**Contents**

[I. Introduction 2](#_Toc459039958)

[II. Install an Run OpenWSN on Linux 3](#_Toc459039959)

[1. Install 3](#_Toc459039962)

[2. Download OpenWSN 3](#_Toc459039963)

[3. Now with real hardware 13](#_Toc459039971)

[III. Component Pictures 26](#_Toc459039984)

[1. OpenStack 26](#_Toc459039985)

[2. IE stack 27](#_Toc459039986)

[3. OpenVisualizer Code Organization 28](#_Toc459039987)

[IV. Sensor Cloud Testbed 31](#_Toc459039989)

[1. Network setup 31](#_Toc459039990)

[2. OpenStack setup (If you don’t know details how to install, you can ask Minsik or Young Ki brother) 31](#_Toc459039991)

[2.1. OpenStack model 31](#_Toc459039992)

[2.2. Hardware requirement 31](#_Toc459039993)

[2.3. Software requirement 31](#_Toc459039994)

[2.4. How to install 31](#_Toc459039995)

[2.5. Install VM OS 33](#_Toc459039996)

[2.6. Install Intel NUC OS 34](#_Toc459039999)

[2.7. Enable Distributed Scheduling SF0 34](#_Toc459040001)

[3. Control Setup (Remote control your mote through NUC) 34](#_Toc459040005)

|  |  |  |
| --- | --- | --- |
| **Author** | **Action** | **Date** |
| Phan Duy Thang | Created | 2016/08/05 |
|  |  |  |
|  |  |  |

1. **Introduction**

Hello! My name is Phan Duy Thang - Master Student - 08/2016. This is document created by me to highlight as follows:

* How to install and use 6TiSCH network environment.
* How to setup a Sensor-Cloud system.

Some main links you should refer here to download:

*6TiSCH Guide.docx*:

[\\114.71.50.192\personal\99.Backup\Phan\_Duy\_Thang\6TiSCH\](file:///\\114.71.50.192\personal\99.Backup\Phan_Duy_Thang\6TiSCH\)

*OpenWSN Source Code*:

<https://github.com/openwsn-berkeley/openwsn-fw>

<https://github.com/openwsn-berkeley/openwsn-sw>

<https://github.com/openwsn-berkeley/coap>

1. **Install an Run OpenWSN on Linux**

**Note:**  Because 6TiSCH source code is always changed, you should refer the following link to install:

<https://openwsn.atlassian.net/wiki/display/OW/Kickstart+Linux>

1. **Install**

* Lastest Ubuntu Desktop : 14.04 // will get lastest gcc version
* Packets :

apt-get update && apt-get upgrade

apt-get install vim

apt-get install git

apt-get install python-dev

apt-get install python-pip

apt-get install python-tk

pip install bottle

pip install PyDispatcher

apt-get install scons

apt-get install binutils-msp430 gcc-msp430 msp430-libc mspdebug

apt-get install wireshark

apt-get install wine

apt-get install doxygen

apt-get install graphviz

*Note: In case Ubuntu use IPv6 to download packet (It spends a lot of time). You have to disable IPv6 in Ubuntu*

*sudo gedit /etc/sysctl.conf*

# IPv6 disabled

net.ipv6.conf.all.disable\_ipv6 = 1

net.ipv6.conf.default.disable\_ipv6 = 1

net.ipv6.conf.lo.disable\_ipv6 = 1

sudo sysctl -p

Refer: <http://www.binarytides.com/disable-ipv6-ubuntu/>

After disabling IPv6 for downloading, you have to enable it again for tune interfaces.

## Download OpenWSN

OpenWSN is a collection of repositories hosted on GitHub. We will download and use the following:

* <https://github.com/openwsn-berkeley/openwsn-fw> holds the firmware source code which runs on the (possibly emulated) motes
* <https://github.com/openwsn-berkeley/openwsn-sw> holds the software source code which runs on your computer
* <https://github.com/openwsn-berkeley/coap> is a Python module which implements CoAP

We will download these repositories side-by-side in an openwsn/ directory in your desktop using Git.

|  |
| --- |
| ~$ cd Desktop/  ~/Desktop$ mkdir openwsn  ~/Desktop$ cd openwsn/  ~/Desktop/openwsn$ git clone https://github.com/openwsn-berkeley/openwsn-fw.git  [...]  ~/Desktop/openwsn$ git clone https://github.com/openwsn-berkeley/openwsn-sw.git  [...]  ~/Desktop/openwsn$ git clone https://github.com/openwsn-berkeley/coap.git  [...] |

At any time, you can make sure that you are using the latest code by using git pull:

|  |
| --- |
| ~$ cd Desktop/  ~/Desktop$ cd openwsn/  ~/Desktop/openwsn$ cd openwsn-sw/  ~/Desktop/openwsn/openwsn-sw$ git pull  Already up-to-date.  ~/Desktop/openwsn/openwsn-sw$ cd ..  ~/Desktop/openwsn$ cd openwsn-fw/  ~/Desktop/openwsn/openwsn-fw$ git pull  remote: Counting objects: 440, done.  remote: Compressing objects: 100% (273/273), done.  remote: Total 440 (delta 283), reused 212 (delta 128)  Receiving objects: 100% (440/440), 127.67 KiB | 99 KiB/s, done.  Resolving deltas: 100% (283/283), done.  From https://github.com/openwsn-berkeley/openwsn-fw     87f68a6..94f18cd  develop    -> origin/develop     116e0f4..8e36607  develop\_FW-186 -> origin/develop\_FW-186  Updating 87f68a6..94f18cd  Fast-forward   firmware/openos/openwsn/03a-IPHC/iphc.c        |  792 ++++++++++++++----------   firmware/openos/openwsn/03a-IPHC/iphc.h        |   98 +--   firmware/openos/openwsn/03b-IPv6/forwarding.c  |  451 ++++++++------   firmware/openos/openwsn/03b-IPv6/forwarding.h  |   28 +-   firmware/openos/openwsn/07-App/rleds/rleds.c   |   10 +-   firmware/openos/projects/python/SConscript.env |   12 +-   6 files changed, 814 insertions(+), 577 deletions(-) |

# Running a Simulation

Frankly, it's a bit strange to start using OpenWSN with a simulation, since the firmware is really meant (and written) to run on real motes. **But**, not everyone has hardware, not always the same hardware, etc. So to make things nice and easy, we'll start by simulation. Oh, and the simulated code behaves exactly the same as the real code, so what you see now is what you'll get with real hardware.

## Prepare

Before we can start running a simulation, we need to compile the firmware as a Python extension. This is all explained in the [OpenSim](https://openwsn.atlassian.net/wiki/display/OW/OpenSim) page if you want to know what's going on.

**Oops!**

Before you can go on, you need to install the Python header files:

|  |
| --- |
| ~/Desktop/openwsn/openwsn-fw$ sudo apt-get install python-dev |

**Oops!**

Before you can go on, you need to install [SCons](http://www.scons.org/):

|  |
| --- |
| ~/Desktop/openwsn$ sudo apt-get install scons |

|  |
| --- |
| ~/Desktop/openwsn$ cd openwsn-fw/  ~/Desktop/openwsn/openwsn-fw$ cd  [...]  Archiving build/python\_gcc/bsp/boards/python/libbsp.a  Indexing  build/python\_gcc/bsp/boards/python/libbsp.a  Linking (shared)   firmware/openos/projects/common/oos\_openwsn.so  scons: done building targets. |

This step compile the complete OpenWSN firmware as a Python extension module (a form of shared library) which the simulation environment can import at run-time.

The extension module is at ~/Desktop/openwsn/openwsn-fw/firmware/openos/projects/common/oos\_openwsn.so, **no need to move it**.

## Simulate

You can now start a simulation. Running a simulation just means taking the usual software which runs on your computer (and call "openvisualizer"), but running it in simulation mode. That is, instead of the openvisualizer connecting to read motes, it connects to emulated mote code actually running on you machine.

**Oops!**

Before you can go on, you need to install some Python packages:

|  |
| --- |
| ~/Desktop/openwsn/openwsn-sw/software/openvisualizer$ sudo apt-get install python-pip  ~/Desktop/openwsn/openwsn-sw/software/openvisualizer$ sudo pip install bottle  ~/Desktop/openwsn/openwsn-sw/software/openvisualizer$ sudo pip install PyDispatcher |

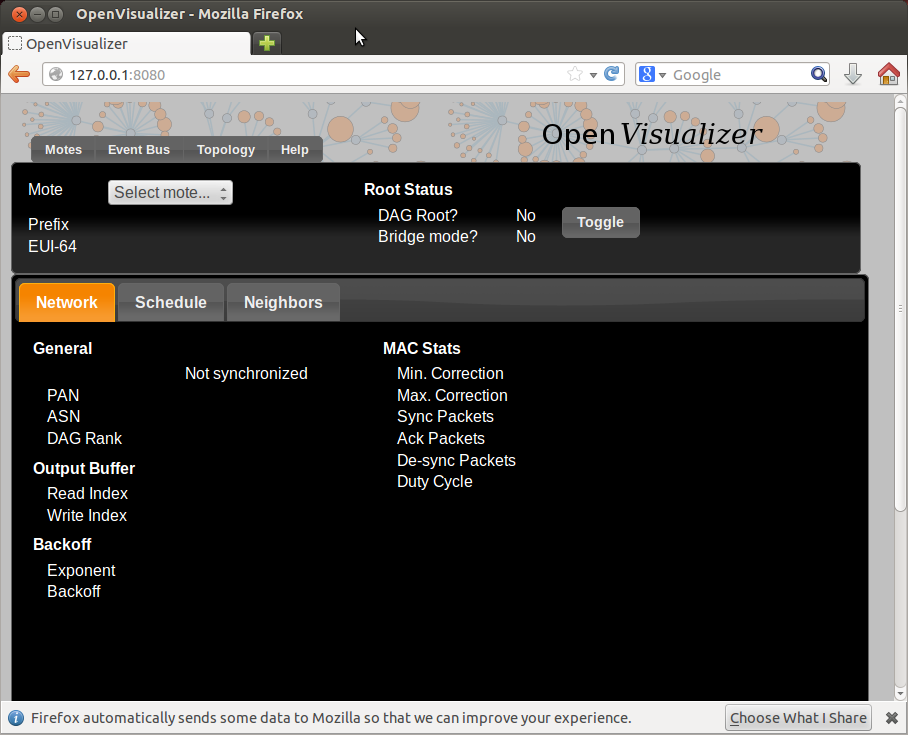
Start a simulation:

|  |
| --- |
| ~/Desktop/openwsn/openwsn-sw/software/openvisualizer$ sudo scons runweb –sim [--simCount <number of simulated nodes>]  scons: Reading SConscript files ...   \_\_\_                 \_ \_ \_  \_\_\_  \_ \_  | . | \_\_\_  \_\_\_ .\_ \_ | | | |/ \_\_>| \ |  | | || . \/ .\_>| ' || | | |\\_\_ \|   |  `\_\_\_'|  \_/\\_\_\_.|\_|\_||\_\_/\_/ <\_\_\_/|\_\\_|       |\_|                  openwsn.org  scons: done reading SConscript files.  scons: Building targets ...  Copy("bin/openVisualizerApp/sim\_files", "../../../openwsn-fw/firmware/openos/bsp/boards/python/openwsnmodule\_obj.h")  Mkdir("bin/openVisualizerApp/sim\_files/linux")  Copy("bin/openVisualizerApp/sim\_files/linux/oos\_openwsn-x86.so", "../../../openwsn-fw/firmware/openos/projects/common/oos\_openwsn.so")  Copy("bin/openVisualizerApp/sim\_files", "../../../openwsn-fw/firmware/openos/projects/common/oos\_openwsn.so")  Delete("build/runui/web\_files")  Mkdir("/home/thomas/Desktop/openwsn/openwsn-sw/software/openvisualizer/build/runui")  Copy("build/runui/web\_files", "bin/openVisualizerApp/web\_files")  Delete("build/runui/sim\_files")  Mkdir("/home/thomas/Desktop/openwsn/openwsn-sw/software/openvisualizer/build/runui")  Copy("build/runui/sim\_files", "bin/openVisualizerApp/sim\_files")  uiRunner(["bin/openVisualizerApp/openVisualizerWeb"], ["bin/openVisualizerApp/openVisualizerWeb.py"])  Child PID is 5144  scons: done building targets.  thomas@Thomas-X61s:~/Desktop/openwsn/openwsn-sw/software/openvisualizer$ ioctl(TUNSETIFF): Device or resource busy  created following virtual interface:  4: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UNKNOWN qlen 500      link/none      inet6 bbbb::1/64 scope global         valid\_lft forever preferred\_lft forever      inet6 fe80::1/64 scope link         valid\_lft forever preferred\_lft forever |

You need to run the openvisualizer with "sudo" since the Python program will create a tun interface.

That's it, an OpenWSN simulation is now running your computer!

Open <http://127.0.0.1:8080/> to see the web interface of OpenWSN



Open the "Topology" tab:

* left click on a mote to move it around
* right-click on two motes to connect them with a wireless link

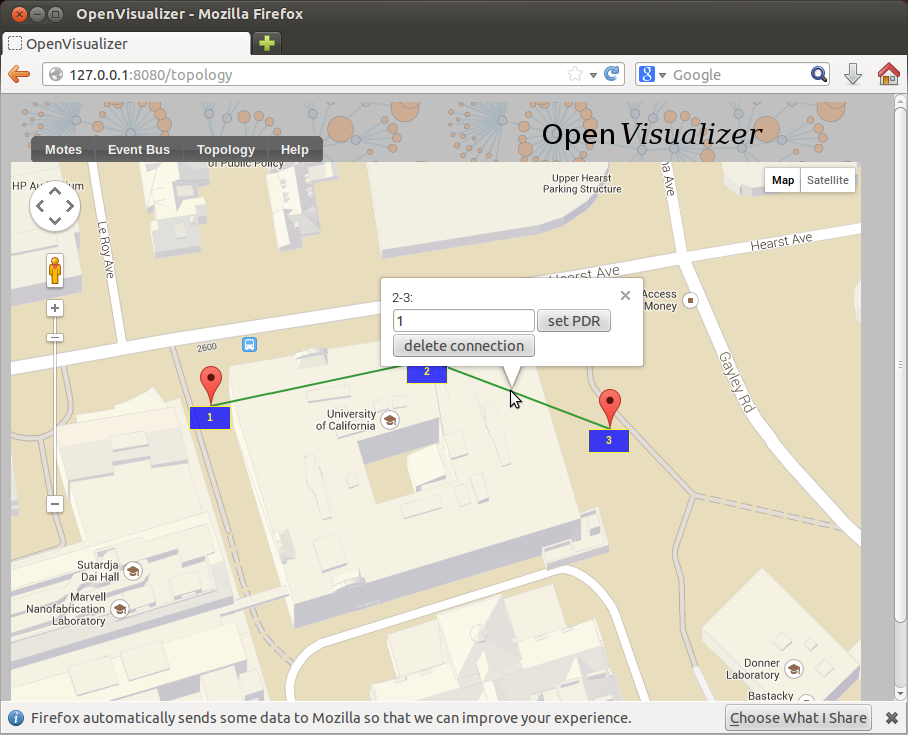
You need to right-click on the red balloon, not the blue rectangle. Confusing, right?

Also, to create a link, you need to **right**-click on both endpoints.



* left-click on a link to change its PDR (packet delivery ratio)

Set up the topology to have a chain 1-2-3, and set each PDR to 1.

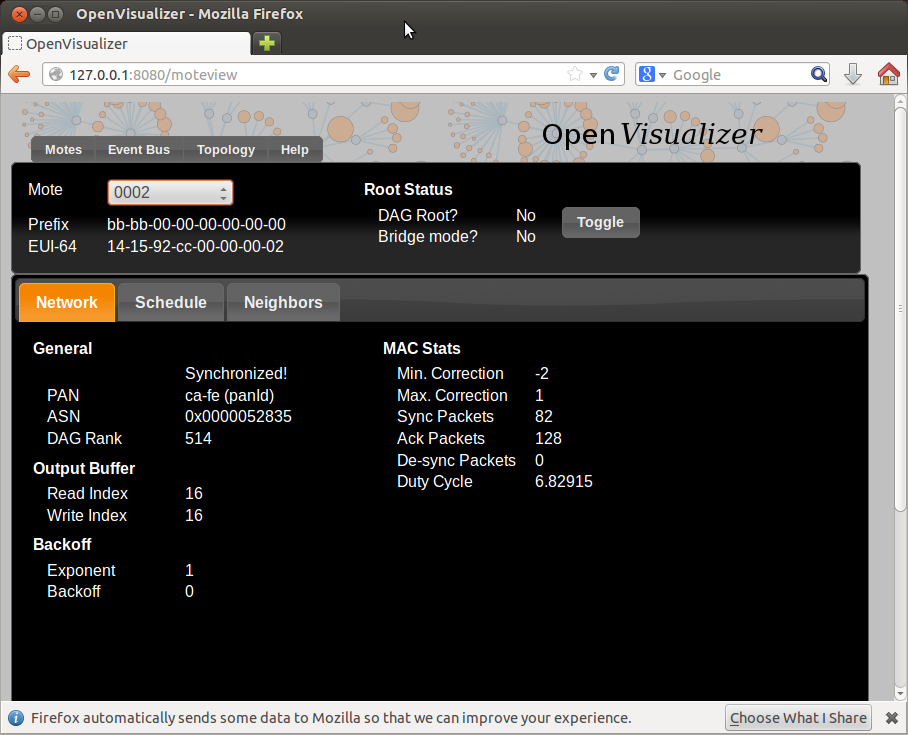


Back in the "Motes" tab, select mote 0001 and click on the "Toggle" button. You just declare mote 1 to be the root of your network (the DAGroot in RPL parlance, the sink in WSN parlance, the gateway, etc). You can select other motes and see that they quickly become "Synchronized!".

Congratulations, you have built your first OpenWSN simulated network!

## Ping a mote

By convention (i.e. this is hard-coded in the software), the IPv6 prefix of the simulated network is bbbb:/64. This means that the IPv6 address of each mote will start with "bbbb::". The remainder of a mote's IPv6 address is it's MAC address (or EUI-64). You can read this in the web interface (under "EUI-64").



Open a new terminal to ping mote 2:

|  |
| --- |
| ~/Desktop/openwsn/openwsn-sw/software/openvisualizer$ ping6 bbbb::1415:92cc:0:2  PING bbbb::1415:92cc:0:2(bbbb::1415:92cc:0:2) 56 data bytes  64 bytes from bbbb::1415:92cc:0:2: icmp\_seq=1 ttl=64 time=51.1 ms  64 bytes from bbbb::1415:92cc:0:2: icmp\_seq=2 ttl=64 time=134 ms  64 bytes from bbbb::1415:92cc:0:2: icmp\_seq=3 ttl=64 time=137 ms  64 bytes from bbbb::1415:92cc:0:2: icmp\_seq=4 ttl=64 time=101 ms  ^C  --- bbbb::1415:92cc:0:2 ping statistics ---  4 packets transmitted, 4 received, 0% packet loss, time 3004ms  rtt min/avg/max/mdev = 51.172/106.146/137.486/34.713 ms |

You can also ping mote 3 which is 3 hops away:

|  |
| --- |
| ~/Desktop/openwsn/openwsn-sw/software/openvisualizer$ ping6 -s 10 bbbb::1415:92cc:0:3  PING bbbb::1415:92cc:0:3(bbbb::1415:92cc:0:3) 10 data bytes  18 bytes from bbbb::1415:92cc:0:3: icmp\_seq=1 ttl=63 time=80.1 ms  18 bytes from bbbb::1415:92cc:0:3: icmp\_seq=2 ttl=63 time=105 ms  18 bytes from bbbb::1415:92cc:0:3: icmp\_seq=3 ttl=63 time=94.0 ms  18 bytes from bbbb::1415:92cc:0:3: icmp\_seq=4 ttl=63 time=147 ms  ^C  --- bbbb::1415:92cc:0:3 ping statistics ---  4 packets transmitted, 4 received, 0% packet loss, time 3001ms  rtt min/avg/max/mdev = 80.115/106.887/147.964/25.359 ms |

We had to reduce size of the ping request for that one (-s 10). The reason is that the packet contains also a source routing header, leaving less bytes for payload.

## Interact over CoAP

[CoAP](https://datatracker.ietf.org/doc/draft-ietf-core-coap/) is a protocol implement on each OpenWSN device, which makes it appear like a web server on the Internet.

By default, an OpenWSN mote implements a CoAP "info" resource which indicate what version of the code is running. You can test that by running the Python test script provided in the firmware:

|  |
| --- |
| ~/Desktop/openwsn/openwsn-fw/openapps/cinfo$ python cinfo.py  /home/thomas/Desktop/openwsn/openwsn-fw/openapps/cinfo  OpenWSN 1.9.0  Python  Python  Python |

## Debugging with Wireshark

In simulation mode, the openvisualizer takes care of simulating the wireless medium. You can use [Wireshark](http://www.wireshark.org/) to take a peek at what goes over this simulated radio space.

**Oops!**

Before you can go on, you need to install [Wireshark](http://www.wireshark.org/):

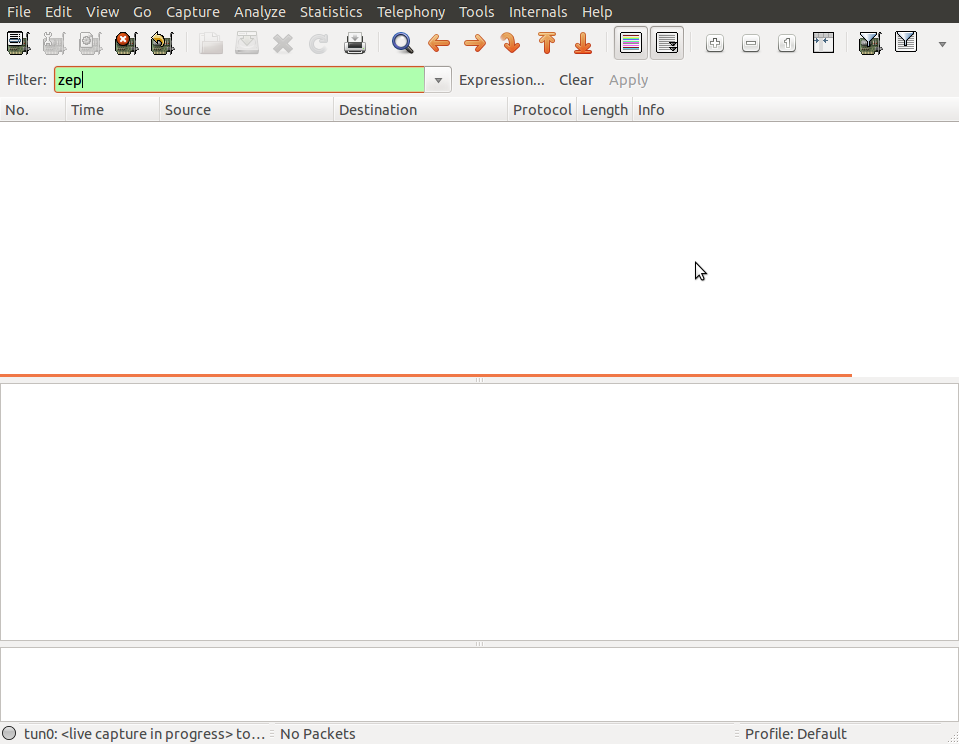
|  |
| --- |
| sudo apt-get install wireshark |

Start Wireshark on the tun interface at address bbbb::1, and configure the filtering for zep (the ZigBee encapsulation protocol).

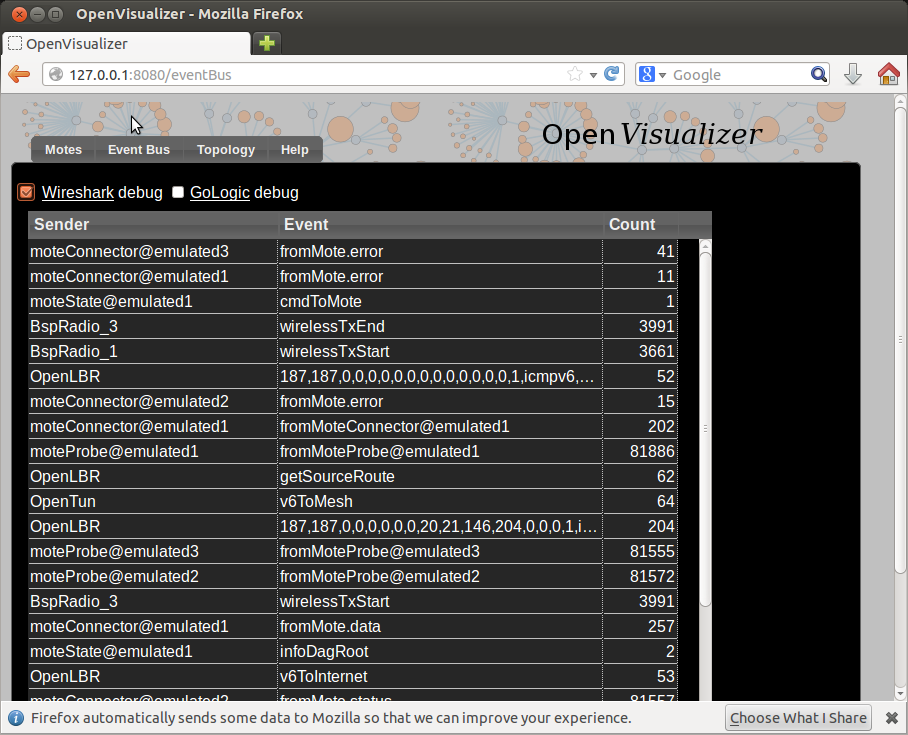
You need to start Wireshark with "sudo":

|  |
| --- |
| sudo wireshark |

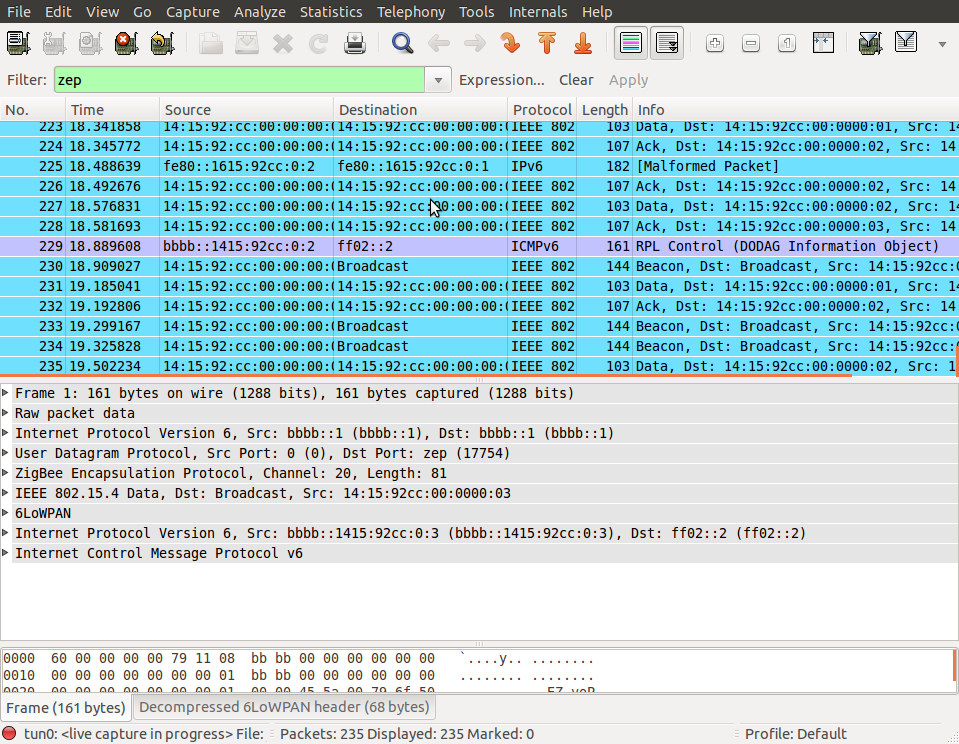
For now, you don't see any packets, that's normal:



In the OpenVisualizer's "Event bus" tab, check the "Wireshark debug" box.



You now see all the packets exchanged over the simulated radio environment, exciting!



## Closing the OpenVisualizer

In the terminal from where you started the OpenVisualizer, type q to close it

## Now with real hardware

User experience with real hardware is identical than when using the simulator. Because it is a popular platform, we will use the TelosB mote, although any other supported hardware platform can be used instead.

## Connecting the boards

When connecting a TelosB board, you Linux will assign it a device file. Reading/writing to/from that file results in bytes being sent to/from your TelosB over its UART interface. To know which device your TelosB is located at, plug it into your computer, wait a second, and type dmesg:

|  |
| --- |
| [ 1181.904524] usb 1-3.3.2: new full-speed USB device number 5 using ehci\_hcd  [ 1182.002641] usb 1-3.3.2: New USB device found, idVendor=0403, idProduct=6001  [ 1182.002651] usb 1-3.3.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3  [ 1182.002659] usb 1-3.3.2: Product: Crossbow Telos Rev.B  [ 1182.002667] usb 1-3.3.2: Manufacturer: XBOW  [ 1182.002674] usb 1-3.3.2: SerialNumber: XBRAHL2O  [ 1182.035737] usbcore: registered new interface driver usbserial  [ 1182.035784] usbcore: registered new interface driver usbserial\_generic  [ 1182.035819] USB Serial support registered for generic  [ 1182.035835] usbserial: USB Serial Driver core  [ 1182.041321] usbcore: registered new interface driver ftdi\_sio  [ 1182.041363] USB Serial support registered for FTDI USB Serial Device  [ 1182.041627] ftdi\_sio 1-3.3.2:1.0: FTDI USB Serial Device converter detected  [ 1182.041785] usb 1-3.3.2: Detected FT232BM  [ 1182.041793] usb 1-3.3.2: Number of endpoints 2  [ 1182.041801] usb 1-3.3.2: Endpoint 1 MaxPacketSize 64  [ 1182.041809] usb 1-3.3.2: Endpoint 2 MaxPacketSize 64  [ 1182.041815] usb 1-3.3.2: Setting MaxPacketSize 64  [ 1182.043704] usb 1-3.3.2: FTDI USB Serial Device converter now attached to ttyUSB0  [ 1182.043742] ftdi\_sio: v1.6.0:USB FTDI Serial Converters Driver |

The penultimate line indicates that this TelosB is attached to /dev/ttyUSB0. You can repeat the same procedure for the other boards. In my case, I have 3 motes connected:

|  |
| --- |
| thomas@Thomas-X61s:~/Desktop/openwsn/openwsn-fw$ ls /dev/ttyU\*  /dev/ttyUSB0  /dev/ttyUSB1  /dev/ttyUSB2 |

## Compiling/Loading firmware

**Wait!**

Before going on, you need to make sure you have a recent version mspgcc:

|  |
| --- |
| thomas@Thomas-X61s:~$ msp430-gcc --version  msp430-gcc (GCC) 4.6.3 20120301 (mspgcc LTS 20120406 unpatched)  Copyright (C) 2011 Free Software Foundation, Inc.  This is free software; see the source for copying conditions.  There is NO  warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. |

In particular, earlier versions Ubuntu (e.g. 12.04 LTS) came with the older msp430-gcc 4.5.3. This will **NOT** work; see [~~[https://openwsn.atlassian.net/images/icons/issuetypes/bug.png](https://openwsn.atlassian.net/browse/FW-209)FW-209~~](https://openwsn.atlassian.net/browse/FW-209) - packing problems in mspgcc Ubuntu **CLOSED** for details.

The build environment allow you to build the firmware and load is on all your boards with a single command (how cool is that?):

|  |
| --- |
| thomas@Thomas-X61s:~/Desktop/openwsn/openwsn-fw$ sudo scons board=telosb toolchain=mspgcc bootload=/dev/ttyUSB2 oos\_openwsn  scons: Reading SConscript files ...   \_\_\_                 \_ \_ \_  \_\_\_  \_ \_  | . | \_\_\_  \_\_\_ .\_ \_ | | | |/ \_\_>| \ |  | | || . \/ .\_>| ' || | | |\\_\_ \|   |  `\_\_\_'|  \_/\\_\_\_.|\_|\_||\_\_/\_/ <\_\_\_/|\_\\_|       |\_|                  openwsn.org  scons: done reading SConscript files.  scons: Building targets ...  msp430-size firmware/openos/projects/common/03oos\_openwsn\_prog     text    data     bss     dec     hex filename    41024       0    4138   45162    b06a firmware/openos/projects/common/03oos\_openwsn\_prog  telosb\_bootload(["firmware/openos/projects/common/03oos\_openwsn\_prog.phonyupload"], ["firmware/openos/projects/common/03oos\_openwsn\_prog.ihex"])  starting bootloading on /dev/ttyUSB2  MSP430 Bootstrap Loader Version: 1.39-telos-8  Mass Erase...  Transmit default password ...  Invoking BSL...  Transmit default password ...  Current bootstrap loader version: 1.61 (Device ID: f16c)  Changing baudrate to 38400 ...  Program ...  41024 bytes programmed.  Reset device ...  done bootloading on /dev/ttyUSB2  scons: done building targets. |

**Programming multiple motes?**

While the command allows you to program multiple motes in one go:

|  |
| --- |
| sudo scons board=telosb toolchain=mspgcc bootload=/dev/ttyUSB0,/dev/ttyUSB1,/dev/ttyUSB2 oos\_openwsn |

Linux seems to have synchronization errors when doing so. See [[https://openwsn.atlassian.net/images/icons/issuetypes/bug.png](https://openwsn.atlassian.net/browse/SW-136)SW-136](https://openwsn.atlassian.net/browse/SW-136) - programming multiple TelosB motes concurrently in Ubuntu fails **OPEN**.

**New Compiling**:

scons board=python toolchain=gcc oos\_openwsn -c

scons board=python toolchain=gcc oos\_openwsn

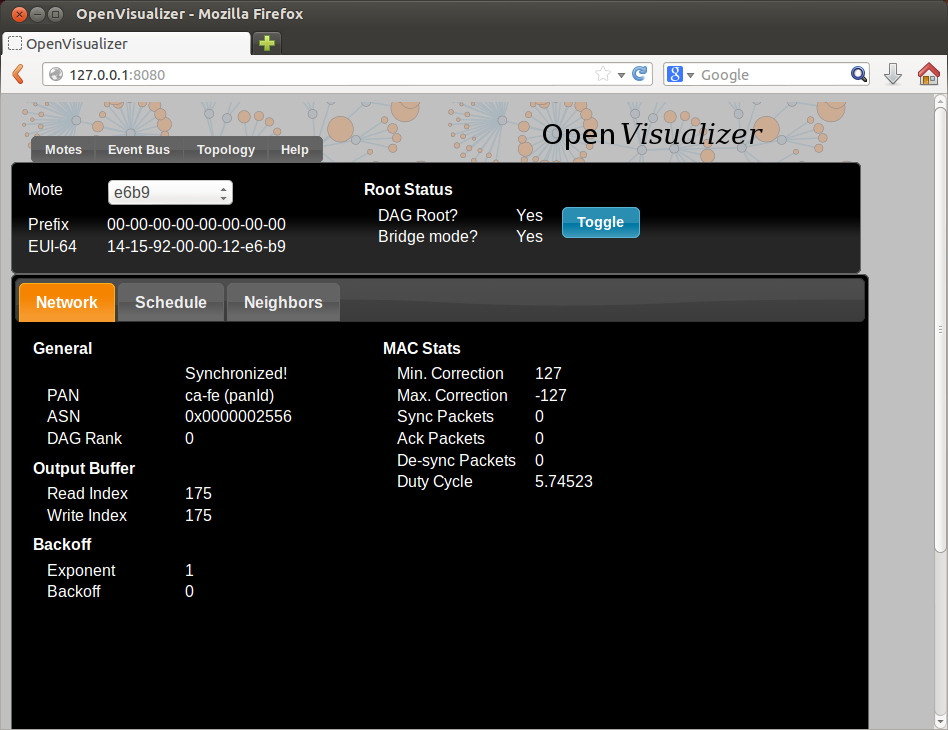
## Start a network

Once the motes are programmed, you have the exact same experience as when running a simulated network.

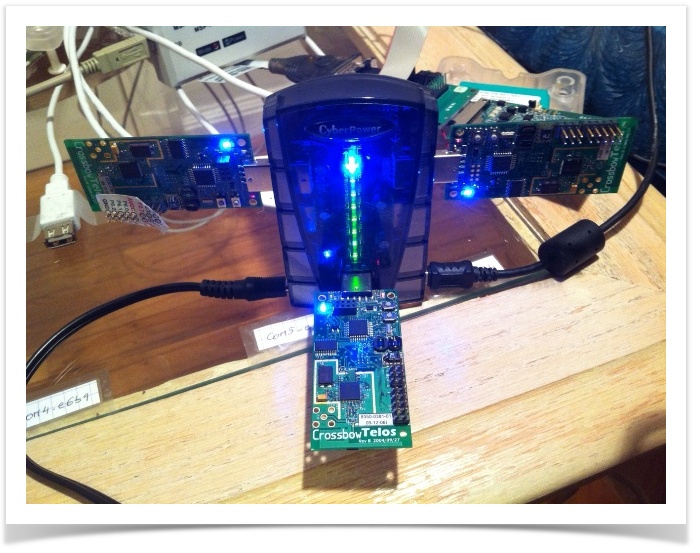
First, start the OpenVisualizer:

|  |
| --- |
| thomas@Thomas-X61s:~/Desktop/openwsn/openwsn-sw/software/openvisualizer$ sudo scons runweb  scons: Reading SConscript files ...   \_\_\_                 \_ \_ \_  \_\_\_  \_ \_  | . | \_\_\_  \_\_\_ .\_ \_ | | | |/ \_\_>| \ |  | | || . \/ .\_>| ' || | | |\\_\_ \|bb   |  `\_\_\_'|  \_/\\_\_\_.|\_|\_||\_\_/\_/ <\_\_\_/|\_\\_|       |\_|                  openwsn.org  scons: done reading SConscript files.  scons: Building targets ...  Delete("build/runui/web\_files")  Mkdir("/home/thomas/Desktop/openwsn/openwsn-sw/software/openvisualizer/build/runui")  Copy("build/runui/web\_files", "bin/openVisualizerApp/web\_files")  Delete("build/runui/sim\_files")  Mkdir("/home/thomas/Desktop/openwsn/openwsn-sw/software/openvisualizer/build/runui")  Copy("build/runui/sim\_files", "bin/openVisualizerApp/sim\_files")  uiRunner(["bin/openVisualizerApp/openVisualizerWeb"], ["bin/openVisualizerApp/openVisualizerWeb.py"])  Child PID is 2820  scons: done building targets.  thomas@Thomas-X61s:~/Desktop/openwsn/openwsn-sw/software/openvisualizer$ ioctl(TUNSETIFF): Device or resource busy  created following virtual interface:  4: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UNKNOWN qlen 500      link/none      inet6 bbbb::1/64 scope global         valid\_lft forever preferred\_lft forever      inet6 fe80::1/64 scope link         valid\_lft forever preferred\_lft forever |

Open <http://127.0.0.1:8080/> to see the web interface:



Your TelosB motes turn on their blue LED once synchronized.



**Can I see the topology?**

You might be tempted to look for a graphical representation of the topology. Unfortunately, this is only implemented in simulation mode. Really want it? Help fix [[https://openwsn.atlassian.net/images/icons/issuetypes/task.png](https://openwsn.atlassian.net/browse/SW-137)SW-137](https://openwsn.atlassian.net/browse/SW-137) - display network topology in non-simulation mode **OPEN** .

## Ping a mote

You can ping a mote exactly as you would in the simulator:

|  |
| --- |
| thomas@Thomas-X61s:~$ ping6 bbbb::1415:9200:12:e63b  PING bbbb::1415:9200:12:e63b(bbbb::1415:9200:12:e63b) 56 data bytes:13  64 bytes from bbbb::1415:9200:12:e63b: icmp\_seq=1 ttl=64 time=365 ms  64 bytes from bbbb::1415:9200:12:e63b: icmp\_seq=2 ttl=64 time=354 ms  64 bytes from bbbb::1415:9200:12:e63b: icmp\_seq=3 ttl=64 time=344 ms  64 bytes from bbbb::1415:9200:12:e63b: icmp\_seq=4 ttl=64 time=340 ms  64 bytes from bbbb::1415:9200:12:e63b: icmp\_seq=5 ttl=64 time=332 ms  ^C  --- bbbb::1415:9200:12:e63b ping statistics ---  5 packets transmitted, 5 received, 0% packet loss, time 4006ms  rtt min/avg/max/mdev = 332.908/347.599/365.737/11.417 ms |

## Interaction over CoAP

You can interact with a mote exactly as you would in the simulator. After modifying the cinfo.py script to communicate with mote bbbb::1415:9200:12:e63b:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| thomas@Thomas-X61s:~/Desktop/openwsn/openwsn-fw/openapps/cinfo$ python cinfo.py  /home/thomas/Desktop/openwsn/openwsn-fw/openapps/cinfo  OpenWSN 1.9  TelosB  MSP430f1611  CC2420  Done. Press enter to close.  CoAP Interaction DEMO  This demo shows how easy it is to write an application to interact with an OpenWSN mote, from your computer, over CoAP. We will use the following:   * *- the firmware from the*[*https://github.com/openwsn-berkeley/openwsn-fw*](https://github.com/openwsn-berkeley/openwsn-fw)*repository, which we will emulate in the simulator (see [OpenSim](https://openwsn.atlassian.net/wiki/display/OW/OpenSim))* * *- the software from*[*https://github.com/openwsn-berkeley/openwsn-sw*](https://github.com/openwsn-berkeley/openwsn-sw)*repository* * *- the Python CoAP library from the*[*https://github.com/openwsn-berkeley/coap*](https://github.com/openwsn-berkeley/coap)*repository*  Setup We start by cloning the 3 repositories side-by-side.  https://openwsn.atlassian.net/wiki/download/attachments/28835868/clone.png?version=1&modificationDate=1395634222200&api=v2  The next step is to build the firmware as a Python extension module so it can be simulated:   |  | | --- | | C:\Users\Thomas\Desktop\openwsn-fw>scons board=python toolchain=gcc oos\_openwsn  scons: Reading SConscript files ...   \_\_\_                 \_ \_ \_  \_\_\_  \_ \_  | . | \_\_\_  \_\_\_ .\_ \_ | | | |/ \_\_>| \ |  | | || . \/ .\_>| ' || | | |\\_\_ \|   |  `\_\_\_'|  \_/\\_\_\_.|\_|\_||\_\_/\_/ <\_\_\_/|\_\\_|       |\_|                  openwsn.org    [...]  scons: done building targets. |   We can now start the OpenVisualizer application in simulation mode, with a web interface:   |  | | --- | | C:\Users\Thomas\Desktop\openwsn-sw\software\openvisualizer>scons runweb --sim  --simCount=2  scons: Reading SConscript files ...   \_\_\_                 \_ \_ \_  \_\_\_  \_ \_  | . | \_\_\_  \_\_\_ .\_ \_ | | | |/ \_\_>| \ |  | | || . \/ .\_>| ' || | | |\\_\_ \|   |  `\_\_\_'|  \_/\\_\_\_.|\_|\_||\_\_/\_/ <\_\_\_/|\_\\_|       |\_|                  openwsn.org  scons: done reading SConscript files.  scons: Building targets ...  Copy("bin\openVisualizerApp\sim\_files", "..\..\..\openwsn-fw\firmware\openos\bsp\boards\python\openwsnmodule\_obj.h")  Mkdir("bin\openVisualizerApp\sim\_files\windows")  Copy("bin\openVisualizerApp\sim\_files\windows\oos\_openwsn-x86.pyd", "..\..\..\openwsn-fw\firmware\openos\projects\common  \oos\_openwsn.pyd")  Copy("bin\openVisualizerApp\sim\_files", "..\..\..\openwsn-fw\firmware\openos\projects\common\oos\_openwsn.pyd")  Delete("build\runui\web\_files")  Mkdir("C:\Users\Thomas\Desktop\openwsn-sw\software\openvisualizer\build\runui")  Copy("build\runui\web\_files", "bin\openVisualizerApp\web\_files")  Delete("build\runui\sim\_files")  Mkdir("C:\Users\Thomas\Desktop\openwsn-sw\software\openvisualizer\build\runui")  Copy("build\runui\sim\_files", "bin\openVisualizerApp\sim\_files")  uiRunner(["bin\openVisualizerApp\openVisualizerWeb"], ["bin\openVisualizerApp\openVisualizerWeb.py"])  Child PID is 5460  scons: done building targets. |   Open <http://127.0.0.1:8080/> shows the web interface of the OpenVisualizer. Edit the topology Start by editing the topology so the two nodes are connected through a perfect wireless link (PDR=1).  https://openwsn.atlassian.net/wiki/download/attachments/28835868/topology.png?version=1&modificationDate=1395634270870&api=v2 Start the network You can now click the "Toggle" button so mote 1 becomes the DAGroot of the network. Mote 2 will synchronize to it, and RPL will set up the routing structure.  https://openwsn.atlassian.net/wiki/download/attachments/28835868/sync.png?version=1&modificationDate=1395634303400&api=v2 Ping a mote To verify connectivity, you can ping mote 2  This page was put together using a Windows computer. If you're using Linux, everything is exactly the same, but you need to use command ping6 rather than ping.   |  | | --- | | C:\Users\Thomas>ping bbbb::1415:92cc:0:2  Pinging bbbb::1415:92cc:0:2 with 32 bytes of data:  Reply from bbbb::1415:92cc:0:2: time=50ms  Reply from bbbb::1415:92cc:0:2: time=67ms  Reply from bbbb::1415:92cc:0:2: time=65ms  Reply from bbbb::1415:92cc:0:2: time=59ms  Ping statistics for bbbb::1415:92cc:0:2:      Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),  Approximate round trip times in milli-seconds:      Minimum = 50ms, Maximum = 67ms, Average = 60ms |  CoAP discovery By default, mote 2 is a CoAP endpoint. To see the resources it has, you can install the Copper plugin for Firefox and navigate to the coap://[bbbb::1415:92cc:0:2]:5683/.well-known/core URI.  https://openwsn.atlassian.net/wiki/download/attachments/28835868/coap_wellknown.png?version=1&modificationDate=1395634383899&api=v2  Alternatively, you can create a Python script which uses the Python CoAP module you just clones:   |  | | --- | | from coap import coap  c = coap.coap()  p = c.GET('coap://bbbb::1415:92cc:0:2/.well-known/core')  print ''.join([chr(b) for b in p]) |   Running is prints the same information   |  | | --- | | </6t>,</.well-known/core>,</l>,</i> |  Getting information about the board Doing a CoAP GET on /i resource gives you information about the board itself, including the firmware version, the micro-controller and radio used. Since the mote is emulation, it returns "Python" for latter.  https://openwsn.atlassian.net/wiki/download/attachments/28835868/coap_i.png?version=1&modificationDate=1395634437664&api=v2 Interact with an LED You can also interact with the debug LED of the mote. This is done through the /l CoAP resource:   * a GET indicates what state it is in * a PUT allows you to set (1), clear (0) or toggle it (2)   The following script toggles the LED and verifies that its value has indeed changed:   |  | | --- | | from coap import coap  c = coap.coap()  p = c.GET('coap://[bbbb::1415:92cc:0:2]/l')  print chr(p[0])  c.PUT(      'coap://[bbbb::1415:92cc:0:2]/l',      payload = [ord('2')],  )  p = c.GET('coap://[bbbb::1415:92cc:0:2]/l')  print chr(p[0]) |   Running it shows that the LED was first OFF, then ON.   |  | | --- | | 0  1 | |

1. **Component Pictures**
2. **OpenStack**



1. **IE stack**

**typedef struct {**

//admin

**uint8\_t creator;** // the component which called getFreePacketBuffer()

**uint8\_t owner;** // the component which currently owns the entry

**uint8\_t\* payload;** // pointer to the start of the payload within 'packet'

**uint8\_t length;** // length in bytes of the payload

//l4

**uint8\_t l4\_protocol;** // l4 protocol to be used

**bool l4\_protocol\_compressed;** // is the l4 protocol header compressed?

**uint16\_t l4\_sourcePortORicmpv6Type;** // l4 source port

**uint16\_t l4\_destination\_port;** // l4 destination port

**uint8\_t\* l4\_payload;** // pointer to the start of the payload of l4 (used for retransmits)

**uint8\_t l4\_length;** // length of the payload of l4 (used for retransmits)

//l3

**open\_addr\_t l3\_destinationAdd;** // 128b IPv6 destination (down stack)

**open\_addr\_t l3\_sourceAdd;** // 128b IPv6 source address

//l2

**owerror\_t l2\_sendDoneError;** // outcome of trying to send this packet

**open\_addr\_t l2\_nextORpreviousHop;** // 64b IEEE802.15.4 next (down stack) or previous (up) hop address

**uint8\_t l2\_frameType;** // beacon, data, ack, cmd

**uint8\_t l2\_dsn;** // sequence number of the received frame

**uint8\_t l2\_retriesLeft;** // number Tx retries left before packet dropped (dropped when hits 0)

**uint8\_t l2\_numTxAttempts;** // number Tx attempts

**asn\_t l2\_asn;** // at what ASN the packet was Tx'ed or Rx'ed

**uint8\_t\* l2\_payload;** // pointer to the start of the payload of l2 (used for MAC to fill in ASN in ADV)

**uint8\_t\* l2\_scheduleIE\_cellObjects;** // pointer to the start of cell Objects in scheduleIE

**uint8\_t l2\_scheduleIE\_numOfCells;** // number of cells were going to be scheduled or removed.

**uint8\_t l2\_scheduleIE\_frameID;** // frameID in scheduleIE

**uint8\_t\* l2\_ASNpayload;** // pointer to the ASN in EB

**uint8\_t l2\_joinPriority;** // the join priority received in EB

**bool l2\_IEListPresent;** //did have IE field?

**bool l2\_payloadIEpresent;** // did I have payload IE field

**bool l2\_joinPriorityPresent;**

**int16\_t l2\_timeCorrection;** // record the timeCorrection and print out at endOfslot

//layer-2 security

**uint8\_t l2\_securityLevel;** //the security level specified for the current frame

**uint8\_t l2\_keyIdMode;** //the key Identifier mode specified for the current frame

**uint8\_t l2\_keyIndex;** //the key Index specified for the current frame

**open\_addr\_t l2\_keySource;** //the key Source specified for the current frame

**uint8\_t l2\_authenticationLength;** //the length of the authentication field

**uint8\_t commandFrameIdentifier;** //used in case of Command Frames

**uint8\_t\* l2\_FrameCounter;** //pointer to the FrameCounter in the MAC header

//l1 (drivers)

**uint8\_t l1\_txPower;** // power for packet to Tx at

**int8\_t l1\_rssi;** // RSSI of received packet

**uint8\_t l1\_lqi;** // LQI of received packet

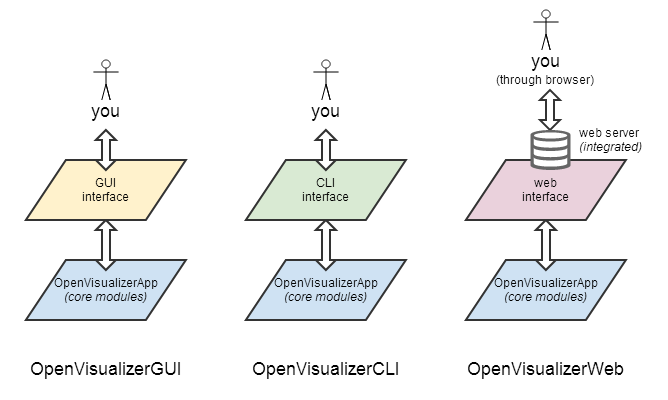
**bool l1\_crc;** // did received packet pass CRC check?

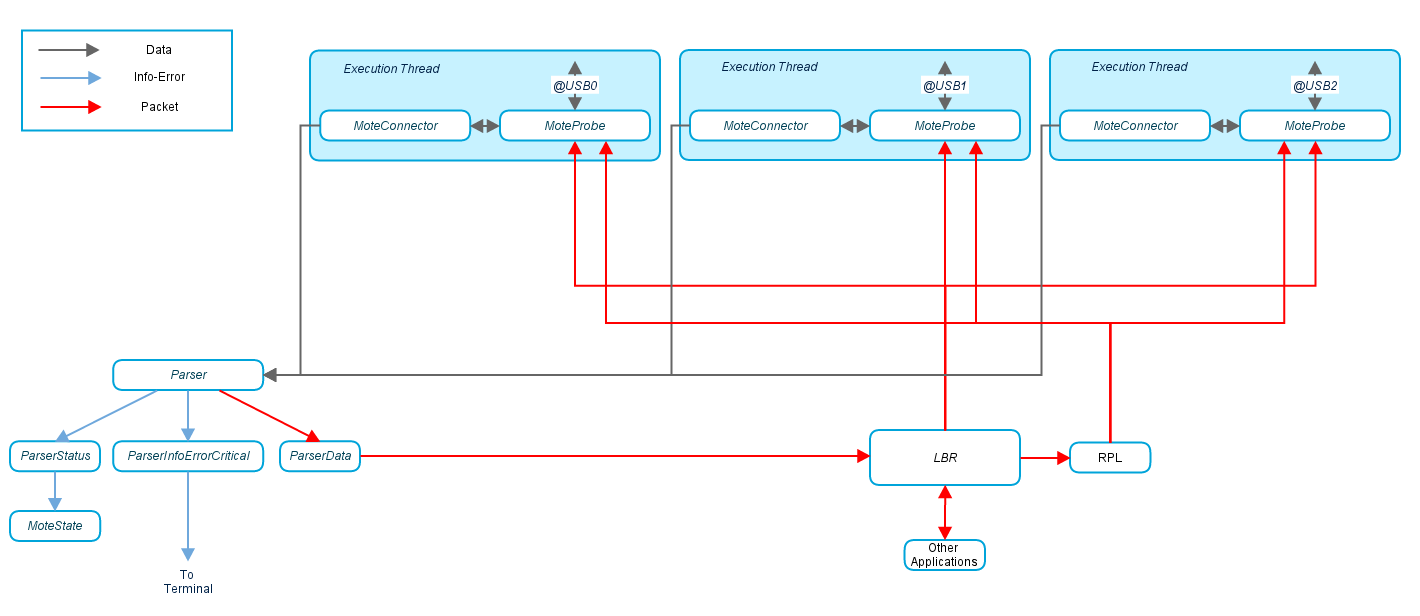
//the packet

**uint8\_t packet[1+1+125+2+1];** // 1B spi address, 1B length, 125B data, 2B CRC, 1B LQI

**} OpenQueueEntry\_t;**

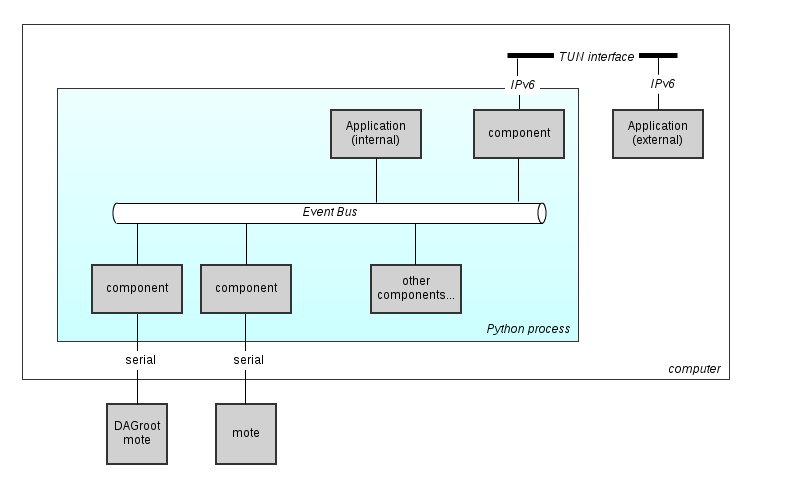
1. **OpenVisualizer Code Organization**

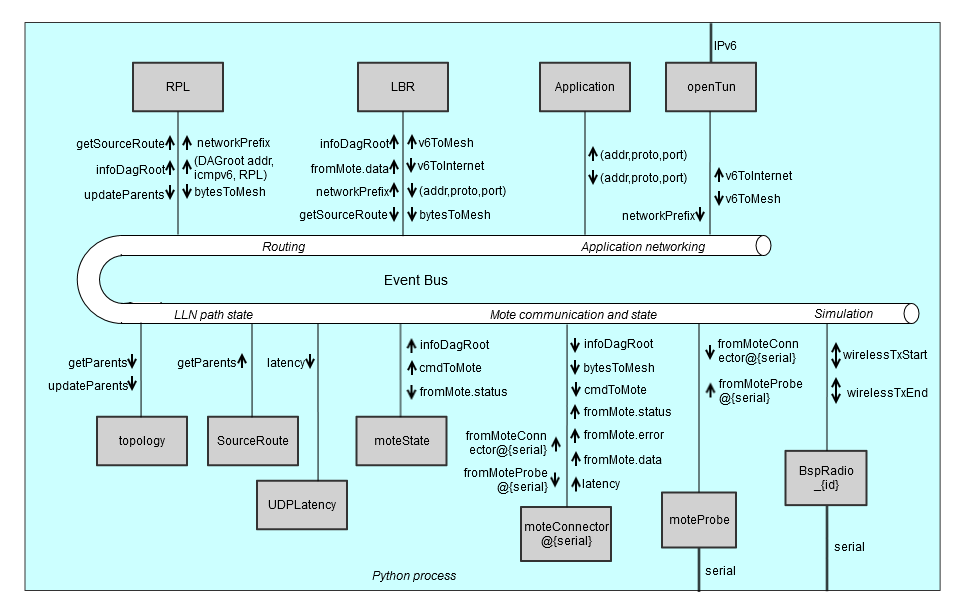
****

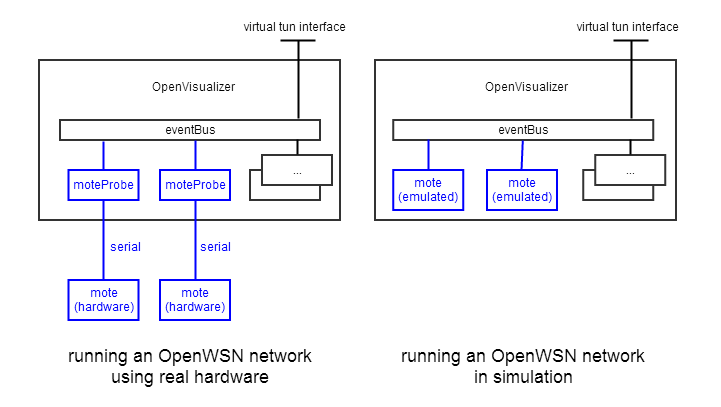
****

Modules

* [openVisualizerGui.py](https://github.com/openwsn-berkeley/openwsn-sw/blob/develop/software/openvisualizer/bin/openVisualizerApp/openVisualizerGui.py) is the master module. It can be run against real or simulated motes (see [OpenSim](https://openwsn.atlassian.net/wiki/display/OW/OpenSim)). It lists all the connected motes and, for each, spawns a different execution thread.
* [moteProbe.py](https://github.com/openwsn-berkeley/openwsn-sw/blob/develop/software/openvisualizer/moteProbe/moteProbe.py) contains the code which is executed by each thread. Whenever data is received from the mote, it is passed along to [Parser.py](https://github.com/openwsn-berkeley/openwsn-sw/blob/develop/software/openvisualizer/moteConnector/Parser.py).
* [Parser.py](https://github.com/openwsn-berkeley/openwsn-sw/blob/develop/software/openvisualizer/moteConnector/Parser.py) dispatches all the data it receives to respective parser module. In case a frame of Status/Error/ErrorCritical is received from the serial port, [ParserInfoErrorCritical.py calls the corresponding display\* function from](https://github.com/openwsn-berkeley/openwsn-sw/blob/develop/software/openvisualizer/moteConnector/ParserInfoErrorCritical.py) [moteState.py.](https://github.com/openwsn-berkeley/openwsn-sw/blob/develop/software/openvisualizer/moteState/moteState.py)
* [openVisualizerGui.py](https://github.com/openwsn-berkeley/openwsn-sw/blob/develop/software/openvisualizer/bin/openVisualizerApp/openVisualizerGui.py) handles everything which is related to the graphical user interface. It is solely based on TkInter, the graphical interface which ships by default with Python. A single semaphore is used to arbitrate the access to the graphic elements.
* [StackDefines.py](https://github.com/openwsn-berkeley/openwsn-sw/blob/develop/software/openvisualizer/moteConnector/StackDefines.py) is used by [ParserInfoErrorCritical.py](https://github.com/openwsn-berkeley/openwsn-sw/blob/develop/software/openvisualizer/moteConnector/ParserInfoErrorCritical.py) to translate error and status code into human-readable text.







1. **Sensor Cloud Testbed**



Figure 1. Testbed Architecture

1. **Network setup**

Network is configured as shown in Fig.1.

1. **OpenStack setup** (If you don’t know details how to install, you can ask Minsik or Young Ki brother)
   1. **OpenStack model**

All in one (AIO)

* 1. **Hardware requirement**

Tiny computer with 16G Ram and 500 GB HDD

* 1. **Software requirement**
* Ubuntu server 14.04.3 x64. Other versions may cause errors. You can create a BOOT USB for OS installation by using YUMI-2.0.2.6 app.
* OpenStack Liberty script from DCN git server (<http://114.71.50.187/openstack_script/openstack_script/tree/master/liberty>)
  1. **How to install**

Installation must be working in the root account!!

**Step 1. Script download**

cd ~/

git clone http://114.71.50.187/openstack\_script/openstack\_script.git cloud

cd cloud/liberty

**Step 2. Configuration files**

1) Create ssh-keygen (Very important!!)

ssh-keygen

All questions enter

2) Configuration hosts

cd ~/cloud/liberty

vi configure.cfg

AIO\_HOST=10.0.2.10 # all-in-one node ip

NET\_HOST=10.0.2.11 # controller+network node ip(if multi node)

COM\_HOST=10.0.2.12 # compute node ip(if multi node)

3) Configuration Target server passwd

vi chpasswd.sh

...

USER\_NAME=ubuntu # target normal-user name

USER\_PASS=ubuntu # target normal-user passwd

CHANGED\_PASS=1234 # changed target normal and root passwd

...

vi chpasswd\_shell.sh

...

CURR\_PASS=ubuntu # target user before change passwd

CH\_PASS=1234 # target user change passwd

...

4) Configuration target information

Below is a case of when the All-in-One.

In the case of Multi-Node it must be modified with a net-config.cfg and com-config.cfg.

vi aio-config.cfg

NET\_LIST="eth0 eth1" # if eth0(external), eth1(data)

BR\_LIST="br-eth0 br-eth1 br-tacker" # added "br-tracker" when using tracker

VLAN\_BR\_LIST="br-eth1" # only data bridge for VLAN

VLAN\_START=100

BR\_MAPPING\_LIST="br-eth0 br-eth1 br-tacker" # bridge mapping list

BR\_MODE="static static" # set dhcp or static, manual to bridge br-eth0 and br-eth1

# set ip to br-eth0, br-eth1 (If the zero is ignored)

# if using tacker : "192.168.120.1/24"

BR\_IP\_LIST="10.0.2.10/24 192.168.56.100/24 192.168.120.1/24"

BR\_GW\_LIST="10.0.2.1 0 0" # set gateway (only external bridge)

BR\_DNS\_LIST="8.8.8.8 0 0" # set dns (only external bridge)

MGMT\_IP='192.168.56.100' # set management ip

LOCAL\_IP='192.168.56.100' # set local ip for GRE or VxLAN

CINDER\_VOLUME=sdb # set block storage

HOSTNAME='controller' # set node name

DEFAULT\_PASS='1234' # set openstack passwd

REMOVE\_PACKAGE='0' # Remove Option

TACKER\_VERSION= # tacker version(empty is master, default "-b stable/liberty")

**Step 3. Install**

./configure.sh

select "aio" or "net and com" or "mul"

Good Luck!

* 1. **Install VM OS**
* Create 3 VM with NIC attached directly with external network (don’t use floating IP). Floating IP may cause errors such as disabling ssh or git clone https (may due to security group policy in OpenStack-TODO)
* Use image of Ubuntu server version 14.04 or *yakkety-server-cloudimg-i386.img* (Ubuntu server version 16.10).
* 3 VMs includes rmBR (remote Border Router), Web server (Grafana), Database (InfluxDB)
* How to install InfluxDB: <https://docs.influxdata.com/influxdb/v0.9/introduction/installation/>
  + Create **6tisch** database in InfluxDB
* How to install rmBR mentioned in **section 3**
* How to install Grafana: <http://docs.grafana.org/installation/debian/>
* In Grafana UI, query as following

*SELECT max("value") FROM "6tisch" WHERE "mote" = '****Mote IP'*** *AND $timeFilter GROUP BY time($interval) fill(null)*  to display sensor data to web UI

For Example:

*SELECT max("value") FROM "6tisch" WHERE "mote" = 'bbbb::1415:9200:13cb:460' AND $timeFilter GROUP BY time($interval) fill(null)*  to display sensor data to web UI

* 1. **Install Intel NUC OS**
* Use Ubuntu Server or Desktop 14.04
  1. **Enable Distributed Scheduling SF0**

Open openwsn-fw/openstack/02b-MAChigh/sf0.c

Edit sf0\_vars.numAppPacketsPerSlotFrame = 2;

Build and load edited openwsn-fw to sensors as mentioned above.

1. **Control Setup** (Remote control your mote through NUC)

**Step 1. Prerequisites**

To try this feature, you need one computer, one or more NUC and some motes.

Before trying this page, please make sure all softwares (especially the Python packages specified in *requirements.txt* under openwsn-sw root directory) are installed on your computer and NUC. To do this, you can refer to [Kickstart Linux.](https://openwsn.atlassian.net/wiki/display/OW/Kickstart+Linux)

Note that, this project requires new python packages to run, which is added in requirements.txt. To make sure everything is setup, you need:

|  |
| --- |
| ~/Desktop/openwsn/openwsn-sw$ git pull |

to update the requirements.txt file and

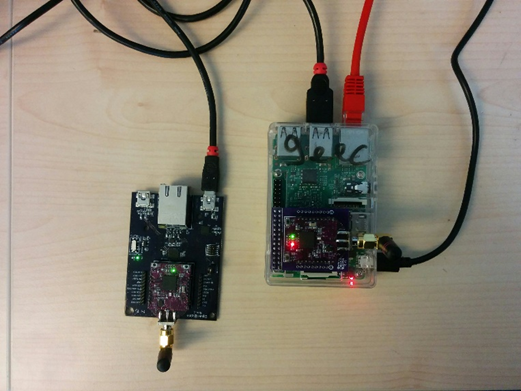
|  |
| --- |
| ~/Desktop/openwsn/openwsn-sw/software/openvisualizer$ sudo pip install -r requirements.txt |

to install all necessary packages.

Also we assume the motes already have the openwsn firmware code loaded.

**Step 2. Setup**

1. Connect your NUC to the network (preferably using ethernet, wifi works too but may cause interferences with your motes radio signals). Make sure you remember its IP address (v4/v6).  
   To setup the communication between the NUC and your rmBR you will need to have port 5683 opened on the NUC side and port 50000 opened on your computer and on the NUC. Make sure that no firewall is blocking them.
2. Connect your motes to the NUC (through USB or GPIO pins)



**Step 3. Play**

1. On NUC, open a terminal and type the following command

|  |
| --- |
| > cd openwsn-sw/software/openvisualizer  > sudo scons runrover |

1. On your rmBR, open a terminal and type the following command

> cd openwsn-sw/software/openvisualizer

> scons runweb --rover

1. On your rmBR, create sensor.py file with code :

**import socket**

**import struct**

**import argparse**

**import time**

**from influxdb import InfluxDBClient**

**from datetime import datetime**

**def main(host='192.168.10.163', port=8086):**

**################Collect Sensor Data#######################**

**socket\_handler = socket.socket(socket.AF\_INET6,socket.SOCK\_DGRAM)**

**socket\_handler.bind(('',2000))**

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

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

**user = 'root'**

**password = 'root'**

**dbname = 'openwsn'**

**dbuser = 'root'**

**dbuser\_password = 'root'**

**client = InfluxDBClient(host, port, user, password, dbname)**

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

**while True:**

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

**# wait for a request**

**request,dist\_addr = socket\_handler.recvfrom(1024)**

**hisAddress = dist\_addr[0]**

**hisPort = dist\_addr[1]**

**asn = struct.unpack('<HHB',request[:5])**

**counter = struct.unpack('<h',request[-2:])[0]**

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

**current\_time = datetime.utcnow().strftime('%Y-%m-%dT%H:%M:%SZ')**

**json\_body = [**

**{**

**"measurement": "6tisch",**

**"tags": {**

**"mote": hisAddress,**

**},**

**"time": current\_time,**

**"fields": {**

**"value": float(counter)**

**}**

**}**

**]**

**client.write\_points(json\_body)**

**if \_\_name\_\_ == '\_\_main\_\_':**

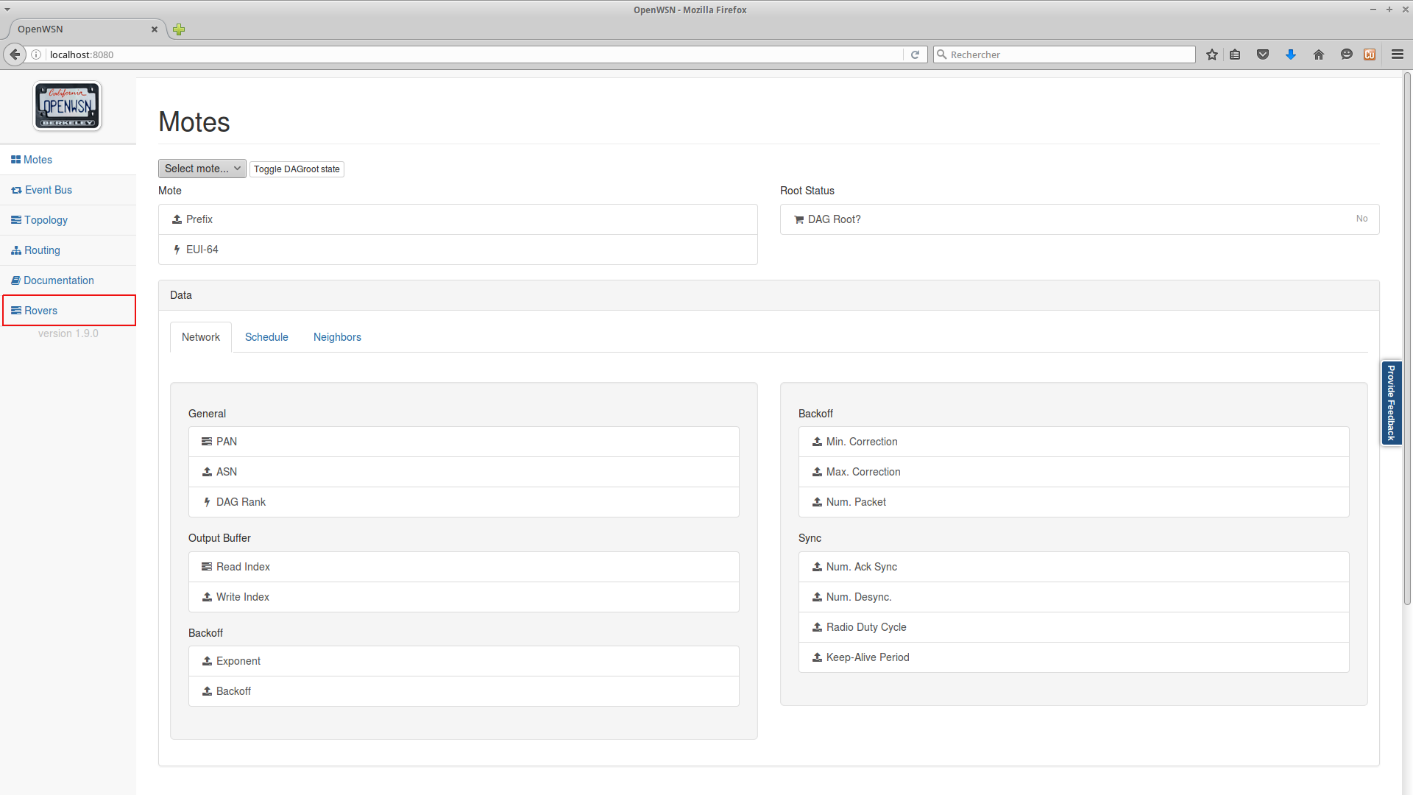
**main()**

Open a terminal and type the following command

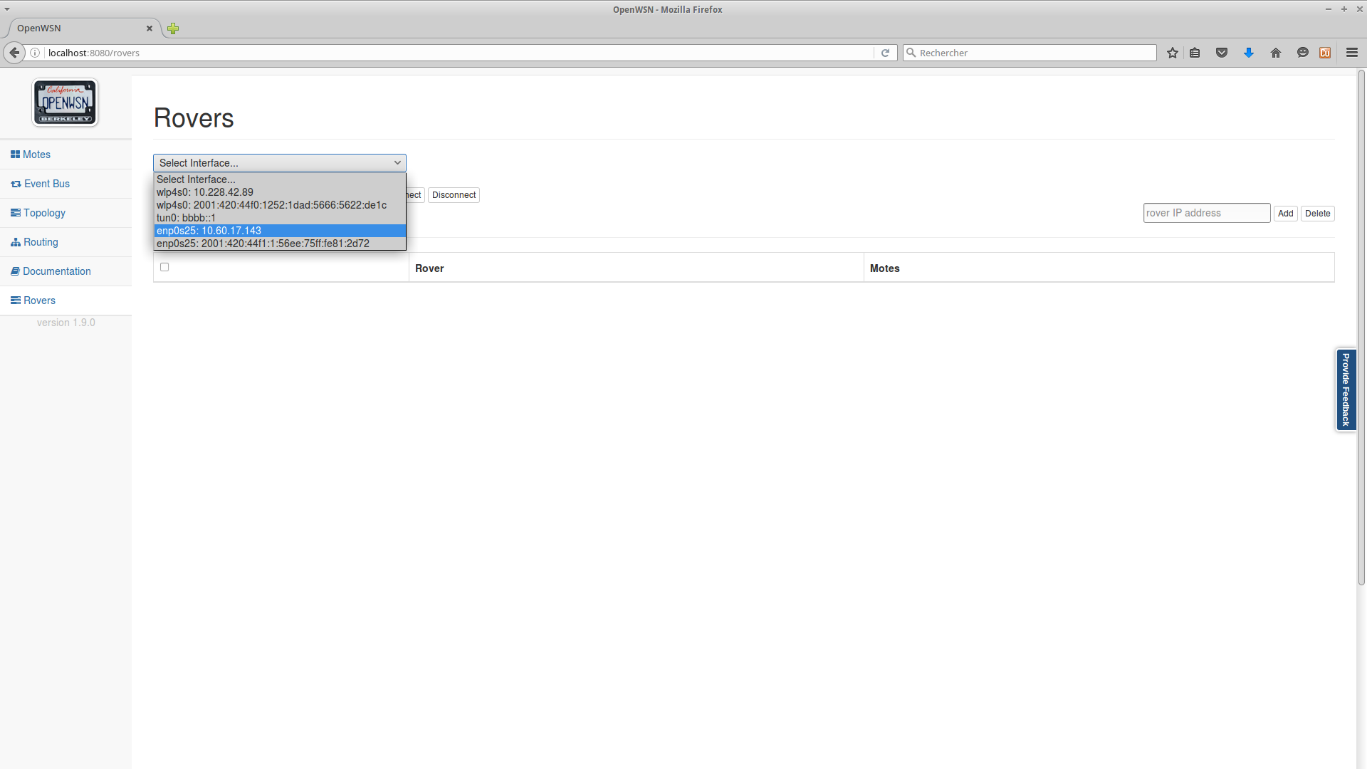
> python sensor.py

|  |
| --- |
|  |

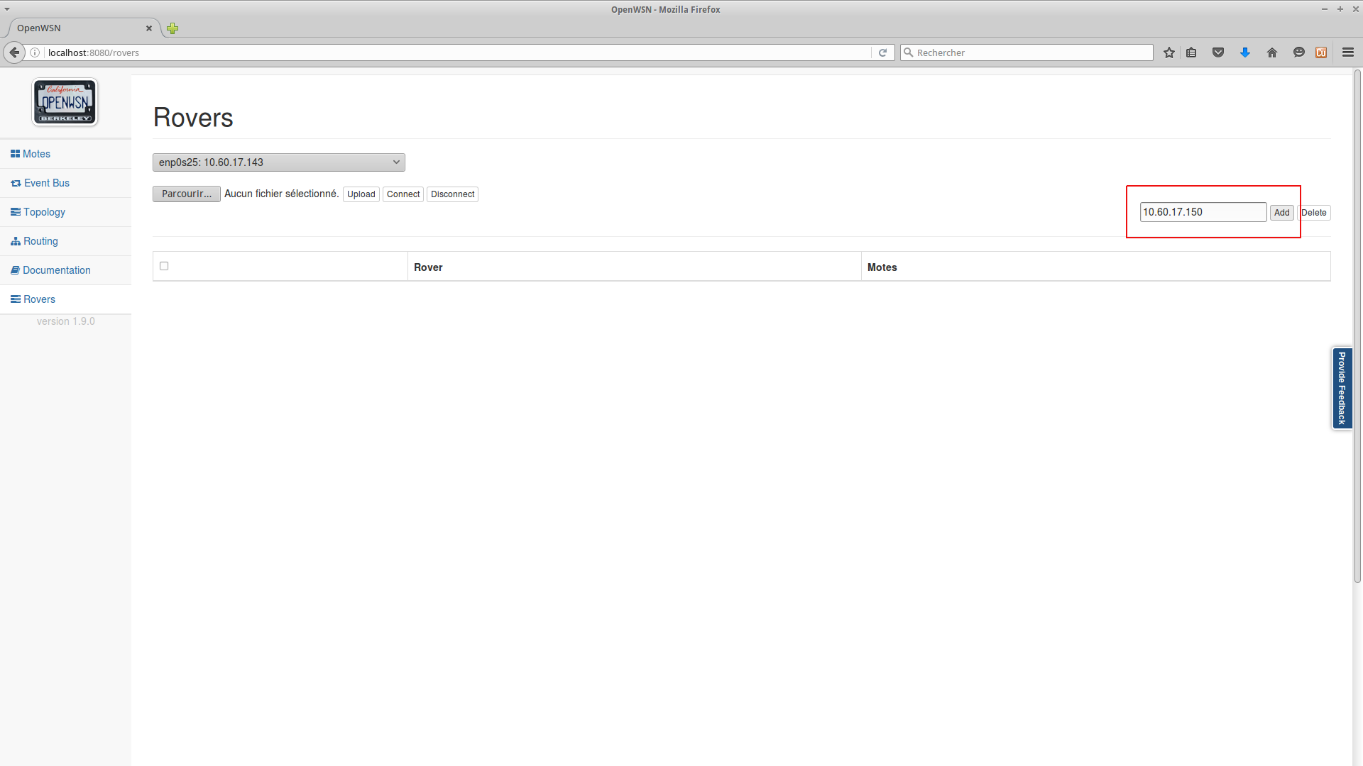
1. Open your favorite browser and type in: 192.168.10.160:8080 in the address box.



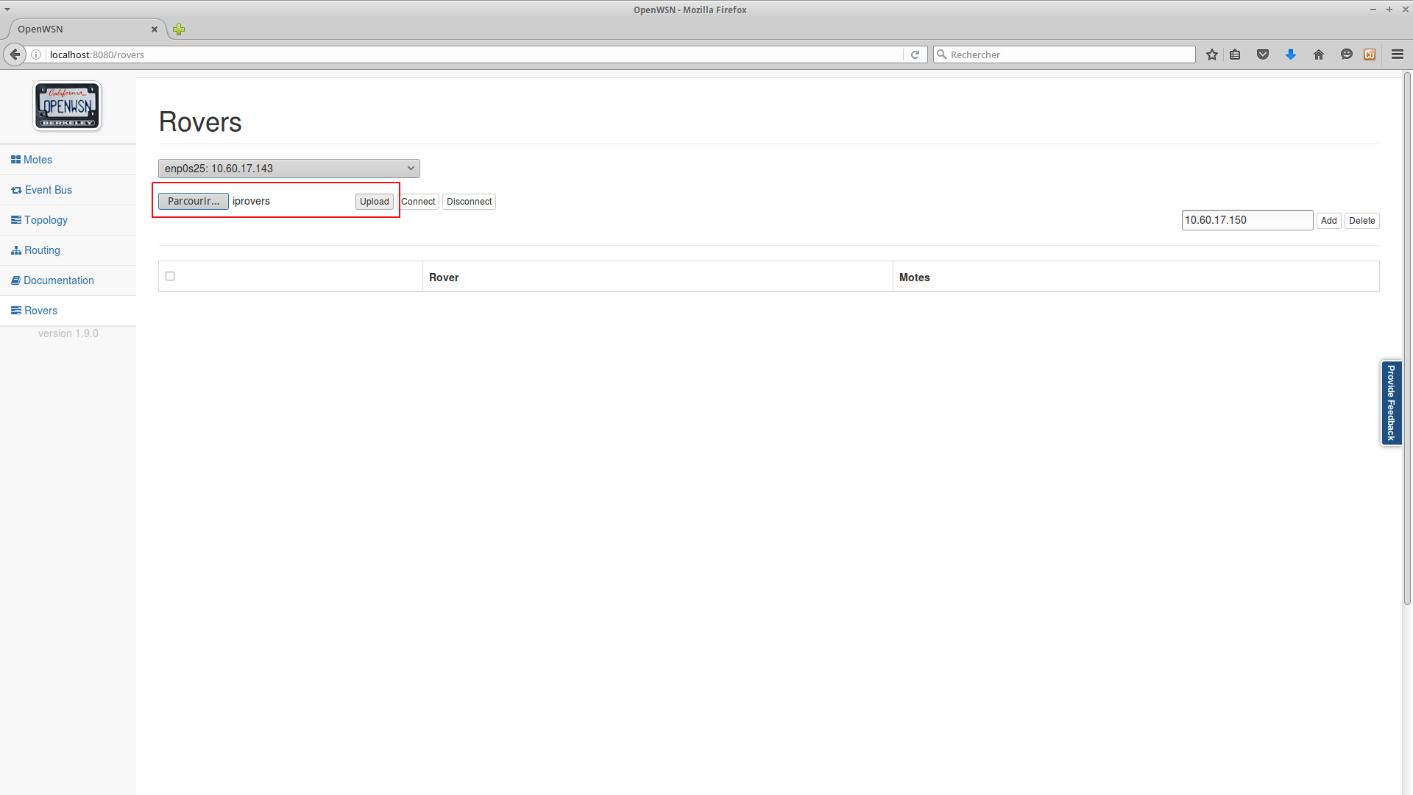
1. Go to Rover tab and choosing the interface to be connected with NUC



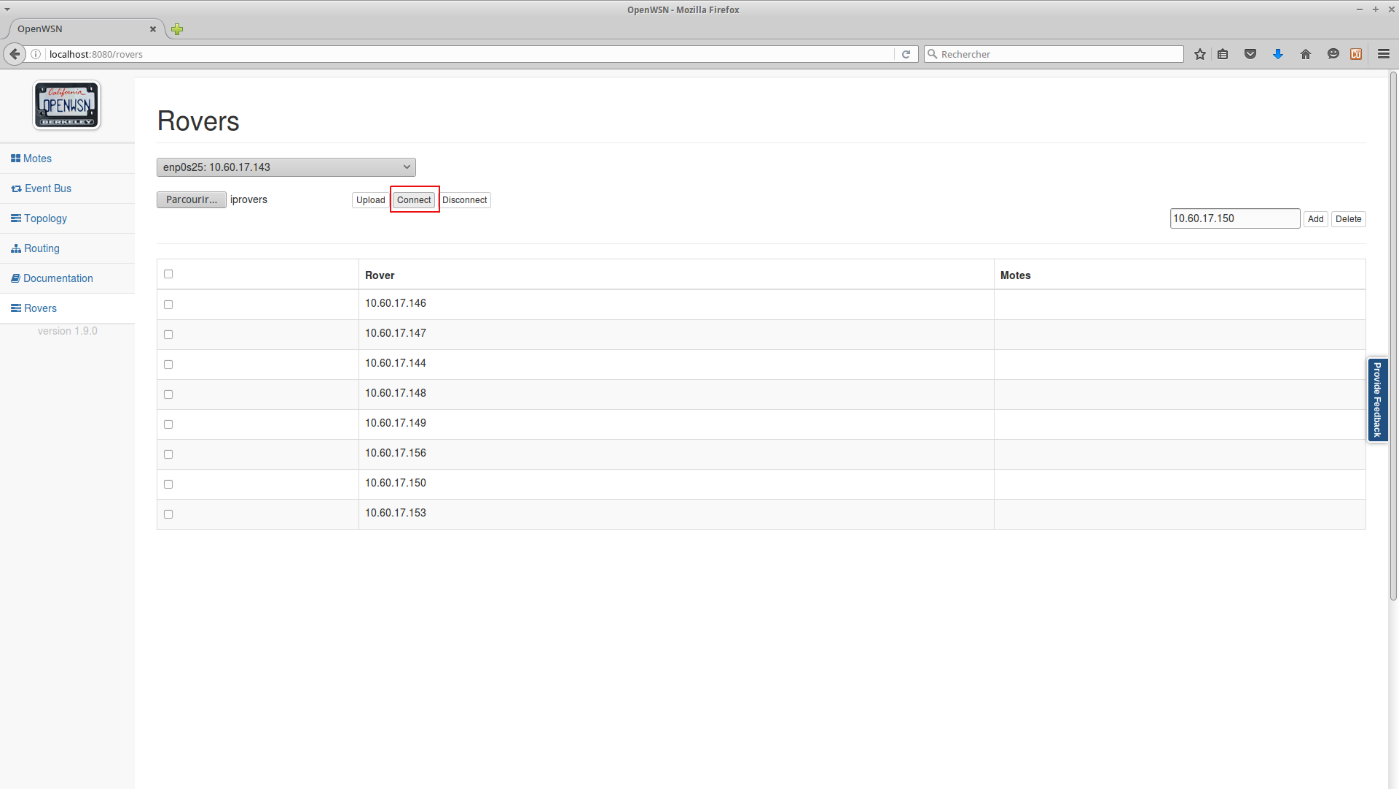
1. For each NUC you want to add type in the raspberry address in the rover and click "Add", (or select a text file containing a list of NUC IP addresses separated by "," and press "Upload"



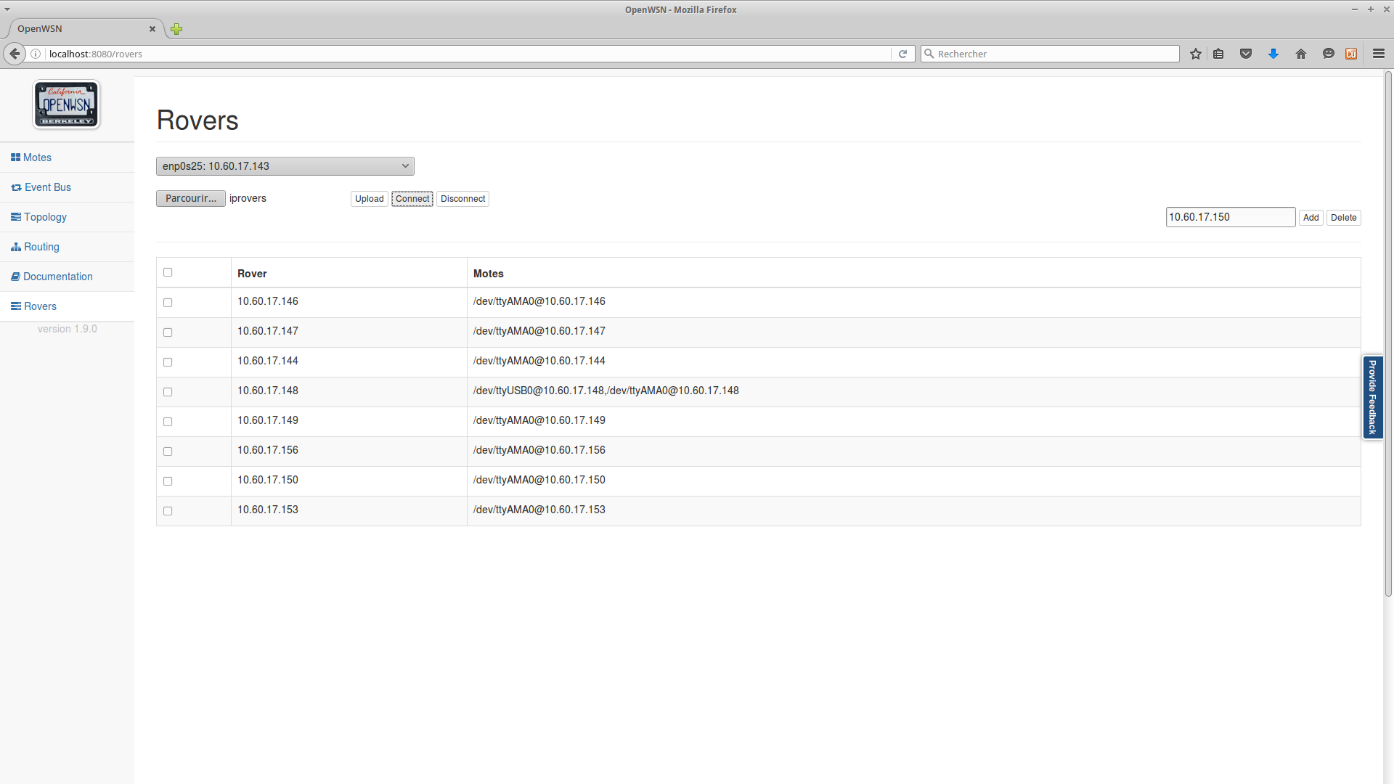
or :



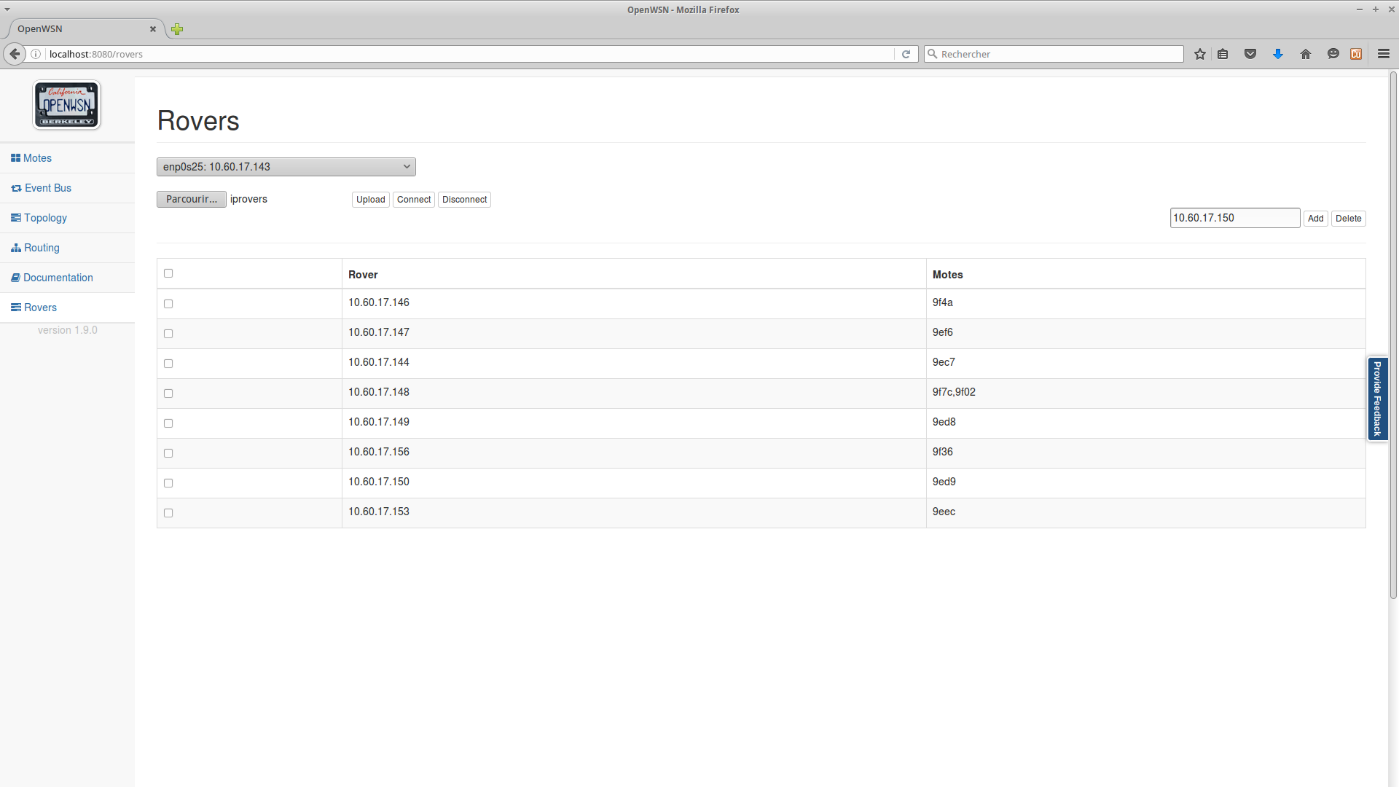
1. Once you have entered all your NUC IP addresses on the page press "Connect" (no need to select them, the select boxes are used to delete or disconnect IPs from the list). The ID of the motes connected to each NUC should appear.



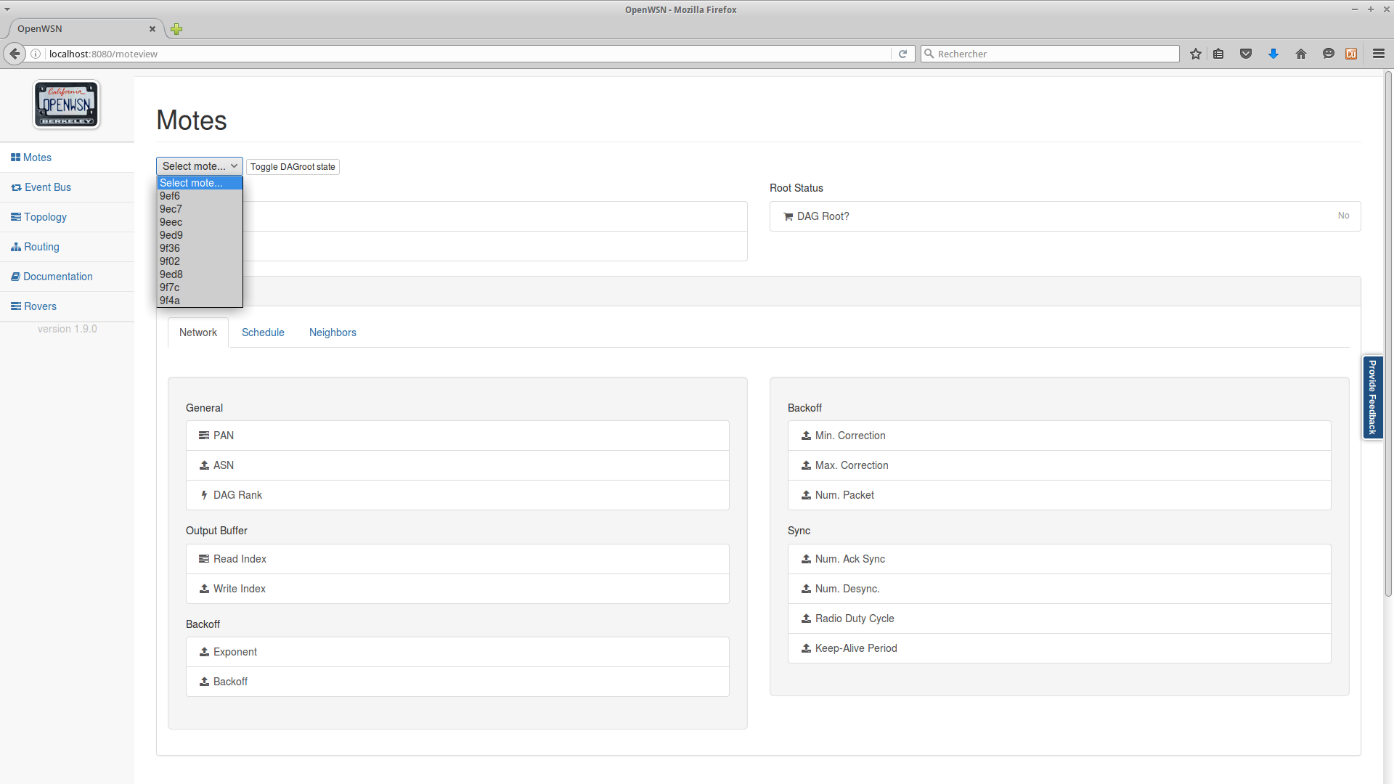
and then :



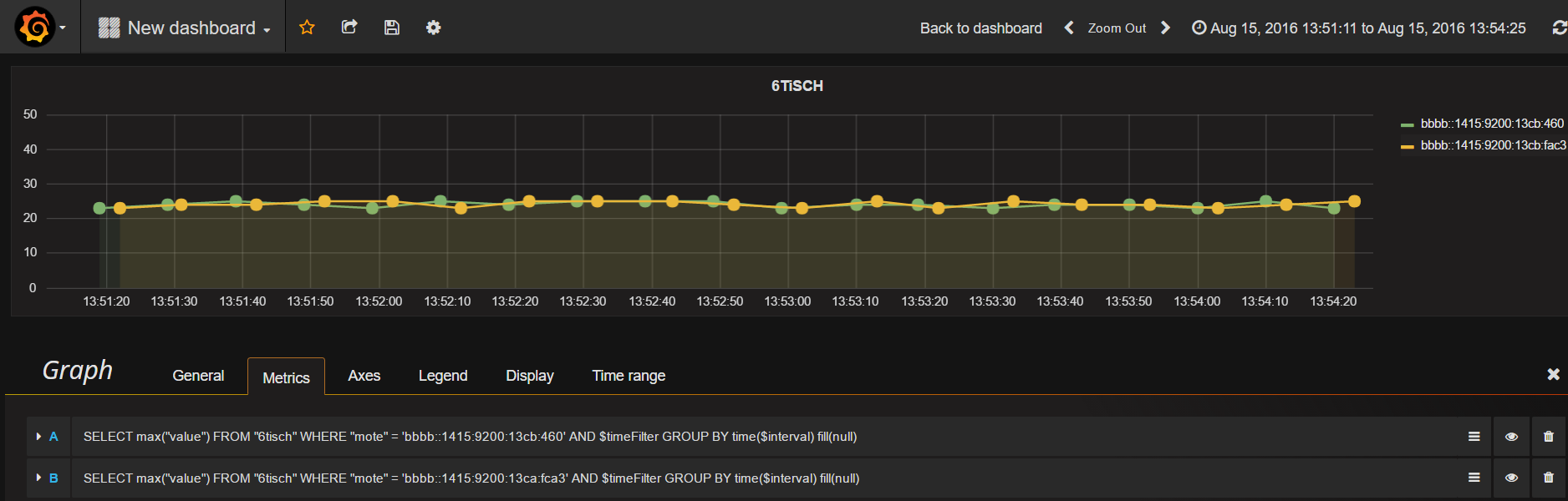
1. After a few seconds the ID should be updated to the last four bytes of the mote EUI-64



1. Go to Mote tab and pull down the mote list, then you can see all mote connected with NUC showing in your local computer.



1. Choosing one mote and click "toggle" to start the network. Done!
2. Display Web UI to see sensor value: <http://192.168.10.165:3000/>

****