FUTEBOL

Federated Union of Telecommunications Research Facilities for an EU-Brazil Open Laboratory

FUTEBOL UFMG User Manual

Authors	Fernanda Aparecida R. Silva - Universidade Federal de Minas Gerais Julio Cesar Tadeu Guimarães - Universidade Federal de Minas Gerais Matheus Henrique do Nascimento Nunes - Universidade Federal de Minas Gerais Fábio Alves Pereira - Universidade Federal de Minas Gerais Marcos Magno de Carvalho - Universidade Federal de Minas Gerais Vinicius Fonseca e Silva - Universidade Federal de Minas Gerais Daniel Fernandes Macedo - Universidade Federal de Minas Gerais Erik de Britto e Silva - Universidade Federal de Minas Gerais
Version	0.4
Abstract	This document is a manual for the end-users of the FUTEBOL UFMG testbed. It describes how to reserve the resources available at the UFMG testbed, and also presents simple experiments that can be performed using those resources. Using those examples, the user will be able to build his/her own experiments.

This project has received funding from the European Union's Horizon 2020 for research, technological development, and demonstration under grant agreement no. 688941 (FUTEBOL), as well from the Brazilian Ministry of Science, Technology and Innovation (MCTI) through RNP and CTIC.

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.1	08/01/2018	Description of the allocation of Raspberry Pi, USRP and Wi-Fi on a PC.	Fernanda Aparecida R. Silva, Julio Cesar Tadeu Guimarães, Matheus Henrique do Nascimento Nunes, Daniel Fernandes Macedo
V0.2		Description of the allocation of Advanticsys nodes.	Marcos Magno de Carvalho
V0.3		Description of the allocation of 4G resources	Vinicius Fonseca e Silva, Marcos Magno de Carvalho, Erik de Britto e Silva
V0.4	28/02/2019	- Updates to the description of the allocation of 4G resources - Description of the allocation of LG Nexus smartphones	Vinicius Fonseca e Silva, Matheus Henrique do Nascimento Nunes, Erik de Britto e Silva, Marcos Magno de Carvalho, Fábio Alves Pereira

Table of Contents

1 - Introduction

The main objective of this document is to serve as a user guide for experimenters wishing to make an experiment in the FUTEBOL UFMG testbed. As such, we describe the resources available, how to make a reservation of those resources and we also present simple experiments using each type of resource.

We assume that the reader is familiar with jFed, that is, the reader already has an account in jFed, and already knows how to book resources using the graphic user interfaces. If you are not familiar with jFed, please read first the tutorial available at http://futebol.dcc.ufmg.br/jfed_account.html#getaccount.

Readers wishing to understand how the FUTEBOL UFMG testbed reserves resources are directed to the FUTEBOL deliverables, most notably the deliverables related to Work Package 4 - Converged Optical/Wireless Control Framework. Those can be obtained in the main FUTEBOL web site at http://www.ict-futebol.org.br.

If you feel that there is something missing from this manual, or that a certain point requires further explanation, please contact the FUTEBOL UFMG team using the e-mail web-futebol@dcc.ufmg.br.

2 - Overall Description of the Testbed

The UFMG testbed was designed to allow the experimentation in a number of wireless technologies related to the SDN and/or IoT concepts. Thus, UFMG provides a rich set of resources, from VMs to resource-constrained wireless devices. It also supports a number of wireless technologies, such as Wi-Fi and Bluetooth, and many others using programmable radios (USRPs). Further, the testbed supports SDN by using a physical OpenFlow switch.

2.1 - Testbed Resources

The resources made available to experimenters by the UFMG's Testbed include:

- Eight Dell Alienware Alpha R2 Mini Gaming PCs (MiniPCs) as a platform for experimentation on Wi-Fi, Bluetooth and USRP. The MiniPCs run Ubuntu 16.04.2 LTS linux. Each Mini PC is equipped with a Software-defined Radio as follows:

    - Four USRP B200 SDR Kits with GPSDO[1] and two 2.4GHz antennae
    - Four USRP B210 SDR Kits with GPSDO and four 2.4GHz antennae

- Sixteen Advanticsys MTM-CM5000-MSP IoT Nodes. It is an IEEE 802.15.4 WSN mote fully compatible with TelosB platform, and TinyOS 2.x & ContikiOS Compatible. It has temperature, relative humidity and light sensors. They are connected via an USB interface. More information at https://www.advanticsys.com/shop/mtmcm5000msp-p-14.html.

- Sixteen Raspberry Pi 3 Model B. These raspberries have a Quad Core 1.2GHz Broadcom BCM2837 64bit CPU, with 1GB RAM, one gigabit ethernet port, one wireless LAN and Bluetooth Low Energy (BLE) on board. The raspberries run Raspbian Jessie OS. More information can be found at https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

- Eight LG Nexus 5X smartphones. More details about the hardware and software specifications can be found at https://www.gsmarena.com/lg_nexus_5x-7556.php.

- One OpenFlow switch Pica8 model P-3297. This switch is a 1 RU low-latency, high performance Ethernet switch with 48 ports 1Gigabit 1000Base-T and 4 slots SFP 10 Gigabit. It is equipped with Triumph 2 switch ASIC with extended TCAM. The switch comes pre-loaded with PicOS software for full Layer-2, Layer-3, and OpenFlow 1.4. It can handle 4096 VLANs, 32k MAC addresses, 12k routes, 8k MPLS labels. It implements Open-vSwitch (OVS) 2.0 and provides MPLS over OVS. More information can be seen at http://www.pica8.com/wp-content/uploads/2015/09/pica8-datasheet-48x1gbe-p3297.pdf

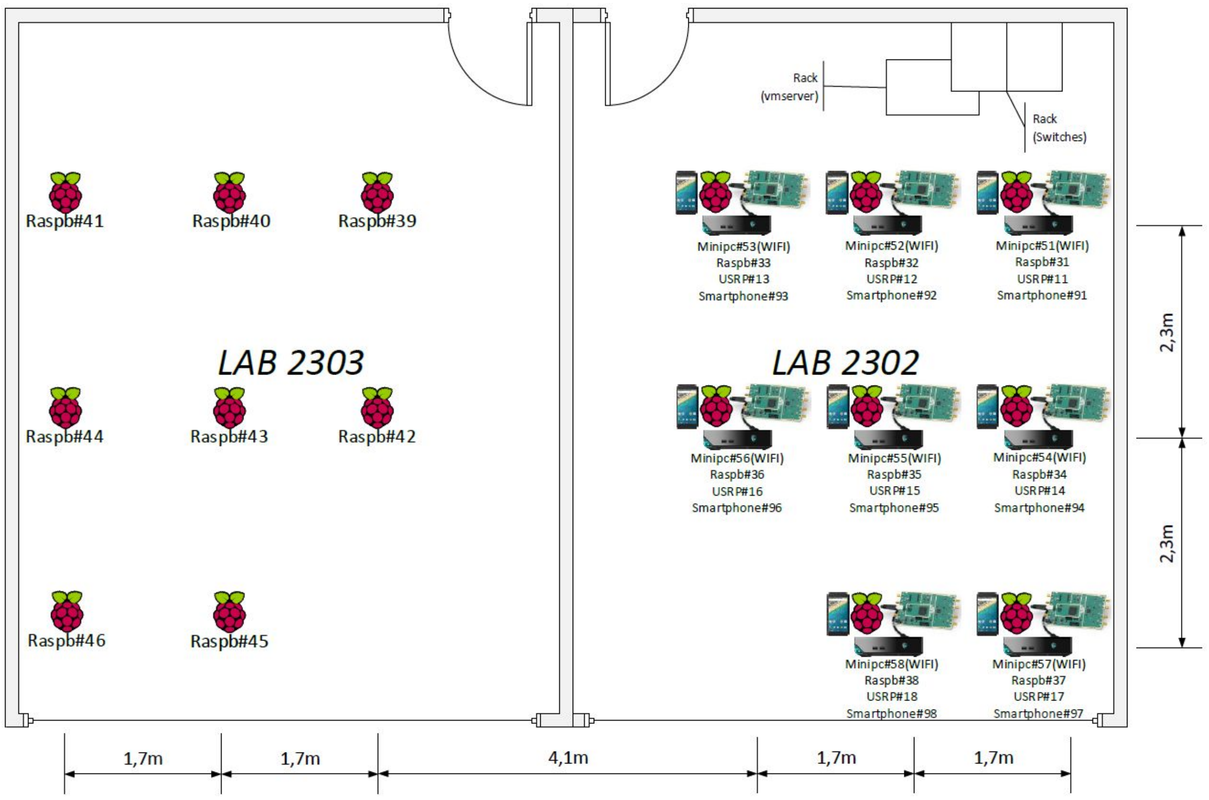
- One DELL server model PowerEdge R430 used as virtualization server, equipped with one Intel Xeon 8 cores 5-2609 1.7 GHz, 8GB RAM, 1TB SATA Hot-plug Hard Drive, 4 ethernet 1 gigabit NIC. This server runs Ubuntu 16.04.2 LTS linux with KVM virtualization platform.

The switches and the virtualization server are indirectly involved in the experiments, being used in the infrastructure that supports the testbed. The MiniPCs, USRPs, Raspberries and Smartphones are placed on the ceiling in two laboratories in UFMG as shown in the Figure below. Figure 1 also shows the placement of the switches and virtualization server.

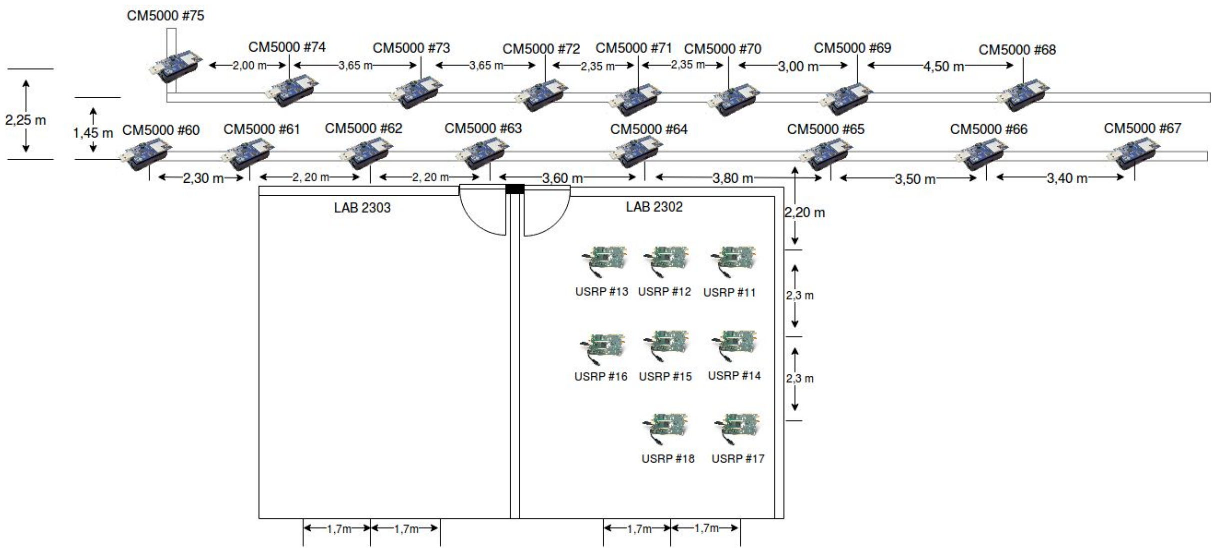
2.2 - Maps of the Testbed

The resources are placed in the testbed as indicated in the following maps:

[1] In order to provide more accurate timings for the SDR experiments, the hardware is fitted with GPS-disciplined oscillators (GPSDO).


Rack (vmserver)
Rack (Switches)
Raspb#41
Raspb#40
Raspb#39
LAB 2303
Raspb#44
Raspb#43
Raspb#42
Raspb#46
Raspb#45
Minipc#53(WIFI) Raspb#33 USRP#13 Smartphone#93
Minipc#52(WIFI) Raspb#32 USRP#12 Smartphone#92
Minipc#51(WIFI) Raspb#31 USRP#11 Smartphone#91
LAB 2302
Minipc#56(WIFI) Raspb#36 USRP#16 Smartphone#96
Minipc#55(WIFI) Raspb#35 USRP#15 Smartphone#95
Minipc#54(WIFI) Raspb#34 USRP#14 Smartphone#94
Minipc#58(WIFI) Raspb#38 USRP#18 Smartphone#98
Minipc#57(WIFI) Raspb#37 USRP#17 Smartphone#97
2,3m
2,3m
1,7m
1,7m
4,1m
1,7m
1,7m

Map of the placement of the USRPs, mini PCs, Raspberry Pis and Smartphones



Map of the placement of the Advanticsys Sensor Nodes

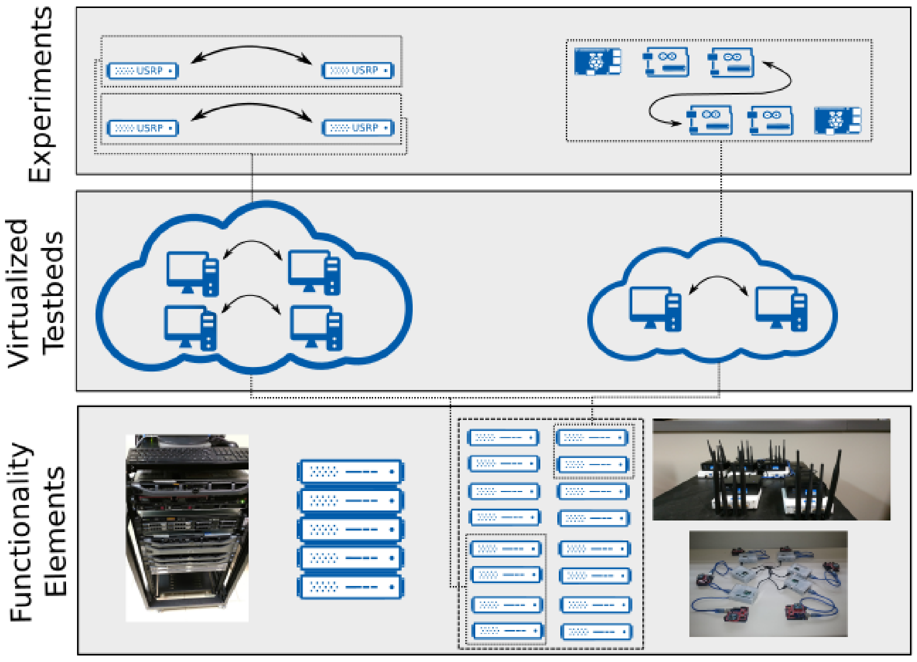
By definition, during the allocation, the choice of the equipment used by a resource is made in a random form. That way, the user have no power about the physical location of the equipment used. If a user want to allocate a resource in a specific equipment, he/she can use the attribute **component id** on the tag **node** in the RSpec. The following example shows the allocation of a specific USRP:

```
<node client_id="node1" exclusive="false"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
component_id="urn:publicid:IDN+futebol.dcc.ufmg.br+node+11">
    <sliver_type name="usrp-vm">
      <disk_image
name="urn:publicid:IDN+futebol.dcc.ufmg.br+image+gnuradio"/>
    </sliver_type>
</node>
```

The number that the attribute **component id** are given as param represents the id of the resource, following the organization of testbed map.

2.3 - Functional Layers of the Testbed

Logically, one can think of the testbed as consisting of four layers: The bottom layer provides the physical elements, such as servers, raspberries, arduino, XBees, storage, etc., that can be controlled through one hypervisors. The next layer corresponds to the virtualized testbed, comprising containers/VM associated to the physical devices. Finally, at the top sits the definition of each experiment that uses the resources provided by the lower layer.



Functional layers

2.4 - Setting Up an Experiment

Users will be able to setup an experiment using jFed. After the user sends the login information, the AM will authenticate the user, tell him/her which resources will be available to them through RSpecs, and interact with the CBTM on behalf of the user in order to instantiate the available resources. The AM uses the GENI v3 API which is written as a wrapper of the reference AM.

3. Experiments with Raspberry Pi

The Raspberries PI are single-board computers, used in this testbed to simulate Access Points, Wireless Stations or plain computers. This section describes how to allocate a Raspberry PI and how to run simple experiments to check the operability of the devices.

3.1 - Raspberry Pi RSpec Description

The Raspberry Pi nodes can be allocated as Access Points, Wireless Stations or Without Both Configurations. Note that the raspberry Pi is selected by using the sliver type **raw-raspberry**. The example below presents an RSPEC for a Raspberry Pi mote operating as an Access Point:

```
<node client_id="node1" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
>
   <sliver_type name="raw-raspberry" mode="ap">
       <wifi-settings ssid="eee" psk="iii" auth="jjj" channel="www"
tx_power="xxx" sens="yyy" rts="zzz"/>
   </sliver_type>
 </node>
```

The example below shows an RSPEC for allocating a Raspberry Pi as a Wireless Station (note that the mode of the Wi-Fi card is defined by the mode parameter of the sliver, being either ap or sta):

```
<node client_id="node1" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
>
   <sliver_type name="raw-raspberry" mode="sta">
       <wifi-settings ssid="eee" psk="iii" auth="jjj" tx_power="xxx"
sens="yyy"/>
   </sliver_type>
 </node>
```

The **wifi-settings** tag holds the configuration items for the Wireless interface. The options in this tag are:

- **ssid:** Defines the SSID for the network

- **psk:** Defines the password for the network
- **auth:** Defines the authentication type for the network, should be one of the following:
    - WEP/WEP2
    - WPA-PSK/WPA2-PSK
- **channel:** Defines in which channel the network is going to be distributed
- **tx_power:** Defines the transmission power for the antenna (in dBm)
    - "**auto**" for automatically chosen transmission power
    - A numeric (integer) value
- **sens:** set the sensitivity threshold
    - "**auto**" for automatically chosen sensitivity threshold
    - A numeric (integer) value.
- **rts:** adds a RTS/CTS handshake before each packet transmission to make sure that the channel is clear.The Arduino is programmed through raspberry pi terminal or IDE where the experimenter can upload the arduino software to manage the sensors. The sensor are plugged on pins X, Y and Z for luminosity, humidity and temperature, respectively. A default arduino software is provided with sensor data update period of W ms.
    - "**off**" for no RTS
    - A numeric (integer) value

The example below shows an RSPEC for allocating the Raspberry Pi without configuring its Wi-Fi interface:

```
<node client_id="node1" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
>
   <sliver_type name="raw-raspberry">
   </sliver_type>
</node>
```

3.2 - Examples of Raspberry Pi Experiments

3.2.1 - Plain Raspberry Pi

The example below show a complete RSpec to allocate a plain Raspberry Pi.

```
<?xml version='1.0'?>
<rspec xmlns="http://www.geni.net/resources/rspec/3" type="request"
generated_by="jFed RSpec Editor"
generated="2017-11-10T09:40:59.688-02:00"
xmlns:emulab="http://www.protogeni.net/resources/rspec/ext/emulab/1"
xmlns:delay="http://www.protogeni.net/resources/rspec/ext/delay/1"
xmlns:jfed-command="http://jfed.iminds.be/rspec/ext/jfed-command/1"
xmlns:client="http://www.protogeni.net/resources/rspec/ext/client/1"
xmlns:jfed-ssh-keys="http://jfed.iminds.be/rspec/ext/jfed-ssh-keys/1"
xmlns:jfed="http://jfed.iminds.be/rspec/ext/jfed/1"
xmlns:sharedvlan="http://www.protogeni.net/resources/rspec/ext/shared-vl
an/1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.geni.net/resources/rspec/3
http://www.geni.net/resources/rspec/3/request.xsd ">
  <node client_id="node0" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
>
      <sliver_type name="raw-raspberry">
      </sliver_type>
      <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="282.0"
y="109.5"/>
  </node>
</rspec>
```

After the allocation of the Raspberry Pi node is done, the device can be tested using the simple code below:

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    printf("Hello World!\n");
    return 0;
}
```

The example code can be downloaded, compiled and executed as follows:

```
wget futebol.dcc.ufmg.br/documentation/examples/helloword_raspberrypi.c
gcc -c helloword_raspberrypi.c
gcc helloworld -o helloword_raspberrypi.o
./helloworld
```

TERMINAL OUTPUT

```
Hello World!
```

3.2.2 - Raspberry Pi as Access Points and Wireless Stations

The following example shows a complete RSpec to allocate **one** Raspberry Pi as **Access Point** and **one** Raspberry Pi as a **Wireless Station**:

```
<?xml version='1.0'?>
<rspec xmlns="http://www.geni.net/resources/rspec/3" type="request"
generated_by="jFed RSpec Editor"
generated="2017-11-10T09:40:59.688-02:00"
xmlns:emulab="http://www.protogeni.net/resources/rspec/ext/emulab/1"
xmlns:delay="http://www.protogeni.net/resources/rspec/ext/delay/1"
xmlns:jfed-command="http://jfed.iminds.be/rspec/ext/jfed-command/1"
xmlns:client="http://www.protogeni.net/resources/rspec/ext/client/1"
xmlns:jfed-ssh-keys="http://jfed.iminds.be/rspec/ext/jfed-ssh-keys/1"
xmlns:jfed="http://jfed.iminds.be/rspec/ext/jfed/1"
xmlns:sharedvlan="http://www.protogeni.net/resources/rspec/ext/shared-vl
an/1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.geni.net/resources/rspec/3
http://www.geni.net/resources/rspec/3/request.xsd ">
  <node client_id="node0" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
>
<node client_id="node0" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
>
   <sliver_type name="raw-raspberry" mode="ap">
       <wifi-settings ssid="test" psk="password" auth="wpa" channel="7"
tx_power="auto" sens="auto" rts="off"/>
   </sliver_type>
 </node>
<node client_id="node1" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
>
   <sliver_type name="raw-raspberry" mode="sta">
       <wifi-settings ssid="test" psk="password" auth="wpa"
tx_power="auto" sens="auto"/>
   </sliver_type>
 </node>
       </node>
</rspec>
```

When both allocations of the Raspberries Pi are done, the user is able to connect the wireless station to the access point by changing the file /etc/wpa_supplicant/wpa_supplicant.conf in the station. The user should add the following lines in the file:

```
network={
      ssid="test"
      psk="password"
      key_mgmt=WPA-PSK
}
```

After, also in the wireless station, the wpa_supplicant service must be started with the command below:

```
 sudo wpa_supplicant -Dnl80211 -c/etc/wpa_supplicant/wpa_supplicant.conf
-iwlan0 -dt
```

Once this step is executed, it will be possible to verify the connection between the station and the access point using NetCat, by using the following command to open a port in the access point:

```
user@<access point> nc -l -vvv -p 5000
```

Next, use the command below in the station to establish the communication with that port:

```
user@<wireless station> nc -vvv 192.168.0.1 (access point IP) 5000
```

TERMINAL OUTPUT

If the allocation and configuration processes are performed successfully, all that is typed in the wireless station terminal will be shown the access point terminal and vice versa.

4 - Experiments with Wi-Fi

It is possible to use the miniPCs to perform experiments using Wi-Fi. The Mini PCs use a Dell Dual-band Wireless AC 8620 chipset, which has 2x2 MIMO and supports IEEE 802.11 a, b, g, n and ac standards. The Raspberry Pis can also be used for Wi-Fi experiments (please check the appropriate section), however the Mini PCs have a much wider configuration range.

Besides using the command line tools to configure the Wi-Fi, it is also possible to use a SDN interface called Ethanol to control them. Ethanol is able to control a number of parameters of the wireless interface from a SDN controller. Those can be used in SDN-based experiments, or as a tool to control the Wi-Fi links in your experiment. For example, you can force a node disconnect using Ethanol, to emulate a node failure. The full documentation of the Ethanol API can be found in github: https://github.com/h3dema/ethanol_controller/tree/master/ethanol/documentation

4.1 - RSpec Description

The Wi-Fi nodes can be allocated as a machine with wireless network card. Note that the Wi-Fi node is selected by using the sliver type **raw-wifi**. The example below presents an RSPEC for Wi-Fi node, using Ubuntu 16.04:

```
<node client_id="node0" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am">
      <sliver_type name="raw-wifi">
       <disk_image
name="urn:publicid:IDN+futebol.dcc.ufmg.br+image+ubuntu16"/>
      </sliver_type>
      <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="176.0"
y="117.0"/>
</node>
```

The example below shows an RSPEC for allocating the Wi-Fi node using Ethanol:

```
<node client_id="node0" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
>
      <sliver_type name="raw-wifi">
      <disk_image
name="urn:publicid:IDN+futebol.dcc.ufmg.br+image+ethanol"/>
      </sliver_type>
      <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="50.0"
y="544.7097400940887"/>
  </node>
```

4.2 - Experiment Examples

4.2.1 - Wi-Fi Node without Ethanol

Below shows an RSPEC for allocating the Wi-Fi node for this experiment:

```
<?xml version='1.0'?>
<rspec xmlns="http://www.geni.net/resources/rspec/3" type="request"
generated_by="jFed RSpec Editor"
generated="2017-09-18T15:37:48.662-03:00"
xmlns:emulab="http://www.protogeni.net/resources/rspec/ext/emulab/1"
xmlns:delay="http://www.protogeni.net/resources/rspec/ext/delay/1"
xmlns:jfed-command="http://jfed.iminds.be/rspec/ext/jfed-command/1"
xmlns:client="http://www.protogeni.net/resources/rspec/ext/client/1"
xmlns:jfed-ssh-keys="http://jfed.iminds.be/rspec/ext/jfed-ssh-keys/1"
xmlns:jfed="http://jfed.iminds.be/rspec/ext/jfed/1"
xmlns:sharedvlan="http://www.protogeni.net/resources/rspec/ext/shared-vl
an/1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.geni.net/resources/rspec/3
http://www.geni.net/resources/rspec/3/request.xsd ">
  <node client_id="node0" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
>
      <sliver_type name="raw-wifi">
       <disk_image
name="urn:publicid:IDN+futebol.dcc.ufmg.br+image+ubuntu16"/>
      </sliver_type>
      <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="176.0"
y="117.0"/>
  </node>
  <node client_id="node1" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
>
      <sliver_type name="raw-wifi">
       <disk_image
name="urn:publicid:IDN+futebol.dcc.ufmg.br+image+ubuntu16"/>
      </sliver_type>
      <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="539.0"
y="106.0"/>
  </node>
  <node client_id="node2" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
```

```
>
      <sliver_type name="raw-wifi">
       <disk_image
name="urn:publicid:IDN+futebol.dcc.ufmg.br+image+ubuntu16"/>
      </sliver_type>
      <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="358.0"
y="257.0"/>
  </node>
</rspec>
```

After allocating the machines with wifi, follow the steps below:

1. Define the IP addresses so that the hosts are on the same network.

```
root@<node 0>Ifconfig wlp2s0 192.168.0.1 netmask 255.255.255.0 up
root@<node 1>Ifconfig wlp2s0 192.168.0.10 netmask 255.255.255.0 up
root@<node 2>Ifconfig wlp2s0 192.168.0.100 netmask 255.255.255.0 up
```

2. Using hostAPd it is possible to create an access point using a configuration file, for example named **hostapd.conf,** such as the one below:

```
interface=wlp2s0
hw_mode=a
channel=149
ieee80211d=1
country_code=BR
ieee80211n=1
ieee80211ac=1
wmm_enabled=1
ssid=test
```

3. Then run hostapd on the AP machine with this file

```
hostapd hostapd.conf
```

4. Next, establish a connection between the hosts and the access point.

iwconfig wlp2s0 essid test mode managed

5. You can test the performance with the Iperf tool, which comes pre-installed in the image. In the server, run:

```
iperf -s
```

In the client, please run:

```
iperf -c 192.168.0.100 (server_IP) -d
```

4.2.2 - Wi-Fi Node with Ethanol

In this example we will make a simple test, in which the Ethanol controller denies a clinet with a certain MAC address to associate in the Wi-Fi network. We show below an RSPEC that allocates two Wi-Fi nodes running Ethanol. This is done by selecting the **ethanol** image on a **wifi** resource.

```
<?xml version='1.0'?>
<rspec xmlns="http://www.geni.net/resources/rspec/3" type="request"
generated_by="jFed RSpec Editor"
generated="2017-11-13T09:04:35.837-02:00"
xmlns:emulab="http://www.protogeni.net/resources/rspec/ext/emulab/1"
xmlns:delay="http://www.protogeni.net/resources/rspec/ext/delay/1"
xmlns:jfed-command="http://jfed.iminds.be/rspec/ext/jfed-command/1"
xmlns:client="http://www.protogeni.net/resources/rspec/ext/client/1"
xmlns:jfed-ssh-keys="http://jfed.iminds.be/rspec/ext/jfed-ssh-keys/1"
xmlns:jfed="http://jfed.iminds.be/rspec/ext/jfed/1"
xmlns:sharedvlan="http://www.protogeni.net/resources/rspec/ext/shared-vl
an/1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.geni.net/resources/rspec/3
http://www.geni.net/resources/rspec/3/request.xsd ">
  <node client_id="node0" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
>
      <sliver_type name="raw-wifi">
      <disk_image
name="urn:publicid:IDN+futebol.dcc.ufmg.br+image+ethanol"/>
      </sliver_type>
      <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="50.0"
y="544.7097400940887"/>
  </node>
  <node client_id="node1" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
>
      <sliver_type name="raw-wifi">
      <disk_image
name="urn:publicid:IDN+futebol.dcc.ufmg.br+image+ethanol"/>
```

```
      </sliver_type>
      <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1"
x="859.0277178762544" y="25.0"/>
  </node>
</rspec>
```

After allocating the Wi-Fi node with configuring its Ethanol, we must configure it properly. In the first part of the experiment, we must modify the Ethanol controller so that it denies the connection from a certain client:

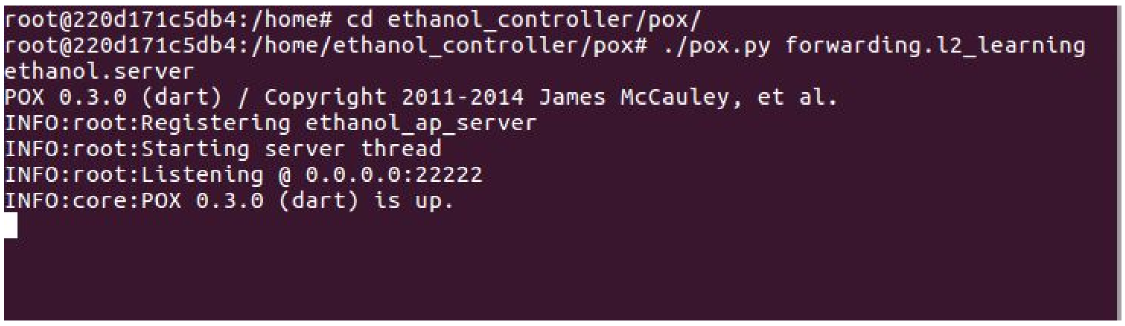
```
cd /home/ethanol_controller/ethanol/ethanol
vim vap.py
```

Modify this function:

```
 def evUserAssociating(mac_station):
        if mac_station == "e4:a7:a0:4e:f0:ca":
           return False
        else:
           return True
```

The AP will not accept the association of the station with MAC address **"e4:a7:a0:4e:f0:ca".** Type the following command to initialize the Ethanol controller:

```
cd /home/ethanol_controller/pox
./pox.py ethanol.server
```



```
root@220d171c5db4:/home# cd ethanol_controller/pox/
root@220d171c5db4:/home/ethanol_controller/pox# ./pox.py forwarding.l2_learning
ethanol.server
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al.
INFO:root:Registering ethanol_ap_server
INFO:root:Starting server thread
INFO:root:Listening @ 0.0.0.0:22222
INFO:core:POX 0.3.0 (dart) is up.
```

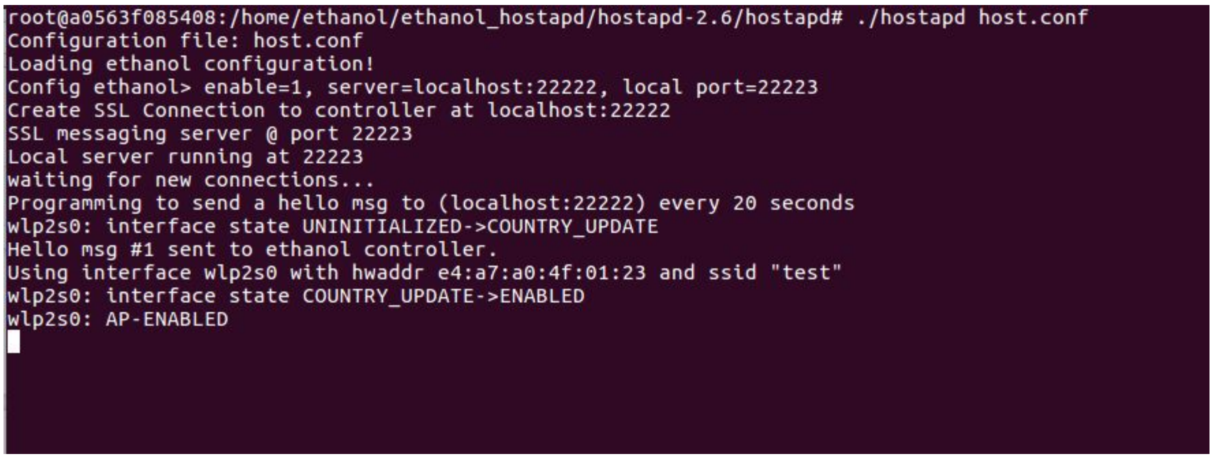
Initializing the Ethanol AP: Here is a sample configuration file for HostAPD (you can create it using the command "**vim /hostapd-2.6/hostapd/host.conf"**

```
interface=wlp2s0
hw_mode=a
channel=149
```

```
ieee80211d=1
country_code=BR
ieee80211n=1
ieee80211ac=1
wmm_enabled=1
ssid=test
```

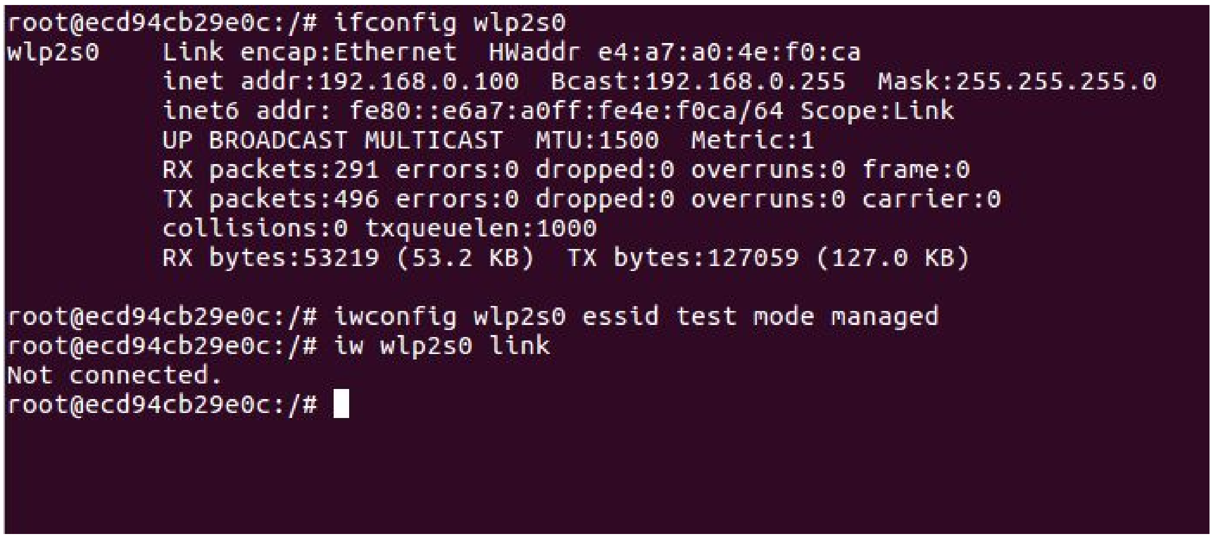
Copy the file mycert.pem into the /hostapd-2.6 and /hostapd-2.6/hostapd directories and the file ethanol.ini into the /etc/ directory using the following commands:

```
cd ethanol_hostapd/certificate
cp mycert.pem ethanol_hostapd/hostapd-2.6
cp ethanol_hostapd/hostapd-2.6/hostapd
cd ..
cd src/ini
cp ethanol.ini /etc/ethanol.ini
cd ../../hostapd
make clean
make ethanol
./hostapd host.conf
```



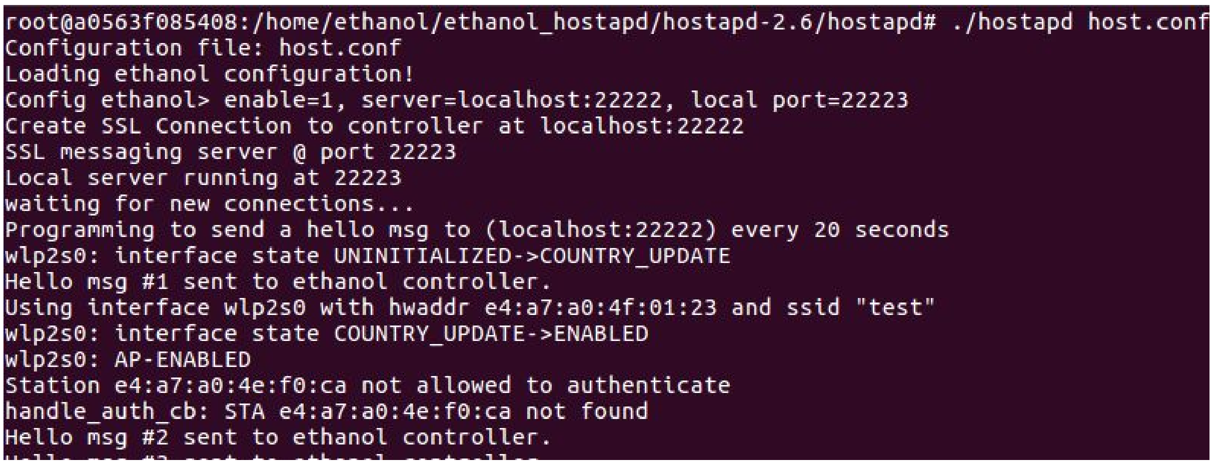
```
root@a0563f085408:/home/ethanol/ethanol_hostapd/hostapd-2.6/hostapd# ./hostapd host.conf
Configuration file: host.conf
Loading ethanol configuration!
Config ethanol> enable=1, server=localhost:22222, local port=22223
Create SSL Connection to controller at localhost:22222
SSL messaging server @ port 22223
Local server running at 22223
waiting for new connections...
Programming to send a hello msg to (localhost:22222) every 20 seconds
wlp2s0: interface state UNINITIALIZED->COUNTRY_UPDATE
Hello msg #1 sent to ethanol controller.
Using interface wlp2s0 with hwaddr e4:a7:a0:4f:01:23 and ssid "test"
wlp2s0: interface state COUNTRY_UPDATE->ENABLED
wlp2s0: AP-ENABLED
```

The station with the MAC address is blocked when trying to connect:



```
root@ecd94cb29e0c:/# ifconfig wlp2s0
wlp2s0    Link encap:Ethernet  HWaddr e4:a7:a0:4e:f0:ca
          inet addr:192.168.0.100  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::e6a7:a0ff:fe4e:f0ca/64 Scope:Link
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:291 errors:0 dropped:0 overruns:0 frame:0
          TX packets:496 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:53219 (53.2 KB)  TX bytes:127059 (127.0 KB)

root@ecd94cb29e0c:/# iwconfig wlp2s0 essid test mode managed
root@ecd94cb29e0c:/# iw wlp2s0 link
Not connected.
root@ecd94cb29e0c:/#
```

The logs on the AP show that is does not accept the connection of the station with MAC address **"e4:a7:a0:4e:f0:ca"**:

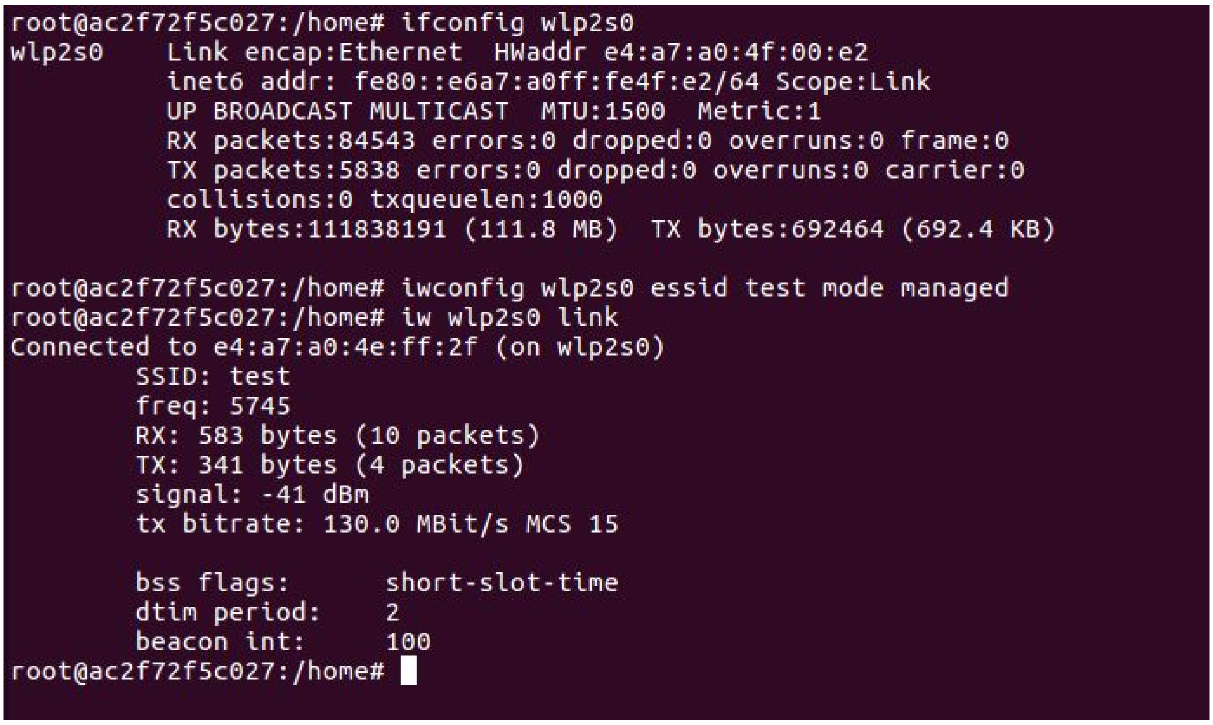


```
root@a0563f085408:/home/ethanol/ethanol_hostapd/hostapd-2.6/hostapd# ./hostapd host.conf
Configuration file: host.conf
Loading ethanol configuration!
Config ethanol> enable=1, server=localhost:22222, local port=22223
Create SSL Connection to controller at localhost:22222
SSL messaging server @ port 22223
Local server running at 22223
waiting for new connections...
Programming to send a hello msg to (localhost:22222) every 20 seconds
wlp2s0: interface state UNINITIALIZED->COUNTRY_UPDATE
Hello msg #1 sent to ethanol controller.
Using interface wlp2s0 with hwaddr e4:a7:a0:4f:01:23 and ssid "test"
wlp2s0: interface state COUNTRY_UPDATE->ENABLED
wlp2s0: AP-ENABLED
Station e4:a7:a0:4e:f0:ca not allowed to authenticate
handle_auth_cb: STA e4:a7:a0:4e:f0:ca not found
Hello msg #2 sent to ethanol controller.
```

Then, we can modify the controller code to accept connections from any station by changing the evUserAssociating function to the following code.

```
def evUserAssociating(mac_station):
           return True
```

The screenshot below shows that the station is now able to connect.



```
root@ac2f72f5c027:/home# ifconfig wlp2s0
wlp2s0    Link encap:Ethernet  HWaddr e4:a7:a0:4f:00:e2
          inet6 addr: fe80::e6a7:a0ff:fe4f:e2/64 Scope:Link
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:84543 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5838 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:111838191 (111.8 MB)  TX bytes:692464 (692.4 KB)

root@ac2f72f5c027:/home# iwconfig wlp2s0 essid test mode managed
root@ac2f72f5c027:/home# iw wlp2s0 link
Connected to e4:a7:a0:4e:ff:2f (on wlp2s0)
        SSID: test
        freq: 5745
        RX: 583 bytes (10 packets)
        TX: 341 bytes (4 packets)
        signal: -41 dBm
        tx bitrate: 130.0 MBit/s MCS 15

        bss flags:      short-slot-time
        dtim period:    2
        beacon int:     100
root@ac2f72f5c027:/home# 
```

For more uses of Ethanol see **https://github.com/h3dema/ethanol_devel**

5 - Experiments with USRP

The USRP (Universal Software Radio Peripheral) is a software-defined radio, which is able to run arbitrary code using an FPGA as well as on the CPU of the server. It can be used to run any wireless standard (including existing standards or even a protocol that has been created by the experimenter), as long as the operating frequency and other parameters of the communication are within the accepted range of the USRP card. This section describes how to book an USRP and how to run a very simple experiment that checks for the spectrum usage on a certain frequency.

5.1 - RSpec Description

The USRP nodes are allocated as virtual machines containing USRP boards attached via USB. Each MiniPC on the testbed can support one USRP node at a time. The RSPEC for allocating an USRP node must contain the following tags:

```
<node client_id="node1" exclusive="false"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
>
    <sliver_type name="usrp-vm">
      <disk_image
```

```
name="urn:publicid:IDN+futebol.dcc.ufmg.br+image+IMG_TYPE"/>
    </sliver_type>
</node>
```

The bold part inside of the “node” tag indicates that this node is being allocated in the UFMG FUTEBOL testbed. It is automatically filled with this information when the UFMG testbed is selected on the dropdown menu on jFed. The name component on the “sliver_type” tag indicates that this node is composed of a virtual machine containing an USRP board attached to it via USB. Inside the “sliver_type” tag is the “disk_image” tag. This tag selects the type of virtual machine that will be allocated. The “name” component inside this tag indicates which type of virtual machine is being allocated, and also reinforces that this machine is being allocated inside of the UFMG Testbed. The string on the “name” component must follow the structure presented above. The IMG_TYPE can be one of the following:

- **gnuradio**: Selects a machine with the basic gnuradio software installed
- **openlte**: Selects a machine with the basic gnuradio software and the OpenLTE software installed

The “disk_image” tag is **optional**. When the “sliver_type” selected is “usrp-vm”, the default image selected for the machine is “**gnuradio”.**

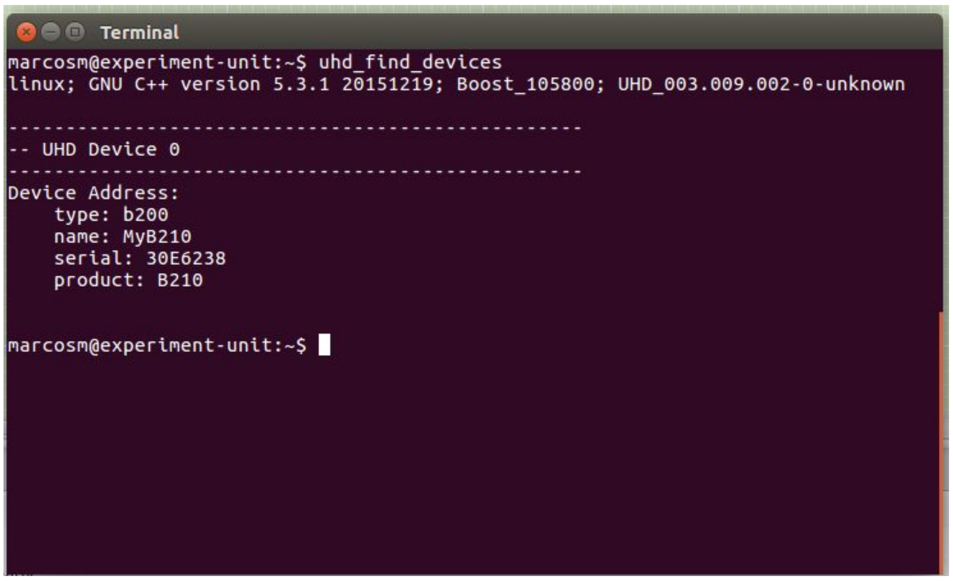
5.2 - Experiment Example

In this experiment we are going to run a spectrum analyser, which will use the USRP to show how is the usage of the medium on a certain frequency. To that end, we will instantiate a USRP resource on the UFMG FUTEBOL testbed. The following RSPEC can be used to instantiate a single VM with a USRP:

```
<?xml version='1.0'?>
<rspec xmlns="http://www.geni.net/resources/rspec/3" type="request"
generated_by="jFed RSpec Editor"
generated="2017-06-09T13:54:59.535-03:00"
xmlns:emulab="http://www.protogeni.net/resources/rspec/ext/emulab/1"
xmlns:jfedBonfire="http://jfed.iminds.be/rspec/ext/jfed-bonfire/1"
xmlns:delay="http://www.protogeni.net/resources/rspec/ext/delay/1"
xmlns:jfed-command="http://jfed.iminds.be/rspec/ext/jfed-command/1"
xmlns:client="http://www.protogeni.net/resources/rspec/ext/client/1"
xmlns:jfed-ssh-keys="http://jfed.iminds.be/rspec/ext/jfed-ssh-keys/1"
xmlns:jfed="http://jfed.iminds.be/rspec/ext/jfed/1"
xmlns:sharedvlan="http://www.protogeni.net/resources/rspec/ext/shared-vl
an/1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.geni.net/resources/rspec/3
http://www.geni.net/resources/rspec/3/request.xsd ">
```

```
  <node client_id="node0" exclusive="false"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
>
    <sliver_type name="usrp-vm">
      <disk_image
name="urn:publicid:IDN+futebol.dcc.ufmg.br+image+gnuradio"/>
    </sliver_type>
    <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="396.0"
y="77.5"/>
  </node>
  <node client_id="node1" exclusive="false"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
>
    <sliver_type name="usrp-vm">
      <disk_image
name="urn:publicid:IDN+futebol.dcc.ufmg.br+image+gnuradio"/>
    </sliver_type>
    <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="52.5"
y="76.5"/>
  </node>
</rspec>
```

After the VM is instantiated, you can execute commands to perform experiments in the USRP. The first basic command is to check if the equipment is detected. To do this, run the command **uhd_find_devices.** This command output will indicate the model, serial number and name of the USRP that is connected to the virtual resource that you allocated.

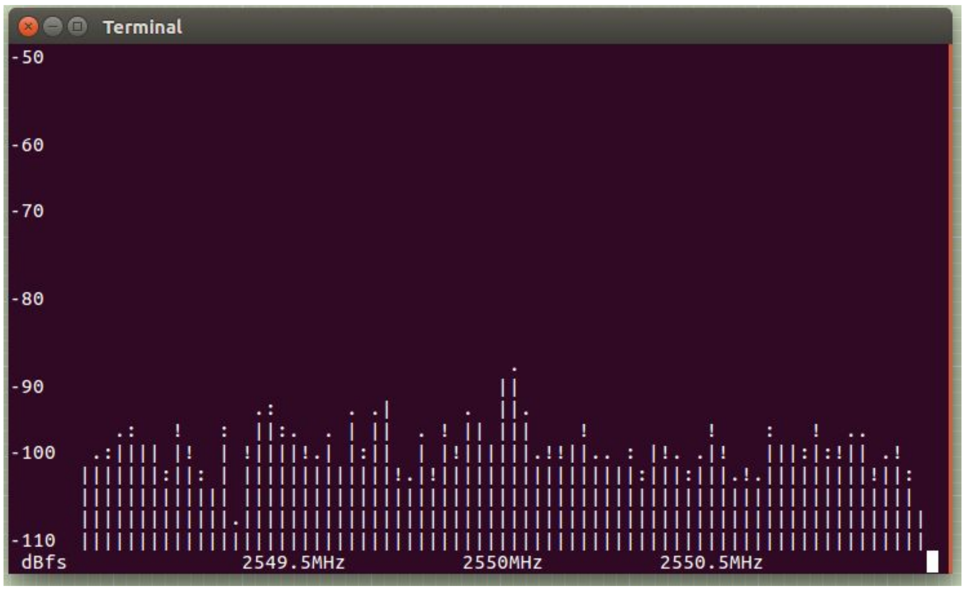


```
marcosm@experiment-unit:~$ uhd_find_devices
linux; GNU C++ version 5.3.1 20151219; Boost_105800; UHD_003.009.002-0-unknown

--------------------------------------------------
-- UHD Device 0
--------------------------------------------------
Device Address:
    type: b200
    name: MyB210
    serial: 30E6238
    product: B210


marcosm@experiment-unit:~$
```

On the ‘rx_node’, run the following code. This code will run a frequency analyzer at 2.55GHz, sampling the medium at a rate of 2MHz. This frequency analyzer runs on the command line (because it may be too slow to open a graphic user interface from the testbed).

```
cd /usr/lib/uhd/examples
./rx_ascii_art_dft --freq 2550000000 --rate 2000000 --ref-lvl -50
```

You should be able to see the ascii representation of the transmitted waveform spectrum, as illustrated below:



6 - Experiments with Advanticsys Sensor Nodes

The Advanticsys MTM-CM5000-MSP is a wireless sensor node with limited processing and network resources. It is recommended for experiments on wireless sensor networks or related to Internet of Things with very resource constrained devices. The nodes transmit wirelessly using the standard IEEE 802.15.4. In order to program the nodes, the programmer must create a binary integrating his/her own code with one operating system, which is either TinyOS or Contiki. The Advanticsys is a node that is compatible to any software developed for the TelosB platform. It has temperature, relative humidity and light sensors. They are connected via an USB interface. More information at https://www.advanticsys.com/shop/mtmcm5000msp-p-14.html.

At the moment, the FUTEBOL UFMG testbed supports programming the Advanticsys using the TinyOS operating system. In the future we plan to support Contiki as well.

6.1 - RSpec Description

Sensor nodes must be allocated together with a virtual machine that contains the system responsible for managing the sensor nodes. To do this, the *sliver_type* attribute is set to **telosb-vm** and *disk_image* as **telosb**. For each instance of the sensor nodes, just configure the *sliver_type* as **raw-telosb**. Note that it is not necessary to use the *disk_image* attribute for the sensor nodes.

You can specify which node to allocate according to the ID shown in Figure 2 by configuring the *component_id* attribute of each node tag with the specific number of the sensor node. The example below shows an RSPEC with one virtual machine (named node 0) and two sensor nodes (60 and 66, named node 1 and node 2)

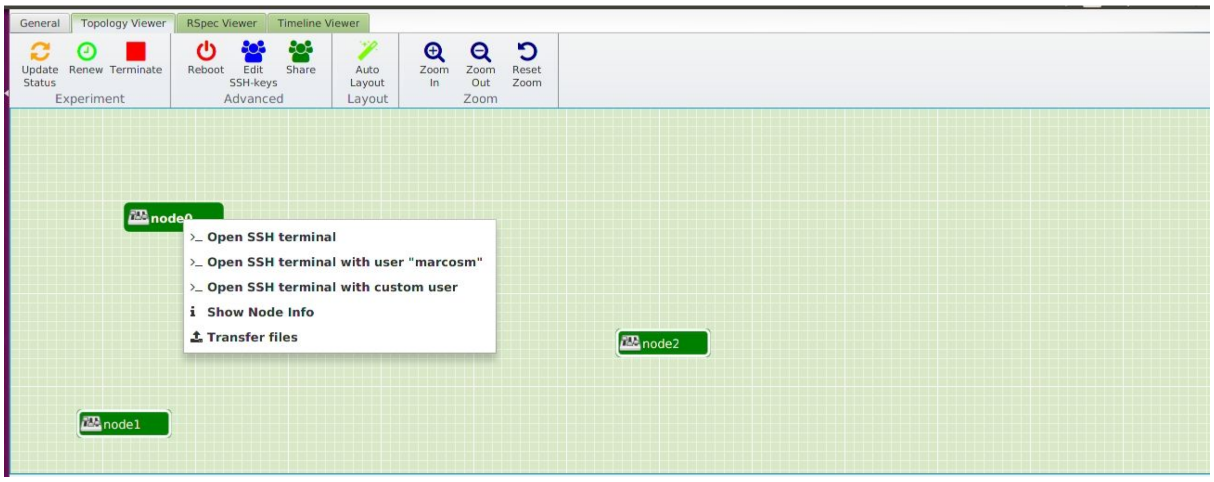
```
<node client_id="node0" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am">
      <sliver_type name="telosb-vm">
       <disk_image name="urn:publicid:IDN+futebol.dcc.ufmg.br+image+telosb"/>
      </sliver_type>
      <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="176.0"
y="117.0"/>
</node>

<node client_id="node1" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
component_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am+node+60">
      <sliver_type name="raw-telosb"/>
      <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="176.0"
y="117.0"/>
</node>

<node client_id="node2" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
component_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am+node+66">
      <sliver_type name="raw-telosb"/>
      <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="176.0"
y="117.0"/>
</node>
```

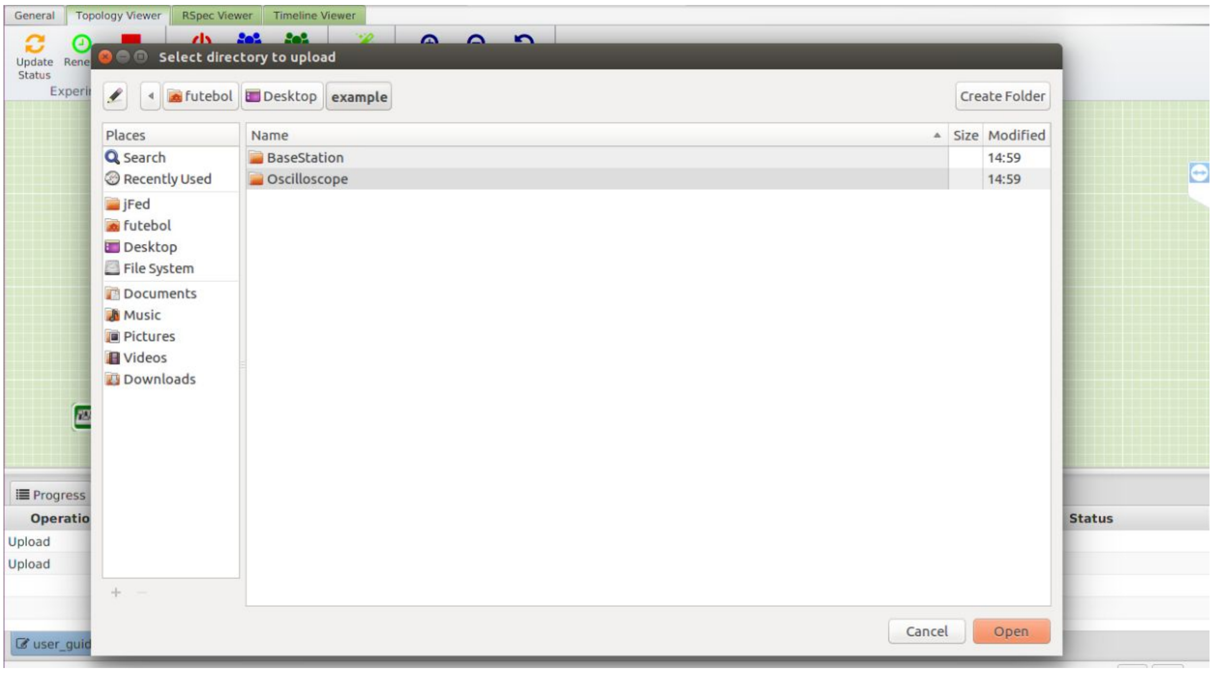
6.2 - Experiment Example

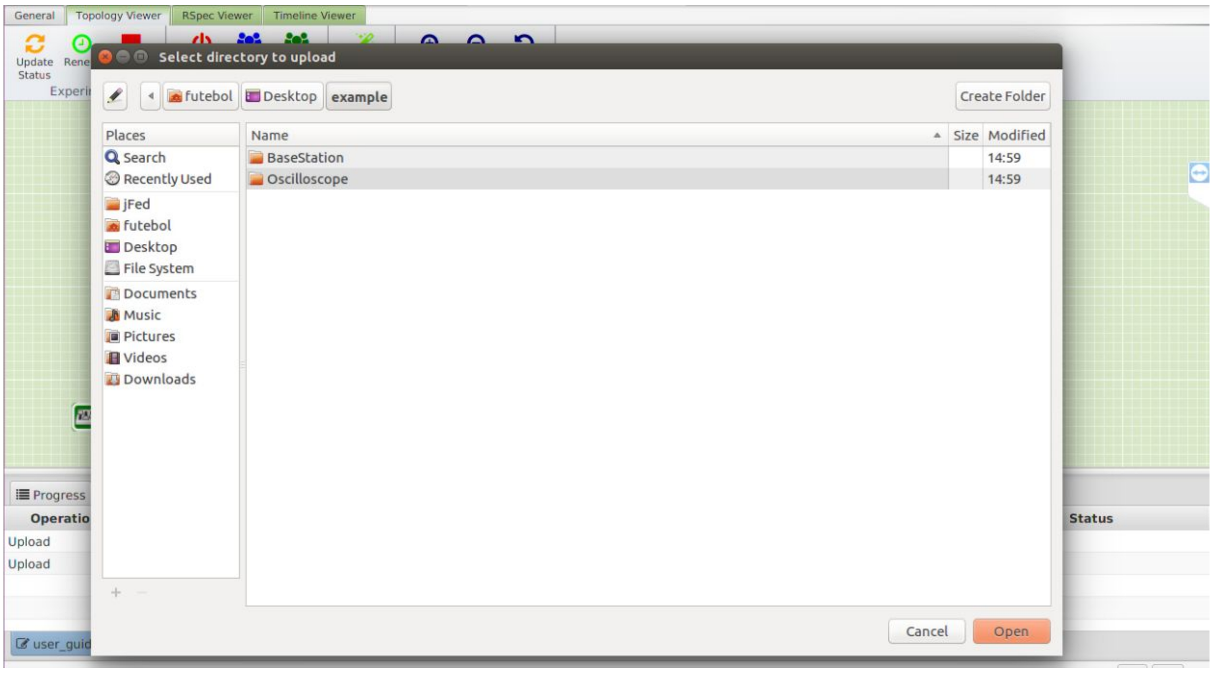
In this experiment we will demonstrate the operation of the applications Oscilloscope and BaseStation which are both examples of the TinyOS Operating System. The first is a simple demo of data collection responsible for periodically the voltage sensor and transmitting on the radio a message with 10 consecutive readings. The second application receives messages from the radio and forwards them to the PC through the USB port. Below we demonstrate the experience, being allocated node 60 for oscilloscope, node 66 for BaseStation and a virtual machine.

1 - Program submission.

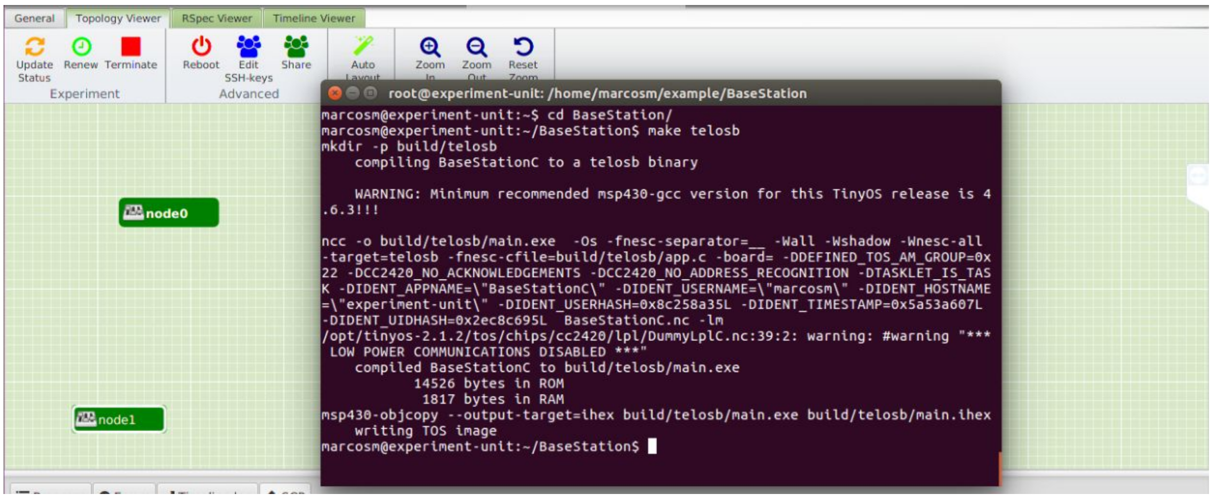
After the nodes are allocated, you must first send your application to the allocated virtual machine, compile it through the TinyOS operating system, and compress the resulting files with a .zip extension. We then send the Oscilloscope and BaseStation folders through the Transfer Files option in Jfed.







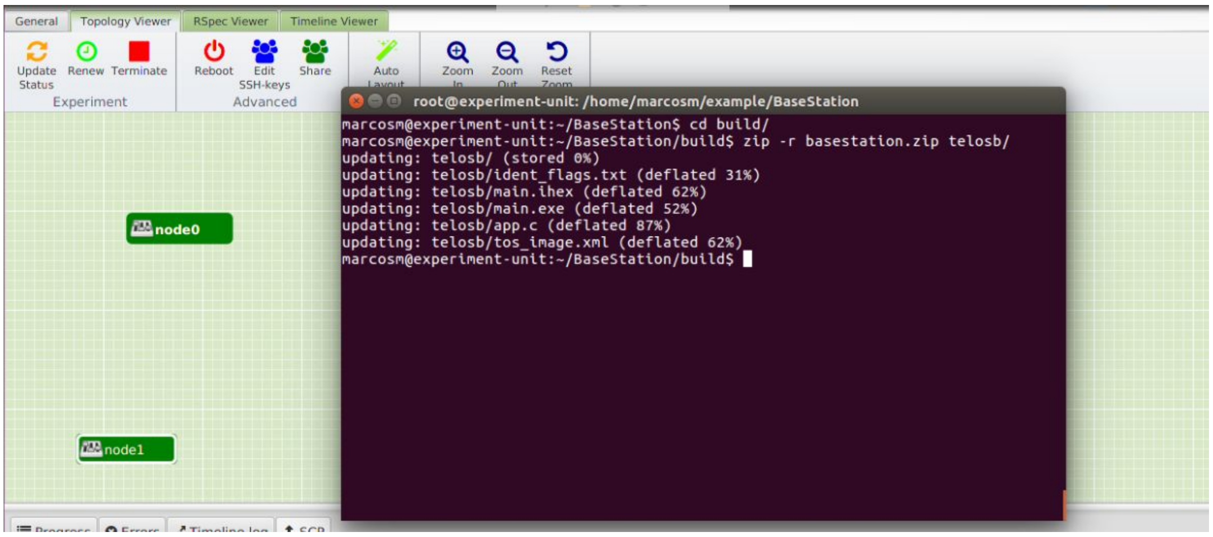
Now, access the virtual machine allocated through the terminal (node 0 in Jfed), navigate to the folder of your application and execute the command **make telosb** to generate the executable that will be installed in advanticsys nodes.



```
root@experiment-unit: /home/marcosm/example/BaseStation
marcosm@experiment-unit:~$ cd BaseStation/
marcosm@experiment-unit:~/BaseStation$ make telosb
mkdir -p build/telosb
    compiling BaseStationC to a telosb binary

    WARNING: Minimum recommended msp430-gcc version for this TinyOS release is 4
.6.3!!!

ncc -o build/telosb/main.exe  -Os -fnesc-separator=__ -Wall -Wshadow -Wnesc-all
-target=telosb -fnesc-cfile=build/telosb/app.c -board= -DDEFINED_TOS_AM_GROUP=0x
22 -DCC2420_NO_ACKNOWLEDGEMENTS -DCC2420_NO_ADDRESS_RECOGNITION -DTASKLET_IS_TAS
K -DIDENT_APPNAME=\"BaseStationC\" -DIDENT_USERNAME=\"marcosm\" -DIDENT_HOSTNAME
=\"experiment-unit\" -DIDENT_USERHASH=0x8c258a35L -DIDENT_TIMESTAMP=0x5a53a607L
-DIDENT_UIDHASH=0x2ec8c695L  BaseStationC.nc -lm
/opt/tinyos-2.1.2/tos/chips/cc2420/lpl/DummyLplC.nc:39:2: warning: #warning "***
 LOW POWER COMMUNICATIONS DISABLED ***"
    compiled BaseStationC to build/telosb/main.exe
           14526 bytes in ROM
            1817 bytes in RAM
msp430-objcopy --output-target=ihex build/telosb/main.exe build/telosb/main.ihex
    writing TOS image
marcosm@experiment-unit:~/BaseStation$ 
```

The build folder will be automatically created, containing the executable and other necessary files. Navigate to it and run the command below to compress the files.

```
cd build/
zip -r [desired name] telosb/
```



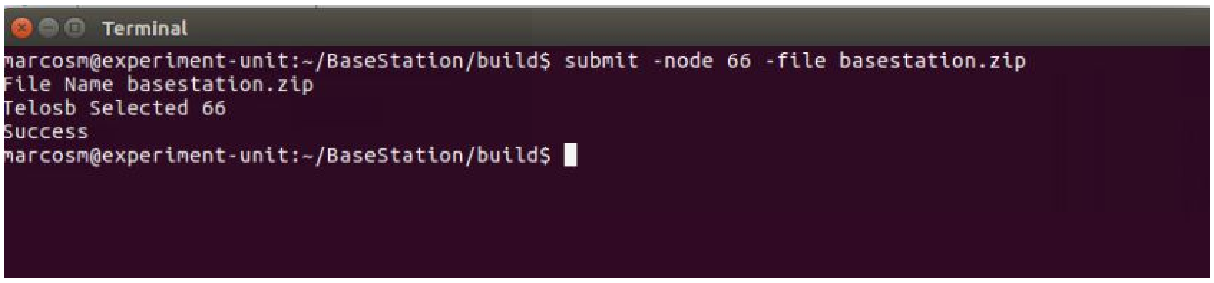
```
root@experiment-unit: /home/marcosm/example/BaseStation
marcosm@experiment-unit:~/BaseStation$ cd build/
marcosm@experiment-unit:~/BaseStation/build$ zip -r basestation.zip telosb/
updating: telosb/ (stored 0%)
updating: telosb/ident_flags.txt (deflated 31%)
updating: telosb/main.ihex (deflated 62%)
updating: telosb/main.exe (deflated 52%)
updating: telosb/app.c (deflated 87%)
updating: telosb/tos_image.xml (deflated 62%)
marcosm@experiment-unit:~/BaseStation/build$ 
```

As an alternative, you can compile[2] and compress the data on your own computer and transfer the .zip file through the Jfed Transfer File function for this, see how to install the TinyOS Operating System on your computer.

2 - Installation of applications on sensor nodes.
To perform the installation of the applications on the sensor nodes, simply run the command below on the terminal of the virtual machine (node 0 in the example allocation).

[2] http://tinyos.stanford.edu/tinyos-wiki/index.php/Getting_Started_with_TinyOS

```
submit - node 60 -file osciloscope.zip
submit - node 66 -file basestatio.zip
```



```
narcosm@experiment-unit:~/BaseStation/build$ submit -node 66 -file basestation.zip
File Name basestation.zip
Telosb Selected 66
Success
narcosm@experiment-unit:~/BaseStation/build$
```

Note that the *-node* and *-file* parameters specify which node to install the application and the file name, respectively.

On your terminal you will receive a success message or some error message. In the event of an error, repeat the installation procedure. On success, the sensor nodes will be working according to their application and the collected data will be saved in text files during the useful period of the experiment.

3 - Download your experiment data.

At any time during your experiment time you can request the data collected through the receiving node, by means of the command below, which should be executed in the terminal of the virtual machine.

```
recv -node 66
```

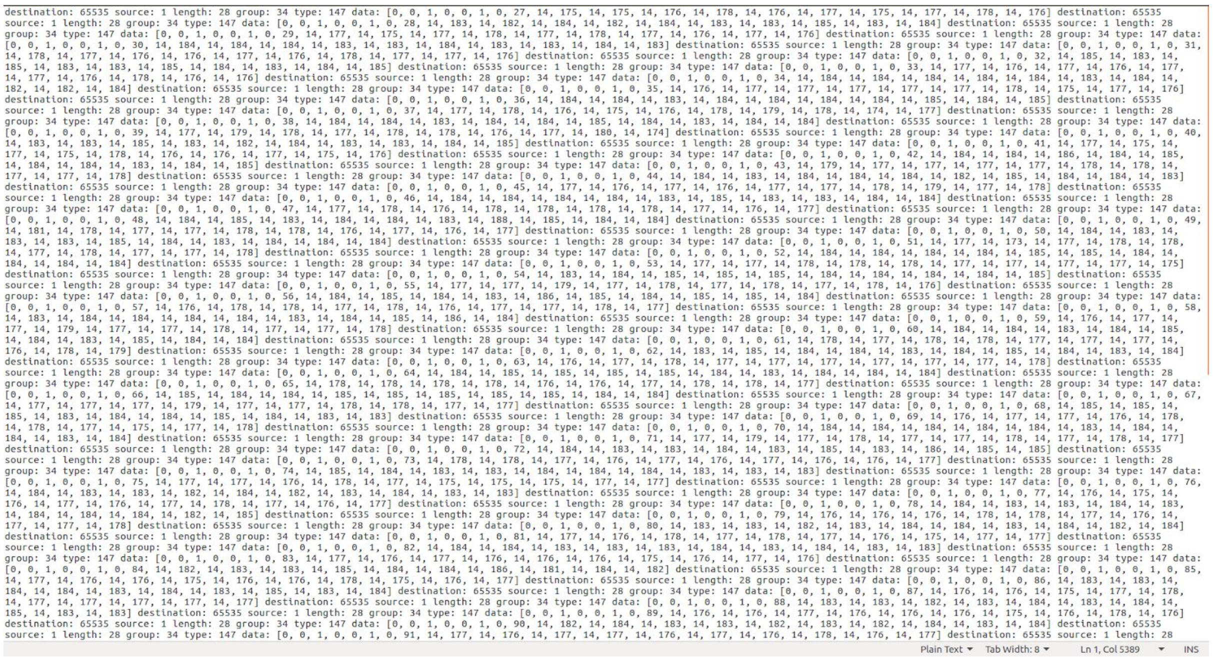
Notice that the recv command calls the procedure for receiving the files, and the *-node* parameter specifies which node you want to receive the data from. The procedure is responsible for sending to the allocated virtual machine a compressed file that contains the data of the experiment. You can view this file in the VM's personal folder or download it through Jfed.

4 - Output file - Experiment data

Your application should be able to write the data collected on the sensor's USB port. Below are the fields that are in the output file for our example experiment

```
destination: [ ] source: [ ] length: [ ] group: [ ] type: [ ] data: [ ]
```

To know more about the fields on this experiment,please visit the official page of tinyos..For our example, see the output file.



```
destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 27, 14, 175, 14, 175, 14, 176, 14, 178, 14, 176, 14, 177, 14, 175, 14, 177, 14, 178, 14, 176] destination: 65535
source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 28, 14, 183, 14, 182, 14, 184, 14, 182, 14, 184, 14, 183, 14, 183, 14, 185, 14, 183, 14, 184] destination: 65535 source: 1 length: 28
group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 29, 14, 177, 14, 175, 14, 177, 14, 178, 14, 177, 14, 178, 14, 177, 14, 176, 14, 177, 14, 176] destination: 65535 source: 1 length: 28 group: 34 type: 147 data:
[0, 0, 1, 0, 0, 1, 0, 30, 14, 184, 14, 184, 14, 184, 14, 183, 14, 183, 14, 184, 14, 183, 14, 183, 14, 184, 14, 183] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 31,
14, 178, 14, 177, 14, 176, 14, 176, 14, 177, 14, 176, 14, 178, 14, 177, 14, 177, 14, 176] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 32, 14, 185, 14, 183, 14,
185, 14, 183, 14, 183, 14, 185, 14, 184, 14, 183, 14, 184, 14, 185] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 33, 14, 177, 14, 176, 14, 177, 14, 176, 14, 177,
14, 177, 14, 176, 14, 178, 14, 176, 14, 176] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 34, 14, 184, 14, 184, 14, 184, 14, 184, 14, 184, 14, 183, 14, 184, 14,
182, 14, 182, 14, 184] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 35, 14, 176, 14, 177, 14, 177, 14, 177, 14, 177, 14, 177, 14, 178, 14, 175, 14, 177, 14, 176]
destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 36, 14, 184, 14, 184, 14, 183, 14, 184, 14, 184, 14, 184, 14, 184, 14, 185, 14, 184, 14, 185] destination: 65535
source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 37, 14, 177, 14, 178, 14, 176, 14, 175, 14, 176, 14, 178, 14, 179, 14, 178, 14, 174, 14, 177] destination: 65535 source: 1 length: 28
group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 38, 14, 184, 14, 184, 14, 183, 14, 184, 14, 184, 14, 185, 14, 184, 14, 183, 14, 184, 14, 184] destination: 65535 source: 1 length: 28 group: 34 type: 147 data:
[0, 0, 1, 0, 0, 1, 0, 39, 14, 177, 14, 179, 14, 178, 14, 177, 14, 178, 14, 178, 14, 176, 14, 177, 14, 180, 14, 174] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 40,
14, 183, 14, 183, 14, 185, 14, 183, 14, 182, 14, 184, 14, 183, 14, 183, 14, 184, 14, 185] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 41, 14, 177, 14, 175, 14,
177, 14, 175, 14, 178, 14, 176, 14, 176, 14, 177, 14, 175, 14, 176] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 42, 14, 184, 14, 184, 14, 186, 14, 184, 14, 185,
14, 184, 14, 184, 14, 183, 14, 184, 14, 185] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 43, 14, 179, 14, 177, 14, 177, 14, 177, 14, 177, 14, 178, 14, 178, 14,
177, 14, 177, 14, 178] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 44, 14, 184, 14, 183, 14, 184, 14, 184, 14, 184, 14, 182, 14, 185, 14, 184, 14, 184, 14, 183]
destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 45, 14, 177, 14, 176, 14, 177, 14, 176, 14, 177, 14, 177, 14, 178, 14, 179, 14, 177, 14, 178] destination: 65535
source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 46, 14, 184, 14, 184, 14, 184, 14, 184, 14, 183, 14, 185, 14, 183, 14, 183, 14, 184, 14, 184] destination: 65535 source: 1 length: 28
group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 47, 14, 177, 14, 178, 14, 176, 14, 178, 14, 178, 14, 178, 14, 178, 14, 177, 14, 176, 14, 177] destination: 65535 source: 1 length: 28 group: 34 type: 147 data:
[0, 0, 1, 0, 0, 1, 0, 48, 14, 184, 14, 185, 14, 183, 14, 184, 14, 184, 14, 183, 14, 188, 14, 185, 14, 184, 14, 184] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 49,
14, 181, 14, 178, 14, 177, 14, 177, 14, 178, 14, 178, 14, 176, 14, 177, 14, 176, 14, 177] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 50, 14, 184, 14, 183, 14,
183, 14, 183, 14, 185, 14, 184, 14, 183, 14, 184, 14, 184, 14, 184] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 51, 14, 177, 14, 173, 14, 177, 14, 178, 14, 178,
14, 177, 14, 178, 14, 177, 14, 177, 14, 178] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 52, 14, 184, 14, 184, 14, 184, 14, 184, 14, 185, 14, 185, 14, 184, 14,
184, 14, 184, 14, 184] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 53, 14, 177, 14, 177, 14, 178, 14, 178, 14, 178, 14, 177, 14, 177, 14, 177, 14, 177, 14, 175]
destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 54, 14, 183, 14, 184, 14, 185, 14, 185, 14, 185, 14, 184, 14, 184, 14, 184, 14, 184, 14, 185] destination: 65535
source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 55, 14, 177, 14, 177, 14, 179, 14, 177, 14, 178, 14, 177, 14, 178, 14, 177, 14, 178, 14, 176] destination: 65535 source: 1 length: 28
group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 56, 14, 184, 14, 185, 14, 184, 14, 183, 14, 186, 14, 185, 14, 184, 14, 185, 14, 185, 14, 184] destination: 65535 source: 1 length: 28 group: 34 type: 147 data:
[0, 0, 1, 0, 0, 1, 0, 57, 14, 176, 14, 178, 14, 178, 14, 177, 14, 178, 14, 176, 14, 177, 14, 177, 14, 178, 14, 177] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 58,
14, 183, 14, 184, 14, 184, 14, 184, 14, 184, 14, 183, 14, 184, 14, 185, 14, 186, 14, 184] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 59, 14, 176, 14, 177, 14,
177, 14, 179, 14, 177, 14, 177, 14, 178, 14, 177, 14, 177, 14, 178] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 60, 14, 184, 14, 184, 14, 183, 14, 184, 14, 185,
14, 184, 14, 183, 14, 185, 14, 184, 14, 184] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 61, 14, 178, 14, 177, 14, 178, 14, 178, 14, 177, 14, 177, 14, 177, 14,
176, 14, 178, 14, 179] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 62, 14, 183, 14, 185, 14, 184, 14, 184, 14, 183, 14, 184, 14, 185, 14, 184, 14, 183, 14, 184]
destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 63, 14, 176, 14, 177, 14, 178, 14, 177, 14, 177, 14, 177, 14, 177, 14, 177, 14, 177, 14, 178] destination: 65535
source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 64, 14, 184, 14, 185, 14, 185, 14, 185, 14, 185, 14, 184, 14, 183, 14, 184, 14, 184, 14, 184] destination: 65535 source: 1 length: 28
group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 65, 14, 178, 14, 178, 14, 178, 14, 178, 14, 176, 14, 176, 14, 177, 14, 178, 14, 178, 14, 177] destination: 65535 source: 1 length: 28 group: 34 type: 147 data:
[0, 0, 1, 0, 0, 1, 0, 66, 14, 185, 14, 184, 14, 184, 14, 185, 14, 185, 14, 185, 14, 185, 14, 185, 14, 184, 14, 184] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 67,
14, 177, 14, 177, 14, 177, 14, 179, 14, 177, 14, 177, 14, 178, 14, 178, 14, 177, 14, 177] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 68, 14, 185, 14, 185, 14,
185, 14, 183, 14, 184, 14, 184, 14, 185, 14, 184, 14, 183, 14, 183] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 69, 14, 176, 14, 177, 14, 177, 14, 176, 14, 178,
14, 178, 14, 177, 14, 175, 14, 177, 14, 178] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 70, 14, 184, 14, 184, 14, 184, 14, 184, 14, 184, 14, 183, 14, 184, 14,
184, 14, 183, 14, 184] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 71, 14, 177, 14, 179, 14, 177, 14, 178, 14, 177, 14, 177, 14, 178, 14, 177, 14, 178, 14, 177]
destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 72, 14, 184, 14, 183, 14, 183, 14, 184, 14, 183, 14, 185, 14, 183, 14, 186, 14, 185, 14, 185] destination: 65535
source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 73, 14, 178, 14, 178, 14, 177, 14, 176, 14, 177, 14, 176, 14, 177, 14, 176, 14, 176, 14, 177] destination: 65535 source: 1 length: 28
group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 74, 14, 185, 14, 184, 14, 183, 14, 183, 14, 184, 14, 184, 14, 184, 14, 183, 14, 183, 14, 183] destination: 65535 source: 1 length: 28 group: 34 type: 147 data:
[0, 0, 1, 0, 0, 1, 0, 75, 14, 177, 14, 177, 14, 176, 14, 178, 14, 177, 14, 175, 14, 175, 14, 175, 14, 177, 14, 177] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 76,
14, 184, 14, 183, 14, 183, 14, 182, 14, 184, 14, 182, 14, 183, 14, 184, 14, 183, 14, 183] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 77, 14, 176, 14, 175, 14,
176, 14, 177, 14, 176, 14, 177, 14, 178, 14, 177, 14, 176, 14, 177] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 78, 14, 184, 14, 183, 14, 183, 14, 184, 14, 183,
14, 184, 14, 184, 14, 184, 14, 182, 14, 185] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 79, 14, 176, 14, 176, 14, 176, 14, 178, 14, 178, 14, 177, 14, 176, 14,
177, 14, 177, 14, 178] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 80, 14, 183, 14, 183, 14, 182, 14, 183, 14, 184, 14, 184, 14, 183, 14, 184, 14, 182, 14, 184]
destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 81, 14, 177, 14, 176, 14, 178, 14, 177, 14, 178, 14, 177, 14, 176, 14, 175, 14, 177, 14, 177] destination: 65535
source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 82, 14, 184, 14, 184, 14, 183, 14, 183, 14, 183, 14, 184, 14, 183, 14, 184, 14, 183, 14, 183] destination: 65535 source: 1 length: 28
group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 83, 14, 177, 14, 176, 14, 177, 14, 176, 14, 176, 14, 176, 14, 175, 14, 176, 14, 177, 14, 176] destination: 65535 source: 1 length: 28 group: 34 type: 147 data:
[0, 0, 1, 0, 0, 1, 0, 84, 14, 182, 14, 183, 14, 183, 14, 185, 14, 184, 14, 184, 14, 186, 14, 181, 14, 184, 14, 182] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 85,
14, 177, 14, 176, 14, 176, 14, 175, 14, 176, 14, 176, 14, 178, 14, 175, 14, 176, 14, 177] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 86, 14, 183, 14, 183, 14,
184, 14, 184, 14, 183, 14, 184, 14, 183, 14, 185, 14, 183, 14, 184] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 87, 14, 176, 14, 176, 14, 175, 14, 177, 14, 178,
14, 177, 14, 177, 14, 177, 14, 177, 14, 177] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 88, 14, 183, 14, 183, 14, 182, 14, 183, 14, 184, 14, 183, 14, 184, 14,
185, 14, 183, 14, 183] destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 89, 14, 176, 14, 176, 14, 177, 14, 176, 14, 176, 14, 176, 14, 175, 14, 176, 14, 178, 14, 176]
destination: 65535 source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 90, 14, 182, 14, 184, 14, 183, 14, 183, 14, 182, 14, 183, 14, 182, 14, 184, 14, 183, 14, 184] destination: 65535
source: 1 length: 28 group: 34 type: 147 data: [0, 0, 1, 0, 0, 1, 0, 91, 14, 177, 14, 176, 14, 177, 14, 177, 14, 176, 14, 177, 14, 176, 14, 178, 14, 176, 14, 177] destination: 65535 source: 1 length: 28
```

7 - Experiments with 4G Nodes

These experiments are recommended for works that demand the use of a cellular infrastructure with few clients.

The 4G scenario can be built on two different ways. The first one (sections 7.1, 7.3 and 7.4-5) is composed by at least 5 nodes, where three of them represent the Evolved Packet Core (EPC), composed by the Home Subscriber Server (HSS), the Mobility Management Entity (MME) and the Serving/Packet Data Network Gateway (SP-GW). The fourth node emulates the eNodeB, which creates a 4G cell using the Universal Software Radio Peripheral (USRP). Finally, the fifth node is the User Equipment (UE), which connects to the eNodeB also through an USRP.

On the other side, the second way (sections 7.2 and 7.3 and 7.4-5) is composed by at least 3 nodes, being one for the whole EPC (HSS+MME+SP-GW), one for the eNodeB and one for the UE. Below is presented the process to allocate the resources and configure the 4G nodes for each way.

7.1 - EPC (First Way): Elements running inside Separate Machines

In this scenario, the EPC elements are available at the FUTEBOL UFMG testbed in the same Virtual Machine (VM) image, and each EPC element (HSS/MME/SP-GW) needs to be allocated in a separate miniPC. The EPC elements are implemented by Open Air Interface (OAI) (see http://www.openairinterface.org/?page_id=25 for more details). In this scenario, it is possible to allocate a maximum of 4 UEs at the same time, since only 8 miniPCs are available at the testbed.

7.1.1 - RSpec Description

As stated in the previous section, the EPC elements (HSS/MME/SP-GW) must be allocated using the same VM image, which contains all the software responsible for the authentication and management of the connected UEs as well as the eNodeB. To do this, the *sliver_type* attribute is set to **usrp-vm** and *disk_image* is set to **oai_epc**.

You can specify any testbed node to allocate each of the above elements. However, some nodes were previously configured in order to make the setup process more simple. This manual will give the instruction to both cases, being the first one using the pre-configured nodes (see Section 7.1.2.1 for more details), and the second one using any node, which needs more steps to be configured (see Section 7.1.2.2 for more details).

The code below shows an RSpec example which allocates the EPC elements.

```
<node client_id="HSS" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
component_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am+node+11"
>
  <sliver_type name="usrp-vm">
      <disk_image
name="urn:publicid:IDN+futebol.dcc.ufmg.br+image+oai_epc"/>
  </sliver_type>
  <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="119.0"
y="63.5"/>
</node>

<node client_id="MME" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
component_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am+node+16"
>
  <sliver_type name="usrp-vm">
      <disk_image
name="urn:publicid:IDN+futebol.dcc.ufmg.br+image+oai_epc"/>
  </sliver_type>
  <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="270.0"
y="63.5"/>
```

```
</node>

<node client_id="SPGW" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
component_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am+node+17"
>
  <sliver_type name="usrp-vm">
      <disk_image
name="urn:publicid:IDN+futebol.dcc.ufmg.br+image+oai_epc"/>
  </sliver_type>
  <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="428.5"
y="61.5"/>
</node>
```

Each 4G element can be allocated at any miniPC from the testbed. This can be done according to the ID specified at the *component_id* attribute of each node tag with the specific number of the miniPC. For VMs, the ID has the “1X” format, where X represents the miniPC ID according to its position at the testbed.

In the code above, it can be observed that nodes 11, 16 and 17 were allocated to run the HSS, the MME and the SP-GW respectively. Each node was named according to its specific role, using the *client_id* attribute of the node tag.

Each EPC element, at the above mentioned nodes, was pre-configured in order to make the setup process more simple. It is also possible to run the EPC at any miniPC, however, the configuration phase will take more time, since more parameters will need to be redefined. More details of both options are available at sections 7.1.2.1 and 7.1.2.2. If you wish to use OAI to connect the LG Nexus smartphones available at the testbed, please refer to the additional configuration steps available at Section 7.1.2.3.

7.1.2 - Configuration

As stated before, this manual gives two configuration options for the OAI EPC: The first one (Section 7.1.2.1) has the EPC elements at predefined positions at the testbed (nodes 11, 16 and 17), with most of the configuration phase concluded. The second option (Section 7.1.2.2) gives to the user the option to configure the EPC at any position, instructing how to configure all the parts needed to have a funcional EPC. For the first option, this manual will use the same RSpec presented in Section 7.1.1.

Finally, as stated before, if you wish to use the LG Nexus smartphones along with OAI, please refer to Section 7.1.2.3 in order to run the additional configuration steps.

7.1.2.1 - Quick Way (Predefined Nodes)

1 - After the allocation of the nodes 11, 16 and 17, according to the RSpec presented in Section 7.1.1, open one terminal for each EPC element, and in all of them run the following:

```
sudo su
cd /home/oai-epc/openair-cn/SCRIPTS/
```

2 - At each EPC element, insert the GTP kernel module by running the following command:

```
modprobe gtp
```

7.1.2.2 - Complete Way (Any Node)

1 - Open a terminal for each EPC element. At each terminal, open the */etc/hosts* file. Associate the names "*X.testbed.lan X*" (being X 'hss', 'mme' or 'spgw') to the IP 127.0.1.1, like the example for the HSS below:

```
127.0.0.1        localhost
127.0.1.1        hss.testbed.lan hss
192.168.0.X      mme.testbed.lan mme    #X:MME's IP address at eth0
192.168.0.Y      spgw.testbed.lan spgw  #Y:SPGW's IP address at eth0
```

2 - At each terminal, change the content of the */etc/hostname* file to *hss*, *mme* and *spgw*.

3 - Change the hostname of each EPC element to *hss*, *mme* and *spgw*, like the example below for the HSS:

```
hostname hss
```

4 - Reopen all the terminals in order to make the hostname changes.

5 - At the HSS, open the file *hss_fd.conf* at */usr/local/etc/oai/freeDiameter/* and change the MME's IP address located at the last line of the file, inside the *ConnectPeer* parameter.

```
ConnectPeer = "mme.testbed.lan" { ConnectTo = "192.168.0.X"; No_TLS;
}; #X:MME’s IP address at eth0
```

6 - At the MME, open the file *mme.conf* at */usr/local/etc/oai/* and change the SP-GW’s IP address in the following parameter:

```
# S-GW binded interface for S11 communication (GTPV2-C), if none
selected the ITTI message interface is used
SGW_IPV4_ADDRESS_FOR_S11                = "192.168.0.X/24";  #Y:
SP-GW's IP address at eth0
```

7 - Still at the MME, open the file *mme_fd.conf* at */usr/local/etc/oai/freeDiameter/* and change the HSS’s IP address located at the last line of the file, inside the *ConnectPeer* parameter.

```
ConnectPeer= "hss.testbed.lan" { ConnectTo = "192.168.0.X"; No_SCTP ;
No_IPv6; Prefer_TCP; No_TLS; port = 3868;  realm = "testbed.lan";};
#X: HSS’s IP address at eth0
```

8 - At each EPC element, insert the GTP kernel module by running the following command:

```
modprobe gtp
```

7.1.2.3 - Additional Configuration Steps: Enable the Connection of the LG Nexus Smartphones

This section explains how to modify the OAI EPC in order to enable the connection of the LG Nexus smartphones available at the testbed. Once all the previous configuration steps are made (Section 7.1.2.1 or 7.1.2.2), follow the steps below:

1 - At the HSS, as *sudo*, open the MySQL database with the command below. Enter with the *hsspass* password when asked.

```
mysql -u hssuser -p
```

2 - Once inside the MySQL command line, select the OAI database with the following command:

```
use oai_db;
```

3 - Update the *users* table with the Nexus's USIM card data of all the FUTEBOL UFMG phones, through the queries below:

```
update users set imei = 353626072597862 where imsi = 208920100001102;
update users set imei = 353626073075272 where imsi = 208920100001103;
update users set imei = 353626072478972 where imsi = 208920100001104;
update users set imei = 353626072151181 where imsi = 208920100001105;
update users set imei = 353626072151264 where imsi = 208920100001106;
update users set imei = 353626072167914 where imsi = 208920100001107;
update users set imei = 353626072526127 where imsi = 208920100001108;
update users set users.key = 0xACF088F7E311233C47DA6AD8FD3F4429 where imsi = 208920100001102;
update users set users.key = 0x1F2D5A652CFF8879EE62C88909E982CF where imsi = 208920100001103;
update users set users.key = 0xC0F801B04045739607576762 4E8437D4 where imsi = 208920100001104;
update users set users.key = 0x5324A87995BC49A917B67A0A192A91C0 where imsi = 208920100001105;
update users set users.key = 0x5F8C0C524BEBBD09D41E57F3AE2556BF where imsi = 208920100001106;
update users set users.key = 0xFDFC828E472C42D0FED1E8ED9A5FD4C5 where imsi = 208920100001107;
update users set users.key = 0x1CD4F3D3A33A6ECA5808C879698606CE where imsi = 208920100001108;
update users set Opc = 0x603757709DCA44D64D69337F420589EB where imsi = 208920100001102;
update users set Opc = 0xA109136ED37960178DB6BDF7F88FD498 where imsi = 208920100001103;
update users set Opc = 0x441BCF71EADAA48E2bC2778B9C03EF03 where imsi = 208920100001104;
update users set Opc = 0x0BB7FCED4844F82458ACC6D85671BB74 where imsi = 208920100001105;
update users set Opc = 0xC2428ADA029A0CD57B0201A878ECD9C4 where imsi = 208920100001106;
update users set Opc = 0x67476C8FC23A3E3CC9C49E0893A58E8C where imsi = 208920100001107;
update users set Opc = 0xE1E8F41432CCFAD4AF1CF096F519DECA where imsi = 208920100001108;
```

4 - Exit the MySQL command line interface with the *exit;* command.

5 - At the MME, as *sudo*, open the file *emm_send.c* at */home/oai-epc/openair-cn/SRC/NAS/EMM/SAP/* and change all the occurences of *EPS_ATTACH_RESULT_EPS* by *EPS_ATTACH_RESULT_EPS_IMSI*.

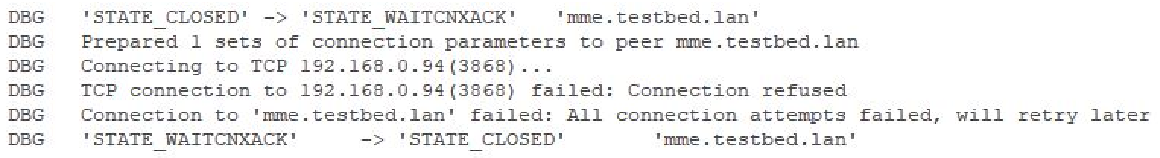
6 - Still at the MME, go to the */home/oai-epc/openair-cn/SCRIPTS/* folder and recompile the MME with the commands below:

```
cd /home/oai-epc/openair-cn/SCRIPTS/
./build_mme --clean
```

7.1.3 - Running

This section explains how to run all the EPC elements in the correct order. Once all the EPC elements are configured, follow the steps below:

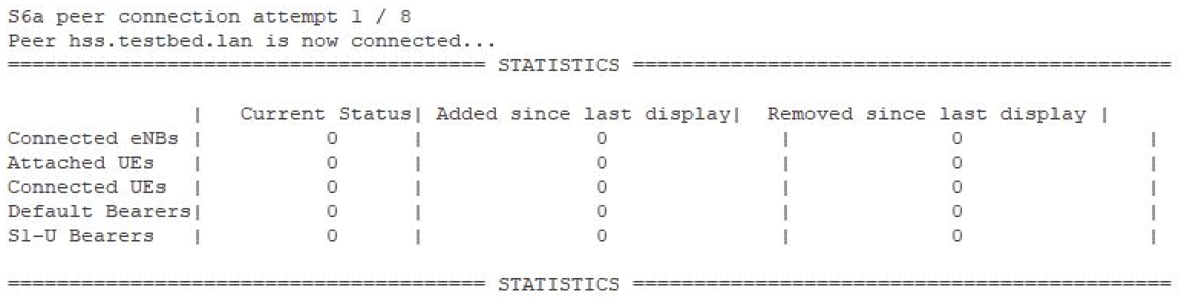
1 - At the HSS, inside the */home/oai-epc/openair-cn/SCRIPTS/* folder, run the *run_hss* script. You should receive an output similar to this one below:



```
DBG   'STATE_CLOSED' -> 'STATE_WAITCNXACK'   'mme.testbed.lan'
DBG   Prepared 1 sets of connection parameters to peer mme.testbed.lan
DBG   Connecting to TCP 192.168.0.94(3868)...
DBG   TCP connection to 192.168.0.94(3868) failed: Connection refused
DBG   Connection to 'mme.testbed.lan' failed: All connection attempts failed, will retry later
DBG   'STATE_WAITCNXACK'     -> 'STATE_CLOSED'       'mme.testbed.lan'
```

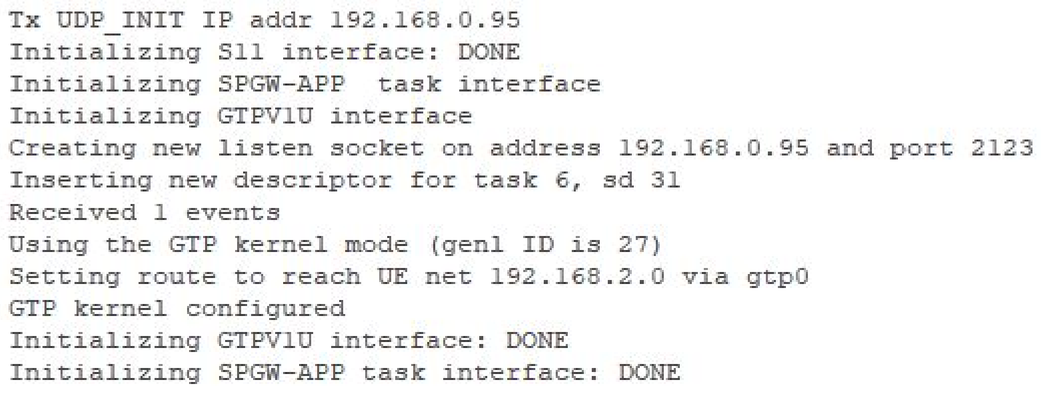
As can be seen in the above output, HSS tries to connect with the MME, but the connection will fail since MME is not started yet at this point. This is the main reason that the *STATE_CLOSED* message is presented at the end of the output. The connection process will keep being restarted until MME is connected successfully (next step).

2 - At the MME, inside the */home/oai-epc-srslte/openair-cn/SCRIPTS/* folder, run the *run_mme* script. The MME should connect to the HSS successfully and you should receive the following output:



```
S6a peer connection attempt 1 / 8
Peer hss.testbed.lan is now connected...
======================================= STATISTICS ============================================

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

======================================= STATISTICS ============================================
```

3 - At the SP-GW, inside the */home/oai-epc-srslte/openair-cn/SCRIPTS/* folder, run the *run_spgw* script. You should receive an output similar to this one below. A successful initialization is represented by all interfaces being associated to the *DONE* state message.



```
Tx UDP_INIT IP addr 192.168.0.95
Initializing S11 interface: DONE
Initializing SPGW-APP  task interface
Initializing GTPV1U interface
Creating new listen socket on address 192.168.0.95 and port 2123
Inserting new descriptor for task 6, sd 31
Received 1 events
Using the GTP kernel mode (genl ID is 27)
Setting route to reach UE net 192.168.2.0 via gtp0
GTP kernel configured
Initializing GTPV1U interface: DONE
Initializing SPGW-APP task interface: DONE
```

7.2 - EPC (Second Way): Elements running inside the Same Machine

The EPC software, in this way, is available in the same LXC container image. The EPC's functions are performed through *srsepc*, which is part of the srsLTE software suite (see https://github.com/srsLTE/srsLTE for more details).

7.2.1 - RSPec Description

As stated above, for the EPC, only one machine is needed to be allocated, with an LXC container that will run *srsepc*. In order to do this, configure your RSPec file the *sliver_type* as **raw-wifi** and the *disk_image* as **srsepc**. The code below shows an RSpec example which allocates the EPC machine.

```
<node client_id="EPC" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
component_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am+node+51"
>
  <sliver_type name="raw-wifi">
      <disk_image
name="urn:publicid:IDN+futebol.dcc.ufmg.br+image+srsepc"/>
  </sliver_type>
  <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="273.0"
y="170.5"/>
</node>
```

The EPC can be allocated at any miniPC from the testbed. This can be done according to the ID specified at the *component_id* attribute of each node tag with the specific number of the miniPC. For LXC containers, the ID has the "5X" format, where X represents the miniPC ID according to its position at the testbed.

In the code above, it can be observed that node 51 was selected to emulate the EPC. This node was named according to its specific role, using the *client_id* attribute of the node tag.

7.2.2 - Configuration

1 - Open a terminal window for the EPC and change the hostname to *epc*. In order to do that run the following command:

```
hostname epc
```

2 - Open the *hostname* file at */etc/* and also change the hostname to *epc*.

3 - Re-open the EPC terminal.

4 - Open the *epc.conf* file at */root/.default/srslte/* and edit the following lines to enter the IP address of the container:

```
mme_bind_addr = 192.168.0.X     #EPC’s address at eth0
…
gtpu_bind_addr = 192.168.0.X     #EPC’s address at eth0
```

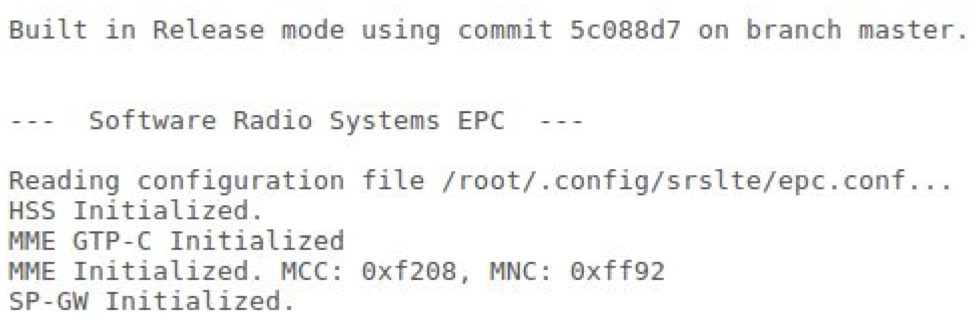
5 - Add the firewall rules below, through *iptables*. These rules are needed in order to enable the UE’s access to the Internet.

```
iptables -t nat -A POSTROUTING -s 192.168.2.0/24 -o eth0 -j MASQUERADE
iptables -A FORWARD -i srs_spgw_sgi -o eth0 -j ACCEPT
iptables -A FORWARD -o srs_spgw_sgi -i eth0 -j ACCEPT
iptables -t nat -A POSTROUTING -s 192.168.2.0/24 -o virtual0 -j MASQUERADE
iptables -A FORWARD -i srs_spgw_sgi -o virtual0 -j ACCEPT
iptables -A FORWARD -o srs_spgw_sgi -i virtual0 -j ACCEPT
```

6 - Open the *users_db.csv* file, at */root/.config/srslte/*. If you wish to connect the USRP-based UEs (see Section 7.4 for more details), comment with “#” all the lines starting with “*nexusue*”. Otherwise, if you wish to connect the LG Nexus-based UEs (see Section 7.5 for more details), comment with “#” all the lines starting with “*usrpue*”.

7.2.3 - Running

1 - Open a terminal window for the EPC. After that, start the EPC software using the the *srsepc* command. You should receive an output similar to this one below.



```
Built in Release mode using commit 5c088d7 on branch master.


---  Software Radio Systems EPC  ---

Reading configuration file /root/.config/srslte/epc.conf...
HSS Initialized.
MME GTP-C Initialized
MME Initialized. MCC: 0xf208, MNC: 0xff92
SP-GW Initialized.
```

7.3 - eNodeB

The eNodeB is available in the same Docker container image as the the *srsue* software (see Section 7.4 for more details), which has an USRP as the main resource to create the 4G connections. The eNodeB's functions are performed through *srsenb* respectively, which is part of the srsLTE software suite (see https://github.com/srsLTE/srsLTE for more details). Along with any of the EPC scenarios presented at sections 7.1 and 7.2, the experimenter has a complete 4G LTE stack, which can be modified according to his/her needs.

7.3.1 - RSPec Description

For the UE, configure the *sliver_type* as **raw-wifi** and the *disk_image* as **srslte**. The code below shows an RSpec example which allocates the eNodeB.

```
<node client_id="eNodeB" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
component_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am+node+55"
>
  <sliver_type name="raw-wifi">
      <disk_image
name="urn:publicid:IDN+futebol.dcc.ufmg.br+image+srslte"/>
  </sliver_type>
  <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="273.0"
y="170.5"/>
</node>
```

The eNodeB can be allocated at any miniPC from the testbed. This can be done according to the ID specified at the *component_id* attribute of each node tag with the specific number of the miniPC. For Docker containers, the ID has the “5X” format, where X represents the miniPC ID according to its position at the testbed.

In the code above, it can be observed that node 55 was selected to emulate the eNodeB. This node was named according to its specific role, using the *client_id* attribute of the node tag.

7.3.2 - Configuration

1 - Open a terminal window for the eNodeB and change the hostname to *enodeb*. In order to do that run the following command:

```
hostname enodeb
```

2 - Open the *hosts* file at */etc/* and also change the hostname to *enodeb* for the IP 172.17.0.2. After that, put the same hostname inside the *hostname* file at */etc/*.

3 - Re-open the eNodeB terminal. After that, at the eNodeB open the *enb.conf* file at */root/.config/srslte/*. Change the MME and the eNodeB's IP addresses at the following parameters:

```
...
mme_addr = 192.168.0.X  #X: MME's IP address
gtp_bind_addr = 192.168.0.Y  #Y: eNodeB's IP address at virtual0
s1c_bind_addr = 192.168.0.Y  #Y: eNodeB's IP address at virtual0
...
```

7.3.2.1 - Additional Configuration Steps: Enable the Connection of the LG Nexus Smartphones

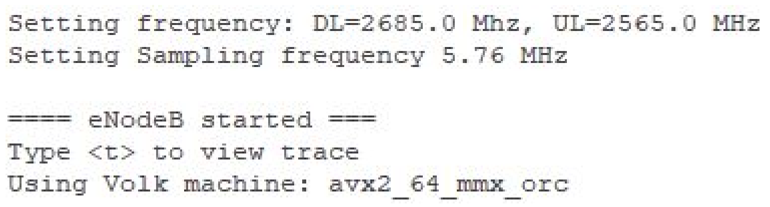
This section explains how to modify the eNodeB in order to enable the connection of the LG Nexus smartphones available at the testbed. Once all the previous configuration steps are made (Section 7.1.3), follow this single step below:

1 - Open the *enb.conf* file, at */root/.config/srslte/*. Change the *dl_earfcn* parameter from 3400 to 1375.

7.3.3 - Running

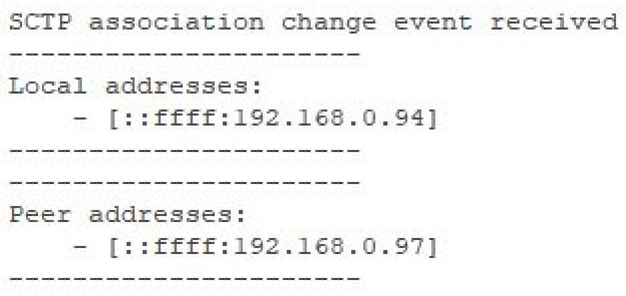
This section explains how to run the eNodeB, and considers that the EPC is already running. Please see sections 7.1.3 or 7.2.3 (depending on which EPC software was configured) for more details.

1 - At the eNodeB, run the **srsenb** command. You need to receive a message informing that the eNodeB has started, as shown below. After this output, type and enter the letter *"t"* in order to activate the signal trace for the connected UEs.



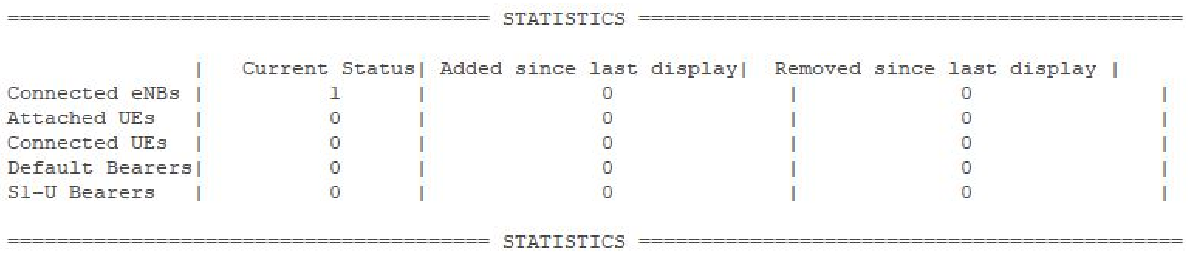
```
Setting frequency: DL=2685.0 Mhz, UL=2565.0 MHz
Setting Sampling frequency 5.76 MHz

==== eNodeB started ===
Type <t> to view trace
Using Volk machine: avx2_64_mmx_orc
```

When the eNodeB starts, it attempts to connect to the EPC's MME through the SCTP protocol. At the same time that the message above is shown, the MME must generate an event indicating that such SCTP connection was done successfully. If the running EPC software is OAI (see Section 7.1 for more details), the following image is shown below at the MME terminal:



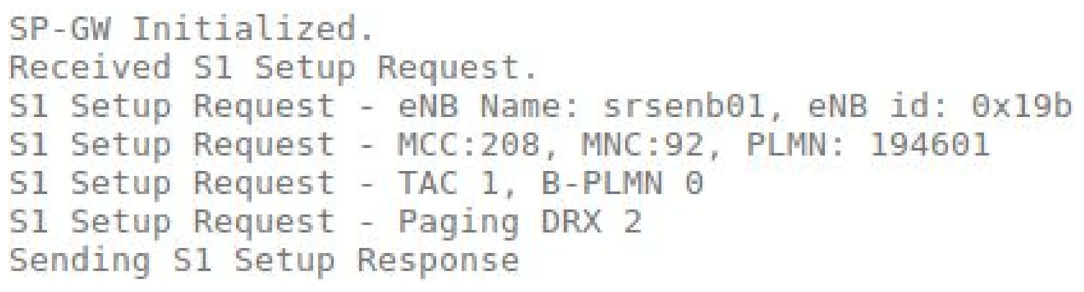
```
SCTP association change event received
----------------------
Local addresses:
    - [::ffff:192.168.0.94]
----------------------
----------------------
Peer addresses:
    - [::ffff:192.168.0.97]
----------------------
```

Right after the SCTP event log at the MME, the *STATISTICS* table (printed continuously at the MME's terminal) should be updated, indicating that one eNodeB is now connected:



```
======================================= STATISTICS ============================================

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

======================================= STATISTICS ============================================
```

On the other side, if the running EPC software is *srsepc* (see Section 7.2 for more details), it is expected the following output at the EPC terminal:



```
SP-GW Initialized.
Received S1 Setup Request.
S1 Setup Request - eNB Name: srsenb01, eNB id: 0x19b
S1 Setup Request - MCC:208, MNC:92, PLMN: 194601
S1 Setup Request - TAC 1, B-PLMN 0
S1 Setup Request - Paging DRX 2
Sending S1 Setup Response
```

7.4 - USRP-based UE

The UE software is available in the same Docker container image as the *srsenb* software (see Section 7.3 for more details), which has an USRP as the main resource to create the 4G connections. The UE's functions are performed through *srsue*, which is part of the srsLTE software suite (see https://github.com/srsLTE/srsLTE for more details).

7.4.1 - RSPec Description

For the UE, configure the *sliver_type* as **raw-wifi** and the *disk_image* as **srslte**. The code below shows an RSpec example which allocates one UE.

```
<node client_id="UE" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
component_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am+node+58"
>
  <sliver_type name="raw-wifi">
      <disk_image
name="urn:publicid:IDN+futebol.dcc.ufmg.br+image+srslte"/>
  </sliver_type>
  <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="428.5"
y="273.5"/>
</node>
```

The UE can be allocated at any miniPC from the testbed. This can be done according to the ID specified at the *component_id* attribute of each node tag with the specific number of the miniPC. For Docker containers, the ID has the "5X" format, where X represents the miniPC ID according to its position at the testbed.

In the code above, it can be observed that node 58 was selected to emulate the UE. Each node was named according to its specific role, using the *client_id* attribute of the node tag.

7.4.2 - Configuration

1 - Open a terminal window for the UE and change the hostname to *ue* with the command below. If you want to connect multiple UEs at the same time, use a specific hostname to each of them (Ex.: *ueX*, where X is any digit).

```
hostname ueX
```

2 - Open the *hosts* file at */etc/* and also change the hostname to *ue* for the IP 172.17.0.2. After that, put the same hostname inside the *hostname* file at */etc/*.

3 - Re-open the UE terminal. The UE's configuration file is already ready for experiments with one UE. In other words, the parameters related to the authentication of an UE at the EPC (the same ones that should be recorded at the SIM card from a smartphone) (see *USIM configuration/[usim]* parameters at the */root/.config/srslte/ue.conf* file for more details) are already set up by default, and also match with one entry already stored by default at the HSS's database. However, if you allocated other miniPCs to act as an UE, at each additional UE modify the following parameters at the *ue.conf* file:

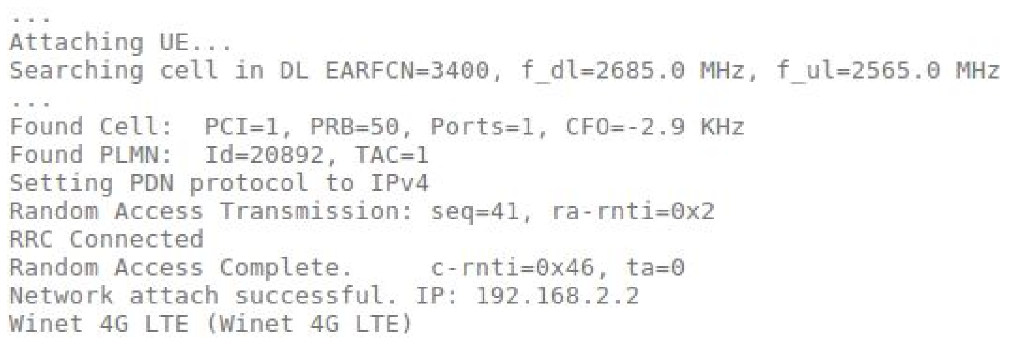
```
imsi = 20892010000110X  #X: Client’s ID (1, 2, 3 or 4)
imei = 3560920407920Y0  #Y: Client’s ID (1, 2, 3 or 4)
```

7.4.3 - Running

This section explains how to run the UE, as well as how to transmit data between it and the EPC.

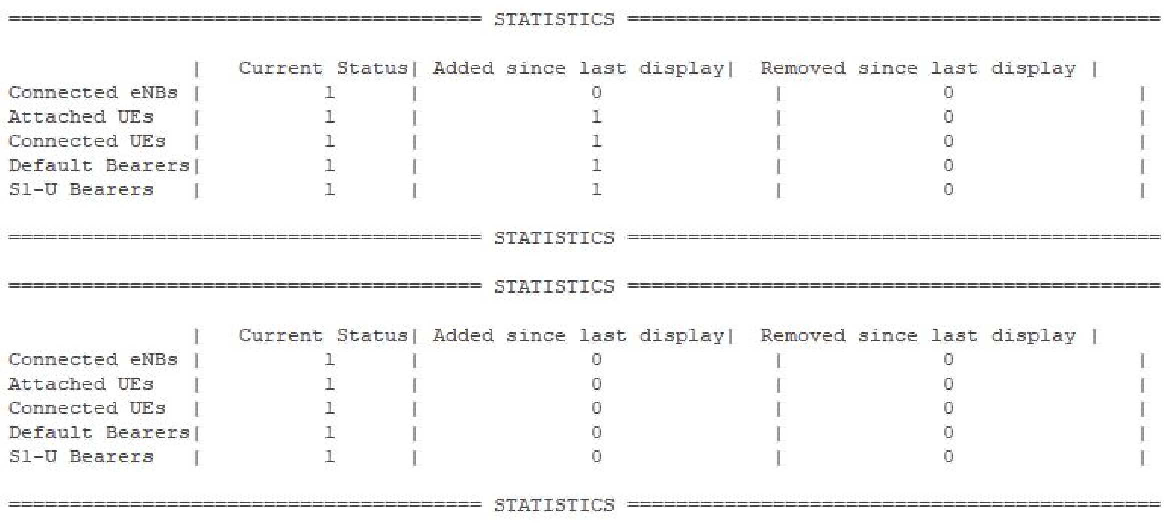
This section considers that the EPC and the eNodeB are already running. Please see sections 7.1.3 or 7.2.3 (depending on which EPC software was configured) and 7.3 (for the eNodeB) for more details.

1 - At the UE, run the **srsue** command. The UE should try to find the 4G cell and connect with the respective eNodeB. The connection is successful when the UE receives an IP address from the SP-GW, and the network attach event is logged by the UE, like it's shown below:



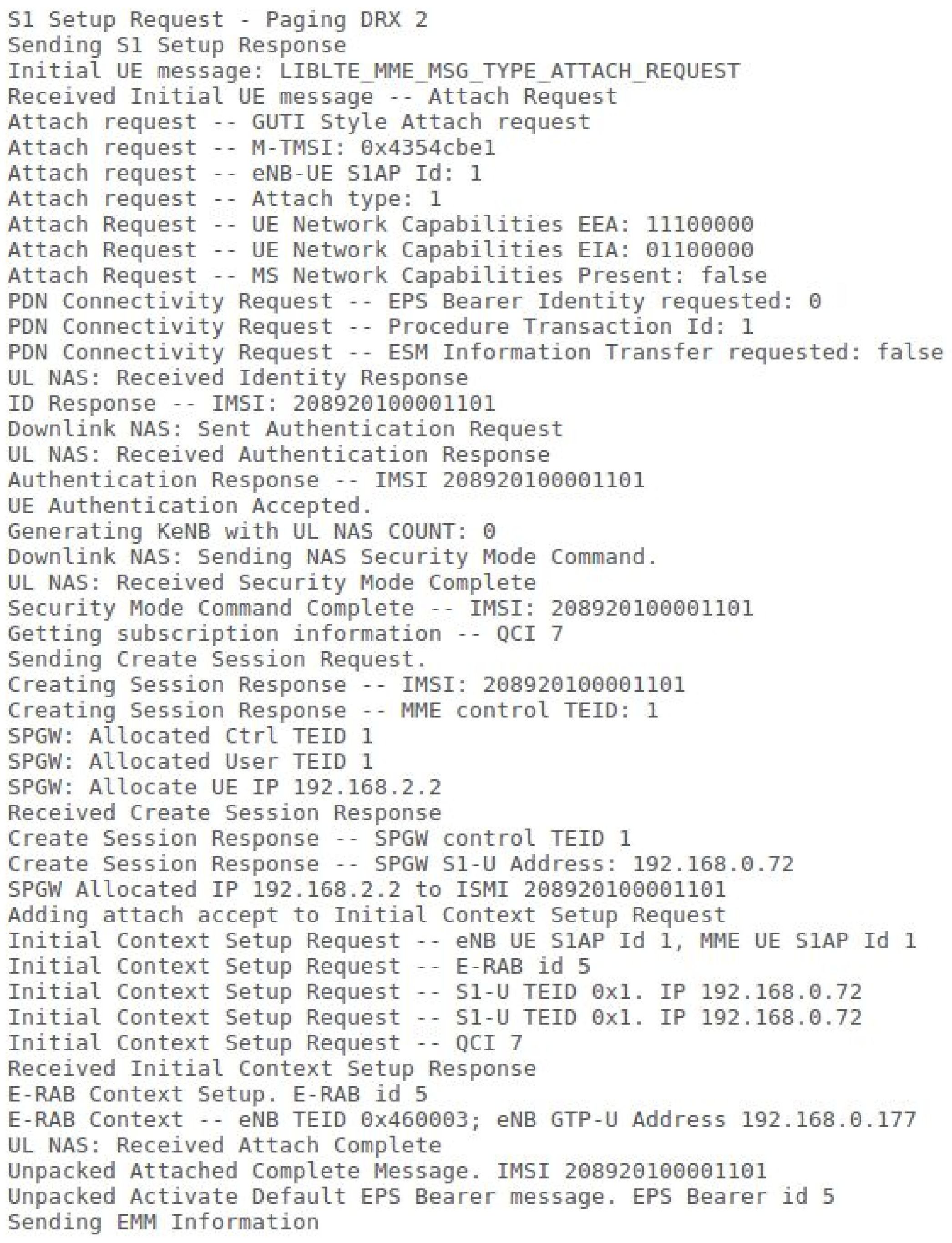
```
...
Attaching UE...
Searching cell in DL EARFCN=3400, f_dl=2685.0 MHz, f_ul=2565.0 MHz
...
Found Cell:  PCI=1, PRB=50, Ports=1, CFO=-2.9 KHz
Found PLMN:  Id=20892, TAC=1
Setting PDN protocol to IPv4
Random Access Transmission: seq=41, ra-rnti=0x2
RRC Connected
Random Access Complete.     c-rnti=0x46, ta=0
Network attach successful. IP: 192.168.2.2
Winet 4G LTE (Winet 4G LTE)
```

The IP address received by the UE is associated to a tunneling interface named **tun_srsue**, which will be used to transmit data between the UE and the eNodeB, both of them emulated by the USRP. After a successful connection from the UE, the MME must report some updates in the terminal. If the running EPC software is OAI (see Section 7.1 for more details), the MME should update the STATISTICS table, indicating the attachment of the new UE:



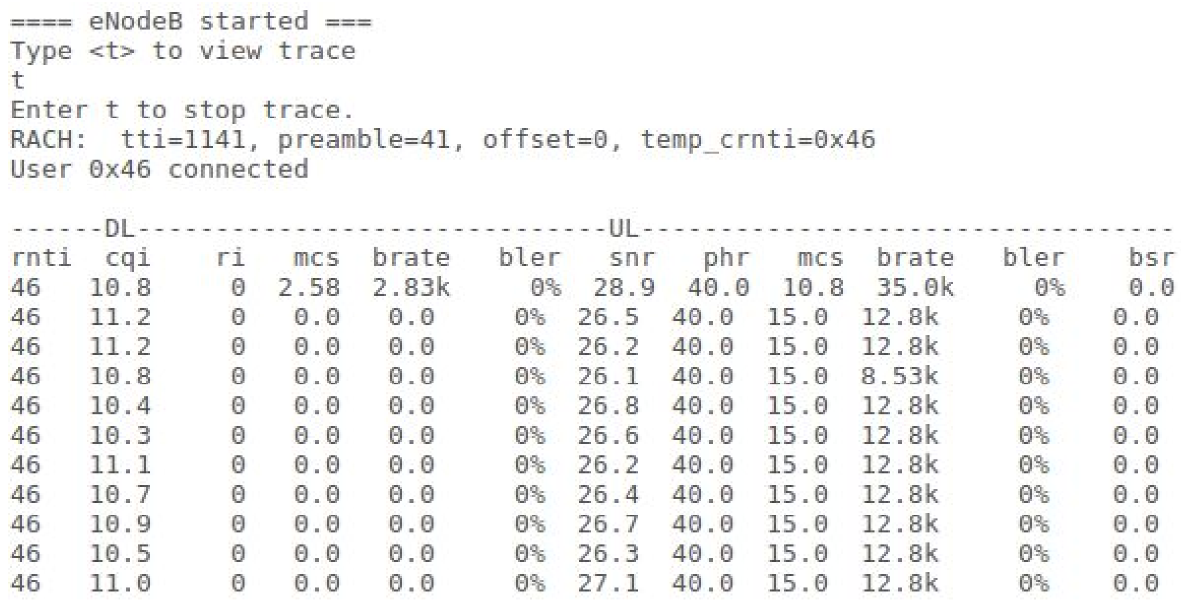
```
======================================= STATISTICS ============================================

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

======================================= STATISTICS ============================================

======================================= 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                |

======================================= STATISTICS ============================================
```

On the other side, if the running EPC software is *srsepc* (see Section 7.2 for more details), is is expected the following similar output at the EPC terminal:



```
S1 Setup Request - Paging DRX 2
Sending S1 Setup Response
Initial UE message: LIBLTE_MME_MSG_TYPE_ATTACH_REQUEST
Received Initial UE message -- Attach Request
Attach request -- GUTI Style Attach request
Attach request -- M-TMSI: 0x4354cbe1
Attach request -- eNB-UE S1AP Id: 1
Attach request -- Attach type: 1
Attach Request -- UE Network Capabilities EEA: 11100000
Attach Request -- UE Network Capabilities EIA: 01100000
Attach Request -- MS Network Capabilities Present: false
PDN Connectivity Request -- EPS Bearer Identity requested: 0
PDN Connectivity Request -- Procedure Transaction Id: 1
PDN Connectivity Request -- ESM Information Transfer requested: false
UL NAS: Received Identity Response
ID Response -- IMSI: 208920100001101
Downlink NAS: Sent Authentication Request
UL NAS: Received Authentication Response
Authentication Response -- IMSI 208920100001101
UE Authentication Accepted.
Generating KeNB with UL NAS COUNT: 0
Downlink NAS: Sending NAS Security Mode Command.
UL NAS: Received Security Mode Complete
Security Mode Command Complete -- IMSI: 208920100001101
Getting subscription information -- QCI 7
Sending Create Session Request.
Creating Session Response -- IMSI: 208920100001101
Creating Session Response -- MME control TEID: 1
SPGW: Allocated Ctrl TEID 1
SPGW: Allocated User TEID 1
SPGW: Allocate UE IP 192.168.2.2
Received Create Session Response
Create Session Response -- SPGW control TEID 1
Create Session Response -- SPGW S1-U Address: 192.168.0.72
SPGW Allocated IP 192.168.2.2 to ISMI 208920100001101
Adding attach accept to Initial Context Setup Request
Initial Context Setup Request -- eNB UE S1AP Id 1, MME UE S1AP Id 1
Initial Context Setup Request -- E-RAB id 5
Initial Context Setup Request -- S1-U TEID 0x1. IP 192.168.0.72
Initial Context Setup Request -- S1-U TEID 0x1. IP 192.168.0.72
Initial Context Setup Request -- QCI 7
Received Initial Context Setup Response
E-RAB Context Setup. E-RAB id 5
E-RAB Context -- eNB TEID 0x460003; eNB GTP-U Address 192.168.0.177
UL NAS: Received Attach Complete
Unpacked Attached Complete Message. IMSI 208920100001101
Unpacked Activate Default EPS Bearer message. EPS Bearer id 5
Sending EMM Information
```

From eNodeB's side it is printed, one time per second, the signal data from the connected UE, as shown below:

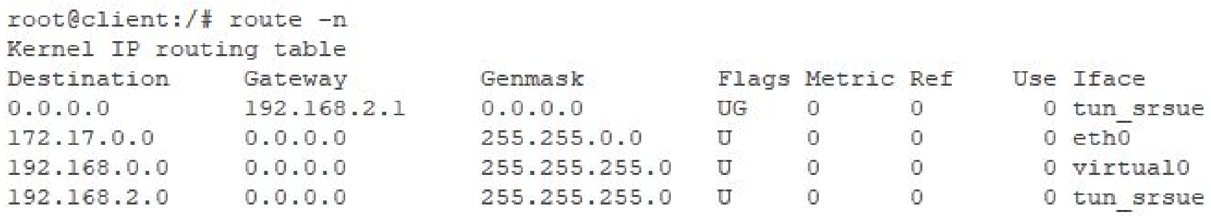


```
==== eNodeB started ===
Type <t> to view trace
t
Enter t to stop trace.
RACH:  tti=1141, preamble=41, offset=0, temp_crnti=0x46
User 0x46 connected

------DL-------------------------------UL---------------------------------
rnti  cqi     ri   mcs  brate   bler   snr   phr   mcs  brate   bler    bsr
46    10.8     0  2.58  2.83k     0%  28.9  40.0  10.8  35.0k     0%    0.0
46    11.2     0   0.0   0.0     0%  26.5  40.0  15.0  12.8k     0%    0.0
46    11.2     0   0.0   0.0     0%  26.2  40.0  15.0  12.8k     0%    0.0
46    10.8     0   0.0   0.0     0%  26.1  40.0  15.0  8.53k     0%    0.0
46    10.4     0   0.0   0.0     0%  26.8  40.0  15.0  12.8k     0%    0.0
46    10.3     0   0.0   0.0     0%  26.6  40.0  15.0  12.8k     0%    0.0
46    11.1     0   0.0   0.0     0%  26.2  40.0  15.0  12.8k     0%    0.0
46    10.7     0   0.0   0.0     0%  26.4  40.0  15.0  12.8k     0%    0.0
46    10.9     0   0.0   0.0     0%  26.7  40.0  15.0  12.8k     0%    0.0
46    10.5     0   0.0   0.0     0%  26.3  40.0  15.0  12.8k     0%    0.0
46    11.0     0   0.0   0.0     0%  27.1  40.0  15.0  12.8k     0%    0.0
```

2 - In order to transmit data, open another terminal for the UE, and redefine the default routes and the default MTU (Maximum Transmission Unit) from the **tun_srsue** interface with the commands below. The MTU redefinition is needed since frames bigger than 1460 bytes are not accepted by such interface.

```
route del default gw 172.17.0.1 eth0
route add default gw 192.168.2.1 tun_srsue
ifconfig tun_srsue mtu 1460
```

The routing table must stay like this:



```
root@client:/# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.2.1     0.0.0.0         UG    0      0        0 tun_srsue
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 eth0
192.168.0.0     0.0.0.0         255.255.255.0   U     0      0        0 virtual0
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 tun_srsue
```

As shown at the above output, any packet that goes outside the testbed now must be sent through the **tun_srsue** interface. The output example below shows ICMP packets sent to the FUTEBOL website at UFMG.

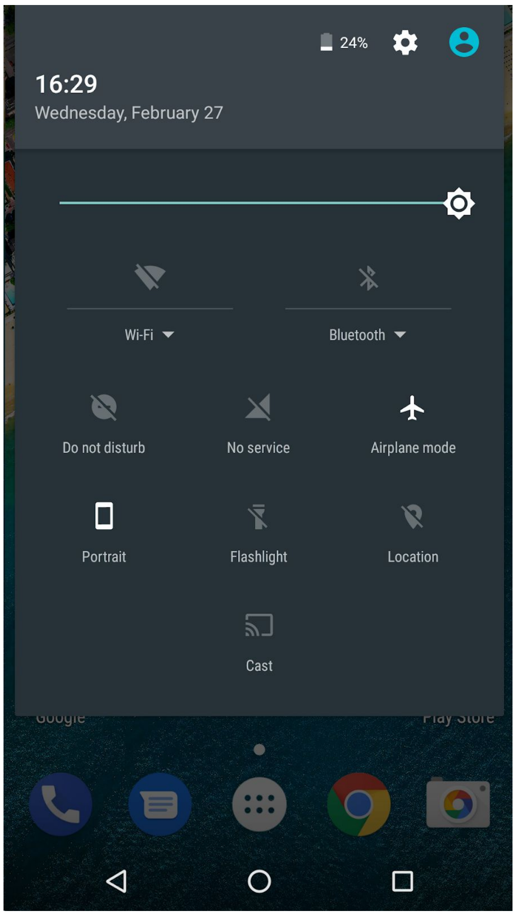
```
root@client:/# ping futebol.dcc.ufmg.br
PING vmserver.winet.dcc.ufmg.br (150.164.10.23) 56(84) bytes of data.
64 bytes from 150.164.10.23: icmp_seq=1 ttl=63 time=64.8 ms
64 bytes from 150.164.10.23: icmp_seq=2 ttl=63 time=40.7 ms
64 bytes from 150.164.10.23: icmp_seq=3 ttl=63 time=40.0 ms
64 bytes from 150.164.10.23: icmp_seq=4 ttl=63 time=37.9 ms
64 bytes from 150.164.10.23: icmp_seq=5 ttl=63 time=36.8 ms
64 bytes from 150.164.10.23: icmp_seq=6 ttl=63 time=34.9 ms
```

7.5 - UE (LG Nexus-based)

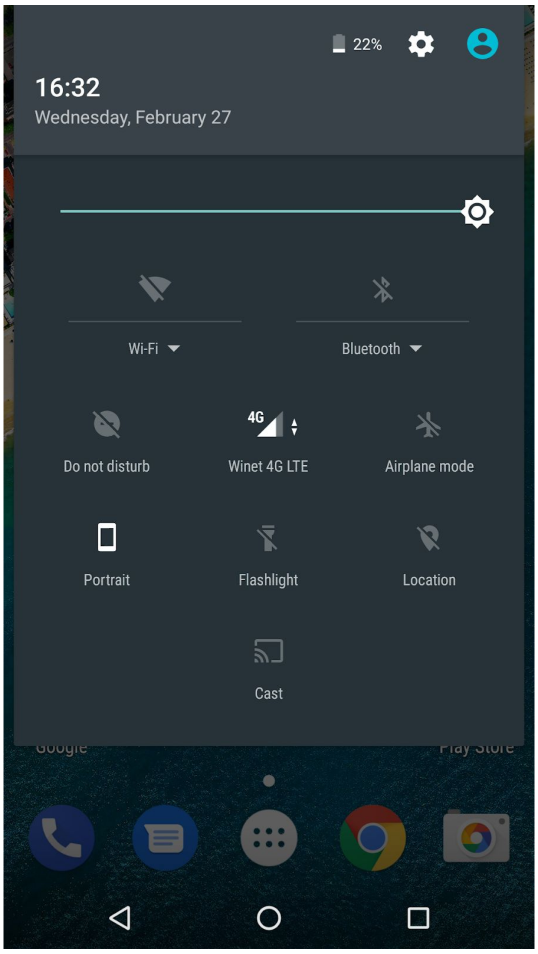
This section explains how to use the LG Nexus smartphones from the testbed in order to connect them to the 4G network. It is considered that the EPC and the eNodeB are already running. Please see sections 7.1.3 or 7.2.3 (depending on which EPC software was configured) and 7.3 (for the eNodeB) for more details.

1 - Allocate the LG Nexus smartphones according to the instructions available at Section 8. No further configuration is needed.

2 - Access the smartphone through the *OpenSTF* interface and open the notification menu, as shown below:



3 - Deactivate the *Airplane Mode*. The smartphone should connect automatically to the 4G network, as shown below:



8 - Experiments with LG Nexus Smartphones

This feature is recommended for experiments that require the use of one or more mobile phones, whose control is done remotely.

The scenario for experiments using *OpenSTF* (see Section 8.1 for more details) is available in the UFMG FUTEBOL testbed through virtual machine (VM) images, where each VM is responsible for the administration and orchestration of one LG Nexus smartphone. The user is able to allocate up to eight smartphones at the same time, being one *OpenSTF* instance dedicated to each smartphone. The allocation process of this resource is presented in sections 8.2 and 8.3, and the running process is presented at Section 8.4.

8.1 - OpenSTF Description

OpenSTF is a web application responsible for facilitating the remote access and manipulation of smartphones, tablets and other electronic devices. Through this application, it is possible to install new mobile applications, change one or more device's settings and use all other available features quickly, efficiently and without direct contact.

Currently, the *OpenSTF* initiative is maintained by *The OpenSTF Project* under the "Apache License, Version 2.0" license. For more information and instructions for using *OpenSTF*'s features, the user may refer to the project's repository (https://github.com/openstf/stf).

8.2 - RSpec Description

To perform the allocation of one VM with *OpenSTF* (i.e. one LG Nexus smartphone), set the *sliver_type* parameter to *openstf-vm* and the *disk_image* parameter to *openstf*. The code below exemplifies the RSpec to allocate one *OpenSTF* node.

```
<node client_id="CellPhone" exclusive="true"
component_manager_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am"
component_id="urn:publicid:IDN+futebol.dcc.ufmg.br+authority+am+node+98"
>
  <sliver_type name="openstf-vm">
     <disk_image
name="urn:publicid:IDN+futebol.dcc.ufmg.br+image+openstf"/>
  </sliver_type>
  <location xmlns="http://jfed.iminds.be/rspec/ext/jfed/1" x="273.0"
y="170.5"/>
</node>
```

OpenSTF can be allocated on any miniPC from the testbed, in the node's ID range from 90 to 98. This can be done by the node specified in the *component_id* parameter of the RSpec. In the example above, it can be seen that node 98 was specified to be allocated for *OpenSTF*.

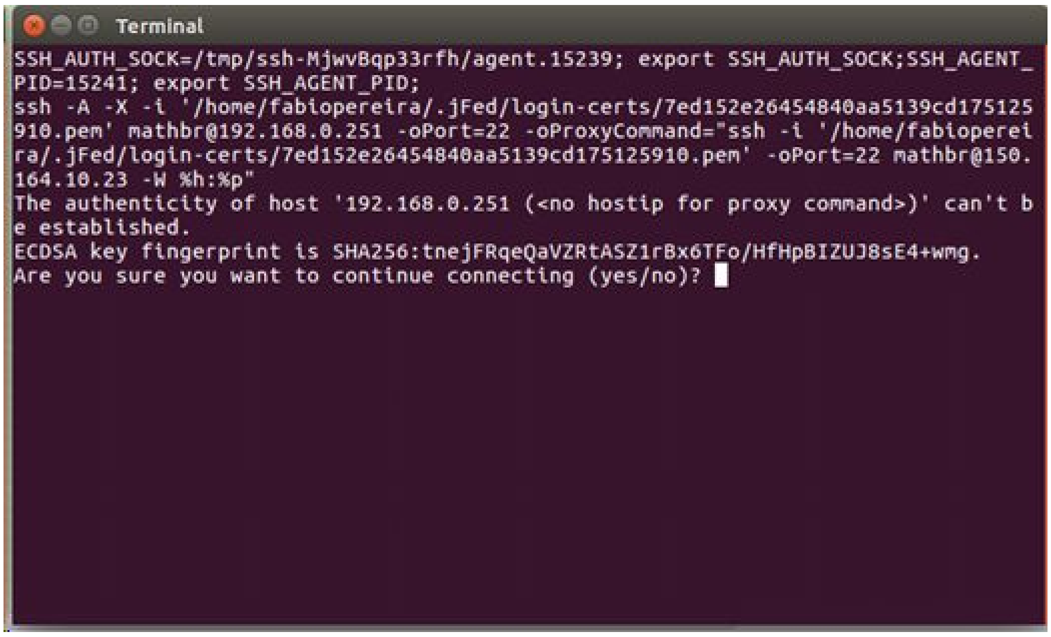
Finally, it is recommended that the *client_id* parameter be assigned to an exclusive name, like *CellPhone* as done in the RSpec example, in order to differ this node from the other allocated resource types, if it is the case.

8.3 - Configuring

In this section it is presented the configuration procedure of the LG Nexus smartphone resource at the UFMG FUTEBOL testbed, using *jFed*. In this example, it is applied the RSpec presented at Section 8.2, that selects node 98 to run the VM image with *OpenSTF*.

1 - After allocating the resource with *jFed*, as shown in the image below, right-click under the node and select the *Open SSH terminal* option.

2 - With the terminal window already open, accept the addition of the SSH key as requested by the terminal:



```
SSH_AUTH_SOCK=/tmp/ssh-MjwvBqp33rfh/agent.15239; export SSH_AUTH_SOCK;SSH_AGENT_
PID=15241; export SSH_AGENT_PID;
ssh -A -X -i '/home/fabiopereira/.jFed/login-certs/7ed152e26454840aa5139cd175125
910.pem' mathbr@192.168.0.251 -oPort=22 -oProxyCommand="ssh -i '/home/fabioperei
ra/.jFed/login-certs/7ed152e26454840aa5139cd175125910.pem' -oPort=22 mathbr@150.
164.10.23 -W %h:%p"
The authenticity of host '192.168.0.251 (<no hostip for proxy command>)' can't b
e established.
ECDSA key fingerprint is SHA256:tnejFRqeQaVZRtASZ1rBx6TFo/HfHpBIZUJ8sE4+wmg.
Are you sure you want to continue connecting (yes/no)?
```

3 - Since *OpenSTF* is a web application running inside one of the testbed's miniPCs and allocated by *jFed*, it is necessary to redirect the application interface to the user's local machine, through port mapping. In order to do this, obtain the authentication command used at the previous terminal, that is, from *SSH_AUTH*_SOCK (the beginning of the terminal output) to *"–W %h:%p"*.

4 - Close this terminal, and open a new one at your local machine. At this new terminal, paste the copied command string to open a new ssh section. Before entering the *ssh* command, add the port mapping parameters, as shown in red at the example below (you do not need to change the text in red):

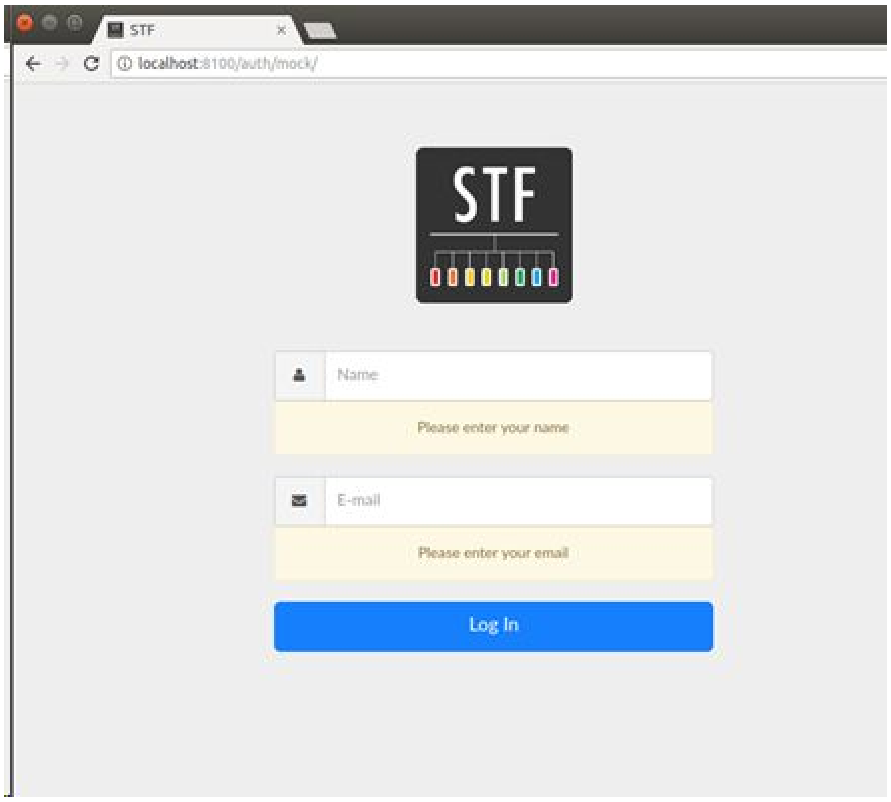
```
SSH_AUTH_SOCK=/tmp/ssh-MjwvBqp33rfh/agent.15239; export
SSH_AUTH_SOCK;SSH_AGENT_PID=15241; export SSH_AGENT_PID;
ssh -A -X -i
'/home/fabiopereira/.jFed/login-certs/7ed152e26454840aa5139cd175125910
.pem' mathbr@192.168.0.251 -oPort=22 -oProxyCommand="ssh -i
'/home/fabiopereira/.jFed/login-certs/7ed152e26454840aa5139cd175125910
.pem' -oPort=22 mathbr@150.164.10.23 -W %h:%p" -L 8100:localhost:8100
-L 8100:localhost:8100
```

8.4 - Running

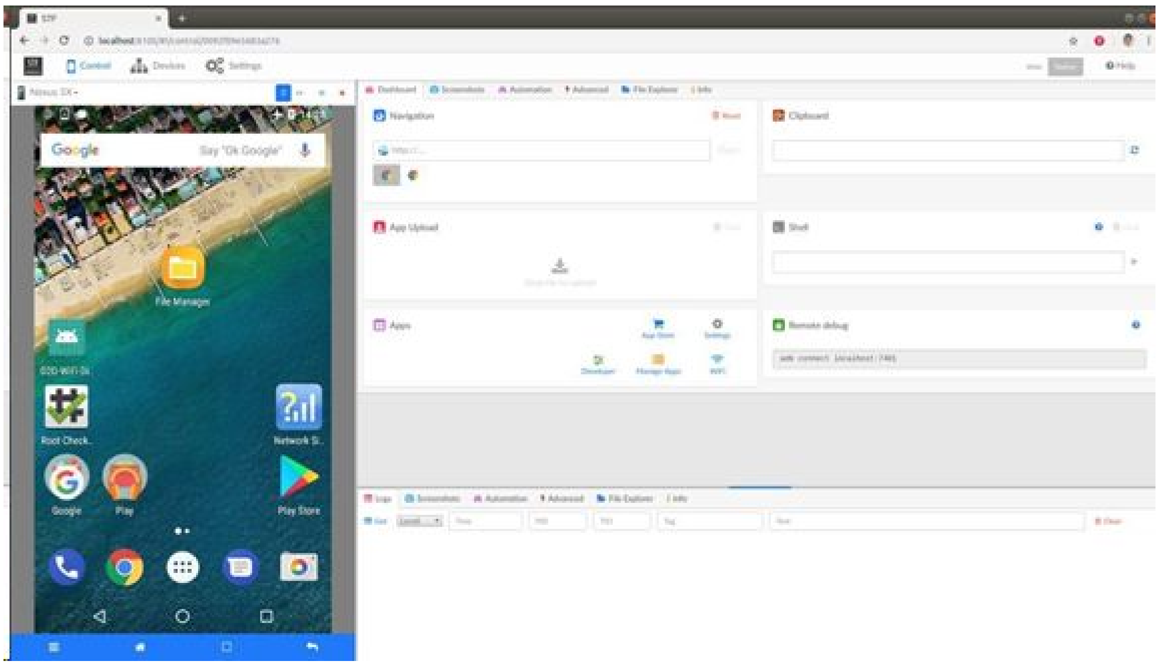
With the modified SSH access presented at step 4 from Section 8.3, keep the terminal minimized and open a local browser of your choice. Access the following URL:

```
http://localhost:8100
```

All OpenSTF features will be available through this remote browser access. The screen below shows the initial screen displayed when accessing the above URL. This screen requests the name and email information of the user that will use the resource. You can enter any valid email at this screen.



After entering a valid name and email, click on *Log In* and the next available screen lists all devices available for use. In the case of the UFMG FUTEBOL testbed, each miniPC (each *OpenSTF* node) will have one Nexus 5X, as shown in the image below:



With the interface above, the user is able to emulate touch commands at the smartphone, through his screen presented at the left side. In the right side, it is also possible to, among other things, upload an Android application, run Shell commands, debug some features and access the folder browser.