WRR based Two-Level Slice Scheduler for 5G RAN Sharing in the context of Neutral Host

by

Yekta Demirci

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2021

Author's Declaration

Abstract

We investigate the sub-channel sharing problem in the context of Neutral Host Radio Access Network (RAN) slicing. We consider a single user Multiple Input Multiple Output system where each Virtual Network Operator (VNO) brings their own sub-channels and they use the RAN infrastructure in a common way. We assume that the power budget is constant per Physical Resource Block (PRB). We consider only the Downlink and a system made of one single cell. We assume, the RAN infrastructure is operated by the Neutral Host. First, we consider the State of the Art (SOA), static sub-channel allocation where the VNOs are allocated only the sub-channels they brought. In the case of SOA, if a VNO experiences no traffic then the sub-channels will remain idle and the resources would be wasted. Considering this we propose a two-level-slice scheduler to utilize the idle resources better than the SOA. We implement the proposed scheduler in an open source RAN platform, Open-Air-Interface (OAI). We provid detailed set-up guides for the OAI platform in emulation and with Commercial-Off-The-Shelf (COTS) hardware. Finally we compare the performance of the SOA and the proposed scheduler in terms of system throughput and delay in four different scenarios. The proposed two-level-slice scheduler provides strictly less delay than the SOA in all the experiments.

Acknowledgements

Anneme ve aileme...

To my family and to my mother who will keep living in our hearts and memories...

I would like express my thanks to my supervisors Professor Mahesh Tripunitara and Professor Catherine Rosenberg for providing me a chance to conduct real research and supporting me through my one of the most challenging times.

I also thank my amazing friends who supported me thoroughly during my journey. I could not be where I am without your help. I kindly express my gratitude to you in my heart.

Table of Contents

List of Figures

List of Abbreviations

3GPP	3^{rd} Generation Partnership Project
4G	4^{th} Generation
5G	5^{th} Generation
ARQ	Automatic Repeat Request
BSS	Business Support System
CAPEX	CAPital EXpenditure
CN	Core Network
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CQI	Channel Quality Indicator
DCI	Downlink Control Information
DCN	Dedicated Core Network
DL	Down-Link
eMBB	enhanced Mobile BroadBand
EMM	Enterprise Mobility Management
eNB	Evolved Node B
ePC	Evolved Packet Core

ETSI	European Telecommunications Standards Institute
FDD	Frequency Division Duplex
FEC	Forward Error Correction
gNB	NR Node B
HARQ	Hybrid Automatic Repeat Request
HSS	Home Subscriber Server
ICCID	Integrated Circuit Card IDentifier
IMSI	International Mobile Subscriber Identity
IP	Internet Protocol
KI	Individual subscriber authentication Key
LAN	Local Area Network
LTE	Long-Term Evolution
M-MIMO	Massive-Multiple Input Multiple Output
MANO	MANagement and Orchestration
MCC	Mobile Country Code
MCS	Modulation and Coding Scheme
mIoT	massive Internet of Things
MNC	Mobile Network Code
MSIN	Mobile Subscriber Identification Number
nFAPI	network Functional Application Platform Interface
NFV	Network Functions Virtualization
NR	New Radio

O-RAN	Open-Radio Access Network
OAI	Open Air Interface
OPc	Operator Code
OPEX	OPerating EXpenses
OS	Operating System
OSS	Operational Support System
PLMN	Public Land Mobile Network
PRB	Physical Resource Block
QoS	Quality of Service
RAN	Radio Access Network
RAT	Radio Access Technology
RBG	Resource Block Group
REST	REpresentational State Transfer
RRC	Radio Resource Control
SC	Software Community
SDR	Software Defined Radio
SIM	Subscriber Identity Module
SISO	Single Input Single Output
SLA	Service Level Agreement
TDD	Time Division Duplex
UE	User Equipment
UL	Up-Link
URLLC	Ultra-Reliable Low Latency Communications

USB	Universal Serial Bus
USIM	Universal Subscriber Identity Module
USRP	Universal Software Radio Peripheral
VM	Virtual Machine
VNO	Virtual Network Operator
WRR	Weighted Round Robin

Chapter 1

Introduction

1.1 Overview

Fifth Generation (5G) networks are embracing the Network Function Virtualization (NFV) which consists of decomposing large monolithic network functions into software-based modular network functionalities [2]. This empowers network slicing which is about creating several end-to-end logical networks on top of a common physical infrastructure. From a business perspective, network slicing enables new offerings. Two such offerings are external slicing to industrial/business customers and external slicing to Virtual Network Operators (VNOs) in a Neutral Host (NH) context. Slicing can also be used internally to enable an operator to offer different classes of service.

In this thesis, we focus on external slicing in the context of a NH where an infrastructure owner provides Radio Access Network (RAN) service to different VNOs at a venue. The venue can be a university campus, a shopping mall or a hospital. More precisely, the infrastructure owner which we call the NH in the following, provides RAN infrastructure composed of Base Band Units (BBU), Radio Units (RUs), coaxial cables and power to the VNOs whereas the VNOs bring their own spectrum. The VNOs share the RAN infrastructure to offer cellular service for their subscribers in the venue. Consequently, the NH provides cellular service to the subscribers of VNOs using the common infrastructure and the spectrum brought by the VNOs. The NH and the VNOs can communicate using the Operation/Business Support System (OSS/BSS) unit. The OSS/BSS unit is responsible for the management of the slices and it can work as a bridge between the NH and the VNOs via some Application Programming Interface (API). Some of the Core Network (CN) functions can be either in common or can be slice-specific as can be seen in Figure 1.1. More information about the CN functions as well as the 5G network architecture can be found in [3] from where Figure 1.1 is derived.

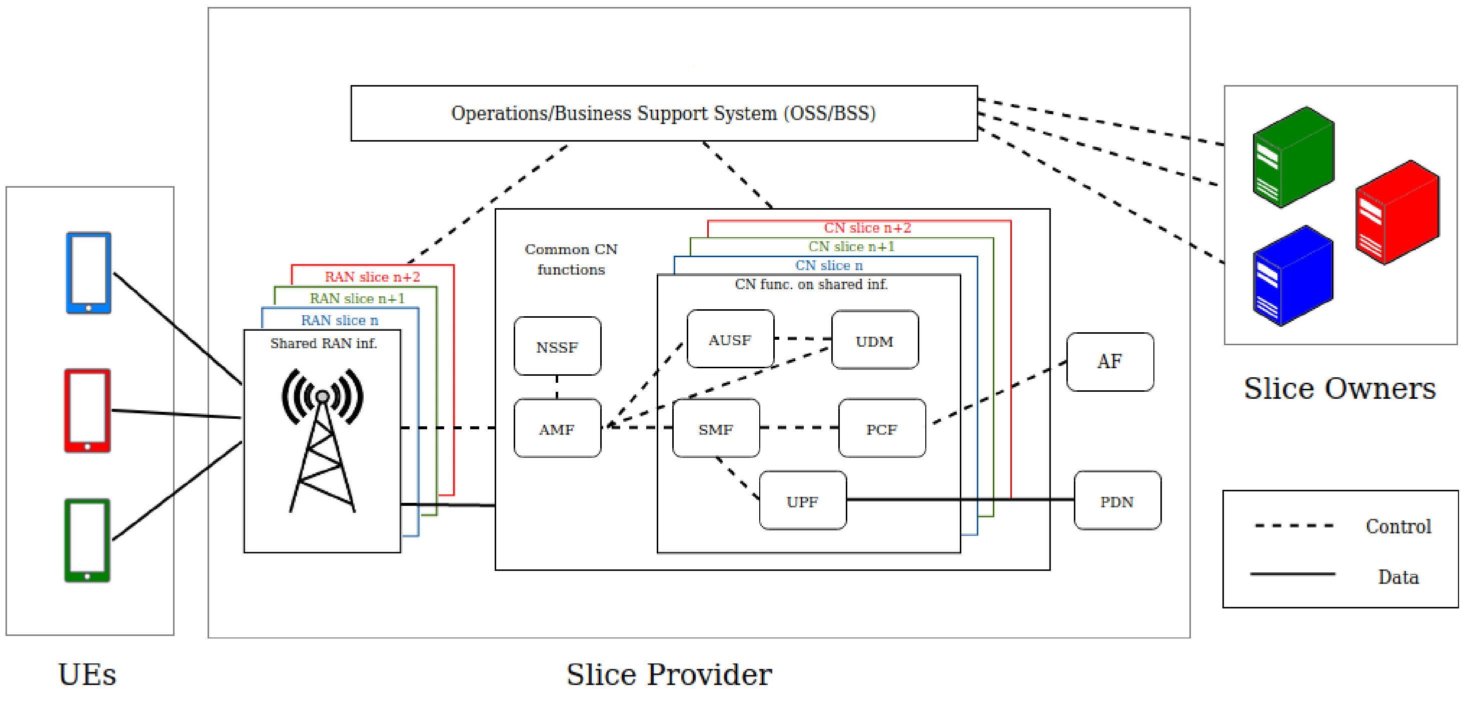


Figure 1.1: 5G network architecture with network slicing

Designing and operating a NH is not without challenge. Some of the challenges are

- Creating the process to start and close new slices (for new VNOs).
- Resource management during operation.
- Design of enforceable Service Level Agreements (SLAs) between the VNOs and the NH.
- Ensuring data-privacy among different VNOs as they access to a common infrastructure.

In this thesis we focus on the challenge of resource management among the slices owned by different VNOs in the Downlink (DL) of a single cell using single user Single Input Single Output (SISO) system. As the VNOs share a common infrastructure, it is important to ensure that operation of a VNO is not affected by the traffic of others. In other words, different slice owners should be able to use their resources concurrently without interfering with each other and there should be an isolation between different slices.

In the NH context, the State Of the Art (SOA) is to restrict the use of the spectrum to the VNO that brings it. There is a scheduler per slice that allocates the corresponding Physical Resource Blocks (PRB) to the User Equipments (UE)s of the VNO. In this thesis we consider a Round Robin (RR)-based algorithm for that scheduler. A PRB is a two-dimensional radio resource which consists of frequency and time. The frequency domain is represented in terms of some sub-channels and the time domain is represented in terms of sub-frame.

Even though the SOA provides perfect isolation in terms of sub-channel usage between different slices, it has some limitations. For instance, if a VNO has no-traffic, then its sub-channels would be wasted. Therefore in this thesis we ask the following **research question**: Can the VNOs share their sub-channels and can a scheduler be designed that has the following properties:

- C1: Do not give any resources of VNO q away if any of the VNO's subscribers have any non-empty DL buffers at the base station. (This provides isolation)
- C2: Do not waste any PRBs if at least one subscriber has non-empty DL buffer at the base station. (This provides better performance)
- C3: Free riders (i.e. VNOs that have more traffic load per sub-channel than others) should not be overly rewarded even if it is understood that Condition 2 (C2) implies that free riders will benefit.

To answer this question we propose a two-level scheduler where the first level scheduler adopts a Weighted Round Robin (WRR)-based algorithm to allocate sub-channels to the slices. Once the first level scheduling is completed, the second level scheduler allocates the PRBs consist of the allocated sub-channels to the UEs of the respective VNO based on a RR-based algorithm at a given sub-frame.

Specifically, we make the following contributions:

- We propose a two-level scheduler which embraces Weighted Round Robin (WRR) algorithm.
- We implement the proposed scheduler in a platform called Open-Air-Interface (OAI) and contribute to an open-source project.
- We design and run several experiments to compare the performance of the proposed scheduler to the SOA in the OAI emulation.

- We provide also detailed guides to set-up a private Long-Term Evolution (LTE) network in the emulation mode as well as with COTS UEs and bladeRF Software Defined Radio (SDR) hardware.

1.2 Outline

The remainder of the thesis is structured as follows. In Chapter 2, we provide the literature survey. In Chapter 3, we provide the system model and the slice schedulers. In Chapter 4, we introduce an open-source RAN platform called OpenAirInterface (OAI) and we explain how to set-up a private LTE network with network slicing both in the emulation and with actual COTS hardware. In Chapter 5, we explain how we implemented the proposed scheduler in OAI. In Chapter 6, we compare the performances of the slice schedulers. In Chapter 7, we conclude the thesis and discuss some future research directions.

Chapter 2

Related Work

2.1 Literature Review

2.1.1 Network sharing in 4G

Network slicing in 5G, inherits some existing network sharing solutions from 4G and enhances them with end-to-end partitioning to provide higher levels of isolation. For instance, 4G offers the concept of Dedicated Core Network (DCN) which enables deploying more than one DCN within a Public Land Mobile Network (PLMN) [2]. Each DCN can be dedicated to a specific type of application or subscriber where control and user plane functions can be customized depending on different needs. Yet, DCN selection is performed by the control plane function of the default DCN which results poor isolation between different DCNs. Considering RAN sharing, 4G offers the concept of Multi-Operator Core Network (MOCN) where different operators can use the same RAN infrastructure. The shared RAN forwards the traffic of different operators to their corresponding CNs. However there is no isolation provided between different network operators at the RAN-site [4]. Moreover, as it can be seen in figure 2.1 (the figure 2.1 is inspired from [4]), MOCN provides only a domain level sharing unlike network slicing in 5G.

To sum up, 4G provides some network sharing solutions at specific domain levels, however none of these solutions provide an end-to-end partitioning like the notion of "network slicing" in 5G. Additionally, in 4G there is no explicit way to distinguish traffics with different requirements which generated from the same UE. This creates a problem if a UE would need to connect to different slices to meet various service requirements. Therefore, it is not possible to fully embody network slicing in 4G. However, some early frameworks are implemented where a UE can only connect to a single slice.

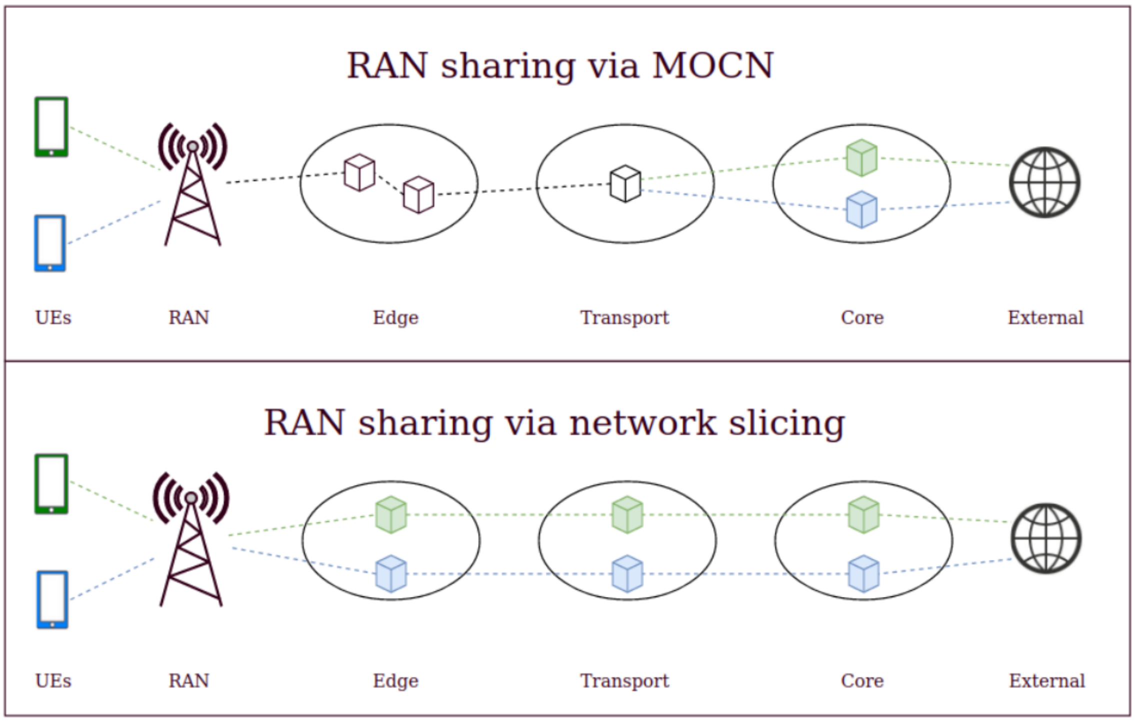


Figure 2.1: MOCN vs network slicing

2.1.2 RAN sharing in the context of NH

The authors in [5] conceived a small cells wireless network deployment framework at a venue where the customers share the spectrum. On top of the conceived framework, they also provide a complementary business model, referred as NH micro operator that leverages a single shared wireless infrastructure. In [6], the authors proposed an orchestration and Virtual Infrastructure Management (VIM) solution to address some of the open questions identified by European Telecommunications Standards Institute (ETSI) in order to pave the way for 5G NH settings. In [7], the creators of flexRAN platform provided a solution that combines a NH based shared small-cell infrastructure with a common pool of spectrum for dynamic sharing. They proposed a shared spectrum access architecture which uses a reinforcement learning-based dynamic pricing mechanism to mediate access to the shared spectrum.

2.1.3 Network slicing in 5G

Network slicing concept has drawn considerable attention from both academia and the industry. The early publications highlight how the network slicing would shape 5G and the challenges ahead as well as some possible research directions [2], [8], [9]. These papers provide a high level overview of network slicing at the different domains of 5G networks. There are also some works which specifically target radio access domain such as [10]. Unlike the other papers, the authors considered the unique resource of RAN spectrum, and they explained some possible configurations for Layer 1 (L1), L2, L3. However, all of these papers provide only a high level conceptual work and there is still no gold standard for 5G RAN slicing architecture [8].

2.1.4 RAN slicing frameworks

The authors in [11] proposed a Software Defined RAN platform called FlexRAN. FlexRAN provides a flexible control plane designed with support for real-time RAN control applications, following the NFV principles [11]. Besides its NFV based characteristics, the flexRAN platform also supports slicing at the RAN domain. With the flexRAN framework, an UE can connect to a single slice therefore this platform can be considered as an early prototype for 5G RAN slicing. In [12] the authors extended the functionalities of the proposed platform further. Recently, the authors from University of Utah [13] published a paper where they implemented a network controller with RAN slicing functionalities based on the O-RAN SC specifications. Even though they make a discussion about 5G, they used 4G RAT in their experiments.

2.1.5 SLAs in the context of RAN sharing

Considering SLAs in the context of 5G network slicing, there have been some publications, as well. The authors in [14] mentioned the importance of enforcing network slicing and they explicitly considered RAN. The authors in [15] considered an end-to-end SLA which is expected to define reliability, availability and performance of the delivered telecommunication services. The authors in [16] proposed an adaptation algorithm to create an abstraction layer to allocate radio resources to the slices based on minimizing deviations from some requirements such as latency, coverage, energy efficiency. In another work [17], the authors examined three distinct service requirements and proposed a slice scheduling solution. In [18] the authors revised the SLA-related models and workflows for the case of 5G slicing and proposed an adaptive Quality of Service (QoS) parameter computation formula for mapping of low-level metrics to high-level parameters. However none of these works took into account the fact that physical channel conditions may vary beyond the control of the slice provider. Therefore guaranteeing some performance-based metrics might be infeasible in real life scenarios. Our proposed two-level scheduler guarantees a resource based objective rather than a performance based one.

Chapter 3

The RAN Model and the Schedulers

3.1 The RAN model

We consider a system made of one single cell. We use a Single User Multiple Input Multiple Output (SU-MIMO) system. We focus on the DownLink (DL).

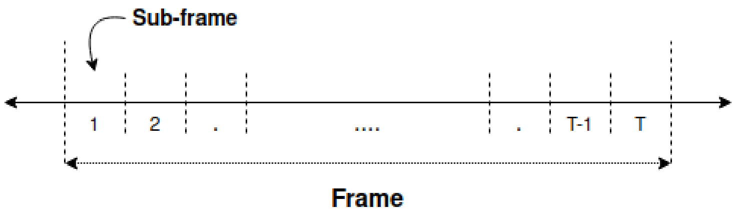


Figure 3.1: Time structure

We consider a frame made of T sub-frames as in Figure 3.1. We consider a PRB as a two dimensional radio resource which is made of one sub-channel and one sub-frame as it can be seen in Figure 3.2, additionally we assume a power per PRB = P_c. With a sub-frame periodicity, the eNB sends Downlink Control Indicator (DCI) messages which we call "scheduling maps". The purpose of the scheduling maps are to inform the UEs about the PRBs they are allocated in the next sub-frame so that they can listen to their respective PRB and decode their DL data correctly.

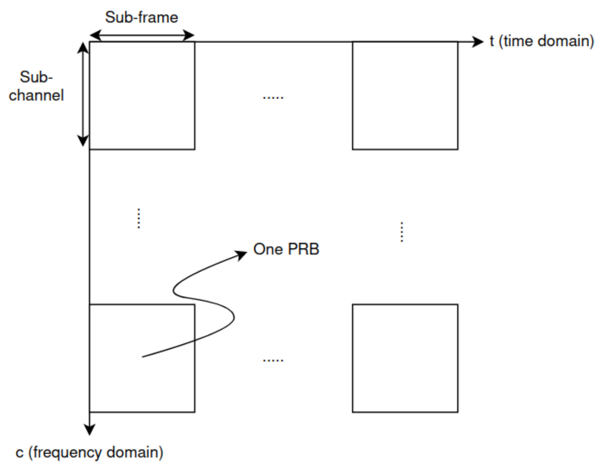


Figure 3.2: Physical Resource Block (PRB)

We consider Q VNOs where each VNO brings its sub-channels. Let C_q be the number of sub-channels brought by VNO q. Let m_q be the number of UEs of VNO$_q$. We consider that each VNO has a slice where each slice has a Meta-Buffer (MB). More precisely we consider the following buffer organization at the Base Station (BS).

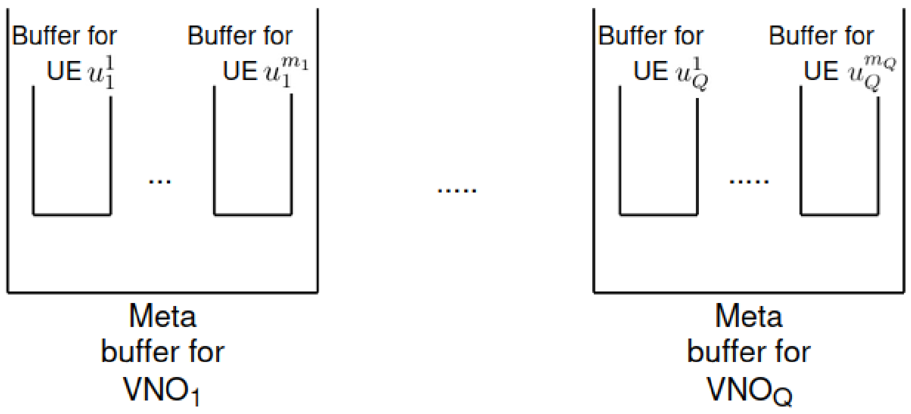


Figure 3.3: Organization of the DL buffers at the BS

The minimum number of PRBs to be allocated at a time may vary depending on the configuration of the Radio Access Technology (RAT) (e.g. 3 or 5 PRBs) To avoid confusion, we assume in the following discussion that we can allocate individual PRBs.

3.2 The Slice-Schedulers

We consider two slice schedulers: (i) The State Of the Art and (ii) the proposed two-level scheduler.

Let $\mathcal{C}_q$ be the set of sub-channels brought by VNO q, where

$$|\mathcal{C}_q| = C_q$$

3.2.1 The State of the Art

The State Of the Art (SOA) scheduler statically allocates the sub-channels $\mathcal{C}_q$ to VNO q and this allocation does not change in time. In other words, UEs of VNO q can only be allocated the PRBs that consist of $\mathcal{C}_q$ at a given sub-frame. More formally:

PRB(c,t) is allocated to the meta-buffer of VNO q if and only if $c \in \mathcal{C}_q, \forall t \in \{1, .., T\}$

Since the allocation of $\mathcal{C}_q$ does not vary in time and $\mathcal{C}_q$ is known apriori, the SOA scheduler is only required to allocate the PRBs that consist of $\mathcal{C}_q$ to the UEs of VNO q at a given sub-frame. We call the scheduler that allocates PRBs to the UEs as the "low-level scheduler" in the following.

The Low-level scheduler:

There is a separate low-level scheduler for each VNO. The low-level scheduler allocates the PRBs that consist of $\mathcal{C}_q$ to the UEs of VNO q based on Round Robin (RR) at every sub-frame. There exists a vector of UEs belonging to VNO q. The RR-based algorithm allocates PRBs to the UEs starting from a pointer position (based on the previous frame) which makes a cycle over the vector. The pointer makes a circular cycle with modulo (m$_q$) meaning that when the pointer reaches to the end of the vector, it goes back to the first index. More specifically, at the end of sub-frame $t-1$ if the pointer was on UE x, then at the beginning of sub-frame t the pointer will point to UE x+1 (modulo m$_q$). If an UE has no bits in its buffer nor any re-transmission request then the scheduler will skip allocating a PRB to this UE and it will continue with the RR cycle.

Let $\mathcal{U}_q$ be the set of UEs belonging to VNO q and let p_{rr} be the pointer that points to an UE in $\mathcal{U}_q$.

$$\mathcal{U}_q = \{u_q^1, u_q^2, ..., u_q^{m_q-1}, u_q^{m_q}\}$$

∀ sub-frame f, do the following at the low-level scheduler of VNO q:

Algorithm 1: The low-level scheduler: A Round Robin algorithm

```
1  while (PRBs (composed of $\mathcal{C}_q$) at $MetaBuffer_q$ >0) AND (Bits to be
     transmitted at $MetaBuffer_q$ >0) do
2      Update $p_{rr}$ to point to the next element in $\mathcal{U}_q$ (modulo $m_q$);
3      if The UE pointed by $p_{rr}$ requests HARQ re-transmission then
4          Call HARQ re-transmission sub-routine;
5          Update # of available PRBs at $MetaBuffer_q$;
6      else
7          if Bits in the buffer of the UE pointed by $p_{rr}$ >0 then
8              Allocate one available PRB to the UE pointed by $p_{rr}$;
9              Free some bits from the UE buffer according to the CQI value of
                 the UE;
10             Update # of bits to be transmitted at $MetaBuffer_q$;
11             Update # of available PRBs at $MetaBuffer_q$;
```

At the line 4 of Algorithm 1, we use HARQ sub-routine. HARQ stands for Hybrid-Automatic Repeat reQuest. It is a combination of Automatic Repeat reQuest (ARQ) and Forward Error Correction (FEC) protocols. HARQ is a fairly complicated process and its implementation differs for DL/UpLink (UL) or Frequency Division Duplex (FDD) or Time Division Duplex (TDD). As a part of the ARQ protocol, when a transmitter sends a packet then it starts a timer to track a timeout period. If the transmitter does not receive any Acknowledgement (ACK) from the receiver before the timeout period, then it re-transmits the packet. As a part of the FEC protocol, the receiver can combine two or more received packets to recover a corrupted packet. The detailed implementation of a HARQ process is beyond the scope of this work, however the following 3GPP document [19] can be referred for further information about HARQ process in LTE-RAT system.

At the line 9 of Algorithm 1, we use Channel Quality Indicator (CQI). CQI is reported from an UE to the base station to state how the physical channel quality is. According to the CQI value, the base station inserts varying amount of bits from the UE buffers into a PRB. If the channel conditions are good, the base station can achieve better code-rates than a bad channel meaning that less redundancy bits are inserted into a PRB. As less redundancy occurs during channel coding, more bits can be sent from the UE buffers. The relation between the CQI and the code rate is determined based on a table provided by 3GPP. The 3GPP document [20] can be referred for further information about CQI and channel coding for a LTE-RAT system.

3.2.2 The proposed Two-Level-Scheduler

The proposed 2-Level-Scheduler consist of two schedulers: (i) the meta-scheduler which allocates PRBs to the MBs of the slices at a given sub-frame, (ii) the low-level scheduler which allocates the given PRBs for a slice to the respective UEs.

Meta-scheduler

The meta-scheduler adopts a Weighted Round Robin (WRR) algorithm to allocate some PRBs to the meta buffers of VNO q. Let $\mathcal{W}$ be the vector of all the weights where w_q be the weight of VNO q. w_q can be determined based on the ratio between the number of sub-channels VNO q brings and the total number of sub-channels used in the system.

$$\mathcal{W} = \{w_1, w_2, ..., w_{Q-1}, w_Q\}$$

$$W = \sum_{q=1}^{Q} w_q$$

$$\mathcal{W}_{wrr} = \{1_1, ..., 1_{w_1}, 2_1, ..., 2_{w_2}, ..., Q_1, ..., Q_{w_Q}\}$$

Based on $\mathcal{W}$, the meta scheduler creates a vector $\mathcal{W}_{wrr}$ with the length of W. Each VNO has a certain number of opportunity to be allocated a PRB based on its weight. Let p_{wrr} be a pointer that points to an element in $\mathcal{W}_{wrr}$. Then, the pointer makes a cycle of length W and VNO q gets w_q opportunity to be allocated a PRB. The elements of $\mathcal{W}_{wrr}$ represent the Meta Buffers (MBs) of the VNOs. Consequently, the pointer p_{wrr} makes a cycle over the $\mathcal{W}_{wrr}$ with modulo W.

The meta-scheduler follows the Algorithm 2 at the beginning of each sub-frame. It is assumed that the following information is known apriori before running Algorithm 2.

- $\mathcal{C}$: The set of all the sub-channels used in the DL.
- $\mathcal{W}_{wrr}$: The vector that encodes the weights of the VNOs.
- p_{wrr} : The pointer which points to an element in $\mathcal{W}$.

At the beginning of $\forall$ sub-frame f, do the following:

Algorithm 2: The meta-scheduler: A WRR algorithm

```
1  Get the DL buffer status of each UEs ;
2  Get the HARQ process status ;
3  Get the current CQI values reported by the UEs ;
4  for Each Meta-Buffer (MB) do
5      for Each UE attached to the MB do
6          if HARQ re-transmission is needed for the current UE then
7              Calculate the # of required PRBs for re-transmission with the old
                 CQI value used in the first transmission
8          Calculate the # of required PRBs for the UE with the current CQI ;
9  Calculate # of PRBs needed per MB ;
10 Calculate the # of PRBs needed in total ;
11 while (Available PRBs >0) AND (total # of PRBs needed >0) do
12     Update $p_{wrr}$ to point to the next element in $U_{wrr}$ (modulo $|\mathcal{C}|$);
13     if # of PRBs needed for the pointed MB >0 then
14         Allocate one available PRB to the MB;
15         Update the # of PRBs needed for the MB ;
16         Update the # of PRBs needed in total ;
17 for Each MB do
18     Call the low-level scheduler sub-routine ;
```

Low-level scheduler

The low-level scheduler of the proposed Two-Level-Scheduler is the same as the low-level scheduler of SOA. It is the RR-based algorithm given in Algorithm 1. However it is important to mention that, in the case of SOA, the PRBs allocated to the UEs of VNO q always consist of $\mathcal{C}_q$. Whereas in the proposed scheduler, the PRBs allocated to VNO q are distributed based on the WRR algorithm. Therefore the PRBs may not be consisted of $\mathcal{C}_q$.

Chapter 4

The Open-Air-Interface (OAI) platform

Open-Air-Interface (OAI) is an open-source Software-Defined Radio (SDR) framework developed by EUROCOM to provide a flexible platform for 4G and 5G research. Initially, the OAI is developed for 4G RAT. It provides a full stack eNB that can be run either in emulation or on COTS hardware. Besides the eNB, OAI also offers UE, Evolved Packet Core (EPC) and real-time RAN controller FlexRAN. The FlexRAN controller is an interface between the eNB and the infrastructure administrator to change configuration of the eNB on the fly (e.g downlink-uplink schedulers, used bandwidth etc.).

Using OAI, it is possible to set-up a RAN in various modes. The full-stack eNB can be run on a COTS SDR (e.g bladeRF, NI B210) where COTS UEs can be used with the eNB. Additionally, the platform offers network Functional Application Platform Interface (nFAPI) emulation. nFAPI is a functional split between the MAC and PHY layers that enables virtualization of the MAC functions [21]. One of the important MAC functions is the radio resource allocation to UEs and with virtualization it is possible to run the MAC functions on different vendor equipment. nFAPI is provided by the Small Cell Forum to provide inter-operability and innovation among the different vendors. It leverages the architecture split of Option 6 specified by 3GPP which is provided in [22]. Different split options describe how the functions can be deployed at different logical nodes (e.g. Radio Unit) and how these logical nodes interrelate to one another. Consequently, OAI-nFAPI emulator provides an almost full-stack eNB with the exception of PHY layer. It is possible to run the emulation with EPC (S1-mode) or without EPC (noS1 mode). Furthermore, OAI eNB and FlexRAN support RAN slicing. This feature can be considered as a prototype for 5G slicing since an UE can only be attached to a single slice and it is not possible to fully embrace all the 5G slicing features with the eNB, yet.

Considering the 5G RAT, EUROCOM released a full-stack Stand-Alone (SA) gNB in the summer of 2021. However, by the time of writing this thesis, it does not support RAN slicing, yet. Furthermore, gNB (RAN) controller is still under development and it is not available either.

In this work, OAI platform is used to implement the proposed scheduler. The first reason of choosing OAI is that, it offers a full-stack eNB application & a real-time RAN controller. Additionally, both of the eNB and RAN controller applications support slicing at the RAN level. Secondly, the OAI platform provides an open-source repository which enables huge flexibility in terms of implementing new algorithms on top of the existing full-stack system. Thirdly, the OAI platform offers both emulation and non-emulation that runs on COTS hardware. Therefore it is possible to test the system in various configurations. Finally, the OAI platform for the 4G RAT is stable and there have been several papers in the academia where OAI platform is used; [11], [8], [12], [17], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33].

OAI 4G RAT is a SISO, LTE only system. We configured it in two ways (i) nFAPI emulation with EPC (S1-mode) and (ii) with COTS hardware. The components used in the emulation mode can be seen in Figure 4.1. In this first configuration, we emulate a full-stack private LTE network without the realization of the physical layer. We run an EPC, eNB, flexRAN controller and several UEs. Then we create a traffic between the UEs and the EPC. Further information about the number of UEs and the traffic generation can be found in Section 6. In the case of COTS hardware configuration, we used the components given in Figure 4.4. In this second configuration, we run Radio Unit (RU) part of the eNB on bladeRF SDR and connected a COTS UE to it.

The nFAPI emulation has the following advantages:

- The emulation provides an almost full-stack eNB platform with the exception of PHY layer.
- The source code is open-source and open to be manipulated. This gives full control over the eNB implementation.
- It provides many MAC layer functionalities including different HARQ re-transmission depending on the configuration.
- It supports multiple UE connection.
- It is possible to run an application on top of the emulated UEs to create DL traffic.
- It supports flexRAN controller and RAN slicing.

The nFAPI emulation has the following disadvantages:

- The nFAPI emulator does not provide a physical channel simulation.
- There is another mode of emulation (*phy_simulators*) which emulates an RF unit and some physical channel models. However, it supports only a single UE.

Consequently, the nFAPI emulator can be useful to test any functionality down to the MAC layer. Even though it does not provide a physical channel simulation, the physical channel characteristics can be partially mimicked by changing the CQI values reported to the MAC layer and dropping some packets received by the UEs.

In the following, how to set-up both OAI-nFAPI emulation and the COTS hardware with OAI are explained.

4.1 Kernel configuration

Before setting up the the OAI platform, the very first thing is to configure the Operating System (OS) kernel Ubuntu OS is required to run the OAI platform. Firstly, install Ubuntu 20.04.

```
Ubuntu 20.04.3 LTS
```

Then the next thing is to install the low-latency kernel.

```
$ sudo apt-get install linux-image-lowlatency linux-headers-
   lowlatency
```

It is required to choose the low-latency kernel every-time you boot the OS unless you permanently change the default kernel. The running kernel can be checked with the following command:

```
$ uname -r
4.15.0-156-lowlatency
```

Then the next step is to disable CPU frequency scaling. For the sake of simplicity and to avoid synchronizing issues, we use the maximum frequency that the computer can support with the following: **(It is required to scale the CPUs every-time the computer is rebooted)**

```
$ for GOVERNOR in /sys/devices/system/cpu/cpu*/cpufreq/
   scaling_governor; \
do \
    echo "performance" | sudo tee $GOVERNOR; \
done
```

After successfully setting the CPU scaling, CPUs should be working with the maximum supported frequency without much variation. It can be checked by:

```
$ watch grep \"cpu MHz\" /proc/cpuinfo
```

Given the current kernel is low-latency and the CPUs are working almost at a constant frequency, then the kernel setup is completed.

For further information, the following web-page can be used: https://gitlab.eurecom.fr/oai/openairinterface5g/-/wikis/OpenAirKernelMainSetup

4.2 OAI nFAPI emulation mode

It is possible to emulate eNBs and UEs down to nFAPI (layer 2). In other words, it is possible to emulate almost the full-stack of an eNB and UEs without the realization of physical layer or without needing any SDR or COT UE. The physical layer is replicated with some hard-coded MCS values and the rest of the stacks run as normal.

The nFAPI emulator can be run with or without an EPC (noS1 or S1 mode), however I found some bugs in the noS1-nFAPI emulator and I have reported them to the OAI community: https://lists.eurecom.fr/sympa/arc/openair5g-user/2021-05/msg00053.html. Due to the bugs, I used S1-nFAPI emulator (with EPC).

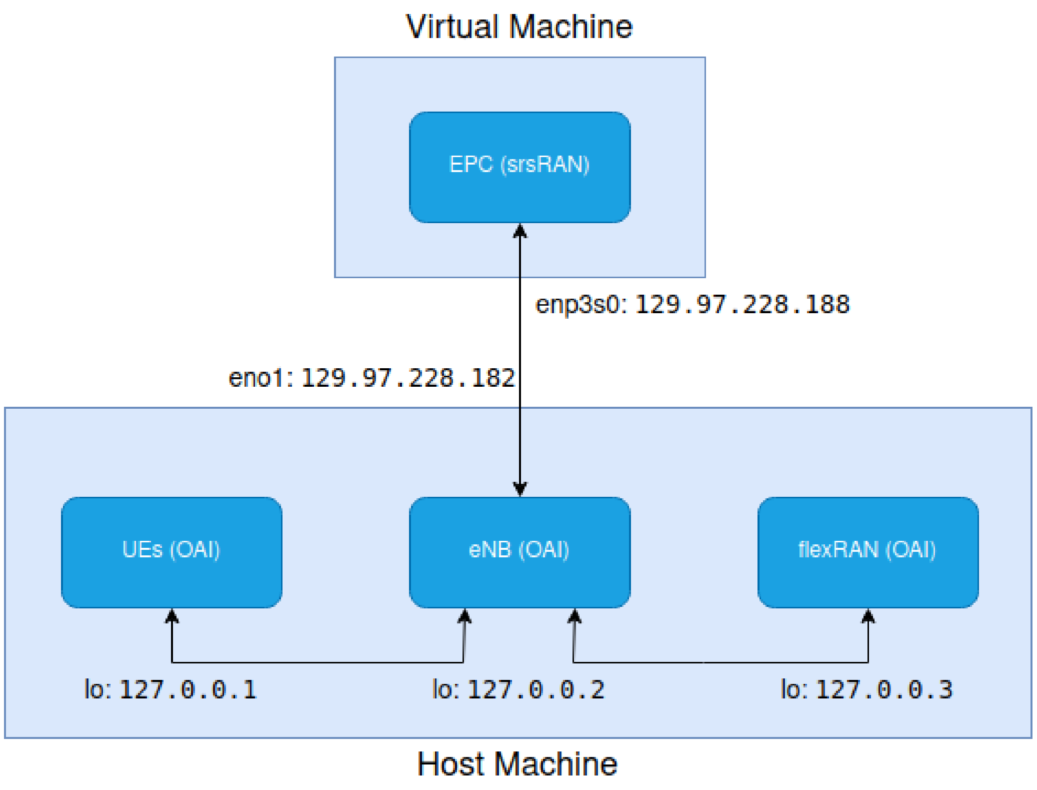
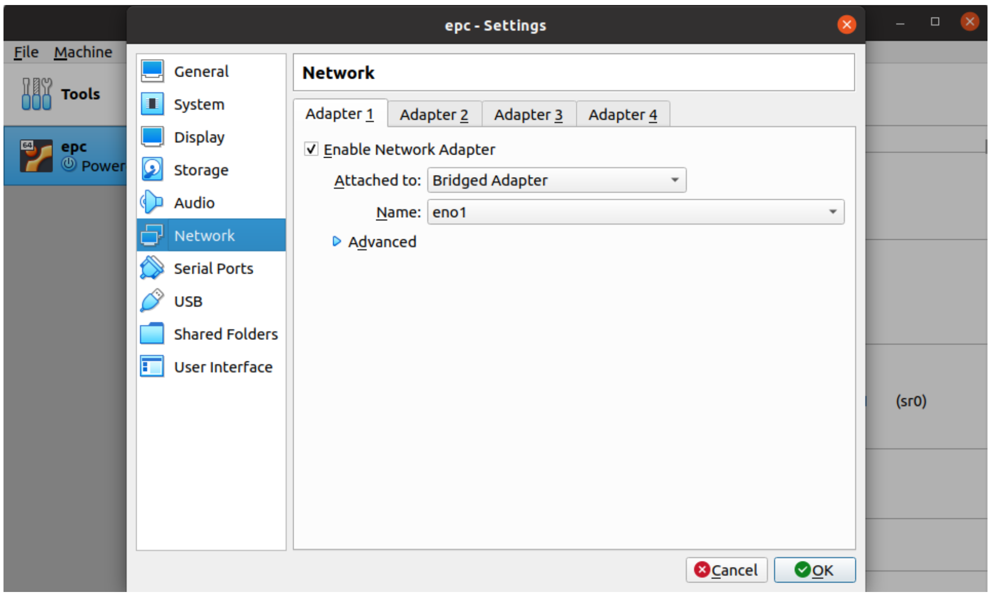


Figure 4.1: Interface names and IP addresses of the configured components

4.2.1 Configuring the EPC

Besides the eNB and UEs, OAI also offers an EPC solution. However, configuring the OAI EPC is fairly complex. I couldn't manage to configure the OAI EPC and decided to use srsEPC. It is lightweight enough to run on a virtual machine and it is easier to configure compared to OAI EPC.

Firstly, a virtual machine application is needed. In this work, (VM) VirtualBox, v6.1.26 is used. Once virtual machine is up and running, make sure its network is attached by a "*BridgedAdapter*". It can be enabled under Network settings.



Network

✓ Enable Network Adapter

Attached to: Bridged Adapter

Name: eno1

Figure 4.2: VM instance connected via Bridged Adapter

Once the VM is up and running, the following can be used to install and configure the srsEPC: https://docs.srsran.com/en/latest/usermanuals/source/srsepc/source/2_epc_getstarted.html

There are 3 main points to be careful:

1. Home Subscriber Server (HSS) database (*user_db.csv*) must match the used USIM *.conf* file. Configuration of the *.conf* file is explained under the configuring UE part. Each UE should be added to the HSS respectively.

    - MCC: 208
    - MNC: 92
    - MSIN: 0000000001
    - Key: 8baf473f2f8fd09487cccbd7097c6862
    - OPC: e734f8734007d6c5ce7a0508809e7e9c

    It is required to populate the entries in the HSS database, according to the number of UEs desired to be configured.

2. It is required to update the IP addresses of the mme and spgw. They can be assigned as they are initiated. For instance, if the VM has the IP adress of 129.97.228.188, simply run the following:

$ sudo srsepc –mme.mme_bind_addr 129.97.228.188 –spgw.gtpu_bind_addr 129.97.228.188

3. Every-time the device running EPC is rebooted, it is required to perform IP masquerading, otherwise you may experience packet forwarding issues at the EPC-side. It is explained further in the https://docs.srsran.com/en/latest/usermanuals/source/srsepc/source/2_epc_getstarted.html

4.2.2 Configuring the eNB and the UEs

In the official documentations, it is recommended to use a separate computer for eNB, UEs and the EPC to avoid synchronization issues. However, in this guide everything is run on the same machine.

Before starting, make sure the computer is connected to the internet via Local Area Network (LAN), **cable connection**. Wireless connection may not work with OAI setup.

Firstly, download the OAI source codes for eNB and UE.

```
$ git clone https://gitlab.eurecom.fr/oai/openairinterface5g/
   enb_folder
$ cd enb_folder
$ git checkout -f develop
$ cd ..
$ cp -Rf enb_folder ue_folder
```

Then populate USIM information inside the ue_folder. Ideally speaking, it is possible to emulate up to 256 UEs. However, the maximum number of UEs are bounded by the CPU power. In the default mode, the emulator can populate up to 4 UEs. In order to enable populating up to 256 UEs, it is required to change *platform_constants.h* script both in enb and ue folders.

```
# Edit enb_folder/openair2/COMMON/platform_constants.h
# Edit ue_folder/openair2/COMMON/platform_constants.h
```

Append the following definitions in *platform_constants.h* scripts both for end and ue folders.

```
...
#include "LTE_asn_constant.h"
#include "NR_asn_constant.h"
#define NR_MAXDRB 14
#define UE_EXPANSION <- Add this
#define LARGE_SCALE <- Add this
...
```

It is important to populate UE instances with unique MSIN values in the configuration files. Another important point is that SIM information must match the HSS database of the EPC for each UE.

```
$ cd ue_folder
# Edit openair3/NAS/TOOLS/ue_eurecom_test_sfr.conf
```

```
UE0:
{
    USER: {
        IMEI="356113022094149";
        MANUFACTURER="EURECOM";
        MODEL="LTE Android PC";
        PIN="0000";
    };

    SIM: {
        MSIN="0000000001";
        USIM_API_K="8baf473f2f8fd09487cccbd7097c6862";
        OPC="e734f8734007d6c5ce7a0508809e7e9c";
        MSISDN="33611123456";
    };
...
};
UE1:  <- You can append new UEs like this
{
...
    SIM: {
        MSIN="0000000002"; <- Modify Here
        USIM_API_K="8baf473f2f8fd09487cccbd7097c6862";
        OPC="e734f8734007d6c5ce7a0508809e7e9c";
        MSISDN="33611123456";
    };
...
};
...
```

As the next step, IP addresses should be configured. ”lo” interface (internal loopback of the OS) is used for the communication between eNB and UEs. In order to arrange correct IPs for the UEs do the followings:

```
$ cd ue_folder
# Edit ci-scripts/conf_files/ue.nfapi.conf
```

```
L1s = (
        {
        num_cc = 1;
        tr_n_preference = "nfapi";
        local_n_if_name = "lo"; <- Modify Here
        remote_n_address = "127.0.0.2"; <-
        local_n_address = "127.0.0.1"; <-
        local_n_portc     = 50000;
        remote_n_portc    = 50001;
        local_n_portd     = 50010;
        remote_n_portd    = 50011;
        }
);
```

In order to configure the conf file for the eNB:

```
$ cd enb_folder
# Edit ci-scripts/conf_files/rcc.band7.tm1.nfapi.conf
```

- IP adresses of the UEs and the eNBs

```
MACRLCs = (
        {
        num_cc = 1;
        local_s_if_name = "lo:"; <- Modify Here
        remote_s_address = "127.0.0.1"; <-
        local_s_address = "127.0.0.2"; <-
        local_s_portc     = 50001;
        remote_s_portc    = 50000;
        local_s_portd     = 50011;
        remote_s_portd    = 50010;
        tr_s_preference = "nfapi";
        tr_n_preference = "local_RRC";
        }
);
```

- **mme_ip_address** is the IP address of the EPC which is under MME parameters. In this explanation, the IP is 129.97.228.188.

- Under NETWORK_INTERFACES, it is required to change the interface that connects eNB to the EPC. Use Bridged Adapter for the Virtual Machine where the EPC runs. In order to find the respective interface name and the IP address, you can run *ifconfig* in the terminal of the host and the virtual machines. In this guide, the interface name is eno1 and the IP adress is 129.97.228.182 for the LAN connection of the host machine whereas the IP adress is 129.97.228.188 for the virtual machine.

```
$ ifconfig
eno1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 129.97.228.182  netmask 255.255.254.0  broadcast
   129.97.229.255
        ...
```

```
////////// MME parameters:
    mme_ip_address      = ( {
                              ipv4 = "129.97.228.188"; <- Modify Here
                              ipv6       = "192:168:30::17";
                              active     = "yes";
                              preference = "ipv4";
                            }
                          );

    NETWORK_INTERFACES :
    {
        ENB_INTERFACE_NAME_FOR_S1_MME            = "eno1"; <- Modify
        ENB_IPV4_ADDRESS_FOR_S1_MME              = "129.97.228.182"; <-
        ENB_INTERFACE_NAME_FOR_S1U               = "eno1"; <-
        ENB_IPV4_ADDRESS_FOR_S1U                 = "129.97.228.182"; <-
        ENB_PORT_FOR_S1U                         = 2152; # Spec 2152
        ENB_IPV4_ADDRESS_FOR_X2C                 = "129.97.228.182"; <-
        ENB_PORT_FOR_X2C                         = 36422; # Spec
   36422

    };
```

If flexRAN app is required to be run, add the followings to the end of the file (*rcc.band7.tm1.nfapi.conf*)

```
NETWORK_CONTROLLER : <- Add the entire section
{
    FLEXRAN_ENABLED = "yes"; <- Modify Here
    FLEXRAN_INTERFACE_NAME = "lo"; <-
    FLEXRAN_IPV4_ADDRESS = "127.0.0.3"; <-
    FLEXRAN_PORT             = 2210;
    FLEXRAN_CACHE            = "/mnt/oai_agent_cache";
    FLEXRAN_AWAIT_RECONF     = "no";
};
```

Then you can build your eNB and UEs.

To build the eNB:

```
$ cd enb_folder/cmake_targets/
$ ./build_oai -I --eNB
```

To build the UE:

```
$ cd ue_folder/cmake_targets/
$ ./build_oai -I --UE
```

Once the UEs are compiled, you need to copy UE binaries (ue.nvram, .ue_emm.nvram, .usim.nvram binaries for each UE) from $ue\_folder/targets/bin$ file to $ue\_folder/cmake\_targets$ file (they might be hidden files).

Once these steps are completed, eNB and UE emulator should be ready to be run.

4.2.3 Configuring the flexRAN controller

Source code of flexRAN controller can be found using the following link: https://gitlab.eurecom.fr/flexran/flexran-rtc flexRAN uses Pistache libraries for its REST framework. By the time of flexRAN being coded, the Pistache library was written in C11. However, the current Pistache requires C17. During the compilation of flexRAN, this causes problems since the authors hard-coded to checkout to "develop" branch of Pistache in the installation script. However, the current "develop" branch is quite different than the "develop" branch used during the implementation of flexRAN. Therefore, it is required to change the branch of the Pistache to an old commit like "efe54d9e53a5e257da03d27ce6a644643f31cb1d" to properly compile the flexRAN controller. In order to do that:

```
$ cd flexran-rtc-master/tools
# Edit install_dependencies
```

```
function install_pistache {
    echo "Installing pistache"
    git clone https://github.com/oktal/pistache.git
    cd pistache || exit
    git checkout -f efe54d9e53a5e257da03d27ce6a644643f31cb1d <- Modify Here
    mkdir build
    cd build
    cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Release ..
    make
    sudo make install
    cd -
}
```

Once the branch of Pistache is edited, you can return to flexran-rtc-master folder and run the *build_flexran_rtc.sh* script. Then flexRAN controller should be ready to be run.

4.2.4 Running the EPC, the flexRAN controller, the eNB and the UEs in the emulator mode

1. Firstly, make sure the low-latency kernel is running and the CPUs are scaled properly as described in the Kernel configuration.

2. Then, start the EPC in the virtual machine.

3. Then, start flexRAN controller.

4. Finally, run eNB & UE as described below.

Connect to the virtual machine via ssh and start the EPC (assuming the IP of the VM is 129.97.228.188: Start EPC:

```
$ sudo srsepc --mme.mme_bind_addr 129.97.228.188 --spgw.
   gtpu_bind_addr 129.97.228.188
```

Start flexRAN:

```
$ cd flexran-rtc-master/tools
$ sudo ./run_flexran_rtc.sh
```

Start eNB:

```
$ cd enb_folder/cmake_targets
$ sudo -E ./ran_build/build/lte-softmodem -O ../ci-scripts/
   conf_files/rcc.band7.tm1.nfapi.conf
```

Start UEs (You may need to change $--num-ues$ flag depending on the number of UE required to be run):

```
$ cd ue_folder/cmake_targets
$ sudo -E ./ran_build/build/lte-uesoftmodem -O ../ci-scripts/
   conf_files/ue.nfapi.conf --L2-emul 3 --num-ues 4 --nums_ue_thread
    1 --nokrnmod 1
```

If everything runs correctly, UEs should establish RRC connections and the UE interfaces should show up once you type $ifconfig$ in the terminal. The names should be oaitun_ue1, oaitun_ue2 ... oaitun_ue4. After that, some traffic can be generated between the UEs and the EPC using $iperf$ application. It is possible to create simultaneous traffic. To create a DL traffic between the EPC and the UE1:

At a terminal of the host that runs UEs

```
$ iperf -B 10.0.1.2 -u -s -i 1 -fm -p 5002
```

At a terminal of the virtual machine that runs EPC:

```
$ iperf -c 10.0.1.2 -u -t 30 -b 3M -i 1 -fm -B 10.0.1.1 -p 5002
```

As an example, the following bash script is given to create traffic simultaneously by 4 UEs. Name, password and the IP of the VM instance may vary as well as the number of UEs in the setup. In this tutorial, the VM instance name is epc, the password is **** and the IP adress is 129.97.228.188.

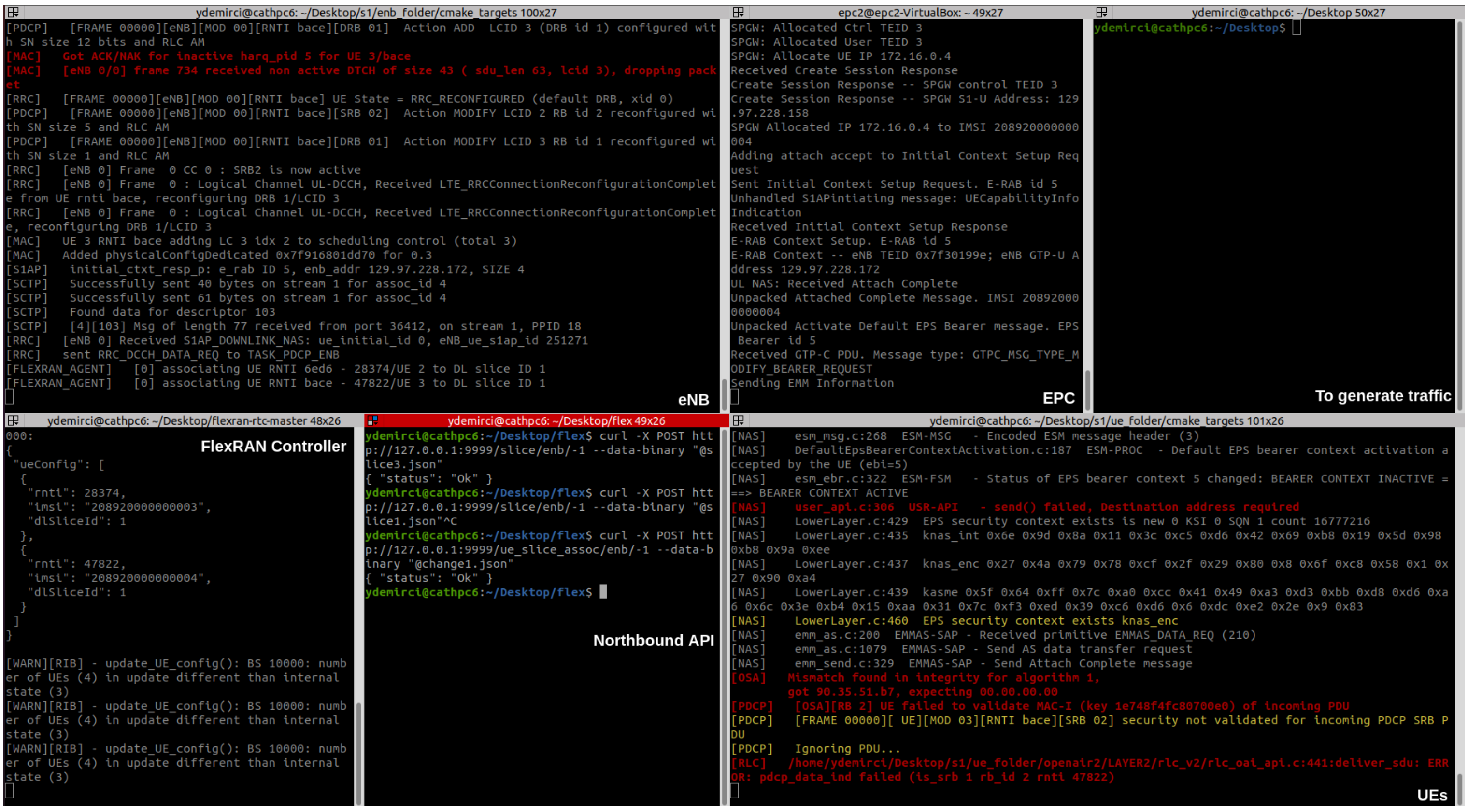
```
ip1=$(/sbin/ip -o -4 addr list oaitun_ue1 | awk '{print $4}' | cut -
    d/ -f1)
ip2=$(/sbin/ip -o -4 addr list oaitun_ue2 | awk '{print $4}' | cut -
    d/ -f1)
ip3=$(/sbin/ip -o -4 addr list oaitun_ue3 | awk '{print $4}' | cut -
    d/ -f1)
ip4=$(/sbin/ip -o -4 addr list oaitun_ue4 | awk '{print $4}' | cut -
    d/ -f1)

iperf -B $ip1 -u -s -i 1 -fm -p 5002 &
iperf -B $ip2 -u -s -i 1 -fm -p 5003 &
iperf -B $ip3 -u -s -i 1 -fm -p 5004 &
iperf -B $ip4 -u -s -i 1 -fm -p 5005 &


sshpass -p **** ssh epc@129.97.228.188 iperf -c $ip1 -u -t 10 -b 9M
    -i 1 -fm -p 5002 &
sshpass -p **** ssh epc@129.97.228.188 iperf -c $ip2 -u -t 10 -b 9M
    -i 1 -fm -p 5003 &
sshpass -p **** ssh epc@129.97.228.188 iperf -c $ip3 -u -t 10 -b 9M
    -i 1 -fm -p 5004 &
sshpass -p **** ssh epc@129.97.228.188 iperf -c $ip4 -u -t 10 -b 9M
    -i 1 -fm -p 5005 &
```

Using the Northbound API of flexRAN, it is possible to create and configure slices on the fly. It is possible to change slice and UE scheduling algorithms, the UE-sliec associations and many other features. More information about the north-bound API can be found here: https://mosaic5g.io/apidocs/flexran/

Once all the steps are successfully completed, the terminal should look similar to the Figure 4.3


ydemirci@cathpc6: ~/Desktop/s1/enb_folder/cmake_targets 100x27
eNB
epc2@epc2-VirtualBox: ~ 49x27
EPC
ydemirci@cathpc6: ~/Desktop 50x27
To generate traffic
ydemirci@cathpc6: ~/Desktop/flexran-rtc-master 48x26
FlexRAN Controller
ydemirci@cathpc6: ~/Desktop/flex 49x26
Northbound API
ydemirci@cathpc6: ~/Desktop/s1/ue_folder/cmake_targets 101x26
UEs

```
[PDCP]   [FRAME 00000][eNB][MOD 00][RNTI bace][DRB 01]  Action ADD  LCID 3 (DRB id 1) configured wit
h SN size 12 bits and RLC AM
[MAC]   Got ACK/NAK for inactive harq_pid 5 for UE 3/bace
[MAC]   [eNB 0/0] frame 734 received non active DTCH of size 43 ( sdu_len 63, lcid 3), dropping pack
et
[RRC]   [FRAME 00000][eNB][MOD 00][RNTI bace] UE State = RRC_RECONFIGURED (default DRB, xid 0)
[PDCP]   [FRAME 00000][eNB][MOD 00][RNTI bace][SRB 02]  Action MODIFY LCID 2 RB id 2 reconfigured wi
th SN size 5 and RLC AM
[PDCP]   [FRAME 00000][eNB][MOD 00][RNTI bace][DRB 01]  Action MODIFY LCID 3 RB id 1 reconfigured wi
th SN size 1 and RLC AM
[RRC]   [eNB 0] Frame  0 CC 0 : SRB2 is now active
[RRC]   [eNB 0] Frame  0 : Logical Channel UL-DCCH, Received LTE_RRCConnectionReconfigurationComplet
e from UE rnti bace, reconfiguring DRB 1/LCID 3
[RRC]   [eNB 0] Frame  0 : Logical Channel UL-DCCH, Received LTE_RRCConnectionReconfigurationComplet
e, reconfiguring DRB 1/LCID 3
[MAC]   UE 3 RNTI bace adding LC 3 idx 2 to scheduling control (total 3)
[MAC]   Added physicalConfigDedicated 0x7f916801dd70 for 0.3
[S1AP]   initial_ctxt_resp_p: e_rab ID 5, enb_addr 129.97.228.172, SIZE 4
[SCTP]   Successfully sent 40 bytes on stream 1 for assoc_id 4
[SCTP]   Successfully sent 61 bytes on stream 1 for assoc_id 4
[SCTP]   Found data for descriptor 103
[SCTP]   [4][103] Msg of length 77 received from port 36412, on stream 1, PPID 18
[RRC]   [eNB 0] Received S1AP_DOWNLINK_NAS: ue_initial_id 0, eNB_ue_s1ap_id 251271
[RRC]   sent RRC_DCCH_DATA_REQ to TASK_PDCP_ENB
[FLEXRAN_AGENT]   [0] associating UE RNTI 6ed6 - 28374/UE 2 to DL slice ID 1
[FLEXRAN_AGENT]   [0] associating UE RNTI bace - 47822/UE 3 to DL slice ID 1
```

```
SPGW: Allocated Ctrl TEID 3
SPGW: Allocated User TEID 3
SPGW: Allocate UE IP 172.16.0.4
Received Create Session Response
Create Session Response -- SPGW control TEID 3
Create Session Response -- SPGW S1-U Address: 129
.97.228.158
SPGW Allocated IP 172.16.0.4 to IMSI 208920000000
004
Adding attach accept to Initial Context Setup Req
uest
Sent Initial Context Setup Request. E-RAB id 5
Unhandled S1APintiating message: UECapabilityInfo
Indication
Received Initial Context Setup Response
E-RAB Context Setup. E-RAB id 5
E-RAB Context -- eNB TEID 0x7f30199e; eNB GTP-U A
ddress 129.97.228.172
UL NAS: Received Attach Complete
Unpacked Attached Complete Message. IMSI 20892000
0000004
Unpacked Activate Default EPS Bearer message. EPS
 Bearer id 5
Received GTP-C PDU. Message type: GTPC_MSG_TYPE_M
ODIFY_BEARER_REQUEST
Sending EMM Information
```

```
ydemirci@cathpc6:~/Desktop$
```

```
000:
{
 "ueConfig": [
  {
   "rnti": 28374,
   "imsi": "208920000000003",
   "dlSliceId": 1
  },
  {
   "rnti": 47822,
   "imsi": "208920000000004",
   "dlSliceId": 1
  }
 ]
}

[WARN][RIB] - update_UE_config(): BS 10000: numb
er of UEs (4) in update different than internal
state (3)
[WARN][RIB] - update_UE_config(): BS 10000: numb
er of UEs (4) in update different than internal
state (3)
[WARN][RIB] - update_UE_config(): BS 10000: numb
er of UEs (4) in update different than internal
state (3)
```

```
ydemirci@cathpc6:~/Desktop/flex$ curl -X POST htt
p://127.0.0.1:9999/slice/enb/-1 --data-binary "@s
lice3.json"
{ "status": "Ok" }
ydemirci@cathpc6:~/Desktop/flex$ curl -X POST htt
p://127.0.0.1:9999/slice/enb/-1 --data-binary "@s
lice1.json"^C
ydemirci@cathpc6:~/Desktop/flex$ curl -X POST htt
p://127.0.0.1:9999/ue_slice_assoc/enb/-1 --data-b
inary "@change1.json"
{ "status": "Ok" }
ydemirci@cathpc6:~/Desktop/flex$
```

```
[NAS]    esm_msg.c:268  ESM-MSG   - Encoded ESM message header (3)
[NAS]    DefaultEpsBearerContextActivation.c:187  ESM-PROC  - Default EPS bearer context activation a
ccepted by the UE (ebi=5)
[NAS]    esm_ebr.c:322  ESM-FSM   - Status of EPS bearer context 5 changed: BEARER CONTEXT INACTIVE =
==> BEARER CONTEXT ACTIVE
[NAS]    user_api.c:306  USR-API   - send() failed, Destination address required
[NAS]    LowerLayer.c:429  EPS security context exists is new 0 KSI 0 SQN 1 count 16777216
[NAS]    LowerLayer.c:435  knas_int 0x6e 0x9d 0x8a 0x11 0x3c 0xc5 0xd6 0x42 0x69 0xb8 0x19 0x5d 0x98
0xb8 0x9a 0xee
[NAS]    LowerLayer.c:437  knas_enc 0x27 0x4a 0x79 0x78 0xcf 0x2f 0x29 0x80 0x8 0x6f 0xc8 0x58 0x1 0x
27 0x90 0xa4
[NAS]    LowerLayer.c:439  kasme 0x5f 0x64 0xff 0x7c 0xa0 0xcc 0x41 0x49 0xa3 0xd3 0xbb 0xd8 0xd6 0xa
6 0x6c 0x3e 0xb4 0x15 0xaa 0x31 0x7c 0xf3 0xed 0x39 0xc6 0xd6 0x6 0xdc 0xe2 0x2e 0x9 0x83
[NAS]    LowerLayer.c:460  EPS security context exists knas_enc
[NAS]    emm_as.c:200  EMMAS-SAP - Received primitive EMMAS_DATA_REQ (210)
[NAS]    emm_as.c:1079  EMMAS-SAP - Send AS data transfer request
[NAS]    emm_send.c:329  EMMAS-SAP - Send Attach Complete message
[OSA]   Mismatch found in integrity for algorithm 1,
        got 90.35.51.b7, expecting 00.00.00.00
[PDCP]   [OSA][RB 2] UE failed to validate MAC-I (key 1e748f4fc80700e0) of incoming PDU
[PDCP]   [FRAME 00000][ UE][MOD 03][RNTI bace][SRB 02] security not validated for incoming PDCP SRB P
DU
[PDCP]   Ignoring PDU...
[RLC]   /home/ydemirci/Desktop/s1/ue_folder/openair2/LAYER2/rlc_v2/rlc_oai_api.c:441:deliver_sdu: ERR
OR: pdcp_data_ind failed (is_srb 1 rb_id 2 rnti 47822)
```

Figure 4.3: How the terminals look like after running all the components successfully

For further information about nFAPI emulation, the following web-page can be referred: https://gitlab.eurecom.fr/oai/openairinterface5g/-/blob/master/doc/L2NFAPI_S1.md However, the setup given there is quite different than the one described in this paper. As long as the IP addresses are set correctly, it is possible to run EPC, UE and eNB-flexRAN on separate machines.

4.3 OAI with bladeRF and COTS UEs

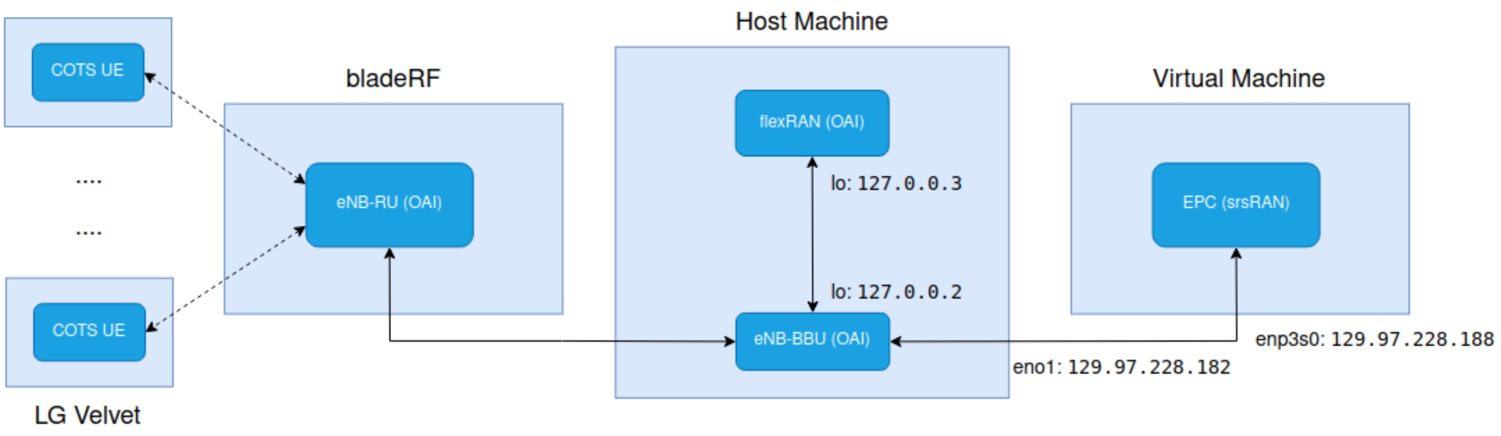


Figure 4.4: Interface names and IP addresses of the configured components

Configuring the eNB with a COTS hardware and connecting COTS UEs (e.g smartphones) is more challenging than configuring the emulation mode. Normally, EUROCOM works with Ettus USRP B210 device and its configuration scripts are up-to-date. However due to the long shipment time of USRP B210, we decided to use with bladeRF 2.0 micro xA4 device. The problem with the bladeRF is that the configuration guide provided by EUROCOM is not up-to-dated and there is no any other configuration tutorial out there to the best of my knowledge. As the OAI platform evolved in time, the tutorials have not been updated, therefore there are several details involved to configure the hardware setup.

4.3.1 Configuring the bladeRF device

A version of Ubuntu 20 is required for the bladeRF application. Once a version Ubuntu 20 is up and running, the dependencies of bladeRF application can be installed:

```
$ sudo apt-get install libusb-1.0-0-dev libusb-1.0-0 build-essential
    cmake libncurses5-dev libtecla1 libtecla-dev pkg-config git wget
```

Then install the bladeRF application:

```
$ sudo add-apt-repository ppa:nuandllc/bladerf
$ sudo apt-get update
$ sudo apt-get install bladerf
$ sudo apt-get install bladerf-fpga-hostedxa4
```

It is required to update FX3 firmware and fpga images. The latest versions can be downloaded from

- http://www.nuand.com/fx3_images/
- http://www.nuand.com/fpga_images/

Then install the images. (Depending on the root permissions, you may not be able to upload (write) the images. In this case you need to give write permission to the USB port that you are using.)

```
$ sudo bladeRF-cli -f bladeRF_fw_latest.img
$ sudo bladeRF-cli -L hostedx40-latest.rbf
```

After running these steps, the bladrf application should be ready. Plug your bladeRF device to the computer and make the calibrations by following the instructions given in `https://www.nuand.com/calibration`, you can see your device information by:

```
$ sudo bladeRF-cli -i
info
```

After completing these steps, bladeRF device should be ready to be used. Even though the guide (README) provided by OAI is out of date, it can be found in $enb\_folder/targets/ARCH/BLADERF$.

4.3.2 Configuring the EPC

Configuring the srsEPC for the non-emulation mode is slightly different than the one described in Configuring the EPC because of a bug in the OAI eNB. Due to the bug, it cannot be installe from the ppa repository directly but it is required to be build from the source code.

Download the source codes of srsRAN then edit the *nas.cc* script.

```
$ git clone https://github.com/srsran/srsRAN.git srs
# Edit srs/srsepc/src/mme/nas.cc
```

It is required to comment out the codes within the `if (m_emm_ctx.state...)` statement. The statement is around line 1070 but it may vary in different git branches. The problem is: The OAI eNB crashes if the EPC sends EMM information while the UE is in the deregistered state. In the OAI open-forum, the authors solved this issue by disabling to send *downlink_nas_transport* message in the OAI EPC. We follow the same approach with srsEPC. Disabling this message does not interfere with the other stages of connection between the UE and the eNB.

```
if (act_bearer.eps_bearer_id < 5 || act_bearer.eps_bearer_id > 15) {
    m_logger.error("EPS Bearer ID out of range");
    return false;
  }
  /*  <- Comment out from here
  if (m_emm_ctx.state == EMM_STATE_DEREGISTERED) {
    // Attach requested from attach request
    m_gtpc->send_modify_bearer_request(
        m_emm_ctx.imsi, act_bearer.eps_bearer_id, &m_esm_ctx[
   act_bearer.eps_bearer_id].enb_fteid);

    // Send reply to EMM Info to UE
    srsran::unique_byte_buffer_t nas_tx = srsran::make_byte_buffer()
   ;
    if (nas_tx == nullptr) {
      m_logger.error("Couldn't allocate PDU in %s().", __FUNCTION__)
   ;
      return false;
    }
    pack_emm_information(nas_tx.get());

    m_s1ap->send_downlink_nas_transport(
        m_ecm_ctx.enb_ue_s1ap_id, m_ecm_ctx.mme_ue_s1ap_id, nas_tx.
   get(), m_ecm_ctx.enb_sri);

    srsran::console("Sending EMM Information\n");
    m_logger.info("Sending EMM Information");
  }
  */ <- To here
  m_emm_ctx.state = EMM_STATE_REGISTERED;
  return true;
```

Install the srsran dependencies:

```
$ sudo apt-get install build-essential cmake libfftw3-dev libmbedtls
   -dev libboost-program-options-dev libconfig++-dev libsctp-dev
```

After editing the *nas.cc* and installing dependencies, install the srsEPC using CMake:

```
$ cd srs
$ mkdir build
$ cd build
$ cmake ../
$ make
$ make srsepc

$ sudo make install
$ sudo srsran_install_configs.sh use
```

After this, the srsEPC should be installed. You can configure the HSS database by adding the USIM information as it is described in Configuring the EPC. Then the srsEPC is ready to be run.

4.3.3 Configuring the COTS UEs and the SIM cards

It is required to configure the SIM cards according to the USIM entries of HSS database. The SIM card reader & writer application is provided by the device vendor. Windows OS is required to run the application.

Once the SmartCard reader device is plugged to the computer, use the following third party software to configure SIM cards. I uploaded the application to my personal github.

https://github.com/YektaDemirci/OYSIMwrite

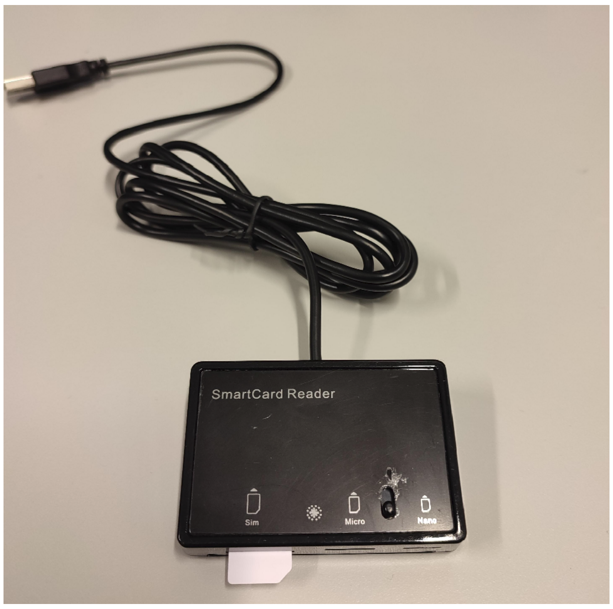
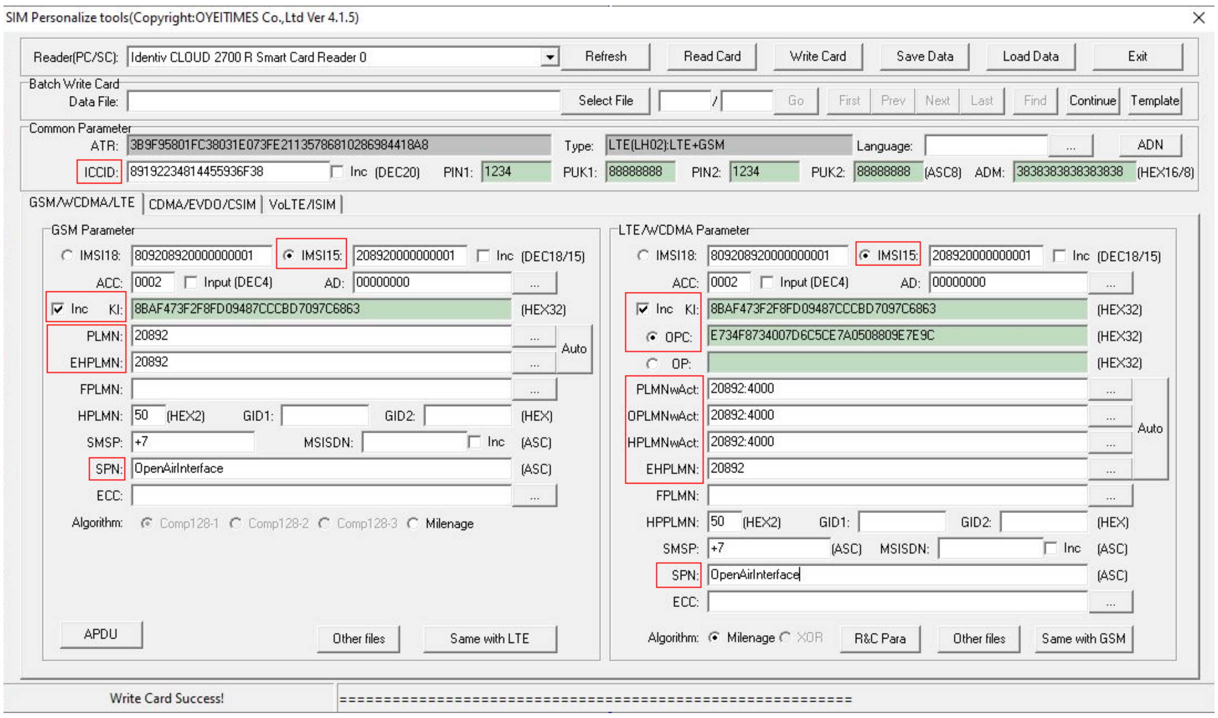


Figure 4.5: SIM card reader & writer device

Likewise the emulator mode, we can use the same USIM information for the COTS UEs. Fill the variables highlighted with the red-boxes in figure 4.6. ICCID and IMSI information must be unique for each UE whereas KI, OPC and PLMN information can be the same. Once SIM cards are successfully written, “Write Card Success!” message can be seen at the left bottom part of the app.



SIM Personalize tools(Copyright:OYEITIMES Co.,Ltd Ver 4.1.5)

Reader(PC/SC): Identiv CLOUD 2700 R Smart Card Reader 0

Batch Write Card
Data File:

Common Parameter
ATR: 3B9F95801FC38031E073FE211357868102869844 18A8
Type: LTE(LH02):LTE+GSM
Language:
ICCID: 89192234814455936F38 ☐ Inc (DEC20) PIN1: 1234 PUK1: 88888888 PIN2: 1234 PUK2: 88888888 (ASC8) ADM: 3838383838383838 (HEX16/8)

GSM/WCDMA/LTE | CDMA/EVDO/CSIM | VoLTE/ISIM

GSM Parameter
IMSI18: 809208920000000001 [✓ IMSI15:] 208920000000001 ☐ Inc (DEC18/15)
ACC: 0002 ☐ Input (DEC4) AD: 00000000
☑ Inc KI: 8BAF473F2F8FD09487CCCBD7097C6863 (HEX32)
PLMN: 20892
EHPLMN: 20892
FPLMN:
HPLMN: 50 (HEX2) GID1: GID2: (HEX)
SMSP: +7 MSISDN: ☐ Inc (ASC)
SPN: OpenAirInterface (ASC)
ECC:
Algorithm: [✓ Comp128-1] Comp128-2 Comp128-3 Milenage

LTE/WCDMA Parameter
IMSI18: 809208920000000001 [✓ IMSI15:] 208920000000001 ☐ Inc (DEC18/15)
ACC: 0002 ☐ Input (DEC4) AD: 00000000
☑ Inc KI: 8BAF473F2F8FD09487CCCBD7097C6863 (HEX32)
[✓ OPC:] E734F8734007D6C5CE7A0508809E7E9C (HEX32)
OP: (HEX32)
PLMNwAct: 20892:4000
OPLMNwAct: 20892:4000
HPLMNwAct: 20892:4000
EHPLMN: 20892
FPLMN:
HPPLMN: 50 (HEX2) GID1: GID2: (HEX)
SMSP: +7 (ASC) MSISDN: ☐ Inc (ASC)
SPN: OpenAirInterface (ASC)
ECC:
Algorithm: [✓ Milenage] XOR

Write Card Success! ==

Figure 4.6: SIM card reader & writer application

After configuring the SIM cards, it is required to configure the COTS UEs. We are going to use LTE RAT in band-7. We need to update these configurations manually to make UE discover the eNB. In this work LG Velvet, LM-G900UM2 COTS UEs are used.

Depending on the UE model, the "HiddenMenu" code may vary. In order to access LG Velvet " HiddenMenu" use the following code:

#∗462633∗#900#

Then navigate to RAT selection and choose "LTE only":

```
Field Test−>Modem Settings−>RAT Selection−>LTE only
```

After choosing "LTE only", update the band. You should navigate to LTE BAND, enable Band7 then scrool down and click to "Save" button.

```
Field Test−>Modem Settings−>Band Selection−>LTE BAND−>#Enable Band7
```

Now exit the "HiddenMenu". Once eNB starts running, the phone should be able to detect the MIB signals. Yet, to establish an RRC connection and to access to the internet, it is required to set-up an Access Point Name (APN). The default APN name is “srsapn” in srsEPC . The APN name in the COTS phone must match the APN name at the EPC. Navigate to APN settings on the UE:

```
Settings->Network->Mobile networks->Access Point Names->Add APN
```

Set the following variables and “Save”

- Name: oai
- APN: srsapn
- MCC: 208
- MNC: 92
- APN typle: default,mms,supl

After setting-up the APN, the phone should be able to connect to the internet. Now the COTS UEs are ready to successfully establish a connection with the eNB.

4.3.4 Configuring the eNB

Download the eNB source codes and switch to the develop branch.

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

The script to build eNB with bladeRF was written for the bladeRF 1.0 devices. However, in this work we use bladeRF 2.0 which does not support DC calibrating. Therefore we need to make a small edit on the bladeRF helper source codes, otherwise it throws an error during the eNB compilation. It is required to turn-off the DC calibration. Navigate to *bladerf_lib.c* in the enb_folder and comment out the if condition which is right after *calibrate* line. It is around line 1128 but it may vary depending on the branch.

```
$ cd enb_folder
# Edit /targets/ARCH/BLADERF/USERSPACE/LIB/bladerf_lib.c
```

```
     ...
    if ((status=bladerf_enable_module(brf->dev, BLADERF_MODULE_RX,
    true)) != 0) {
         fprintf(stderr,"Failed to enable RX module: %s\n",
    bladerf_strerror(status));
```

```
        brf_error(status);
    } else
        printf("[BRF] RX module enabled \n");

    // calibrate

    /*  <- Comment out from here
    if ((status=bladerf_calibrate_dc(brf->dev, BLADERF_DC_CAL_LPF_TUNING)) != 0 ||
        ...
        ...
        ){
        fprintf(stderr, "[BRF] error calibrating\n");
        brf_error(status);
    } else
        printf("[BRF] calibration OK\n");
    */ <- To here

    bladerf_log_set_verbosity(get_brf_log_level(openair0_cfg->
   log_level));
    ...
```

Build eNB with the following commands. Unlike the emulator mode, it is required to compile with the "-w BLADERF" flag.

```
$ cd enb_folder/cmake_targets
$ ./build_oai  --eNB -w BLADERF
```

The conf file for the bladeRF (enb-band7-5mhz.conf) is located at the following location. Move this conf file to the conf_files folder.

```
$ cd enb_folder/configuration/bladeRF
# Move enb-band7-5mhz.conf to enb_folder/ci-scripts/conf_files
```

Edit the following variables in the enb-band7-5mhz.conf file:

- plmn_list
- mme_ip_address
- NETWORK_INTERFACES
- THREAD_STRUCT
- RUs

```
...
plmn_list = (
        mcc = 208; mnc = 92; mnc_length = 2;
```

```
    );
...
..
////////// MME parameters:
    mme_ip_address = ( { ipv4 = "129.97.228.188";
                              ipv6       = "192:168:30::17";
                              active     = "yes";
                              preference = "ipv4";
                            }
                          );
NETWORK_INTERFACES :
    {
        ENB_INTERFACE_NAME_FOR_S1_MME = "eno1";
        ENB_IPV4_ADDRESS_FOR_S1_MME = "129.97.228.182";

        ENB_INTERFACE_NAME_FOR_S1U = "eno1";
        ENB_IPV4_ADDRESS_FOR_S1U = "129.97.228.182";
        ENB_PORT_FOR_S1U                         = 2152; # Spec 2152

        ENB_IPV4_ADDRESS_FOR_X2C = "129.97.228.182";
        ENB_PORT_FOR_X2C                         = 36422; # Spec
   36422
    };
...
...
THREAD_STRUCT = (
  {
    parallel_config = "PARALLEL_SINGLE_THREAD";
    worker_config = "WORKER_ENABLE";
  }
);
...
...
RUs = (
    {
       local_rf       = "yes";
         nb_tx           = 1;
         nb_rx           = 1;
         att_tx = 66;
         att_rx = 66;
         bands           = [7];
         max_pdschReferenceSignalPower = -28;
         max_rxgain = 117;
         eNB_instances   = [0];

    }
);
...
```

In order to have a proper performance it is required to adjust att_tx, att_rx, max_rxgain according to the setup. After completing all the steps, if the UE connects to the eNB but cannot connect to the internet it might be because of the badly configured gains.

4.3.5 Configuring the flexRAN controller

Configuring the flexRAN controller for the hardware setup is the same as the emulation mode. It is possible to follow the exact same steps as Section 4.2.3.

4.3.6 Running the flexRAN, the EPC, the eNB on bladeRF and connecting the COTS UEs

Once the following steps are done, it is possible to run the eNB with the bladeRF device and to connect the COTS UEs.

- Configuring the bladeRF device
- Configuring the srsEPC
- Configuring the SIM cards and the COTs UEs
- Configuring the eNB.
- Installing the flexRAN

"Terminator" application can be used to open multiple terminals and manage all the applications (bladeRF, eNB, flexRAN, srsEPC) from one window.

Firstly, turn on the airplane mode in the UEs. Then start the VM and run the EPC. It is required to enable packet forwarding at the EPC side. Otherwise, Linux will block the packets sent from the EPC and phones will not have an access to the internet. Assuming the VM has the IP adress of 129.97.228.188 and the interface name of *enp3s0*, first enable packet forwarding:

```
$ cd /srs/srsepc
$ sudo srsepc_if_masq.sh enp3s0
```

Then start the EPC:

```
$ sudo srsepc --mme.mme_bind_addr 129.97.228.188 --spgw.
   gtpu_bind_addr 129.97.228.188
```

Then, start flexRAN:

```
$ cd flexran-rtc-master/tools
$ sudo ./run_flexran_rtc.sh
```

Then start the eNB:

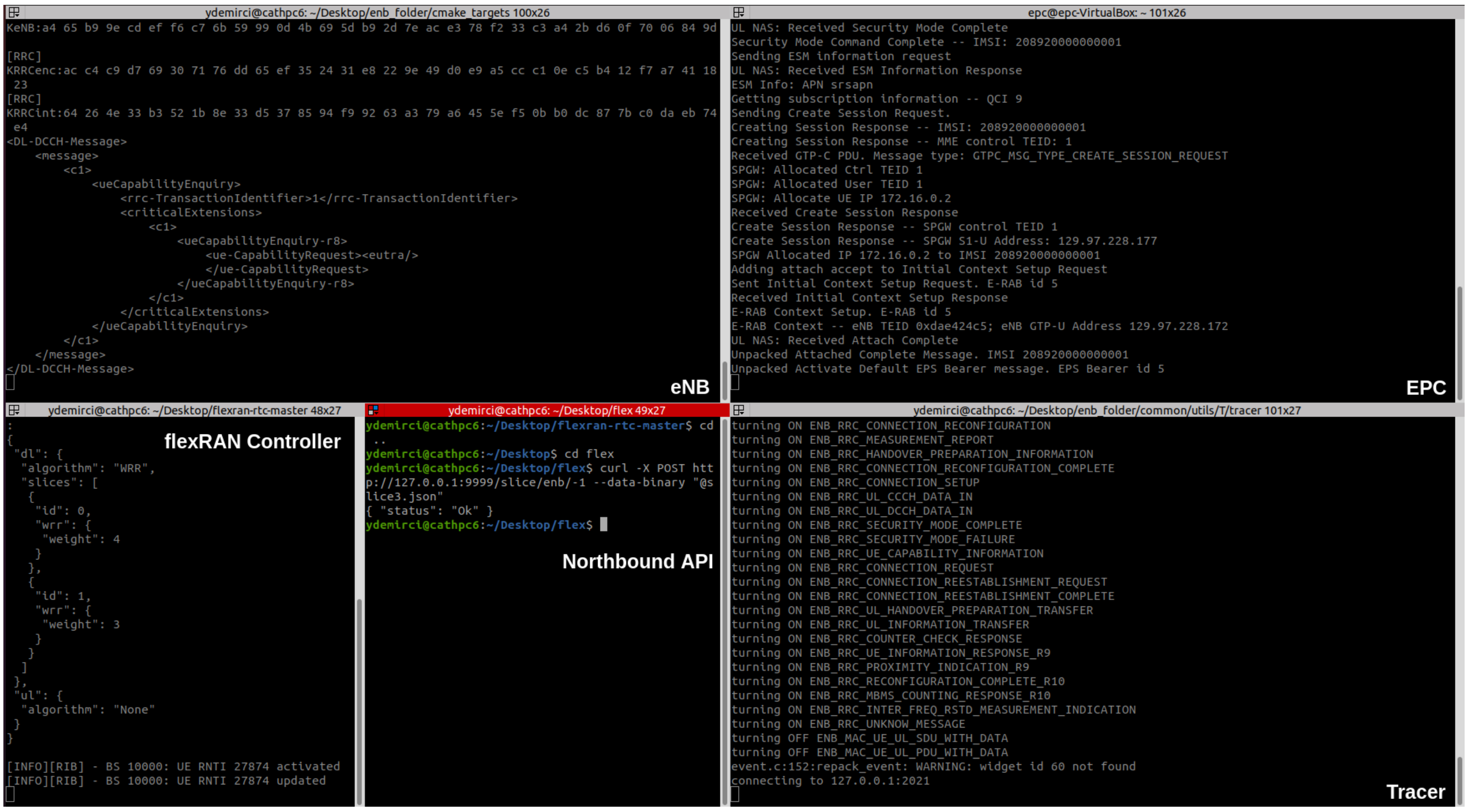
```
$ cd enb_folder/cmake_targets
$ sudo -E ./ran_build/build/lte-softmodem -O ../ci-scripts/
   conf_files/enb-band7-5mhz.conf --T_stdout 0
```

Then start the tracer:

```
$ cd enb_folder/common/utils/T/tracer
$ ./enb -d ../T_messages.txt
```

Once the eNB is up and running, turn off the airplane mode on your UEs. After that, the UE should automatically connect to the eNB. Once everything runs smoothly, the terminal should look like the Figure given in 4.7. 15Mbps can be achieved with a single UE if everything is set correctly. If the phone connects to the eNB but cannot access to the internet it might be because of the physical channel. Adjust att_tx, att_rx, max_rxgain variables according to your setup in the "enb-band7-5mhz.conf" file. Additionally, control if the packet forwarding is enabled at the EPC, using *srsepc_if_masq.sh*. It is possible to establish multiple UE attachments to the eNB as long as HSS is set-up accordingly at the EPC-side. However, we couldn't make multiple UEs work properly because of the following:

- The down-link channel was very poor. Even though I used several different gains (including the automatic gain controller provided by the bladeRF) and several different channels, the down-link was not consistent. Even if I could obtain a good DL channel for a single UE, the channel quality for other UEs were very poor. Furthermore, once I changed the UE location slightly, the connection between the UE and the eNB got lost and random hand-overs were triggered. In short, we couldn't make multiple phones work with COTS hardware.


ydemirci@cathpc6: ~/Desktop/enb_folder/cmake_targets 100x26
eNB
epc@epc-VirtualBox: ~ 101x26
EPC
ydemirci@cathpc6: ~/Desktop/flexran-rtc-master 48x27
flexRAN Controller
ydemirci@cathpc6: ~/Desktop/flex 49x27
Northbound API
ydemirci@cathpc6: ~/Desktop/enb_folder/common/utils/T/tracer 101x27
Tracer

```
KeNB:a4 65 b9 9e cd ef f6 c7 6b 59 99 0d 4b 69 5d b9 2d 7e ac e3 78 f2 33 c3 a4 2b d6 0f 70 06 84 9d

[RRC]
KRRCenc:ac c4 c9 d7 69 30 71 76 dd 65 ef 35 24 31 e8 22 9e 49 d0 e9 a5 cc c1 0e c5 b4 12 f7 a7 41 18
 23
[RRC]
KRRCint:64 26 4e 33 b3 52 1b 8e 33 d5 37 85 94 f9 92 63 a3 79 a6 45 5e f5 0b b0 dc 87 7b c0 da eb 74
 e4
<DL-DCCH-Message>
    <message>
        <c1>
            <ueCapabilityEnquiry>
                <rrc-TransactionIdentifier>1</rrc-TransactionIdentifier>
                <criticalExtensions>
                    <c1>
                        <ueCapabilityEnquiry-r8>
                            <ue-CapabilityRequest><eutra/>
                            </ue-CapabilityRequest>
                        </ueCapabilityEnquiry-r8>
                    </c1>
                </criticalExtensions>
            </ueCapabilityEnquiry>
        </c1>
    </message>
</DL-DCCH-Message>
```

```
UL NAS: Received Security Mode Complete
Security Mode Command Complete -- IMSI: 208920000000001
Sending ESM information request
UL NAS: Received ESM Information Response
ESM Info: APN srsapn
Getting subscription information -- QCI 9
Sending Create Session Request.
Creating Session Response -- IMSI: 208920000000001
Creating Session Response -- MME control TEID: 1
Received GTP-C PDU. Message type: GTPC_MSG_TYPE_CREATE_SESSION_REQUEST
SPGW: Allocated Ctrl TEID 1
SPGW: Allocated User TEID 1
SPGW: Allocate UE IP 172.16.0.2
Received Create Session Response
Create Session Response -- SPGW control TEID 1
Create Session Response -- SPGW S1-U Address: 129.97.228.177
SPGW Allocated IP 172.16.0.2 to IMSI 208920000000001
Adding attach accept to Initial Context Setup Request
Sent Initial Context Setup Request. E-RAB id 5
Received Initial Context Setup Response
E-RAB Context Setup. E-RAB id 5
E-RAB Context -- eNB TEID 0xdae424c5; eNB GTP-U Address 129.97.228.172
UL NAS: Received Attach Complete
Unpacked Attached Complete Message. IMSI 208920000000001
Unpacked Activate Default EPS Bearer message. EPS Bearer id 5
```

```
:
{
 "dl": {
  "algorithm": "WRR",
  "slices": [
   {
    "id": 0,
    "wrr": {
     "weight": 4
    }
   },
   {
    "id": 1,
    "wrr": {
     "weight": 3
    }
   }
  ]
 },
 "ul": {
  "algorithm": "None"
 }
}

[INFO][RIB] - BS 10000: UE RNTI 27874 activated
[INFO][RIB] - BS 10000: UE RNTI 27874 updated
```

```
ydemirci@cathpc6:~/Desktop/flexran-rtc-master$ cd
 ..
ydemirci@cathpc6:~/Desktop$ cd flex
ydemirci@cathpc6:~/Desktop/flex$ curl -X POST htt
p://127.0.0.1:9999/slice/enb/-1 --data-binary "@s
lice3.json"
{ "status": "Ok" }
ydemirci@cathpc6:~/Desktop/flex$
```

```
turning ON ENB_RRC_CONNECTION_RECONFIGURATION
turning ON ENB_RRC_MEASUREMENT_REPORT
turning ON ENB_RRC_HANDOVER_PREPARATION_INFORMATION
turning ON ENB_RRC_CONNECTION_RECONFIGURATION_COMPLETE
turning ON ENB_RRC_CONNECTION_SETUP
turning ON ENB_RRC_UL_CCCH_DATA_IN
turning ON ENB_RRC_UL_DCCH_DATA_IN
turning ON ENB_RRC_SECURITY_MODE_COMPLETE
turning ON ENB_RRC_SECURITY_MODE_FAILURE
turning ON ENB_RRC_UE_CAPABILITY_INFORMATION
turning ON ENB_RRC_CONNECTION_REQUEST
turning ON ENB_RRC_CONNECTION_REESTABLISHMENT_REQUEST
turning ON ENB_RRC_CONNECTION_REESTABLISHMENT_COMPLETE
turning ON ENB_RRC_UL_HANDOVER_PREPARATION_TRANSFER
turning ON ENB_RRC_UL_INFORMATION_TRANSFER
turning ON ENB_RRC_COUNTER_CHECK_RESPONSE
turning ON ENB_RRC_UE_INFORMATION_RESPONSE_R9
turning ON ENB_RRC_PROXIMITY_INDICATION_R9
turning ON ENB_RRC_RECONFIGURATION_COMPLETE_R10
turning ON ENB_RRC_MBMS_COUNTING_RESPONSE_R10
turning ON ENB_RRC_INTER_FREQ_RSTD_MEASUREMENT_INDICATION
turning ON ENB_RRC_UNKNOW_MESSAGE
turning OFF ENB_MAC_UE_UL_SDU_WITH_DATA
turning OFF ENB_MAC_UE_UL_PDU_WITH_DATA
event.c:152:repack_event: WARNING: widget id 60 not found
connecting to 127.0.0.1:2021
```

Figure 4.7: How the terminals look like after running all the components successfully and a COTS UE establishes a connection

Chapter 5

Implementation of the schedulers

5.1 The State of the Art

The OAI platform already provides the SOA. It is possible to create multiple slices and assign sub-channels exclusively to each slice. Then each slice uses a RR algorithm to allocate the PRBs to their corresponding UEs. The RR algorithm provided by OAI is quite similar to the one given in Algorithm 1. Power budget per PRB can either be configured in the initial *conf* file or it can be changed using flexRAN controller on the fly.

Considering a 5MHz channel where half of the spectrum is used for DL and the other half is used for UL, there would be 50PRBs per sub-frame with 15kHZ sub-carrier spacing. Considering 4G RAT, 3 PRBs will form a single Resource Block Group (RBG) hence there would be 17 RBGs at a given sub-frame. As an example, the following *"slice.json"* file can be used to create two static slices where the first slice would have the RBGs with the indices from 0 to 6 and the second slice would have RBGs with the indices from 9 to 15 for all sub-frames.

```
{
    "dl": {
        "algorithm": "Static",
        "slices": [
            {
                "id": 0,
                "static": {
                    "posLow": 0,
                    "posHigh": 6
                }
            },
            {
                "id": 1,
                "static": {
                    "posLow": 9,
                    "posHigh": 15
                }
            }
        ]
    }
}
```

DL slice configuration can be updated on the fly by sending this json file to northbound of flexRAN controller using the following curl command:

```
$ curl -X POST http://127.0.0.1:9999/slice/enb/-1 --data-binary "
   @slice3.json"
```

5.2 The proposed Two-Level Scheduler

The proposed two-level scheduler is not provided by the OAI platform therefore we implemented Algorithm 2 at the eNB side. Furthermore, to be able to use the proposed scheduler with the flexRAN controller, we made some additions to flexRAN source codes, as well.

The implemented functions can be found under the following link, however a team access is required to access them:

https://gitlab.com/oai-wrr/wrr_source_codes/-/tree/main

flexRAN and eNB communicates through Google protobuffs. Therefore the first step is to manipulate the *config_common.proto* scripts both at the eNB and flexRAN sides. Both proto scripts should be the same. I introduced a single variable called "weight" to denote the WRR weight of a slice.

```
#Edit enb_folder/openair2/ENB_APP/MESSAGES/V2/config_common.proto
#Edit flexran-rtc-master/src/MESSAGES/V2/config_common.proto
```

There is no further addition needed for the flexRAN-side. However, we need to manipulate a few more scripts at the eNB-side.

In order for eNB to deploy the commands administered by the flexRAN, we need to manipulate the *flexran_agent_ran_api.c* script and update the following four functions:

1. *flexran_get_dl_slice_algo*
2. *flexran_set_dl_slice_algo*

3. *flexran_create_dl_slice*
4. *flexran_get_dl_slice*

```
# Edit enb_folder/openair2/ENB_APP/flexran_agent_ran_api.c
```

Then, there are two more scripts to be manipulated. The first one is *slicing.h* where the variables used for Algorithm 2 are declared. The second one is *slicing.c* where Algorithm 2 is implemented.

In the *slicing.h* header file, the following new variables are declared,:

1. *wrr_slice_param_t*;
2. *wrr_dl_init*;

In the *slicing.c* script, the following new functions are created:

1. *wrr_dl_init*
2. *addmod_wrr_slice_dl*
3. *wrr_dl*
4. *wrr_destroy*
5. *remove_wrr_slice_dl*
6. *next_slice_list_looped*

```
# Edit enb_folder/openair2/LAYER2/MAC/slicing/slicing.h
# Edit enb_folder/openair2/LAYER2/MAC/slicing/slicing.c
```

wrr_dl is the function that eNB runs at every-subframe to allocate PRBs first to the slices and then to the UEs. It is the implementation of the pseudo-code given in Algorithm (2).

5.2.1 Examination of the wrr_dl algorithm

RBGs are allocated to the UEs with a period of sub-frame which corresponds to 1ms in LTE-RAT. Therefore the *wrr_dl* algorithm should run less than 1ms.

Run time of *wrr_dl* function linearly depends on the number of UEs and the number of slices. If the number of UEs are M and the number of slices are N, then the *wrr_dl* function will have the big O as $\mathcal{O}(M * N)$. In order to see the run-time characteristic of the *wrr_dl* function, *timespec* struct of C language with *pthread_getcpuclockid*() is used. The time difference between right after *wrr_dl* functions starts and right before it terminates is calculated. An equal amount of traffic per second is generated for the system and $\tilde{4}5000$ samples are collected. The mean value of the collected samples can be seen in Figure 5.1 for various number of UEs and slices. The run times are in microseconds for the given number of UEs and slices.

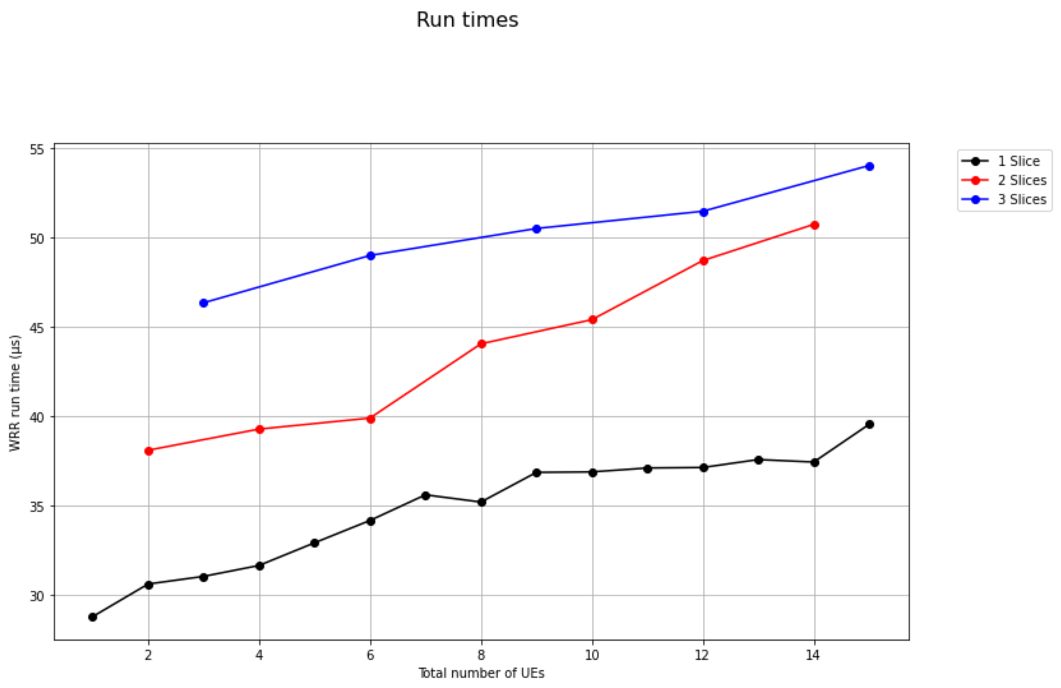


Figure 5.1: Run times of wrr_dl function

The following snapshot provides an example to configure WRR based slices using FlexRAN. The following json file can be used to create 2 slices where the first slice has a weight of 4 and the second slice has a weight of 3.

```
{
    "dl": {
        "algorithm": "WRR",
        "slices": [
            {
                "id": 0,
                "wrr": {
                    "weight": 4
                }
            },
            {
                "id": 1,
                "wrr": {
                    "weight": 3
                }
            }
        ]
    },
    "ul": {
        "algorithm": "None"
    }
}
```

This would result with a WRR vector ($\mathcal{W}_{wrr}$) of $\{0, 0, 0, 0, 1, 1, 1\}$ where the elements of the vector represent ID of the slices. Assuming the json file is named as "*slice.json*", it can be sent to northbound of flexRAN app using the following command:

```
$ curl -X POST http://127.0.0.1:9999/slice/enb/-1 --data-binary "
   @slice.json"
```

Considering a 5MHz channel where half of the spectrum is used for DL and the other half is used for UL, there would be 50PRBs per sub-frame with 15kHZ sub-carrier spacing. Considering 4G RAT, 3 PRBs will form a Resource Block Group (RBG) hence there would be 17 RBGs at a given sub-frame. With the given json file above, there will be two slices (slice 0 and slice 1) with the weights of 4 and 3 respectively. Consequently the wrr_dl function allocates the RBGs to the slices as follows (the snapshot is taken at frame 747, from subframe 2 to subframe 4 where the slices had full buffers):

Frame.Subframe	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
747.2	0	0	0	0	1	1	1	0	0	0	0	1	1	1	0	0	0
747.3	0	1	1	1	0	0	0	0	1	1	1	0	0	0	0	1	1
747.4	1	0	0	0	0	1	1	1	0	0	0	0	1	1	1	0	0

Chapter 6

The experiments and the performance comparison of the schedulers

6.1 The experimental set-up

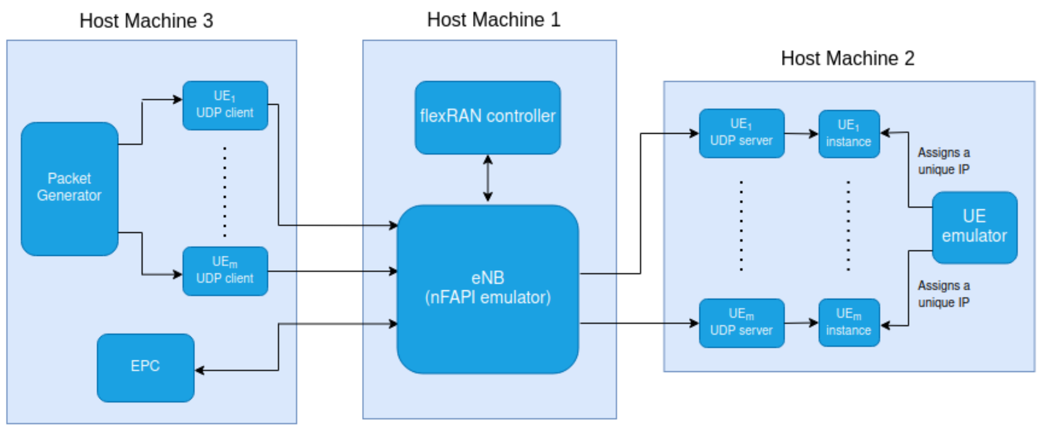


Figure 6.1: The components used for the experiments

As it was explained in the Section 4, OAI-nFAPI emulator can emulate multiple UEs. The UE emulator assigns a unique IP address for the each UE instance, hence it is possible to create DL traffic from EPC to the each UE, respectively. In order to create a controlled traffic, we implement a pair of UDP client-server as well as a traffic generator in Python language. Additionally, as it is not feasible to run all the components manually, we provide a bash script that automates most of the experimental steps. Once the EPC, flexRAN controller, eNB & UE emulators are started then the bash script can be executed to conduct an experiment. The bash script takes seven inputs and executes the following steps:

1. Create some slices for the DL (either static or wrr) by sending "POST" curl to flexRAN controller
2. Initialize UDP servers (one for each UE instance)
3. Initialize UDP clients (one for each UE instance)
4. Initialize the traffic generator
5. Sleep for the experiment duration (seconds)
6. Terminate the eNB and UE emulators as well as the UDP clients and servers

The bash script needs to be run at the UE-side. Assuming the EPC runs at a machine called "epc2" with IP adress of 129.97.228.199 and the eNB runs at a machine called "ydemirci" with the IP address of 129.97.228.172, then the bash script is going to be as follows:

```
#Inputs: 1. <Number of UEs(int)>
#        2. <Packet length(int)>
#        3. <Number of UEs(int)>
#        4. <Lambda weights(string)>
#        5. <Number of UEs per slice(string)>
#        6. <Gama(int)>
#        7. <Experiment Duration(int)>

# An example use of the script:
#./exp.sh 15 1500 3 "1,1,1" "5,5,5" 2954 15

#In case if any server is left open
declare -i k=5002
for (( c=1; c<=$1; c++ ))
do
    #PID of the process
    pid=$(lsof -i :$k| awk '{print $2}' | sed -n '2p')
    #Kill it, in case if the server does not close
    kill -9 $pid &
    k=$(( k + 1 ))
done

#Kill the UE clients in case if any left open
sshpass -p pw ssh epc2@129.97.228.199 \
" echo \"pw\" | sudo -S kill -9 \
\$(ps -aux | grep \"python3 client\" | awk '{print \$2}')"
sleep 0.5


#Create 3 static slices
sshpass -p pw ssh ydemirci@129.97.228.172 \
"cd Desktop/flex && curl -X POST \
http://127.0.0.1:9999/slice/enb/-1 --data-binary \"@slice1.json\" "
sleep 1
#Create 3 wrr slices
sshpass -p pw ssh ydemirci@129.97.228.172 \
"cd Desktop/flex && curl -X POST \
http://127.0.0.1:9999/slice/enb/-1 --data-binary \"@slice2.json\" "
sleep 1
#Assign UEs to the respective slices
sshpass -p pw ssh ydemirci@129.97.228.172 \
"cd Desktop/flex && curl -X POST \
http://127.0.0.1:9999/ue_slice_assoc/enb/-1 --data-binary \"@change.json\" "
sleep 1

#Create the UDP servers for the UEs
declare -i x=5002
for (( c=1; c<=$1; c++ ))
do
    #interface name of the UE
    ueName=$( printf 'oaitun_ue%d' $c )
    #IP of the UE
    ip=$(/sbin/ip -o -4 addr list $ueName | awk '{print $4}' | cut -d/ -f1)

    python3 server.py $ip $x &
    sleep 0.05
    x=$(( x + 1 ))
done
sleep 0.01

#Create the UDP clients for the UEs
declare -i y=5002
declare -i q=6000
for (( c=1; c<=$1; c++ ))
do
    #interface name of the UE
    ueName=$( printf 'oaitun_ue%d' $c )
    #IP of the UE
    ip=$(/sbin/ip -o -4 addr list $ueName | awk '{print $4}' | cut -d/ -f1)

    sshpass -p pw ssh epc2@129.97.228.199 python3 client.py $ip $y $2 $q &
    y=$(( y + 1 ))
    q=$(( q + 1 ))
    sleep 0.1

done
sleep 0.01

#Initialize the traffic generator
sshpass -p pw ssh epc2@129.97.228.199 python3 trafficGenerator.py
$3 $4 $5 $6 $7 $2 &

#Let the experiment run for the given time duration
sleep $7

echo "Killing the eNB and the servers"

#Kill eNB
sshpass -p pw ssh ydemirci@129.97.228.172 \
" echo \"pw\" | sudo -S kill -9 \
\$(ps -aux | grep ./ran_build | awk '{print \$2}'| sed -n '2p')"

#Kill the UE
pidUE=$(ps -aux | grep ./ran_build | awk '{print $2}'| sed -n '2p')
echo "pw" | sudo -S kill -9 $pidUE
```

```
#Kill the UE servers
declare -i z=5002
for (( c=1; c<=$1; c++ ))
do
    #PID of the process
    pid=$(lsof -i :$z| awk '{print $2}' | sed -n '2p')

    #Kill it, in case if the server does not close
    echo "pw" | sudo -S kill -9 $pid

    z=$(( z + 1 ))
done
```

6.1.1 UDP servers-clients

We implement a very straightforward UDP server and client using Python. For each UE instance, a pair of a server and a client are initialized. The clients run at the EPC-side whereas the servers run at the UE-side.

Whenever a packet is generated by the traffic generator, it is sent to one of the clients. Then the client forwards the packet to the eNB and the eNB sends the packet to the respective server (UE-instance). Consequently, it is possible to create simultaneous DL traffic over the emulated LTE-network for multiple UEs.

6.1.2 The traffic generator

We implement a traffic generator where the inter-arrival time of the packets are exponentially decaying and the packet size is fixed. Consequently, the traffic generation model is a Poisson process with M/D/1 queuing. The generation of a packet at a time is not dependent on the previous time-states hence the traffic generation is a memoryless process. Once the UDP clients and servers are up, the traffic generator is run at the EPC-side by the bash script. The traffic generator takes five inputs. The pseudo-code for the traffic generator can be seen in Algorithm (3)

```
Algorithm 3: The traffic generator
  Input: 1. sliceList
         2. lambdaValues
         3. UEsInSlices
         3. Gama
         4. experimentDuration
         5. packetSize
1 Get currentTime ;
2 startTime = currentTime ;
3 sentTime = currentTime ;
4 sampledInterval = expovariate(Gama) ;
5 while ((currentTime - starTime) < experimentDuration) do
6     if (currentTime-sentTime) > sampledInterval then
7         sentTime = currentTime ;
8         sampledInterval = expovariate (Gama) ;
9         selectedSlice = choices (sliceList , lamdaValues, k=1) ;
10        select a UE from the selectedSlice ;
11        Send a packet to the selected UE (a packet size of packetSize) ;
12    Update currentTime ;
```

expovariate given in the line 4 and 8 of Algorithm (3) is a method provided by “random” library of Python. It is possible to sample an inter-arrival time with exponential decaying distribution using expovariate(). It takes Lambda as the input.

Once a packet is generated, a slice is randomly selected according to the weights given by $lambdaValues$. choice() method provided by “random” library of Python can be used to randomly select a slice. choice() method returns a randomly selected element from the first given sequence (sliceList) with the probability distribution given by the second sequence (lambdaValues). Once a slice is selected, then a UE can be chosen with an equal probability among the UEs of the slice.

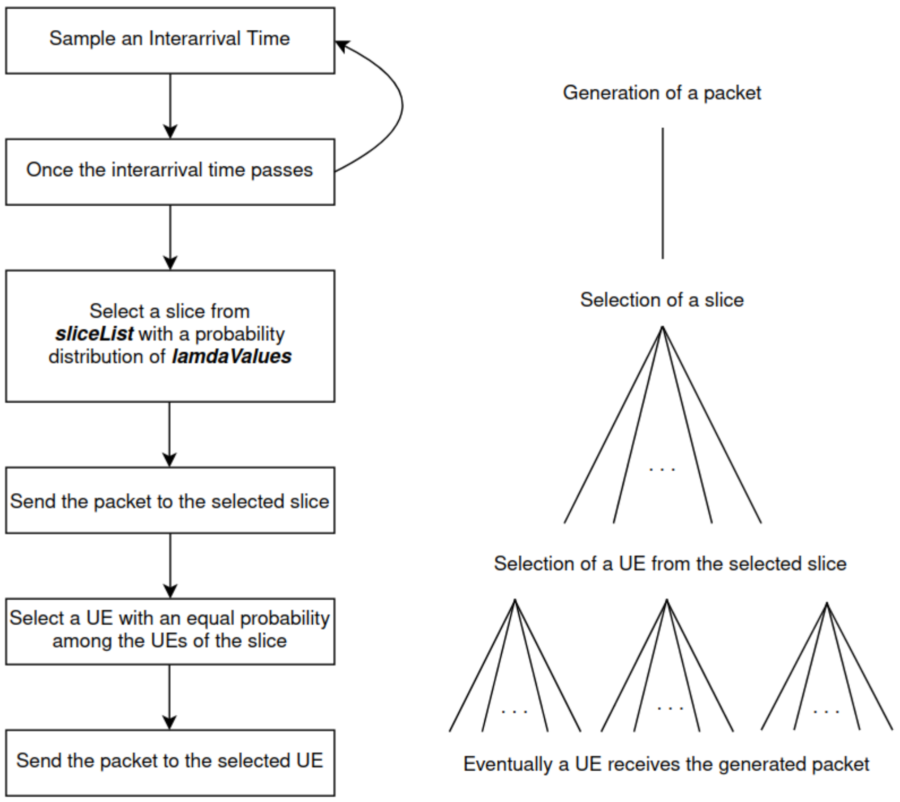

Sample an Interarrival Time
Once the interarrival time passes
Select a slice from *sliceList* with a probability distribution of *lamdaValues*
Send the packet to the selected slice
Select a UE with an equal probability among the UEs of the slice
Send the packet to the selected UE
Generation of a packet
Selection of a slice
Selection of a UE from the selected slice
Eventually a UE receives the generated packet

Figure 6.2: Packet generation diagram

6.1.3 Sampling MCS values

Since the nFAPI emulator does not provide the PHY layer, the CQI/MCS values are hard-coded with some constants (MCS value of 28) in the source codes. In order to understand the impacts of CQI/MCS values better, we decided to sample the MCS values rather than using the hard-coded constant value.

The MCS values can be manipulated by changing the $eNB\_UE\_stats->dlsch\_mcs1$ variable given in the $eNB\_scheduler\_dlsch.c$ script:

```
$ cd enb_folder/openair2/LAYER2/MAC
$ Edit eNB_scheduler_dlsch.c
```

We implement a sub-routines called *myRand* which samples new MCS values for all the UEs attached to the eNB once it is called. The sub-routine can be called in varying periodicity (e.g. every x sub-frame or every frame). The probability of sampling a specific MCS value depends on the MCS distribution given in Figure 6.3.

This distribution is taken from [1] which is published in EURASIP Journal on Wireless Communications and Networking (2017). The authors collected MCS values reported by some UEs during busy hours from a real LTE network. They mentioned that MCS 10 and 17 are very close to their adjacent MCS in terms of spectral efficiency and are hence not utilized in the network. Consequently, the probability of sampling MCS 10 or 17 is considered as 0.

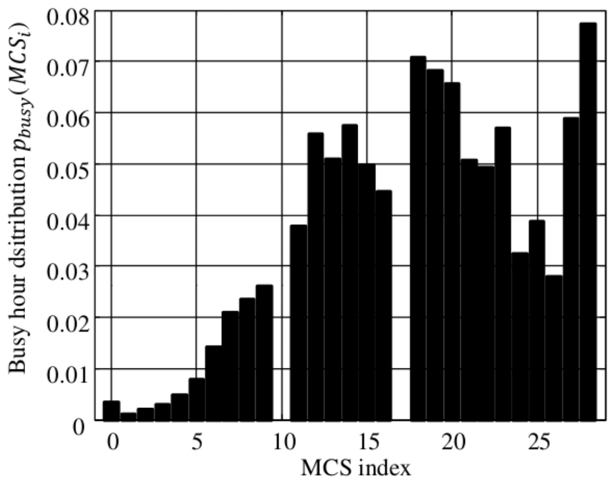


Figure 6.3: The MCS distribution which is taken from [1]

We run the *myRand* sub-routine 10000 times and aggregated the occurrence of each index. The obtained probability distribution can be seen in Figure 6.4 which is consistent with distribution given in Figure 6.3

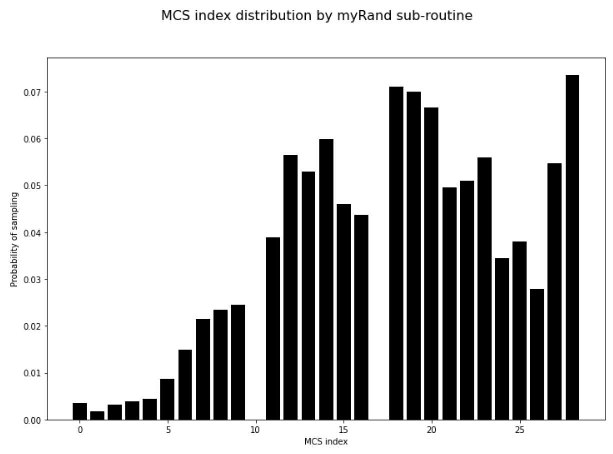


Figure 6.4: The MCS distribution obtained from running *myRand* sub-routine 10000 times

6.1.4 Triggering HARQ re-transmission

As aforementioned, the nFAPI emulator does not provide a physical channel model. Consequently, the transmission from eNB to the UEs occurs through wired connection. Therefore no packet loss occurs during the transmission from eNB to the UEs. This is clearly not the case in a wireless channel. In order to compensate this flaw and to mimic the characteristics of a wireless channel, we decided to drop the packets received by the UEs intentionally and trigger HARQ re-transmissions.

In the default configuration, the UEs are hard-coded to send ACK. In order to change this and send NACK with some probability, we manipulate the $phy\_stub\_UE.c$ script:

```
$ cd ue_folder/openair2/PHY_INTERFACE
$ Edit phy_stub_UE.c
```

In order to send NACK (in FDD mode), it is required to manipulate $pdu->harq\_indication\_fdd\_rel13.harq\_tb\_n[0]$ variable in $fill\_uci\_harq\_indication\_UE\_MAC()$ function. The default value is hard-coded as "1". If this value is changed to "2", UE will send a NACK instead of an ACK. This will trigger HARQ re-transmission at the eNB-side for the next sub-frame. There are two important points to be careful before manipulating the ACK messages:

1. If the UE sends a NACK during the initial attachment to the eNB, then the UE may not be able to establish a connection at all. Therefore, it is important to start sending NACK once all the UEs have successfully established a connection to the eNB.

2. Depending on the configuration, HARQ can be encoded to a more than 1 byte (e.g. 2 bytes). In such a case it is required to change both of the bytes to “2”.

6.2 The experimental scenarios

We consider three VNOs and three UEs per each VNO. Depending on the amount of sub-channels brought by the VNOs and the traffic load variation, we consider four different scenarios:

1. In the first scenario, all the VNOs bring the same amount of sub-channels and experience similar traffic loads.
2. In the second scenario, the third VNO brings less sub-channels than the others, however the traffic load is still the same among the VNOs.
3. In the third scenario, all the VNOs bring the same amount of sub-channels however the third VNO experiences significantly more traffic than the others.
4. In the forth scenario, the third VNO brings less amount of sub-channels and it experiences significantly more traffic than the others.

In other words, the first two VNOs show similar characteristics in all the scenarios. Whereas the third VNO behaves as a free-rider except the first scenario. For each scenario:

- MCS values are hard-coded as 28 for all the UEs.
- BLock Error Rate (BLER) is set as 0, hence no packet corruption or loss occurs during the packet transmission from eNB to UEs.
- There are three slices (3 VNOs).
- Each VNO has three active UEs.
- α_q variable represents the sub-channels brought by the VNO q.
- p_q variable represents the probability of forwarding a generated packet to slice q.
- 3 PRBs form a single RBGs at a given sub-frame.
- A packet length is 1500 bytes.

Exps:	MCS	BLER	# of slices	UEs per slice	α_q	p_q	# of RBGS per sub-frame
1	28	0	3	3	$\alpha_1 = \alpha_2 = \alpha_3$	$p_1 = p_2 = p_3$	24
2	28	0	3	3	$\alpha_1 = \alpha_2 = 2\alpha_3$	$p_1 = p_2 = p_3$	25
3	28	0	3	3	$\alpha_1 = \alpha_2 = \alpha_3$	$10p_1 = 10p_2 = p_3$	24
4	28	0	3	3	$\alpha_1 = \alpha_2 = 2\alpha_3$	$10p_1 = 10p_2 = p_3$	25

Table 6.1: The experimental scenarios

The experiments are run considering a confidence interval. At time $t = 0$, we consider a booting period for 10 seconds where the system is initialized. Then, we start recording the data with a period of 30 seconds. At the end of a period, we check if the new period varies less than 5% of the former period. If so, we use the data from the new period, otherwise the experiment continues.

We consider 2 metrics:

1. Average system throughput which is the total number of bytes sent by eNB and acknowledged by an UE per second.
2. Average transmission time of a packet. This is the difference between the time-stamp when a packet is received by a UE and the time-stamp when the packet is sent from the EPC.

6.3 Results

6.3.1 Average Throughput

Experiment 1 and 2

The first and the second experiments are run for the following λ values respectively.

$$\lambda \in \{1150, 1532, 1916, 2300, 2682\}$$
$$\lambda \in \{770, 1056, 1343, 1630, 1916\}$$

Even though the same λ is used for the proposed scheduler or the SOA, a negligible difference may occur in terms of average generated packet per second, due to the non-stochastic nature of the "random" library of Python.

The average system throughput is almost the same for both of the experiments, as it can be seen in Figure 6.5 and 6.6. The proposed algorithm does not provide any better throughput than the SOA.

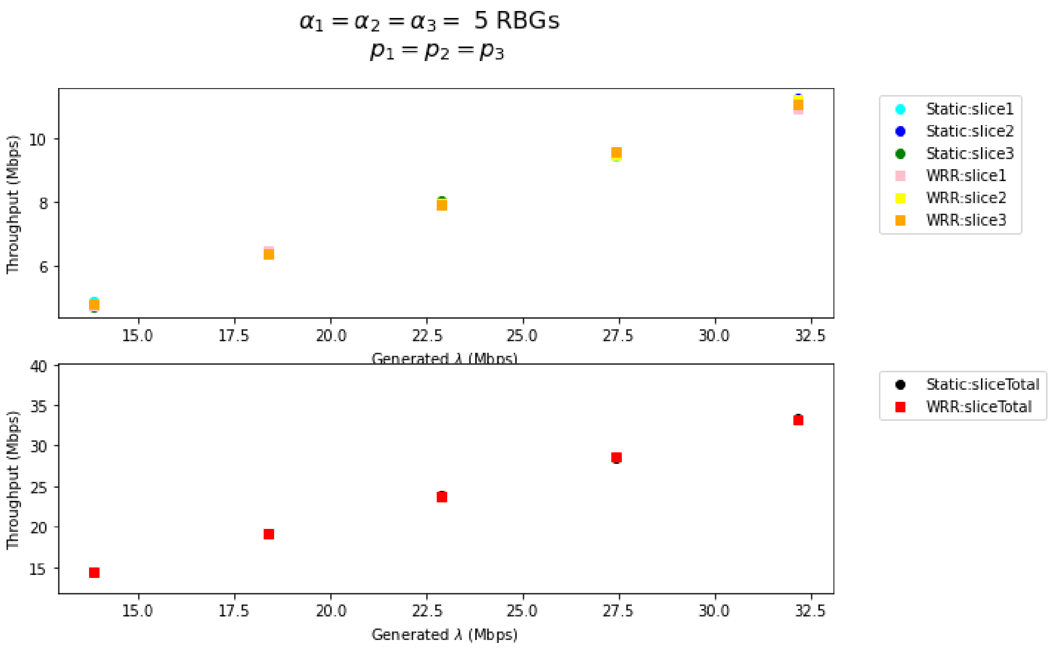


Figure 6.5: The average throughput for the first experimental scenario

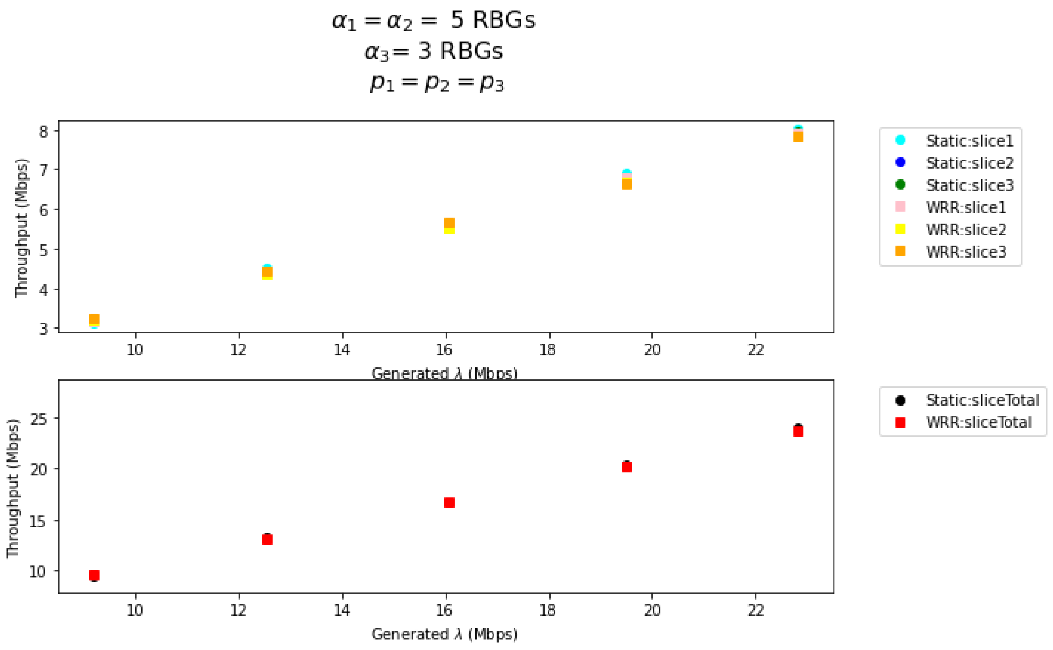


Figure 6.6: The average throughput for the second experimental scenario

The average ratios of the number of idle RBGs divided by the total number of RBGs at a given sub-frame are given in Figure 6.7 and 6.8. Under low traffic conditions, the ratios are almost identical for both the SOA and the proposed scheduler. As the traffic load increases, the proposed scheduler utilizes more RBGs. However, the difference can be considered negligible except the case when $\lambda = 1916$ for slice 3, for experiment 2. As the traffic increases, the sub-channels of the third slice become not sufficient in the case of static sharing. Therefore, the proposed scheduler starts to utilize more RBGs.

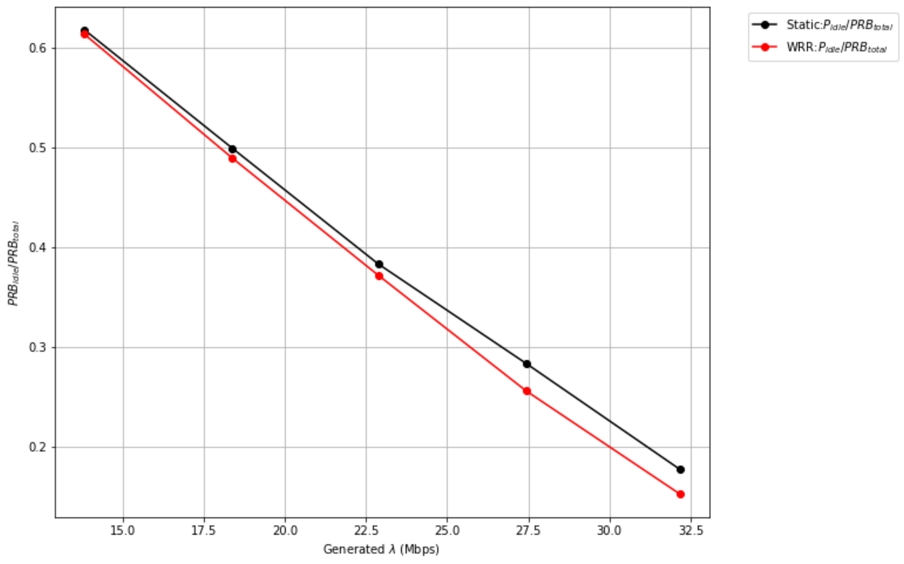


Figure 6.7: Ratio of RBG_{idle}/RBG_{total} for the first experimental scenario

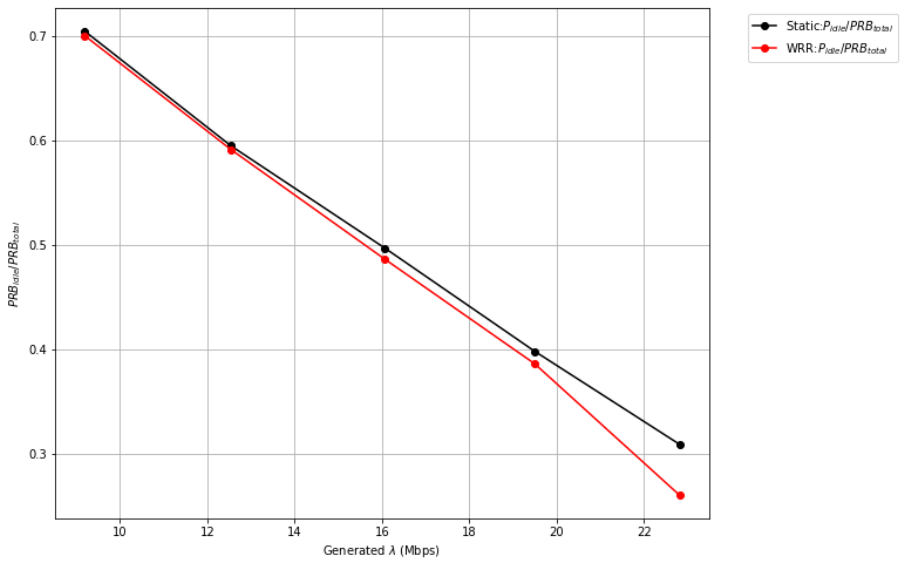


Figure 6.8: Ratio of RBG_{idle}/RBG_{total} for the second experimental scenario

Experiment 3 and 4

The third and the fourth experiments are run for the following λ values respectively.

$$\lambda \in \{253, 453, 653, 853, 1053\}$$

$$\lambda \in \{300, 400, 500, 600, 700\}$$

The lambda values are chosen considering the point where the third slice starts experiencing significant delays compared to the other two slices.

The average throughput characteristics for the third and the fourth experiments are similar to the first two experiments as they are given in Figure 6.9 and 6.10. Both the SOA and the proposed scheduler provide similar average throughput.

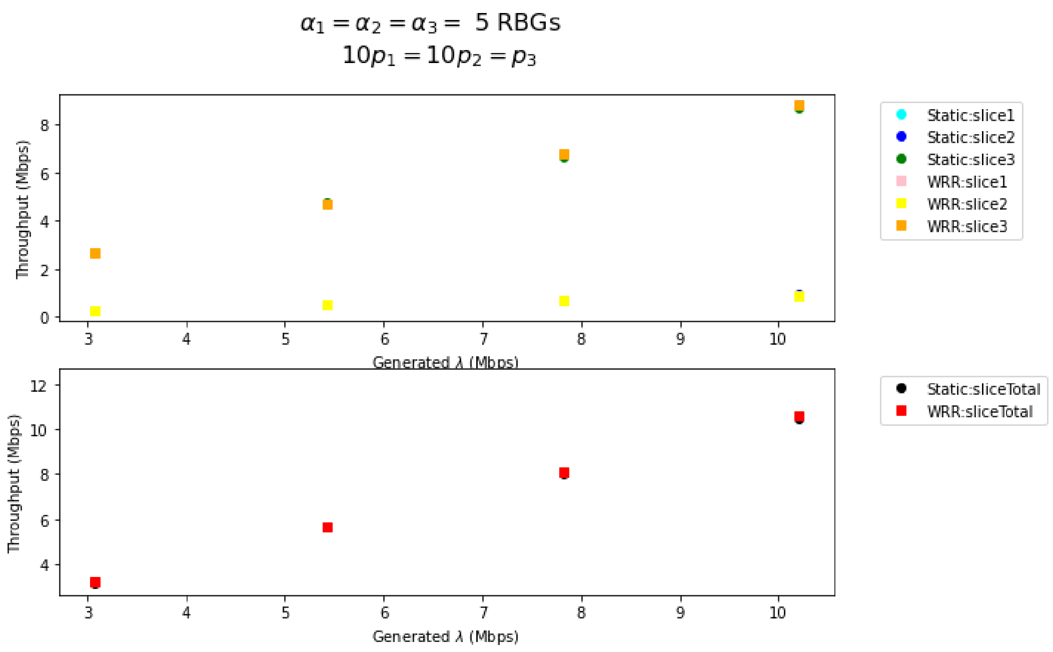


Figure 6.9: The average throughput for the third experimental scenario

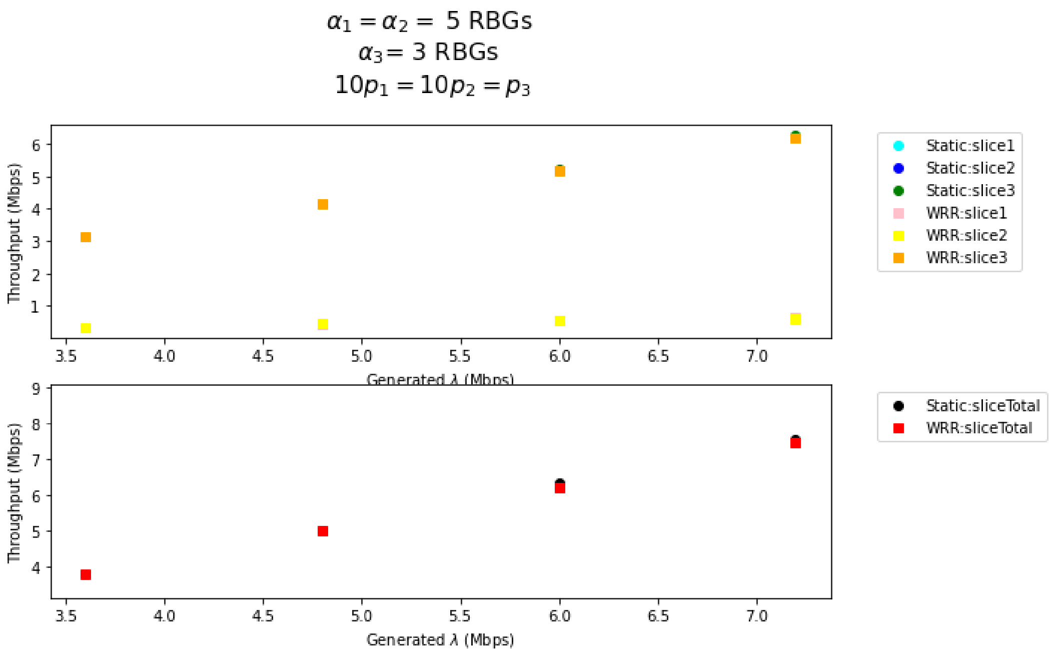


Figure 6.10: The average throughput for the fourth experimental scenario

In the case of RBG utilization per sub-frame, both the SOA and the proposed scheduler show similar RBG utilization under low-traffic. As the traffic load increases, the proposed scheduler utilizes more RBGs. However, this happens due to the third slice which brings less sub-channels but more traffic.

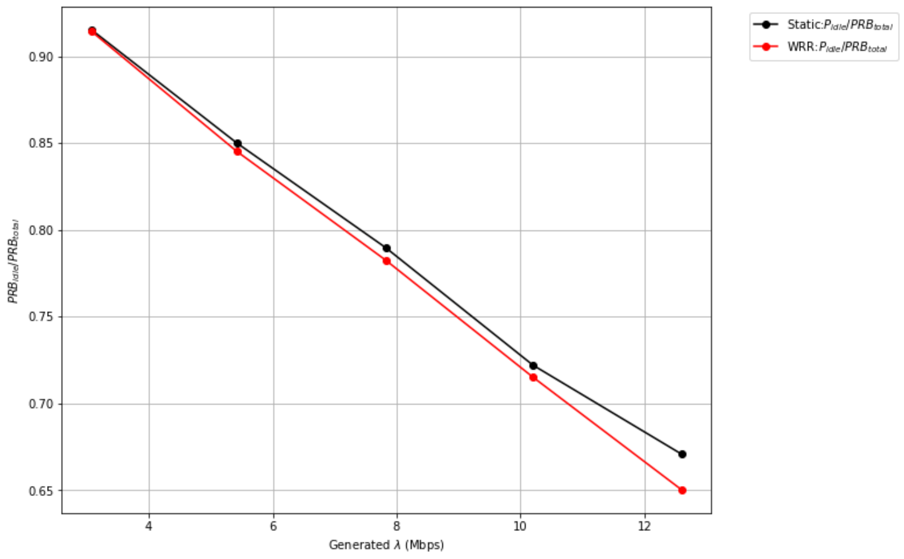


Figure 6.11: Ratio of RBG_{idle}/RBG_{total} for the third experimental scenario

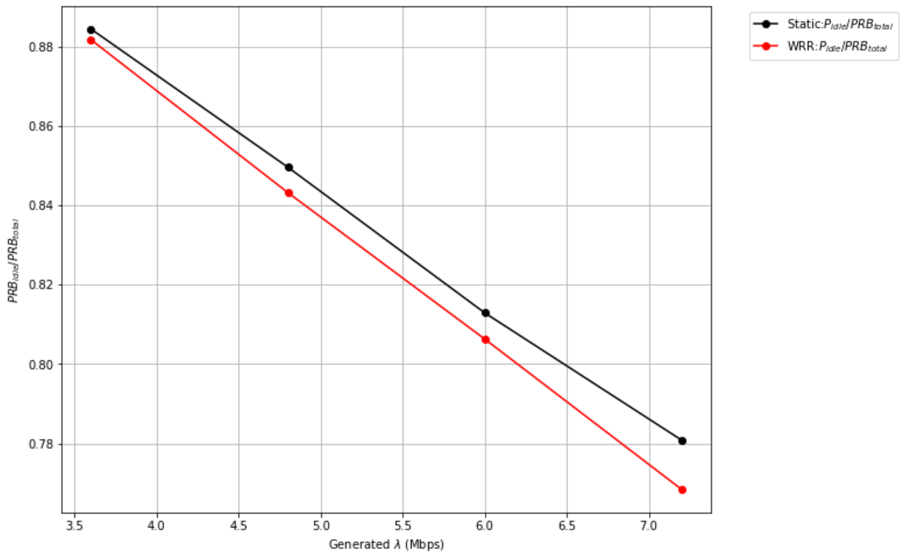


Figure 6.12: Ratio of RBG_{idle}/RBG_{total} for the fourth experimental scenario

6.3.2 Average Transmission time of a packet

In order to measure the transmission time of a packet, time-stamps are obtained at the application layer once a packet is sent from the EPC and once it is received by the UEs. Since the UEs and the EPC are running on different host machines, it is required to synchronize the clocks of the machines. For this purpose, Network Time Protocol (NTP) is used. The machine that hosts the UEs synchronizes to the machine that hosts the EPC. In other words, the EPC machine works as a NTP server whereas the UE machine is the client. In order to measure the transmission time, the time-stamp is encoded into a packet right before it is sent from the EPC. Once this packet is received by the UE, the current time stamp is recorded. Then the time-stamp encoded into the packet is subtracted from the current time stamp to find the transmission time. NTP provides precision in milliseconds.

Experiment 1 and 2

The λ (packet/sec) values used for the experiment 1 and 2 are the followings, respectively:

$$\lambda \in \{1150, 1532, 1916, 2300, 2682\}$$

$$\lambda \in \{770, 1056, 1343, 1630, 1916\}$$

In the first and the second experiments, regardless of the used λ values, the proposed scheduler provides strictly less transmission time than the SOA as it can be seen in Figure 6.13 and 6.14 for all the slices. Especially, in the second experiment, once a certain traffic load is passed, the transmission time significantly increases for the the third slice in the case of static sharing.

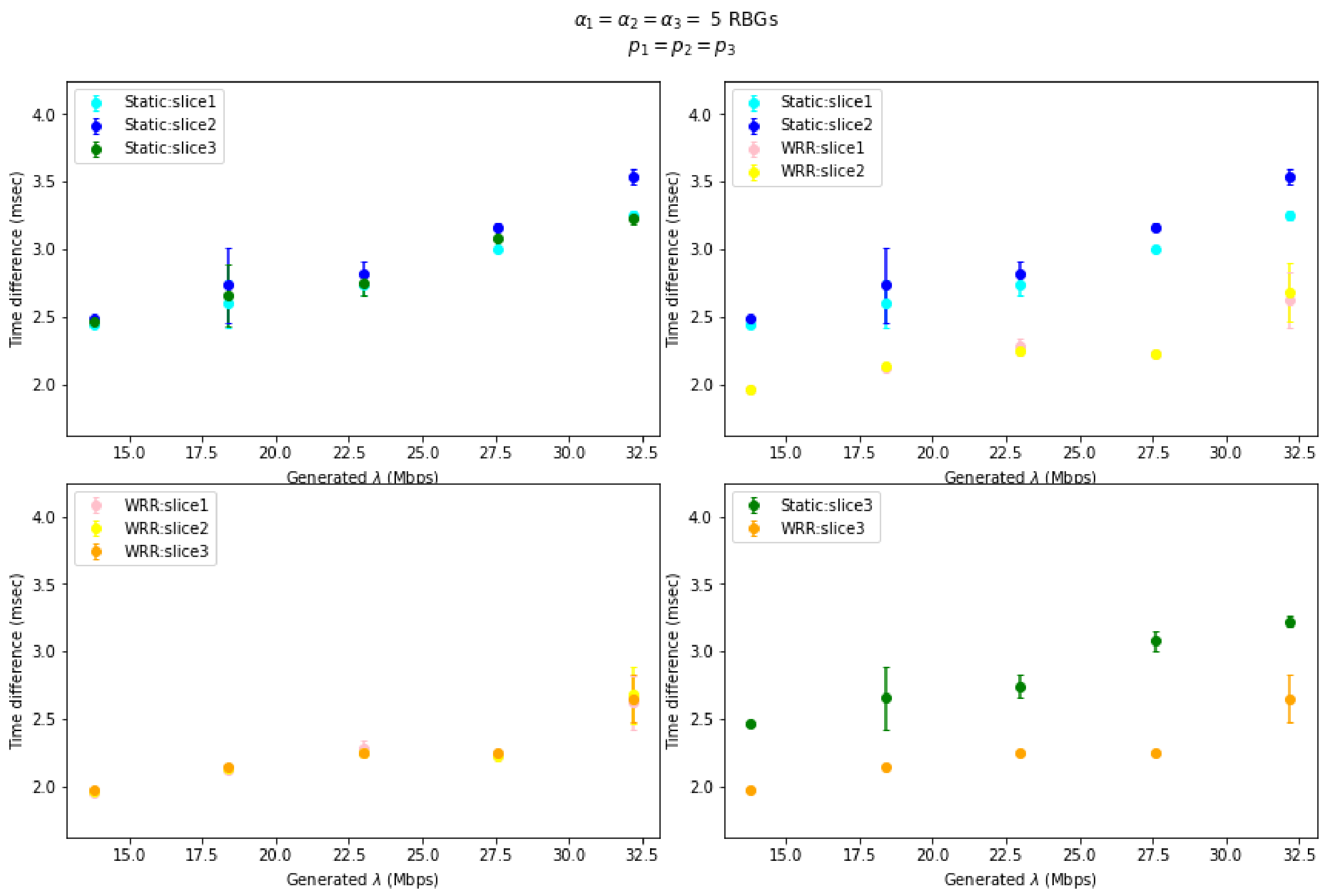


Figure 6.13: Average transmission time of the packets for the first experimental scenario

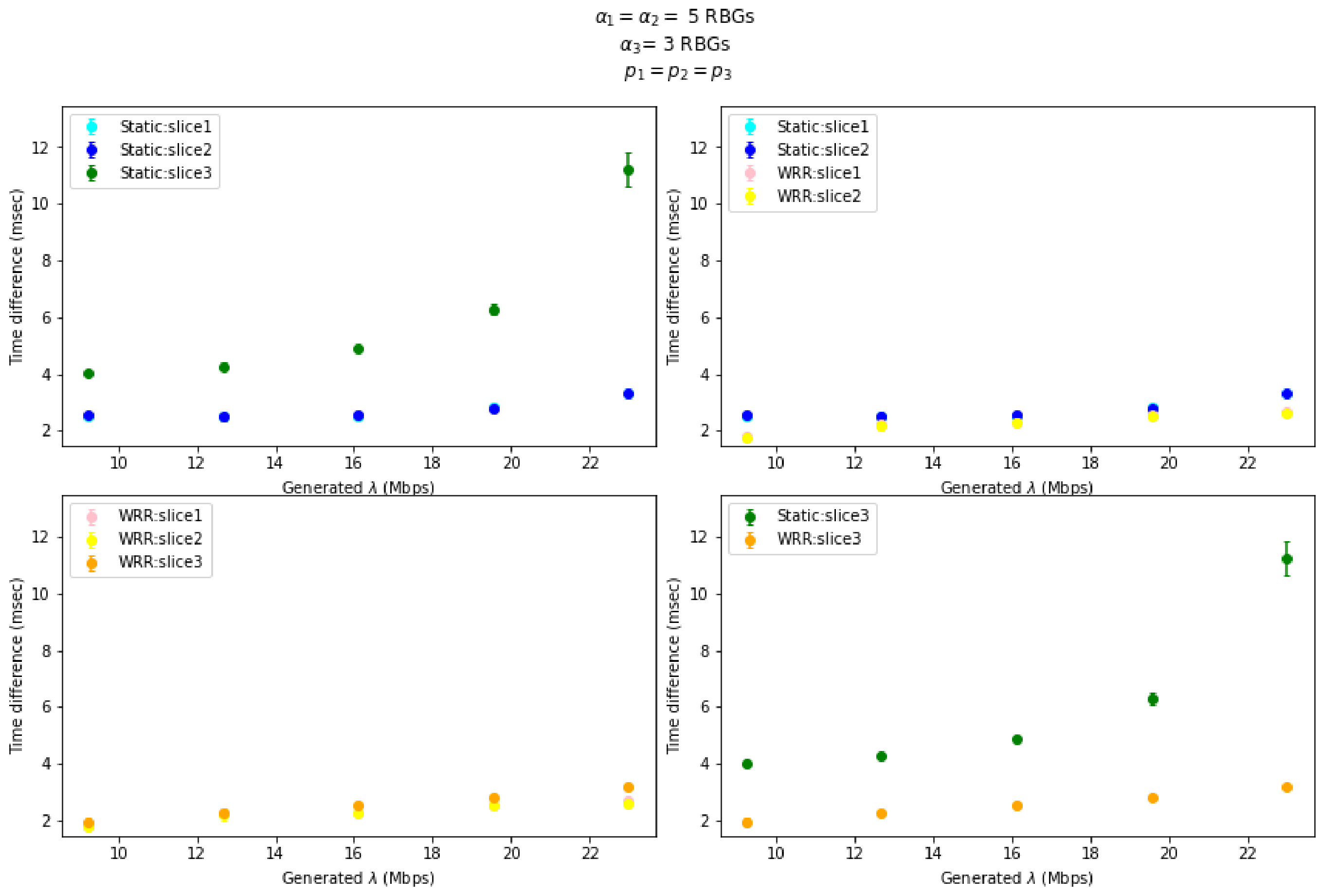


Figure 6.14: Average transmission time of the packets for the second experimental scenario

Experiment 3 and 4

The lambda (packet/sec) values for experiment 3 and 4 are the following, respectively:

$$\lambda \in \{920, 1303, 1686, 2069, 2452\}$$

$$\lambda \in \{750, 900, 1050, 1200, 1350\}$$

Likewise the first two experiments, the proposed scheduler provides strictly less delay than the SOA for the given traffic loads as it can be seen in Figure 6.15 and 6.16.

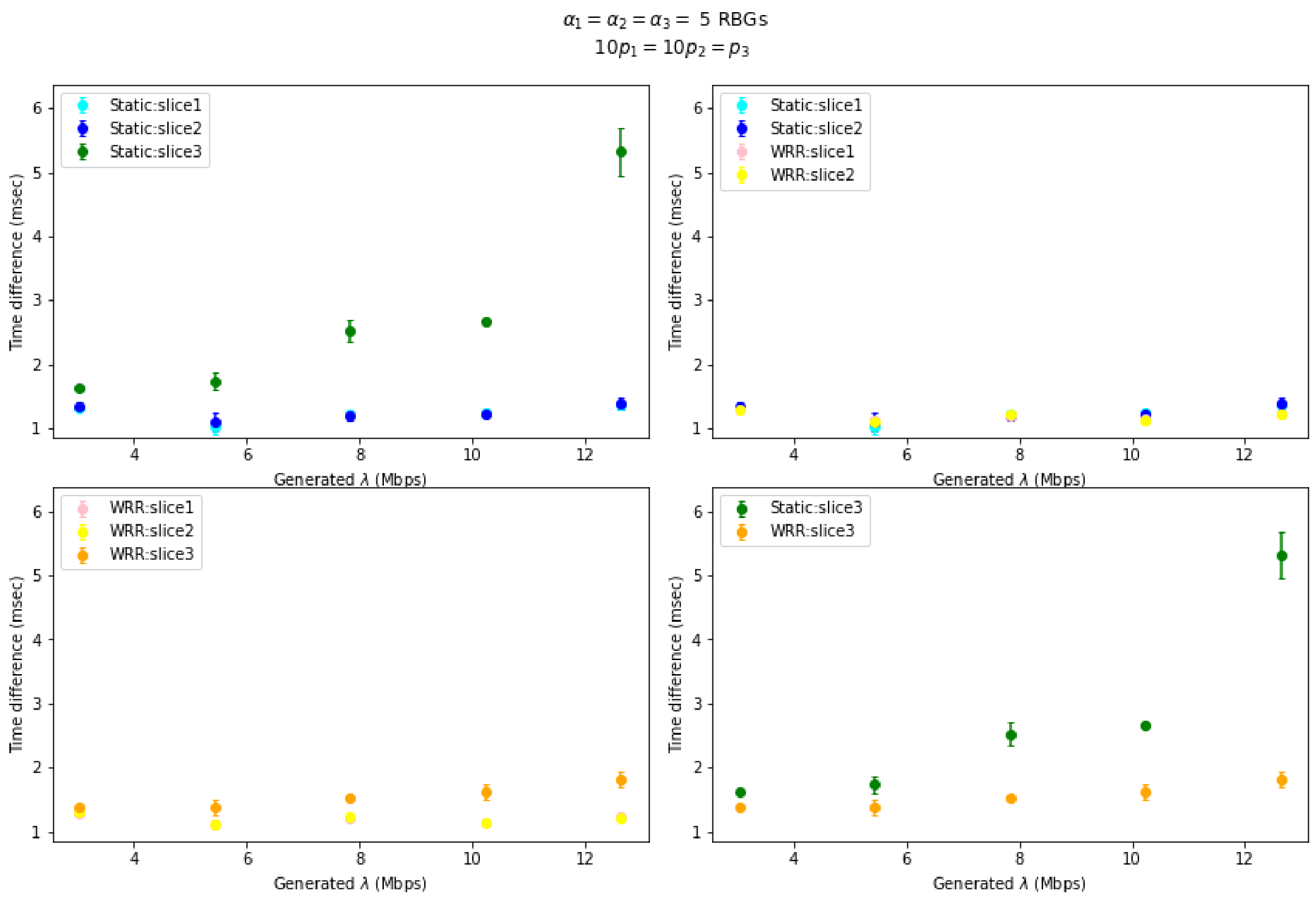


Figure 6.15: Average transmission time of the packets for the third experimental scenario

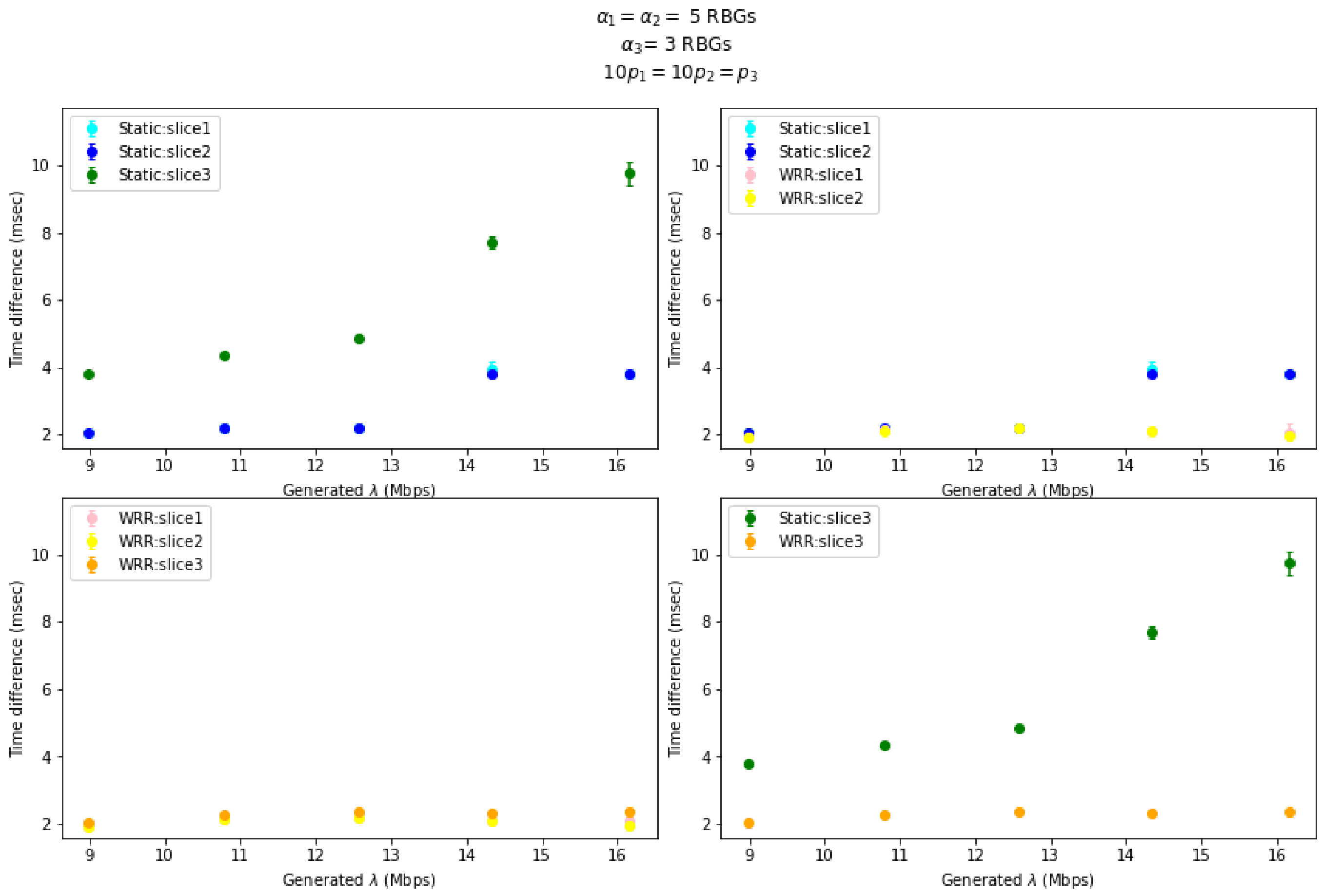


Figure 6.16: Average transmission time of the packets for the fourth experimental scenario

In short, it can be concluded that the proposed scheduler provides less delay than the SOA for all the slices under different traffic loads. Especially, once a certain amount of traffic load is passed, the proposed scheduler provides significantly less delay than the SOA, for the third slice. This might be undesirable to the first two slices which bring more resource but less traffic. Nonetheless, all the slices achieve less delay with the proposed scheduler compared to the SOA.

Chapter 7

Conclusion

We propose a two level slice scheduler for RAN sharing in the context of Neutral Host and we analyze the average throughput and packet transmission time compared to the SOA. The SOA embraces a static channel allocation to the VNOs whereas the proposed two-level-scheduler embraces a WRR sharing. The results show that the proposed scheduler provides similar average throughput compared to the SOA. However, in the case of packet transmission time, the proposed scheduler provides less delay for all the slices than the SOA for the given traffic-loads. After passing a certain traffic-load, the proposed scheduler provides significantly less delay for the slice which brings less resources but more traffic. Considering this may lead to a free-riding problem, it might be undesired. Nonetheless, all the slices achieve less delay with the proposed scheduler compared to the SOA.

Besides the proposed algorithm, we also create detailed configuration guides to set-up OAI platform both in nFAPI emulation and with COTS hardware.

Even though the proposed scheduler provides less delay for all the slices under certain traffic loads, it favors the slices that experience more traffic than the slices with less traffic. This may lead to a free-rider problem which would discourage some VNOs to participate in the RAN sharing setting. As a future work, we are planning to provide another slice-scheduler where we enhance the two-level-scheduler in terms of "fairness". With the enhancement, when a VNO would not use its chance of acquiring a PRB, the scheduler will give the priority to the slices which have less tendency to free-ride. Additionally, we are going to use a different method than NTP to synchronize the computers for achieving higher precision for the delay measurements.

References

[1] Bartelt et al. “5G transport network requirements for the next generation fronthaul interface”. *EURASIP Journal on Wireless Communications and Networking*, 2017.

[2] S. Zhang. “An Overview of Network Slicing for 5G”. *IEEE Wireless Communications*, 26(3), June 2019.

[3] 3GPP. “System Architecture for the 5G System, v.16.8.0”. 2021.

[4] ETSI TS. “LTE Network sharing Architecture and functional description”.

[5] M. G. Kibria et al. “Shared Spectrum Access Communications: A Neutral Host Micro Operator Approach”. *IEEE Journal on Selected Areas in Communications*, 35(8):1741 – 1753, May 2017.

[6] G. Baldoni et al. “Edge Computing Enhancements in an NFV-based Ecosystem for 5G Neutral Hosts”. *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, November 2018.

[7] X. Foukas et al. “Iris: Deep Reinforcement Learning Driven Shared Spectrum Access Architecture for Indoor Neutral-Host Small Cells”. *IEEE Journal on Selected Areas in Communications*, 37(8):1820 – 1837, July 2019.

[8] X. Foukas et al. “Network Slicing in 5G: Survey and Challenges”. *IEEE Communications Magazine*, 55(5):94–100, May 2017.

[9] J. Ordonez-Lucena et al. “Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges”. *IEEE Communications Magazine*, 55(5):80–87, May 2017.

[10] R. Ferrus et al. “On 5G Radio Access Network Slicing: Radio Interface Protocol Features and Configuration”. *IEEE Communications Magazine*, 56(5):184–192, May 2018.

[11] X. Foukas et al. “FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks”. *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 427–441, December 2016.

[12] X. Foukas et al. “Orion: RAN slicing for a flexible and cost-effective multi-service mobile network architecture”. *Proceedings of the 23rd Annual International Conference on mobile Computing and Networking (MobiCom)*, pages 127–140, 2017.

[13] D. Johnson et al. “NexRAN: Closed-loop RAN slicing in POWDER - A top-to-bottom open-source open-RAN use case”. *The 15th ACM Workshop on Wireless Network Testbeds, Experimental evaluation & CHaracterization (WiNTECH 2021)*, 2021.

[14] A. Ksentini et al. “Toward Enforcing Network Slicing on RAN: Flexibility and Resources Abstraction”. *IEEE Communications Magazine*, 55(6), June 2017.

[15] M. A. Habibi et al. “The Structure of Service Level Agreement of Slice-based 5G Network”. *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, September 2018.

[16] B. Khodapanah et al. “Fulfillment of Service Level Agreements via Slice-Aware Radio Resource Management in 5G Networks”. *IEEE 87th Vehicular Technology Conference*, June 2018.

[17] R. Schmidt et al. “Slice Scheduling with QoS-Guarantee towards 5G”. *IEEE Global Communications Conference*, December 2019.

[18] A. Papageorgiou et al. “SLA Management Procedures in 5G Slicing-based Systems”. *European Conference on Networks and Communications*, June 2020.

[19] 3GPP. “Evolved Universal Terrestrial Radio Access Medium Access Control protocol specification, v.16.6.0”. 2021.

[20] 3GPP. “Evolved Universal Terrestrial Radio Access Physical layer procedures, v.16.7.1”. 2021.

[21] Small Cell Forum. “5G FAPI: Network Monitor Mode API, scf224.10.01”. 2020.

[22] 3GPP. “Study on CU-DU lower layer split for NR, v15.0.0”. 2017.

[23] Navid Nikaein et al. “OpenAirInterface: A flexible platform for 5G research”. *ACM SIGCOMM Computer Communication Review*, 44(5):33–38, June 2020.

[24] Q. Wang et al. "Enable Advanced QoS-Aware Network Slicing in 5G Networks for Slice-Based Media Use Cases". *IEEE Transactions on Broadcasting*, 65(2):444–453, June 2019.

[25] Robert Schmidt et al. "FlexVRAN: A flexible controller for virtualized RAN over heterogeneous deployments". *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, 2019.

[26] X. Vasilakos et al. "ElasticSDK: A Monitoring Software Development Kit for enabling Data-driven Management and Control in 5G". *IEEE/IFIP Network Operations and Management Symposium*, 2020.

[27] M. Irazabal et al. "Dynamic buffer sizing and pacing as enablers of 5G low-latency services". *IEEE transactions on mobile computing*, 2020.

[28] A. Ksentini et al. "Providing low latency guarantees for slicing-ready 5G systems via two-level MAC scheduling". *IEEE Network*, 32(6):116 – 123, November 2018.

[29] K. Katsalis et al. "JOX: An event-driven orchestrator for 5G network slicing". *IEEE/IFIP Network Operations and Management Symposium*, pages 1–9, April 2018.

[30] M. Irazabal et al. "Preventing RLC Buffer Sojourn Delays in 5G". *IEEE Access*, 9:39466–39488, March 2021.

[31] N. Makris et al. "Cloud-based Convergence of Heterogeneous RANs in 5G Disaggregated Architectures". *IEEE International Conference on Communications (ICC)*, pages 1–6, May 2018.

[32] B.Z. Hsieh et al. "Design of a UE-specific uplink scheduler for narrowband Internet-of-Things (NB-IoT) systems". *International Conference on Intelligent Green Building and Smart Grid (IGBSG)*, pages 1–5, April 2018.

[33] S. Khatibi et al. "Modelling and implementation of virtual radio resources management for 5G Cloud RAN". *EURASIP Journal on Wireless Communications and Networking*, 9:1–16, Dec 2017.

[34] "ETSI GS NFV-MAN" 001 v1.1.1. "Network Functions Virtualisation (NFV); Management and Orchestration". 2014.

[35] O-RAN Alliance. "O-RAN Architecture Overview".