FUTURE INTERNET TESTBEDS
EXPERIMENTATION BETWEEN
BRAZIL AND EUROPE

Grant Agreement No.: 288356 (FP7)
CNPq Grant Agreement No.: 590022/2011-3

FIBRE-EU

Future Internet testbeds/experimentation between BRazil and Europe – EU

Instrument: ***Collaborative Project***
Thematic Priority: ***[ICT-2011.10.1 EU-Brazil] Research and Development cooperation, topic c) Future Internet – experimental facilities***

D5.6 – Seamless mobility controller application software

Author: WP5
Revised by: Natalia Fernandes
Due date of the Deliverable: Month 30
Actual submission date: 07/05/2014
Start date of project: June 1st 2011 Duration: 34 months
Version: v.1.0

Project co-funded by the European Commission in the 7th Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	✓

FP7 Grant Agreement No.	288356
CNPq Grant Agreement No.:	590022/2011-3
Project Name	Future Internet testbeds/experimentation between BRazil and Europe – EU
Document Name	FIBRE-D5.6-Seamless_mobility_controller_application_software
Document Title	Seamless mobility controller application software
Workpackage	WP5
Authors	Cledson Souza (UFF) Vitor Farias (UFF) Natalia Castro Fernandes (UFF) Luiz Claudio Schara Magalhães (UFF)
Editor	Natalia Castro Fernandes (UFF)
Reviewers	Natalia Castro Fernandes (UFF) Carlos Bermudo (i2CAT)
Delivery Date	07/05/2014
Version	V1.0

Abstract

The aim of Pilot 1 is to develop and validate a local technology pilot for FIBRE facilities through showcases, which have been pre-selected from a number of use cases. The development activities follow a common process based on the requirements analysis of all the pre-defined use cases and the detailed design of the local and federated facilities.

This document describes the developed software to monitor the testbed and to control the developed infrastructure. The use of this software, however, depends also on the replication of the hardware.

TABLE OF CONTENTS

List of Figures

1 Acronyms

DB	Database
IP	Internet Protocol
MRTG	Multi Router Traffic Grapher
NFS	Network File system
OML	ORBIT Measurement Library
PoE	Power over Ethernet
RRDTools	Round Robin Database Tool
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
USB	Universal Serial Bus

2 Scope

This deliverable presents the software developed for FIBRE Use Case 1: Seamless mobility.

3 Reference Documents

[1] F. Fainelli, “The OpenWrt embedded development framework,” in *Proceedings of the Free and Open Source Software Developers European Meeting*, 2008.

[2] T. Oetiker, “RRDtool,” *http://people.ee.ethz.ch/oetiker/webt\\ool/rrdtool*, 2005.

[3] T. Oetiker and D. Rand, “MRTG: The Multi Router Traffic Grapher.,” in *LISA*, 1998, vol. 98, pp. 141–148.

[4] Python Software Foundation, “Python Programming Language,” 2000. [Online]. Available: http://www.python.org/. [Accessed: 01-Apr-2012].

[5] J. D. Brutlag, “Aberrant Behavior Detection in Time Series for Network Monitoring.,” in *LISA*, 2000, pp. 139–146.

[6] Office for National Statistics - UK, “The Holt-Winters Forecasting Method,” *http://www.ons.gov.uk/ons/guide-method/user-guidance/index-of-services/index-of-services-annex-b--the-holt-winters-forecasting-method.pdf*.

[7] http://oml.mytestbed.net/projects/oml/wiki, accessed on 06/05/2014

4 Pilot 1 Software Applications

Pilot 1 consists of hardware and software building blocks, which are strongly correlated. The software applications are divided into three main groups: monitoring system, mobile node control, and demo script.

4.1 Monitoring system

Our goal with the monitoring system is to monitor all spectrum activity in 802.11 frequencies to enable the experimenter to compare data as band, rate or loses obtained in layer 2 experiments with dynamic changes in physical spectrum. Since the testbed is in an open area in the university, the environment is not completely controlled. The dynamic changes in the spectrum could be caused by jammers or emergence of unpredictable devices as cordless phones, microwave ovens, or wireless cameras. A well-known spectrum can still allow experimenters to carefully select specific channels for both, worst and best scenarios. Depending on the experiment, one can choose polluted or no polluted channels. As a bonus, an alert notification could be set up for detect these anomalous behaviours if the experimenter would like to see any unexpected behaviour. Moreover, the application keeps a database that can store power measurements which we intend make available for scientific community.

4.1.1 Building Blocks

As researchers, but even more as a team of engineers, we sought simplicity, efficiency, and cost effectiveness in the conception of our framework. With this in mind we employed, whenever possible, open-source tools and when not, low-cost hardware solutions. In the following subsections we explain each of these building blocks.

4.1.2 Client Nodes

We decided to use the access point TP-Link 842nd among others due its small dimensions, its USB interface (which allow us to use different analyzers in multiple bands), price (US$ 30,00), and also due to the possibility of storage expansion using USB solid state or compatible devices.

All nodes are operating with OpenWrt [1], an open-source operating systems based on Linux and designed for wireless routers. In the future we intend to use others access points with PoE (Power over Ethernet) and other interesting features that make it a suitable platform for monitoring applications (which normally imply unattended operation and infrequent maintenance), among them:

- Programmable wake-up event interface;
- Watch dog timer support.

4.1.3 Spectrum analyzer

We selected the airView from Ubiquiti Networks, a low-cost spectrum parser that consists of a small USB module. The airView combined with the TP-Link 842nd provides spectrum information that is not available to a common wireless interface. It allows, for example, the scanning of all the 2.4 GHz ISM band in steps of 500 kHz and to observe narrow band interferences more accurately. Moreover, since we are using a USB device such as a spectrum analyzer the wireless interface is free to be used as its original purpose: Access points. Making our, before passive monitoring nodes, in multifunction active devices, helping to increase the quality and range of our testbed.

4.1.4 OML

OML [7] is a measurement data collection framework that enables the experimenter to define the measurement points and parameters, collect and pre-process measurements, and organize the collected data into a single database with the experiment's context, avoiding logging files in various formats. The OML framework is a client/server architecture and uses IP stack to report the collected data to the server in real-time. It defines data structures and functions for sending/receiving, encoding/decoding and storing experiment data. OML supports instances of the collection server per experiment to enhance the network scalability and provide reliability of data collection by load balancing and redundancy. An SQL database is used for persistent storage of experiment data that also provide an easy manner to extract information for future analysis. To make it possible, in our scripts we use oml4py to permit database connection and data extraction as we see in Figure 1.

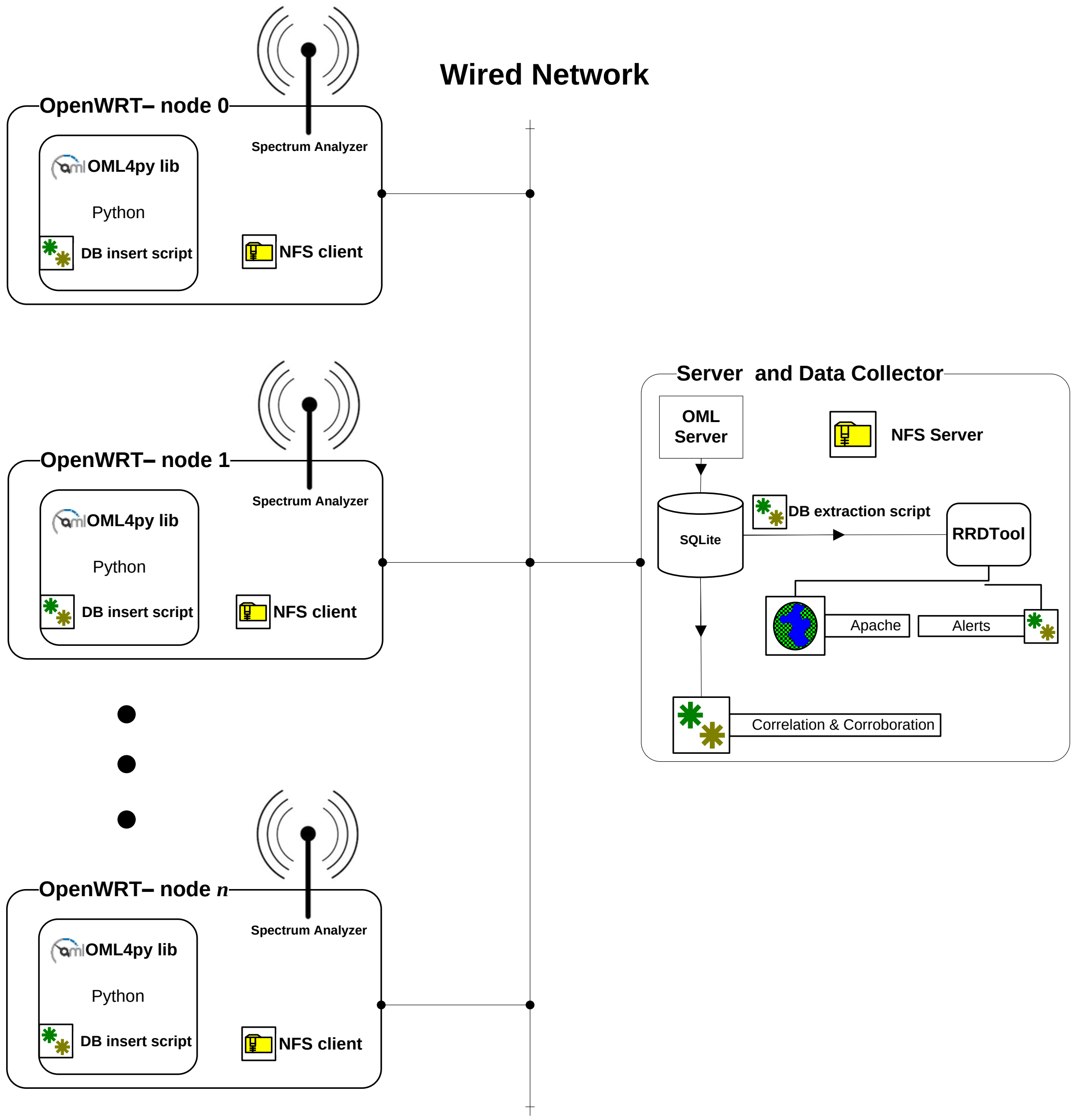

Figure 1 Building blocks of our collecting and monitoring system

4.1.5 RRDTool

The Round Robin Database Tool (RRDTool) [2], not only collects SNMP (Simple Network Management Protocol) data . It can be configured for any kind of data. One could, for example, use it to store and display solar radiation, power dissipation information and spectral measurements. It suffices to have a sensor measure the data and send it to a database for storage. If the data is available RRDtool will store it and provide useful statistics, averages, maximum, minimum, change scale, etc. As it works in a Round Robin, it does not store expired information that has lower value to the purposes of presentation and it does not exceed the system storage capacity (old data gives room for new data). The user can easily change the data calculation to convert bytes per second to bits per second, for example. Also, the database does not confuse a measure valued zero, with an absent measurement, as does many systems (as MRTG [3], for example). All of our scripts (represented with gears icon in Figure 1) were coded in Python [4]. As a result the presented framework is easy to apprehend and to customize.

4.1.6 Detecting abnormal behaviour

A spectrum analyzer such as the one used in our experiments outputs a time series of power measurements taken at regular intervals at each of the monitored frequencies. In our case, this resulted in one measurement every 260 ms for each frequency in the interval from 2399 MHz to 2483 MHz with increments of 0.5 MHz. In order to detect jammers we will monitor this time series in search of aberrant behaviour [5]. The fundamental premise is that during regular utilization the power levels will follow certain trends and seasonal effects (day and night utilization, for instance) but its normal behaviour may be predicted to a certain degree. We will, therefore, employ methods of forecasting to estimate a reasonable range for the next sample and issue alerts if the measured value falls outside this range. As a proof of concept we will make use of a method known as the Holt-Winters forecasting method [6]. The methodology presented needs further refinements, however the preliminary results are promising.

4.1.7 Easy to see (Apache)

In order to give visual clues for the anomalous behaviour and make it more noticeable, the RRDTool was configured to plot (through apache web server) the confidence intervals (the two red lines seen in Figure 2) and to highlight violations (samples that fall outside the expected power level range) as also seen in the Figure 3.

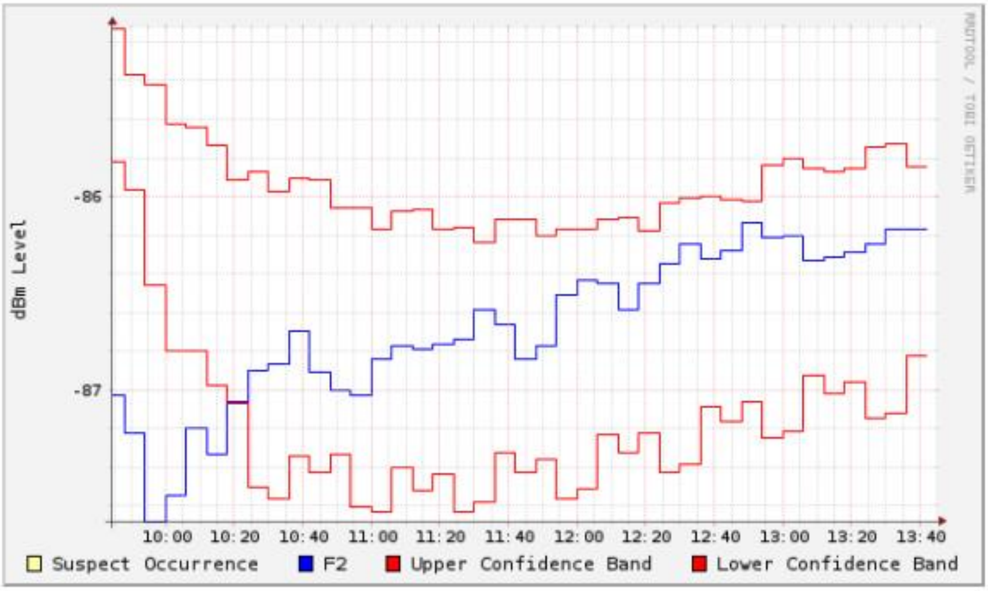

Figure 2: Signal power and confidence bands training

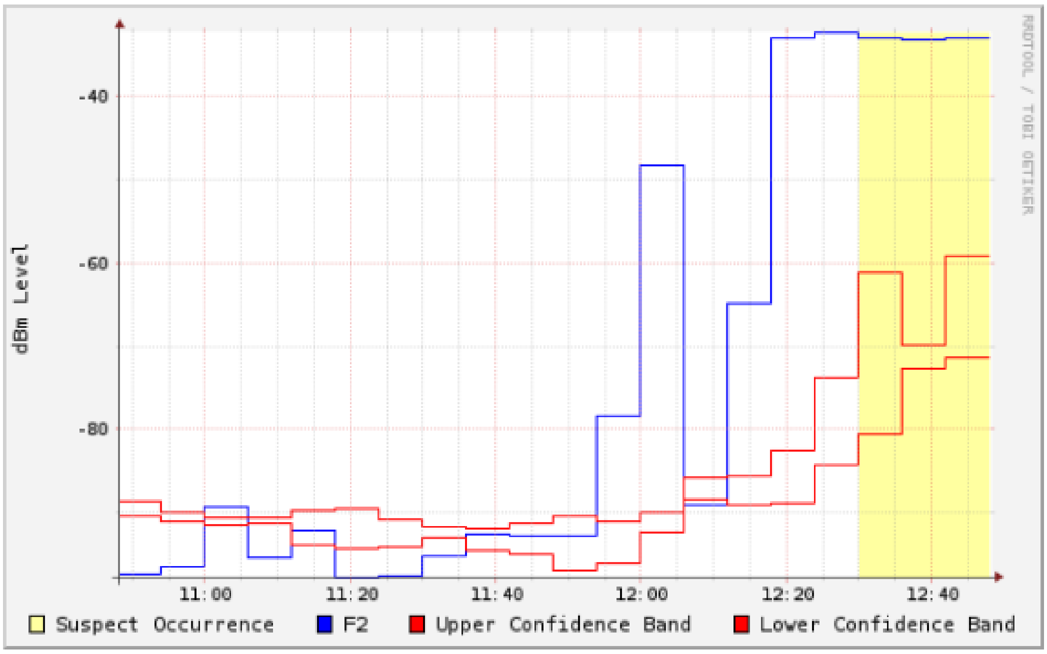

Figure 3: Anomalous behaviour

4.1.8 Code

The developed code for this application is in Section 6.1, in the Annexes.

4.2 Mobile Node Control

The mobile node control is mainly designed in hardware. We used a running belt motor to move a laptop over the corridor of the Engineering School building in UFF. We developed a small circuit which receives commands through a web service.

The first version of this system allows user to move the train forward or backward and stop. A future version will also include the velocity selection control. In order to accomplish the first tests using the mobile node, we also developed a small web interface to control the train.

4.2.1 Code

The developed code for this application is on Section 6.2, in the Annexes. There, we show how to start and stop the hardware through command line commands and also the html for accessing the developed web page.

5 Conclusions

The application software for monitoring system is complete and allows the testbed users to observe whether an external source interfered or not in the experiment. Also, it allows the user to study the infrastructure in order to decide the best moment and the best channel to develop the experiments.

The mobile train control application is still under development and we wish to insert other facilities in order to help the experimenter to control the train. By now, it is possible to control the mobile node in real time, but this kind of control is suitable only for local experimenters. For remote experimenters, which are the main focus on FIBRE, we will allow the experimenter to make a half trip or a round trip for security reasons.

The scripts to show a handoff experiment are still under development.

6 Annexes

6.1 Monitoring System

We have three codes running in our instrumentation testbed.

1 – nodeclient.py – responsible for get data from spectrum and insert in a Sqlite database via OML

2 – rrdsqlite.py – responsible for extract data from database and create RRDtools time series.

3 – graph.py – responsible for create html files to follow spectrum behavior.

```
#-------------------------------------------------------------------
# Name:          nodeclient.py
# Purpose:        Get and treat power spectral information from airview
(173 channels step 500kHz)
#                and insert (time, power in dBm ) in a sqlite database.
#
# Author:        Cledson Sousa, David Lacos and Manuel Carraminhana
#
# Created:       06/10/2013
# Licence:       GPL
#-------------------------------------------------------------------
import os, sys, time, serial, sched, io, rrdtool
import oml4py
import socket
from oml4py import OMLBase
myHostName = socket.gethostname()
#----------------------OML  CONFIGURATION------------------------------
-----
x=oml4py.OMLBase("Spectrum",
myHostName,"miniITX_Airwiew2","tcp:localhost:3003")
Potencia_registrada:string Hora_registrada:string")
x.addmp("power_level", "Frequency:string Power:string Time:string")
x.start()

#-------------------SERIAL  PORT  CONFIGURATION  (AIRVIEW2)---------------
-----
serial.PARITY_NONE,timeout=0,rtscts=0,bytesize=serial.EIGHTBITS,stopbi
ts=serial.STOPBITS_ONE,xonxoff=0)
serialport    =    serial.Serial("/dev/ttyACM0",    115200,    parity    =
serial.PARITY_NONE,timeout=1,rtscts=1,bytesize=serial.EIGHTBITS,stopbi
ts=serial.STOPBITS_ONE,xonxoff=0)
sio = io.TextIOWrapper(io.BufferedRWPair(serialport, serialport))
serialport.flushInput()
print "Starting Airview2 device..."
```

```
print "Airview2 starts scanning on 2.399GHz and ends at 2.485GHz. There are 86KHz to be scanned, then there will be 1 initial value and 172 sampled values."
serialport.write("init\n");
print serialport.read(serialport.inWaiting());
print "Getting data from Airview2..."
time.sleep(2)

#----------------------------BEGIN THE EXPERIMENT---------------------------------
numSweepPerMean = 1#The number of sweeps stored before calculate de mean to store in the DB
sweep_stock = []#This will save 100 sweeps, then the mean for each frequency (column) will be calculated
start",'N',data_sources,'RRA:AVERAGE:0.5:1:600','RRA:MAX:0.5:1:600',)
cnt=1
serialport.write("bs\n");
while 1 == 1:
    time.sleep(0.2)# Airview2 takes 260ms per sweep according the Ubiquity website.
    raw_data = serialport.readline()
    sweep_data = raw_data.lstrip('scan|0,')#Delete the string "scan|0,". Instead the next line...
    #sweep_list = sweep_raw_data.split("scan|0,");

    sweep_values = sweep_data.split(' ')#After this command. The vector keeps "\n" in the last value (174)... it would have length=173...
    del sweep_values[-1]#...so, this "\n" is deleted

    sweepFloat = [ float(s) for s in sweep_values ]#Convert to float each sample (string)
    sweepInt = [ int(s) for s in sweep_values ]
    #-------- Processing only the right sweeps (with 173 samples)---------
    if len(sweepInt)==173:
    #-------- After cnt values the last one is stored in the DB and RRD -----
        if cnt==4:

            #------ Saving data to the DB (OML) -----------------------------
            frequency = 2399
            f_offset = 0.5
            samples_timestamp = time.mktime(time.localtime())#Seconds since the epoc: 1970 (Unix)

            sweepRRD = ":".join(str(n) for n in sweepInt)
            print samples_timestamp
            rrdtool.update('test.rrd','N:'+sweepRRD)
            for f_power_level in sweepInt:
                x.inject("power_level", [frequency, f_power_level, samples_timestamp])
                frequency = frequency+f_offset
            print "Sweep data stored. 173 values.\n\n"
            cnt=0
    cnt=cnt+1
x.close()
#------------------------------END------------------------------------
```

```
#-----------------------------------------------------------------
# Name:        rrdsqlite.py
# Purpose:      for extract data from database and create RRDtools time
series.
#
# Author:      Cledson Sousa, David Lacos and Manuel Carraminhana
#
# Created:     06/10/2013
# Licence:     GPL
#-----------------------------------------------------------------

#!/usr/bin/python
import sys
import sqlite3
import os.path
import socket
import glob
import time
import rrdtool


#Find database files
def findDBFiles():
    sq3Files = glob.glob('/var/lib/oml2/node*.sq3')
    if sq3Files==[]:
        print("No sqlite databases found. Check /var/lib/oml2")
    print(sq3Files)
    return sq3Files

#Get the node name using sq3 filename
def getNodeName(fileName):
    file = fileName.split("/")[4]
    node = file.split(".")[0]
    return node

#Create RRDTool database
def createRRDDB(nodeName):
    string1 = "DS:F"
    string2 = ":GAUGE:300:U:U"
    data_sources = [string1+`i`+string2 for i in range(173)]
    rrdtool.create(nodeName+".rrd",
            "--step","360",
            "--start",'N',
            data_sources,
            'RRA:AVERAGE:0.5:360:576',
            'RRA:HWPREDICT:576:0.1:0.0035:5',
            'RRA:MAX:0.5:1:576',)

def getDBData(dbFilename,nodeName):
    db = sqlite3.connect(dbFilename)
    db.text_factory = str#Making sqlite3 return bytestrings instead
Unicode when using SELECT
    cursor = db.cursor()
#   assert type(lastEntryTime[0]) is str
#   print(lastEntryTime)
#           sqliteSelect    =    "SELECT    Frequency,Power,Time    FROM
'Spectrum_power_level'  WHERE  Time  ="+lastEntryTime[0]+"  ORDER  BY
Frequency"
```

```
    sqliteSelect = "SELECT Frequency,Power,Time FROM
'Spectrum_power_level' WHERE Time = (SELECT MAX(Time) FROM
'Spectrum_power_level') ORDER BY Frequency"
#            cursor.execute("SELECT Frequency,Power,Time FROM
'Spectrum_power_level' ORDER BY Time DESC LIMIT 173")
#   rowtest = cursor.fetchone()
    cursor.execute(sqliteSelect)
    all_rows = cursor.fetchall()
    if len(all_rows)==173:
        data = [row[1] for row in all_rows]
        timeStamp = row[2]
        dataFloat = [float(d) for d in data] #Convert string to float
        dataInt = [ int(s) for s in dataFloat ]
        #if len(dataInt)==173:
        print("Updating "+nodeName+" using "+dbFilename+" "+timeStamp)
        dataReady = ":".join(str(n) for n in dataInt)
        if not os.path.exists(nodeName+'.rrd'):
            print(nodeName+".rrd does not exists (new node?).
Creating...")
            createRRDDB(nodeName)
        try:
            rrdtool.update(nodeName+'.rrd',timeStamp+':'+dataReady)
        except:
            print "Oops"
    db.close()
    return

#Main loop
while 1 == 1:
    sq3DBs = findDBFiles()#Find sqlite databases
    for dbFilename in sq3DBs:
        nodeName = getNodeName(dbFilename)
        getDBData(dbFilename,nodeName)
    time.sleep(5)
#------------------------------END----------------------------------

#-----------------------------------------------------------------
# Name:       graph.py
# Purpose:     create htm files to follow through apache the spectrum
behavior.
#
# Author:     Cledson Sousa, David Lacos and Manuel Carraminhana
#
# Created:    06/10/2013
# Licence:      GPL
#-----------------------------------------------------------------

import os
import sys
import rrdtool
import glob
img_path="/var/www/intel/"
actualPath=os.getcwd()
#sweeps = ','.join(str(i) for i in xrange(1,174))
def findRRDFiles():
        rrdDBs = glob.glob('node*.rrd')
        if rrdDBs==[]:
                print("No RRD databases found.")
```

```
        return rrdDBs

def createGraph(nodeName):
    rrdfile=actualPath+"/"+nodeName+".rrd"
    graph=rrdtool.graph(img_path+nodeName+".png",
    "--alt-autoscale",
    "--alt-y-grid",
    "--rigid",
    "-v dBm Level",
    "--slope-mode",
    "--font","LEGEND:7",
    "-h 200",
    "--start", "-1h",#defs+lines+
    "--title",nodeName+" - Last 1 hour",
    "DEF:freq1="+rrdfile+":F1:MAX",
    "DEF:freq2="+rrdfile+":F2:MAX",
    "DEF:freq3="+rrdfile+":F3:MAX",
    "DEF:freq4="+rrdfile+":F4:MAX",
    "DEF:freq5="+rrdfile+":F5:MAX",
    "LINE:freq1#0000FF:F1 ",
    "LINE:freq2#00FF00:F2 ",
    "LINE:freq3#FF0000:F3 ",
    "LINE:freq4#00FFFF:F4 ",
    "LINE:freq5#FFFF00:F5",
    "GPRINT:freq1:MAX:Avg F1\: %6.0lf")
    if graph:
        print(rrdtool.error())
    return

#Main Loop
nodes = findRRDFiles()
for node in nodes:
    print(node)
    nodeName = node.split(".")[0]
    createGraph(nodeName)

rrdfile = "/home/intel/spectrum/node02.rrd"
ret = rrdtool.graph( img_path+"ch2.png",
 "--alt-autoscale",
 "--alt-y-grid",
 "--rigid",
 "-v dBm Level",
 "--slope-mode",
 "--font","LEGEND:7",
 "--start", "-12h",
 "--title","Channel 1 frequencies (2.401 - 2.423 GHz) Central 2412 GHz
",
 "DEF:freq1="+rrdfile+":F1:MAX",
 "DEF:freq2="+rrdfile+":F2:MAX",
 "DEF:freq3="+rrdfile+":F3:MAX",
 "DEF:freq4="+rrdfile+":F4:MAX",
 "DEF:freq5="+rrdfile+":F5:MAX",
 "DEF:freq7="+rrdfile+":F7:MAX",
 "DEF:freq8="+rrdfile+":F8:MAX",
 "DEF:freq9="+rrdfile+":F9:MAX",
 "DEF:freq10="+rrdfile+":F10:MAX",
 "DEF:freq11="+rrdfile+":F11:MAX",
 "DEF:freq12="+rrdfile+":F12:MAX",
 "DEF:freq13="+rrdfile+":F13:MAX",
```

```
 "DEF:freq14="+rrdfile+":F14:MAX",
 "DEF:freq15="+rrdfile+":F15:MAX",
 "DEF:freq16="+rrdfile+":F16:MAX",
 "DEF:freq17="+rrdfile+":F17:MAX",
 "DEF:freq18="+rrdfile+":F18:MAX",
 "DEF:freq19="+rrdfile+":F19:MAX",
 "DEF:freq20="+rrdfile+":F20:MAX",
 "COMMENT:Frequency \t  Max. \t  Last\\n",
 "LINE:freq1#00FF00:2.4115GHz\t",
 "GPRINT:freq1:MAX: %3.0lf\t",
 "GPRINT:freq1:LAST: %3.0lf\t",
 "COMMENT:\\n",
 "LINE:freq2#0000FF:2.4120GHz\t",
 "GPRINT:freq2:MAX: %3.lf\t",
 "GPRINT:freq2:LAST: %3.lf\t",
 "COMMENT:\\n",
 "LINE:freq3#FF0000:2.4125GHz\t",
 "GPRINT:freq3:MAX: %3.lf\t",
 "GPRINT:freq3:LAST: %3.lf\t",
 "COMMENT:\\n",
 "LINE:freq4#00FFFF:2.4130GHz\t",
 "GPRINT:freq4:MAX: %3.lf\t",
 "GPRINT:freq4:LAST: %3.lf\t",
 "COMMENT:\\n",
 "LINE:freq5#FFFF00:2.4135GHz\t",
 "GPRINT:freq5:MAX: %3.lf\t",
 "GPRINT:freq5:LAST: %3.lf\t",
 "COMMENT:\\n")
if ret:
 print rrdtool.error()
ret = rrdtool.graph( img_path+"channel2.png",
 "--start", "-0.5h",
 "DEF:freq41="+rrdfile+":F41:AVERAGE",
 "DEF:freq42="+rrdfile+":F42:AVERAGE",
 "DEF:freq43="+rrdfile+":F43:AVERAGE",
 "DEF:freq44="+rrdfile+":F44:AVERAGE",
 "DEF:freq45="+rrdfile+":F45:AVERAGE",
 "LINE:freq41#0000FF:F41 ",
 "LINE:freq42#00FF00:F42 ",
 "LINE:freq43#FF0000:F43 ",
 "LINE:freq44#00FFFF:F44 ",
 "LINE:freq45#FFFF00:F45",
 "GPRINT:freq41:MAX:Avg F41\: %6.0lf")
if ret:
 print rrdtool.error()
ret = rrdtool.graph( img_path+"channel3.png",
 "--start", "-1h",
 "DEF:freq81="+rrdfile+":F81:AVERAGE",
 "DEF:freq82="+rrdfile+":F82:AVERAGE",
 "DEF:freq83="+rrdfile+":F83:AVERAGE",
 "DEF:freq84="+rrdfile+":F84:AVERAGE",
 "DEF:freq85="+rrdfile+":F85:AVERAGE",
 "LINE:freq81#0000FF:F81",
 "LINE:freq82#00FF00:F82",
 "LINE:freq83#FF0000:F83",
 "LINE:freq84#00FFFF:F84",
 "LINE:freq85#FFFF00:F85",
 "GPRINT:freq81:MAX:Avg F81\: %6.0lf")
if ret:
```

```
 print rrdtool.error()
ret = rrdtool.graph( img_path+"channel6.png",
 "--start", "-1h",
 "--title","2.412 GHz - Last 2 hours",
  "--alt-autoscale",
  "--alt-y-grid",
  "--rigid",
  "-w 500",
  "-v dBm Level",
  "-h 300",
 "DEF:freq2="+rrdfile+":F2:MAX",
 "DEF:pred="+rrdfile+":F2:HWPREDICT",
 "DEF:dev="+rrdfile+":F2:DEVPREDICT",
 "DEF:fail="+rrdfile+":F2:FAILURES",
 "CDEF:upper=pred,dev,2,*,+",
 "CDEF:lower=pred,dev,2,*,-",
 "CDEF:scaledupper=upper,1,*",
 "CDEF:scaledlower=lower,1,*",
 "LINE:freq2#0000FF:2.412 GHz",
 "TICK:fail#ffffa0:1.0:Suspect Occurrence",
  "LINE1:upper#ff0000:Upper Confidence Band",
 "LINE1:lower#ff0000:Lower Confidence Band",)
if ret:
 print rrdtool.error()
ret = rrdtool.graph( img_path+"channel1_week.png",
 "--start", "-168h",
 "--title","2.412 GHz - Week",
  "--alt-autoscale",
  "--alt-y-grid",
  "--rigid",
  "-w 500",
  "-v dBm Level",
  "-h 300",
 #"DEF:obs="+rrdfile+":F121:AVERAGE",
 "DEF:freq2="+rrdfile+":F2:MAX",
 "DEF:pred="+rrdfile+":F2:HWPREDICT",
 "DEF:dev="+rrdfile+":F2:DEVPREDICT",
 "DEF:fail="+rrdfile+":F2:FAILURES",
 "LINE:freq2#0000FF:2.412 GHz",
 "TICK:fail#ffffa0:1.0:Suspect Occurrence",
 "CDEF:upper=pred,dev,2,*,+",
 "CDEF:lower=pred,dev,2,*,-",
 "CDEF:scaledupper=upper,1,*",
 "CDEF:scaledlower=lower,1,*",
 #"LINE2:scaledobs#0000ff: Signal Strength",
 "LINE1:upper#ff0000:Upper Confidence Band",
 "LINE1:lower#ff0000:Lower Confidence Band",)
# "GPRINT:freq121:MAX:Avg F1\: %6.01f")
if ret:
 print rrdtool.error()
#-----------------------------END-----------------------------------
```

6.2 Mobile Node Control

6.2.1 Controller code

```
#Commands for shell

#LED LAN3 is power enable
#LED LAN2 is direction control

#ACTION HALT
#Turns the motor off
echo 0 >
/sys/devices/platform/leds-gpio/leds/tp-link\:green\:lan3/brightness


#ACTION FORWARD
#makes the train go forward
echo 0 >
/sys/devices/platform/leds-gpio/leds/tp-link\:green\:lan3/brightness

sleep 1
#forward direction
echo 0 >
/sys/devices/platform/leds-gpio/leds/tp-link\:green\:lan2/brightness
sleep 1
#turns the motor on
echo 1 >
/sys/devices/platform/leds-gpio/leds/tp-link\:green\:lan3/brightness


#ACTION BACKWARD
#makes the train go backward

#Turns the motor off
echo 0 >
/sys/devices/platform/leds-gpio/leds/tp-link\:green\:lan3/brightness

sleep 1
#Backward direction
echo 1 >
/sys/devices/platform/leds-gpio/leds/tp-link\:green\:lan2/brightness
sleep 1
#turns the motor on
echo 1 >
/sys/devices/platform/leds-gpio/leds/tp-link\:green\:lan3/brightness
```

6.2.2 Web interface

This is the web page developed for the local tests using the train. The CGIs calls the scripts for forward, backward, and halt described in the previous section.

```
<html>

<head>

</head>

<body bgcolor="#000000">

  <div align="center">

<table>
<tr>
<td><p align="center" style="height: 30%; width: 99%;font: bold 100px
Arial;color:white">T1</p></td>
<td><p align="center" style="height: 30%; width: 99%;font: bold 100px
Arial;color:white">T2</p></td>
<td><p align="center" style="height: 30%; width: 99%;font: bold 100px
Arial;color:white">T3</p></td>
<td><p align="center" style="height: 30%; width: 99%;font: bold 100px
Arial;color:white">T4</p></td>
</tr>
 <tr>
<td>
<form method="LINK" action="/cgi-bin/T1on.cgi">
<input type="submit"  style="height: 30%; width: 99%;font: bold 100px
Arial" value="ON"/>
</form>
</td>

<td>
<form method="LINK" action="/cgi-bin/T2on.cgi">
<input type="submit"  style="height: 30%; width: 99%;font: bold 100px
Arial" value="ON"/>
</form>
</td>

<td>
<form method="LINK" action="/cgi-bin/T3on.cgi">
<input type="submit"  style="height: 30%; width: 99%;font: bold 100px
Arial" value="ON"/>
</form>
</td>

<td>
<form method="LINK" action="/cgi-bin/T4on.cgi">
<input type="submit"  style="height: 30%; width: 99%;font: bold 100px
Arial" value="ON"/>
</form>
</td>
```

```
</tr>

<tr>
<td>
 <form method="LINK" action="/cgi-bin/T1off.cgi">
<input type="submit" style="height: 30%; width: 99%;font: bold 100px Arial"
value="OFF"/>
</form>
</td>

<td>
 <form method="LINK" action="/cgi-bin/T2off.cgi">
<input type="submit" style="height: 30%; width: 99%;font: bold 100px Arial"
value="OFF"/>
</form>
</td>

<td>
 <form method="LINK" action="/cgi-bin/T3off.cgi">
<input type="submit" style="height: 30%; width: 99%;font: bold 100px Arial"
value="OFF"/>
</form>
</td>

<td>
 <form method="LINK" action="/cgi-bin/T4off.cgi">
<input type="submit" style="height: 30%; width: 99%;font: bold 100px Arial"
value="OFF"/>
</form>
</td>

</tr>

</table>

</div>

</body>

</html>
```

"*This work makes use of results produced by the FIBRE project, co-funded by the Brazilian Council* for Scientific and Technological Development (CNPq) and by the *European Commission within its Seventh Framework Programme.”*

END OF DOCUMENT