Live eNodeB Container Migration in LTE Mobile Networks

Master thesis
Faculty of Science, University of Bern

handed in by

Jesutofunmi Ademiposi Ajayi

2019

Supervisor

Prof. Dr. Torsten Braun

MASTER IN COMPUTER SCIENCE

Live eNodeB Container Migration in LTE Mobile Networks

Technical Requirements, Design, and Implementation of the Live eNodeB Container Migration in LTE Mobile Networks

Master Thesis

Jesutofunmi Ademiposi Ajayi

University of Bern

September 2019

UNIVERSITÄT BERN

UNIVERSITÉ DE NEUCHÂTEL

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

"A Good Craftsman Never Blames His Tools"

Abstract

This master thesis aims to provide the technical requirements, design, and implementation for the live migration of an Evolved Node B (eNodeB) in Long Term Evolution (LTE) Mobile Networks. We use the OpenAirInterface (OAI) implementation of an LTE network to carry out our research. We focused on the eNodeB, which provides the Radio Access Network (RAN) in a containerized environment using Docker and the necessary steps that are required to perform a live migration of the container among physical machines. We show the challenges of successfully performing a live eNodeB container migration, as well as the current limitations of migration tools. Our evaluation shows that running the eNodeB inside a Docker container or directly on the machine has significant impact on the performance of the application.

Prof. Dr. Torsten Braun, Communication and Distributed Systems, Institute of Computer Science, University of Bern, Supervisor

Dr. Eryk Schiller, Communication Systems Group, Department of Informatics, University of Zurich, Assistant

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Due to the increasing number of mobile users worldwide, there has been a continuous rise in the demand for wireless data use, primarily due to an increasing number of 3G and 4G mobile phones [2, 3]. It is believed that by 2021, an estimated 10 billion devices could be connected to mobile networks all over the world [4]. This implies a growing pressure on Mobile (Virtual) Network Operators (M(V)NO) to provide better services whilst keeping their CAPital and OPerating EXpenses (CAPEX/OPEX) low. It has become evident that keeping up with increasing demand will require mobile network operators to find new ways of increasing network capacity while also improving their service offerings in terms of the Quality of Service (QoS) and system capacity [2].

The management of IT infrastructures, platforms, and applications in the Everything as a Service (XaaS) manner allows for the significant cost reduction in terms of CAPEX and OPEX. For example, no upfront investment is required as resources are traded on-demand, thus zero-CAPEX; no risk of ill-estimated CAPEX versus revenue estimations since resources and thus infrastructures, platforms as well as applications scale continuously up, etc. When the XaaS operational model is applied to the telecommunications industry, the significant benefits of the XaaS model can counter the effect of the ever-decreasing Average Revenue Per User (ARPU) in the telco ecosystem. Cloud computing and virtualization have stood out as two important technologies that can be used to create new opportunities that will meet the MNOs goals. Cloud computing enables ubiquitous and on-demand access to a shared pool of scalable computational resources (i.e. processing, networking and storage), while virtualization techniques such as Network Function Virtualization (NFV) and Software Defined Networking (SDN) both use

network abstraction to virtualize network functions and enable network programming and intelligence.

The emergence of the Central/Cloud Radio Access Network (C-RAN/Cloud-RAN) as an advanced mobile network architecture that can be deployed to leverage features such as network slicing, statistical multiplexing, energy efficiency, and higher capacity, has provided one possible way to meet this challenge [4]. C-RAN systems replace traditional Base Stations (BS) where distributed (passive) radio elements, such as the Remote Radio Head (RRH), are connected to a centralized baseband processing pool. In C-RAN systems, the baseband processing pool is located at a remote location (i.e. a Data Center), whilst the front-end entity, RRH, are located at sites that are closer to users [5]. Centralized baseband processing brings about several advantage such as: lower energy consumption costs due to the reduced number of sites, easy software upgrades and maintenance, as well as performance improvements for multi-cell signal processing (due to increased spectral efficiency from joint spatio-temporal processing of radio signals) [6]

1.2 Motivation

The transition from Fourth-generation (4G) networks, to Fifth-generation (5G) networks, is likely to see an increase in the uses and applications of cloud computing and similar virtualization-like techniques, in the mobile telecommunications industry.

Already, current developments [7] have shown it is possible to virtualize and deploy parts of a mobile network, on General purpose Processors (GPPs). Such a deployment is likely to prove cost effective for MNOs as they can centralize baseband processing and improve their service offerings. Using cloud computing and virtualization, we can run the EPC and RAN on GPPs rather than expensive hardware. They also provide us with a means to scale and share computational resources based on the needs of the mobile network, and ensure the mobile network can handle different cell variations by applying techniques such as load balancing.

With this thesis, we aim to deliver a cloudified Long Term Evolution (LTE) mobile telephony infrastructure. We also look to address the issue of load balancing for handling different cell variations by performing a live migration of a container running a RAN service. Our RAN is deployed as a real-time service which offers direct access to real-time radio information (i.e. radio status, network statistics) for low-latency, high bandwidth services deployed at the network edge [7], and therefore it is important that the migration of such a service does not have a significant impact on the users that are connected to it (i.e. negligible network down time during migration).

We believe that performing a live migration of a containerized RAN service will bring us a step closer to having a fully cloudified LTE network that can handle various cell loads and other instances of poor network performance.

1.3 Goal

The goal of this thesis is to provide the technical requirements, design decisions, and implementation details of the live eNodeB container migration in Long Term Evolution (LTE) Mobile Networks. The eNodeB which provides a radio access network will be deployed in a cloud setup as a containerized application. Our eNodeB is a real-time application that is placed within a Docker [1] container that we look to migrate towards another physical machine. For this migration to be successful, all states of the eNB application before the migration has taken place, need to be maintained after the migration has occurred (i.e. reestablishment of the connection to the core (EPC), and maintaining connection and service information about mobile devices). We believe that by doing this, we can show a way to handle load balancing in cloud data centers.

1.4 Contributions

The main contributions of this thesis can be summarized as follows:

- **A cloudified RAN setup**: We provide a fully operational RAN network that is deployed in a containerized environment. We use virtualization-like techniques such as Docker to containerize the Radio Access Network (eNodeB), opening up the possibility for the application to be migrated between machines.
- **Challenges of Live Migration**: To the best of our knowledge, there hasn't been many attempts to live migrate a containerized eNodeB using a Next Generation Fronthaul Architecture (NGFI), in real time as most attempts have been carried out in simulated environments. Our work looks at the challenges in doing this using a real-time implementation of an LTE system. We explain a key shortcoming in the open-source migration tool we use, Checkpoint and Restore in Userspace (CRIU), and the effect it has on our goal of performing a live migration. We also point out that look into how the MME responds to interruptions from the connection to the UE when the eNodeB is down for a short while.

[1] https://www.docker.com/

- **Evaluation of containerized eNodeB** We compare the performance of the network when the eNodeB is running inside a container as compared to the host. We also look at how the network performs when there's a disruption in the running of the service (i.e. when the eNodeB is shut down and restore after a period of time). Through our performance analysis of the eNodeB application, we show that providing an eNB service as a container does has a significant impact on its performance compared to running it directly on the physical machine in terms of throughput.

1.5 Thesis Structure

In this chapter, we motivated the need for live migration of the eNB to enable load balancing in data centers and introduced other related concepts, the rest of this thesis is organized into 6 chapters. In Chapter 2, we provide a background analysis and present work related to the aims of this thesis, where we explain the concepts, such as Next Generation Fronthaul Interface architecture that our work is based on. In the third chapter we focus on the architecture of our work and explain the technical requirements and design needed to achieve our setup. In chapter 4 we talk about the implementation of our setup. We focus on our efforts to achieve the migration the eNodeB and how we try to use CRIU to achieve this. In Chapter 5 we evaluate our work and present results from the experiments we conducted and our assessment of the eNodeB and UE. We conclude in Chapter 6 and look to possible future work that could improve on our work.

Chapter 2

Related Work

2.1 Introduction

The Long Term Evolution (LTE) Mobile Network is the basis on which our work is built on, hence we present the significant components of this network such as the different interfaces, channels, and protocols that permit communication between the various components of the network. We look at the types of handovers availble, and how the handover procedure is done in LTE. We present the developments to RAN deployments, such as Cloud RAN and Next Generation Fronthaul Interface (NGFI), that are meant provide MNOs better services and increase network capacity, as well the challenges these new deployments bring. Finally, we briefly describe Software Defined Radios and their use in our work.

2.2 Long Term Evolution

The Long Term Evolution (LTE) network is currently one of the most advanced mobile telecommunications technology. It provides high-speed data and voice capabilities and outperforms the previous generation GSM network. It is also one of the most widely deployed mobile network infrastructures across the world, serving billions of users worldwide. The LTE network provides Internet Protocol (IP) connectivity between User Equipment (UE) and a Packet Data Network (PDN), such as the internet, without significant disruption to the end users connectivity during mobility. It offers faster data transmission than previous networks, lower latency, and higher reliability and robustness against unforeseen failures.

The LTE mobile network architecture is comprised of three components: (i) **Evolved Packet Core (EPC)**: which connects user equipment with the internet or a mobile network operators network, (ii) **Radio Access Network (RAN)**: which handles wireless radio connections between user equipment and the base station(s), and (iii) **User Equipment (UE)**: which is essentially any device that sends and receives radio signals and can be used to access the internet.

2.2.1 Evolved Packet Core (EPC)

As previously explained, the EPC provides UEs with a way to connect to the internet or to an MNO/MVNOs network. The EPC has 5 components that enable it to achieve this: Home Subscriber System, Mobility Management Entity, Serving and Packet Gateways, and the Policy Control and Charging Rules Function.

- **Home Subscriber Server (HSS)**: The HSS is a database that contains user-related and subscriber-related information. The server also provides support functions in mobility management, call and session setup, user authentication and access authorization, and is primarily hosted by the mobile network provider. The HSS is able to communicate with the Mobility Management Entity over the s6a interface.

- **Mobility Management Entity (MME)**: The Serving Gateway deals with the User Plane. It is used to transport IP data traffic between the UE and external networks, and serves the UE by routing incoming and outgoing IP packets. The S-GW becomes an anchor point when the UE moves from one base station to another (i.e. in the case of a handover), and it is logically connected to another gateway known as the Packet Data Network Gateway (PDN-GW). It is the point of interconnect between the radio-side and the Evolved Packet Core.

- **Serving Gateway (S GW)**: The Packet Data Network is the point that connects the EPC with external IP networks such as the internet and is essentially a route between the two entities. The PDN-GW performs tasks such as IP address/prefix allocation, as well as policy control and charging. Since the interface of the Serving Gateway is based on the GPRS Tunnelling Protocol (GTP-U), the PDN-GW matches IP data flows towards external networks.

- **Packet Data Network Gateway (PDN GW)**: The MME deals with the control plane. It is responsible for handling signalling related to mobility and security for the Evolved Universal Terrestrial Radio Access Network (E-UTRAN). It is also responsible for the tracking and paging of the UE in idle mode, where the

UE doesnt have an established Radio Resource Control (RRC) connection and is unknown to the network at the cell level. It is the termination point of the Non-Access Stratum (NAS).

- **Policy Control and Charging Rules Function (PCRF)**: The PCRF is a node in the EPC that is in charge of policy control and decision making within the core. It controls the flow-based charging functionalities in the Policy Control Enforcement Function (PCEF) which resides inside the Packet Data Network Gateway (P-DN). The PCRF handles the Quality of Service (QoS) (i.e. bit rates) authoritzation, and determines how the PCEF will treat certain data flows to ensure they are in accordance with the user's subscription profile [8].

2.2.2 Radio Access Network (RAN)

The Radio Access Network is used to provide wireless radio connection between mobile devices and the Base Station (BS), and is mainly made up of the evolved Node B (eNodeB).

- **Evolved Node B (eNodeB)**: An eNodeB provides the radio interfaces for communication in LTE and allows User Equipment to wirelessly connect to the LTE network. It's main purpose is to perform typical functions of a Base Station by providing Radio Resource Management functions such as dynamic resource allocation, eNodeB provisioning, eNB measurement configuration, radio admission control, connection mobility, radio bearer control and inter-cell interference coordination. The eNodeB contains a Baseband Processing Unit (BBU) that processes (baseband) signals, and it can be connected to one or more Remote Radio Units. The eNB is typically connected to the MME for control-plane communication, and to the S & PDN gateways for control-plane and user-plane data transmissions. The eNodeB communicates with the MME and the S-GW over the S1-MME and S1-U interfaces, respectively. This element is also responsible for deciding whether handovers are required. The decision to perform a handover is typically based on measurements sent by the UE, and the eNodeB is responsible for implementing this handover.

2.2.3 User Equipment

A User Equipment is essentially any device that can be used by an end-user for communication. Examples of common UE include: Mobile Phones, and personal computers with mobile broadband adapters (such as a USB dongle).

- **User Equipment (UE)**: The User Equipment typically refers to mobile devices which can be used to connect users to subscription services such as data and voice call, by connecting users to a Base Station (BS). A UE can be identified by either its unique International Mobile Station Equipment Identity (IMEI), or by the use of a Universal Subscriber Identity Module (USIM) which can be physically inserted in the UE and can also be used for authentication, security, and protection of radio transmission between the UE and another RAN component known as the Evolved Node B.

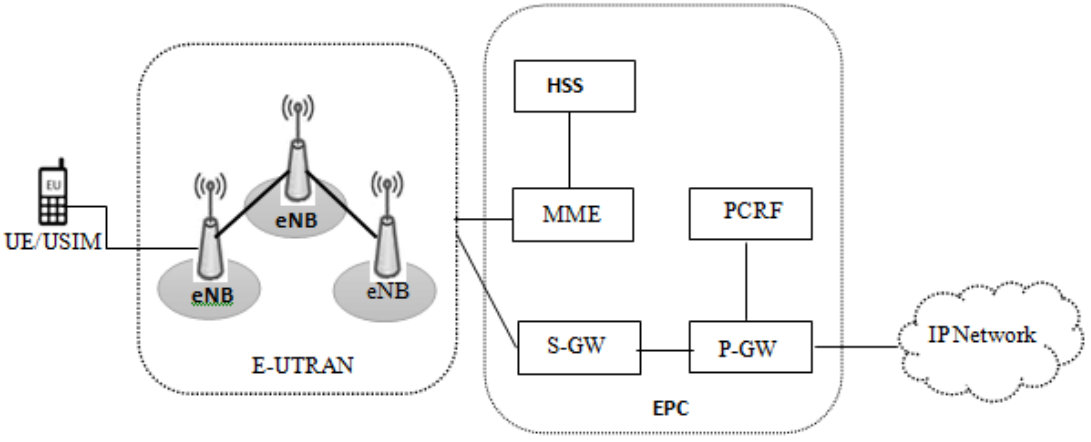

FIGURE 2.1: General LTE architecture

For the purpose of this thesis we refer to the S-GW and PDN-GW as a single component called the Serving Packet Gateway (SPGW). In the EPC implementation we use, the PCRF is not included as a component of the Core Network. Figure 2.1 is an example of the general LTE architecture[1], including the key components and sub-components that have been mentioned (for a more comprehensive look at the interfaces used to facilitate communication between the EPC components and the UTRAN, see [8]).

2.2.4 Channels, Interfaces and Protocols

In this section we describe the channels, interfaces and protocols that allow different elements of the network to communicate with each other. Figure 2.2 shows the protocol structure that allows the UE to communicate with the eNodeB and the MME. Note that since the communication between the UE and eNodeB is done over an air interface, LTE-Uu, it is different from the other links as it handles radio transmissions and related details.

[1] https://www.researchgate.net/figure/LTE-Network-architecture_fig1_318502441

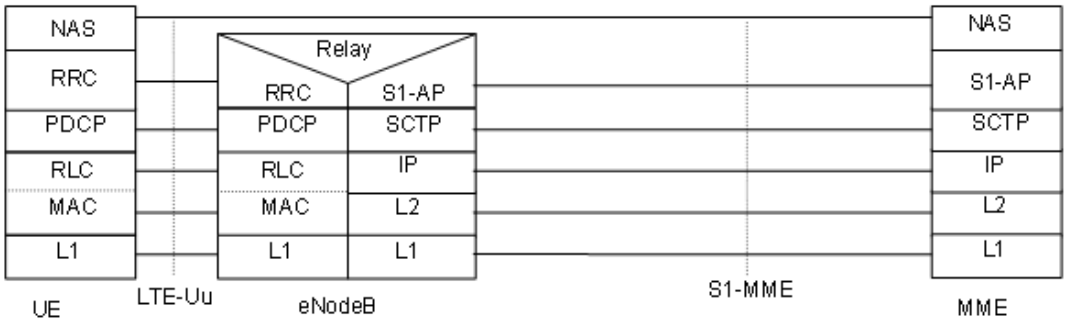

FIGURE 2.2: Control Plane Structure in LTE

2.2.4.1 LTE-Uu Interface

The LTE-Uu interface protocols are divided into two types: user-plane protocols and control-plane protocols which handle tasks such as the request of the service, control of transmission resources and inter/intra eNodeB handovers [9].

- **L1 (Layer 1)**: This is a physical layer. It monitors the downlink (DL) quality and alerts the RRC of any potential problems. [10]
- **MAC (Medium Access Control)**: This layer performs multiplexing and demultiplexing for the uplink and downlink directions respectively. It also controls scheduling of different logical channels.
- **RLC (Radio Link Layer)**: Handles the delivery of data and it's duplicate detection. Can also perform segmentation and concatenation of sent data units.
- **PDCP (Packet Data Convergence Protocol)**: Encrypts IP packets and performs header compression to improve effeciency of over the air transmission. Used to transfer User and Control plane packets to and from upper layers in the stack-/structure.
- **RRC (Radio Resource Control)**: Used to signal exchange between UE and eNodeB entity (i.e. in the case of handovers).
- **NAS (Non Access Spectrum)**: This protocol supports mobility management functionality and user plane bearer activation in LTE. It also handles ciphering and integrity protection of NAS signals. NAS signalling occurs between UE and the MME, with the eNodeB relaying the messages between the entities, not processing them.

2.2.4.2 S1-MME Interface

This is a signalling interface which is used to support functions and procedures that occur between the eNodeB and the MME. Below we give brief descriptions of each protocol.

- **L1 (Layer 1)**: Physical layer which connects the eNodeB and MME together and can be implemented with fixed cabling such as optical fibre.
- **L2 (Layer 2)**: Supports any data link layer protocol, such a MAC, using Ethernet.
- **IP (Internet Protocol)**: In this interface, IP is used to route signalling and user data messages through the EPC.
- **SCTP (Stream Control Transmission Protocol)**: This protocol is used in the control plane and guarantees the delivery of control or signalling messages between the eNodeB and the MME.[11]. It's main features include: association setup, and reliable data delivery.
- **S1-AP (S1-Application Part)**: The S1-AP is the control signalling protocol between the eNodeB and the MME. It fulfils S1 functions such as paging, NAS signaling transport function, error reporting and UE context release.

2.2.4.3 S1-U Interface

This interface is used for communication between the eNodeB and the S GW/PDN GW. It is implements the bottom two layers (L1 and L2) in the S1-MME interface but has two other interfaces: GTP-U and UDP.

- **GTP-U (GPRS Tunneling Protocol User plane)**: The GTP-U tunnel is used to carry IP packets through the core network.
- **UDP (User Datagram Protocol)**: In LTE UDP has the task of carrying signalling messages between specified endpoints.

2.2.4.4 X2 Interface

The X2 interface is used to tunnel user packet data between eNodeBs. It handles load/interference related functions, handovers and influences radio resource management processes in real time. The interface is composed of the control plane and user plane. The control plane (X2-CP) prepares and performs handovers between eNodeBs, while the

user plane (X2-U) is needed for downlink forwarding during a handover. The transport layer of X2-CP is built on SCTP, which functions on top of IP, and UDP on top of IP for X2-U.

2.2.4.5 Channels

In LTE, data and control information is encoded down from the MAC layer to the physical layer (L1) and decoded back from physical layer to the MAC layer to serve both transport and control channels [12]. The main channels are used in downlink (DL) and uplink (UL) to carry data, messages in the protocol stack. Some examples of such channels are listed below:

Downlink (DL)

- **Physical Broadcast Channel (PBCH):** This channels carries system information for UEs that need to access the LTE network,
- **Physical Downlink Control Channel (PDCCH):** The PDDCH is responsible for scheduling information (such as paging and downlink resource scheduling)
- **Physical Downlink Shared Channel (PDSCH):** This channel carries UE-specific data (such as the DL payload)

Uplink (UL)

- **Physical Random Access Channel (PRACH):** A physical channel that is used for random access functions, such as the initiation of a data transfer [13].
- **Physical Uplink Shared Channel (PUSCH):** This is the uplink counter part of the PDSCH mentioned above.
- **Physical Uplink Control Channel (PUCCH):** Handles the Hybrid Automated Repeat ReQuest (HARQ) ACK/NACK in the network [13].

2.3 Handovers in LTE

As stated in Section 2.2, one of the key features of LTE is the ability to afford users free mobility without significant disruption to their connectivity to the internet. One of the ways this is done is through handovers of the UE. There are two main types of handovers: **intra-eNodeB handover** and **inter-eNodeB handover**. A third type of

handovers (inter Radio Access Networks: inter-RAT) exist, but that's beyond the scope of this thesis.

- **Inter-LTE Handover(using X2/S1)**: Occurs when UE handover is between two cells/eNodeBs connected to the same MME, hence the X2 or S1 interfaces can be used to control this handover.
- **Inter-LTE Handover(without X2)**: Here, the handover over is between two MMEs/S GWs as the source and target cells are located in different networks, which the the X2 interface cannot deal with.
- **Intra-LTE Handover**: In this case the source and target cell, between which the UE(s) will be handed over, reside within the same LTE network. In such a case, the X2 interface is the interface between the two eNodeBs and the EPC is not explicitly involved in the handover as the release of resources is activated by the source eNodeB. If the X2 interface is unavailable the S1 interface provides a way for the handover to occur, when the source and target eNodeB belong to the same MME/S GW.

The handover procedure is mainly executed in three steps/phases:

- **preparation phase**: During this phase the decisions about the need for a cell change and resource reservation are made;
- **execution phase**: In this second stage, the mobile connection to the target eNB entity is established;
- **completion phase**: In the final stage, the establishment of final bearers are configured and old resources are released.

Figure 2.3 below gives a more detailed explanation of the handover procedure, as well as the functions/requests (i.e. X2 Handover requests, HO admission and resource setup) that are executed before, during and after the handover has occurred.

2.4 Virtual Radio Access Networks (vRAN)

The virtualization of the Radio Access Network has emerged as a possible solution for MNOs to improve their services, while keeping costs down. This involves decoupling the software that controls the access network from the underlying hardware, allowing for fast upgrades and scaling to meet varying network traffic demands, the deployment of

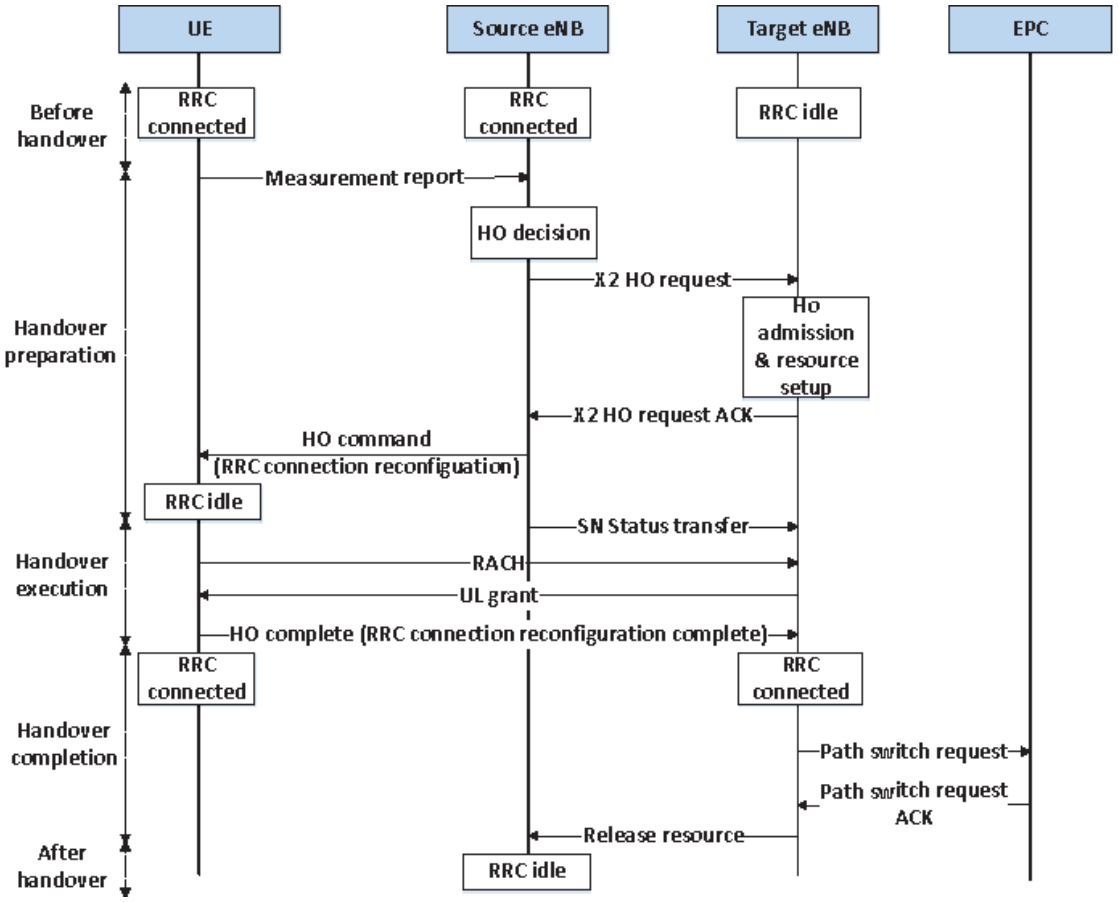

FIGURE 2.3: X2 Handover procedure in LTE/LTE-A Networks [1]

new services automatically, as well as the ability to centralize and pool various resources together. Virtual RANs consume significantly less power than traditional Radio Access Networks that are provided by Base Stations as a result of this abstraction. With Virtual (Centralized) RAN, Baseband Units are not deployed along with physical Base Stations at a remote location, they are decoupled and moved to a centralized processing pool that includes other BBUs. Remote Radio Heads still remain at their current locations in the network (i.e. at Physical locations). Centralized BBU pooling and processing of radio signals could lead to more sophisticated joint spatiotemporal processing of the signals, and possibly improve spectral efficiency [2]. VRANs also support advanced features of LTE-A, such as Coordinated MultiPoint Operation (CoMP)[14] and enhanced Inter-Cell Interference Coordination (eICIC), which are considered important features in small cell deployments [2]. Having a central location that can cater to many different users also benefits MNOs as they can offer better Service Level Agreements (SLAs), as the processing pool is closer to users and therefore the response time for services is shorter if the requested data has already been cached at the processing pool [2].

2.4.1 Cloud Radio Access Networks

Recent developments have seen cloud services being transformed from monolithic architectures towards a microservice oriented architecture in which services are a collection of microservices running/executing some set of functions [15] [16]. The microservice architecture brings better benefits in the cloud computing paradigm such as maintainability, flexibility, scalability and reduced complexity. As a single microservice can be deployed, scaled and operated independently of the whole service, this makes it very flexible in regards to the geographic distribution of computational tasks [17]. The microservice architecture also support the ETSI NFV architecture[2], where a Virtual Network Function, such as eNodeB/RAN, can be considered a service.

RAN-as-a-Service (RANaaS) is seen as a possibly new cloud computing paradigm, in which a radio access network is delivered as a pay as you go service that has been instantiated on top of a cloud infrastructure [3], giving us a Cloudified Radio Access Network (Cloud RAN).

Effectively, Cloud RAN takes the concept of Virtual/Central RAN deployments and adapts them for the cloud computing paradigm and other virtualization-like tools. This has the advantage of giving MNOs even better opportunities for scalability and resource allocation, as it can support multi-layer, ultra-dense operations in many different deployment scenarios [18]. The Cloud RAN architecture also allows for the use of Network Function Virtualization (NFV) and Software Defined Networking (SDN) techniques, and can take advantage of data center processing capabilities such as coordination, centralization and virtualization in mobile networks[19]. It is also believed that the adaptation and development of Cloud RANs are set to play an important role in the upcoming Fifth Generation (5G) mobile networks. Cloudification of the radio access network means a move away from specific expensive hardware to more general purpose computing platforms, as well as other benefits including: load balancing and rapid deployment and service provisioning. Deploying RAN into the cloud is not a particularly new concept, however [20] showed that such a setup could save up to 71% in terms of power consumption.

2.4.2 Next Generation Fronthaul Interface

Recent developments in Central RAN have focused on redefining the current architecture of Central RAN deployments. In current C-RAN deployments, in-phase and quadrature (I/Q) samples are carried from the BBU to the RRH, which places very high bandwidth

[2]https://portal.etsi.org/NFV/NFV_White_Paper_5G.pdf

requirements on the fronthaul transport network [3]. Such a architecture will likely struggle to meet scalability and performance requirements of future mobile networks, such as the upcoming Fifth Generation (5G) mobile network [3].

The Next Generation Fronthaul Interface (NGFI) is an architecture proposed by China Mobile [20], that focuses on splitting the radio stack between two main components, namely the BBU and the RRH, and therefore redefining the positioning of the eNodeB stack components (e.g. physical layer (PHY) Reception (RX)/transmission (TX)) between the BBU/RRH. The two components can be connected to each other through Ethernet or IP interfaces. In this architecture, the BBU is redefined as the Remote Cloud Center (RCC), and resides in a Cloud (Data) Center with multiple other BBUs. The RRH is redefined as the Radio Remote System. This architecture allows for point-to-multipoint connections between RCC and RRH, and for a third component called the Radio Aggregator Unit (RAU), to control multiple RRHs that may be operating in different bands and with different coverages. Figure 2.2[3] shows how the eNodeB protocol stack is split using this architecture.

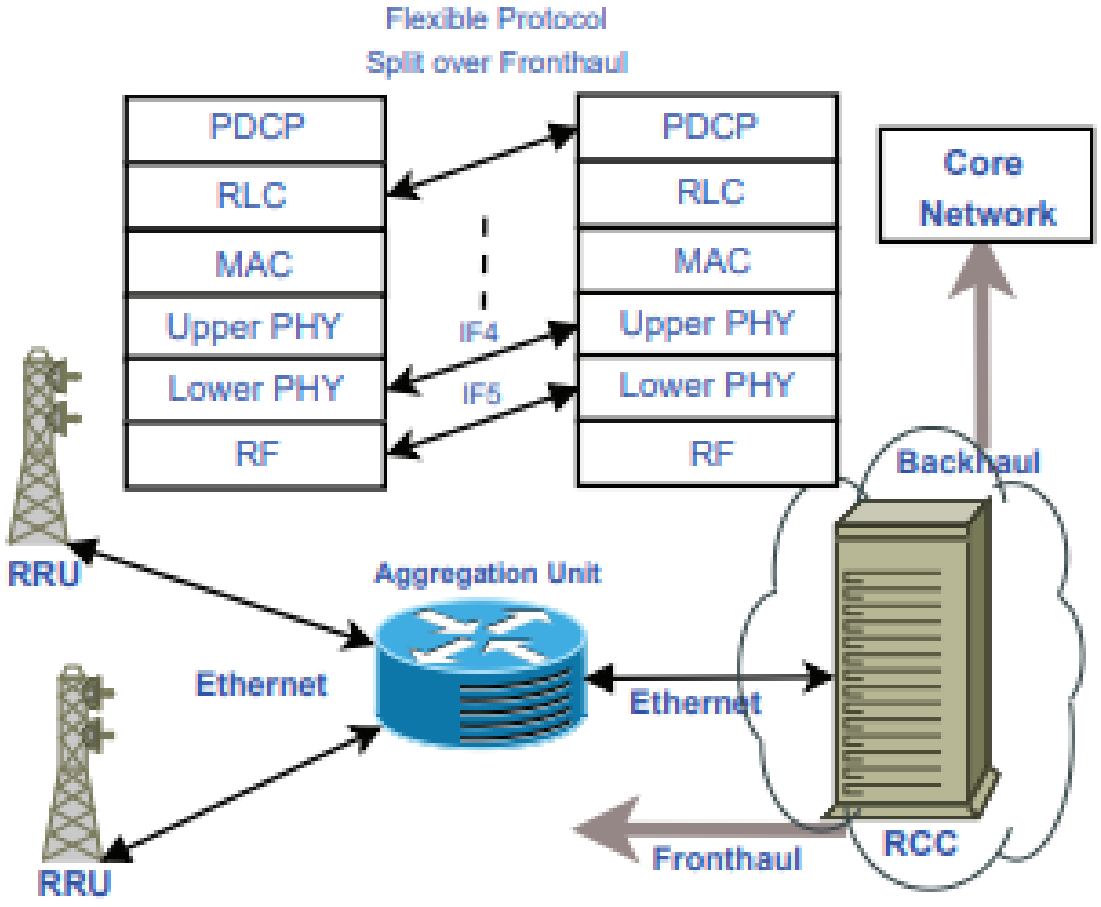

FIGURE 2.4: NGFI based C-RAN architecture with protocol splits

[3]http://www.eurecom.fr/en/publication/5079/download/comsys-publi-5079.pdf

The functional split established by the NGFI architecture makes Central RAN deployment more flexible. There are 2 methods on how to perform this split: **IF5, IF4**.

- **IF5/IF4.5**: This split is similar to the traditional RRH-BBU interface, and transports baseband time domain I/Q samples between the BBU and RRH. The I/Q samples are exchanged between the RCC and RRH as Ethernet samples across the fronthaul link. [21]
- **IF4**: The IF4 split is done at the input (RX) and output (TX) of the OFDM symbol generator (i.e. frequency domain signals), and transports transmitted or received resource elements in usable channel bandwidth. Specifically, the RCC and RRH swap I/Q samples across the fronthaul link. With this split, the RRHs are responsible for some of the lower PHY layer processing, namely Fast Fourier Transform (FFT) and Inverse Fourier Transform (IFFT). A-law compression is also applied to the I/Q data that is transported across the Ethernet fronthaul links. This is done to reduce the fronthaul link capacity requirement to 28% of the time-domain I/Q split case. Therefore an RRH with a bandwidth of 20MHz, can be provisioned using a 1 Gbps link between RRH and RCC. [21]

2.5 OpenAirInterface

OpenAirInterface[4] (OAI) is an open source software project that implements 3GPP technology on general purpose computing hardware. It is the first open source software based implementation of an LTE system which contains the full protocol stack of the 3GPP standard, which includes the Evolved Universal Terrestrial Radio Access Network (E-UTRAN) and the Evolved Packet Core (EPC), as well as the User Equipment (UE). The OAI platform provides a way to build and customize an LTE RAN[5] (eNB + UE) and EPC[6] (HSS + MME + SPGW) on general purpose commodity x86-based computers to test multiple network configurations, as well as monitor the performance of the network and mobile devices in real time. The receiver and transceiver functions of a typical Base Station (eNodeB) are realized via an SDR card (i.e. ExpressMIMO2, USRP, and LimeSDR [22]) that is connected to a host computer for baseband processing.

OAI has two key features: (1) *Real-time experimentation* (2) *Emulation*. For the purpose of this thesis we use real-time experimentation to carry out all of our work. The details on the other features of the platform as well as our implementation are given below.

[4]https://www.openairinterface.org/
[5]https://gitlab.eurecom.fr/oai/openairinterface5g
[6]https://github.com/OPENAIRINTERFACE/openair-cn

2.5.1 OAI eNodeB

The OAI eNodeB implements the MAC layer which provides the Hybrid Automatic Repeat ReQuest (HARQ) and introduces short deadlines for processing of received signals in the PHY layer. As a result of this deadline, every piece of information received by the eNodeB has to be acknowledged to the transmitter. If this information isn't acknowledged by the receiver a retransmission request has to be sent. Typically, in FDD the retransmission time is 8ms, which essentially means that for every subframe N that is processed, the acknowledgment (ACK) or negative acknowledgment (NACK) of the subframe has to be sent at subframe N+4 and decoded by the transmitter before subframe N+8 is processed [3]. Based on the received ACK/NACK, the transmitter decides which piece of information to send, whether new data in the case of an ACK or missing data in the case of a NACK.

Table 2.1 below summarizes the key features of OAI in the current release:

Supported Releases	Rel-8.6, part of Rel-10
Duplexing modes	FDD, TDD
Carrier bandwidths	5, 10 and 20 MHz
Tx modes	1 (SISO), 2, 4, 5 and 6 (MIMO 2x2)
DL channels	PSS, SSS, PBCH, PCFICH, PHICH, PDCCH, PDSCH, PMCH
Multi-RRU support	Over air synch b/w multi RRU in TDD
UL channels	PRACH, PUSCH, PUCCH (format 1/1a/1b), SRS, DRS
HARQ Support	

TABLE 2.1: Main OAI Features

OAI's eNodeB contains the full protocol stack (physical (PHY), Medium Access Control (MAC), Radio Link Control (RLC), and Radio Resource Control (RRC) and Packet Data Convergence Protocol (PDCP) layers), and it can be configured for both Time Division Duplex (TDD) and Frequency Division Duplex (FDD). Currently, the OAI eNodeB is able to operate at 3 channel bandwidths: 5MHz, 10MHz, and 20MHz and can theoretically achieve throughput of up 70 Mbps on the DL and 35 Mbps on the UL. The channels, interfaces and protocols mentioned in 2.2.4 are all implemented in OAI

as means of communication between the eNodeB and the UE and the eNodeB and the MME/S GW.

For an even more comprehensive look at the OpenAirInterface platform and its current development refer to [22].

2.6 Software Defined Radio (SDR)

Software Defined Radios (SDRs) are essentially any radios in which some or all physical layer functions are software defined. Centralized RAN and Cloud RAN can therefore be considered as SDR Systems running on general purpose IT platforms and the cloud, respectively. Typical SDRs have two components: Software and Hardware, where the software component handles functions such as signal processing, while the hardware component consists of the radio front-end (i.e. SDR cards like the ETTUS USRP, Lime SDR, Blade RF and ExpressMIMO2). The radio front-end is primarily used to transmit and receive radio signals and can run on dedicated hardware platforms such as General Purpose Processors (GPPs) or other general purpose computing platforms (x86). In this thesis we use a USRP B210 board (via UHD) for our setup.

Chapter 3

Radio Access Network Architecture

3.1 Introduction

Based on the related work and the OAI platform described in Chapters 2 and 3, we describe the design of our main architecture (which is based on NGFI) in this section. The architecture and design of our setup is born from the aim of creating a Cloudified Radio Access Network (containerized eNodeB) in order to fulfill our aim of a live migration. Containerization is a tool that can be used to take full advantage of the benefits of cloud paradigm, and achieve RAN as a Service (RANaaS). There are different tools and platforms we used in our setup, including: Docker, Pipework, Open vSwitch, and CRIU. To enable unfiltered communication and access between the container eNodeB and the bridge we created to direct traffic to and from the EPC, we needed to bind our container to the bridge. In this chapter, we explain how Docker networks failed to provide this functionality and how we use the Pipework tool to overcome this challenge. Meeting the hard processing deadlines (e.g. HARQ) in LTE is one of the challenges of running BBU processing on GPPs, to account for this we made some modifications to the hosts and configure them for real time performance, we describe this in section 4.4.

3.2 General Architecture

Our RAN setup runs on 3 machines, where each machine hosts a different component of the radio network: eNB, RRH and UE (USB dongle). The EPC we use and it's components are hosted on machine in a different location at the Communication and

Distributed Systems building. Figure 3.1 depicts how we replicate the NGFI architecture proposed by [3], which splits the Base Station functionality between the BBU and the RRH. This setup has the advantage of reducing the fronthaul data requirements, which is important for such a deployment as explained in the previous chapter. Table 3.1 shows the hardware specifications of our 3 host machines.

Figure 3.1 shows the USRP B210 board connected to the telmat duplexer radio (band 7), using SMA cables, and the antenna.

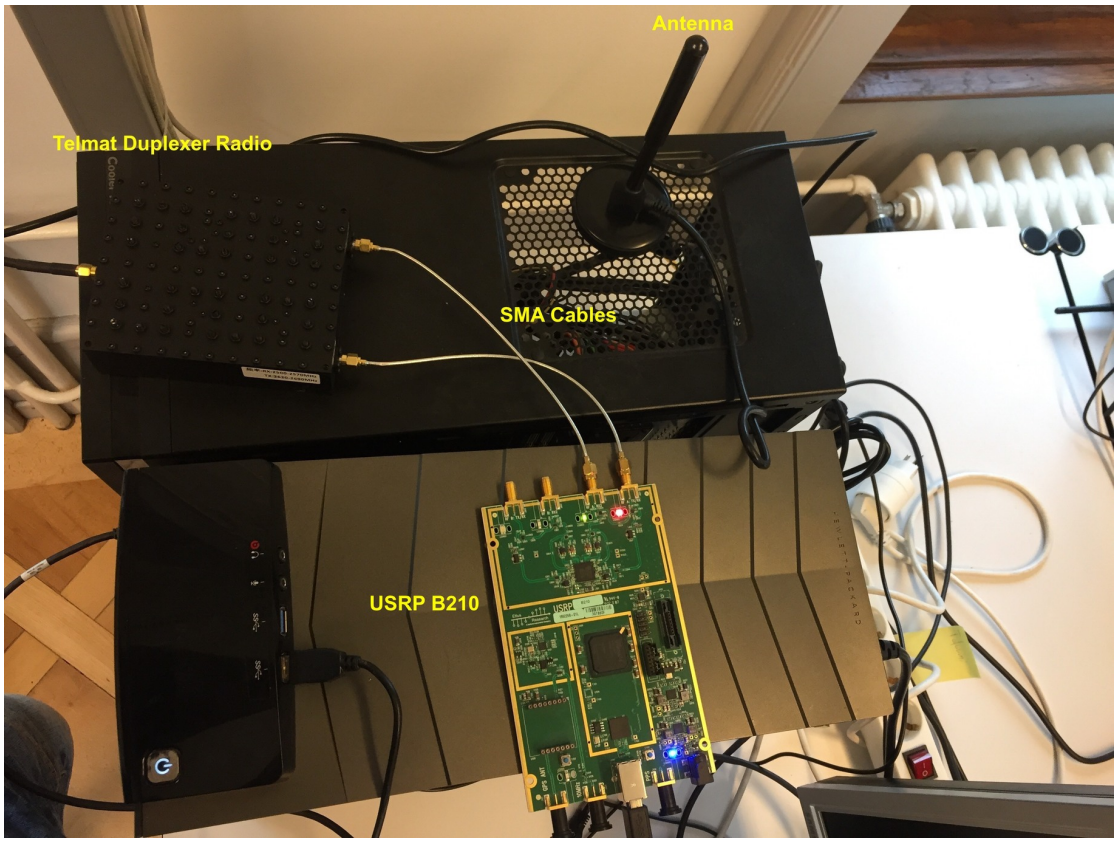

FIGURE 3.1: USRP B210, Telmat Duplexer Radio and Antenna configured at the RRU

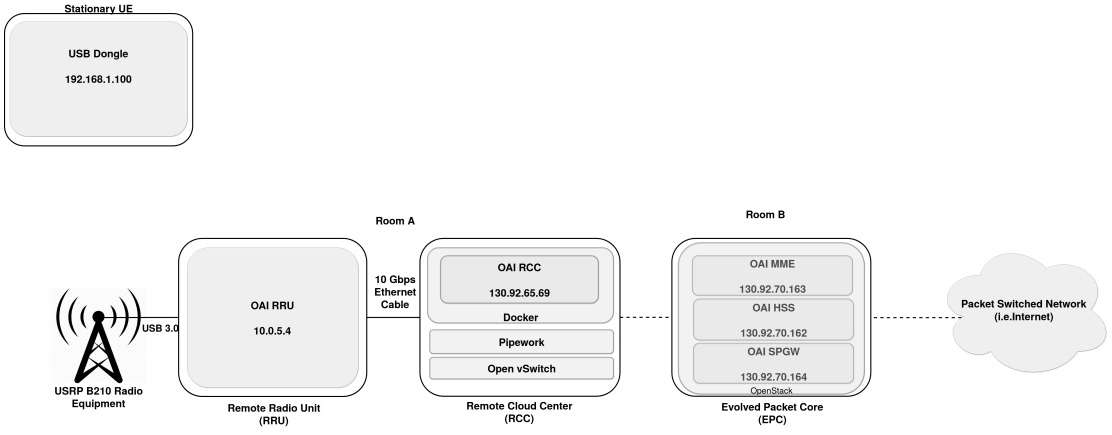

FIGURE 3.2: Central RAN architecture

		Host		
		Farnsworth (eNodeB)	Silverter (RRU)	Host 3 (UE)
	Item	Value	Value	Value
Hardware	CPU	Intel Core i7-4790 CPU @ 3.60GHz, 32GB RAM	Intel Core i7-3770 @ 3.40GHz, 16GB RAM	Intel Core i5-2400 @ 3.1 GHz, 8GB RAM
	USRP	N/A	Ettus Research USRP B210	N/A
Software	OS	64-bit Ubuntu 16.04 LTS with low-latency kernel 4.15.0	64-bit Ubuntu 16.04 LTS with low-latency kernel 4.4.0	64-bit Ubuntu 16.04 LTS with generic kernel 4.14.24
	OAI	Master Branch (v1.0.0)	Master Branch (v1.0.0)	N/A
Parameters	Duplex mode FDD			
	Transmission mode TM1(SISO)			
	Carrier frequency 2.6/2.7 GHz (band7)			
	Cyclic Prefix: Normal			
	System bandwidths: 5 MHz, 10 MHz, 20 MHz			
	Modulation schemes: QPSK,16-QAM,64-QAM			

TABLE 3.1: Cloud-RAN System: Host Specifications

3.3 Containerization

As mentioned 2.4.1, Cloud RAN will enable MNOs to provide better, improved services, and increase the capacity of their networks. In this paradigm we deploy the RAN as a Service (RANaaS), which is made up of a host or microservices that carry out different functions and operate in a containerized or virtualized environment (such as Docker of Kernel Virtual Machines (KVM)). With this in mind, we deploy our OAI eNodeB in a container delivering it as a service that can take advantage of the benefits mentioned in chapter 2. Specifically, we use Docker as the containerization tool to deliver this service.

3.3.1 Docker

Our background research showed that there have already been efforts to use virtualization/containerization tools such as KVM LXC, and Docker to containerize an eNB/BBU application. [3] compared the BBU processing budget of a GPP platform on the Downlink and Uplink for different virtualized environments. Their work showed that while the average processing times where close for their chosen virtualized environments, Docker and KVM had higher time variations compared to GPP and LCX, and even moreso when there was an increase in the PRB and Modulation Coding Scheme (MCS) used by the BBU. However for our work we do not necessarily consider the effects of these results. Also note that containers like Docker are built on modern kernel features (i.e. cgroups, namespace, and chroot) which is very important for ensuring the host scheduler can meet real time deadlines. [3]

Placing the eNodeB application inside of a Docker container means we can improve the provision of the RAN service by making it more scalable to meet various levels of demands. Containerizing the eNodeB also allows us to achieve our aim of migrating the

eNodeB between physical machines, as Docker provides us with an experimental feature to do this.

In general, containers are more efficient for computing as they do not require a virtualization layer, compared to other virtualization platforms (such as Kernel Virtual Machines) and they provide direct access to the underlying host machines resources. Containers can also be migrated between machines with small overhead, unlike Kernel Virtual Machines (KVMs), which suffer from virtualization overhead, which is critical for a real-time applications like the eNodeB.

To be able to use non-standard Docker functions like checkpoint and restore, we set the Docker daemon to experimental mode.

3.4 Host Modifications

To ensure the containerized eNodeB process is able to meet the various processing and scheduling deadlines, such as those imposed by the Hybrid Automatic Repeat reQuest (HARQ) mentioned in 2.5.1, We give the eNodeB real-time prioritization using the Linux's chrt command: *chrt -p -rr 1*, where p is the process id of the eNodeB.

To improve the performance of the hosts where the containers are running (RCC and RRU) even further, we use a userspace governor scaling policy and set the frequency scaling of the machines to their maximum (3.9GHz @ Silvester and 3.60GHz @ Farnsworth) capacity. This ensures frame and subframe processing meet the strict deadlines.

3.5 Networking

Careful consideration was given to our networking setup. We needed to ensure the related components could communicate with each other and that the traffic between our eNodeB and EPC was . We mention those below and justify their use in our setup. We also look at the limitations that they pose and how we overcome them.

It allows for the on-demand establishment of virtual switches among Windows or Linux operating systems. On the north-bound interface, the switch uses OpenFlow to communicate with the controller. It supports various matching rules at different levels of the IP stack as well as many actions (e.g., modification of addresses; tunneling, encapsulation, decapsulation in GRE, VxLAN, etc.) that allow for advanced traffic engineering

3.5.1 Open vSwitch

To direct and control traffic between components in our setup, we use a open-source distributed multi-layer SDN virtual switch called Open vSwitch (OVS)[23]. OVS provides a switching stack for virtualization environments and supports multiple interfaces and protocols including TCP, SCTP, UDP, and GTP. It is able to support many matching rules at all various levels of the IP stack. It can also be used for modification of addresses, tunneling, encapsulation and decapsulation in GRE, etc, which allows for advanced traffic engineering [24]. The virtual switch can also support transparent distribution across a range of physical servers, which means it can be used to connect Virtual Machines or containers between different hosts and across multiple networks. Traffic management is handle by flow table, and in our setup OVS is primarily used to handle communication between the eNodeB and the EPC. We also use it to handle the IP traffic that is exchanged between the UE and the EPC.

The following commands show how we create an OVS bridge at our RCC, and bind a Physical Network Interface Card (NIC) on the host machine to our bridge:

```
$ ovs-vsctl add-br ovs-br
$ ovs-vsctl add-port ovs-br em1
```

It should be noted that ordinarily binding the NIC to the bridge does not ensure that all traffic is passed to the bridge. To solve this we place the physical network interface in promiscuous mode which ensures that all traffic that is passed to the NIC is forwarded to the bridge and hence any traffic that our host receives from the EPC is accessible to the eNodeB that is connected to the OVS bridge.

3.5.2 Container Networking

Using Docker, we were able to create a macvlan bridge network and set the subnet and gateway that it operates in. We place this bridge on top of OVS bridge to enable it to communicate with the switch. However, during our setup we discovered that the bridge network didn't allow for direct communication between the container and the underlying host (or bridge), and so we are unable to ping the host from inside the container and vice-versa. This is due to the fact that macvlan networks do not allow us to ping or communicate with the default namespace IP address, as the traffic is explicitly filtered by the kernel modules to offer the containers additional isolation and security from the underlying host. To navigate this, we use an opensource Software Defined Networking (SDN) tool called Pipework. Pipework is networking tool for Linux (&

Docker) containers[1]. Pipework enables us to connect containers in complex scenarios and is a viable alternative to docker bridge networking. It can be used to create or configure network interfaces and bind containers to them, even when they're running. The tool uses cgroups and namespaces and work with plain containers. Using Pipework, we are able to bind our containers directly onto the previously created OVS bridge and assign them an IP address on this bridge which corresponds to one of the 3 addresses that the MME at the EPC accepts (130.92.65.68 and 130.92.65.69 and 130.92.65.83). This allowed us to communicate directly with the different components of our cloudified EPC, as well as the underlying host. We use the following Pipework command to bind our container to the OvS bridge:

```
$ pipework ovs-br $CONTAINERID 130.92.65.69/24
```

As our setup is based on the NGFI, our RRH host and eNodeB host were connected via a 10Gbps Ethernet cable. To provide communication between the eNodeB container and the RRH host we looked to bind the container to a NIC that detected the cable, however Pipework was unable to do this, as it seemingly can only bind a container to a single physical interface. We solved this by using a docker network and setting the appropriate NIC to it. This way we were able to communicate with the RRH host from inside the container.

The following command show how we created this new network, place it on the appropriate NIC (p1p1) and connect the container to it:

```
$docker network create -d macvlan
--subnet=10.0.5.0/24
--gateway=10.0.5.1
-o parent=p1p1 ethnet

$docker network connect ethnet $CONTAINERID
```

3.6 Checkpoint and Restore in Userspace

Checkpoint and Restore in Userspace (CRIU) is an open-source tool that can be used to achieve the migration of processes between different hosts without a significant amount of downtime. CRIU works by freezing a running application and checkpointing it's state to the disk. The data that has been saved on the disk can be used to restore

[1]https://github.com/jpetazzo/pipework

the application back to it's previous state on the same host, or alternatively, a different host. Another important feature of CRIU is it's ability to preserve network connections across different hosts by dumping active connections (i.e. TCP, UDP) and restoring them later, meaning that user connections are not dropped after the migration process. In Chapter 4 we explain how we used CRIU to dump the a containerized eNodeB as well as an eNodeB running directly on the host/GPP.

Chapter 4

Cloud RAN Implementation and Live Migration

4.1 Introduction

We were able to implement the Cloud RAN using our setup. However, before attempting a live container migration, we decided to first study the behaviour of the eNodeB, and network at large, when the eNodeB is stopped and restored on the same host after a short period. Based on our initial observations, we noticed that the eNodeB crashed when it was stopped/paused for a period of time and hence the connection to the MME was lost. This also caused the connection to the UE to go down instantaneously too.

In this section we talk about the modifications (see Appendix A) we applied to the OAI code to ensure that it doesn't crash immediately after the lte-softmodem(that starts the eNodeB) process is stopped. First we show how we instantiate our RAN and the EPC to deliver a cloudified/virtualized network, including commands to reproduce it based on our architecture. Then we motivate the need for live migration. Next we talk about the procedures required for a live migration in general and the challenges (and solutions) we had trying to perform a dump of the eNodeB process running on the host as well as in the container.

4.2 OAI eNodeB Implementation

To run the OAI eNodeB on our host systems (RCC and RRU), we download the required resources from the git repository[1], and we used the master branch (v1.0.0). We install

[1] https://gitlab.eurecom.fr/oai/openairinterface5g

the appropriate packages and drivers and build the eNodeBs on both hosts using the following commands:

```
$./build_oai -I
$./build_oai -t ETHERNET -c --eNB
$./build_oai -w USRP -c --eNB
```

Here, the -I tag tells the build script to install all required packages, the -t tag sets Ethernet as the transport protocol, and the -w tag specifies which hardware to is being used for the radio frequency board (i.e. USRP in our case).

We make use of the cloudified EPC that had already been deployed on the open-source, cloud computing platform, OpenStack. The 3 main components of the EPC (MME, HSS, and SPGW) are run as Virtual Network Functions. Juju charms [2] are used to deploy and manage this cloudified EPC, as they can be used to scale the network based on demand and available resources. We instantiate each EPC component using the following commands:

- **HSS**: `./srv/openair-cn/scripts/run_hss`
- **MME**: `./srv/openair-cn/scripts/run_mme`
- **SPGW**:`./srv/openair-cn/scripts/run_spgw`

With the EPC up and running, we can now instantiate our RAN service. As previously mentioned our setup follows the NGFI architecture, so our RAN implementation is made up of an eNodeB (started using the lte-softmodem) process at the RCC (where baseband processing occurs), and another one at the RRU (where the USRP board is located), with the two hosts link via a 10Gbps Ethernet cable. We start the lte-softmodem on the two hosts using the following commands:

- **RCC**: `./lte-softmodem -O rcc.band7.tm1.if4p5.50prb.conf`
- **RRH**: `./lte-softmodem -O rru.oaisim.conf`

The configuration files can be found in Appendix A.

[2] https://jaas.ai/

4.3 Live Migration

Live Migration is the process of moving applications or processes (e.g. services) running on a machine from one host to another, with the aim of minimal disruption to the users that are using that application or service. It can be used to solve two problems that are often experienced in data centers - system maintenance and load balancing. In the case of system maintenance, the downtime that a server undergoes can prove costly, especially when its running mission critical applications, such as a RAN, that are being used by many users. While load balancing is usually applied when a host system is overloaded which leads to a fall in the performance of the processes running on it. There a number of uses cases where the live migration of the eNodeB would be critical, including: video streaming, gaming, virtual reality applications, industrial automation, remote medical procedures, and autonomous driving.

Typically, the Source Node (Figure 4.1) is the where the process/service to be migrated is placed before the migration and the Destination Node is where the container will resume the process/service after migration. As Figure 4.1 shows, the live migration process is a 5-step procedure:

- **Freeze**: The application/service is frozen by the migration tool at the source node and blocks memory, processes, file systems and network connections
- **Get the state**: The current state of the memory, processes, file systems and network connections of the application or service are saved as an image or as pages.
- **Copy the state**: The saved imaged containing information about the application state is copied onto the destination node.
- **Restore**: The applications processes, file system and network connections are then restored on the destination node based on the information from copied image.
- **Unfreeze**: The application is then unfrozen at the destination node and runs as normal.

4.4 Live Migration in LTE

The role of the RAN in LTE systems means that it should be readily available (i.e. fault tolerant), and offer a continuously high level of performance (to maintain the end-user's

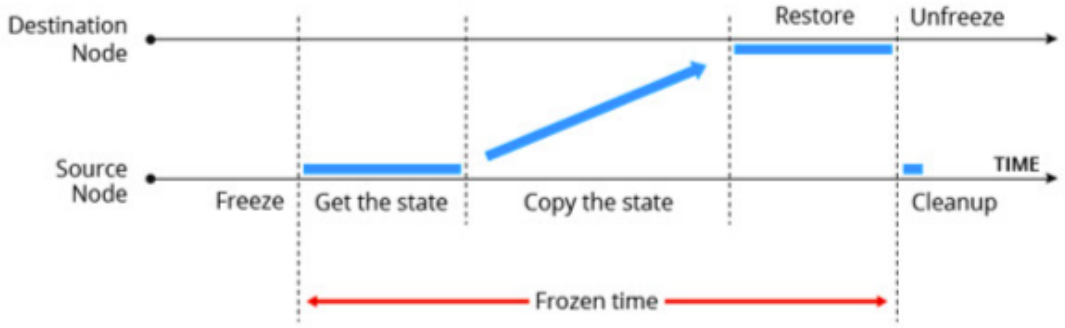

FIGURE 4.1: General Live Process Migration

experience). Deploying the RAN into the cloud means we can take advantage of features such as load balancing that is offered by the infrastructure, to essentially protect the eNodeB in cases where the cloud instance or host it's running on cannot provide the desired availability or performance.

The idea of migrating network resources to perform load balancing is not a particularly new concept in mobile network systems. As mentioned in Section 2.3, the X2 and S1 interfaces are used to perform the handover of UEs for intra-LTE handovers, when an eNodeB cell is overloaded and a nearby eNodeB has fewer users. This often requires the cooperation and synchronization of the source and target eNodeBs (using X2), and at times the reconfiguration of the UE (e.g. inter-LTE handovers with no X2 interface). The migration of eNodeBs to carry out load balancing is, to the best of our knowledge, a relatively novel concept that could have applications in future mobile networks.

Figure 4.2 shows a simple diagram on how a container can be migrated between two hosts using Docker api and the CRIU feature either by the user or by the use of an orchestration tool.

4.5 Challenges in Live Migration

Despite the obvious benefits that would come with a successful eNodeB live migration, there are some challenges to carrying it out. Ideally we would like to have a seamless migration of the containerized eNodeB between hosts without the end-user experiencing little to no interruptions. However, as [25] show, migration of the PUSCH stack introduces a service disruption time of several seconds when using VMware or KVM, and for real-time services (such as video streaming) the maximum interruption time in

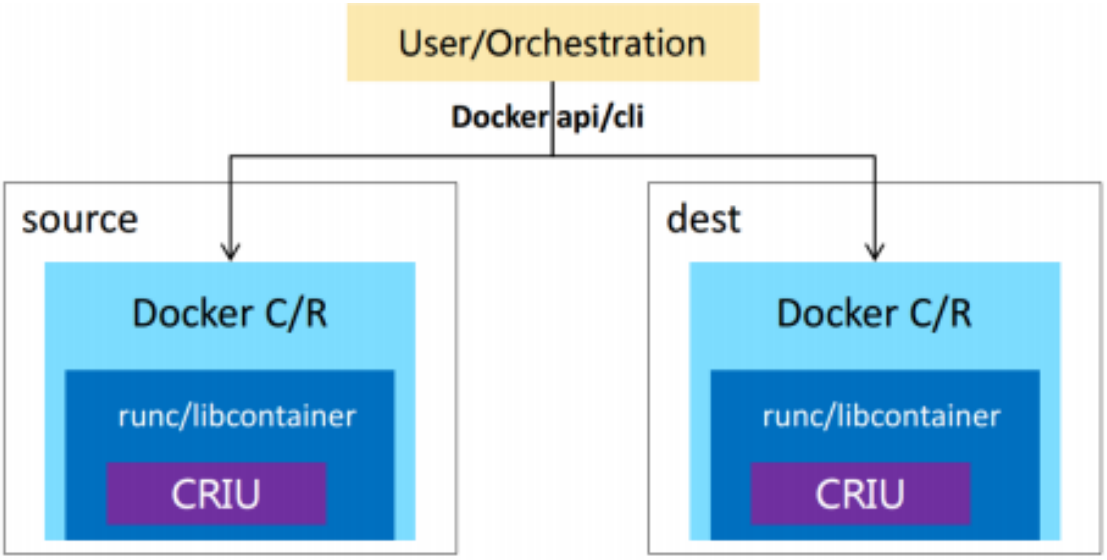

FIGURE 4.2: Container Migration with CRIU

a handover is 300ms, which means the eNB must be migrated within this time for the user to not experience a significant degradation of the service.

4.5.1 CRIU Live Migration

CRIU's Live Migration procedure can be effectively realized in three steps: Checkpoint (Dump), Copy (or pre-copy) and Restore. Figure 4.3 show how this process is achieved between the source host and destination in CRIU. In our implementation, we use a shared file system (NFS) between the source and target host, and therefore don't need the copy phase. This allows us to reduce the downtime of the service as copying the file from one location to the another could further unwanted delays. Our NFS configurations on the RCC and RRU can be found in Appendix A

4.5.2 Checkpoint and Restore

We use CRIU to attempt a checkpoint the eNodeB process running directly on the host machine. The following commands can be used to perform a dump (more on this in section) of the eNodeB running on the host or in a container. The dump saves the process states, and other related information (such as number of threads, sockets, etc) into image files (memory pages) and gives the information in a log file, dump.log. The dump log has over 9000 outputs, hence we do not provide it in this thesis.

```
Host: $criu dump -v4 -o dump.log -t $PID --shell-job
--images-dir=/hostDumps && echo OK
```

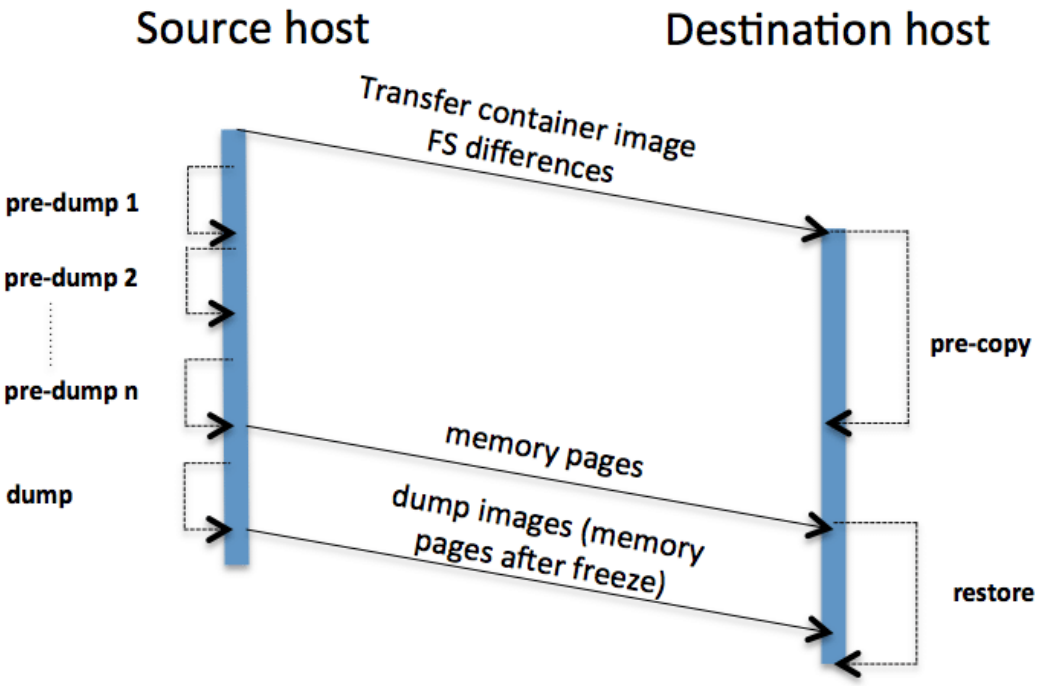

FIGURE 4.3: Container Migration Procedure with CRIU

```
Container: $docker checkpoint create eNodeBContainer checkpoint1
--checkpoint-dir=/dockerCheckpoints
```

Once all the required images have been copied and the states saved, the process/container could be restored on either the RCC or the RRU using the following commands:

```
$ criu restore -d -vvv -o dump.log && echo OK
$ docker start --checkpoint checkpoint1 eNodeBContainer
```

4.6 Observations on CRIU

During checkpointing, we found out that CRIU does not currently support linux-kernel SCTP (lk-sctp) sockets, so were unable to complete a full dump of the eNodeB application and it's connections which we require for the process restoration. We looked to modify CRIU to support SCTP. Having studied how CRIU checkpoints other sockets used by the eNodeB, we looked to implement the same functions for SCTP sockets, however this was ultimately unsuccessful. We considered an alternative solution which was to replace the current linux SCTP with a Userspace SCTP [3] (usrsctp) implementation that allows encapsulated SCTP packets via UDP datagrams (a protocol CRIU

[3] https://github.com/sctplab/usrsctp

supports), but we realized that this was not possible due to the fact the lk-sctp is one of the modules/packages OpenvSwitch depends on, so we could not offload it without affecting how OvS behaves. We therefore decided to use CRIU's dump function to instead evaluate the behaviour and performance of the eNodeB, and the network setup at large, when the eNodeB is down momentarily. Our observations and evaluation of the network setup is detailed in the next chapter.

4.7 Towards Live Migration

Live migrating a containerized eNodeB was the original goal of this thesis, but as we show in this section being able to checkpoint and resume the eNodeB doesnt guarantee that the service will run normally, and provide the user with the consistent access to the network service in real time.

To evaluate our setup, we looked at how the network handles common network scenarios. One of the benefits of LTE systems is the ability to provide users with consistent mobile connections even in times of mobility. We wanted to observe how our network setup handles users being disconnected for a period of time (approximately 2 minutes) and coming back into range of the cell, in real time. This time interval was chosen to allow the network to detect the UE is no longer available in the cell and disconnect it from the EPC as well as allow the eNodeB to perform other radio resource control/management related tasks. To test this, we connected the UE to the EPC and maintained a stable connection for a period of 5 minutes. We then disconnected the UE from the network for 2 minutes. During this time the UE is detached from the EPC, although the EPC shows the it remains attached/registered. After 2 minutes we switch on our UE and try to reconnect to the network. The eNodeB is able to pick up the UE and reconnect it to the EPC, however in this instance the UE is given a new ID or Radio Network Temporary Identifier (RNTI) value. This shows that our setup works in such a case.

Live container migration causes the network setup to be down for a period of time. We decided to analyse the behaviour/response of the network in a similar situation to a live eNodeB container migration. To this end, we observed the changes in the network when the eNodeB service is disrupted for a short period of time (less than 1s). Checkpointing the eNodeB container affects the eNodeB process running inside, as CRIU pauses the application to collect the latest information about the state and connections of eNodeB into image files, which can be used to restore the container later. As the time between checkpoint and restore needed to be as short as possible, we opted to resume the container as soon as the checkpointing process was completed (using the *–leave-running* tag).

From our observations, we could see that the eNodeB-MME connection is immediately resumed upon restoration. The output from the MME shows that this short disruption doesn't trigger any events at the MME (this is not the case when the eNodeB is unavailable for extended periods, or shut down). However, the connection between the OAI eNodeB and the UE is impacted by this disruption. The OAI eNodeB has a local counter that counts how many frames and subframes have been processed since the establishment of the eNodeB and radio unit connection. When the eNodeB process is disrupted, the reception and transmission (RX/TX) threads that process the LTE frames and subframes are also stopped. However, since we are following the NGFI architecture the RRH is still on and continues to communicate with the UE over the air interface during this downtime. Restoring the eNodeB resumes the RX/TX thread, however, at this point in time the local frame and subframe counter, which continues where it left off, and the frames and subframes received from the radio fronthaul differ, which causes an error in the eNodeB process. To fix this, we modify the OAI code (Appendix B.1). Our patch sets the local frame and subframe counter to correspond with the underlying frame and subframe number received from the radio (Appendix B.1). This ensures that the eNodeB doesn't crash immediately upon being restored.

To better motivate the need for the local counter frame/subframe counter implemented in the eNodeB, it is important to look at how timing/synchronization affects the connection between the UE and RRU/eNodeB. Before the UE and the eNodeB can be initially connected, they have to be time-synchronized (e.g. LTE/radio frame timing). When the two components are synchronized in this way, the PRACH, which is responsible for requesting uplink resources, can begin to request for such resources which allows the UE to be selected for uplink transmissions [26]. When the eNodeB is momentarily stopped, this breaks the synchronization, causing the radio connection between the eNodeB and UE to be lost. As a result of the lost radio connection between the components, the eNodeB initiates a *UEContextReleaseRequest* which allows the eNodeB to request for the MME to release the UE-associated logical S1 connection due to E-UTRAN generated reasons (e.g. lost connection with UE), at which point the UE is removed from the list of actively connected devices in the core network.

As the UE is now disconnected from the network, it sends periodic Traffic Area Update (TAU) requests, via the NAS protocol, to the MME to let it know that it is available. The UE controls this procedure using the periodic TAU timer (T3412). The value of the timer is initially sent by the network to the UE upon initial attachment in an ATTACH ACCEPT message or a TRACKING AREA UPDATE ACCEPT message. After the eNodeB has been dumped and resumed, the UE goes from *EMM_CONNECTED* to *EMM_IDLE* mode, where an EMM is the Evolve Packet System Mobility Management layer controlled by the NAS layer and that tracks the UEs in the network. In this state

the T3412 timer is reset and started with it's initial value. We deduce that due to the change in the state of the UE which triggers this difference in TAU timer, the MME rejects any periodic TAU request that is received and therefore the UE is unable to connect to the MME and re-establish it's connection to the internet.

In the experiments we performed, we left the eNodeB running after a checkpoint. We noticed that after approximately one hour of the UE trying to make a connection with the MME, the connection between the two components is re-established ad the UE has access to the internet again. By tracing the actions of the network, we see that the UE is re-attached to the MME/core network as a new UE (based on new user context, and new RNTI). Hence, we can conclude that it takes a long time before the EPC is able to accept UE attachments requests due to disruption of the service and the lack of synchronization in the network. A live migration would not have been possible if the EPC was not able to accept new connections from the same (or different) UEs, based on our tried experiments.

Chapter 5

Evaluation

5.1 Introduction

We evaluate the performance of our setup by measuring the throughput whilst the eNodeB resides in the container and directly on the host (GPP).

To evaluate our network setup, we used a stationary UE that was attached to one of the computers in our setup and connect are able to connect the UE to the EPC and internet through the eNodeB via the RRU/RRH. The figures from our evaluation are based on such a setup. The RCC (eNodeB), RRU, and UE are all in the same room, and as such the UE is close to the RRU during all our experiments. The EPC that is used for our work is located in the server on the third floor on the Communication and Distributed Systems building at the University of Bern (Room B in Figure 5.1).

In terms of video streaming, We noticed that our network setup did not perform as well when the eNodeB was running in the container as on the host. We were able to stream videos in high quality from the UE and were able to download rather large files (100 MB) when the eNodeB ran directly on the host, but when deployed in a container, the eNodeB would often crash upon such heavy usage. The video quality on the UE was also poorer when using a containerized eNodeB. Based on this, we would say that the challenges to meet the strict time requirements in LTE play a part in the poorer performance of the network in a containerized environment as such an environment brings it's own processing overheads.

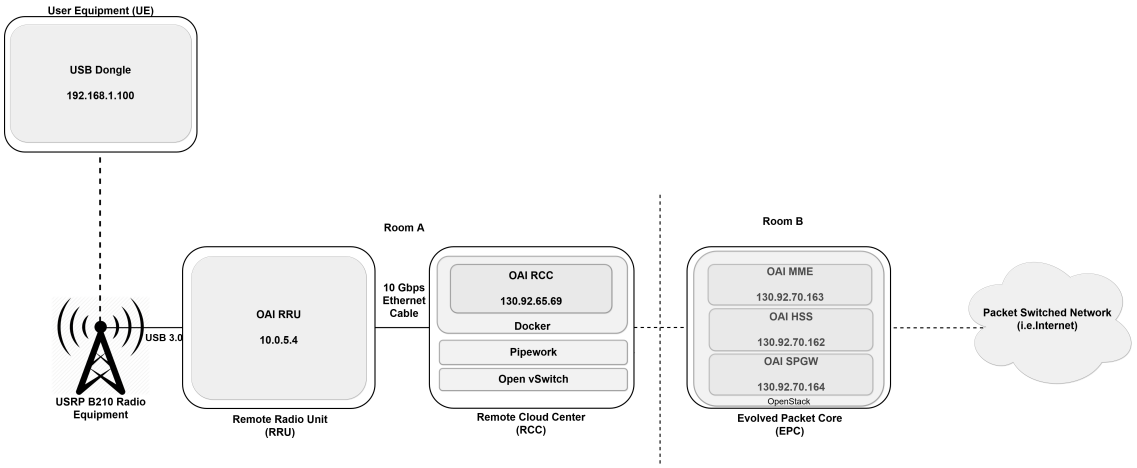

FIGURE 5.1: Network setup with stationary UE

5.2 Throughput

During the time the eNodeB and UE were connected, we measured the throughput of the network using different Personal Resource Blocks provided by the configuration of the eNodeB. Our results showed that the throughput of the UE whilst the eNodeB was running in the docker container, was not significantly worse than when running it was running on bare metal (directly on host). Note that for each bandwidth measurement we take only the maximum recorded value of each experiment we performed.

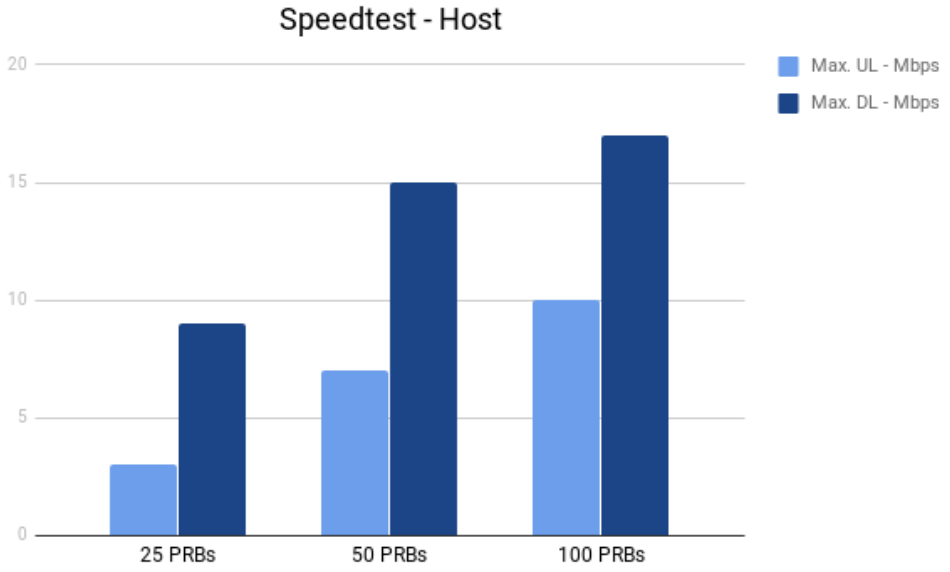

FIGURE 5.2: Throughput on Host

As can be seen in Figure 5.2 we were able to achieve a maximum throughput of up to 17Mbps for download speed and 10Mbps for upload speed when using 100 PRBs and running on the host. With the same configuration we were only able to get 6Mbps

DL and 3Mbps UL. These results were collected from the speedtest[1] website. This performance is consistent with what we would expect and have seen in the literature, where the downlink is often almost twice as much as the uplink. From our results we

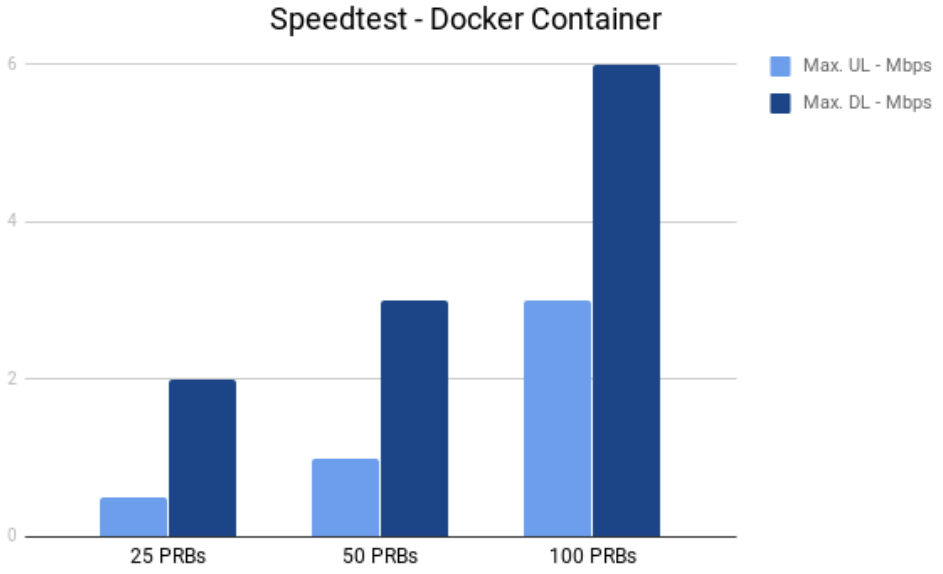

FIGURE 5.3: Throughput on Docker container

can conclude that OAI performs considerably better when running directly on the host, compared to running on the container.

In Appendix C, we show the running logs of the eNodeB and MME at the different points in the network such as: Connection establishment between the eNodeB and RRU, UE connection to EPC via RAN (eNodeB), registration of the UE with the MME, and the output of the eNodeB and MME when the eNodeB has been checkpointed.

[1] https://www.speedtest.net/

Chapter 6

Conclusion

In this thesis we provided the technical requirements, design and implementation details on how to we implemented a cloudified RAN.

In chapter 2 we presented a background on LTE mobile networks, and looked at related technologies such as Virtual Radio Access Networks (Cloud-RAN) and Next Generation Fronthaul Interfaces, and the benefits they bring in terms of keeping expenses down, while also improving the network's service in terms of scalability and reliability. We also briefly looked at the LTE software platform on which our work is based. Finally, we explained Software Defined Radios and their used in our setup.

In Chapter 3 We presented the architecture of our Cloud RAN implementation as well as the modifications that needed to be made to ensure our hosts could meet the strict processing deadlines involved in LTE networks. We talked about the used of containerization to cloudify our RAN and how we setup the networking between the container and the host to be able to have unfiltered communication with the EPC from the container. Finally, we laid the ground work for the subsequent chapter by briefly introducing the CRIU, the migration tool we intended to use to live migrate the containerized eNodeB.

Chapter 4 focused on our efforts to perform the live migration of the container. We explained how we setup the eNodeB inside the container. We then looked at the live migration procedure in general, and how CRIU can be used to perform live migration. Finally, we document the challenges of performing a live container migration in real time. We describe the behaviour of the LTE network in certain scenarios, and the patch we implemented to ensure that the eNodeB service doesn't die immediately after it has been checkpointed. We showed that whilst our patch achieves it's aim, the behaviour of the network after the eNodeB has been temporarily unavailable means that a live migration of the containerized eNodeB is not yet feasible, due in part to the strict LTE subframe

and frame processing deadlines, and the network needing time to re-synchronize and reconnect UEs.

We evaluated and compared the performance of our RAN setup when running inside a container and on the host in Chapter 5. We showed that our results are consistent with what would be expected when the RAN is deployed in those two environments. Finally we present a evaluation scenario

From our analysis, the disruption of the eNodeB service that would occur in the case of a live migration is not the primary cause of network failure. The MME and S GW play significant roles in how the UE is reconnected after the service is resumed. We deduce that the OAI EPC implementation is unable to cater to live eNodeB container migrations in real time, even though there have been some successes in simulated environments.

The performance of a live eNodeB container migration between hosts did not occur as we intended, but nevertheless the details we presented could contribute to eventually being able to achieve this problem and carry out the migration successfully. Firstly, a means to checkpoint and restore linux SCTP sockets that works with Open vSwitch is needed before the eNodeB can be fully dumped. Secondly, a way to ensure the downtime of the eNodeB is as minimal as possible will also provide a way to successfully migrate the container without breaking any LTE timing restrictions. We posit that once these are solved, it would be possible to successfully checkpoint and ensure that the UE remains unaffected when it is restored.

To keep up with the theme of cloud computing, we believe that manually migrating the eNodeB between hosts is inefficient. In large scale deployments of the RAN in the cloud, a more efficient (and potentially quicker) way would be to use an orchestration system that could automatically detect what machine to migrate the container to, and when the best to perform a migration of the eNodeB service is.

6.0.1 Future Work

Future work would focus on understanding in detail how the EPC handles different use case scenarios in real time that could occur as a result of network deployment in virtual environments such as the cloud. OAI is currently able to handle handovers using the X2 interface described in this thesis. In a handover the UE is temporarily disconnected from the source eNodeB (typically milliseconds) and handed over to the target eNodeB. Similarly, a live eNodeB migration performs the same procedure and we expect therefore that the network should reconnect the UE as is the case in a handover. In other words, the UE is not supposed to notice this change in operation of the eNodeBs. For a live

migration to be successful the migration procedure would have to adhere to extremely strict time requirements in LTE (due to arrival of subframes every millisecond, HARQ, etc). Most migration tools take several milliseconds, and from our observations we have seen that even just a second of the RAN being unavailable leads to a non-convergence of the operation of the network. We posit that a true real-time live migration of the RAN might not be feasible unless: (1) container/Virtual Machine migrations can occur in the milliseconds range, allowing for the network to stay synchronized, or (2) modifying the network (EPC and RAN) to deal with disruptions in the eNodeB service during a live migration, i.e. allowing registered/attached UEs to create new uplink transmissions with the MME/S GW, despite the temporary disconnection.

Appendix A

Software and NFS Configuration

A.1 Configuration file of the eNodeB at the RCC

```
1
2  Active_eNBs = ( "eNB-Eurecom-LTEBox");
3  # Asn1_verbosity, choice in: none, info, annoying
4  Asn1_verbosity = "none";
5
6  eNBs =
7  (
8   {
9      # real_time choice in {hard, rt-preempt, no}
10     real_time        =  "no";
11     ////////// Identification parameters:
12     eNB_ID    =  0xe00;
13     cell_type =  "CELL_MACRO_ENB";
14     eNB_name  =  "eNB-Eurecom-LTEBox";
15     // Tracking area code, 0x0000 and 0xfffe are reserved values
16     tracking_area_code = 1;
17     plmn_list = ( { mcc = 208; mnc = 95; mnc_length = 2; } );
18     tr_s_preference     = "local_mac"
19
20     ////////// Physical parameters:
21     component_carriers = (
22       {
23       node_function              = "NGFI_RCC_IF4p5";
24       node_timing                = "synch_to_ext_device";
25       node_synch_ref             = 0;
26       frame_type                 = "FDD";
27       tdd_config                 = 3;
28       tdd_config_s                        = 0;
29       prefix_type                         = "NORMAL";
```

```
      eutra_band                              = 7;
      downlink_frequency                      = 2685000000L;
      uplink_frequency_offset                 = -120000000;
      Nid_cell                = 0;
      N_RB_DL                                 = 100;
      Nid_cell_mbsfn                          = 0;
      nb_antenna_ports                                    = 1;
      nb_antennas_tx                          = 1;
      nb_antennas_rx                          = 1;
      tx_gain                                             = 90;
      rx_gain                                             = 125;
      pbch_repetition                                     = "FALSE";
      prach_root                              = 0;
      prach_config_index                      = 0;
      prach_high_speed                        = "DISABLE";
      prach_zero_correlation                  = 1;
      prach_freq_offset                       = 2;
      pucch_delta_shift                       = 1;
      pucch_nRB_CQI                           = 0;
      pucch_nCS_AN                            = 0;
      pucch_n1_AN                             = 0;
      pdsch_referenceSignalPower                = -27;
      pdsch_p_b                                 = 0;
      pusch_n_SB                                = 1;
      pusch_enable64QAM                         = "DISABLE";
      pusch_hoppingMode                                   = "interSubFrame
    ";
      pusch_hoppingOffset                                 = 0;
      pusch_groupHoppingEnabled               = "ENABLE";
      pusch_groupAssignment                   = 0;
      pusch_sequenceHoppingEnabled                  = "DISABLE";
      pusch_nDMRS1                                        = 1;
      phich_duration                                      = "NORMAL";
      phich_resource                                      = "ONESIXTH";
      srs_enable                                          = "DISABLE";
....
      pusch_p0_Nominal                                    = -96;
      pusch_alpha                                         = "AL1";
      pucch_p0_Nominal                                    = -104;
      msg3_delta_Preamble                                 = 6;
      pucch_deltaF_Format1                                = "deltaF2";
      pucch_deltaF_Format1b                               = "deltaF3";
      pucch_deltaF_Format2                                = "deltaF0";
      pucch_deltaF_Format2a                               = "deltaF0";
      pucch_deltaF_Format2b                   = "deltaF0";
      rach_numberOfRA_Preambles                           = 64;
      rach_preamblesGroupAConfig                          = "DISABLE";
....
```

```
      rach_powerRampingStep                                 = 4;
      rach_preambleInitialReceivedTargetPower               = -108;
      rach_preambleTransMax                                 = 10;
      rach_raResponseWindowSize                             = 10;
      rach_macContentionResolutionTimer                     = 48;
      rach_maxHARQ_Msg3Tx                                   = 4;
      pcch_default_PagingCycle                              = 128;
      pcch_nB                                               = "oneT";
....
      }
    );
....
    # ------- SCTP definitions
    SCTP :
    {
        # Number of streams to use in input/output
        SCTP_INSTREAMS  = 2;
        SCTP_OUTSTREAMS = 2;
    };

    ////////// MME parameters:
    mme_ip_address      = ( { ipv4       = "130.92.70.163";
                              ipv6       = "192:168:30::17";
                              active     = "yes";
                              preference = "ipv4";
                            }
                          );
    enable_measurement_reports = "no";
    ///X2
    enable_x2 = "no";
    t_reloc_prep      = 1000;      /* unit: millisecond */
    tx2_reloc_overall = 2000;      /* unit: millisecond */

    NETWORK_INTERFACES :
    {
        ENB_INTERFACE_NAME_FOR_S1_MME            = "eth0";
        ENB_IPV4_ADDRESS_FOR_S1_MME              = "130.92.65.69/24";
        ENB_INTERFACE_NAME_FOR_S1U               = "eth0";
        ENB_IPV4_ADDRESS_FOR_S1U                 = "130.92.65.69/24";
        ENB_PORT_FOR_S1U                         = 2152; # Spec 2152
        ENB_IPV4_ADDRESS_FOR_X2C                 = "130.92.65.69/24";
        ENB_PORT_FOR_X2C                         = 36422; # Spec 36422
    };
  }
);
....
RUs = (
    {
```

```
      local_if_name  = "eth1";
      remote_address = "10.0.5.4";
      local_address  = "10.0.5.2";
      local_portc    = 50000;
      remote_portc   = 50000;
      local_portd    = 50001;
      remote_portd   = 50001;
      local_rf       = "no"
      tr_preference  = "udp_if4p5"
      nb_tx          = 1
      nb_rx          = 1
      att_tx         = 0
      att_rx         = 0;
      eNB_instances  = [0];
      is_slave       = "no"
    }
);
THREAD_STRUCT = (
  {
    #three config for level of parallelism "PARALLEL_SINGLE_THREAD", "
    PARALLEL_RU_L1_SPLIT", or "PARALLEL_RU_L1_TRX_SPLIT"
    parallel_config    = "PARALLEL_RU_L1_TRX_SPLIT";
    #two option for worker "WORKER_DISABLE" or "WORKER_ENABLE"
    worker_config      = "WORKER_ENABLE";
  }
);
....
```

Listing A.1: rcc.band7.tm1.if4p5.50PRB.conf

A.2 Configuration file of the Radio at the RRU

```

RUs = (
    {
      local_if_name                   = "p1p2";
      remote_address                  = "10.0.5.2"
      local_address                   = "10.0.5.4";
      local_portc                      = 50000;
      remote_portc                     = 50000;
      local_portd                      = 50001;
      remote_portd                     = 50001;
      local_rf                        = "yes"
      tr_preference                   = "udp_if4p5";
      nb_tx                           = 2;
      nb_rx                           = 2;
```

```
      max_pdschReferenceSignalPower    = -27;
      max_rxgain                       = 125;
      bands                            = [7,13];
      is_slave                         = "no";
    }
);

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

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

Listing A.2: rru.oaisim.conf

A.3 NFS Configuraton

```
$ sudo apt-get update
$ sudo apt-get install nfs-kernel-server
```

Listing A.3: Downloading and Installing the Components on RCC

```
$ sudo apt-get update
$ sudo apt-get install nfs-common
```

Listing A.4: Downloading and Installing the Components on RRU

```
$ sudo mkdir -p /nfs/home
sudo mount 130.92.65.83:/home/tofunmi /nfs/home
```

Listing A.5: Creating the Mount Points and Mounting Directory on RRU

```
$ sudo nano /etc/exports
# Add below line to exports file:
/home       130.92.65.32(rw,sync,no_root_squash,no_subtree_check)
# Restart nfs-kernel-server service
$ sudo systemctl restart nfs-kernel-server
```

Listing A.6: Configuring the NFS Exports on the RCC

```
# First, check firewall status
$ sudo ufw status
# If ufw is inactive, use the below command to enable ufw:
$ sudo ufw enable
# Make ufw allow incoming and outgoing:
$ sudo ufw default allow incoming
$ sudo ufw default allow outgoing
# Make client server can access host server
$ sudo ufw allow from 130.92.65.32 to any port nfs
# Check ufw status
$ sudo ufw status numbered
```

Listing A.7: Adjusting firewall on RCC

Appendix B

Implementation

B.1 Modification of lte-ru.c

When the eNodeB is restored, the difference in time caused it to crash. Modifying the in-built OAI subframe timer ensures the application doesn't crash upon resumption. More future work in this area could also ensure that the threads are processing the right subframes, taking into account the time difference.

```
1  // Synchronous if4p5 from south
2  void fh_if4p5_south_in(RU_t *ru,int *frame,int *subframe) {
3
4    LTE_DL_FRAME_PARMS *fp = &ru->frame_parms;
5    RU_proc_t *proc = &ru->proc;
6    int f,sf;
7
8
9    uint16_t packet_type;
10   uint32_t symbol_number=0;
11   uint32_t symbol_mask_full;
12   uint64_t t;
13   //uint64_t t2;
14   //uint64_t t_diff;
15
16   if ((fp->frame_type == TDD) && (subframe_select(fp,*subframe)==SF_S))
17     symbol_mask_full = (1<<fp->ul_symbols_in_S_subframe)-1;
18   else
19     symbol_mask_full = (1<<fp->symbols_per_tti)-1;
20   if (proc->symbol_mask[*subframe] == symbol_mask_full) proc->symbol_mask
       [*subframe] = 0;
21   do {
22     recv_IF4p5(ru, &f, &sf, &packet_type, &symbol_number);
```

```
23    if (packet_type == IF4p5_PULFFT) proc->symbol_mask[sf] = proc->
      symbol_mask[sf] | (1<<symbol_number);
24    else if (packet_type == IF4p5_PULTICK) {
25      if ((proc->first_rx==0) && (f!=*frame)) LOG_E(PHY,"rx_fh_if4p5:
      PULTICK received frame %d != expected %d\n",f,*frame);
26      if ((proc->first_rx==0) && (sf!=*subframe)) LOG_E(PHY,"rx_fh_if4p5:
       PULTICK received subframe %d != expected %d (first_rx %d)\n",sf,*
      subframe,proc->first_rx);
27      break;
28    } else if (packet_type == IF4p5_PRACH) {
29      // nothing in RU for RAU
30    }
31    LOG_D(PHY,"rx_fh_if4p5: subframe %d symbol mask %x\n",*subframe,proc
      ->symbol_mask[*subframe]);
32  } while(proc->symbol_mask[*subframe] != symbol_mask_full);
33
34  //caculate timestamp_rx, timestamp_tx based on frame and subframe
35  proc->subframe_rx  = sf;
36  proc->frame_rx     = f;
37  proc->timestamp_rx = ((proc->frame_rx * 10)  + proc->subframe_rx ) * fp
      ->samples_per_tti ;
38
39  if (proc->first_rx == 0) {
40  //printf("rdtsc_diff: %" PRIu64 "\n", t_diff);
41    if (proc->subframe_rx != *subframe){ printf("Correcting Subframe\n");
42     LOG_E(PHY,"Received Timestamp doesn't correspond to the time we
      think it is (proc->subframe_rx %d, subframe %d)\n",proc->subframe_rx,
      sf);
43    }
44
45    if (proc->frame_rx != *frame) {printf("Correcting Frame\n");
46        *subframe = proc->subframe_rx;
47        *frame = proc->frame_rx;
48      LOG_E(PHY,"Received Timestamp doesn't correspond to the time we
      think it is (proc->frame_rx %d, frame %d)\n",proc->frame_rx,*frame);
49    }
50  }
51    else {
52    proc->first_rx = 0;
53    *frame = proc->frame_rx;
54    *subframe = proc->subframe_rx;
55  }
56
57  if (ru == RC.ru[0]) {
58    VCD_SIGNAL_DUMPER_DUMP_VARIABLE_BY_NAME(
      VCD_SIGNAL_DUMPER_VARIABLES_FRAME_NUMBER_RX0_RU, f );
59    VCD_SIGNAL_DUMPER_DUMP_VARIABLE_BY_NAME(
      VCD_SIGNAL_DUMPER_VARIABLES_SUBFRAME_NUMBER_RX0_RU, sf );
```

```
60      //VCD_SIGNAL_DUMPER_DUMP_VARIABLE_BY_NAME(
        VCD_SIGNAL_DUMPER_VARIABLES_FRAME_NUMBER_TX0_RU, proc->frame_tx );
61      //VCD_SIGNAL_DUMPER_DUMP_VARIABLE_BY_NAME(
        VCD_SIGNAL_DUMPER_VARIABLES_SUBFRAME_NUMBER_TX0_RU, proc->subframe_tx
        );
62    }
63    proc->symbol_mask[sf] = 0;
64    VCD_SIGNAL_DUMPER_DUMP_VARIABLE_BY_NAME(
        VCD_SIGNAL_DUMPER_VARIABLES_TRX_TS, proc->timestamp_rx&0xffffffff );
65    LOG_D(PHY,"RU %d: fh_if4p5_south_in sleeping ...\n",ru->idx);
66    usleep(100);
67  }
```

Listing B.1: lte-ru.c

Appendix C

eNodeB and MME Logs

C.1 Connecting the eNodeB and RRU

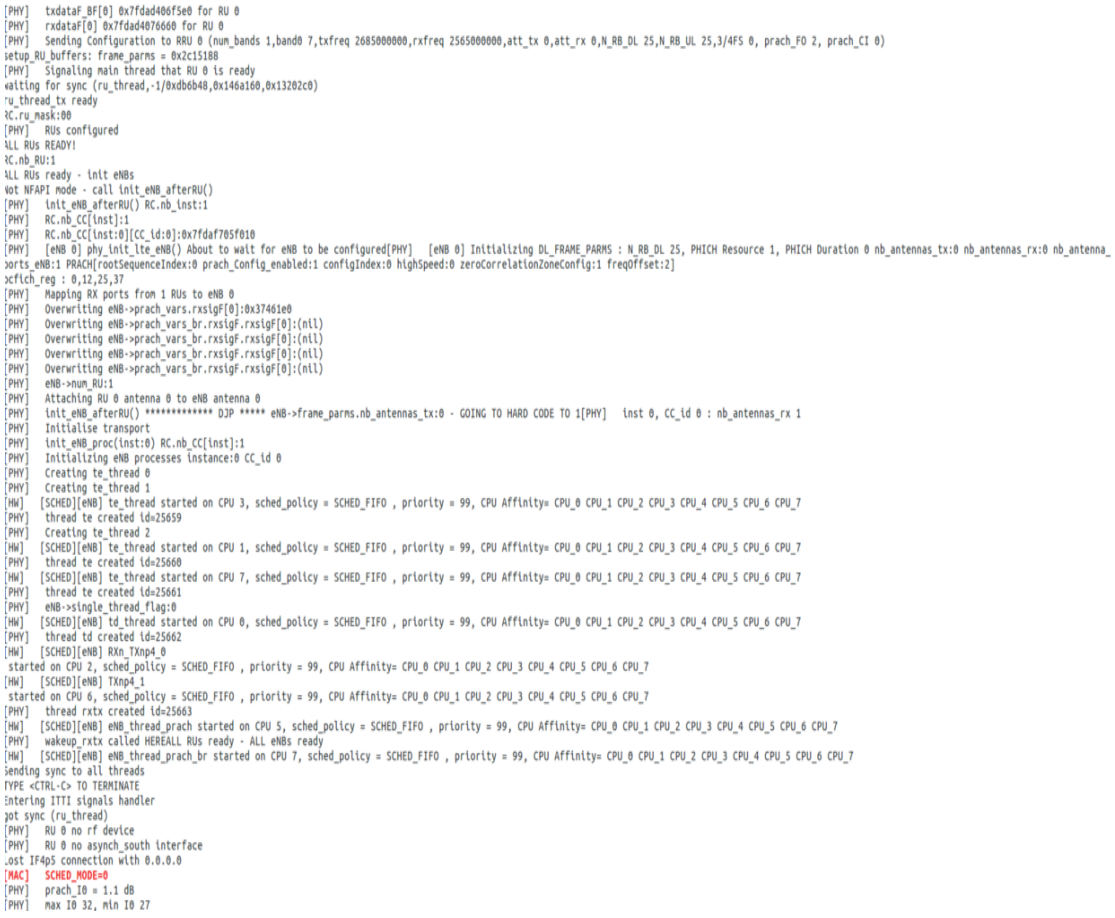

FIGURE C.1: Connection established between eNodeB and RRU

C.2 UE connection to EPC via eNodeB

```
[RRC]
KeNB:e9 45 94 16 fe 40 6b 27 37 aa 3f 0d 17 d9 c5 db 9a b2 81 c9 08 e0 4b 74 d1 17 1f e7 78 c4 f7 d4
[RRC]
KRRCenc:6b a1 e1 38 03 4e 25 80 01 53 94 55 f9 ee 06 41 93 3c 39 1e ca ba 34 94 32 f9 3a 8f 7a a1 97 f8
[RRC]
KRRCint:16 2c eb d0 06 f0 bf cc 3e e6 c7 45 35 40 c0 19 00 32 ac 61 f4 79 f5 10 cb 6c fe b9 1f 62 e5 54
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 3c48] Logical Channel DL-DCCH, Generate SecurityModeCommand (bytes 3)
[RLC]   [FRAME 00000][eNB][MOD 00][RNTI 3c48][SRB AM 01] RLC_AM_DATA_REQ size 8 Bytes,  NB SDU 4 current_sdu_index=3 next_sdu_index=4 conf 0 mui 0 vtA 3 vtS 3
[RRC]   [eNB 0] Frame 61: received a DCCH 1 message on SRB 1 with Size 2 from UE 3c48
[RRC]   Received message RRC_DCCH_DATA_IND
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 3c48] Received on DCCH 1 RRC_DCCH_DATA_IND
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 3c48] received securityModeComplete on UL-DCCH 1 from UE
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 3c48] Logical Channel DL-DCCH, Generate UECapabilityEnquiry (bytes 3)
[RLC]   [FRAME 00000][eNB][MOD 00][RNTI 3c48][SRB AM 01] RLC_AM_DATA_REQ size 8 Bytes,  NB SDU 5 current_sdu_index=4 next_sdu_index=5 conf 0 mui 1 vtA 4 vtS 4
[RRC]   [eNB 0] Frame 64: received a DCCH 1 message on SRB 1 with Size 44 from UE 3c48
[RRC]   Received message RRC_DCCH_DATA_IND
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 3c48] Received on DCCH 1 RRC_DCCH_DATA_IND
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 3c48] received ueCapabilityInformation on UL-DCCH 1 from UE
[RRC]   got UE capabilities for UE 3c48
[RRC]   RRCConnectionReconfiguration Encoded 1077 bits (135 bytes)
[RRC]   [eNB 0] Frame 0, Logical Channel DL-DCCH, Generate LTE_RRCConnectionReconfiguration (bytes 135, UE id 3c48)
[SCTP]   Successfully sent 71 bytes on stream 1 for assoc_id 125
[RLC]   [FRAME 00000][eNB][MOD 00][RNTI 3c48][SRB AM 01] RLC_AM_DATA_REQ size 140 Bytes,  NB SDU 6 current_sdu_index=5 next_sdu_index=6 conf 0 mui 2 vtA 5 vtS 5
[RRC]   [eNB 0] Frame 66: received a DCCH 1 message on SRB 1 with Size 2 from UE 3c48
[RRC]   Received message RRC_DCCH_DATA_IND
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 3c48] Received on DCCH 1 RRC_DCCH_DATA_IND
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 3c48] UE State = RRC_RECONFIGURED (default DRB, xid 1)
[PDCP]   [FRAME 00000][eNB][MOD 00][RNTI 3c48][SRB 02]  Action ADD  LCID 2 (SRB id 2) configured with SN size 5 bits and RLC AM
[PDCP]   [FRAME 00000][eNB][MOD 00][RNTI 3c48][DRB 01]  Action ADD  LCID 3 (DRB id 1) configured with SN size 12 bits and RLC UM
[RLC]   [FRAME 00000][eNB][MOD 00][RNTI 3c48] [SRB 2] rrc_rlc_add_rlc  SRB
[RLC]   [FRAME 00000][eNB][MOD 00][RNTI 3c48][SRB AM 02][CONFIGURE] max_retx_threshold 32 poll_pdu 8 poll_byte 16960 t_poll_retransmit 15 t_reordering 35 t_status_prohibit 10
[RLC]   [FRAME 00000][eNB][MOD 00][RNTI 3c48] [DRB 1] rrc_rlc_add_rlc  DRB
[RRC]   [eNB 0] Frame  0 CC 0 : SRB2 is now active
[RRC]   [eNB 0] Frame  0 : Logical Channel UL-DCCH, Received LTE_RRCConnectionReconfigurationComplete from UE rnti 3c48, reconfiguring DRB 1/LCID 3
[RRC]   [eNB 0] Frame  0 : Logical Channel UL-DCCH, Received LTE_RRCConnectionReconfigurationComplete from UE 0, reconfiguring DRB 1/LCID 3
[S1AP]   initial_ctxt_resp_p: e_rab ID 5, enb_addr 130.92.65.83, SIZE 4
[SCTP]   Successfully sent 40 bytes on stream 1 for assoc_id 125
[RRC]   [eNB 0] Frame 69: received a DCCH 2 message on SRB 2 with Size 16 from UE 3c48
[RRC]   Received message RRC_DCCH_DATA_IND
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI 3c48] Received on DCCH 2 RRC_DCCH_DATA_IND
[SCTP]   Successfully sent 61 bytes on stream 1 for assoc_id 125
[SCTP]   Found data for descriptor 59
[SCTP]   Received notification for sd 59, type 32777
[MAC]   UE  rnti 3c48 : in synch, PHR 40 dB DL CQI 15 PUSCH SNR 18 PUCCH SNR 10
[MAC]   UE  rnti 3c48 : in synch, PHR 40 dB DL CQI 15 PUSCH SNR 18 PUCCH SNR 10
[MAC]   UE  rnti 3c48 : in synch, PHR 40 dB DL CQI 15 PUSCH SNR 18 PUCCH SNR 10
[MAC]   UE  rnti 3c48 : in synch, PHR 40 dB DL CQI 14 PUSCH SNR 18 PUCCH SNR 10
[MAC]   [eNB 0][PUSCH 6] CC_id 0 543.0 ULSCH in error in round 0, ul_cqi 116
[MAC]   [eNB 0][PUSCH 6] CC_id 0 543.8 ULSCH in error in round 1, ul_cqi 164
[MAC]   [eNB 0][PUSCH 6] CC_id 0 544.6 ULSCH in error in round 2, ul_cqi 164
[MAC]   UE  rnti 3c48 : in synch, PHR 40 dB DL CQI 15 PUSCH SNR 18 PUCCH SNR 10
[MAC]   UE  rnti 3c48 : in synch, PHR 40 dB DL CQI 14 PUSCH SNR 18 PUCCH SNR 10
[MAC]   UE  rnti 3c48 : in synch, PHR 40 dB DL CQI 14 PUSCH SNR 21 PUCCH SNR 18
[MAC]   UE  rnti 3c48 : in synch, PHR 40 dB DL CQI 15 PUSCH SNR 18 PUCCH SNR 17
[RRC]   UE rnti 3c48 failure timer 0/8
[PHY]   prach_I0 = 8.5 dB
[PHY]   max_I0 33, min_I0 27
[MAC]   UE  rnti 3c48 : in synch, PHR 40 dB DL CQI 14 PUSCH SNR 18 PUCCH SNR 17
[MAC]   UE  rnti 3c48 : in synch, PHR 40 dB DL CQI 15 PUSCH SNR 18 PUCCH SNR 18
[MAC]   UE  rnti 3c48 : in synch, PHR 40 dB DL CQI 13 PUSCH SNR 18 PUCCH SNR 18
[MAC]   UE  rnti 3c48 : in synch, PHR 40 dB DL CQI 13 PUSCH SNR 18 PUCCH SNR 18
```

FIGURE C.2: Full connection between UE and MME: Frames being processed

C.3 MME registration

```
001422 00102:195100 7F649661E700 TRACE NAS-ES openair-cn/src/nas/esm/esm_ebr.c:0530                        Entering esm_ebr_set_status()
001423 00102:195108 7F649661E700 INFO  NAS-ES openair-cn/src/nas/esm/esm_ebr.c:0566                          ESM-FSM   - Status of EPS bearer context 5 changed: BEARER CONTEXT ACTIVE PENDING ===> BEARE
R CONTEXT ACTIVE
001424 00102:195121 7F649661E700 TRACE NAS-ES openair-cn/src/nas/esm/esm_ebr.c:0570                        Leaving esm_ebr_set_status() (rc=0)
001425 00102:195129 7F649661E700 TRACE NAS-ES aultEpsBearerContextActivation.c:0275                      Leaving esm_proc_default_eps_bearer_context_accept() (rc=0)
001426 00102:195137 7F649661E700 TRACE NAS-ES ir-cn/src/nas/esm/sap/esm_recv.c:0464                    Leaving esm_recv_activate_default_eps_bearer_context_accept() (rc=-1)
001427 00102:195145 7F649661E700 TRACE NAS-ES air-cn/src/nas/esm/sap/esm_sap.c:0745                  Leaving _esm_sap_recv() (rc=0)
001428 00102:195150 7F649661E700 TRACE NAS-ES air-cn/src/nas/esm/sap/esm_sap.c:0273                Leaving esm_sap_send() (rc=0)
001429 00102:195155 7F649661E700 TRACE NAS-EM air-cn/src/nas/emm/sap/emm_sap.c:0118                Entering emm_sap_send()
001430 00102:195160 7F649661E700 TRACE NAS-EM air-cn/src/nas/emm/sap/emm_reg.c:0106                  Entering emm_reg_send()
001431 00102:195164 7F649661E700 TRACE NAS-EM air-cn/src/nas/emm/sap/emm_fsm.c:0256                    Entering emm_fsm_process()
001432 00102:195171 7F649661E700 INFO  NAS-EM air-cn/src/nas/emm/sap/emm_fsm.c:0263                      EMM-FSM   - Received event ATTACH_CNF (5) in state DEREGISTERED
001433 00102:195180 7F649661E700 TRACE NAS-EM rc/nas/emm/sap/EmmDeregistered.c:0094                      Entering EmmDeregistered()
001434 00102:195188 7F649661E700 TRACE NAS-EM air-cn/src/nas/emm/sap/emm_fsm.c:0176                        Entering emm_fsm_set_status()
001435 00102:195196 7F649661E700 INFO  NAS-EM air-cn/src/nas/emm/sap/emm_fsm.c:0182                          UE 0x80000001 EMM-FSM   - Status changed: DEREGISTERED ===> REGISTERED
001436 00102:195208 7F649661E700 TRACE MME-AP cn/src/mme_app/mme_app_context.c:1014                        Entering mme_ue_context_update_ue_emm_state()
001437 00102:195218 7F649661E700 TRACE MME-AP cn/src/mme_app/mme_app_context.c:1033                        Leaving mme_ue_context_update_ue_emm_state()
001438 00102:195226 7F649661E700 TRACE NAS-EM air-cn/src/nas/emm/sap/emm_fsm.c:0194                      Leaving emm_fsm_set_status() (rc=0)
001439 00102:195233 7F649661E700 TRACE NAS-EM rc/nas/emm/sap/EmmDeregistered.c:0155                    Leaving EmmDeregistered() (rc=0)
001440 00102:195241 7F649661E700 TRACE NAS-EM air-cn/src/nas/emm/sap/emm_fsm.c:0271                  Leaving emm_fsm_process() (rc=0)
001441 00102:195247 7F649661E700 TRACE NAS-EM air-cn/src/nas/emm/sap/emm_reg.c:0119                Leaving emm_reg_send() (rc=0)
001442 00102:195252 7F649661E700 TRACE NAS-EM air-cn/src/nas/emm/sap/emm_sap.c:0143              Leaving emm_sap_send() (rc=0)
001443 00102:195257 7F649661E700 TRACE NAS-EM /openair-cn/src/nas/emm/Attach.c:0723            Leaving emm_proc_attach_complete() (rc=0)
001444 00102:195262 7F649661E700 TRACE NAS-EM ir-cn/src/nas/emm/sap/emm_recv.c:0349          Leaving emm_recv_attach_complete() (rc=0)
001445 00102:195267 7F649661E700 TRACE NAS-EM nair-cn/src/nas/emm/sap/emm_as.c:0434        Leaving _emm_as_recv() (rc=0)
001446 00102:195272 7F649661E700 TRACE NAS-EM nair-cn/src/nas/emm/sap/emm_as.c:0541      Leaving _emm_as_data_ind() (rc=0)
001447 00102:195277 7F649661E700 TRACE NAS-EM nair-cn/src/nas/emm/sap/emm_as.c:0243    Leaving emm_as_send() (rc=0)
001448 00102:195282 7F649661E700 TRACE NAS-EM air-cn/src/nas/emm/sap/emm_sap.c:0143  Leaving emm_sap_send() (rc=0)
001449 00102:195287 7F649661E700 TRACE NAS-EM rv/openair-cn/src/nas/nas_proc.c:0326  Leaving nas_proc_ul_transfer_ind() (rc=0)
001450 00102:197553 7F649561C700 DEBUG UDP    /src/udp/udp_primitives_server.c:0187  Received 1 events
001451 00102:197566 7F649561C700 DEBUG UDP    /src/udp/udp_primitives_server.c:0105  Looking for sd 33
001452 00102:197578 7F649561C700 DEBUG UDP    /src/udp/udp_primitives_server.c:0108  Found matching task desc
001453 00102:197575 7F649561C700 DEBUG UDP    /src/udp/udp_primitives_server.c:0212  Inserting new descriptor for task 6, sd 33
001454 00102:197589 7F649561C700 DEBUG UDP    /src/udp/udp_primitives_server.c:0239  Msg of length 18 received from 130.92.70.164:2123
001455 00102:197612 7F6494E1B700 TRACE GTPv2- 2-c/nwgtpv2c-0.11/src/NwGtpv2c.c:1113  Entering nwGtpv2cProcessUdpReq()
001456 00102:197674 7F6494E1B700 TRACE GTPv2- 2-c/nwgtpv2c-0.11/src/NwGtpv2c.c:1535    Entering nwGtpv2cStopTimer()
001457 00102:197680 7F6494E1B700 DEBUG GTPv2- 2-c/nwgtpv2c-0.11/src/NwGtpv2c.c:1542      Stopping active timer 0x7f6478011360 for info 0x0x7f64780112a0!
001458 00102:197686 7F6494E1B700 DEBUG GTPv2- 2-c/nwgtpv2c-0.11/src/NwGtpv2c.c:1545      Stopping active timer 0x7f6478011360 for info 0x0x7f64780112a0!
001459 00102:197697 7F6494E1B700 TRACE GTPv2- 2-c/nwgtpv2c-0.11/src/NwGtpv2c.c:1570    Leaving nwGtpv2cStopTimer() (rc=0)
001460 00102:197702 7F6494E1B700 DEBUG GTPv2- /nwgtpv2c-0.11/src/NwGtpv2cMsg.c:0146    Purging message 7f6478011560!
001461 00102:197707 7F6494E1B700 DEBUG GTPv2- nwgtpv2c-0.11/src/NwGtpv2cTrxn.c:0328   Purging  transaction 0x0x7f6478011d0
001462 00102:197712 7F6494E1B700 DEBUG GTPv2- /nwgtpv2c-0.11/src/NwGtpv2cMsg.c:0134    Created message 0x7f6478011560!
001463 00102:197723 7F6494E1B700 DEBUG GTPv2- .11/src/NwGtpv2cMsgIeParseInfo.c:0665    Received IE 2 with instance 0 of length 2 in msg-type 35!
001464 00102:197729 7F6494E1B700 TRACE GTPv2- 2-c/nwgtpv2c-0.11/src/NwGtpv2c.c:0770    Entering nwGtpv2cSendTriggeredRspIndToUlp()
001465 00102:197734 7F6494E1B700 DEBUG S11    penair-cn/src/s11/s11_mme_task.c:0072      Received triggered response indication
001466 00102:197875 7F6494E1B700 DEBUG GTPv2- v2c-0.11/src/NwGtpv2cMsgParser.c:0203     Received IE 2 of length 2!
001467 00102:197881 7F6494E1B700 DEBUG S11    ir-cn/src/s11/s11_ie_formatter.c:0441          - Cause 16
001468 00102:197893 7F6494E1B700 DEBUG GTPv2- /nwgtpv2c-0.11/src/NwGtpv2cMsg.c:0146      Purging message 7f6478011560!
001469 00102:197906 7F6494E1B700 TRACE GTPv2- 2-c/nwgtpv2c-0.11/src/NwGtpv2c.c:0770    Leaving nwGtpv2cSendTriggeredRspIndToUlp() (rc=0)
001470 00102:197911 7F6494E1B700 TRACE GTPv2- 2-c/nwgtpv2c-0.11/src/NwGtpv2c.c:1187  Leaving nwGtpv2cProcessUdpReq() (rc=0)
001471 00110:225070 7F647F7FE700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0033    ======================================== STATISTICS ========================================

001472 00110:225961 7F647F7FE700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0034                |  Current Status| Added since last display|  Removed since last display |
001473 00110:225971 7F647F7FE700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0036    Connected eNBs |        1       |           0            |           0            |
001474 00110:225986 7F647F7FE700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0038    Attached UEs   |        1       |           1            |           0            |
001475 00110:225996 7F647F7FE700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0040    Connected UEs  |        1       |           1            |           0            |
001476 00110:226001 7F647F7FE700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0042    Default Bearers|        1       |           1            |           0            |
001477 00110:226006 7F647F7FE700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0044    S1-U Bearers   |        1       |           1            |           0            |

001478 00110:226019 7F647F7FE700 DEBUG MME-AP src/mme_app/mme_app_statistics.c:0045    ======================================== STATISTICS ========================================
```

FIGURE C.3: MME registering the UE to the network

C.4 eNodeB and MME during Checkpoint

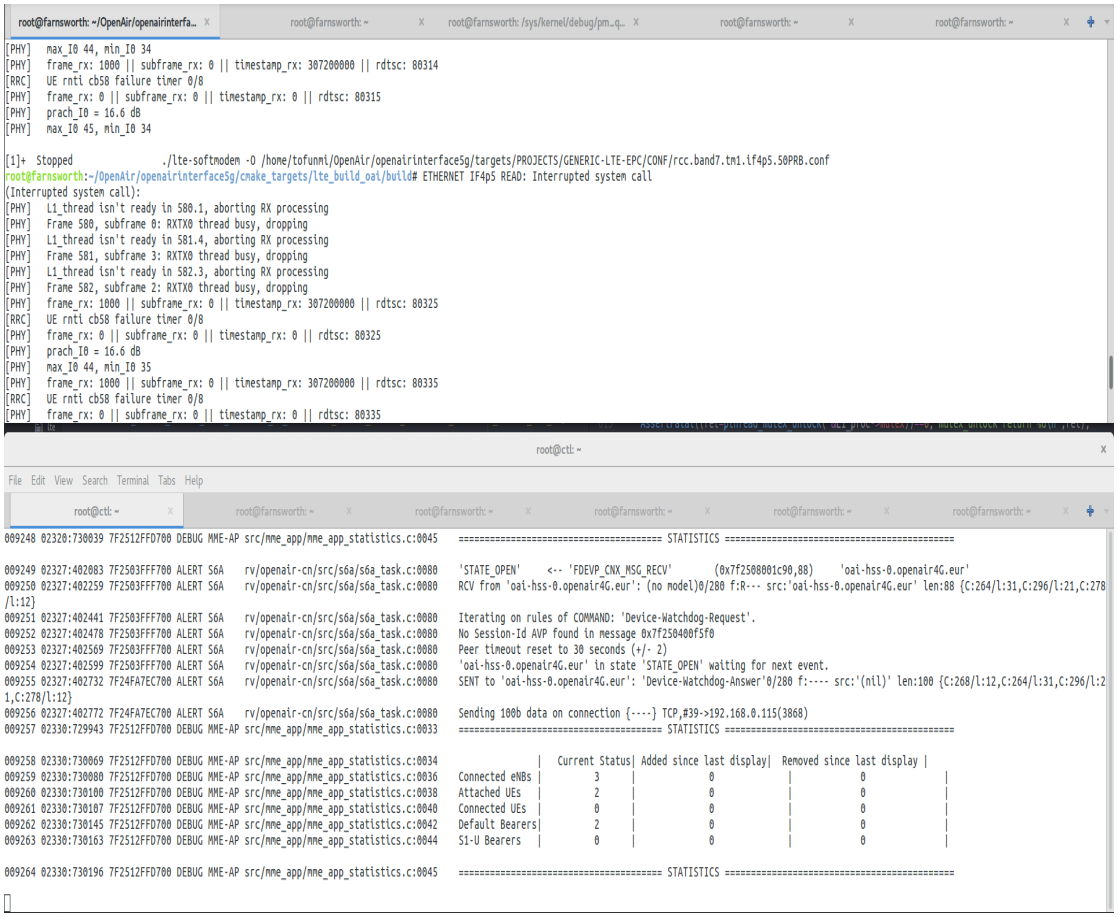

FIGURE C.4: Post-Checkpoint eNode and EPC Log

Bibliography

[1] Konstantinos Alexandris, Navid Nikaein, Raymond Knopp, and Christian Bonnet. Analyzing x2 handover in lte/lte-a. *2016 14th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, pages 1–7, 2016.

[2] Aleksandra Checko, Henrik Lehrmann Christiansen, Ying Yan, Lara Scolari, Georgios Kardaras, Michael S. Berger, and Lars Dittmann. Cloud ran for mobile networksa technology overview. *IEEE Communications Surveys Tutorials*, 17:405–426, 2015.

[3] Navid Nikaein, Eryk Schiller, Romain Favraud, Raymond Knopp, Islam Alyafawi, and Torsten Braun. Towards a cloud-native radio access network. 2017.

[4] Dimitrios Pliatsios, Panagiotis Sarigiannidis, Sotirios Goudos, and George K. Karagiannidis. Realizing 5g vision through cloud ran: technologies, challenges, and trends. *EURASIP Journal on Wireless Communications and Networking*, 2018 (1):136, May 2018. ISSN 1687-1499. doi: 10.1186/s13638-018-1142-1. URL `https://doi.org/10.1186/s13638-018-1142-1`.

[5] G. C. Valastro, D. Panno, and S. Riolo. A sdn/nfv based c-ran architecture for 5g mobile networks. In *2018 International Conference on Selected Topics in Mobile and Wireless Networking (MoWNeT)*, pages 1–8, June 2018. doi: 10.1109/MoWNet.2018.8428882.

[6] Eryk Schiller, Islam Alyafawi, Torsten Braun, Navid Nikaein, Andr Gomes, and Desislava Dimitrova. Critical issues of centralized and cloudified lte-fdd radio access networks. 06 2015. doi: 10.1109/ICC.2015.7249202.

[7] Eryk Schiller, Navid Nikaein, R Knopp, L Gauthier, Torsten Braun, Dominique Pichon, Christian Bonnet, Florian KALTENBERGER, and Dominique Nussbaum. Demo – closer to cloud-ran: Ran as a service. 09 2015. doi: 10.1145/2789168.2789178.

[8] Alcatel Lucent. The lte network architecturea comprehensive tutorial. 2009. URL `http://www.cse.unt.edu/~rdantu/FALL_2013_WIRELESS_NETWORKS/LTE_Alcatel_White_Paper.pdf`.

[9] Sassan Ahmadi. Chapter 3 - e-utran and epc protocol structure. In Sassan Ahmadi, editor, *LTE-Advanced*, pages 121 – 152. Academic Press, 2014. ISBN 978-0-12-405162-1. doi: https://doi.org/10.1016/B978-0-12-405162-1.00003-4. URL `http://www.sciencedirect.com/science/article/pii/B9780124051621000034`.

[10] A Perez. *LTE and LTE Advanced: 4G Network Radio Interface*. 11 2015. doi: 10.1002/9781119145462.

[11] Sassan Ahmadi. Chapter 2 - network architecture. In Sassan Ahmadi, editor, *LTE-Advanced*, pages 29 – 119. Academic Press, 2014. ISBN 978-0-12-405162-1. doi: https://doi.org/10.1016/B978-0-12-405162-1.00002-2. URL `http://www.sciencedirect.com/science/article/pii/B9780124051621000022`.

[12] Hamid Mousavi, Iraj Sadegh Amiri, Mohammad Ali Mostafavi, and Chang Yoong Choon. Lte physical layer: Performance analysis and evaluation. 2017.

[13] Srikanth H Kamath and Deepashikha. Decoding of pbch in lte. 2014.

[14] Navid Nikaein, M Marina, S Manickam, A Dawson, Raymond Knopp, and Christian Bonnet. OpenAirInterface: A flexible platform for 5G research. *ACM Sigcomm Computer Communication Review, Volume 44, N5, October 2014*, 10 2014. doi: http://dx.doi.org/10.1145/2677046.2677053. URL `http://www.eurecom.fr/publication/4434`.

[15] C. Esposito, A. Castiglione, and K. R. Choo. Challenges in delivering software in the cloud as microservices. *IEEE Cloud Computing*, 3(5):10–14, Sep. 2016. ISSN 2325-6095. doi: 10.1109/MCC.2016.105.

[16] M. Villamizar, O. Garcs, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In *2015 10th Computing Colombian Conference (10CCC)*, pages 583–590, Sep. 2015. doi: 10.1109/ColumbianCC.2015.7333476.

[17] Jude Okwuibe, Juuso Haavisto, Erkki Harjula, Ijaz Ahmad, and Mika Ylianttila. Orchestrating service migration for low power mec-enabled iot devices. *CoRR*, abs/1905.12959, 2019. URL `http://arxiv.org/abs/1905.12959`.

[18] Gabriel Brown. Huawei white paper: Cloud ran the next-generation mobile network architecture. 04 2017.

[19] Ericsson. Cloud ran: the benefits of virtualization, centralization and coordination[online]. 2015. URL `https://focustech.it/download/tecnologia-cloud-ran.pdf`.

[20] China Mobile Research Institute. C-ran white paper: The road towards green ran [online]. 2014. URL `https://pdfs.semanticscholar.org/eaa3/ca62c9d5653e4f2318aed9ddb8992a505d3c.pdf`.

[21] S. S. Kumar, R. Knopp, N. Nikaein, D. Mishra, B. R. Tamma, A. A. Franklin, K. Kuchi, and R. Gupta. Flexcran: Cloud radio access network prototype using openairinterface. In *2017 9th International Conference on Communication Systems and Networks (COMSNETS)*, pages 421–422, Jan 2017. doi: 10.1109/COMSNETS.2017.7945423.

[22] Navid Nikaein, R Knopp, Florian Kaltenberger, L Gauthier, Christian Bonnet, Dominique Nussbaum, and Riadh Ghaddab. Demo: Openairinterface: an open lte network in a pc. *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM*, 09 2014. doi: 10.1145/2639108.2641745.

[23] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. The design and implementation of open vswitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130, Oakland, CA, May 2015. USENIX Association. ISBN 978-1-931971-218. URL `https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff`.

[24] E. Schiller, N. Nikaein, E. Kalogeiton, M. Gasparyan, and T. Braun. Cds-mec: Nfv/sdn-based application management for mec in 5g systems. *Computer Networks*, 135:96 – 107, 2018. ISSN 1389-1286. doi: https://doi.org/10.1016/j.comnet.2018.02.013. URL `http://www.sciencedirect.com/science/article/pii/S138912861830080X`.

[25] C. Wang, Y. Wang, C. Gong, Y. Wan, L. Cai, and Q. Luo. A study on virtual bs live migration a seamless and lossless mechanism for virtual bs migration. In *2013 IEEE 24th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 2803–2807, Sep. 2013. doi: 10.1109/PIMRC.2013.6666624.

[26] R Buvaneswaran and S Srikanth. Cell search and uplink synchronization in lte. 2013.

Declaration of consent

on the basis of Article 30 of the RSL Phil.-nat. 18

Name/First Name: AJAYI Jesutofunmi Ademiposi

Registration Number: 15-126-840

Study program: Computer Science

Bachelor ☐ Master ☑ Dissertation ☐

Title of the thesis: Technical Requirements, Design, and Implementation of the live eNB container migration in LTE Mobile Networks

Supervisor: Prof. Dr. Torsten Braun

I declare herewith that this thesis is my own work and that I have not used any sources other than those stated. I have indicated the adoption of quotations as well as thoughts taken from other authors as such in the thesis. I am aware that the Senate pursuant to Article 36 paragraph 1 litera r of the University Act of 5 September, 1996 is authorized to revoke the title awarded on the basis of this thesis.

For the purposes of evaluation and verification of compliance with the declaration of originality and the regulations governing plagiarism, I hereby grant the University of Bern the right to process my personal data and to perform the acts of use this requires, in particular, to reproduce the written thesis and to store it permanently in a database, and to use said database, or to make said database available, to enable comparison with future theses submitted by others.

Bern, 09/09/2019

Place/Date

Signature