# 1 Introduction to emulation platform

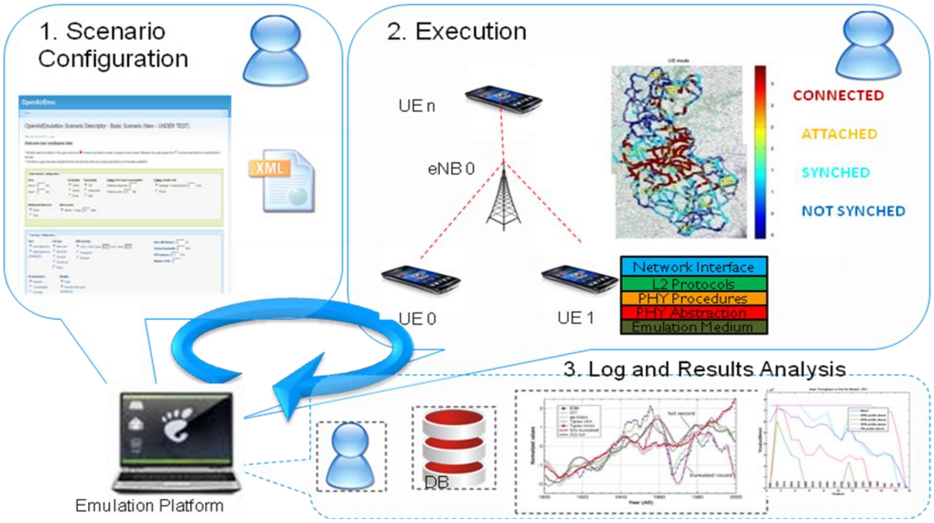
This instruction makes use of openairinterface.org emulation platform to analyze the control plane and data plane of LTE protocol stack. More specifically, we trace system information, MAC, and RRC in the control plane, and packet segmentation and reassembly in the data plane.

## 1.1 Emulation Process

As shown in the figure, the emulation consists of three phases:

- **scenario configuration:** layout of the experiment and sequence of component initialization
- **execution:** synchronize node instance and run the experiment
- **log/result analysis:** watch the experiment and process raw data (log)

The scenario is defined in xml format (see targets/SIMU/EXAMPLES/OSD/WEBXML), which is used to configure environment, network topology, application traffic, and emulation IO parameters. The behavior of the wireless medium is modeled using a PHY abstraction unit which emulates the error events in the channel decoder and provides emulated measurements from the PHY in real-time.



## 1.2 Installation and checking

To install from the scratch, checkout the code from the read-only svn repository as follows:

- mkdir openair4G
- svn co http://svn.eurecom.fr/openair4G/trunk openair4G

Then go to the target subdirectory, and follow the instructions in readme.txt. Three main steps are required as follows:

- Install the required packages
- Set the environment variables
- Install the asn1 compiler
- Run “make one_eNB_one_UE_nas” in target/SIMU/EXAMPLES/VIRT_EMUL_1eNB
- Open two shells and run “ping 10.0.1.2” and “ping 10.0.2.1”

# 2 Virtual Machine Set up

For the OAI sessions, we make use of VMware to bring up the OAI emulation environment. To start, go to E :\TPData\Tp_winterschool_2012\openair, and click on "Ubuntu.vmx", and select "moved it" to relocate the image of virtual machine. Then click on "start virtual machine". You should see the Ubuntu 11.10 starting. If needed, use the full screen mode. The login and password of VMvare is "**openair**"

To use the vm, use this URL: https://emu.openairinterface.org/openairlab/openairlab.zip
When downloaded, install libs defined in targets/README.txt if required before starting.

# 3 Log Format

The emulator provide the following log format:

[COMP][LOG LEVEL][FUNC][NODE ID][FRAME NUM] [CONTENT]

- **COMP :** RRC, PDCP, RLC, MAC, PHY, OCM (openair channel model), OMG (openair mobility generator), OCG (openair config generator), OIP (openair IP), EMU (openair emulation)
- **LOG LEVEL:** **E**merge, **A**lert, **C**ritic, **E**rror, **W**arning, **N**otice, **I**nfo, **D**ebug, **T**race
- **FUNC :** name of the function inside which the log is called. This is optional
- **NODE ID:** eNB or UE with their ID
- **FRAME NUM:** frame counter
- **CONTENT:** content of the log message

# 4 Experiment 1

The purpose of the first experiment is to follow the control plan signaling of an UE from NOT_SYNC state to establishing a DRB. We will make use of command line options instead of XML configuration for our experiment.

## 4.1 Run the emulation

Open a "terminal" and go to the Openair4G/targets/SIMU/USER

1. Optionally: check the "Makefile" with the help of an editor (e.g. emacs, gedit)
2. Run "make clean; make all": clean the emulation and compile the entire emulator
3. Check the options: ./oaisim -h
4. Run "./oaisim –a –n30 > lte.log : execute emulation with 1xeNB and 1xUE (default configuration) with phy abstraction "-a" (without coding/decoding and modulation/demodulation) for 30 frames "-n (frame duration is 10ms) "; and pipe the output of the emulation into a file called "lte.log"

Open the log.txt file with the help of an editor (e.g. gedit lte.log or emacs). You should be able to see the following messages. The keywords that you need to follow are in **bold**.

1. INIT

- Initialized all the MAC/PHY procedures
- Initialized SIB1, SIB2, and SIB3

2. **RRC_IDLE** (NOT_SYNCHED)
    - eNB Encode SI (**SIB1, SIB2, SIB3**)
    - eNB generates SI (**SIB1, SIB2, SIB3**)
    - eNB sends SI through **BCCH**
3. **RRC_SI_RECEIVED** (SI-RNTI)
    - UE received **SIB1** through BCCH
    - UE received **SIB2, SIB3** through BCCH
4. **RAPROC** MAC Random access procedure (RA-RNTI, MAC Attached )
    - Random access preamble (UE -> eNB) : msg1
    - Random access response (eNB->UE) : msg2
5. **RRC_CONNECTED** (RRC connection establishment, successful)
    - **RRCConnectionRequest** (UE->eNB) : msg3
    - **RRCConnectionSetup** (eNB->UE) : msg4
        i. Add UE
        ii. Accept new connection from UE
        iii. Contention resolution (terminate RAPROC)
    - **RRCConnectionSetupComplete** (UE->eNB) : msg5
        i. Contention resolution
        ii. (MME registered )
6. Establish a default data radio bearer (DRB) through RRC connection reconfiguration
    - **RRCConnectionReconfiguration** (eNB->UE)
    - **RRCConnectionReconfigurationComplete** (UE->eNB)

You can also further filter the messages by component, e.g. RRC and node type UE or eNB like this

- cat lte.log | grep RRC > rrc.log : RRC messages for both eNB and UE
- cat lte.log | grep RRC | grep eNB > rrc_enb.log : RRC messages for eNB

**Here is the questions:**

- In which Frame/subframe SIB1, SIB2, and SIB3 has been received by UE? What information is transported over SIB1, SIB2, and SIB3? What happened after successful reception of SIB1/SIB2/SIB3?
- Which procedure triggered MAC random access procedure ?
- When a UE receives its C-RNTI ? and when eNB adds a new UE ?
- Which logical channel is used to transport RRC messages ?
- Apart from basic RRC messages, what additional information is transported on each messages and what happens with such information?
- Which RLC mode is used for SRB0, SRB1, SRB2, and DRB ?
- What is the DRB id for UE and eNB?

# 5 Experiment 2

The purpose of the second experiment is to follow the data plan by sending IP packet to the OAI network device driver.

Figure below shows a time domain view. At the bottom are radio frames. A full frame is 10 ms but we normally think in terms of the 1-ms subframe, which is the entity that contains the transport block. Within the transport block is the MAC header and any extra space (padding). Within that there is the RLC header, then within the RLC header there can be a number of PDCPs. There is a somewhat arbitrary relationship between the IP packets coming in, which form the SDUs, and how the RLC PDUs are formed. Therefore you can make the maximum effective use of radio resources in a fixed period of time.

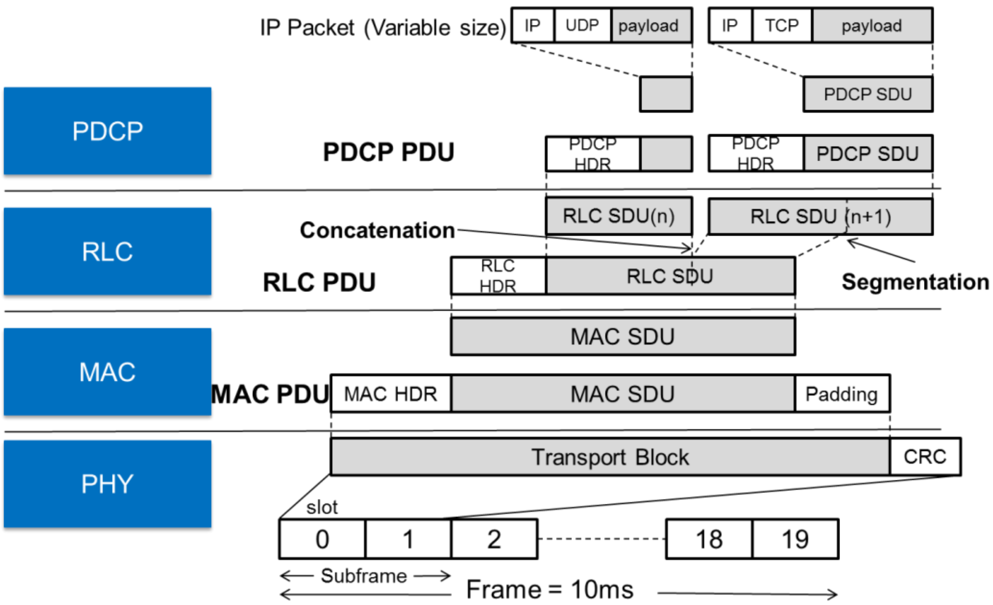
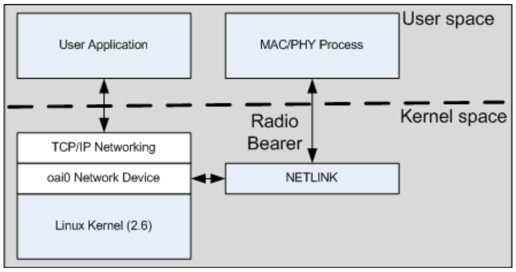


Figure below show the setup. As you see the driver in kernel space communicates with the LTE MAC/PHY process in the user space through the netlink sockets. We now need to setup the IP interface.



To accept IP connections (Non-access stratum)

1. Run “make clean; make NAS=1”: clean the emulation and compile the entire emulator
2. Run “make nasmesh_fix” and “make nasmesh_fix_install” to compile the oai driver and insert the module to the kernel space, respectively, as well as the radio bear configuration tool, called “rb_tool” by running “make rb_tool” (see

targets/SIMU/EXAMPLES/VIRT_EMUL_1eNB/ for more infromation)

- Check that the driver is inserted into kernel using “dmesg”

3. Bring up oai0 interface for eNB and oai1 interface for UE (**note : passwd = openair**)
    - sudo ifconfig oai0 10.0.1.1 netmask 255.255.255.0 broadcast 10.0.1.255
    - sudo ifconfig oai1 10.0.2.2 netmask 255.255.255.0 broadcast 10.0.2.255
4. Associate the DRB id to the oai0 (**note : passwd = openair**)
    - sudo $OPENAIR2_DIR/NAS/DRIVER/MESH/RB_TOOL/rb_tool -a -c0 -i0 -z0 -s 10.0.1.1 -t 10.0.1.2 -r 1
    - sudo $OPENAIR2_DIR/NAS/DRIVER/MESH/RB_TOOL/rb_tool -a -c0 -i1 -z0 -s 10.0.2.2 -t 10.0.2.1 -r 1

Now, check the interfaces using ifconfig, and ping the interface address using “ping 10.0.1.1”, and “ping 10.0.2.2”

**Note:** To send an icmp packet from 10.0.1.1 (eNB) to 10.0.2.2 (UE), use “ping 10.0.1.2” (this is because when transmitting data among the virtualized protocol instances, the third and fourth byte of IPv4 address represents who is the source and destination, e.g. ping 10.0.1.2 means 10.0.1.1 is sending to 10.0.2.2). For this to work, you need to run the emulator (./oaisim –a >/dev/null).

In the following two subsections, we will trace the data packet in PDCP, RLC, and MAC sub-layers.

## 5.1 PDCP

PDCP (Packet Data Convergence Protocol) is a standard designated by 3GPP to transfer user/control plane data by maintaining sequence numbers so to provide in-order delivery and duplication-avoidance, to perform IP header compression (RFC 3095, Robust Header Compression) for better bandwidth utilization (*user plane* only), cipher/decipher functionality for security purposes, and integrity protection facilities (*control plane* only). PDCP functions are symmetrical for the uplink and the downlink.

Run the experiment without defining number of frames as follows:

- ./oaisim –a | grep PDCP
- Open 2 new terminals and run the ping for both sides, i.e. “ping 10.0.1.2 – p 2B” and “ping 10.0.2.1 –p 2B”, where “p” specifies the pattern to be sent through ICMP message.
    - You may change the payload size using “-s 256” or inter-departure time using “-i 0.5”

You should see that the PDPC of eNB (instance 0) is receiving the packet from IP (**IP->PDCP**), and later on the PDCP of UE (instance 1) transmitting the packet to IP (**PDCP->IP**).

**Here are the questions :**

- What is the size of IP packet? What is the size of PDCP PDU?
- What is the number of frames/sub-frames between PDCP PDU transmission and reception for the same PDU?

- Optionally: When RLC AM (Acknowledged Mode) is activated why there still is the need of PDCP to keep track of sequence numbers and handling retransmission?

## 5.2 RLC

RLC performs segmentation and reassembly and operates in three modes: transparent mode (TM), acknowledged mode (AM) and unacknowledged mode (UM). These are used by different radio bearers for different purposes. The RLC provides in-sequence delivery and duplicate detection. RLC functions are symmetrical for the uplink and the downlink.

Run the experiment without defining number of frames as follows:

- ./oaisim –a | grep RLC
- Open a new terminal, and send one big packet like this
  - ping –c 1 –s 1024 10.0.1.2 –p 2B
- Open another terminal, and send one small packet like this
  - ping –c 1 –s 5 10.0.1.2 –p 2B

You should see that the packet is fragmented at the RLC of eNB (instance 0) with a variable size (resource block requested by MAC) and reassembled at the RLC of UE (instance 1). When all fragments of a given packet is correctly received, RLC of UE (instance 1) will pass it to PDCP.

**Here are the questions :**

- What is the size of RLC PDU?
- How many segments have been generated for the big packet and for the small packet?
- Which RLC mode is used for the data connection?

## 5.3 MAC

MAC functions are different in the uplink and downlink (not symmetric). Common functions include mapping between logical channels and transport channels, multiplexing/demultiplexing; specific eNB functions are transport format selection, and scheduling; specific UE logical channel prioritization, scheduling information reporting (**LCID, BSR, SR**).

Run the experiment as follows:

- Let the eNB ping the UE, i.e. “ping 10.0.1.2 –s 1024 –p 2B”
- Run “./oaisim –a | grep **SR_indication**
- Run “./oaisim –a | grep **BSR**
- Run “./oaisim –a | grep **LCID**

You should see that the UE is sending a scheduling request (SR_indication) to eNB and followed by the buffer status report (BSR).

**Here are the questions :**

- When the SR is transmitted ? When the BSR is transmitted ? What are the usage of SR and BSR?
- Illustrate the format and content of the received MAC PDU (subheaders, control elements, and payload) by eNB.

# 6 Shutdown the emulator

Shutdown Ubuntu (click on the right hand side of the menu bar, and select shutdown) will end the virtual machine. **Then close the VMware and log out from Windows.**