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 Daniel Fernandes Macedo - Universidade Federal de Minas Gerais
Version	0.1
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 how to allocate Raspberry Pis, USRP, WiFi on a PC and the Advanticsys nodes	

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 WiFi 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 two 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/.

- 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 and Raspberries 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).

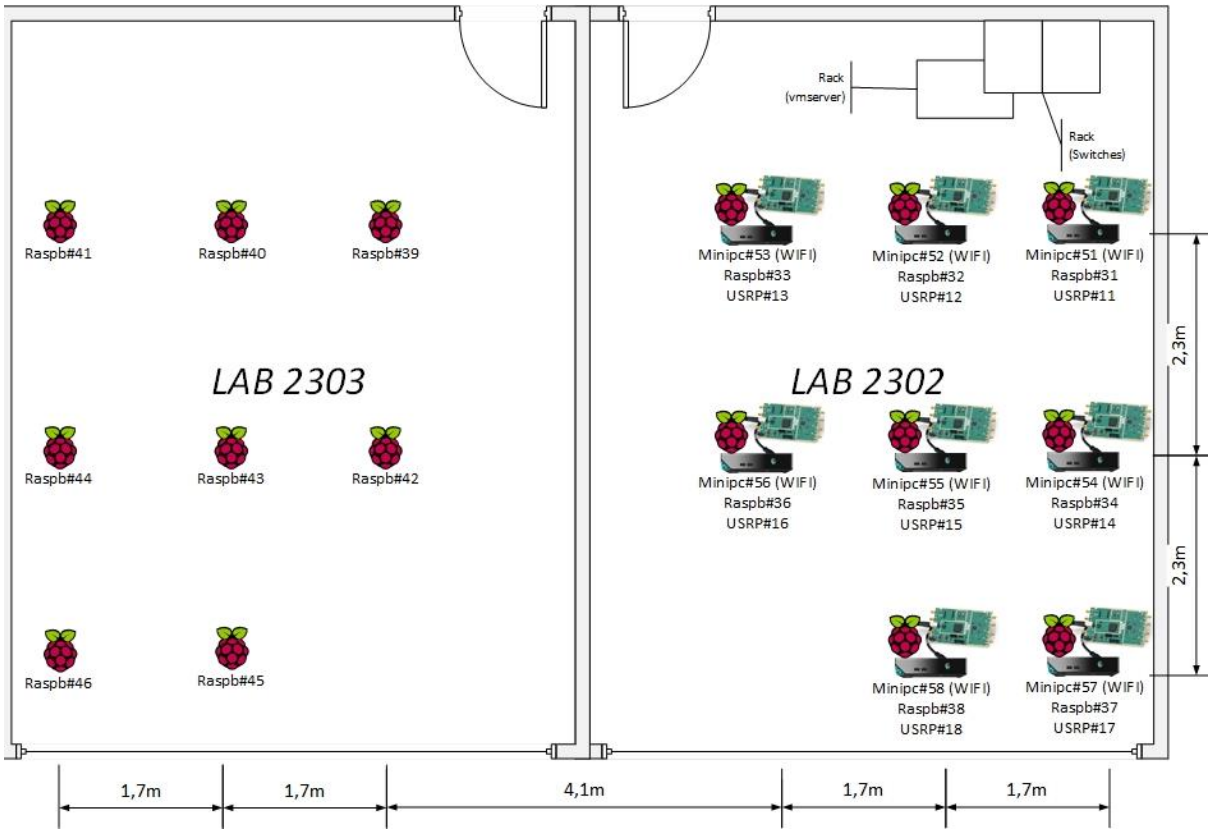


Fig 1 - Map of the placement of the USRPs, mini PCs and Raspberry Pis.

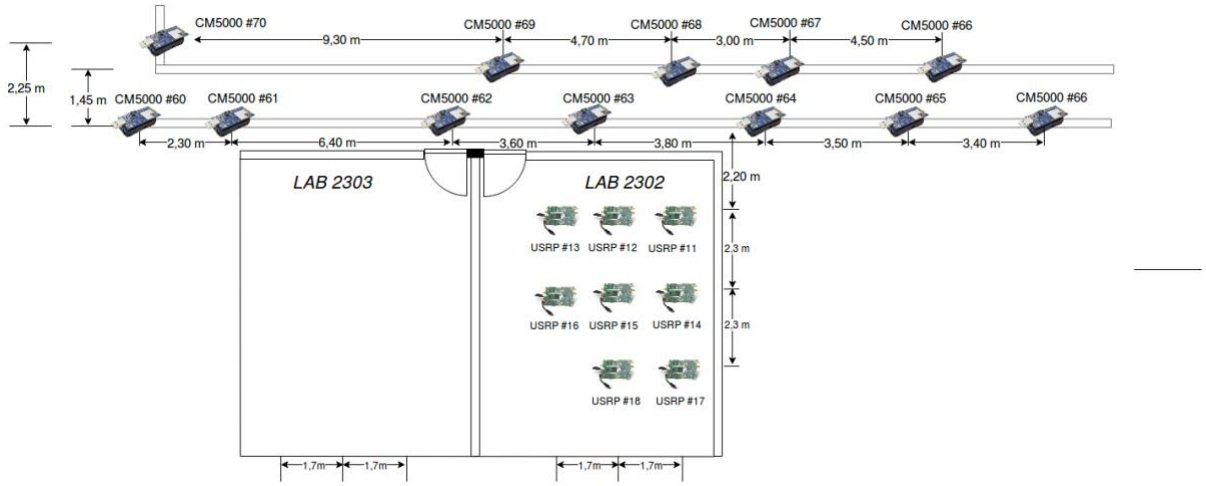


Fig 2 - Map of the placement of the Advanticsys Sensor Nodes.

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.

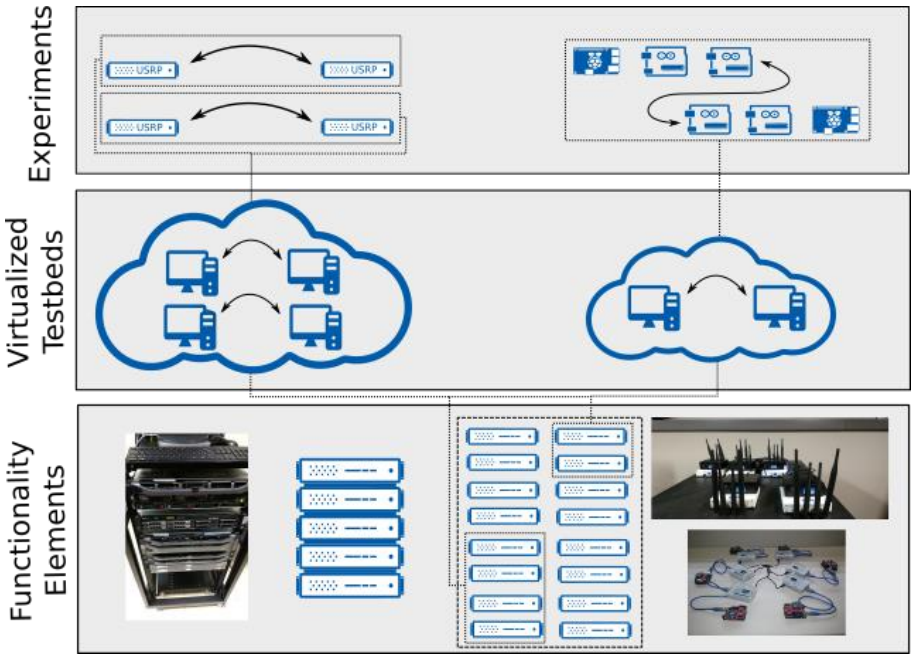


Fig 2. Functional layers

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 WiFi 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 WiFi 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 a 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-
vlan/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 - Raspberries 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-
vlan/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 WiFi

It is possible to use the MiniPCs to perform experiments using WiFi. 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 WiFi 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 WiFi, 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 WiFi 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 WiFi nodes can be allocated as a machine with wireless network card. Note that the WiFi node is selected by using the sliver type **raw-wifi**. The example below presents an RSPEC for WiFi 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 WiFi 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 - Example Experiments

4.2.1 - WiFi node without Ethanol

Below shows an RSPEC for allocating the WiFi 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-
vlan/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 - WiFi 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 WiFi network. We show below an RSPEC that allocates two WiFi 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-
vlan/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 WiFi 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”.**

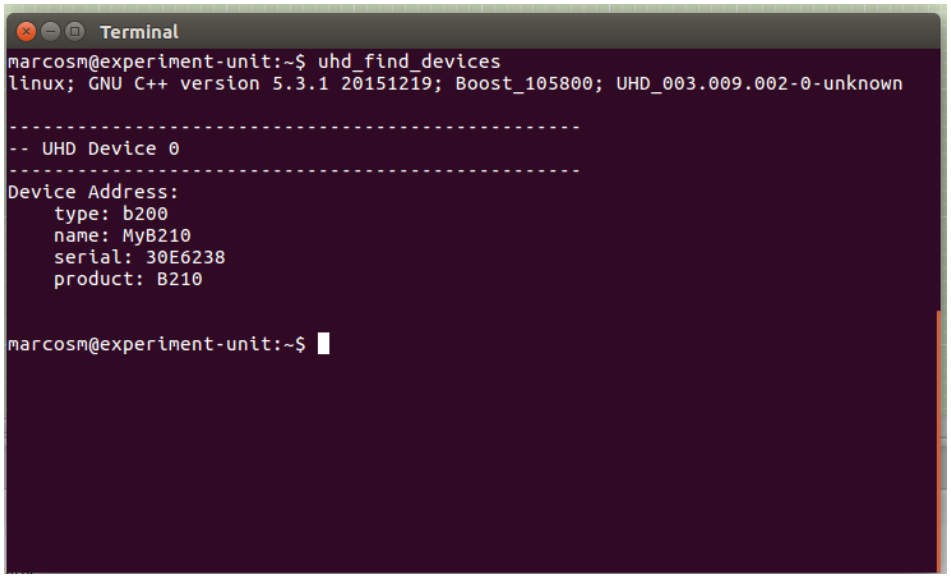
5.2 - Example Experiment

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-
vlan/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>
```

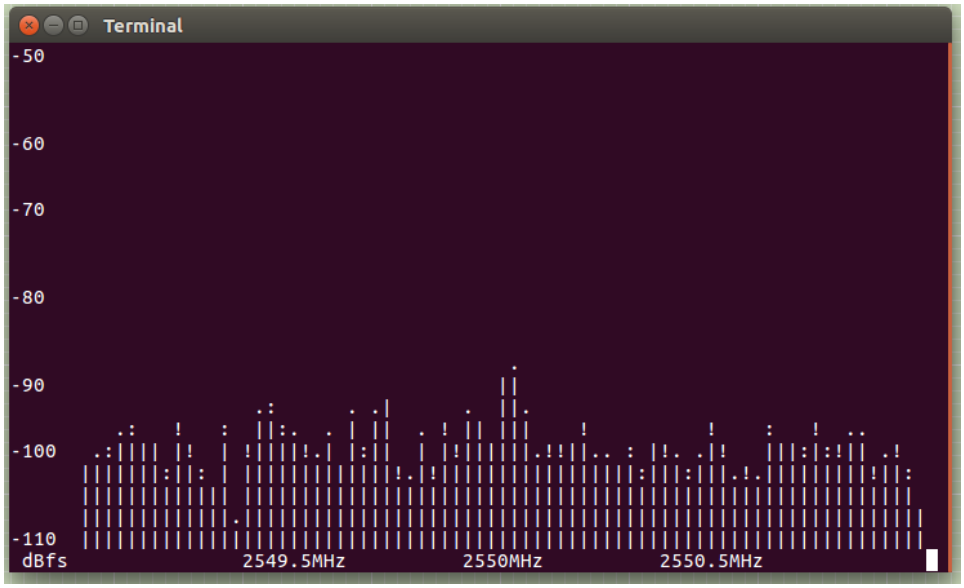
After the VM is instantiated, you can execute commands to perform experiments in the USRP. The first basic command is to check if the equipment is detected. To do this, run the command **uhd_find_devices.** This command output will indicate the model, serial number and name of the USRP that is connected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



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:


Terminal
-50
-60
-70
-80
-90
-100
-110
dBfs
2549.5MHz
2550MHz
2550.5MHz