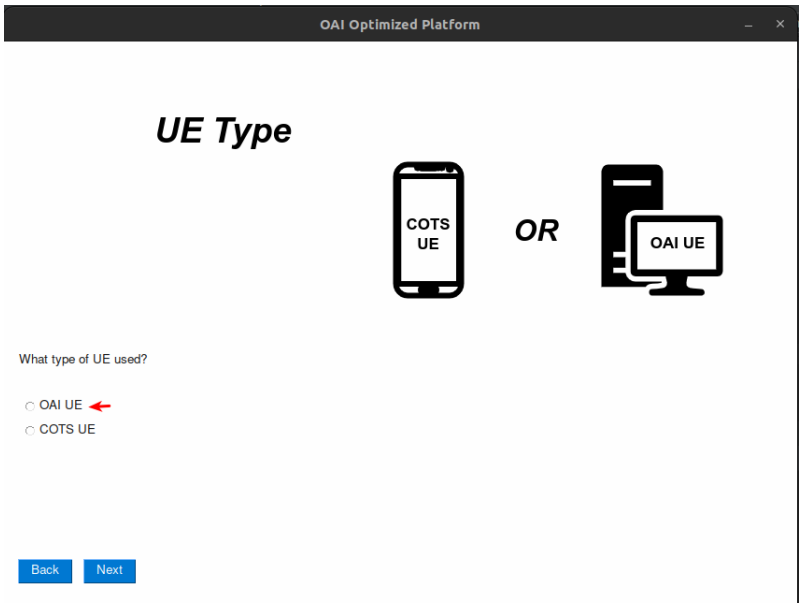

Figure A.74: Configure and run setup with core and gnb on different hosts with OAI UE (Custom mode) 3ª interface.

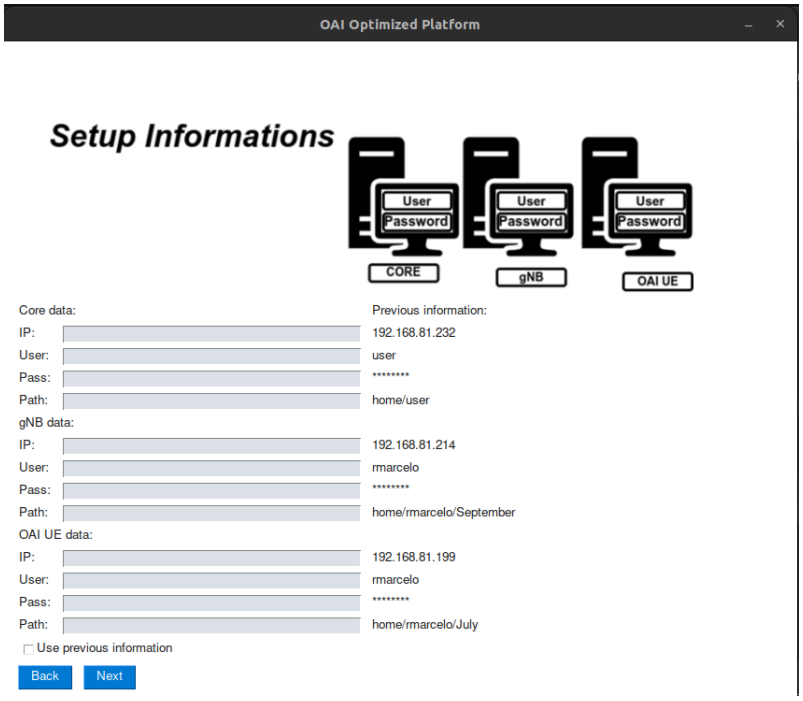

Figure A.75: Configure and run setup with core and gnb on different hosts with OAI UE (Custom mode) 4ª interface.

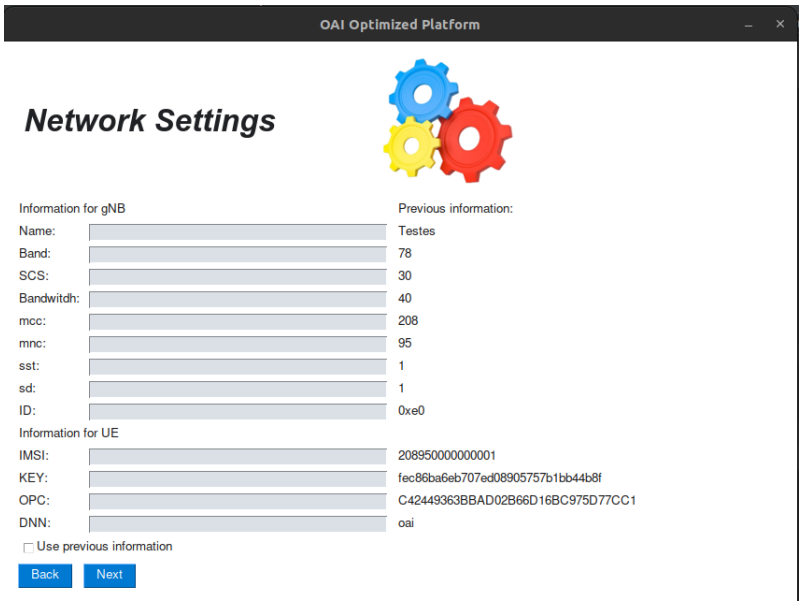

Figure A.76: Configure and run setup with core and gnb on different hosts with OAI UE (Custom mode) 5ª interface.

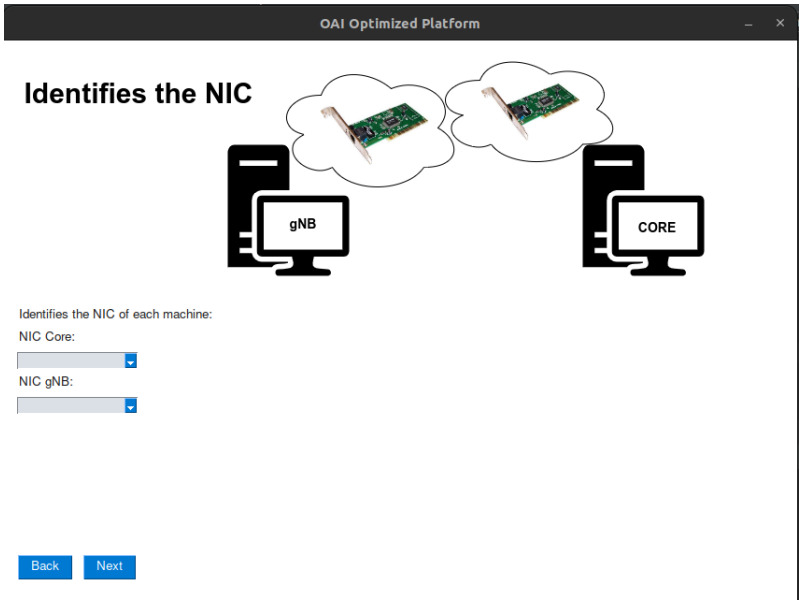

Figure A.77: Configure and run setup with core and gnb on different hosts with OAI UE (Custom mode) 6ª interface.

Figure A.78: Configure and run setup with core and gnb on different hosts with OAI UE (Custom mode) 7ª interface.

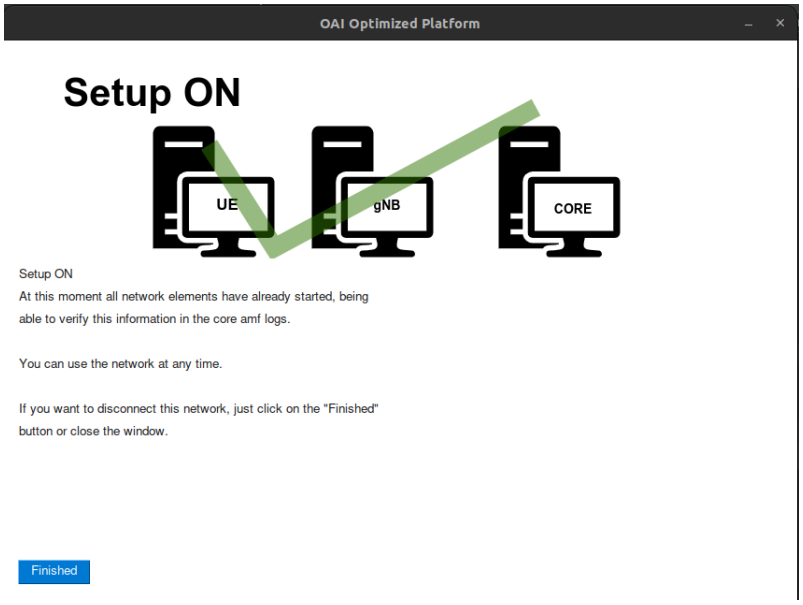

Figure A.79: Configure and run setup with core and gnb on different hosts with OAI UE (Custom mode) 8ª interface.

A.2.6 Configure and run setup with core and gnb on different hosts with COTS UE (Custom mode)

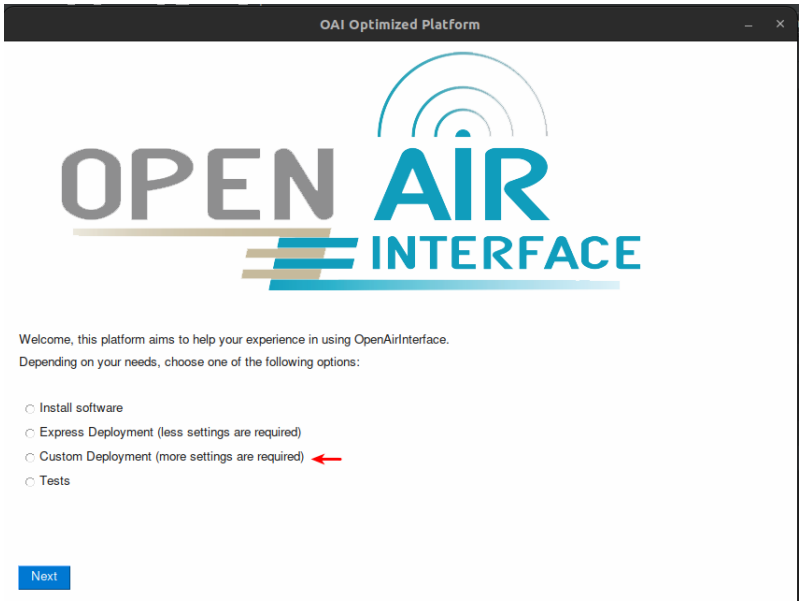

Figure A.80: Configure and run setup with core and gnb on different hosts with COTS UE (Custom mode) 1ª interface.

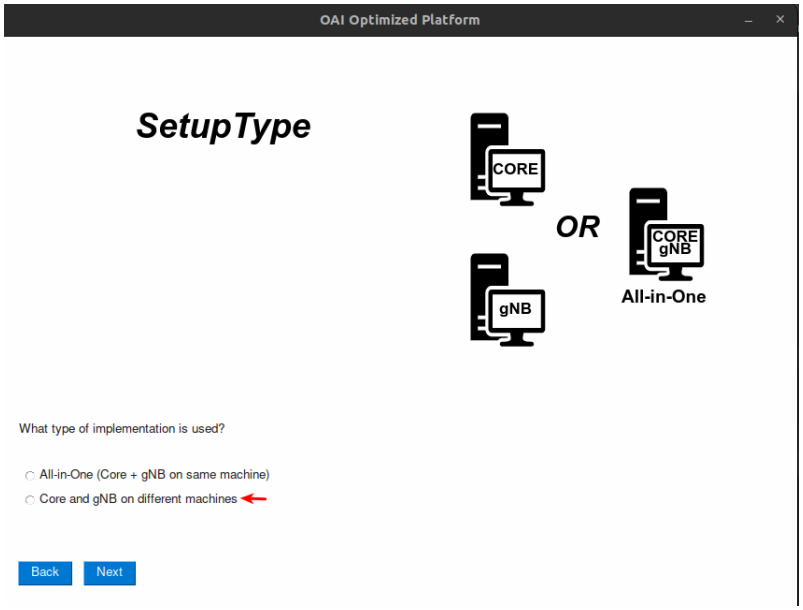

Figure A.81: Configure and run setup with core and gnb on different hosts with COTS UE (Custom mode) 2ª interface.

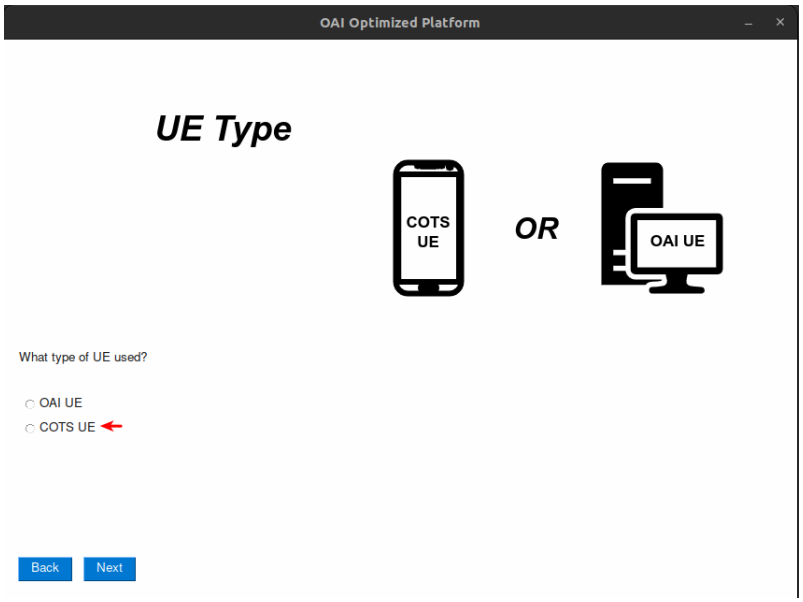

Figure A.82: Configure and run setup with core and gnb on different hosts with COTS UE (Custom mode) 3ª interface.

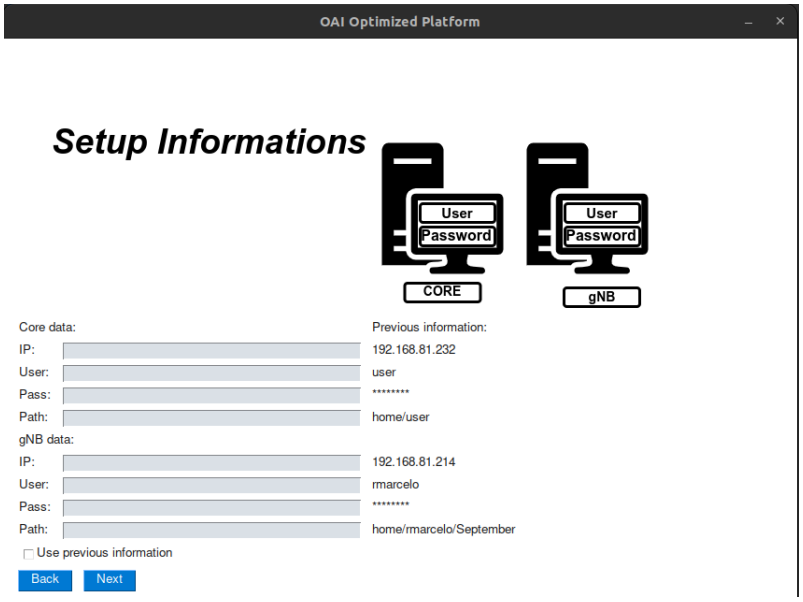

Figure A.83: Configure and run setup with core and gnb on different hosts with COTS UE (Custom mode) 4ª interface.

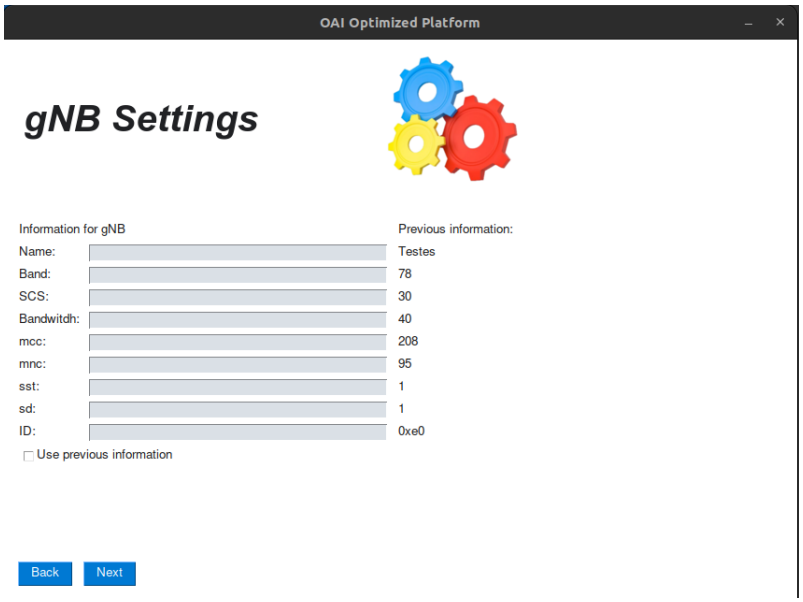

Figure A.84: Configure and run setup with core and gnb on different hosts with COTS UE (Custom mode) 5ª interface.

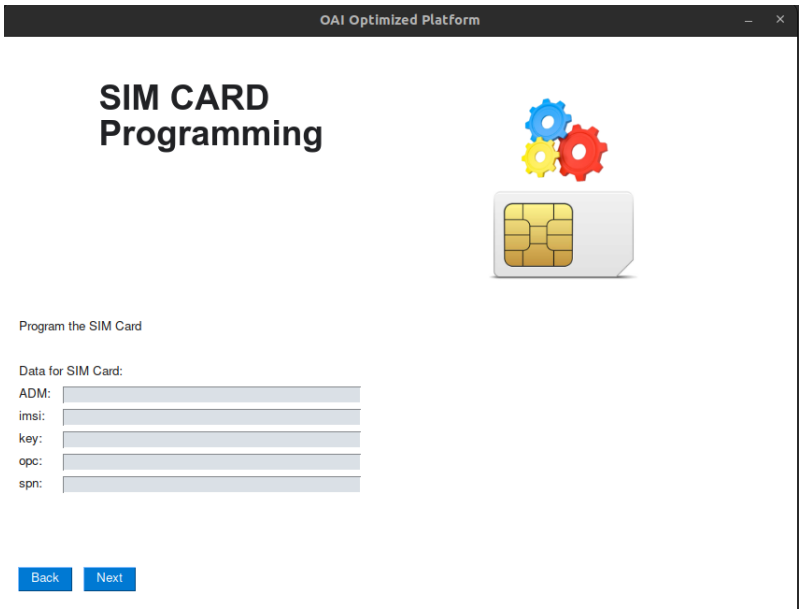

Figure A.85: Configure and run setup with core and gnb on different hosts with COTS UE (Custom mode) 6ª interface.

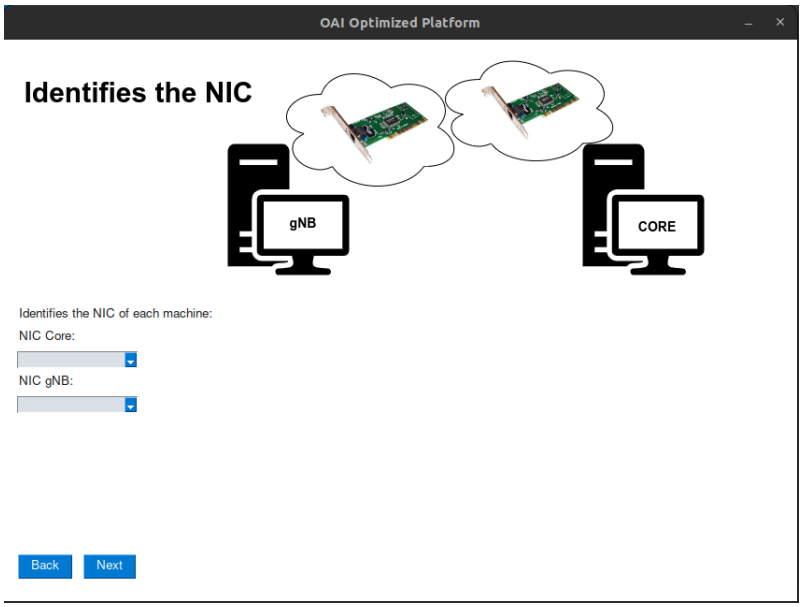

Figure A.86: Configure and run setup with core and gnb on different hosts with COTS UE (Custom mode) 7ª interface.

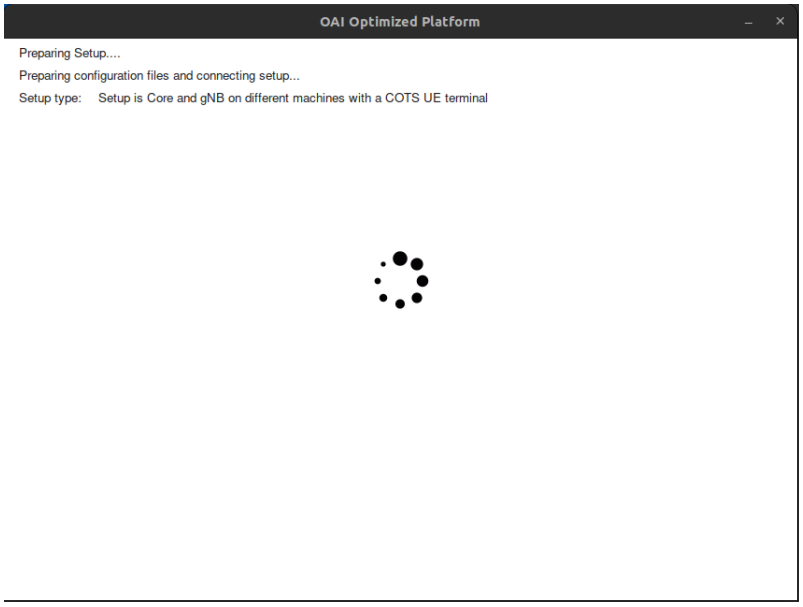

Figure A.87: Configure and run setup with core and gnb on different hosts with COTS UE (Custom mode) 8ª interface.

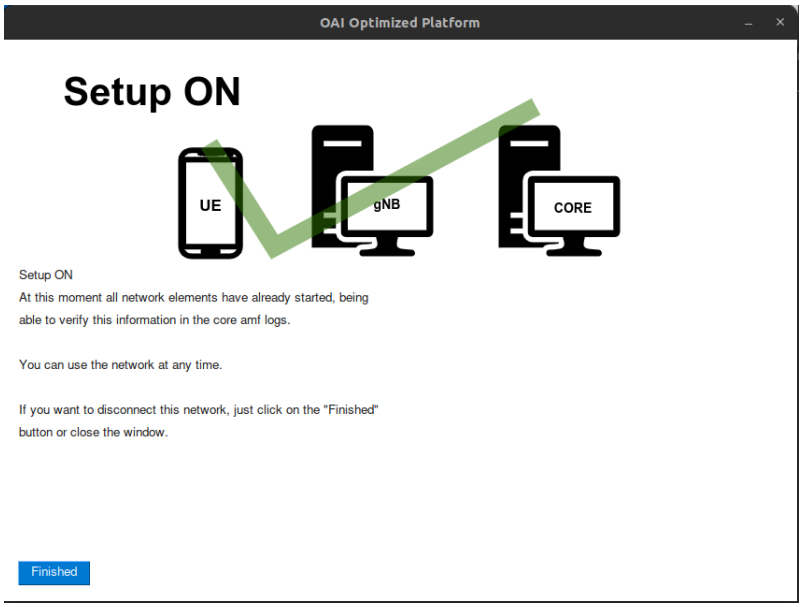

Figure A.88: Configure and run setup with core and gnb on different hosts with COTS UE (Custom mode) 9ª interface.

A.2.7 Configure and run setup All-in-One with OAI UE (Custom mode)

Figure A.89: Configure and run setup All-in-One with OAI UE (Custom mode) 1ª interface.

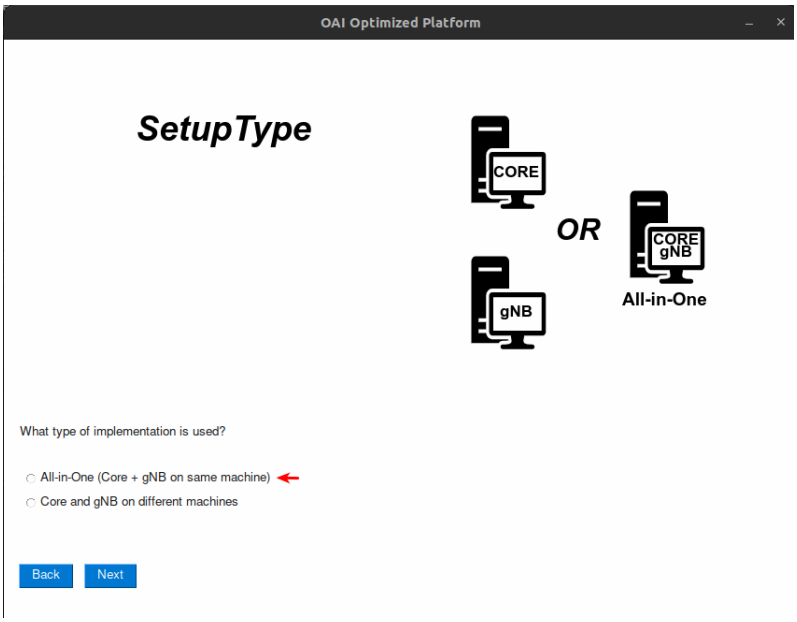

Figure A.90: Configure and run setup All-in-One with OAI UE (Custom mode) 2ª interface.

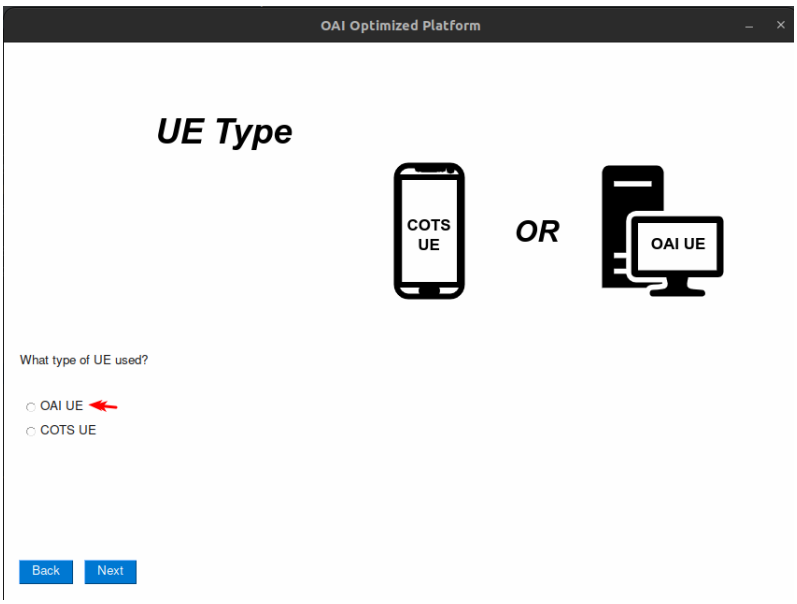

Figure A.91: Configure and run setup All-in-One with OAI UE (Custom mode) 3ª interface.

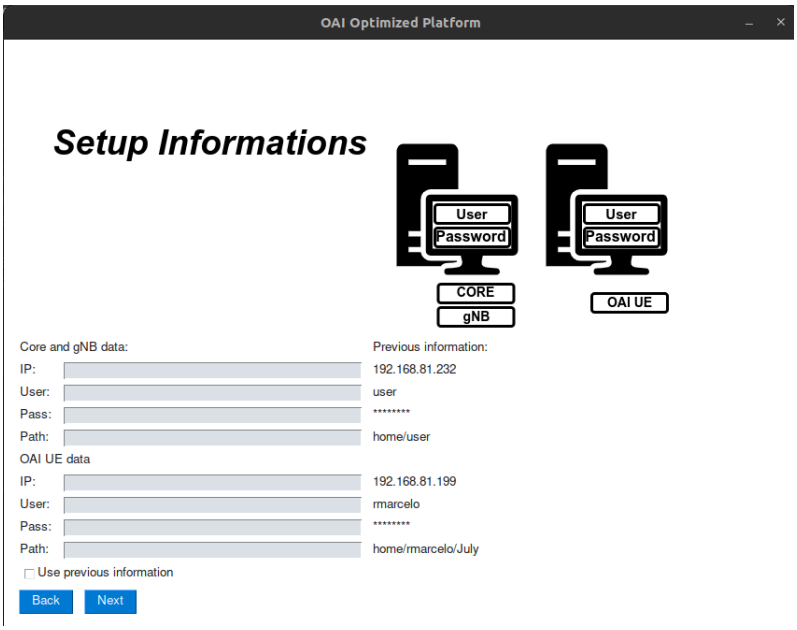

Figure A.92: Configure and run setup All-in-One with OAI UE (Custom mode) 4ª interface.

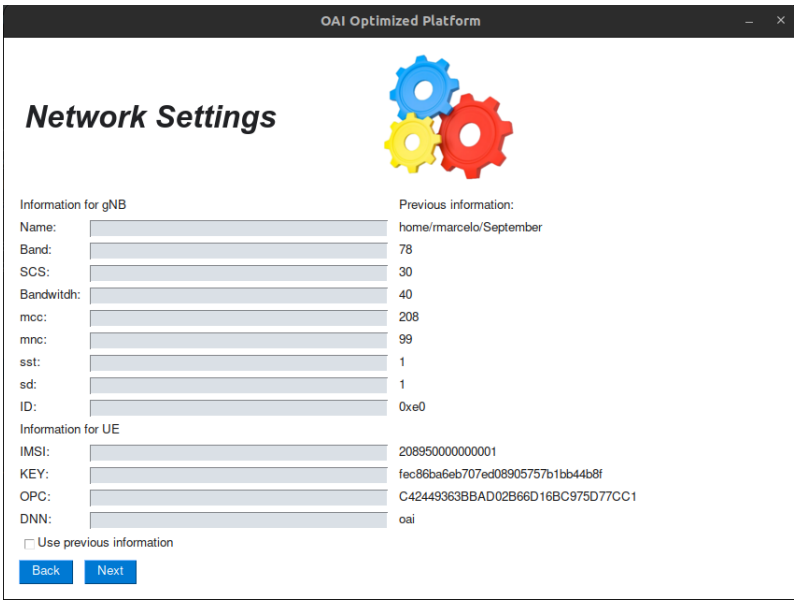

Figure A.93: Configure and run setup All-in-One with OAI UE (Custom mode) 5ª interface.

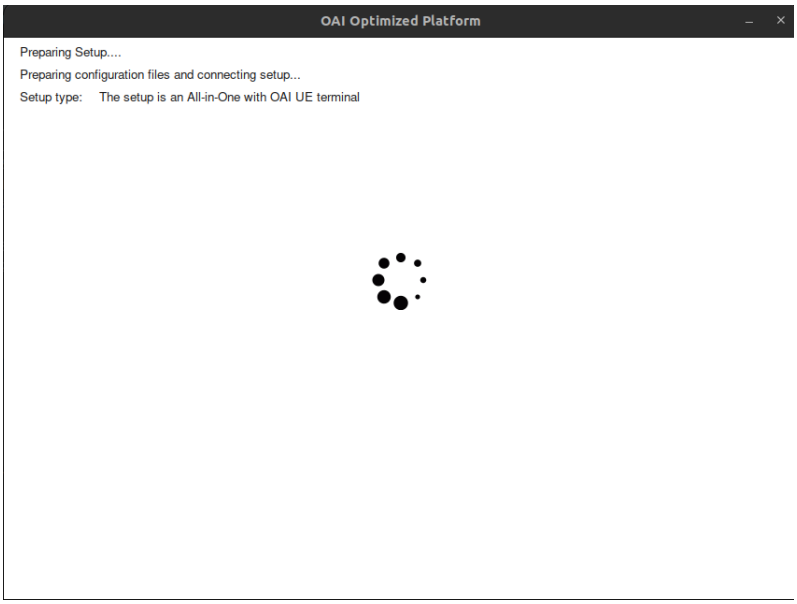

Figure A.94: Configure and run setup All-in-One with OAI UE (Custom mode) 6ª interface.

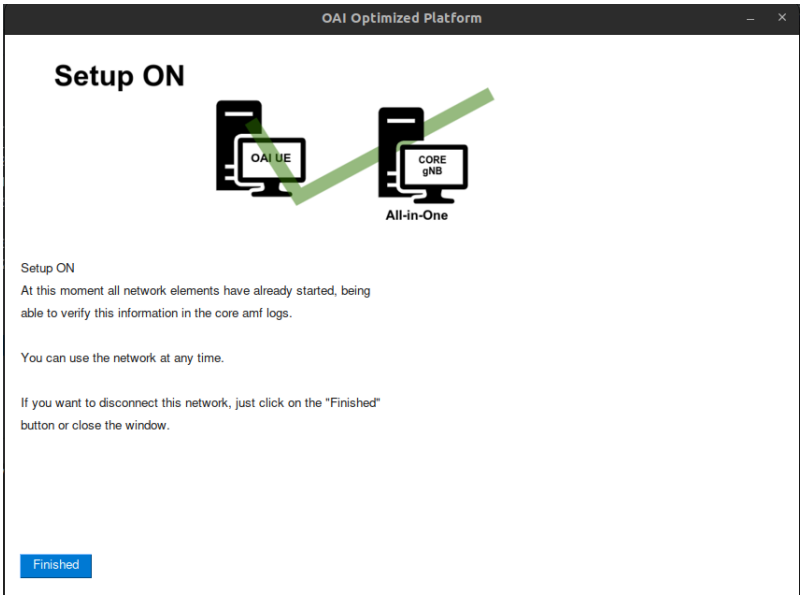

Figure A.95: Configure and run setup All-in-One with OAI UE (Custom mode) 7ª interface.

A.2.8 Configure and run setup All-in-One with COTS UE (Custom mode)

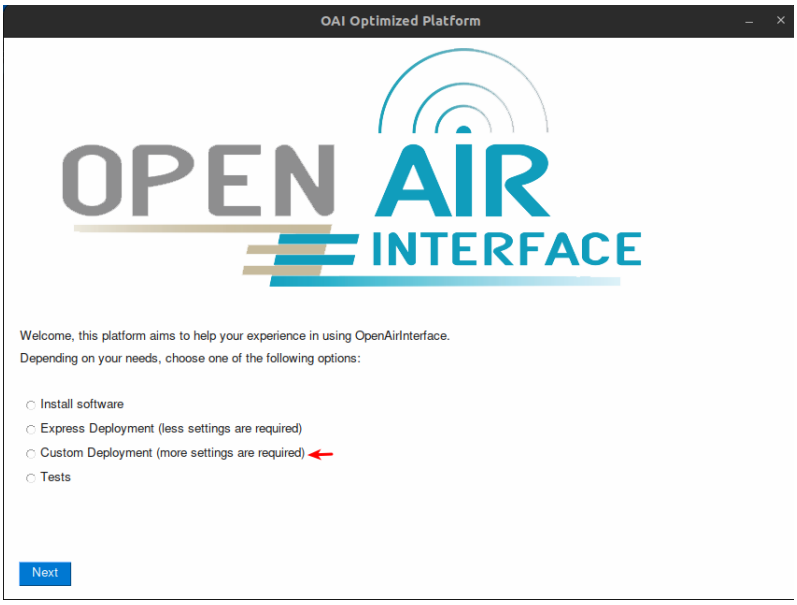

Figure A.96: Configure and run setup All-in-One with COTS UE (Custom mode) 1ª interface.

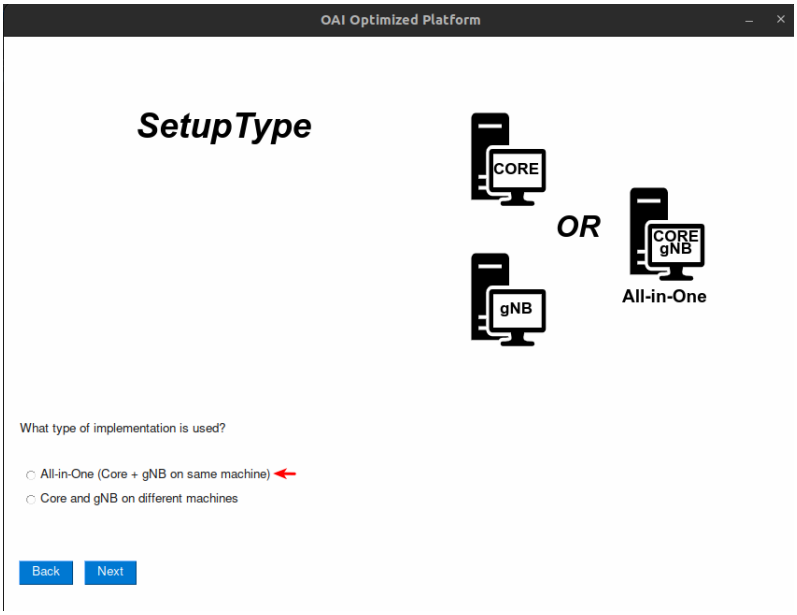

Figure A.97: Configure and run setup All-in-One with COTS UE (Custom mode) 2ª interface.

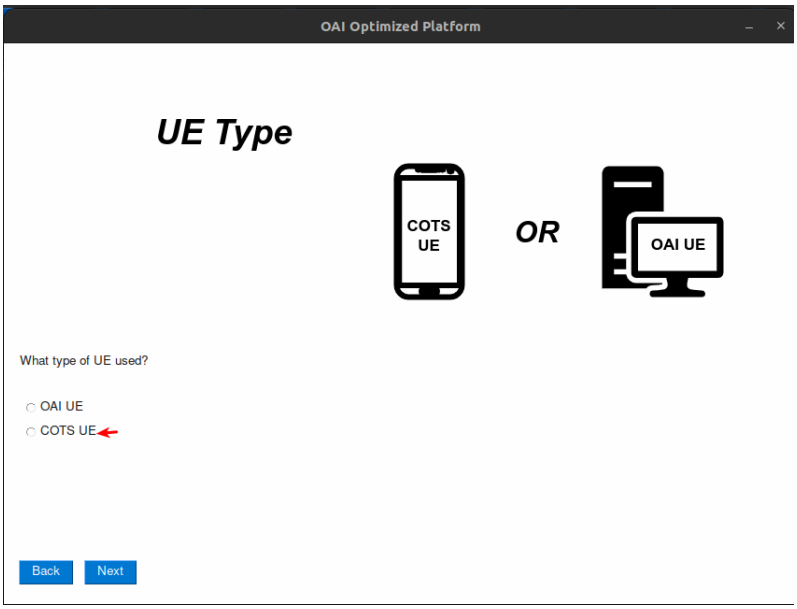

Figure A.98: Configure and run setup All-in-One with COTS UE (Custom mode) 3ª interface.

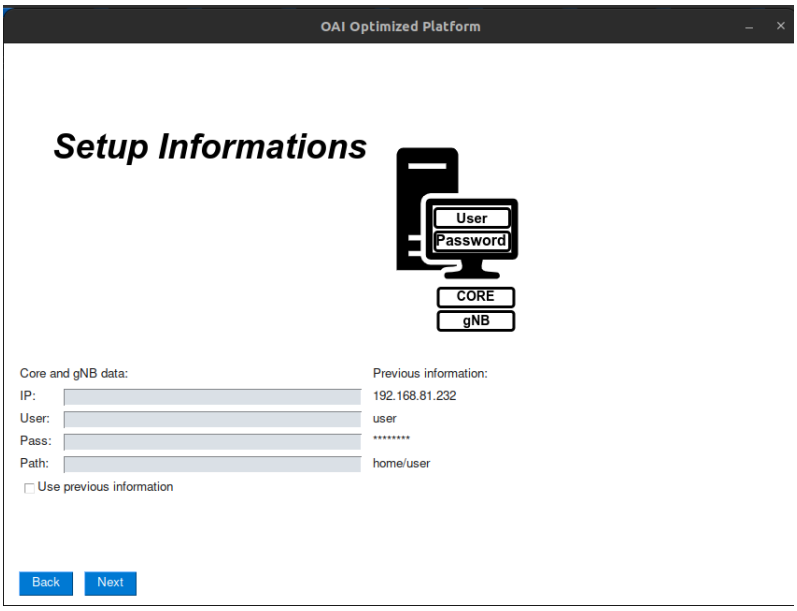

Figure A.99: Configure and run setup All-in-One with COTS UE (Custom mode) 4ª interface.

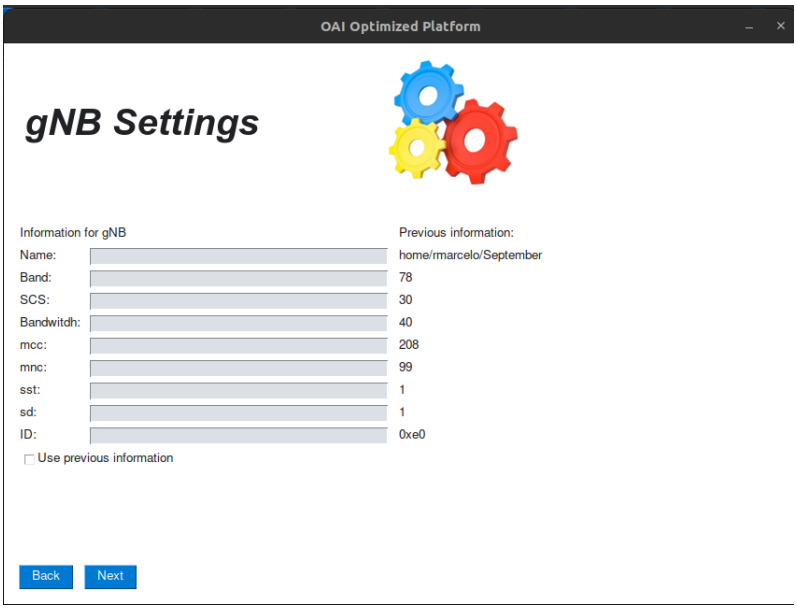

Figure A.100: Configure and run setup All-in-One with COTS UE (Custom mode) 5ª interface.

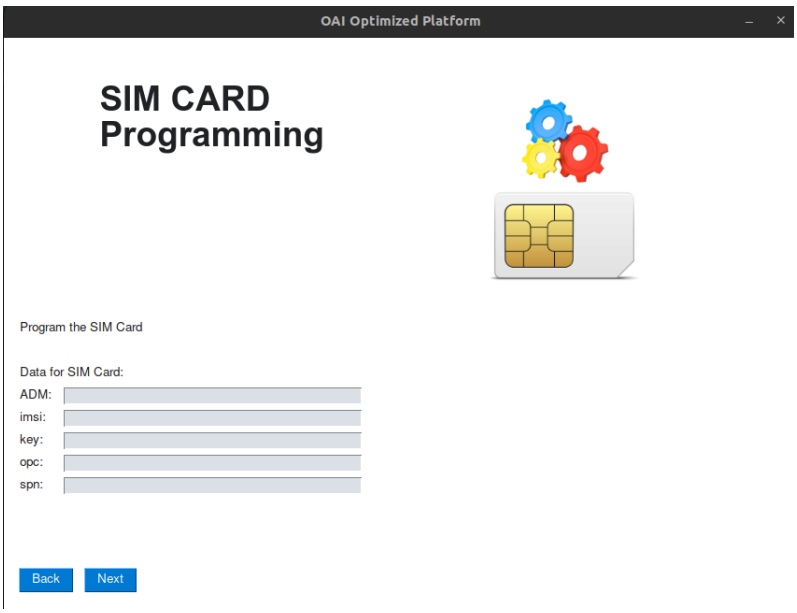

Figure A.101: Configure and run setup All-in-One with COTS UE (Custom mode) 6ª interface.

Figure A.102: Configure and run setup All-in-One with COTS UE (Custom mode) 7ª interface.

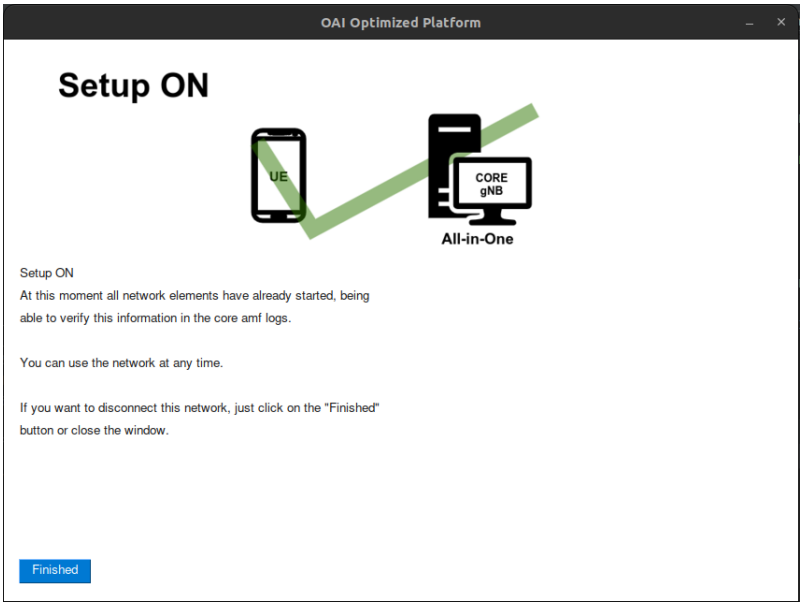

Figure A.103: Configure and run setup All-in-One with COTS UE (Custom mode) 8ª interface.

A.3 Test Features

A.3.1 Test configuration

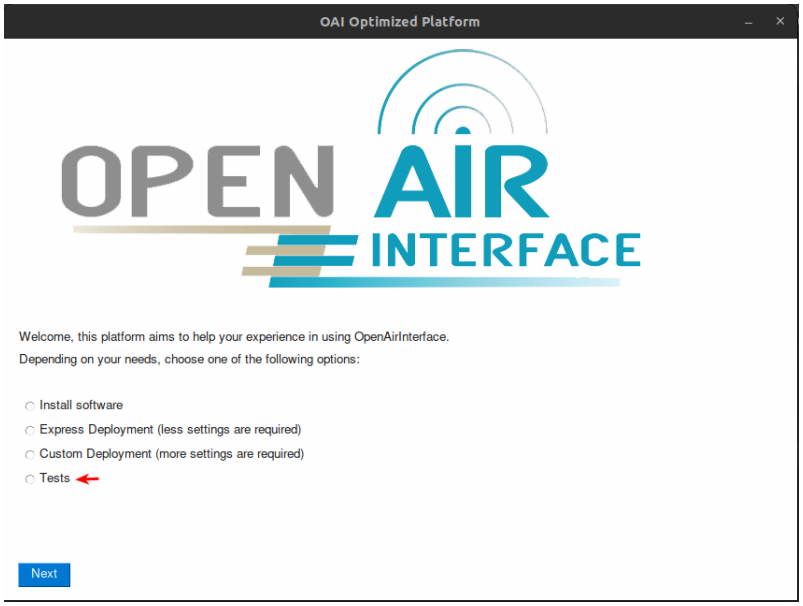

Figure A.104: Test configuration 1ª interface.

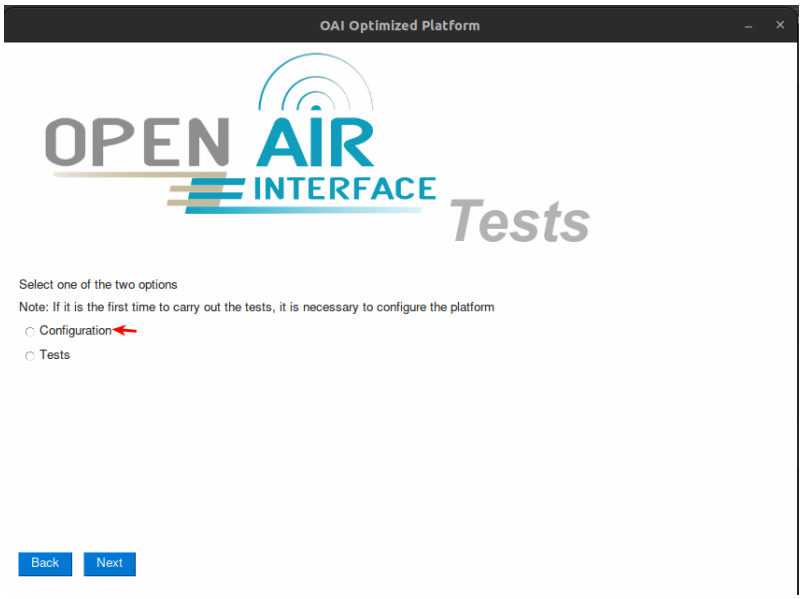

Figure A.105: Test configuration 2ª interface.

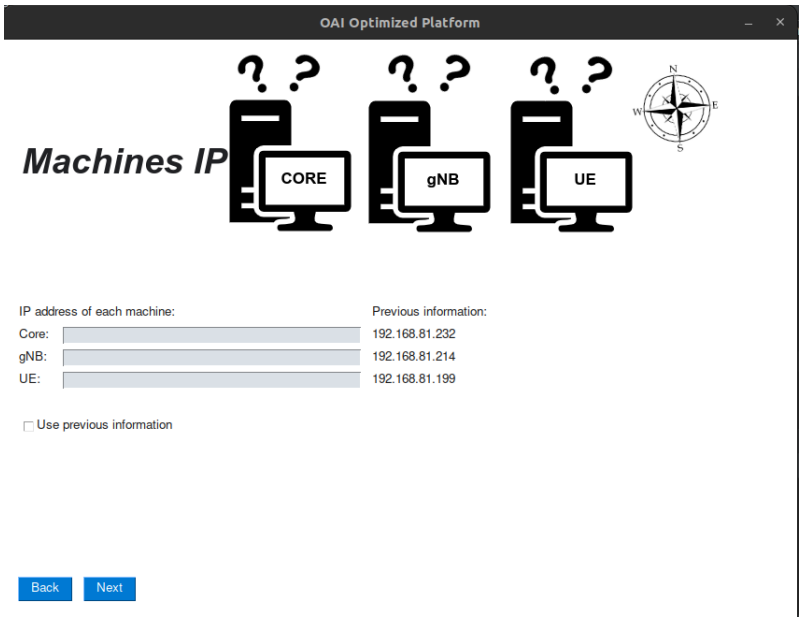

Figure A.106: Test configuration 3ª interface.

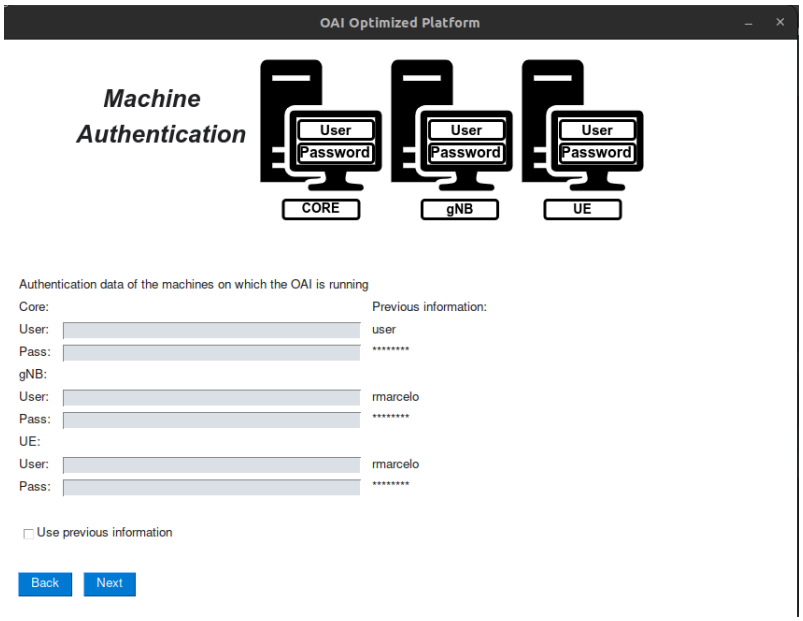

Figure A.107: Test configuration 4ª interface.

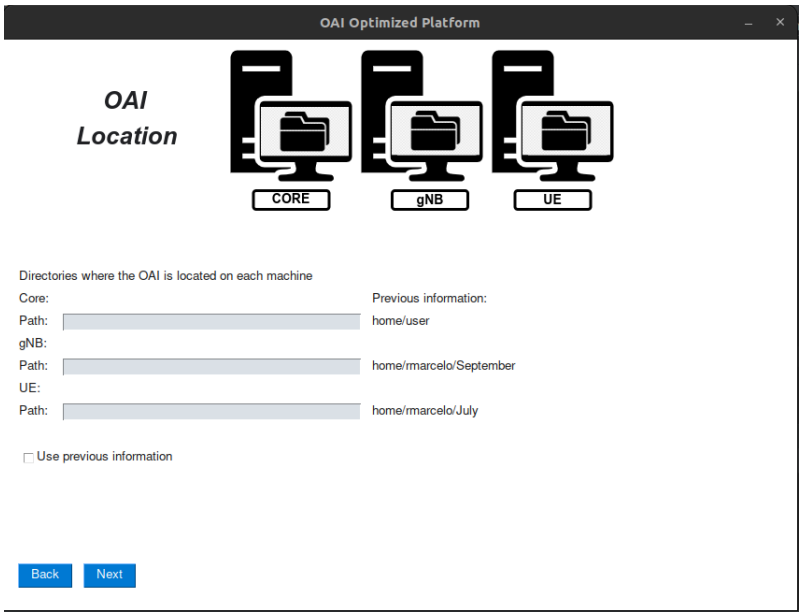

Figure A.108: Test configuration 5ª interface.

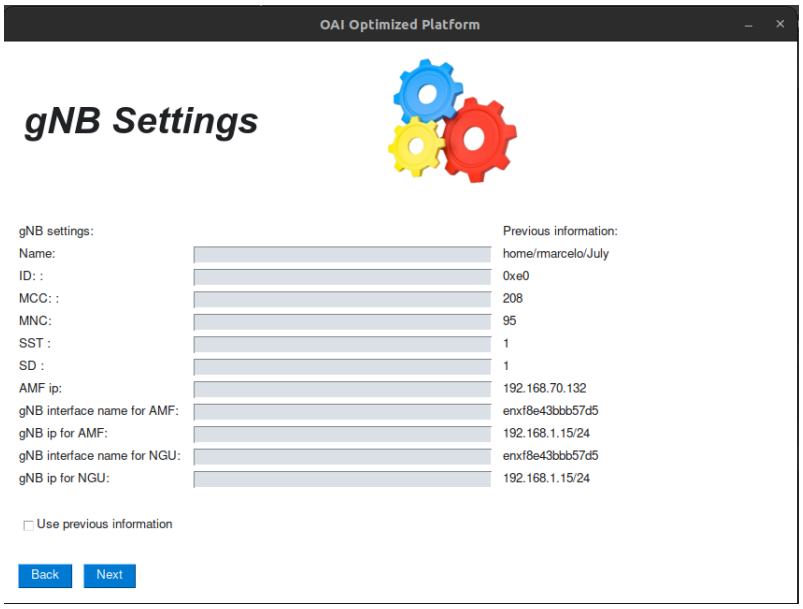

Figure A.109: Test configuration 6ª interface.

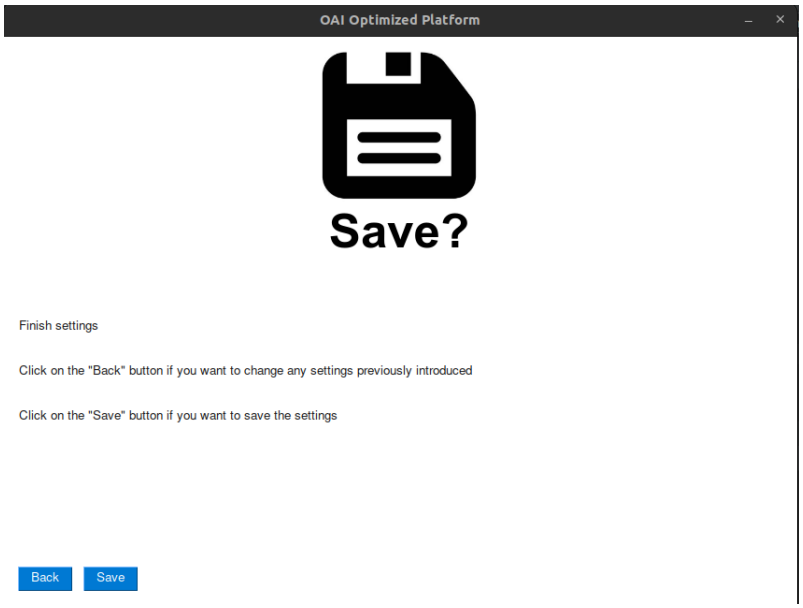

Figure A.110: Test configuration 7ª interface.

A.3.2 Core Test

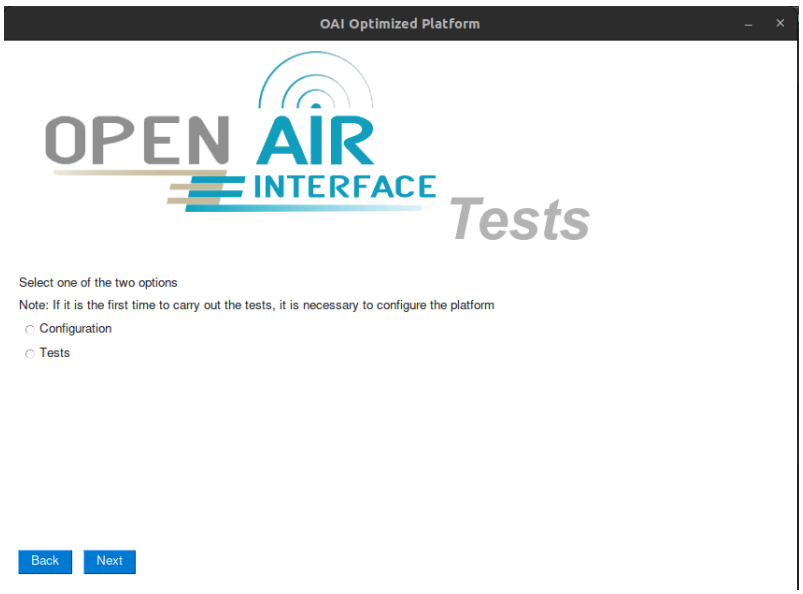

Figure A.111: Core Test 1ª interface.

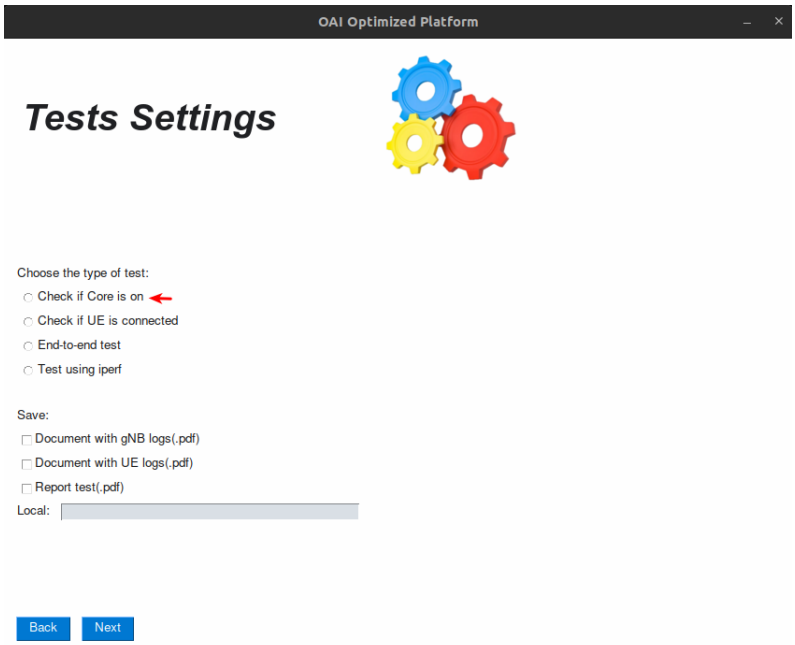

Figure A.112: Core Test 2ª interface.

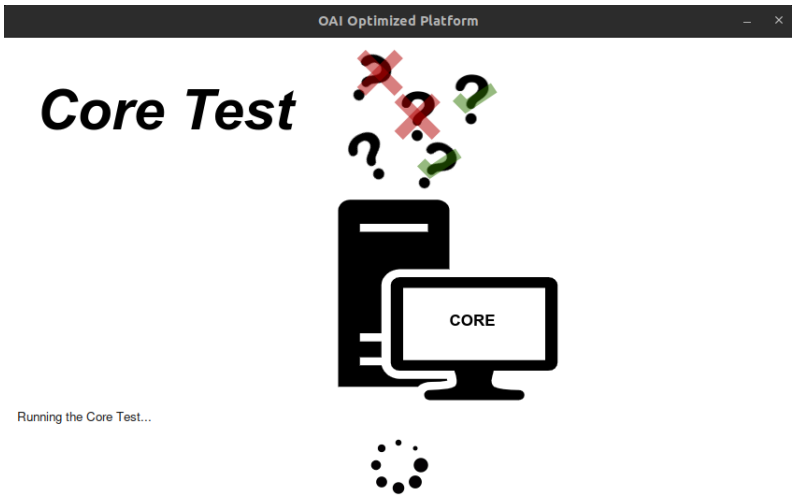

Figure A.113: Core Test 3ª interface.

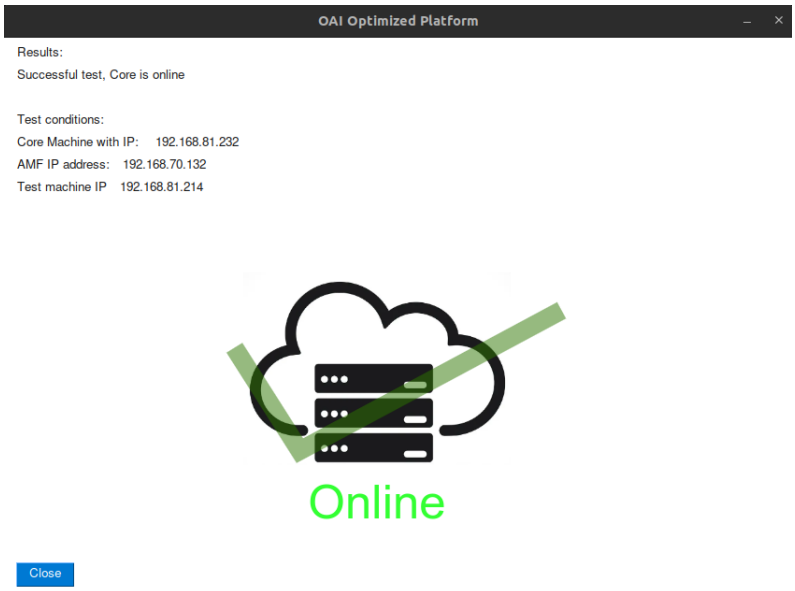

Figure A.114: Core Test 4ª interface.

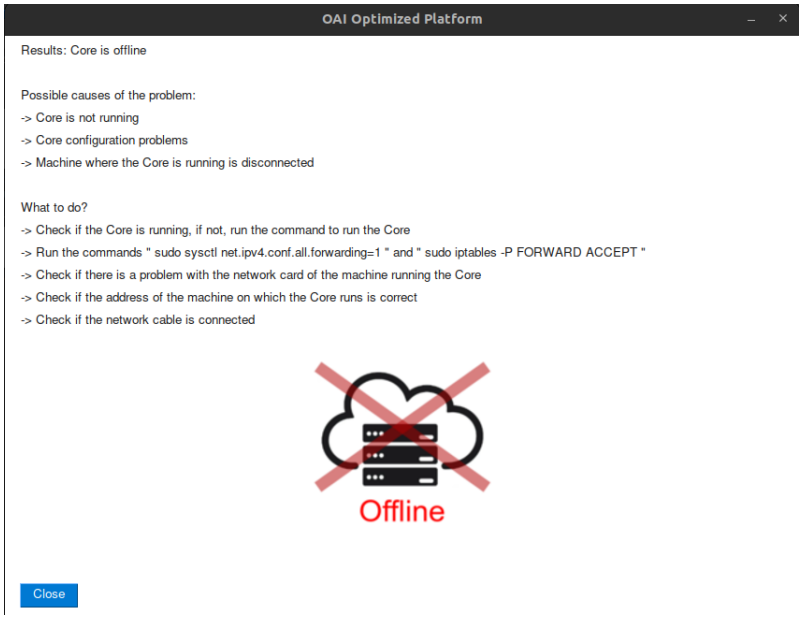

Figure A.115: Core Test 5ª interface.

A.3.3 UE Connection Test

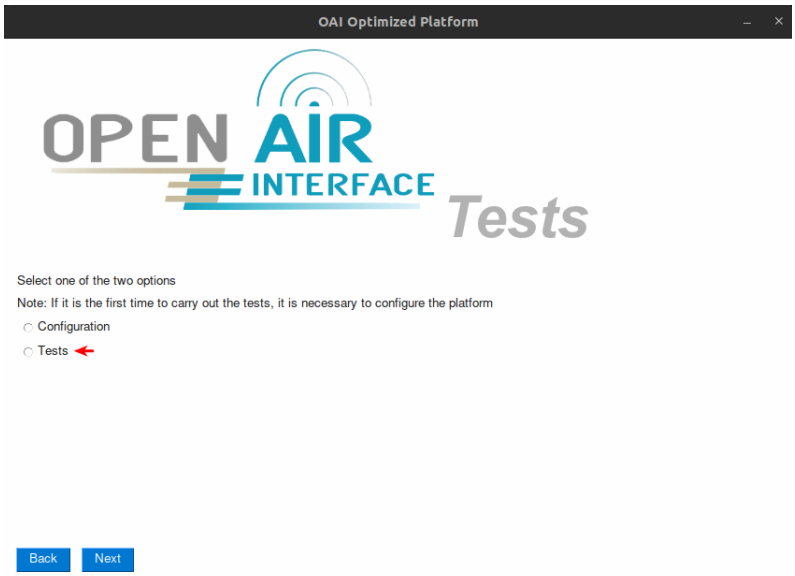

Figure A.116: UE Connection Test 1ª interface.

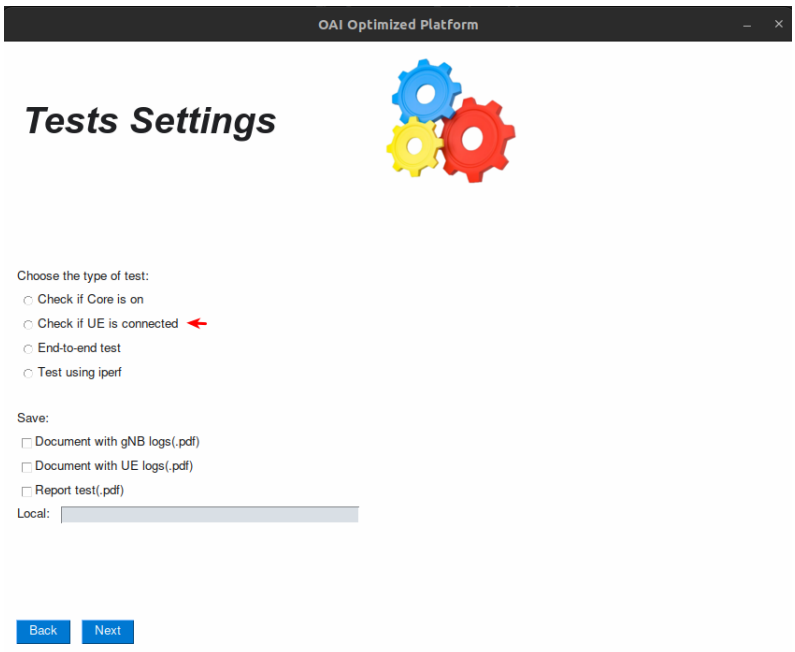

Figure A.117: UE Connection Test 2ª interface.

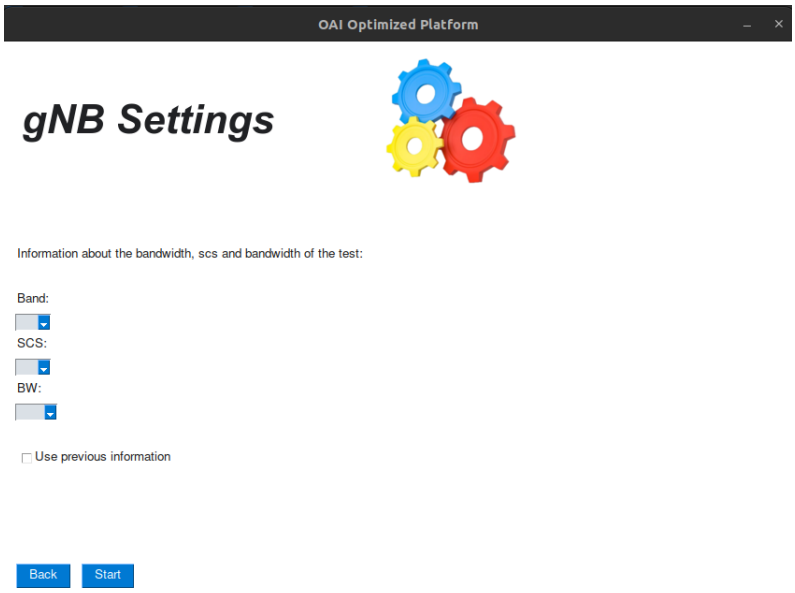

Figure A.118: UE Connection Test 3ª interface.

Figure A.119: UE Connection Test 4ª interface.

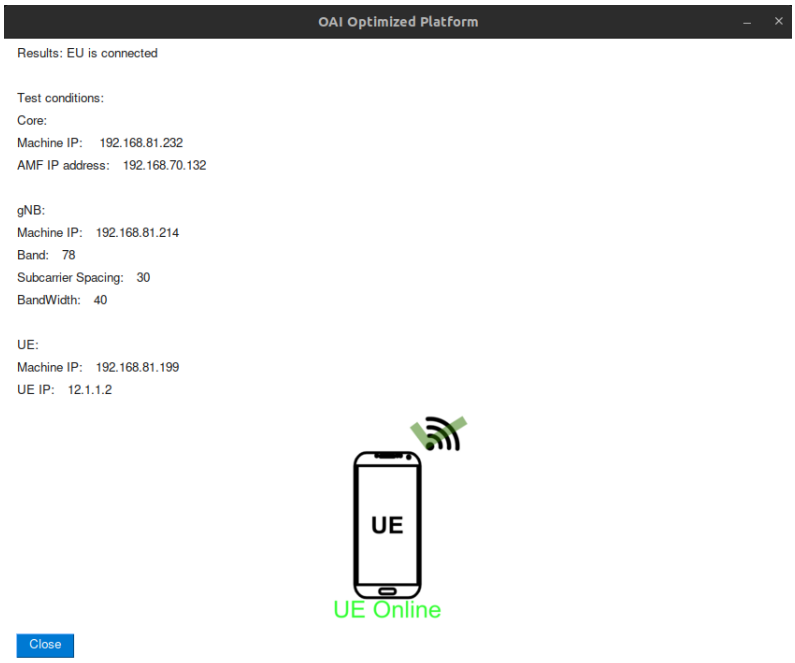

Figure A.120: UE Connection Test 5ª interface.

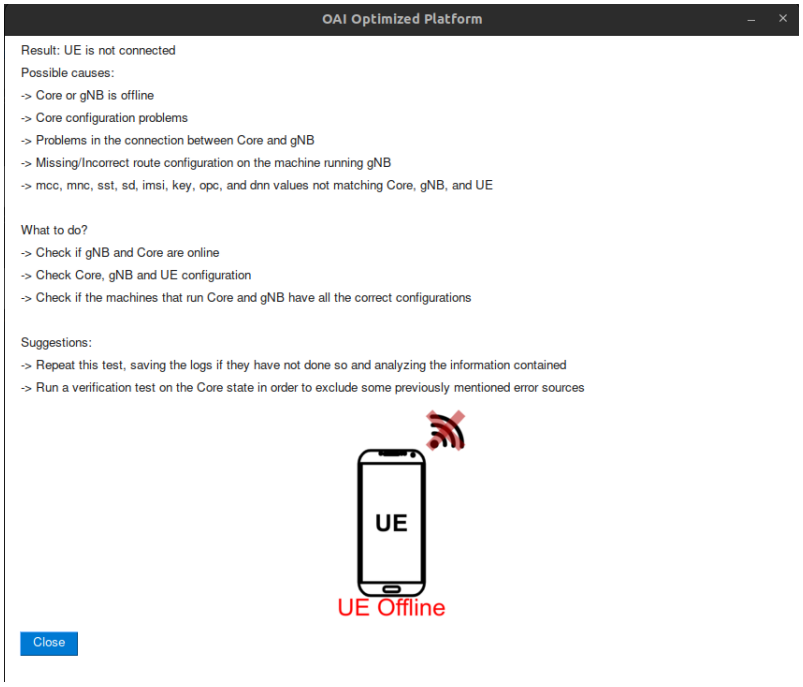

Figure A.121: UE Connection Test 6ª interface.

A.3.4 End-to-End Test

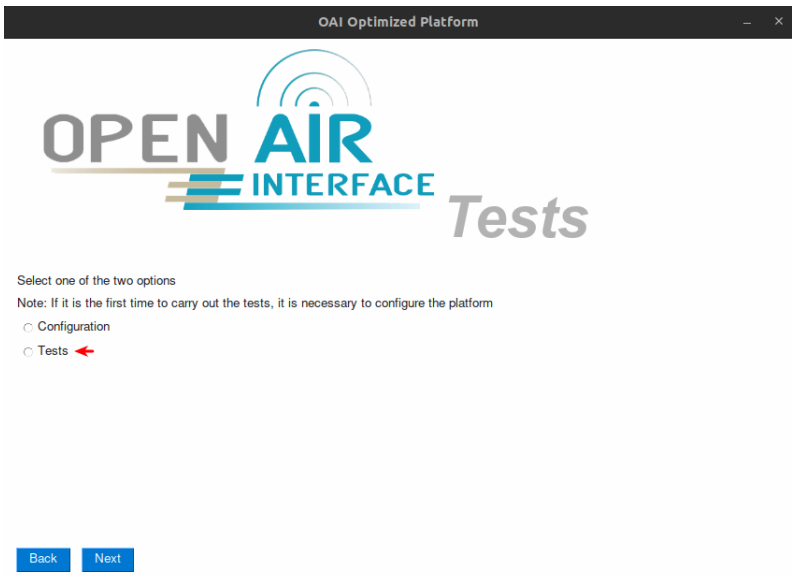

Figure A.122: End-to-End Test 1ª interface.

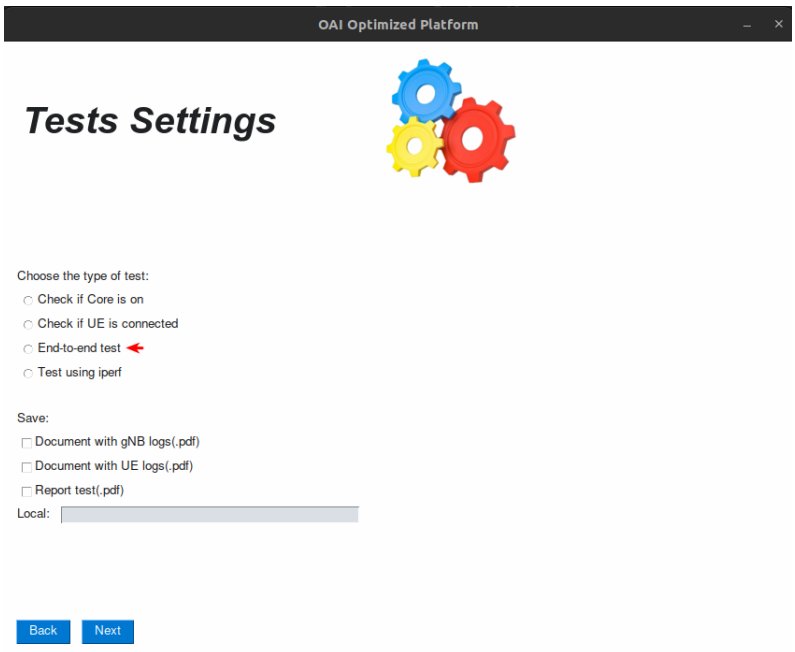

Figure A.123: End-to-End Test 2ª interface.

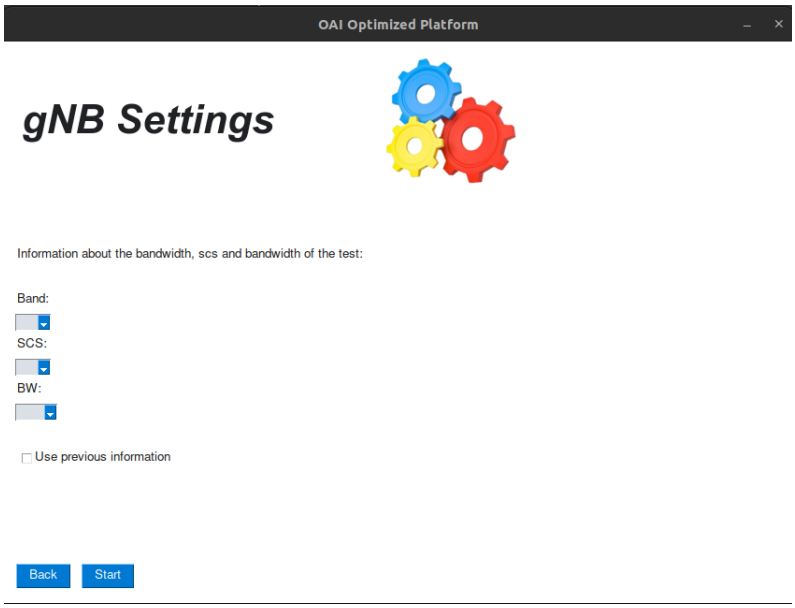

Figure A.124: End-to-End Test 3ª interface.

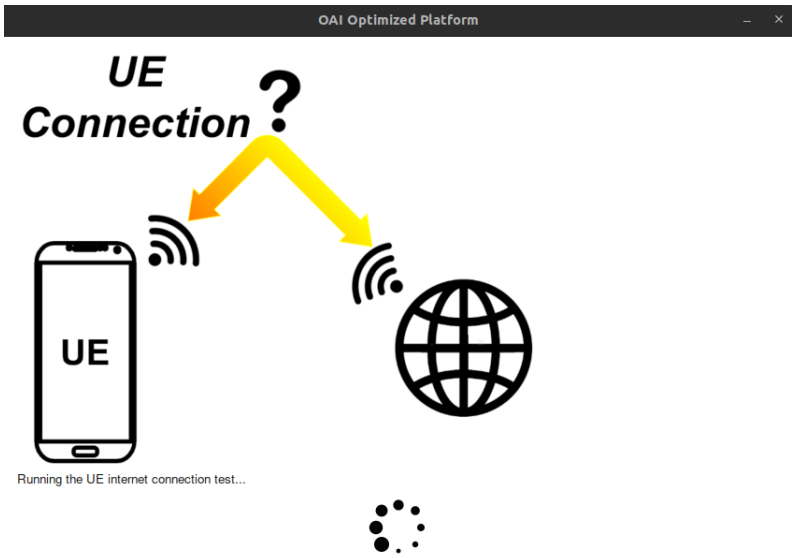

Figure A.125: End-to-End Test 4ª interface.

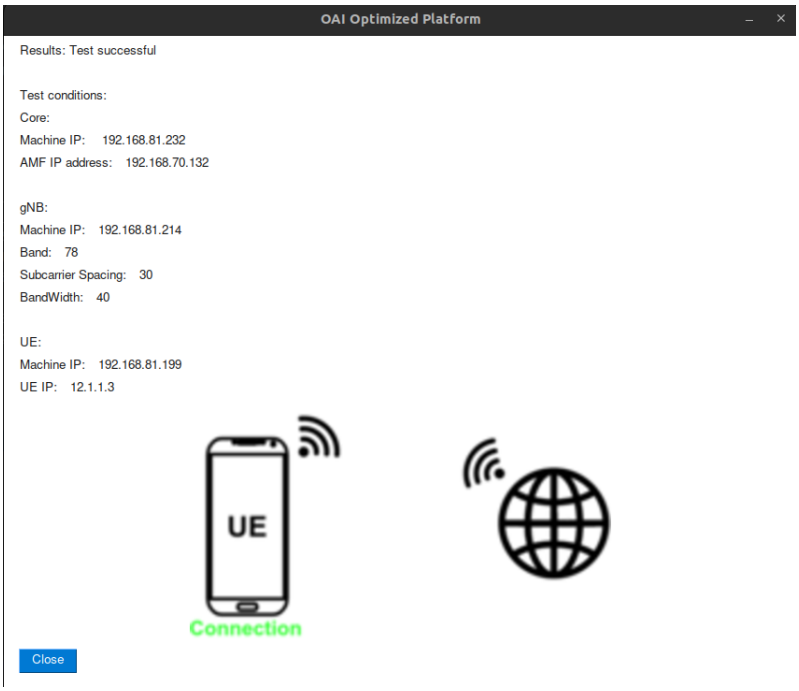

Figure A.126: End-to-End Test 5ª interface.

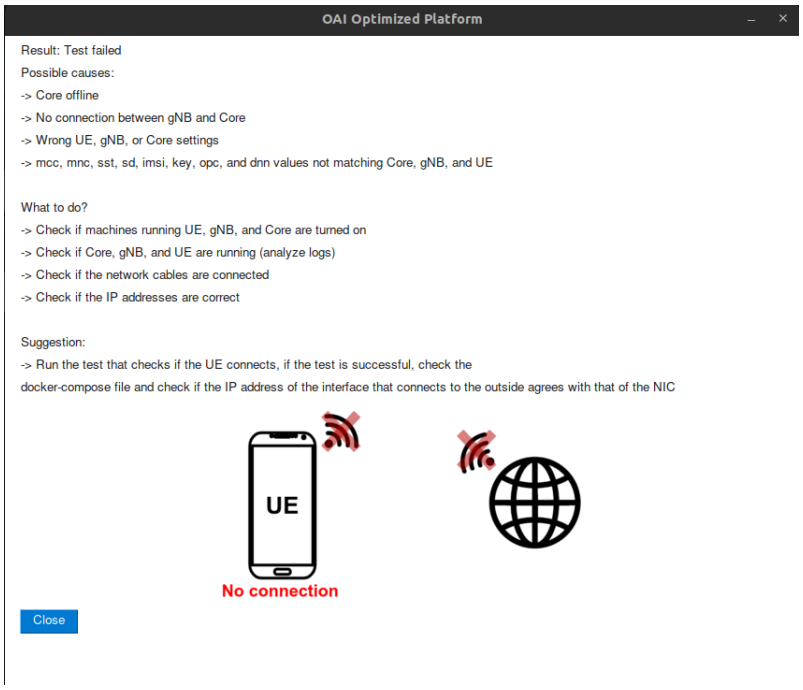

Figure A.127: End-to-End Test 6ª interface.

A.3.5 Performance Test

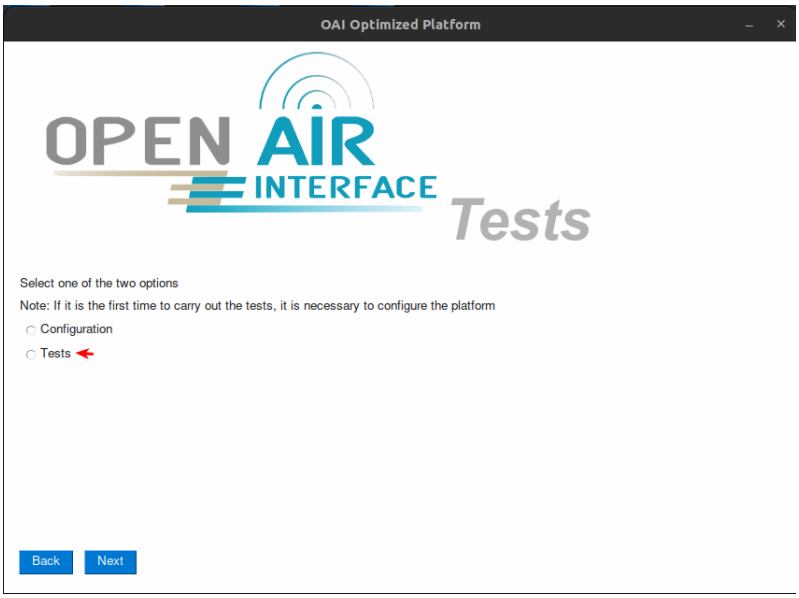

Figure A.128: Performance Test 1ª interface.

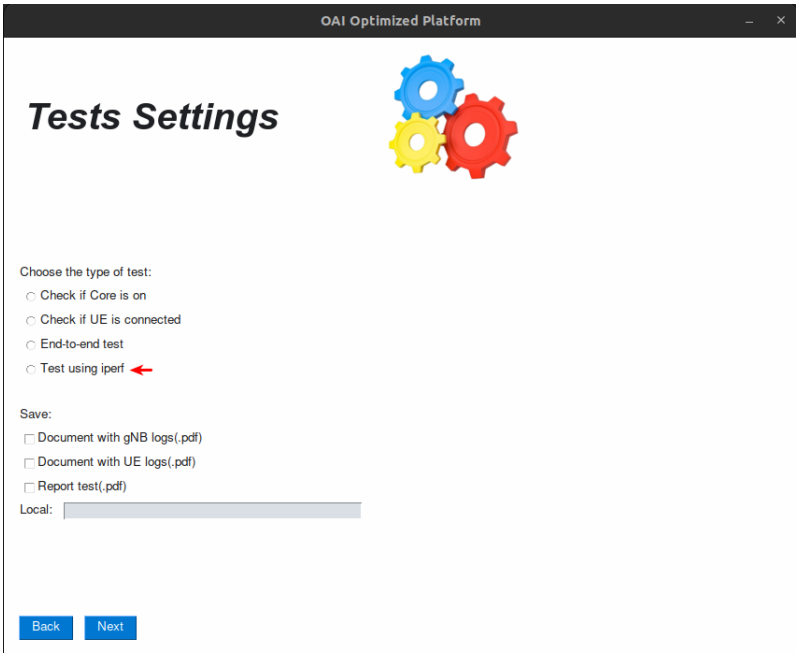

Figure A.129: Performance Test 2ª interface.

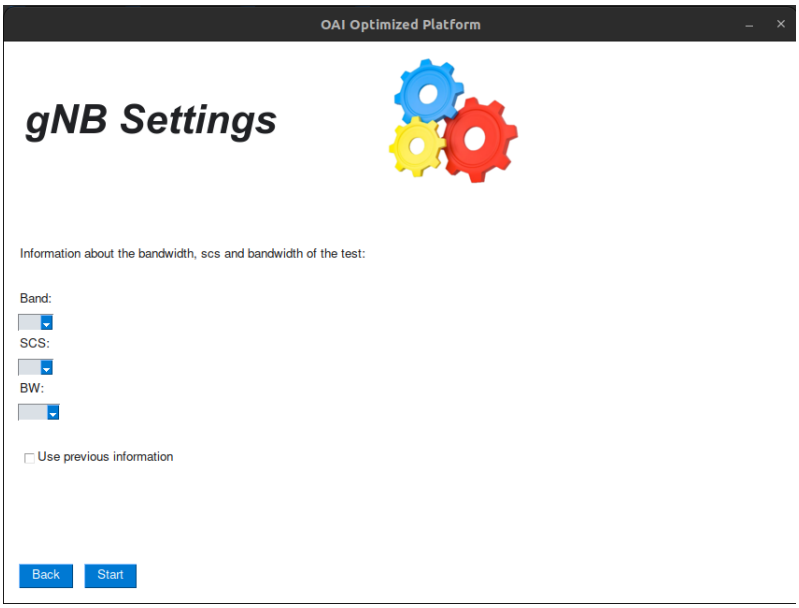

Figure A.130: Performance Test 3ª interface.

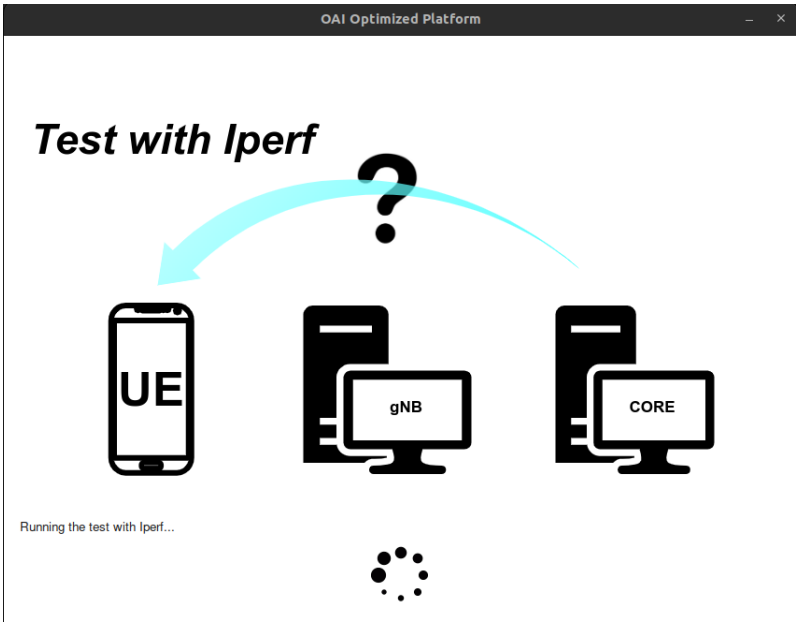

Figure A.131: Performance Test 4ª interface.

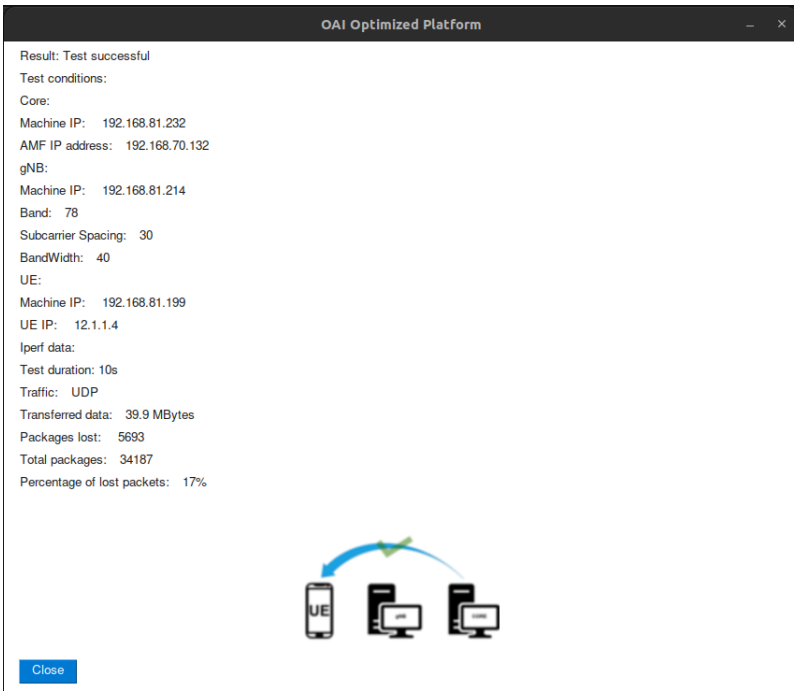

Figure A.132: Performance Test 5[a] interface.

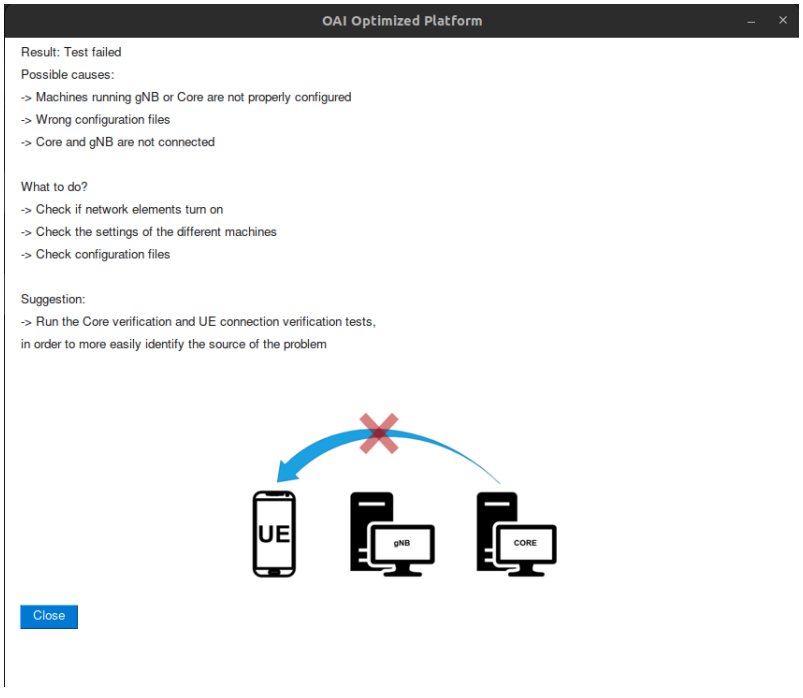

Figure A.133: Performance Test 6[a] interface.

Appendix B

Software Installations

B.1 Install Docker

```
sudo apt install -y apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu  $(lsb_release -cs)  stable
sudo apt update
sudo apt install -y docker docker-ce

sudo usermod -a -G docker $(whoami)
reboot

sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m
sudo chmod +x /usr/local/bin/docker-compose
```

B.2 Install USRP software

```
git clone https://github.com/EttusResearch/uhd.git ~/uhd
cd ~/uhd
git checkout v4.0.0.0
cd host
mkdir build
cd build
cmake ../
make -j 4
sudo make install
sudo ldconfig
sudo uhd_images_downloader
```

Appendix C

Configuration Files

Configuration files used for implementation on band 78 with a subcarrier spacing of 30kHz and bandwidth of 40MHz

C.1 docker-compose file

```
version: '3.8'
services:
    mysql:
        container_name: "mysql"
        image: mysql:5.7
        volumes:
            - ./database/oai_db.sql:/docker-entrypoint-initdb.d/oai_db.sql
            - ./healthscripts/mysql-healthcheck2.sh:/tmp/mysql-healthcheck.sh
        environment:
            - TZ=Europe/Paris
            - MYSQL_DATABASE=oai_db
            - MYSQL_USER=test
            - MYSQL_PASSWORD=test
            - MYSQL_ROOT_PASSWORD=linux
        healthcheck:
            test: /bin/bash -c "/tmp/mysql-healthcheck.sh"
            interval: 10s
            timeout: 5s
            retries: 5
        networks:
            public_net:
                ipv4_address: 192.168.70.131
    oai-udr:
        container_name: "oai-udr"
        image: oai-udr:develop
        environment:
            - TZ=Europe/Paris
            - INSTANCE=0
            - PID_DIRECTORY=/var/run
            - UDR_NAME=OAI_UDR
            - UDR_INTERFACE_NAME_FOR_NUDR=eth0
            - UDR_INTERFACE_PORT_FOR_NUDR=80
            - UDR_INTERFACE_HTTP2_PORT_FOR_NUDR=8080
            - USE_HTTP2=no
            - UDR_API_VERSION=v1
            - MYSQL_IPV4_ADDRESS=192.168.70.131
```

```
        - MYSQL_USER=test
        - MYSQL_PASS=test
        - DB_CONNECTION_TIMEOUT=300 # Reset the connection to the DB after expiring the timeout (in secon
        - MYSQL_DB=oai_db
        - WAIT_MYSQL=120
        - USE_FQDN_DNS=yes
        - REGISTER_NRF=yes
        - NRF_IPV4_ADDRESS=192.168.70.130
        - NRF_PORT=80
        - NRF_API_VERSION=v1
        - NRF_FQDN=oai-nrf
    depends_on:
        - mysql
        - oai-nrf
    networks:
        public_net:
            ipv4_address: 192.168.70.136
    volumes:
        - ./healthscripts/udr-healthcheck.sh:/openair-udr/bin/udr-healthcheck.sh
    healthcheck:
        test: /bin/bash -c "/openair-udr/bin/udr-healthcheck.sh"
        interval: 10s
        timeout: 5s
        retries: 5
oai-udm:
    container_name: "oai-udm"
    image: oai-udm:develop
    environment:
        - TZ=Europe/Paris
        - INSTANCE=0
        - PID_DIRECTORY=/var/run
        - UDM_NAME=OAI_UDM
        - SBI_IF_NAME=eth0
        - SBI_PORT=80
        - SBI_HTTP2_PORT=8080
        - USE_HTTP2=no
        - UDM_VERSION_NB=v1
        - USE_FQDN_DNS=yes
        - UDR_IP_ADDRESS=192.168.70.136
        - UDR_PORT=80
        - UDR_VERSION_NB=v1
        - UDR_FQDN=oai-udr
        - REGISTER_NRF=yes
        - NRF_IPV4_ADDRESS=192.168.70.130
        - NRF_PORT=80
        - NRF_API_VERSION=v1
        - NRF_FQDN=oai-nrf
    depends_on:
        - oai-udr
    networks:
        public_net:
            ipv4_address: 192.168.70.137
    volumes:
        - ./healthscripts/udm-healthcheck.sh:/openair-udm/bin/udm-healthcheck.sh
    healthcheck:
        test: /bin/bash -c "/openair-udm/bin/udm-healthcheck.sh"
        interval: 10s
```

```
        timeout: 5s
        retries: 5
oai-ausf:
    container_name: "oai-ausf"
    image: oai-ausf:develop
    environment:
        - TZ=Europe/Paris
        - INSTANCE_ID=0
        - PID_DIR=/var/run
        - AUSF_NAME=OAI_AUSF
        - SBI_IF_NAME=eth0
        - SBI_PORT=80
        - USE_HTTP2
        - SBI_HTTP2_PORT
        - USE_FQDN_DNS=yes
        - UDM_IP_ADDRESS=192.168.70.137
        - UDM_PORT=80
        - UDM_VERSION_NB=v1
        - UDM_FQDN=oai-udm
        - REGISTER_NRF=yes
        - NRF_IPV4_ADDRESS=192.168.70.130
        - NRF_PORT=80
        - NRF_API_VERSION=v1
        - NRF_FQDN=oai-nrf
    depends_on:
        - oai-udm
    networks:
        public_net:
            ipv4_address: 192.168.70.138
    volumes:
        - ./healthscripts/ausf-healthcheck.sh:/openair-ausf/bin/ausf-healthcheck.sh
    healthcheck:
        test: /bin/bash -c "/openair-ausf/bin/ausf-healthcheck.sh"
        interval: 10s
        timeout: 5s
        retries: 5
oai-nrf:
    container_name: "oai-nrf"
    image: oai-nrf:develop
    environment:
        - TZ=Europe/Paris
        - NRF_INTERFACE_NAME_FOR_SBI=eth0
        - NRF_INTERFACE_PORT_FOR_SBI=80
        - NRF_INTERFACE_HTTP2_PORT_FOR_SBI=8080
        - NRF_API_VERSION=v1
        - INSTANCE=0
        - PID_DIRECTORY=/var/run
    networks:
        public_net:
            ipv4_address: 192.168.70.130
    volumes:
        - ./healthscripts/nrf-healthcheck.sh:/openair-nrf/bin/nrf-healthcheck.sh
    healthcheck:
        test: /bin/bash -c "/openair-nrf/bin/nrf-healthcheck.sh"
        interval: 10s
        timeout: 5s
        retries: 5
```

```
oai-amf:
    container_name: "oai-amf"
    image: oai-amf:develop
    environment:
        - TZ=Europe/paris
        - INSTANCE=0
        - PID_DIRECTORY=/var/run
        - MCC=208
        - MNC=99
        - REGION_ID=128
        - AMF_SET_ID=1
        - SERVED_GUAMI_MCC_0=208
        - SERVED_GUAMI_MNC_0=99
        - SERVED_GUAMI_REGION_ID_0=128
        - SERVED_GUAMI_AMF_SET_ID_0=1
        - SERVED_GUAMI_MCC_1=460
        - SERVED_GUAMI_MNC_1=11
        - SERVED_GUAMI_REGION_ID_1=10
        - SERVED_GUAMI_AMF_SET_ID_1=1
        - PLMN_SUPPORT_MCC=208
        - PLMN_SUPPORT_MNC=99
        - PLMN_SUPPORT_TAC=0x0001
        - SST_0=1
        - SD_0=1
        - AMF_INTERFACE_NAME_FOR_NGAP=eth0
        - AMF_INTERFACE_NAME_FOR_N11=eth0
        - SMF_INSTANCE_ID_0=1
        - SMF_FQDN_0=oai-smf
        - SMF_IPV4_ADDR_0=192.168.70.133
        - SMF_HTTP_VERSION_0=v1
        - SELECTED_0=true
        - SMF_INSTANCE_ID_1=2
        - SMF_FQDN_1=oai-smf
        - SMF_IPV4_ADDR_1=0.0.0.0
        - SMF_HTTP_VERSION_1=v1
        - SELECTED_1=false
        - MYSQL_SERVER=192.168.70.131
        - MYSQL_USER=root
        - MYSQL_PASS=linux
        - MYSQL_DB=oai_db
        - OPERATOR_KEY=1006020f0a478bf6b699f15c062e42b3
        - NRF_IPV4_ADDRESS=192.168.70.130
        - NRF_PORT=80
        - EXTERNAL_NRF=no
        - NF_REGISTRATION=yes
        - SMF_SELECTION=yes
        - USE_FQDN_DNS=yes
        - EXTERNAL_AUSF=yes
        - EXTERNAL_UDM=no
        - EXTERNAL_NSSF=no
        - USE_HTTP2=no
        - NRF_API_VERSION=v1
        - NRF_FQDN=oai-nrf
        - AUSF_IPV4_ADDRESS=192.168.70.138
        - AUSF_PORT=80
        - AUSF_API_VERSION=v1
        - AUSF_FQDN=oai-ausf
```

```
        - UDM_IPV4_ADDRESS=192.168.70.137
        - UDM_PORT=80
        - UDM_API_VERSION=v2
        - UDM_FQDN=oai-udm
    depends_on:
        - mysql
        - oai-nrf
        - oai-ausf
    volumes:
        - ./healthscripts/amf-healthcheck.sh:/openair-amf/bin/amf-healthcheck.sh
    healthcheck:
        test: /bin/bash -c "/openair-amf/bin/amf-healthcheck.sh"
        interval: 10s
        timeout: 15s
        retries: 5
    networks:
        public_net:
            ipv4_address: 192.168.70.132
oai-smf:
    container_name: "oai-smf"
    image: oai-smf:develop
    environment:
        - TZ=Europe/Paris
        - INSTANCE=0
        - PID_DIRECTORY=/var/run
        - SMF_INTERFACE_NAME_FOR_N4=eth0
        - SMF_INTERFACE_NAME_FOR_SBI=eth0
        - SMF_INTERFACE_PORT_FOR_SBI=80
        - SMF_INTERFACE_HTTP2_PORT_FOR_SBI=9090
        - SMF_API_VERSION=v1
        - DEFAULT_DNS_IPV4_ADDRESS=dns1
        - DEFAULT_DNS_SEC_IPV4_ADDRESS=dns2
        - AMF_IPV4_ADDRESS=192.168.70.132
        - AMF_PORT=80
        - AMF_API_VERSION=v1
        - AMF_FQDN=oai-amf
        - UDM_IPV4_ADDRESS=192.168.70.137
        - UDM_PORT=80
        - UDM_API_VERSION=v2
        - UDM_FQDN=oai-udm
        - UPF_IPV4_ADDRESS=192.168.70.134
        - UPF_FQDN_0=oai-spgwu
        - NRF_IPV4_ADDRESS=192.168.70.130
        - NRF_PORT=80
        - NRF_API_VERSION=v1
        - USE_LOCAL_SUBSCRIPTION_INFO=yes  #Set to yes if SMF uses local subscription information instead
        - USE_NETWORK_INSTANCE=no  #Set yes if network instance is to be used for given UPF
        - NRF_FQDN=oai-nrf
        - REGISTER_NRF=yes
        - DISCOVER_UPF=yes
        - USE_FQDN_DNS=yes
        - HTTP_VERSION=1        # Default: 1
        - UE_MTU=1500
        - DNN_NI0=oai
        - TYPE0=IPv4
        - DNN_RANGE0=12.1.1.2 - 12.1.1.253
        - NSSAI_SST0=1
```

```
        - NSSAI_SD0=1
        - SESSION_AMBR_UL0=1000Mbps
        - SESSION_AMBR_DL0=1000Mbps
        - DEFAULT_CSCF_IPV4_ADDRESS=127.0.0.1  # only needed when ims is being used
        - ENABLE_USAGE_REPORTING=no # Set yes if UE USAGE REPORTING is to be done at UPF
    depends_on:
        - oai-nrf
        - oai-amf
    volumes:
        - ./healthscripts/smf-healthcheck.sh:/openair-smf/bin/smf-healthcheck.sh
    healthcheck:
        test: /bin/bash -c "/openair-smf/bin/smf-healthcheck.sh"
        interval: 10s
        timeout: 5s
        retries: 5
    networks:
        public_net:
            ipv4_address: 192.168.70.133
oai-spgwu:
    container_name: "oai-spgwu"
    image: oai-spgwu-tiny:develop
    environment:
        - TZ=Europe/Paris
        - PID_DIRECTORY=/var/run
        - SGW_INTERFACE_NAME_FOR_S1U_S12_S4_UP=eth0
        - SGW_INTERFACE_NAME_FOR_SX=eth0
        - PGW_INTERFACE_NAME_FOR_SGI=eth0
        - NETWORK_UE_NAT_OPTION=yes
        - NETWORK_UE_IP=12.1.1.0/24
        - SPGWC0_IP_ADDRESS=192.168.70.133
        - BYPASS_UL_PFCP_RULES=no
        - MCC=208
        - MNC=99
        - MNC03=099
        - TAC=1
        - GW_ID=1
        - THREAD_S1U_PRIO=80
        - S1U_THREADS=8
        - THREAD_SX_PRIO=81
        - SX_THREADS=1
        - THREAD_SGI_PRIO=80
        - SGI_THREADS=8
        - REALM=openairinterface.org
        - ENABLE_5G_FEATURES=yes
        - REGISTER_NRF=yes
        - USE_FQDN_NRF=yes
        - UPF_FQDN_5G=oai-spgwu
        - NRF_IPV4_ADDRESS=192.168.70.130
        - NRF_PORT=80
        - NRF_API_VERSION=v1
        - NRF_FQDN=oai-nrf
        - NSSAI_SST_0=1
        - NSSAI_SD_0=1
        - DNN_0=oai
    depends_on:
        - oai-nrf
        - oai-smf
```

```
        cap_add:
            - NET_ADMIN
            - SYS_ADMIN
        cap_drop:
            - ALL
        privileged: true
        volumes:
            - ./healthscripts/spgwu-healthcheck.sh:/openair-spgwu-tiny/bin/spgwu-healthcheck.sh
        healthcheck:
            test: /bin/bash -c "/openair-spgwu-tiny/bin/spgwu-healthcheck.sh"
            interval: 10s
            timeout: 5s
            retries: 5
        networks:
            public_net:
                ipv4_address: 192.168.70.134
    oai-ext-dn:
        image: trf-gen-cn5g:latest
        privileged: true
        container_name: oai-ext-dn
        entrypoint: /bin/bash -c \
              "ip route add 12.1.1.0/24 via 192.168.70.134 dev eth0; sleep infinity"
        depends_on:
            - oai-spgwu
        networks:
            public_net:
                ipv4_address: 192.168.70.135
networks:
    public_net:
        driver: bridge
        name: demo-oai-public-net
        ipam:
            config:
                - subnet: 192.168.70.128/26
        driver_opts:
            com.docker.network.bridge.name: "demo-oai"
```

C.2 gNB configuration file

```
    Active_gNBs = ( "gNB");
Asn1_verbosity = "none";

gNBs =
(
 {
    ////////// Identification parameters:
    gNB_ID    =  0xe0;
    gNB_name  =  "gNB";

    // Tracking area code, 0x0000 and 0xfffe are reserved values
    tracking_area_code  =  1;
    plmn_list = ({
                  mcc = 208;
                  mnc = 99;
                  mnc_length = 2;
                  snssaiList = (
                    {
```

```
                    sst = 1;
                    sd  = 1;
                  }
                );

                });

    nr_cellid = 12345678L;

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

    ssb_SubcarrierOffset                                    = 0;
    min_rxtxtime                                            = 5;

    do_CSIRS                                                = 1;
    do_SRS                                                  = 1;

     pdcch_ConfigSIB1 = (
      {
        controlResourceSetZero = 12;
        searchSpaceZero = 0;
      }
      );

    servingCellConfigCommon = (
    {
 #spCellConfigCommon

      physCellId                                                = 0;

#  downlinkConfigCommon
    #frequencyInfoDL
      # this is 3600 MHz + 43 PRBs@30kHz SCS (same as initial BWP)
      absoluteFrequencySSB                                        = 641280;
      dl_frequencyBand                                            = 78;
      # this is 3600 MHz
      dl_absoluteFrequencyPointA                                  = 640008;
      #scs-SpecificCarrierList
        dl_offstToCarrier                                         = 0;
# subcarrierSpacing
# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
        dl_subcarrierSpacing                                      = 1;
        dl_carrierBandwidth                                       = 106;
     #initialDownlinkBWP
      #genericParameters
        # this is RBstart=27,L=48 (275*(L-1))+RBstart
        initialDLBWPlocationAndBandwidth                          = 28875; # 6366 12925 12956 28875 1295
# subcarrierSpacing
# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
        initialDLBWPsubcarrierSpacing                              = 1;
      #pdcch-ConfigCommon
        initialDLBWPcontrolResourceSetZero                         = 12;
        initialDLBWPsearchSpaceZero                                = 0;

  #uplinkConfigCommon
     #frequencyInfoUL
      ul_frequencyBand                                          = 78;
```

```
      #scs-SpecificCarrierList
      ul_offstToCarrier                                       = 0;
# subcarrierSpacing
# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
      ul_subcarrierSpacing                                    = 1;
      ul_carrierBandwidth                                     = 106;
      pMax                                                    = 20;
     #initialUplinkBWP
      #genericParameters
        initialULBWPlocationAndBandwidth                      = 28875;
# subcarrierSpacing
# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
        initialULBWPsubcarrierSpacing                         = 1;
      #rach-ConfigCommon
        #rach-ConfigGeneric
          prach_ConfigurationIndex                            = 98;
#prach_msg1_FDM
#0 = one, 1=two, 2=four, 3=eight
          prach_msg1_FDM                                      = 0;
          prach_msg1_FrequencyStart                           = 0;
          zeroCorrelationZoneConfig                           = 13;
          preambleReceivedTargetPower                         = -96;
#preamblTransMax (0...10) = (3,4,5,6,7,8,10,20,50,100,200)
          preambleTransMax                                    = 6;
#powerRampingStep
# 0=dB0,1=dB2,2=dB4,3=dB6
        powerRampingStep                                      = 1;
#ra_ReponseWindow
#1,2,4,8,10,20,40,80
        ra_ResponseWindow                                     = 4;
#ssb_perRACH_OccasionAndCB_PreamblesPerSSB_PR
#1=oneeighth,2=onefourth,3=half,4=one,5=two,6=four,7=eight,8=sixteen
        ssb_perRACH_OccasionAndCB_PreamblesPerSSB_PR          = 4;
#oneHalf (0..15) 4,8,12,16,...60,64
        ssb_perRACH_OccasionAndCB_PreamblesPerSSB             = 14;
#ra_ContentionResolutionTimer
#(0..7) 8,16,24,32,40,48,56,64
        ra_ContentionResolutionTimer                          = 7;
        rsrp_ThresholdSSB                                     = 19;
#prach-RootSequenceIndex_PR
#1 = 839, 2 = 139
        prach_RootSequenceIndex_PR                            = 2;
        prach_RootSequenceIndex                               = 1;
        msg1_SubcarrierSpacing                                = 1,
# restrictedSetConfig
# 0=unrestricted, 1=restricted type A, 2=restricted type B
        restrictedSetConfig                                   = 0,

        msg3_DeltaPreamble                                    = 1;
        p0_NominalWithGrant                                   =-90;

# pucch-ConfigCommon setup :
# pucchGroupHopping
# 0 = neither, 1= group hopping, 2=sequence hopping
        pucchGroupHopping                                     = 0;
        hoppingId                                             = 40;
        p0_nominal                                            = -90;
```

```
# ssb_PositionsInBurs_BitmapPR
# 1=short, 2=medium, 3=long
      ssb_PositionsInBurst_PR                                    = 2;
      ssb_PositionsInBurst_Bitmap                                = 1;

# ssb_periodicityServingCell
# 0 = ms5, 1=ms10, 2=ms20, 3=ms40, 4=ms80, 5=ms160, 6=spare2, 7=spare1
      ssb_periodicityServingCell                                 = 2;

# dmrs_TypeA_position
# 0 = pos2, 1 = pos3
      dmrs_TypeA_Position                                        = 0;

# subcarrierSpacing
# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
      subcarrierSpacing                                          = 1;


  #tdd-UL-DL-ConfigurationCommon
# subcarrierSpacing
# 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
      referenceSubcarrierSpacing                                 = 1;
      # pattern1
      # dl_UL_TransmissionPeriodicity
      # 0=ms0p5, 1=ms0p625, 2=ms1, 3=ms1p25, 4=ms2, 5=ms2p5, 6=ms5, 7=ms10
      dl_UL_TransmissionPeriodicity                              = 6;
      nrofDownlinkSlots                                          = 7;
      nrofDownlinkSymbols                                        = 6;
      nrofUplinkSlots                                            = 2;
      nrofUplinkSymbols                                          = 4;

      ssPBCH_BlockPower                                          = -25;
  }

  );

# Dedicated Serving Cell Configuration
servingCellConfigDedicated = ({
  # BWP-Downlink
    # BWP 1 Configuration
      dl_bwp-Id_1 = 1;
      dl_bwp1_locationAndBandwidth = 28875; // RBstart=0, L=106 (40 MHz BW)
      # subcarrierSpacing
      # 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
      dl_bwp1_subcarrierSpacing = 1;

    # BWP 2 Configuration
      dl_bwp-Id_2 = 2;
      dl_bwp2_locationAndBandwidth = 13750; // RBstart=0, L=51 (20 MHz BW)
      # subcarrierSpacing
      # 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
      dl_bwp2_subcarrierSpacing = 1;

    # BWP 3 Configuration
      dl_bwp-Id_3 = 3;
      dl_bwp3_locationAndBandwidth = 6325; // RBstart=0, L=24 (10 MHz BW)
      # subcarrierSpacing
```

```
      # 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
      dl_bwp3_subcarrierSpacing = 1;

    firstActiveDownlinkBWP-Id = 1;  #BWP-Id
    defaultDownlinkBWP-Id     = 1;  #BWP-Id

    # bwp-InactivityTimer                ENUMERATED {ms2, ms3, ms4, ms5, ms6, ms8, ms10, ms20, ms30,
    #                                                ms40,ms50, ms60, ms80,ms100, ms200,ms300, ms500,
    #                                                ms750, ms1280, ms1920, ms2560, spare10, spare9, spare8,
    #                                                spare7, spare6, spare5, spare4, spare3, spare2, spare1 }

  # UplinkConfig
    # BWP-Uplink
      # BWP 1 Configuration
        ul_bwp-Id_1 = 1;
        ul_bwp1_locationAndBandwidth = 28875; // RBstart=0, L=106 (40 MHz BW)
        # subcarrierSpacing
        # 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
        ul_bwp1_subcarrierSpacing = 1;

      # BWP 2 Configuration
        ul_bwp-Id_2 = 2;
        ul_bwp2_locationAndBandwidth = 13750; // RBstart=0, L=51 (20 MHz BW)
        # subcarrierSpacing
        # 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
        ul_bwp2_subcarrierSpacing = 1;

      # BWP 3 Configuration
        ul_bwp-Id_3 = 3;
        ul_bwp3_locationAndBandwidth = 6325; // RBstart=0, L=24 (10 MHz BW)
        # subcarrierSpacing
        # 0=kHz15, 1=kHz30, 2=kHz60, 3=kHz120
        ul_bwp3_subcarrierSpacing = 1;

      firstActiveUplinkBWP-Id = 1;  #BWP-Id
  }
);

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


    ////////// AMF parameters:
    amf_ip_address      = ( { ipv4       = "192.168.70.132";
                              ipv6       = "192:168:30::17";
                              active     = "yes";
                              preference = "ipv4";
                            }
                          );


    NETWORK_INTERFACES :
```

```
    {
        GNB_INTERFACE_NAME_FOR_NG_AMF            = "enxf8e43bbb57d5"
        GNB_IPV4_ADDRESS_FOR_NG_AMF              = "192.168.1.15/24";
        GNB_INTERFACE_NAME_FOR_NGU               = "enxf8e43bbb57d5"
        GNB_IPV4_ADDRESS_FOR_NGU                 = "192.168.1.15/24";
        GNB_PORT_FOR_S1U                         = 2152; # Spec 2152
    };

  }
);

MACRLCs = (
{
  num_cc                      = 1;
  tr_s_preference             = "local_L1";
  tr_n_preference             = "local_RRC";
  pusch_TargetSNRx10          = 150;
  pucch_TargetSNRx10          = 200;
  ul_prbblack_SNR_threshold   = 10;
  ulsch_max_frame_inactivity  = 0;
}
);

L1s = (
{
  num_cc = 1;
  tr_n_preference       = "local_mac";
  prach_dtx_threshold   = 120;
  pucch0_dtx_threshold  = 100;
  ofdm_offset_divisor   = 8; #set this to UINT_MAX for offset 0
}
);

RUs = (
{
  local_rf       = "yes"
  nb_tx          = 1
  nb_rx          = 1
  att_tx         = 12;
  att_rx         = 12;
  bands          = [78];
  max_pdschReferenceSignalPower = -27;
  max_rxgain                    = 114;
  eNB_instances  = [0];
  #beamforming 1x4 matrix:
  bf_weights = [0x00007fff, 0x0000, 0x0000, 0x0000];
  clock_src = "internal";
}
);

THREAD_STRUCT = (
{
  #three config for level of parallelism "PARALLEL_SINGLE_THREAD", "PARALLEL_RU_L1_SPLIT", or "PARALLEL_RU_L1
  parallel_config    = "PARALLEL_SINGLE_THREAD";
  #two option for worker "WORKER_DISABLE" or "WORKER_ENABLE"
  worker_config      = "WORKER_ENABLE";
}
```

```
);

rfsimulator :
{
  serveraddr = "server";
  serverport = "4043";
  options = (); #("saviq"); or/and "chanmod"
  modelname = "AWGN";
  IQfile = "/tmp/rfsimulator.iqs";
};

security = {
  # preferred ciphering algorithms
  # the first one of the list that an UE supports in chosen
  # valid values: nea0, nea1, nea2, nea3
  ciphering_algorithms = ( "nea0" );

  # preferred integrity algorithms
  # the first one of the list that an UE supports in chosen
  # valid values: nia0, nia1, nia2, nia3
  integrity_algorithms = ( "nia2", "nia0" );

  # setting 'drb_ciphering' to "no" disables ciphering for DRBs, no matter
  # what 'ciphering_algorithms' configures; same thing for 'drb_integrity'
  drb_ciphering = "yes";
  drb_integrity = "no";
};

log_config :
{
  global_log_level                      ="info";
  hw_log_level                          ="info";
  phy_log_level                         ="info";
  mac_log_level                         ="info";
  rlc_log_level                         ="info";
  pdcp_log_level                        ="info";
  rrc_log_level                         ="info";
  ngap_log_level                        ="debug";
  f1ap_log_level                        ="debug";
};
```

C.3 UE Configuration file

```
uicc0 = {
imsi = "208990000000001";
key = "fec86ba6eb707ed08905757b1bb44b8f";
opc= "C42449363BBAD02B66D16BC975D77CC1";
dnn= "oai";
nssai_sst=1;
nssai_sd=1;
}
```