



**UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH**

**Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona**

**Implementation and Evaluation of Open Source LTE-
EPC Software**

A Degree Thesis

**Submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de
Barcelona**

Universitat Politècnica de Catalunya

by

Chenhan Zhu

**In partial fulfilment
of the requirements for the degree of
MASTER IN TELECOMMUNICATIONS ENGINEERING**

Advisor: Dr. ILKER DEMIRKOL

Barcelona, June 2018

Abstract

In general, Software Defined Radio (SDR) is based on a common hardware platform to use software to implement various communication modules. There are two crucial keywords in this concept: "universal hardware platform" and "software." The "universal hardware platform" means that we can implement a variety of communication functions based on this hardware platform, not that a hardware platform can only implement one kind of communication function. "Software" to implement the communication module is relative to the traditional radio technology. Since the SDR is more and more popular in the scientific research field, lots of different Open Source SDR Software can be used. In this thesis, I used two different open source LTE and EPC software to run the same setup in order to implement and evaluate the advantages and disadvantages of these software.

Keywords: LTE, eNB, EPC, OpenAirInterface, srsLTE.

Acknowledgements

First of all I would like to thank my parents, who has always supported me and encouraged me to overcome one obstacle after another.

Second, I must thank the CSC(China Scholarship Council) for giving me this chance and financial support.

What's more, I was really grateful to the supervision and ideas of my advisor, Ilker Demirkol. During this four months, his way of work and attitude was really inspiring for me.

I would also like to thank Nitin who gived me a very warm welcome when I first came, Christian who give me advices to be a researcher, Sidarth who is my best friend here.

And finally, I am grateful for this amazing city Barcelona, which give me a memory of a lifetime and taught me how to accept new things in my life. And I will never finished my thesis without these helps.

Revision history and approval record

Revision	Date	Purpose
0	18/05/2018	Document creation
1	28/05/2018	Document revision

Written by:		Reviewed and approved by:	
Date	26/05/2018	Date	28/05/2018
Name	Chenhan Zhu	Name	Dr. Ilker Demirkol
Position	Project Author	Position	Project Supervisor

Contents

Abstract	2
Revision history and approval record	4
1. Introduction.....	7
2. Background	8
2.1 LTE.....	8
2.1.1 Concept.....	8
2.1.2 Architecture of LTE	9
2.2 OAI.....	10
2.3 srsLTE	11
3. The State of the Art	12
3.1 OAI Studies	12
3.1.1 Physical layer link simulation	12
3.1.2 LTE system-level simulation.....	13
3.1.3 SDR-based LTE system	13
3.3 srsLTE Studies.....	15
3.3.1 Main Feature of srsLTE.....	15
3.3.2 Test environment of srsLTE.....	16
3.3.3 srsLTE's application.....	16
4. Implementation.....	18
4.1 LTE Link on OAI.....	18
4.1.1 Hardware verification.....	18
4.1.2 Operating system verification.....	20
4.1.3 Get repository	22
4.1.4 Connect OAI eNB with COTS UE.....	22
4.2 LTE Link on srsLTE	42
4.2.1 Get repository	42
4.2.2 LTE network construction.....	44
5. Evaluations	52

5.3 Channel quality of B210 and E3372.....	52
5.2 Performance of OAI in different Bandwidth.....	54
5.2 Performance comparison between srsLTE and OAI	57
6. Conclusions and future work.....	60
Bibliography	61

1. Introduction

Software Radio uses modern software to manipulate and control the traditional "pure hardware circuit" wireless communication technology. The important value of software radio technology lies in the fact that the traditional hardware radio communication equipment only serves as the basic platform for wireless communication, and many communication functions are implemented by software, which breaks the realization of the communication function of the equipment in history and only depends on the development of the hardware. The emergence of software radio technology is the third revolution in the field of communications after from fixed communication to mobile communication and from analog communication to digital communication.

The key idea of the so-called software radio is to construct a general hardware platform with openness, standardization, modularization, and various functions, such as working frequency band, modem type, data format, encryption mode, communication protocol, etc., all completed by software. Also, another important purpose is to make broadband A/D and D/A converters as close to the antenna as possible to develop a new generation of wireless communication systems that are highly flexible and free. It can be said that this platform is a platform that can be controlled and redefined by software. Different software modules can be used to achieve different functions, and the software can be upgraded and updated. Its hardware can also constantly update modules and can be upgraded like a computer. Since software radio's various functions are implemented in software, adding a new software module is a must if you want to implement a new service or modulation method. At the same time, because it can form a variety of modulation waveforms and communication protocols, it can also communicate with various radio stations in the old system, greatly prolonging the service life of radio stations, and also saving costs.

The current mainstream SDR-based LTE platform in the world are OAI, srsLTE, OpenLTE, and so on. And since the function of OpenLTE is too weak, it is not used by a lot people. For Amarisoft, though it is the best performance SDR LTE platform at present, it is not an open-source software. So I am only going to test OAI LTE and srsLTE, along with OAI EPC and srsEPC in this thesis. In this thesis, I also designed and implemented different LTE/EPC

setups with different version of OAI and Ubuntu, analyzing the throughput, delay and channel quality metrics.

2. Background

2.1 LTE

2.1.1 Concept

Long Term Evolution (LTE) is a long-term evolution of the universal mobile telecommunications system (UMTS) technology standard developed by 3GPP (The 3rd Generation Partnership Project), launched and started in the 2004 December's 3GPP Toronto conference formally. The LTE system introduces key technologies such as Orthogonal Frequency Division Multiplexing (Orthogonal Frequency Division Multiplexing) and MIMO (Multi-Input & Multi-Output) and significantly increases spectral efficiency and data transmission rate (In the case of 64QAM 20M bandwidth 2X2 MIMO, the theoretical downlink maximum transmission rate is 201 Mbps, and the signaling overhead is about 150 Mbps. However, it is generally considered that the downlink peak rate is 100 Mbps and the uplink is 50 Mbps, depending on the actual networking and terminal capabilities.) Bandwidth allocation: 1.4MHz, 3MHz, 5MHz, 10MHz, 15MHz and 20MHz, etc. And it also supports the global mainstream 2G/3G frequency band and some new frequency bands. As a result, the spectrum allocation is more flexible and the system capacity and coverage are significantly improved. The network architecture of the LTE system is more flattened and simplified, reducing network node and system complexity, thereby reducing system latency and reducing network deployment and maintenance costs. LTE system supports interoperation with other 3GPP systems. Different LTE systems are divided into FDD-LTE (Frequency Division Duplexing) and TDD-LTE (Time Division Duplexing). The main difference between the two technologies lies in the physical layer of the air interface (like frame structure, time division design, synchronization, etc.). The uplink and downlink of the air interface of the FDD system receive and send data in pairs of bands, while the uplink and

downlink of the TDD system use the same frequency band to transmit in different time slots. Compared with the FDD duplex scheme, the TDD has higher spectrum utilization.

2.1.2 Architecture of LTE

The LTE system only has a packet domain. It is divided into two network elements, an Evolved Packet Core (EPC) and an Evolved Node B (evolved Node B). The EPC is responsible for the core network part, the signaling processing part is a Mobility Management Entity (MME), and the data processing part is a S-GW (Serving Gateway). The eNode B is responsible for the access network part, which is also called E-UTRAN (Evolved UTRAN), as shown in FIG.2.1. The set of radio access and core network of LTE receive the name in the specification of Evolved Packet System (EPS), which has an E-UTRAN and EPC.

The EPC block also include Home Subscriber Server (HSS), Equipment Identity Register (EIR), Charging Rules Function (PCRF), Online Charging System (OCS), Offline Charging System (OFCS).

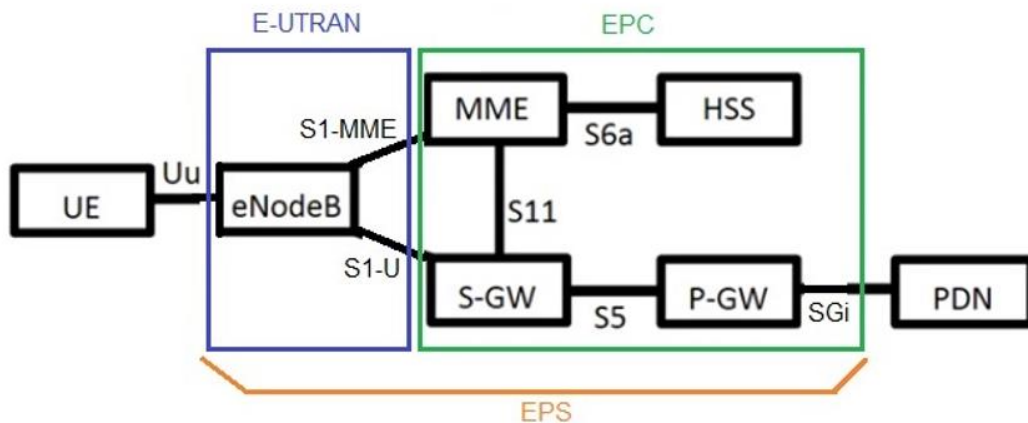


Figure 2.1 LTE Architecture

2.2 OAI

OpenAirInterface (OAI), also known as OpenAirInterface5g, is an open source SDR LTE project initiated and maintained by EURECOM in Europe. EURECOM created the OpenAirInterface (OAI) Software Alliance, a legal entity independent of EURECOM, designed to provide an ecosystem of open source, 3GPP's core (EPC) and access network (EUTRAN) protocols. According to the standard of 3GPP, OAI fully implements three parts of the core network (EPC), base station (eNB) and user (UE) of the LTE protocol. Currently, the OAI supports the function of Release 10 and keeps updating. Recently, the OAI organization is ready to build an open source SDR NB-IoT platform based on the OAI platform, which is very attractive. Cellular systems can interoperate with closed sources in any part of the network. In addition to the huge economic open source model, the alliance will be a huge industrial and academic tool. More importantly, it will ensure the communication mechanism between the two. In order to make academia closer to the complex real world, the system is mainly controlled by industrial engineers in the wireless industry. In the context of the evolution toward 5G, it is clear OAI is a prototype framework to ensure common R & D and for rapid proof-of-concept design.

OAI Gitlab website at <https://gitlab.eurecom.fr/oai/openairinterface5g>, the code and the corresponding tutorial can be seen on this site. However, at present, OAI does not have a forum for users to discuss the issue. There is only one mailing list, and everybody can ask questions and discuss them in the mailing list. OAI users would reply to the mail to help each other.

The function of OAI is very powerful. It implements the functions of the UE, eNB, and EPC full protocol stack according to the 3GPP LTE protocol. Currently, the Release 10 version is supported and is constantly updated. However, OAI platform function is too complex, resulting in its operation and configuration, to be troublesome. For example, for the correct operation of OAI EPC, you need to separately run the HSS, MME and SPGW three modules, and the configuration also need to be configured separately, Therefore, the process is a little bit cumbersome.

2.3 srsLTE

srsLTE is an open source SDR LTE platform developed by a team in Ireland called SoftwareRadio Systems. SoftwareRadioSystems is a commercial company engaged in SDR development.

The company currently includes four products: AirScope, srsLTE, srsUE, and srsENB. AirScope is a set of LTE air interface signal analyzer based on SDR. This product is commercial software and needs to be paid for, which is mostly not suitable for academia. srsLTE is a simple LTE system developed based on SDR. Only the physical layer have the downlink, and the code is open source now.

Until last year, srsLTE only has UEs and eNBs. There was no EPC function, and when srsUE and srsENB were running, they had to connect to an external EPC. So it was necessary to find a third-party EPC to work with the srsLTE. However, in 2018, srs has open-sourced their EPC module, and its configuration and use is very simple. Hence, it is no longer necessary to use a third-party EPC to build a complete LTE system using srs software.

srsUE is a set of LTE UE systems developed based on SDR. It contains layers of protocols from PHY to NAS and is currently open source. srsENB is a set of LTE eNB systems developed based on SDR, including PHY, MAC, RLC, PDCP, RRC, GTP-U and S1-AP layers. This product was commercial software and needs to be paid for in the past. But recently, the srsUE, srsENB and srsEPC are all open source. So I did some test on the srsLTE software in this thesis.

The srsLTE is subject to the 3GPP Release 10 protocol implementation. It only supports the FDD duplex mode, and supports two transmission modes of TM1 (SISO) and TM2 (Transmission Diversity)

Although srsLTE is a set of SDR LTE system, it includes two parts of eNB and UE, but downlink functions only exist in physical layer, i.e., includes PSS, SSS, PBCH, PCFICH, PHICH, PDCCH, PDSCH and other channels.

srsLTE installation and operation method is also very simple, but need to install srsGUI before installing srsLTE. srsLTE will call the srsGUI library

to display the graphical interface in real time. Specific installation and operation methods can refer to README.md on the official website of github¹.

3. The State of the Art

3.1 OAI Studies

OpenAirInterface (OAI) is a continuously updated, open source SDR LTE platform, developed and maintained by Eurecom in France. Compared with other open source SDR LTE platforms, OAI has a wide range of international application scenarios.

OAI platform is mainly written in C language. OAI has realized LTE eNBs, UEs and EPCs according to 3GPP standards and is the most complete one in the open source SDR LTE platforms. In addition, OAI platform also contains a large number of simulation platform for verification of various communication algorithms.

The following describes a variety of simulators of OAI studies.

3.1.1 Physical layer link simulation

In the OAI folder `openairinterface5g/openair1/SIMULATION/LTE_PHY/`, there are six typical physical layer link-level simulation platform.

Simulation platform	Features
dlsim	Dlsim is a simulation platform of PDSCH channels. But in fact, dlsim platform also contains the information of channels like PDCCH. (So dlsim

¹ <https://github.com/srsLTE/srsLTE>

	is more like a lightweight LTE physical layer system simulation platform.)
pbchsim	Pbchsim is a simulation platform of PBCH channel, which contains the transceiver process of PBCH channel.
pdccsim	Pdccsim is a simulation platform of PDCCH channel, which contains the transceiver process of PDCCH channel.
prachsim	Prachsim is a simulation platform of PRACH channel, which contains the transceiver process of PRACH channel.
pucchsim	Pucchsim is a simulation platform of PUCCH channel, which contains the transceiver process of PUCCH channel.
ulsim	Ulsim is a simulation platform of PUSCH channel, which contains the transceiver process of PUSCH channel.

In addition, if we want to verify or test some physical layer algorithm, we can modify or add the corresponding code directly in the corresponding simulation platform. Because the platform is written in C language, the simulation speed is very fast.

3.1.2 LTE system-level simulation

OAI in the folder `openairinterface5g/targets/SIMU/USER/` has a LTE system-level simulation platform `oaisim`. As `oaisim` involves all layers of LTE, various process code, it is suitable for advanced LTE learning. If you want to validate some of the LTE system-level algorithms or features, you can add the appropriate code modules on the `oaisim` platform.

3.1.3 SDR-based LTE system

OAI implements the LTE eNB, the UE and the EPC according to the 3GPP protocol. From a commercial point of view, LTE eNBs and EPCs are

more important. From the scientific point of view, LTE UEs are equally important. OAI is paying more attention to the development and maintenance of the eNB and the EPC. The eNB and the EPC are relatively stable. However, the UE only has the corresponding function and is very unstable.

OAI eNB:

OAI's eNBs now support Release 10 features and are relatively stable. The system has been branched. Holding 5MHz, 10MHz bandwidth, 20MHz bandwidth, support FDD, TDD two modes.

OAI eNB mainly contains three threads, one for sending data, one for receiving data, one for interacting with hardware devices like the USRP.

OAI EPC:

OAI's EPC is basically ready for use today. However, personally I feel a little difficult to use it. There are too many parameters that need to be configured.

OAI UE:

The OAI UE has the function of Release 10. But since the OAI organization emphasize on the OAI eNB and OAI EPC, the organization has fewer test tables for OAI UEs, resulting in very unstable OAI UEs and more system bugs. Based on OAI's complete set of SDR LTE system, we have a lot of application scenarios.

For example,

Wireless Security Research:

The traditional cellular network security research is mainly based on the OpenBTS system. If we want to study the security of LTE network, we can base on OAI's SDR LTE system.

New technology communication test:

By adding the appropriate code modules on the OAI system, we can validate various new communication technologies such as NOMA, MIMO, etc.

Professional network :

In some specific scenarios, we need to build an LTE local area network, and the OAI platform can provide eNBs and EPCs for the LTE system.

NB-IoT:

Recently, the Internet of Things is very popular. OAI organizations are also ready to add NB-IoT capabilities on the OAI platform.

3.3 srsLTE Studies

3.3.1 Main Feature of srsLTE

Compatible with LTE Release 10;

FDD configuration: Test bandwidth: 1.4, 3, 5, 10, 15 and 20 MHz;

Transmission modes 1 (single antenna) and 2 (transmission diversity);

UE's cell search and synchronization process;

UE and eNodeB support all DL channels/signals: PSS, SSS, PBCH, PCFICH, PHICH, PDCCH, PDSCH;

The UE supports all UL channels/signals: PRACH, PUSCH, PUCCH, SRS;

Frequency-based ZF and MMSE equalizers;

Highly optimized turbo decoder for Intel SSE4.1/AVX (+100 Mbps) and C standard (+25Mbps);

The MATLAB and OCTAVE MEX libraries generate many components;

The UE receiver tests and verifies the Amarisoft LTE 100 eNodeB and the commercial LTE network (Ireland's Telefonica Spain, Three.ie and Eircom).

Deleted features:

Closed-loop power control

Semi-Persistent scheduling

3.3.2 Test environment of srsLTE

srsLTE runs on Ubuntu system. It was successfully installed on both the 14.04 and 17.04 systems. Therefore, the Ubuntu system can be selected from 14.04 or higher. If the computer is configured, it is best to use the i7 CPU. The higher the frequency, the better. The installation of the system is recommended to use a U disk to create a boot disk, and then you can install it.

The library currently supports Ettus Universal Hardware Driver (UHD) and bladeRF drivers. In addition, any hardware suitable for UHD or bladeRF drivers is also supported. There is no sample rate conversion, so the hardware should support a sampling rate of 30.72 MHz in order to keep the LTE sampling frequency and the decoded signal in LTE base stations working properly.

SRS have tested the following hardware:

USRP B210

USRP X300

bladeRF

3.3.3 srsLTE's application

srsLTE's downlink only exists in the physical layer of LTE, therefore the application scenario is limited:

1. The LTE signal analyzer in srsUE can use the UE in srsLTE to receive the signal of the commercial base station and demodulate the system information of the commercial base station.
2. Physical layer algorithm verification and improvement

After porting the algorithm to srsLTE, the signal over the air interface can verify the algorithm performance more realistically.

3. The srsLTE provides many function libraries commonly used in LTE communications. The srsLTE library can be used to design certain LTE for different purposes.
4. Building LTE Demo.

The LTE Demo can be demonstrated in the laboratory using the function of srsLTE and some hardware like the USRP.

4. Implementation

4.1 LTE Link on OAI

4.1.1 Hardware verification

It is required at least Intel Core i5-6600 CPU @ 3.30GHz × 4 to implement a User Equipment (UE) or an eNB.

Our project test USRP B210 and a USRP X310. B210 required a free USB 3.0 port in PC to operate so that we can support RF. (USB 2.0 is tested and it is infeasible.) The power was supported by the USB. And then we change our devices into USRP X310, which required an ethernet interface in PC. The ethernet interface of the PC must have a speed faster than 1024MB/S. In addition, since the power consumption of X310 is very high, it required an independent power cable to provide the electricity.

The PC I used in this thesis is a Ubuntu 14.04 machine with a CPU of i7 7700 and a Ubuntu 16.04 machine with a CPU of i9 7900k.



Figure 4.1 USRP B210



Figure 4.2 USRP X310

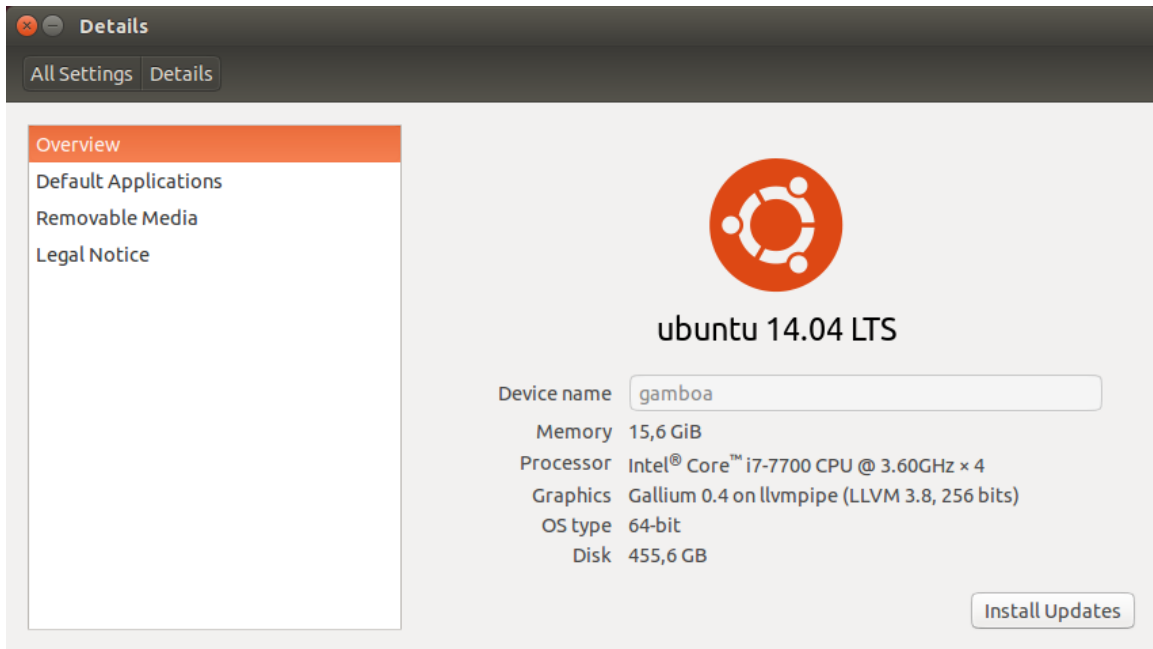


Figure 4.3 System and Hardware

4.1.2 Operating system verification

PCs should have installed Ubuntu LTS 14.04.3 (64bits) with low latency Kernel version 3.19,

if kernel installed is of a different version it can be updated with:

```
$ sudo apt-get install linux-image-3.19.0-61-lowlatency linux-headers-3.19.0-61-lowlatency
```

In order to change into the new kernel, we have to reboot the computer, and to confirm it is typed:

```
$ uname -a
```

```
Linux "NAME" 3.19.0-61-lowlatency #69~14.04.1-Ubuntu SMP PREEMPT Thu Jun 9 10:15:00 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux.
```

And the branch of OAI should be tag v0.3.2 under this circumstance.

We can also use Ubuntu LTS 16.04.3 (64bits) with low latency Kernel version 3.8.

In this case, we should use the develop branch of OAI (\geq tag v0.4.0).

Due to real time communications, it is necessary to disable power management features in the BIOS as p-states, c-states and also disable CPU frequency scaling.

CPU frequency should be constant, so cpufrequtils is required. In file: /etc/default/cpufrequtils a new parameter will be added:

```
$ sudo apt-get install cpufrequtils
```

```
$ sudo vi /etc/default/cpufrequtils
```

And add the following line to it:

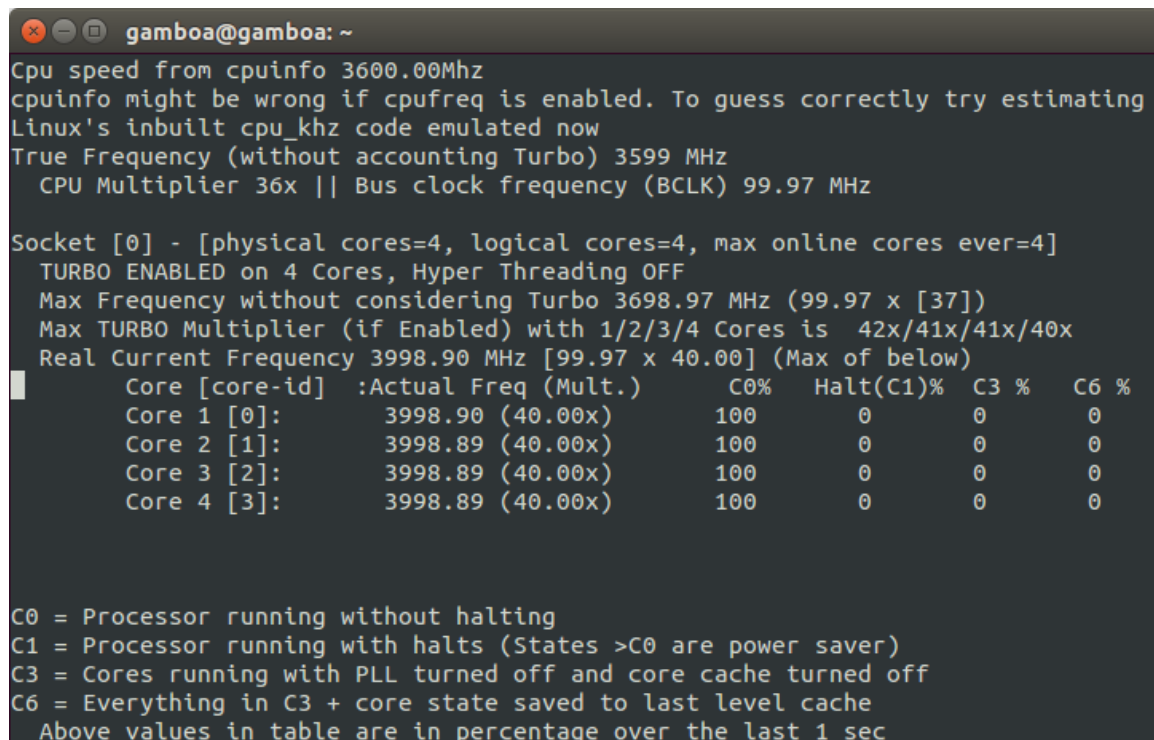
```
GOVERNOR="performance"
```

Finally, we need to disable ondemand daemon so that when the next time that PC be turned on, the settings would not be overwritten.

```
$ sudo update-rc.d ondemand disable
```

Once power management features have been disabled, i7z tool is used to verify that CPU does not change its frequency, and just C0 state remains available. Figure 4.5 shows the result obtained with i7z tool.

Enter `$ sudo i7z` in the terminal, it will show an output similar to: (the C0% should be 100%)



```
gamboa@gamboa: ~
Cpu speed from cpufreq 3600.00Mhz
cpufreq might be wrong if cpufreq is enabled. To guess correctly try estimating
Linux's inbuilt cpu_khz code emulated now
True Frequency (without accounting Turbo) 3599 MHz
CPU Multiplier 36x || Bus clock frequency (BCLK) 99.97 MHz

Socket [0] - [physical cores=4, logical cores=4, max online cores ever=4]
TURBO ENABLED on 4 Cores, Hyper Threading OFF
Max Frequency without considering Turbo 3698.97 MHz (99.97 x [37])
Max TURBO Multiplier (if Enabled) with 1/2/3/4 Cores is 42x/41x/41x/40x
Real Current Frequency 3998.90 MHz [99.97 x 40.00] (Max of below)
Core [core-id] :Actual Freq (Mult.)      C0%  Halt(C1)%  C3 %  C6 %
Core 1 [0]:      3998.90 (40.00x)          100    0         0     0
Core 2 [1]:      3998.89 (40.00x)          100    0         0     0
Core 3 [2]:      3998.89 (40.00x)          100    0         0     0
Core 4 [3]:      3998.89 (40.00x)          100    0         0     0

C0 = Processor running without halting
C1 = Processor running with halts (States >C0 are power saver)
C3 = Cores running with PLL turned off and core cache turned off
C6 = Everything in C3 + core state saved to last level cache
Above values in table are in percentage over the last 1 sec
```

Figure 4.4 Power management features disabled and CPU information

4.1.3 Get repository

In order to obtain the repository for UE/eNB, git must be installed.

```
$ sudo apt-get update
```

```
$ sudo apt-get install subversion git
```

There are different repositories posted on EURECOM gitlab. Here we use the openairinterface5g, because it contains source code for UE/eNB RAN.

To install the openairinterface5g, we need to input following sentence in the terminal.

```
$ cd ~/
```

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

Finally, it is necessary to run once and only once to install missing packages.

```
$ cd ~/openairinterface5g/
```

```
$ cd cmake_targets
```

```
$ ./build_oai -I -w USRP
```

In this thesis, we use usrp, so we have to type USRP in the sentence to make the openairinterface5g support the USRP.

4.1.4 Connect OAI eNB with COTS UE

To connect the OAI eNB with UE, there are mainly three ways to achieve your goal, connecting OAI eNB with OAI UE, connecting OAI eNB with COTS (Commercial Off-The-Shelf) UE and connecting OAI eNB with mobile phones.

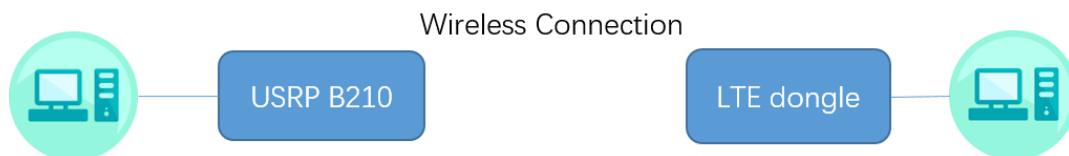


Figure 4.5 Basic structure of COTS connection

Here I use the COTS as an example. The COTS I use here is LTE Dongle: Huawei E3372.



Figure 4.6 LTE Dongle: Huawei E3372

Unlike the OAI UEs, the COTS require the S1 interface and the EPC to work. Therefore, here the core network part of the OAI is required.

Since I only use one computer for the tests, I use a virtual machine to work as the EPC. The virtual machine software I used is VMware workstation 14 and the setup is shown in the following figure.

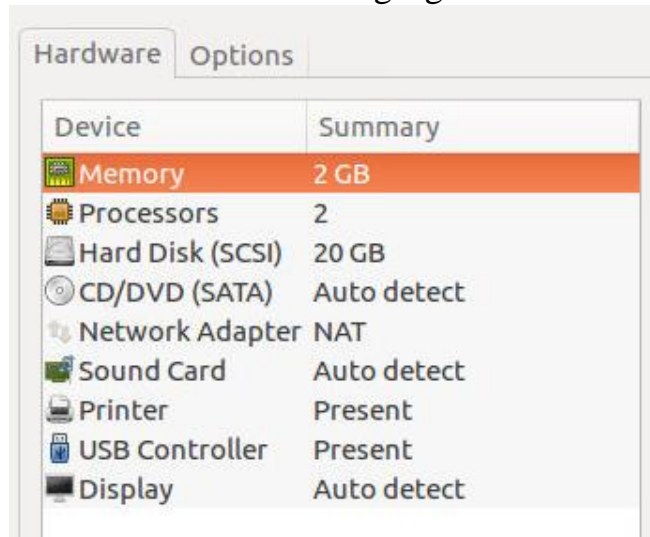


Figure 4.7 Basic setup of the virtual machine

By using the command(in virtual machine):

```
git clone https://gitlab.eurecom.fr/oai/openair-cn.git
```

can build the core network which is stored in a folder called openair-cn.

And then run the automated script for openair-cn:

git checkout v0.3.2

./build_epc -i #(Need to run only once to install missing packages)

./build_hss -i #(Need to run only once to install missing packages)

When we were building the hss, it would automatically install the SQL server. However, the installation of SQL server would fail if we only use the command on the tutorials of OAI. It showed

error2002 : can't connect to local mysql server through socket.

After checking, it was found that mysql-server was not automatically installed. In order to avoid subsequent errors in the database import, I reinstalled the system and kernel, manually installed mysql-server before performing this step, and solved the problem.

Therefore, we need to install the SQL (*Sudo apt-get install mysql-server*) before build the hss to make this succeed.

And then, after entering 127.0.0.1/phpmyadmin in the browser, the “not found interface” appears.

I solved this by adding phpmyadmin in Apache configuration

Fisrt open the apache.conf file:

Vim /etc/apache2/apache2.conf

Then add the following statement:

Include /etc/phpmyadmin/apache.conf

Restart the apache service:

/etc/init.d/apache2 restart

And we can enter the SQL server after the above has been done.

Here is the details of my setup. The ip address of different interfaces can be seen by entering the command “ifconfig”.

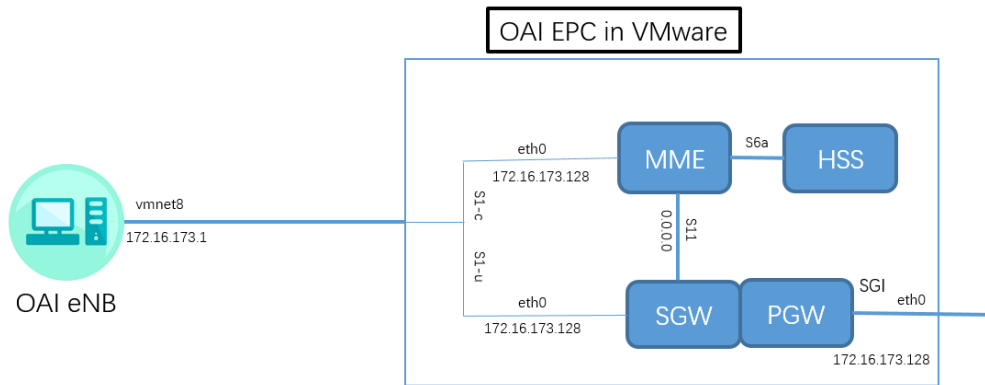


Figure 4.8 Detailed LTE setup with Ethernet interfaces and IP addresses

First use “ifconfig” in the eNB(physical machine) to see the IP address of different interface.

```

gamboa@gamboa: ~
inet addr:192.168.122.1 Bcast:192.168.122.255 Mask:255.255.255.0
UP BROADCAST MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

vmnet1  Link encap:Ethernet HWaddr 00:50:56:c0:00:01
        inet addr:172.16.179.1 Bcast:172.16.179.255 Mask:255.255.255.0
        inet6 addr: fe80::250:56ff:fec0:1/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:51 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

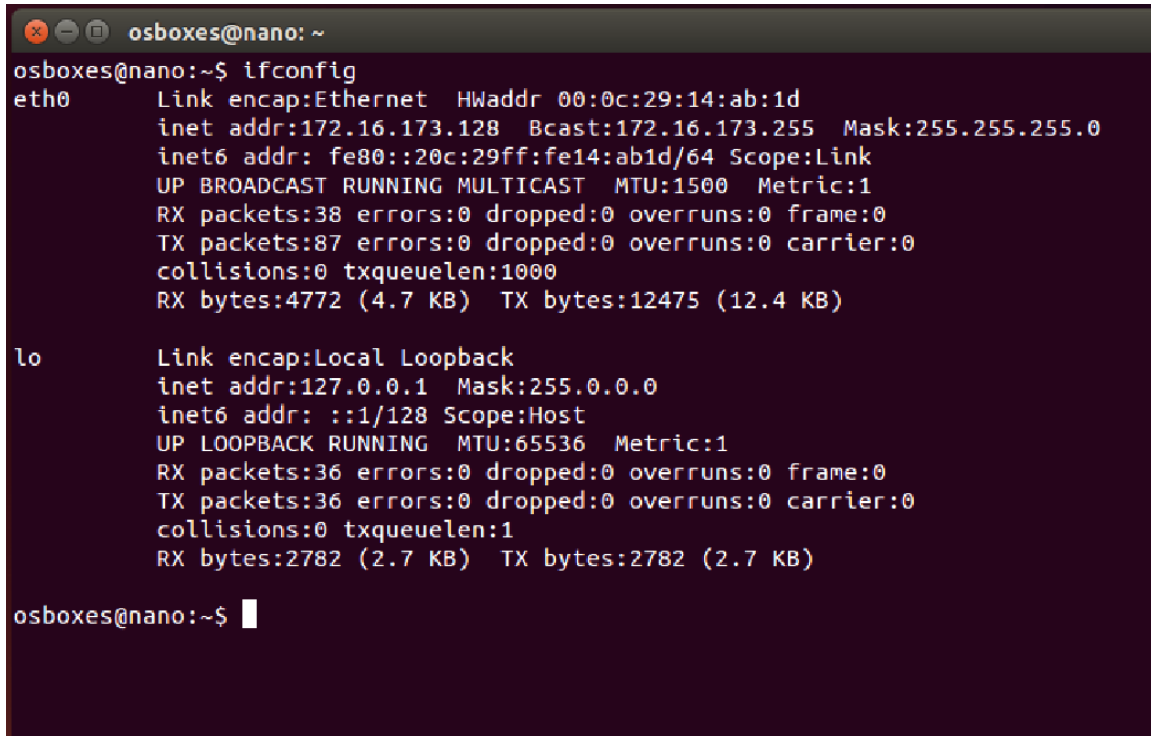
vmnet8  Link encap:Ethernet HWaddr 00:50:56:c0:00:08
        inet addr:172.16.173.1 Bcast:172.16.173.255 Mask:255.255.255.0
        inet6 addr: fe80::250:56ff:fec0:8/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:52 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

```

Figure 4.9 Information of the Interface in eNB

Since the virtual machine I used worked in NAT mode, so the interface which connects the virtual machine is vmnet8. So this interface is the interface of eNB for EPC.

And then do the same command in the EPC side(virtual machine) to check the ip address of the MME. In this case the interface is called eth0.



```
osboxes@nano: ~
osboxes@nano:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:14:ab:1d
          inet addr:172.16.173.128  Bcast:172.16.173.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe14:ab1d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:38 errors:0 dropped:0 overruns:0 frame:0
          TX packets:87 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4772 (4.7 KB)  TX bytes:12475 (12.4 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:36 errors:0 dropped:0 overruns:0 frame:0
          TX packets:36 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:2782 (2.7 KB)  TX bytes:2782 (2.7 KB)

osboxes@nano:~$
```

Figure 4.10 Information of the Interface in EPC(The virtual machine)

In configuration file(`~/openairinterface5g/targets/PROJECTS/GENERIC-LTE-EPC/CONF/enb.band7.tm1.usrb210.conf`), we change the IP address to the IP address of the interface, which we read in the terminal.

So we change the original file into:

```
//////// MME parameters:
```

```
mme_ip_address = ( {ipv4 = "172.16.173.128";
```

```
ipv6="192:168:30::17";
```

```
active="yes";
```

NETWORK_INTERFACES :

{

ENB_INTERFACE_NAME_FOR_S1_MME = "vmnet8";

ENB_IPV4_ADDRESS_FOR_S1_MME = "172.16.173.1";

ENB_INTERFACE_NAME_FOR_S1U = "vmnet8";

ENB_IPV4_ADDRESS_FOR_S1U = "172.16.173.1";

In EPC configuration file (~/.openair-cn/BUILD/EPC/epc.conf.in),

As I mentioned before, we would have to change it into:

NETWORK_INTERFACES :

{

*MME_INTERFACE_NAME_FOR_S1_MME = "eth0"; # YOUR NETWORK CONFIG
HERE*

*MME_IPV4_ADDRESS_FOR_S1_MME = "172.16.173.128/24"; # YOUR NETWORK
CONFIG HERE*

*SGW_INTERFACE_NAME_FOR_S1U_S12_S4_UP = "eth0"; # YOUR NETWORK
CONFIG HERE*

*SGW_IPV4_ADDRESS_FOR_S1U_S12_S4_UP = "172.16.173.128/24"; # YOUR
NETWORK CONFIG HERE*

PGW_INTERFACE_NAME_FOR_SGI = "eth0"; # YOUR NETWORK CONFIG HERE

*PGW_IPV4_ADDRESS_FOR_SGI = "172.16.173.128/24"; # YOUR NETWORK
CONFIG HERE*

PGW_MASQUERADE_SGI = "yes"; # YOUR NETWORK CONFIG HERE

And then, compile and run the EPC and the HSS by entering the following command:

Compile & Run EPC:

```
cd ~/openair-cn
```

```
cd SCRIPTS
```

```
./build_epc -c -l
```

```
./run_epc -i -r
```

Compile & Run HSS:

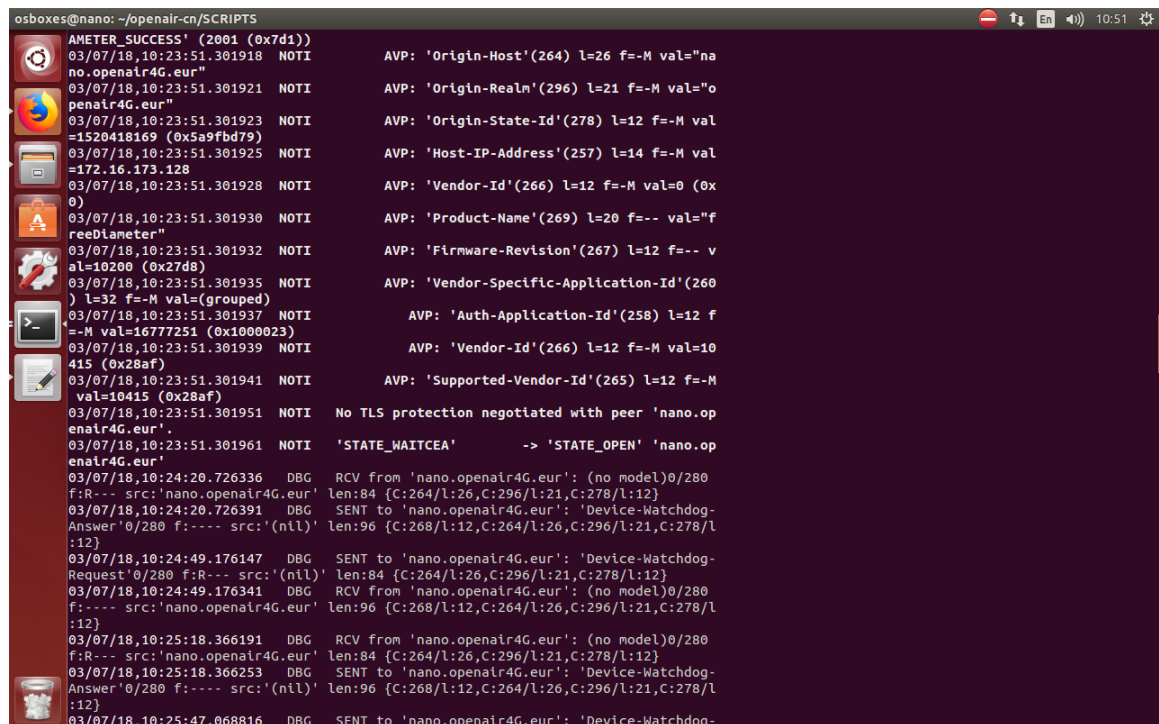
```
cd ~/openair-cn
```

```
cd SCRIPTS
```

```
./build_hss -c -l
```

```
./run_hss
```

You should have seen the following words in the terminal when you are running the EPC and the HSS if you succeed.



```
osboxes@nano: ~/openair-cn/SCRIPTS
AMETER_SUCCESS' (2001 (0x7d1))
03/07/18,10:23:51.301918 NOTI AVP: 'Origin-Host'(264) l=26 f=-M val="na
no.openair4G.eur"
03/07/18,10:23:51.301921 NOTI AVP: 'Origin-Realn'(296) l=21 f=-M val="o
penair4G.eur"
03/07/18,10:23:51.301923 NOTI AVP: 'Origin-State-Id'(278) l=12 f=-M val
=1520418169 (0x5a9fbd79)
03/07/18,10:23:51.301925 NOTI AVP: 'Host-IP-Address'(257) l=14 f=-M val
=172.16.173.128
03/07/18,10:23:51.301928 NOTI AVP: 'Vendor-Id'(266) l=12 f=-M val=0 (0x
0)
03/07/18,10:23:51.301930 NOTI AVP: 'Product-Name'(269) l=20 f=- val="f
reediameter"
03/07/18,10:23:51.301932 NOTI AVP: 'Firmware-Revision'(267) l=12 f=- v
al=10200 (0x27d8)
03/07/18,10:23:51.301935 NOTI AVP: 'Vendor-Specific-Application-Id'(260
) l=32 f=-M val=(grouped)
03/07/18,10:23:51.301937 NOTI AVP: 'Auth-Application-Id'(258) l=12 f
=-M val=16777251 (0x1000023)
03/07/18,10:23:51.301939 NOTI AVP: 'Vendor-Id'(266) l=12 f=-M val=10
415 (0x28af)
03/07/18,10:23:51.301941 NOTI AVP: 'Supported-Vendor-Id'(265) l=12 f=-M
val=10415 (0x28af)
03/07/18,10:23:51.301951 NOTI No TLS protection negotiated with peer 'nano.op
enair4G.eur'.
03/07/18,10:23:51.301961 NOTI 'STATE_WAITCEA' -> 'STATE_OPEN' 'nano.op
enair4G.eur'
03/07/18,10:24:20.726336 DBG RCV from 'nano.openair4G.eur': (no model)0/280
f:R-- src:'nano.openair4G.eur' len:84 {C:264/l:26,C:296/l:21,C:278/l:12}
03/07/18,10:24:20.726391 DBG SENT to 'nano.openair4G.eur': 'Device-Watchdog-
Answer'0/280 f:--- src:'(nil)' len:96 {C:268/l:12,C:264/l:26,C:296/l:21,C:278/l
:12}
03/07/18,10:24:49.176147 DBG SENT to 'nano.openair4G.eur': 'Device-Watchdog-
Request'0/280 f:R-- src:'(nil)' len:84 {C:264/l:26,C:296/l:21,C:278/l:12}
03/07/18,10:24:49.176341 DBG RCV from 'nano.openair4G.eur': (no model)0/280
f:--- src:'nano.openair4G.eur' len:96 {C:268/l:12,C:264/l:26,C:296/l:21,C:278/l
:12}
03/07/18,10:25:18.366191 DBG RCV from 'nano.openair4G.eur': (no model)0/280
f:R-- src:'nano.openair4G.eur' len:84 {C:264/l:26,C:296/l:21,C:278/l:12}
03/07/18,10:25:18.366253 DBG SENT to 'nano.openair4G.eur': 'Device-Watchdog-
Answer'0/280 f:--- src:'(nil)' len:96 {C:268/l:12,C:264/l:26,C:296/l:21,C:278/l
:12}
03/07/18,10:25:47.068816 DBG SENT to 'nano.openair4G.eur': 'Device-Watchdog-
```

```
osboxes@nano: ~/openair-cn/SCRIPTS
075 'STATE_OPEN' <-- 'FDEVP_CN_XMSG_RECV' (0x7f8764000910,96) '
nano.openair4G.eur'
001378 01476:018964 7F8781FFB700 ALERT S6A es/openair-cn/SRC/S6A/s6a_task.c:0
075 RCV from 'nano.openair4G.eur': (no model)0/280 f:---- src:'nano.openair4G
.eur' len:96 [C:268/l:12,C:264/l:26,C:296/l:21,C:278/l:12]
001379 01476:018975 7F8781FFB700 ALERT S6A es/openair-cn/SRC/S6A/s6a_task.c:0
075 Iterating on rules of COMMAND: 'Device-Watchdog-Answer'.
001380 01476:018970 7F8781FFB700 ALERT S6A es/openair-cn/SRC/S6A/s6a_task.c:0
075 Peer timeout reset to 30 seconds (+/- 2)
001381 01476:018981 7F8781FFB700 ALERT S6A es/openair-cn/SRC/S6A/s6a_task.c:0
075 'nano.openair4G.eur' in state 'STATE_OPEN' waiting for next event.
001382 01480:766800 7F87AB7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0
036 ===== Statistics =====
001383 01480:766806 7F87AB7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0
037 Global Since last display |
001384 01480:766807 7F87AB7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0
038 UE | | |
001385 01480:766809 7F87AB7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0
039 Bearers | | |
001386 01490:766800 7F87AB7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0
036 ===== Statistics =====
001387 01490:766806 7F87AB7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0
037 Global Since last display |
001388 01490:766807 7F87AB7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0
038 UE | | |
001389 01490:766808 7F87AB7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0
039 Bearers | | |
001390 01500:766798 7F87AB7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0
036 ===== Statistics =====
001391 01500:766806 7F87AB7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0
037 Global Since last display |
001392 01500:766807 7F87AB7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0
038 UE | | |
001393 01500:766808 7F87AB7FE700 DEBUG MME-AP SRC/MME_APP/mme_app_statistics.c:0
039 Bearers | | |
001394 01507:482554 7F8781FFB700 ALERT S6A es/openair-cn/SRC/S6A/s6a_task.c:0
075 'STATE_OPEN' <-- 'FDEVP_PSM_TIMEOUT' ((nil),0) 'nano.openair4G.
eur'
001395 01507:482585 7F8781FFB700 ALERT S6A es/openair-cn/SRC/S6A/s6a_task.c:0
075 Peer timeout reset to 30 seconds
001396 01507:482587 7F8781FFB700 ALERT S6A es/openair-cn/SRC/S6A/s6a_task.c:0
075 'nano.openair4G.eur' in state 'STATE_OPEN' waiting for next event.
001397 01507:486559 7F87817FA700 ALERT S6A es/openair-cn/SRC/S6A/s6a_task.c:0
```

Figure 4.10 Both HSS and EPC show “state open”

IN the meantime, the SQL server is required to connect the eNB with the LTE dongle. We need to change a few parameters.

Parameters to be changed in `_hss.conf_`:

MYSQL_server `_127.0.0.1_`;

MYSQL_user `_root_`;

MYSQL_pass `_linux_`;

MYSQL_db `_oai_db_`;

OPERATOR_key `_1111111111111111111111111111111111111_`;

Table 4.4: Parameters to be changed in `_hss_fd.conf_`:

Identity `_hss.openair4G.eur_`;

Realm `_openair4G.eur_`;

Then we need to enter the sim card's information in the SQL server so that the eNB can recognize the LTE dongle. The sim card's information can be read by a card reader like SCR 3310.

Here we have the sim card information like this:

MCC (Mobile Country Code)	214
MNC (Mobile Network Code)	03
TAC (Tracking Area Code)	4
IMSI (International Mobile Subscriber Identity)	214030000000004
OP (Operator Key)	11111111111111111111111111111111 (32 digits)

Table 4.1: UE SIM card con_guration

The most important information of the sim card to be authenticated is IMSI, OP and Ki. In this case, the Ki is 8BAF473F2F8FD09487CCBD7097C6862 (32 digits) and the Opc can be calculated by the HSS through Ki and OP.

If the sim card's information is uncertain, we should use a card reader to check the detailed information of the sim card. In this thesis, I use SCR3310.



Figure 4.11 Card reader: SCR3310

To read the information of the sim card, we have to get the code of PySIM:

```
git clone git://git.osmocom.org/pysim pysim
cd pysim
```

and then run the `/pySim-read.py` to read your card:

```
./pySim-read.py
```

if you done everything allright, you will see something similar:

```
Reading ...
ICCID: 8901901550000123456
IMSI: 2014030000000004
KI: 8BAF473F2F8FD09487CCBD7097C6862
SMSP: ffffffffffffffffffffffffffffffffffffffffffffffffff069186770700f9ffffffffffff
fffff
ACC: None
MSISDN: Not available
Done !
```

Figure 4.12 Card information successfully read

All the information of the sim card must be written into the database of HSS. When I was working on the Ubuntu 14.04, inserting a new user in oai database can do the job. All the work can be done in visual interface of phpadmin.

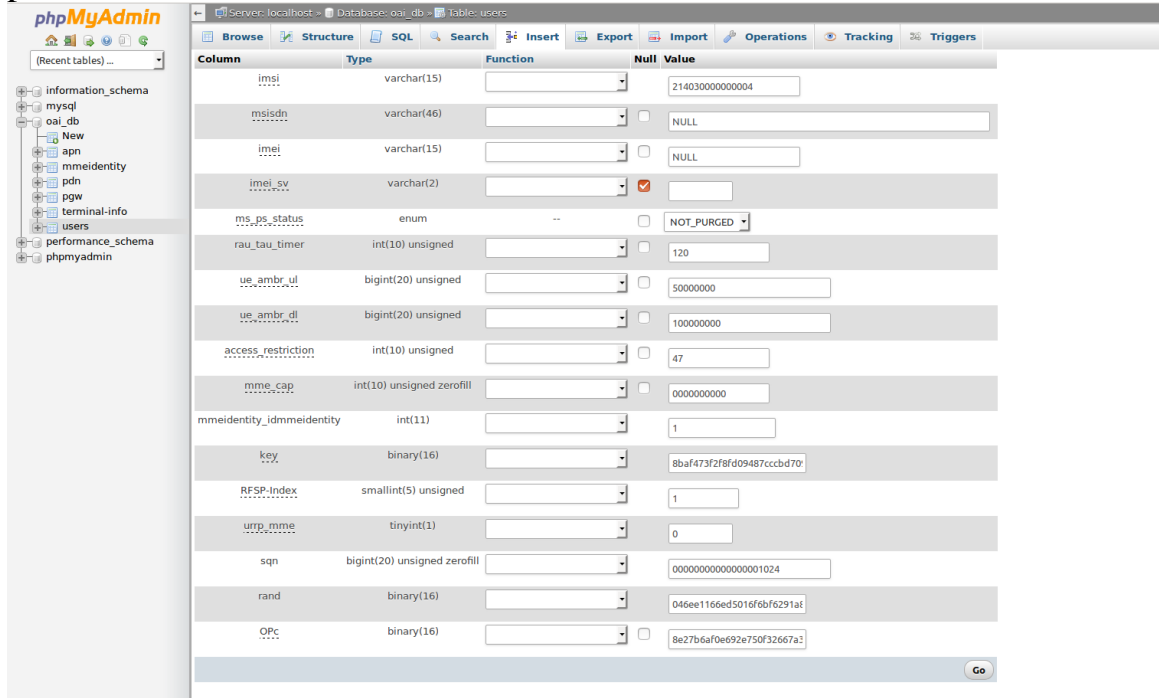


Figure 4.13 sim card's information added in the database

For this LTE Dongle at the UE side, we can enter the graphic interface on the url 192.168.8.1. And then we have to go to Profile Management and add a new APN profile.

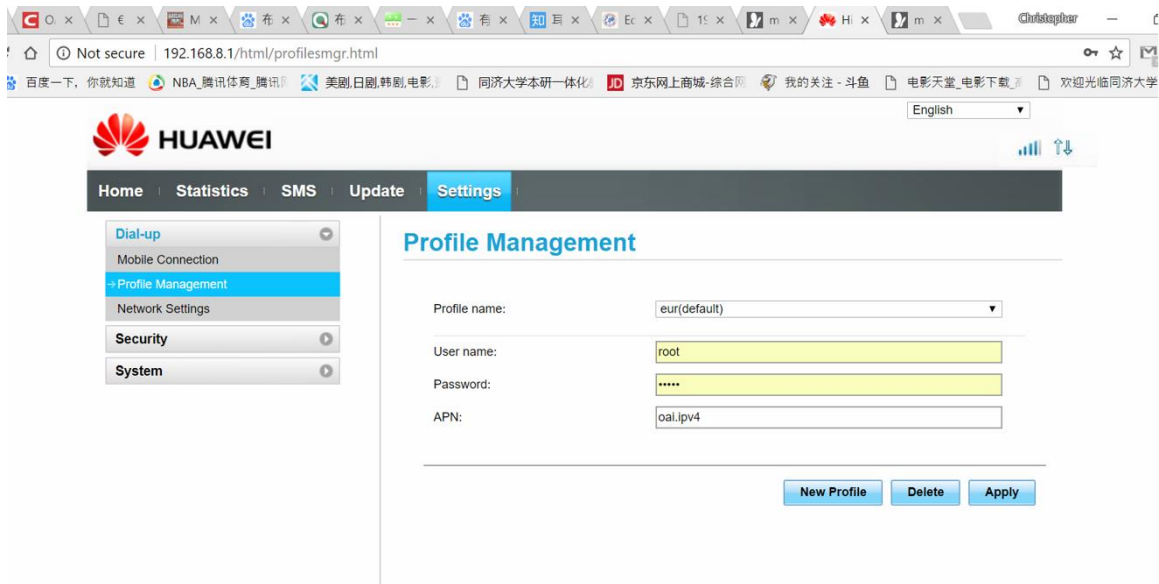


Figure 4.14 Parameters of APN

And the successful connection to the LTE network is shown in Figure 4.13.

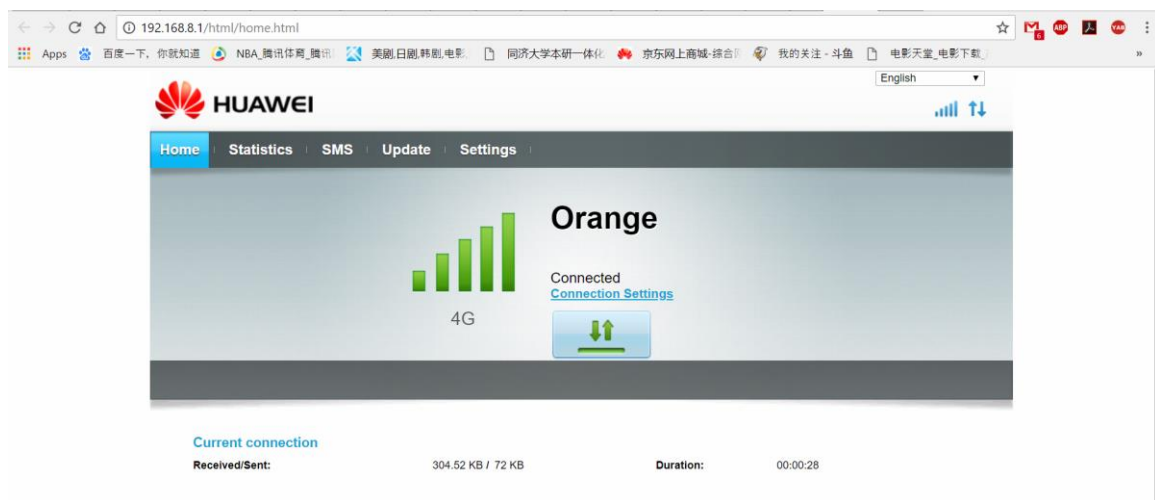


Figure 4.15 Successful connection of the LTE Dongle

And in the eNB machine, the xform will show something like the following figure. And stats like CQI will also be shown in the eNB machine.

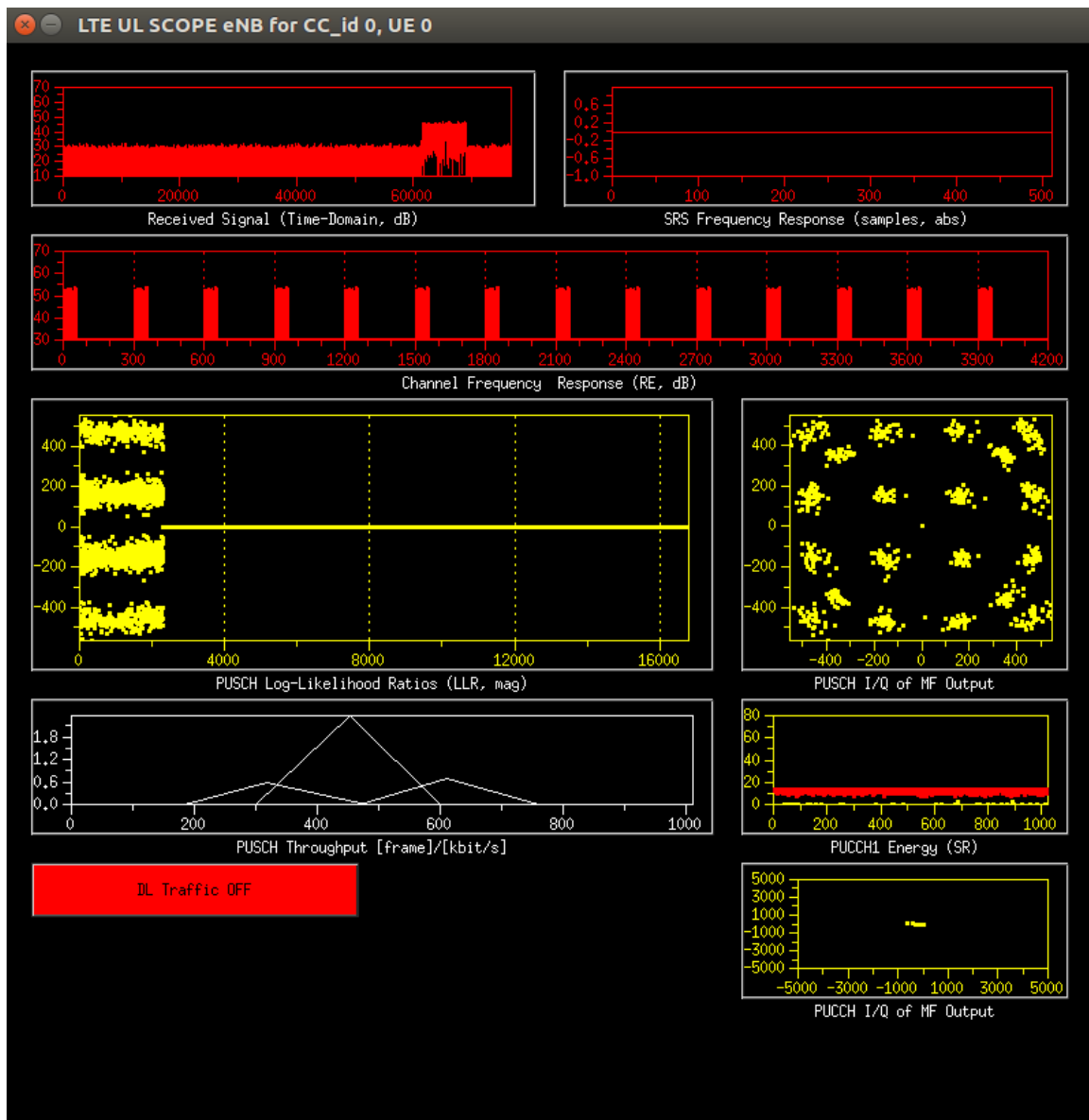


Figure 4.16 LTE uplink scope

```

stats
Reset Stats

eNB 0/1 Frame 288: RX Gain 125 dB, IO -101 dBm (24.0) dB
PRB 10 (0,0,0,0): -115 -115 -112 -115 -115 -115 -115 -112 -117 -115 -115 -110 -115 -115 -115 -110 -117 -115 -115 -115 -112 -115

PERFORMANCE PARAMETERS
Total DLSCH 3 kbits / 289 frames Total DLSCH throughput 0 kbps Total DLSCH trans 24 / 289 frames
UE 0 (7121) Power: (48,0) dB, Po_PUSCH: (-91,0) dBm, Po_PUCCH (-77/-96) dBm, PUCCH1 Thres -115 dBm
DL mcs 0, UL mcs 4, UL rb 3, delta_TF -282, Wideband CQI: (0,0) dB
DL TH 1, DL_cqi 14, DL_pmi_single 0 Timing advance 24 samples (6 16Ts), update 4 Mode = PUSCH(3) UE_id_mac = 0, RRC status = 4
SR received/total: 15/2665 (diff 8651)
DL Subband CQI: 14 14 14 14 13 13
ULSCH errors/attempts per harq (per round):
  harq 0: 0/158 (fer 0) (0/158, 0/0, 0/0, 0/0)   harq 1: 0/298 (fer 0) (0/298, 0/0, 0/0, 0/0)
  harq 2: 0/459 (fer 0) (0/459, 0/0, 0/0, 0/0)   harq 3: 1/720 (fer 0) (1/720, 1/1, 1/1, 1/1)
  harq 4: 0/156 (fer 0) (0/156, 0/0, 0/0, 0/0)   harq 5: 0/295 (fer 0) (0/295, 0/0, 0/0, 0/0)
  harq 6: 1/461 (fer 0) (2/461, 1/2, 1/1, 1/1)   harq 7: 0/720 (fer 0) (0/720, 0/0, 0/0, 0/0)
ULSCH errors/attempts total 2/3267 (3/3267, 2/3, 2/2, 2/2)
DLSCH errors/attempts per harq (per round):
  harq 0: 0/5 (5/0/5, 0/0/0, 0/0/0, 0/0/0)   harq 1: 0/1 (1/0/1, 0/0/0, 0/0/0, 0/0/0)
  harq 2: 0/5 (5/0/5, 0/0/0, 0/0/0, 0/0/0)   harq 3: 0/1 (1/0/1, 0/0/0, 0/0/0, 0/0/0)
  harq 4: 0/4 (4/0/4, 0/0/0, 0/0/0, 0/0/0)   harq 5: 0/2 (2/0/2, 0/0/0, 0/0/0, 0/0/0)
  harq 6: 0/5 (5/0/5, 0/0/0, 0/0/0, 0/0/0)   harq 7: 0/0 (0/0/0, 0/0/0, 0/0/0, 0/0/0)
DLSCH errors/attempts total 0/24 (0/24, 0/0, 0/0, 0/0):
DLSCH total bits from MAC: 0kbit DLSCH total bits ack'ed: 3kbit DLSCH Average throughput (100 frames): 0kbps

EOF

```

Figure 4.17 general stats and CQI values

```

l2 stats
Reset Stats

eNB 0 CC 0 Frame 804: Active UEs 1, Available PRBs 25, nCCE 0, Scheduling decisions 0, Missed Deadlines 0
BCCH , NB_TX_MAC = 49784, transmitted bytes (TTI 15, total 896115) MCS (TTI 2)
DLSCH bitrate (TTI 0, avg 0) kbps, Transmitted bytes (TTI 0, total 0), Transmitted PDU (TTI 0, total 0)
ULSCH bitrate (TTI 0, avg 1573) kbps, Received bytes (TTI 63, total 1583278), Received PDU (TTI 0, total 24856)

[MAC] UE 0 (DLSCH),status RRC_RECONFIGURED, RNTI 7121 : CQI 13, MCS1 25, MCS2 0, RB (tx 2, retx 2, total 162), nCCE (tx 0, retx 0)
[MAC] DLSCH bitrate (TTI 0, avg 0), Transmitted bytes (TTI 4, total 531), Total Transmitted PDU 73, Overhead (TTI 2, total 942, avg 0)
[MAC] UE 0 (ULSCH), Status RRC_RECONFIGURED, Failure timer 0, RNTI 7121 : rx power (normalized -90, target -90), MCS (pre 11, post 11), RB (rx 0, retx 0, total 7479)
[MAC] DLSCH bitrate (TTI 0, avg 1573), received bytes (total 1583278), Total received PDU 24856, Total errors 0
[MAC] Received PHR PH = 40 (db)
[MAC] Received BSR LCGID[0][1][2][3] = 0 0 0 0
[RLC] DCCH Mode AM, NB_SDU_TO_TX = 5 (bytes 219)   NB_SDU_TO_TX_DISC 0 (bytes 0)
[RLC] DCCH Mode AM, NB_TX_DATA = 12 (bytes 462)   NB_TX_CONTROL 6 (bytes 12)   NB_TX_RETX 0 (bytes 0)   NB_TX_RETX_BY_STATUS = 0 (bytes 0)
[RLC] DCCH Mode AM, NB_RX_DATA = 7 (bytes 207)   NB_RX_CONTROL 5 (bytes 10)   NB_RX_DUPL 0 (bytes 0)   NB_RX_DROP = 0 (bytes 0)   NB_RX_OUT_OF_WINDOW =
[RLC] DCCH Mode AM, RX_REORDERING_TIMEOUT = 0   RX_POLL_RET_TIMEOUT 0   RX_PROHIBIT_TIME_OUT 0
[RLC] DCCH Mode AM, NB_SDU_TO_RX = 6 (bytes 193)
[RLC] DTX Mode UM, NB_SDU_TO_TX = 1634882592 (bytes 1768790654)   NB_SDU_TO_TX_DISC 1684370548 (bytes 1954112032)
[RLC] DTX Mode UM, NB_TX_DATA = 673215333 (bytes 541676628)   NB_TX_CONTROL 0 (bytes 0)   NB_TX_RETX 0 (bytes 0)   NB_TX_RETX_BY_STATUS = 0 (bytes 0)
[RLC] DTX Mode UM, NB_RX_DATA = 1768790654 (bytes 1684370548)   NB_RX_CONTROL 5 (bytes 10)   NB_RX_DUPL 740896800 (bytes 1634882592)   NB_RX_DROP = 143054032
[RLC] DTX Mode UM, RX_REORDERING_TIMEOUT = 0   RX_POLL_RET_TIMEOUT 0   RX_PROHIBIT_TIME_OUT 0
[RLC] DTX Mode UM, NB_SDU_TO_RX = 1948265920 (bytes 1818326127)

```

Figure 4.18 L2 stats

Since most of the innovation work in OAI are done in the developed branches, I tried to apply the same setup in recent branch of OAI tag v0.5. In this case, there will be a problem with the GTPU module. To solve this, we should use a new kernel which is higher than 4.7.7. And the eNB should be updated to Ubuntu 16.04 with a kernel of lowlatency. Here we use the kernel 4.13 lowlatency.



Figure 4.19 System and Hardware

Another difference is that the EPC was separated into mme, hss and spgw. So it has to be compiled and run independently.

Build mme,hss and spgw.

```
cd openair-cn
```

```
git checkout develop
```

```
git pull
```

```
cd SCRIPTS
```

```
./build_mme -i #(Need to run only once to install missing packages)
```

```
./build_hss -i #(Need to run only once to install missing packages)
```

```
./build_spgw -i #(Need to run only once to install missing packages)
```

The adjustment in the config file is also different. First we have to copy the EPC config files in /usr/local/etc/oai:

```
sudo mkdir -p /usr/local/etc/oai/freeDiameter
sudo cp ~/openair-cn/ETC/mme.conf /usr/local/etc/oai
sudo cp ~/openair-cn/ETC/hss.conf /usr/local/etc/oai
sudo cp ~/openair-cn/ETC/spgw.conf /usr/local/etc/oai
sudo cp ~/openair-cn/ETC/acl.conf /usr/local/etc/oai/freeDiameter
sudo cp ~/openair-cn/ETC/mme_fd.conf /usr/local/etc/oai/freeDiameter
sudo cp ~/openair-cn/ETC/hss_fd.conf /usr/local/etc/oai/freeDiameter
```

And then, all the adjustment should be done in the new config files, otherwise all the change would not be compile in the EPC.

The changes in the mme.conf and spgw.conf is the same as the epc.config in the master branch.

In MME freediameter configuration file
(/usr/local/etc/oai/freeDiameter/mme_fd.conf):

```
Identity = "nano.openair4G.eur";
Realm = "openair4G.eur";
ConnectPeer= "hss.openair4G.eur"
```

The identity should be your hostname.openair4G.eur. Otherwise error “returning 22” will occur when we run the mme.

Running eNB, EPC and HSS:

Install certificates:

```
cd ~/openair-cn/SCRIPTS
./check_hss_s6a_certificate /usr/local/etc/oai/freeDiameter/
hss.openair4G.eur
```

*./check_mme_s6a_certificate
nano.openair4G.eur*

/usr/local/etc/oai/freeDiameter/

Before running the EPC side, we need to write the information of the sim card into the hss as well.

However, in this case, when we follow the same process as we did in Ubuntu 14.04, the following error will appear.

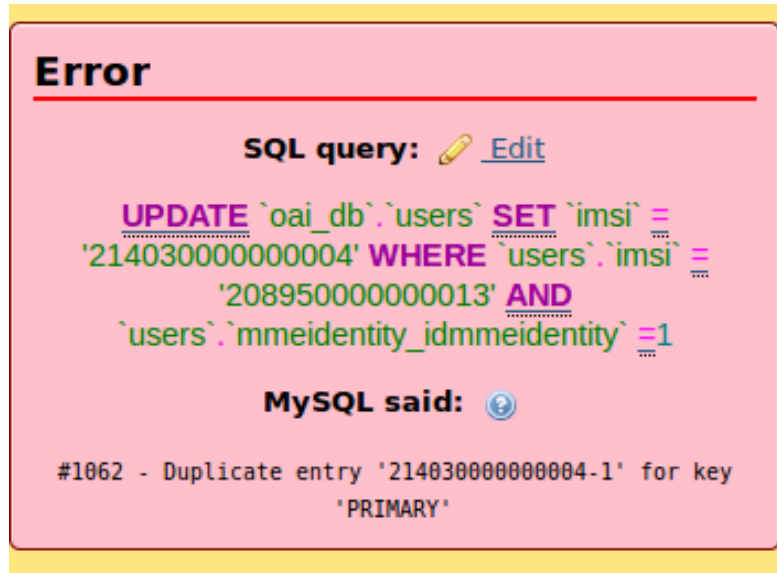


Figure 4.20 Error in database

At first, I thought this happened because I edit the original user's information in the database. So I use the insert instead of edit. But the same problem happened.

The main reason was that the information filled in was inconsistent with the sim card information, and the filling method was not due to the reasons.

1) In the phpmyadmin database, click on the insert directly above to add, the error is that value key, rand and OPc three values are truncated;

2) In the mysql command line to directly perform the add command, there is warning, because the data is truncated;

3) Change the data length of key, rand, and OPc from varbinary16 to varbinary32. The added data will not be truncated, but the data will automatically change when hss is run, resulting in authentication failure.

Finally through learning to find the difference between the data types, I found out that the data must be changed to binary, and add a hexadecimal number written as 0x. And after this, we can successfully run the whole system.

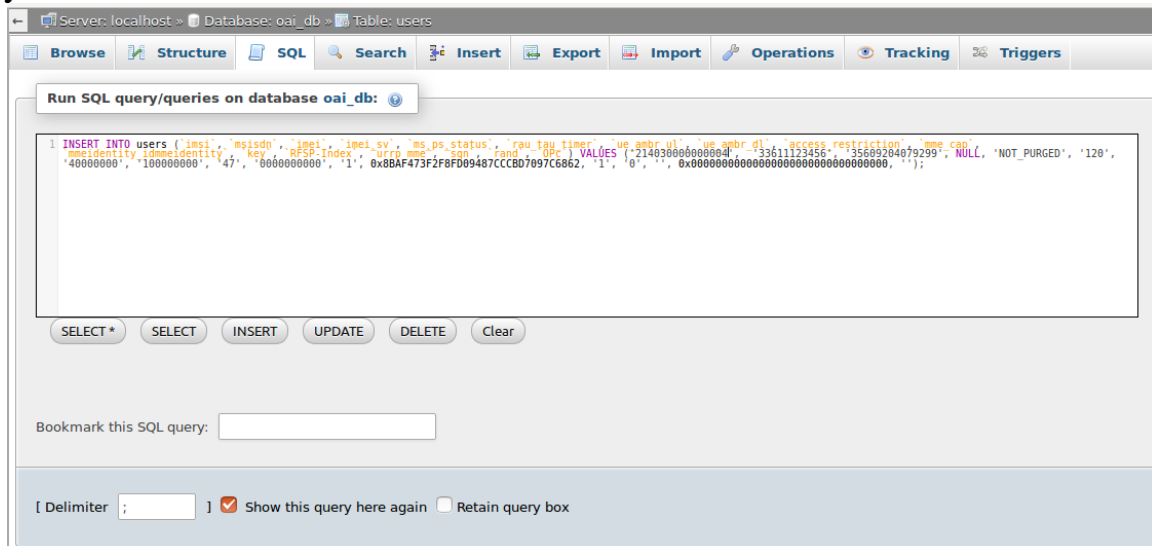


Figure 4.21 Use SQL Language to add user into database

Enter the following into the SQL session:

```

INSERT INTO users (`imsi`, `msisdn`, `imei`, `imei_sv`, `ms_ps_status`,
`rau_tau_timer`, `ue_ambr_ul`, `ue_ambr_dl`, `access_restriction`,
`mme_cap`, `mmeidentity_idmmeidentity`, `key`, `RFSP-Index`, `urrrp_mme`,
`sqn`, `rand`, `OPc`) VALUES ('214030000000004', NULL, NULL, NULL,
'NOT_PURGED', '120', '40000000', '100000000', '47', '0000000000', '1',
0x8BAF473F2F8FD09487CCBD7097C6862, '1', '0', '',
0x00000000000000000000000000000000, '');
  
```

And after this, we can successfully run the whole system.

Compile & Run HSS (ALWAYS RUN HSS FIRST):

```
cd ~/openair-cn
```

```
cd SCRIPTS
```

```
./build_hss -c
```

```
./run_hss -i ~/openair-cn/SRC/OAI_HSS/db/oai_db.sql #Run only once to  
install database
```

```
./run_hss #Run for all subsequent runs
```

Compile & Run MME:

```
cd ~/openair-cn/SCRIPTS
```

```
./build_mme -c
```

```
./run_mme
```

Compile & Run SP-GW:

```
cd ~/openair-cn
```

```
cd SCRIPTS
```

```
./build_spgw -c
```

```
./run_spgw
```

After EPC is connected to HSS, compile and start eNB with the same command used in the master branches.

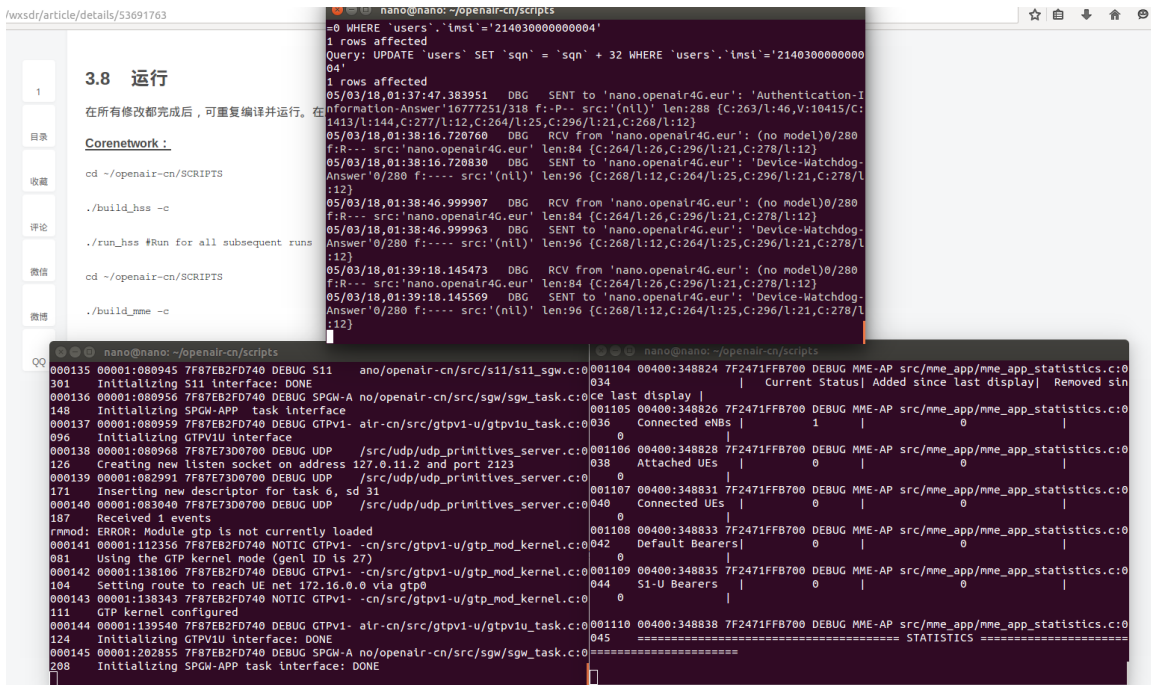


Figure 4.22 mme, hss and spgw successfully running

From the window for the mme, we can see how many eNB is attached to the mme. After the eNB is running, you should see something like this.

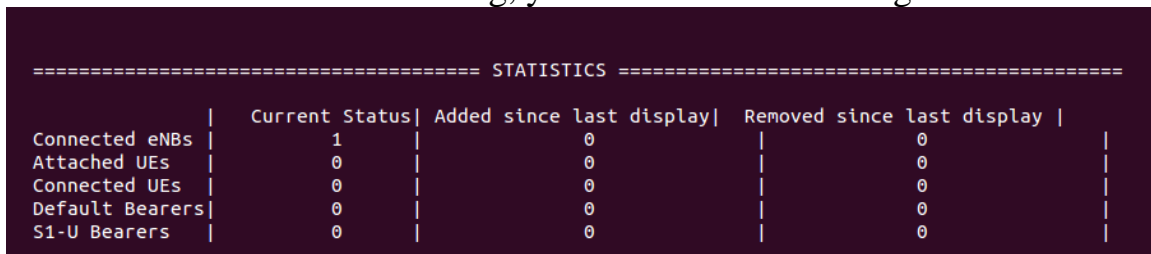


Figure 4.23 Number of eNBs connected to the MME

When the UE is connected to the LTE network, the status of Figure 4.19 will change to the one shown in Figure 4.20. And the the UE side with the dongle will show the same image I showed before.

```

===== STATISTICS =====

```

	Current Status	Added since last display	Removed since last display
Connected eNBs	1	0	0
Attached UEs	1	0	0
Connected UEs	1	0	0
Default Bearers	1	0	0
S1-U Bearers	1	0	0

Figure 4.24 UEs connected to the MME

Later on, I tested this setup with the USRP X310. The only difference is that we have to change the config file in the eNB side. Switch the file into `tm1.usrpx310.conf` and change the address of mme and eNB.

4.2 LTE Link on srsLTE

Since the srsLTE has a much better stability than the OAI, it does not require specific the kernel like the OAI. In this thesis, I choose the same system and kernel as the OAI's setup. Pay attention to the OS when installing. The following tests in this thesis use the OS of Ubuntu 16.04.

4.2.1 Get repository

After the system is installed, you just need to install common tools, such as git, vim, etc. You can use Ubuntu system `apt-get` to install.

First, install git on the PC.

```
sudo apt-get update
```

```
sudo apt-get install git
```

The USRP B210 is selected as the hardware device, so the UHD driver needs to be installed on the system. Currently UHD version 3.10.0.0 is relatively stable for OAI and srsLTE.

```
git clone https://github.com/EttusResearch/uhd
```

```
cd uhd
```

```
git checkout release_003_010_000_000
```

Test: If the above steps are completed, you can insert the usrp into your computer for testing.

```
sudo uhd_find_devices
```

```
sudo uhd_usrp_probe
```

If the usrp device is detected and the information of the USRP is shown on the computer, that means the uhd is successfully installed and the connection between the PC and the usrp is working properly.

And then, on Ubuntu 16.04 system or higher version, execute the following command

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

In addition, srs also provides the srsGUI library, which can be used to view the constellations and signal energy in real time during debugging. The github link is: <https://github.com/srslte/srsgui>. The installation command is as follows:

```
sudo apt-get install libboost-system-dev libboost-test-dev libboost-thread-  
dev libqwt-dev libqt4-dev
```

```
git clone https://github.com/suttonpd/srsgui.git
```

```
cd srsGUI
```

```
mkdir build
```

```
cd build
```

```
cmake ../
```

```
make
```

```
sudo make install
```

```
sudo ldconfig
```

And after this we can install the srsLTE by using the command:

```
git clone https://github.com/srsLTE/srsLTE
mkdir build
cd build
cmake ../
make
make test
sudo make install
sudo ldconfig
```

4.2.2 LTE network construction

The operating parameters of srsLTE are all configured using configuration files. That is, only the configuration file needs to be added after the running program while the configuration parameters are performed in the configuration file. The srsLTE provides corresponding sample configuration files for the UE, eNB, and EPC. Placed in the corresponding srsUE, srsENB, and srsEPC folders, and named as *.example, as shown below srsENB configuration file:

```
nano@nano:~/srsLTE/srsenb$ ls
CMakeLists.txt  drb.conf.example  enb.conf.example  hdr  rr.conf.example  sib.conf.example  src  test
```

Figure 4.25 Configuration files in srsLTE

While running, these configuration files can be copied and the file name modified, such as assigning enb.conf.example and named enb.conf.

srsENB and srsEPC can be run on one computer(with virtual machine) or on two different computers, but it is necessary to ensure the IP reachability between the two machines and modify the related configuration files; srsUE needs to run on another computer. In this thesis, I choose the Huawei E3372 dongle as the UE to compare the performance of OAI and the srsLTE.

After executing `sudo make install`, we can directly use `srsEPC`, `srsENB`, and `srsUE` to run the corresponding module.

To construct the LTE link, we should first build an EPC in the virtual machine.

In `srsEPC`, the hss data base is much simpler than the `oai`, we just have to change the file called `user_db.csv`. Add the information of the sim card in this file will make the authentication of the dongle succeed. The information type is listed below:

```
# .csv to store UE's information in HSS  
# Kept in the following format: "Name,IMSI,Key,OP,AMF"  
# Name: Human readable name to help distinguish UE's. Largely ignored  
by the HSS  
# IMSI: UE's IMSI value  
# Key: UE's key, where other keys are derived from. Stored in  
hexadecimal  
# OP: Operator's code, sotred in hexadecimal  
# AMF: Authentication management field, stored in hexadecimal  
# SQN: UE's Sequence number for freshness of the authentication
```

So here I add the line:

```
zch,214030000000004,8baf473f2f8fd09487cccbd7097c6862,1111111111  
11111111111111111111,9001,000000001234
```

And then we have to change the ip address, `mnc`, `mcc`, etc.

```
[mme]  
mme_code = 0x1a  
mme_group = 0x0001  
tac = 4  
mcc = 214  
mnc = 03
```

mme_bind_addr = 192.168.34.128

apn = srsapn

dns_addr = 8.8.8.8

*#####
#####*

[hss]

auth_algo = xor

db_file = user_db.csv

*#####
#####*

SP-GW configuration

#

gtpu_bind_addr: GTP-U bind adress.

#

*#####
#####*

[spgw]

gtpu_bind_addr=127.0.1.100

sgi_if_addr=172.16.0.1

*#####
#####*

[log]

all_level = debug

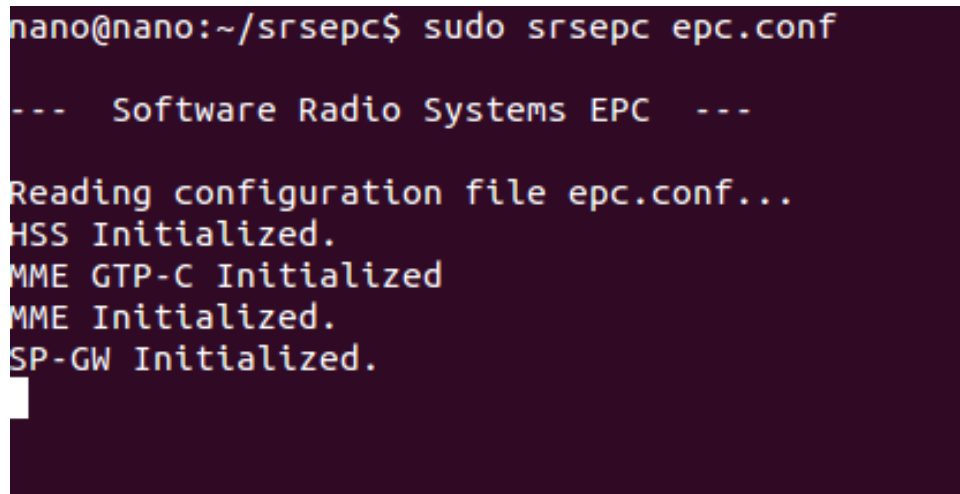
```
all_hex_limit = 32
```

```
filename = /tmp/epc.log
```

After all the files changed correctly, we can run the epc.

```
cd srsEPC/
```

```
sudo srsEPC epc.conf
```



```
nano@nano:~/srsepc$ sudo srsepc epc.conf
--- Software Radio Systems EPC ---
Reading configuration file epc.conf...
HSS Initialized.
MME GTP-C Initialized
MME Initialized.
SP-GW Initialized.
```

Figure 4.26 srsEPC successfully initialized

In the eNB side, we have to run the eNB in the src file. So I copy these conf file and then renamed them to delete the “example” in their name.

```
cd srsLTE/build/srsENB/src
```

```
cp ../../../../srsENB/*.example
```

```
mv sib.conf.example sib.conf
```

```
mv rr.conf.example rr.conf
```

```
mv enb.conf.example enb.conf
```

```
mv drb.conf.example drb.conf
```

The next step is to open enb.conf and configure srsENB to open the GUI mode of the srsENB.

Find the following line:

```
[gui]
```

enable = false

And change it into:

[gui]

enable = true

In addition, the address of the MME needs to be configured in enb.conf to be the same as the address of the MME in the virtual machine. The address of the MME changed in enb.conf is as follows. (Change 127.0.1.100 to the address of your MME)

```
mme_addr = 192.168.34.128
```

And finally, run the srsENB with the command:

```
sudo ./srsENB enb.conf
```

When the UE is successfully attached with the epc and the enb, the figure in the PC is as figure 4.23 showed.

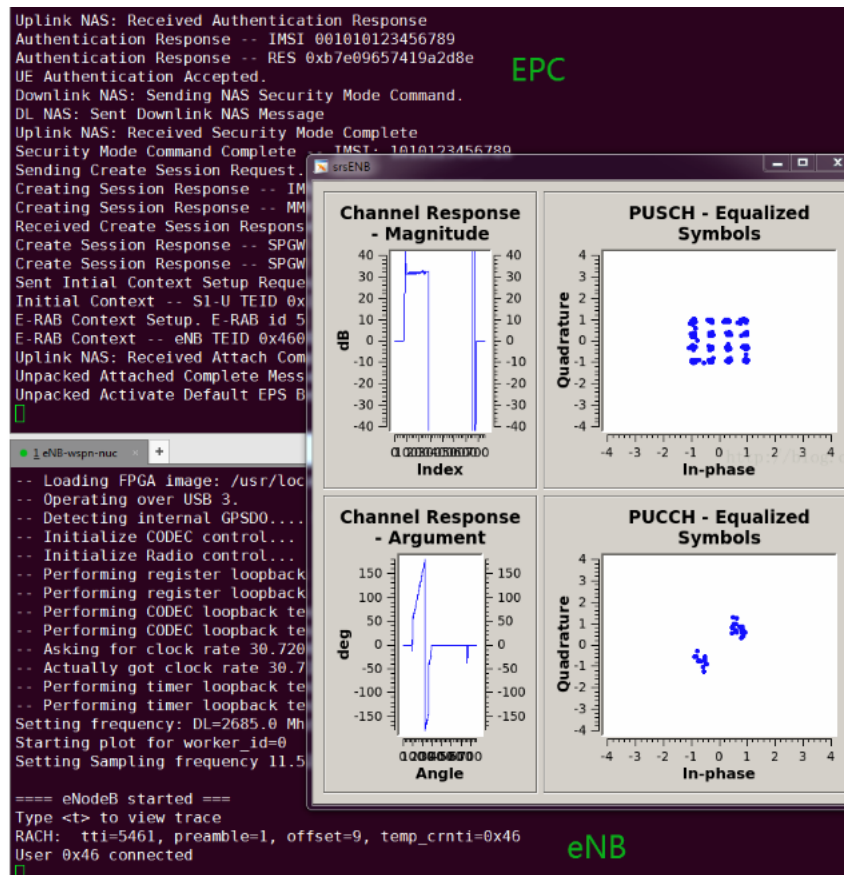


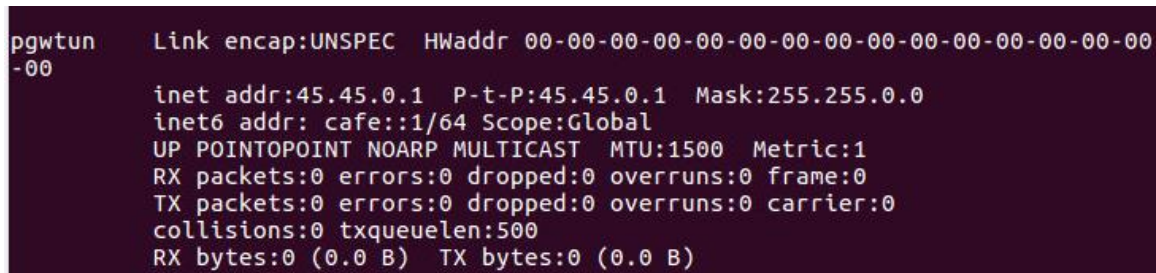
Figure 4.27 srsEPC, srsENB and srsUE successfully connected

4.2.3 srsLTE with Next EPC

Using Next EPC as the EPC to work with the srsLTE is another setup. In this case, we have to first install the next EPC.

```
sudo apt-get update
sudo apt-get -y install software-properties-common
sudo add-apt-repository ppa:acetcom/nextepc
sudo apt-get update
sudo apt-get -y install nextepc
```

This will create a virtual network interface named as pgwtun. It is automatically removed by uninstalling NextEPC. If the interface is successfully set, you should see the following figure after entering “ifconfig”



```
pgwtun  Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
-00
    inet addr:45.45.0.1  P-t-P:45.45.0.1  Mask:255.255.0.0
    inet6 addr: cafe::1/64 Scope:Global
    UP POINTOPOINT NOARP MULTICAST  MTU:1500  Metric:1
    RX packets:0 errors:0 dropped:0 overruns:0 frame:0
    TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:500
    RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Figure 4.28 Information of interface pgwtun

The LTE user subscription information of NextEPC is stored and maintained by Mongo DB. NextEPC provides an alternative management interface for customers to manage their subscriber information in an easy way, that is Web User Interface. The following shows how to install the Web UI of NextEPC.

```
sudo apt-get -y install curl
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
```

curl -sL http://nextepc.org/static/webui/install | sudo -E bash -

Now the web server is running on *http://localhost:3000*.

Then we can log in and enter the information of our sim card.

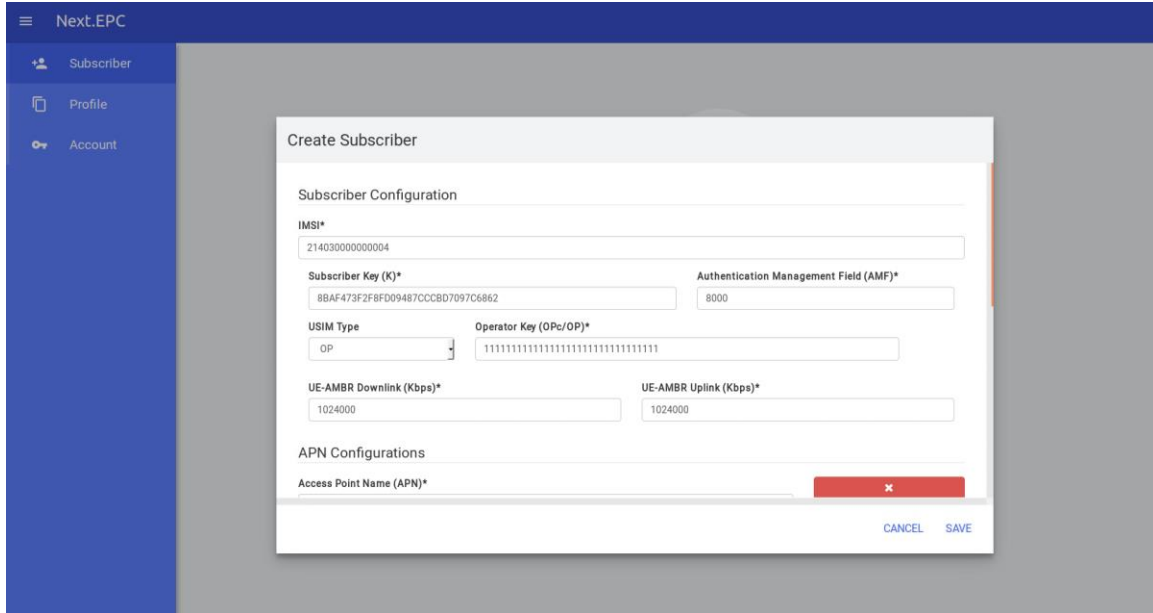


Figure 4.29 Information of sim card saved in the next EPC

Then, S1AP/GTP-C IP address, PLMN ID, TAC updated in the MME Configuration.

Add S1AP address : 127.0.1.100 (srsENB Default Value)

Add GTP-C address : 127.0.1.100 (Use Loopback Interface)

```
diff -u mme.conf.old mme.conf
```

```
--- mme.conf.old 2018-04-15 18:28:31.000000000 +0900
```

```
+++ mme.conf 2018-04-15 19:53:10.000000000 +0900
```

```
@@ -14,18 +14,20 @@
```

```
mme:
```

```
freeDiameter: mme.conf
```

```
s1ap:
```

```
addr: 127.0.1.100
```

```
gtpc:
addr: 127.0.1.100

gummei:

plmn_id:
mcc: 214
mnc:03
mme_gid: 2
mme_code: 1

tai:

plmn_id:
mcc: 214
mnc: 03
tac: 4
```

GTP-U IP address updated in the SGW Configuration.

Add GTP-U address : 127.0.0.2 (Use Loopback Interface)

```
diff -u /etc/nextepc/sgw.conf.old /etc/nextepc/sgw.conf
--- sgw.conf.old 2018-04-15 18:30:25.000000000 +0900
+++ sgw.conf 2018-04-15 18:30:30.000000000 +0900
@@ -14,3 +14,4 @@

gtpc:
addr: 127.0.0.2

gtpu:
addr: 127.0.0.2
```

After changing conf files, restart NextEPC daemons.

```
sudo systemctl restart nextepc-mmed
```

```
sudo systemctl restart nextepc-sgwd
```

After this, the EPC will be successfully run.

5. Evaluations

In this chapter, I test the performance of two different softwares for SDR, and evaluate the performance of USRP B210 with the dongle E3372.

5.1 Channel quality of B210 and E3372

First, I do some tests on the basic setup of our LTE link. Since we always have problems with the strength of the signal without antenna, I test the CQI without antenna when the distance between USRP and Dongle is different.

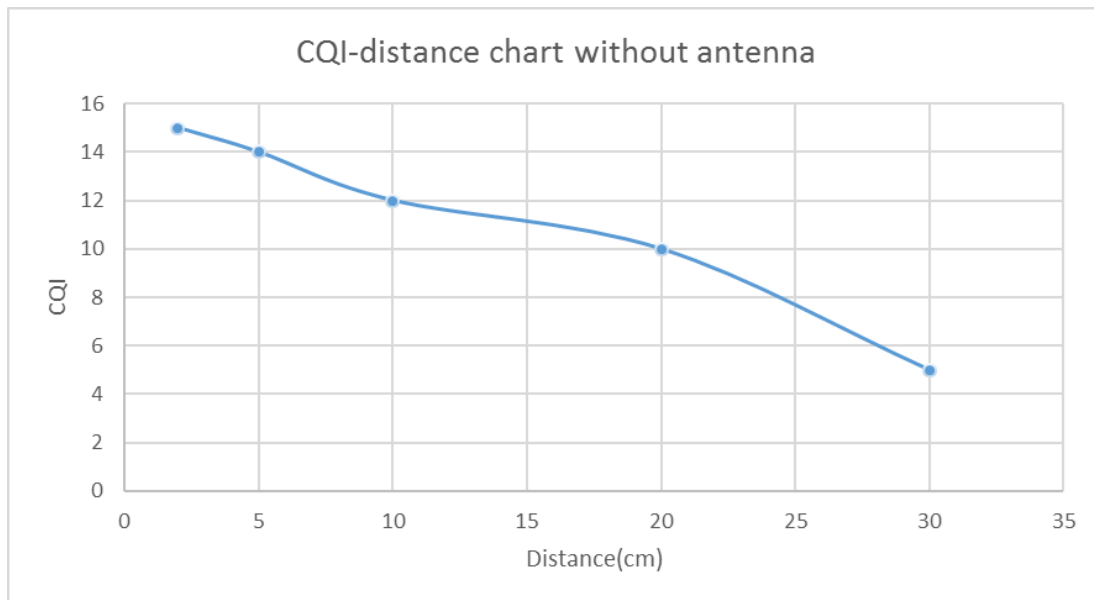


Figure 5.1 CQI-distance chart with antenna

The CQI is an information indicator of channel quality and represents the quality of the current channel, which corresponds to the signal-to-noise ratio of the channel. The value ranges from 0 to 15. When the CQI value is 0, the channel quality is the worst; when the CQI value is 15, the channel quality is the best. In this case the LTE link only works properly within 20cm. This chart

has shown that the USRP B210 has a very poor performance without the antenna. So I did another test with the antenna of 2.4GHz.

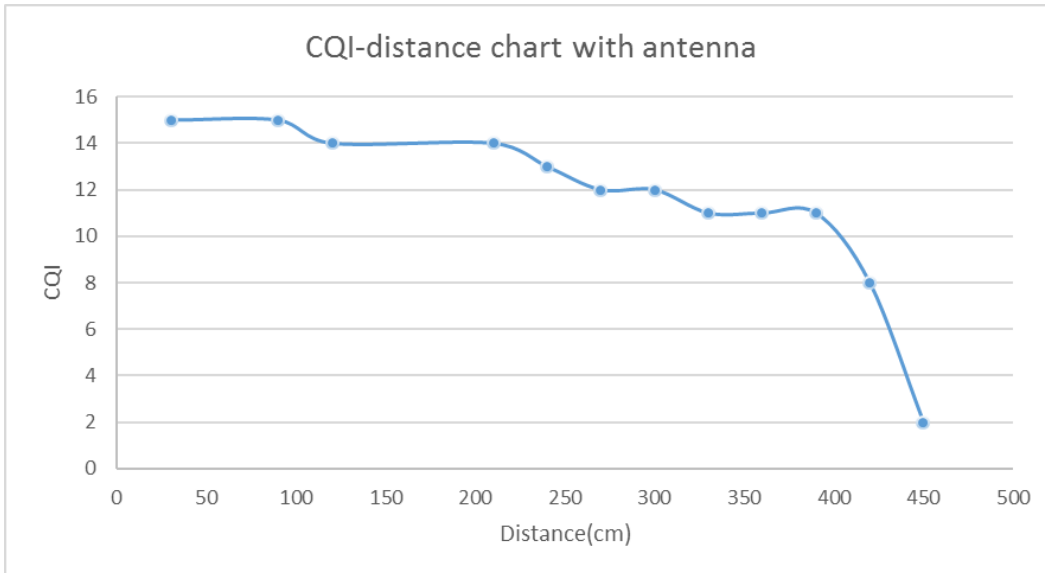


Figure 5.2 CQI-distance chart with antenna(OAI)

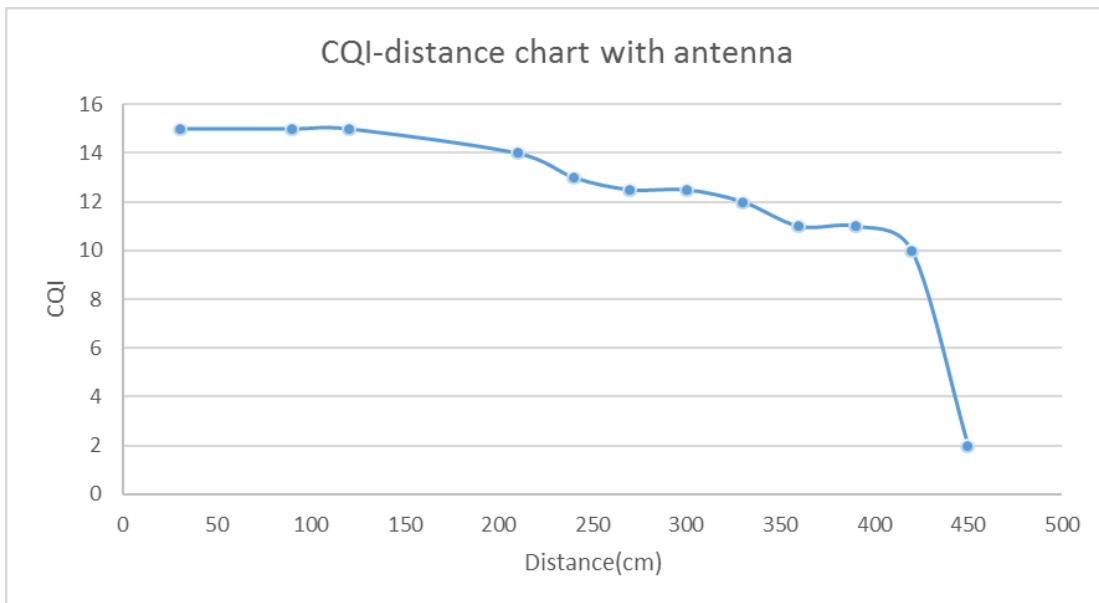


Figure 5.3 CQI-distance chart with antenna(SRSLTE)

According to the result, the setup will work until the distance is larger than 4 meters. So USRP B210 should work with cables or antennas during tests, otherwise the transmitting signal would be too weak to detect. For srsLTE, the signal was more stable, especially when the signal is weak. The

LTE link was more likely to break in OAI when the distance is far away while the srsLTE would have the connection even though the CQI is very low.

5.2 Performance of OAI

In the UE side, the interface of the dongle is shown in the following figure.

```
enx0c5b8f279a64 Link encap:Ethernet HWaddr 0c:5b:8f:27:9a:64
  inet addr:192.168.8.100 Bcast:192.168.8.255 Mask:255.255.255.0
  inet6 addr: fe80::e457:5b9a:5629:19f9/64 Scope:Link
  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
  RX packets:5765 errors:0 dropped:0 overruns:0 frame:0
  TX packets:5106 errors:0 dropped:0 overruns:0 carrier:0
  collisions:0 txqueuelen:1000
  RX bytes:6166591 (6.1 MB) TX bytes:539316 (539.3 KB)
```

Figure 5.4 Interface of the dongle

First, I did some ping test from my laptop (ThinkPad X1 carbon 2016, OS: Windows 10/Ubuntu 16.04), which is connected to the dongle. Ping tests of srsLTE and OAI had the exact same condition. In this case, I use USRP B210 as the SDR, both tests were run without antenna and the distance between the dongle and the USRP is 10cm, with a high CQI=15.

```
C:\Users\Christopher Zhu>ping 192.168.34.128

Pinging 192.168.34.128 with 32 bytes of data:
Reply from 192.168.34.128: bytes=32 time=39ms TTL=63
Reply from 192.168.34.128: bytes=32 time=53ms TTL=63
Reply from 192.168.34.128: bytes=32 time=48ms TTL=63
Reply from 192.168.34.128: bytes=32 time=69ms TTL=63

Ping statistics for 192.168.34.128:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 39ms, Maximum = 69ms, Average = 52ms
```

Figure 5.5 Ping test of OAI

Then, I tested the Bandwidth in OAI when the PRB=25(default setup) both in TCP and in UDP with two 2.4GHz antennas.

```
Server listening on TCP port 8000
TCP window size: 85.3 KByte
-----
[ 3] local 192.168.8.100 port 57186 connected with 192.168.34.128 port 8000
[ ID] Interval      Transfer      Bandwidth
[ 3] 0.0-10.0 sec  6.55 MBytes  4.06 Mbits/sec
```

Figure 5.6 Downlink data rate using TCP

```
Server listening on UDP port 8000
Receiving 1470 byte datagrams
UDP buffer size: 200 Byte
-----
[ 3] local 192.168.8.100 port 62728 connected with 192.168.34.128 port 8000
[ ID] Interval      Transfer      Bandwidth
[ 3] 0.0-10.0 sec  9.79 MBytes  8.21 Mbits/sec
```

Figure 5.7 Downlink data rate using UDP

In addition, in order to find the optimal data rate, I changed the size of the TCP window and the UDP buffer size to finish both TCP and UDP tuning.

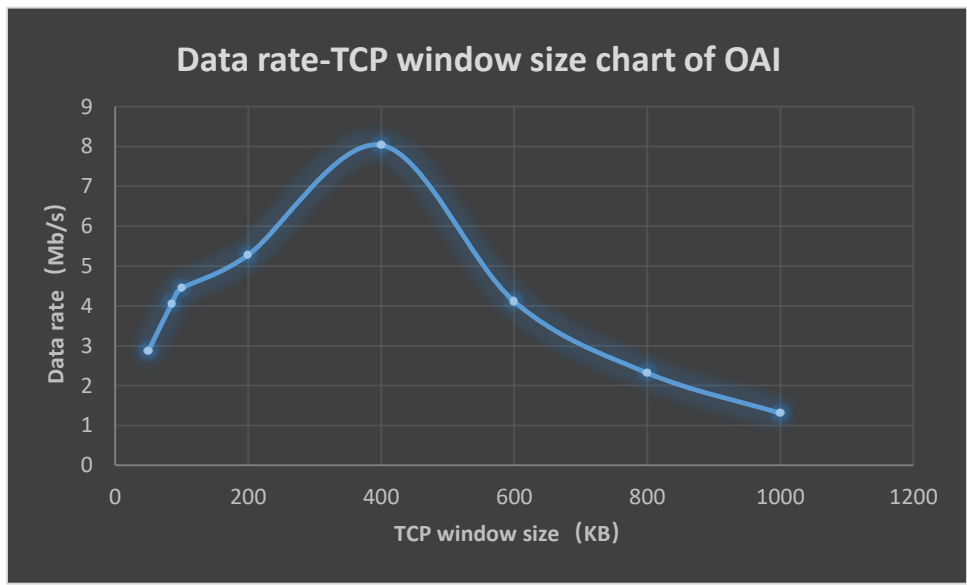


Figure 5.8 Data rate-TCP window size chart of OAI

Here we can observe that the peak of the data rate was near the TCP window size of 400KB while the UDP buffer size was not directly related to the data rate. So in the UDP test, I still use the default number 208KB.

After the TCP and UDP tuning, since the uplink and the downlink speed could be different, I did some test about the uplink and downlink speed in both TCP and UDP.

```
Server listening on TCP port 8000
TCP window size: 400 KByte (WARNING: requested 200 KByte)
-----
[ 4] local 192.168.34.128 port 8000 connected with 172.16.0.2 port 49278
[ ID] Interval      Transfer      Bandwidth
[ 4] 0.0- 5.9 sec   3.00 MBytes   4.24 Mbits/sec
```

Figure 5.9 Bandwidth of TCP uplink

```
Server listening on TCP port 8000
TCP window size: 400 KByte (WARNING: requested 200 KByte)
-----
[ 3] local 192.168.8.100 port 59664 connected with 192.168.34.128 port 8000
[ ID] Interval      Transfer      Bandwidth
[ 3] 0.0- 5.3 sec   5.50 MBytes   8.65 Mbits/sec
```

Figure 5.10 Bandwidth of TCP downlink

```
Client connecting to 192.168.34.128, UDP port 8000
Sending 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 3] local 192.168.8.100 port 59830 connected with 192.168.34.128 port 8000
[ ID] Interval      Transfer      Bandwidth
[ 3] 0.0- 5.0 sec   2.75 MBytes   4.62 Mbits/sec
```

Figure 5.11 Bandwidth of UDP uplink

```
Server listening on UDP port 8000
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
-----
[ 3] local 192.168.8.100 port 46658 connected with 192.168.34.128 port 8000
[ ID] Interval      Transfer      Bandwidth
[ 3] 0.0- 5.0 sec   5.26 MBytes   8.81 Mbits/sec
```

Figure 5.12 Bandwidth of UDP downlink

From the figure above, we can notice that the downlink bandwidth of this link is about 2 times faster than the uplink. And since in this setup, there are very few lost packets, the transmitting speed of TCP and UDP are very similar. If we vary the distance between the eNB and the dongle, the data rate of the UDP would be faster than the TCP when there are more packets lost as the following figure has shown.

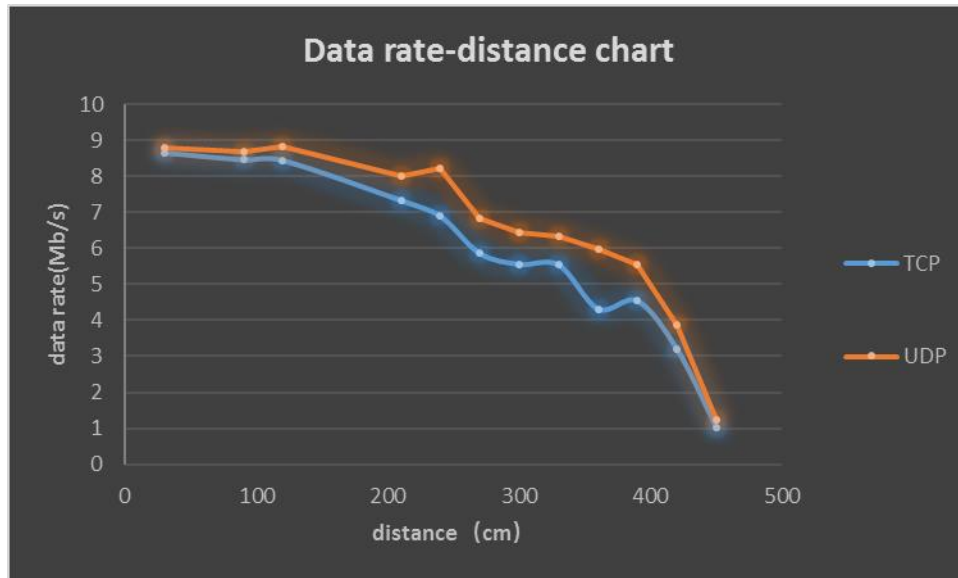


Figure 5.13 Data rate-distance chart of TCP and UDP

After the tests, I found out that the stability is the best when the PRB=25 (bandwidth=5MHZ) when I tried to run the same setup when PRB=50 and 100. In the OAI, the default number of PRB is 25. For PRB=50, the eNB will fail to transmit signal with UE and the error thrown is “retransmission takes more resource than the system has”. And for PRB=100, the setup would connect but running the iperf would not work for the unstability of the LTE link.

5.3 Performance of srsLTE

First, I did the same ping test in srsLTE to compare the data with the OAI’s.

```

C:\Users\Christopher Zhu>ping 192.168.34.128

Pinging 192.168.34.128 with 32 bytes of data:
Reply from 192.168.34.128: bytes=32 time=28ms TTL=63
Reply from 192.168.34.128: bytes=32 time=38ms TTL=63
Reply from 192.168.34.128: bytes=32 time=30ms TTL=63
Reply from 192.168.34.128: bytes=32 time=35ms TTL=63

Ping statistics for 192.168.34.128:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 28ms, Maximum = 38ms, Average = 32ms

```

Figure 5.14 Ping test of srsLTE

From the figure above we can see that the delay of srsLTE is obviously smaller than the OAI. At first, I believed that it is because the code of srsLTE is tidier and more in order than OAI, which make the LTE link more stable and faster. However, when I tried to ping the eNB(192.168.34.1), it turned out both the softwares have a delays of about 32ms. I believed that the cause is the difference between the EPC. The srsEPC is a lightweight LTE Core Network implementation platform, which make it easier to run and has less to process in the computer. Since I was using a virtual machine which do not have a high performance, the OAI EPC could use more time than the srsEPC.

In addition, I use iperf with TCP to test the downlink data rate of both srsLTE using PRB=25. The port I used is 8000 and the TCP window is 85.3KB.

```

Server listening on TCP port 8000
TCP window size: 85.3 KByte (default)
-----
[  4] local 192.168.34.128 port 8000 connected with 172.16.0.2 port 49196
[ ID] Interval           Transfer     Bandwidth
[  4] 0.0- 5.8 sec    2.75 MBytes  4.01 Mbits/sec

```

Figure 5.15 Data rate test in srsLTE

As we can see, the transmitting speed of these software is very similar to OAI.

I did the same TCP tuning to find the optimal TCP window for srsLTE.

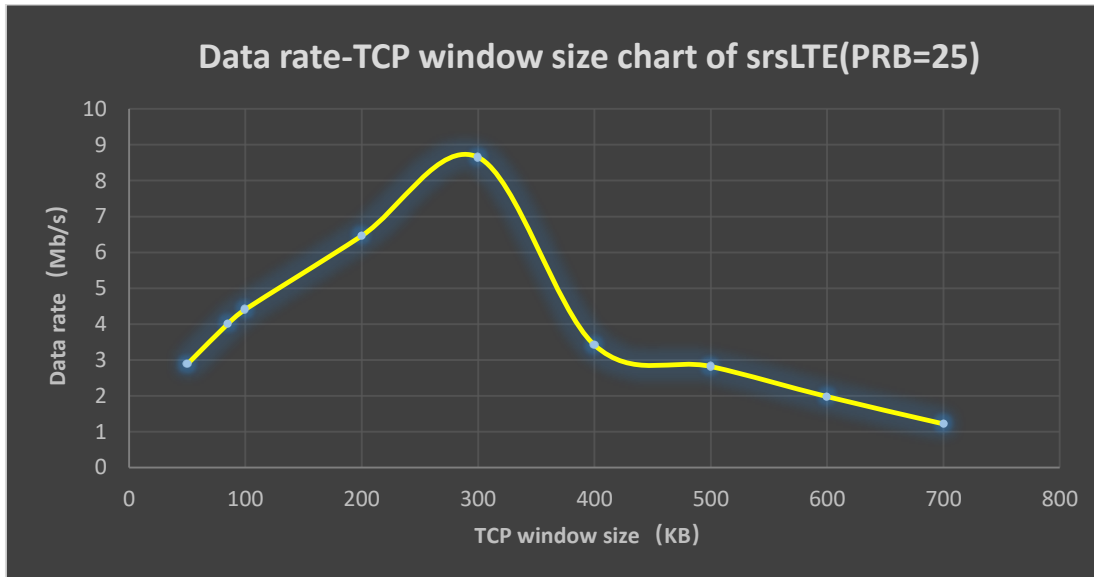


Figure 5.16 Data rate-TCP window size chart of srsLTE(PRB=25)

We can find that the TCP window size in srsLTE is smaller in OAI and when the data rate is highest, which meets our expectation because of the feature of TCP (the delay of srsLTE is smaller.).

Unlike the OAI, srsLTE works perfectly when PRB=50 and 100. So the tuning of TCP was done in both bandwidth to find the optimal TCP window size and the highest data rate.

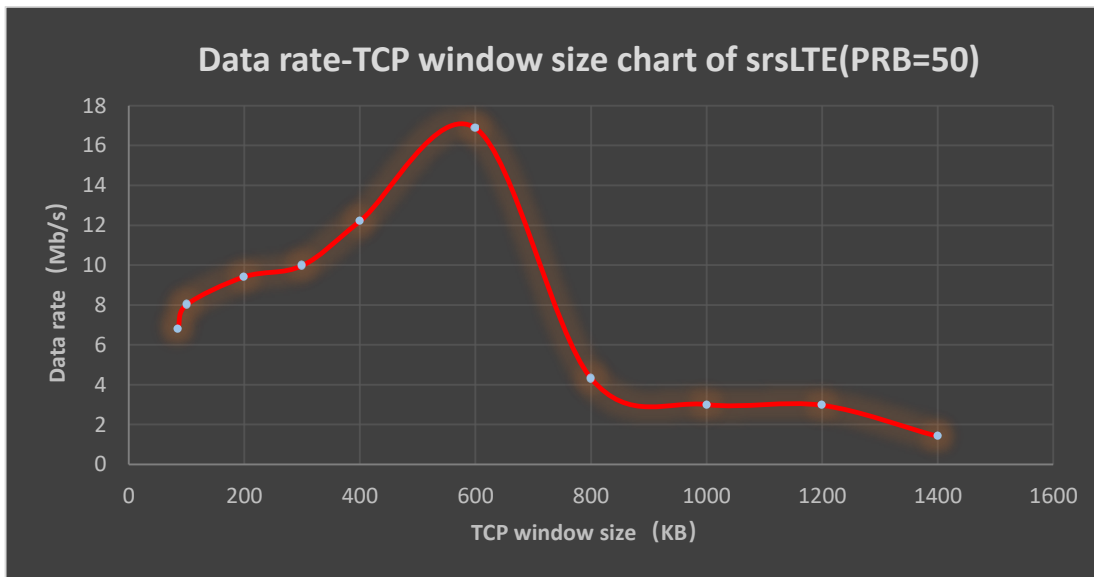


Figure 5.17 Data rate-TCP window size chart of srsLTE(PRB=50)

From the figure above, we can find that the peak of data rate reached 17Mb/s with optimal TCP window size of 580 when PRB=50. Both the data rate and the TCP window size observed in downlink is almost double of 25 PRBs, which is expected as the bandwidth is doubled in this case.

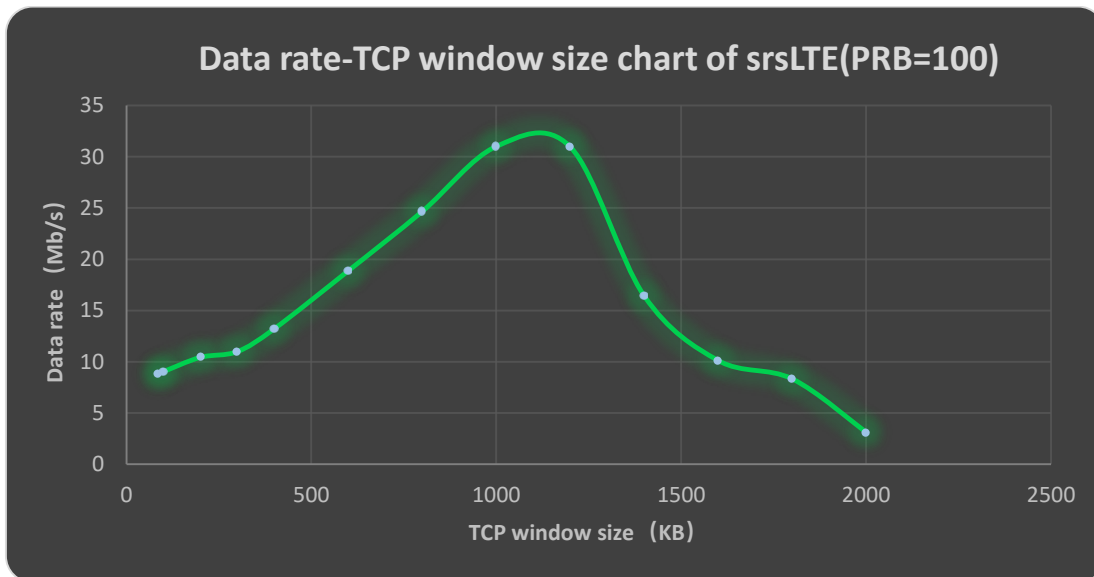


Figure 5.18 Data rate-TCP window size chart of srsLTE(PRB=100)

As for PRB=100, we can find that the peak of data rate reached 33Mb/s with optimal TCP window size of about 1100. Both the data rate and the TCP window size observed in downlink is approximately 4 times of 25 PRBs. All of these has shown that the srsLTE runs perfectly and presents the feature of the LTE network.

6. Conclusions and future work

In this thesis, we test and evaluate the performance of a set up in two different opensource LTE and EPC project (by OpenAirInterface project and srsLTE project) with commodity hardware.

At the beginning, I built a conventional LTE network, which uses COTS UE, a computer as the eNB and a virtual machine in the same computer as the EPC.

And then, based on this setup, I tested two different open source projects. Later I evaluated the stability, bandwidth and the hardware performance in these two setups.

For the hardware, USRP X310 and B210 both work in this construction I built. In this moment, the B210 is more stable due to the requirement is only USB3.0 while the X310's bandwidth and power requirements are rather high, which would cause some problem.

For the software, the OAI is obviously a more popular software. However, in my humble opinion, the OAI is not very friendly for a first time user, especially those who do not have a background knowledge of LTE and SDR. And since the OAI is developing every day, the new errors could appear anytime. For example, there are still some problems with the OAI UE right now in the develop branch. On the other hand, the srsLTE is much simpler than the OAI, the code is more in order and the testing process is very easy. (For example, the hss do not require an external database.) Few error occurred during the testing of srsLTE. But for a more advanced user, the srsLTE only focuses on the LTE part, which means that it is hard to test the technology after 3GPP R14 such as LWA and the tech in 5G. So for these uses, OAI is a much better option.

For future work, the config file of the eNB should be modified to solve the problem when PRB=75. After the OAI UE problems are solved, a more detailed comparison between srs and OAI in this setup should be done to get a more general conclusion of these two softwares.

Bibliography

- [1] C. S. Inc., Cisco visual networking index: global mobile data traffic forecast update 2016-2012, 2017.
- [2] J. Sachs and P. J. Gebert, "Multi-access management in heterogeneous networks" , Wireless Personal Communications, vol. 48, no. 1, pp. 7-32, January 2009.

- [3] V. Raghavan, J. Cezanne, S. Subramanian, A. Sampath, O. Koymen, "Beamforming tradeoffs for initial UE discovery in millimeter-wave MIMO systems", *IEEE J. Sel. Topics Signal Process.*, vol. 10, no. 3, pp. 543-559, Apr. 2016.
- [4] J. G. Andrews, S. Buzzi What will 5g be? Selected Areas in Communications, *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 6, pp. 1065-1082, 2014.
- [5] Y. Kojima, J. Suga, T. Kawasaki, M. Okuda and R. Takechi, "LTE-WiFi Link Aggregation at Femtocell Base Station," *World Telecommunications Congress 2014(WTC 2014)*, pp. 1-6, Berlin, Germany, 2014.
- [6] J. Andrews, and A. Gatherer, *Femtocell networks: a survey*, *IEEE Communications Magazine*, vol. 46, no. 9, pp. 59-67, 2008
- [7] B. Jin, S. Kim, D. Yun, Y. Yi, H. Lee and W. Kim, "Aggregating LTE and Wi-Fi: Fairness and split-scheduling," *2016 14th International Symposium on Modeling and Optimization in Mobile Ad Hoc, and Wireless Networks (WiOpt)*, Tempe, AZ, 2016, pp. 1-8. doi:10.1109/WIOPT.2016.7492936
- [8] NS-3 LTE-EPC Network simulator. Available:<http://networks.cttc.es/mobile-networks/software-tools/lena/>
- [9] Y. Khadraoui, X. Lagrange and A. Gravey, "Very tight coupling between LTE and WiFi: From theory to practice," *2016 Wireless Days Conference*, Toulouse, 2016, pp. 1-3. 2016. DOI: 10.1109/WD.2016.7461502
- [10] Y. Khadraoui, X. Lagrange and A. Gravey, "TCP Performance for Practical Implementation of Very Tight Coupling between LTE and WiFi," *2016 IEEE the 84th Vehicular Technology Conference*, Montreal, QC, 2016, pp. 1-6. doi: 10.1109/VTC.
- [11] P. Sharma, A. Brahmakshatriya and A. Franklin, "LWIR : LTE-WLAN Integration at RLC Layer with Virtual WLAN Scheduler for Efficient Aggregation" , *IEEE Global Communications Conference (GLOBECOM)*, pp. 4-8 Dec. 2016.
- [12] T. Valerrian, S. Patro, B. Reddy, and A. Franklin, "Tight coupling of LTE Wi-Fi Radio Access Networks - A Testbed Evaluation" , *Networked Wirel. Syst. Lab*, 2016.