1. **Install an Run OpenWSN on Linux**
2. **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 |

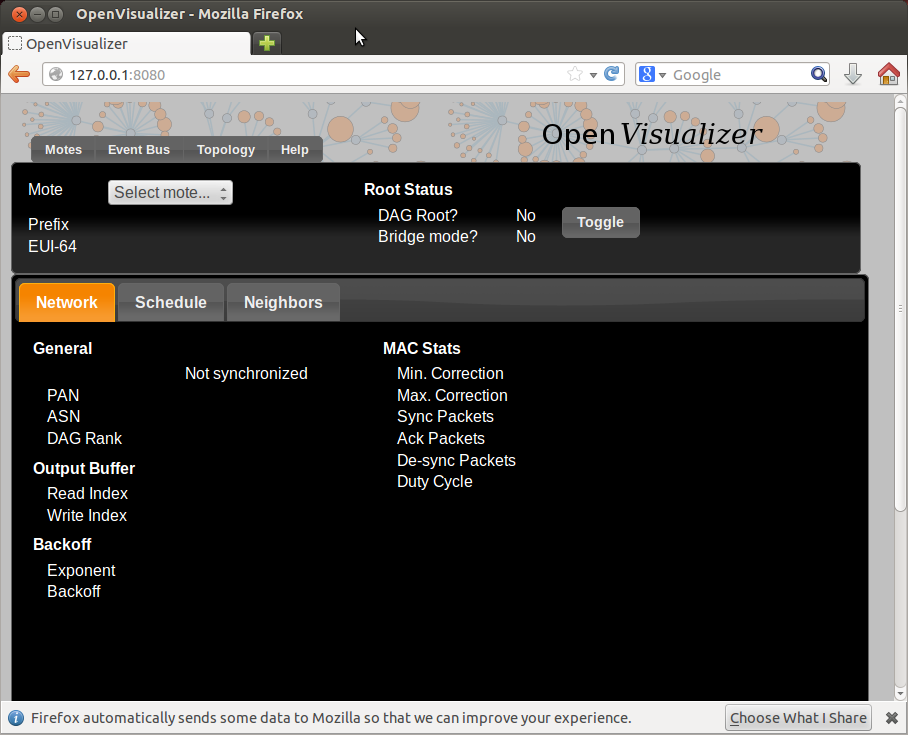
Start a simulation:

|  |
| --- |
| ~/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

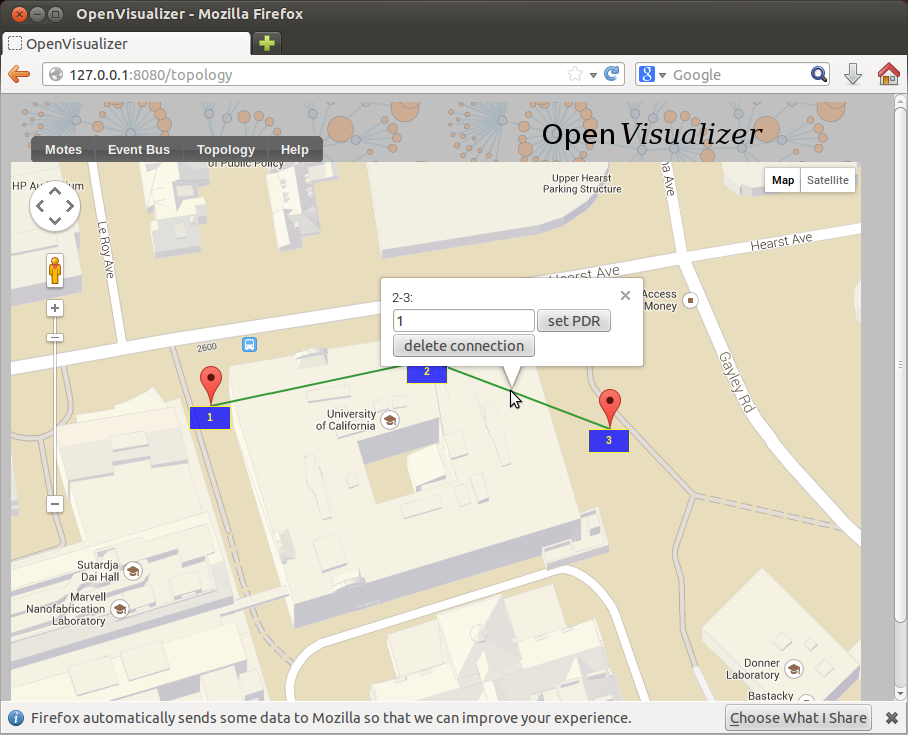
You need to right-click on the red balloon, not the blue rectangle. Confusing, right?

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.

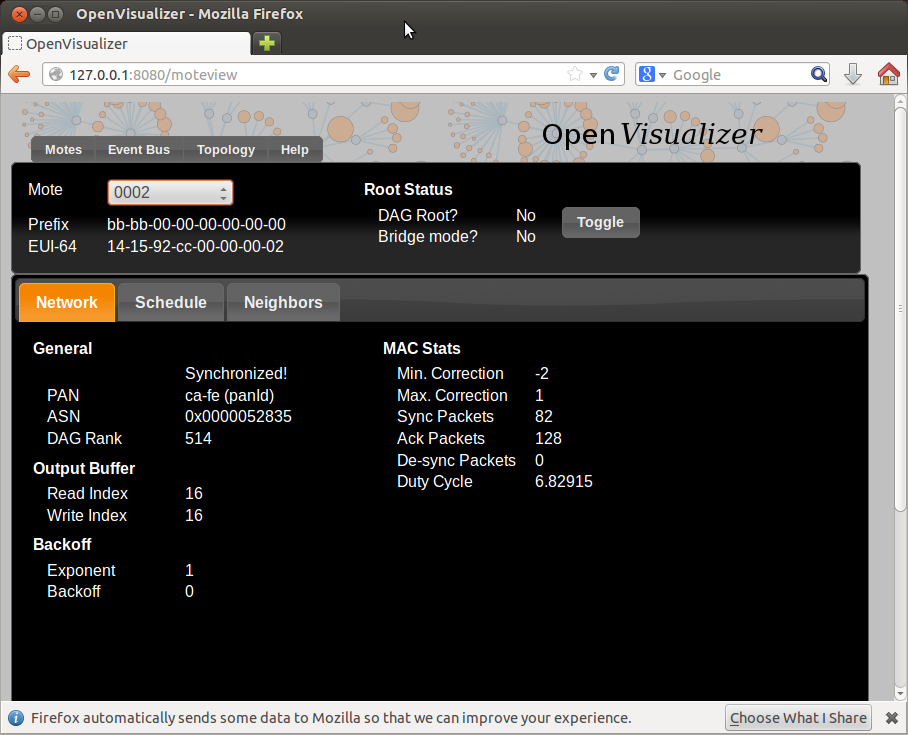


Back in the "Motes" tab, select mote 0001 and click on the "Toggle" button. You just declare mote 1 to be the root of your network (the DAGroot in RPL parlance, the sink in WSN parlance, the gateway, etc). You can select other motes and see that they quickly become "Synchronized!".

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/):

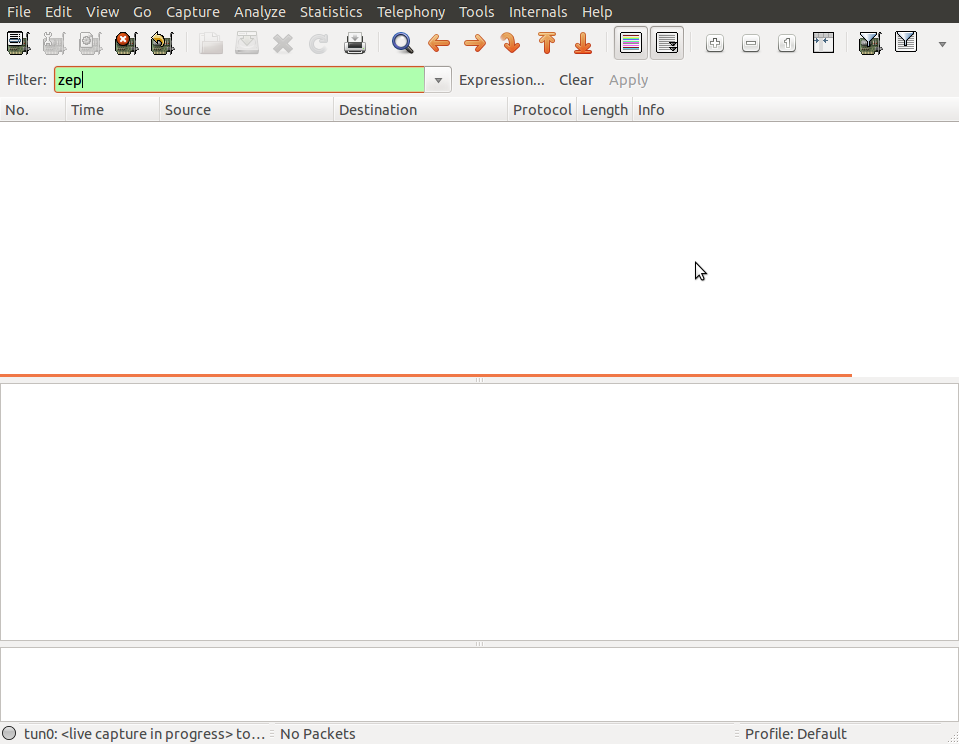
|  |
| --- |
| sudo apt-get install wireshark |

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

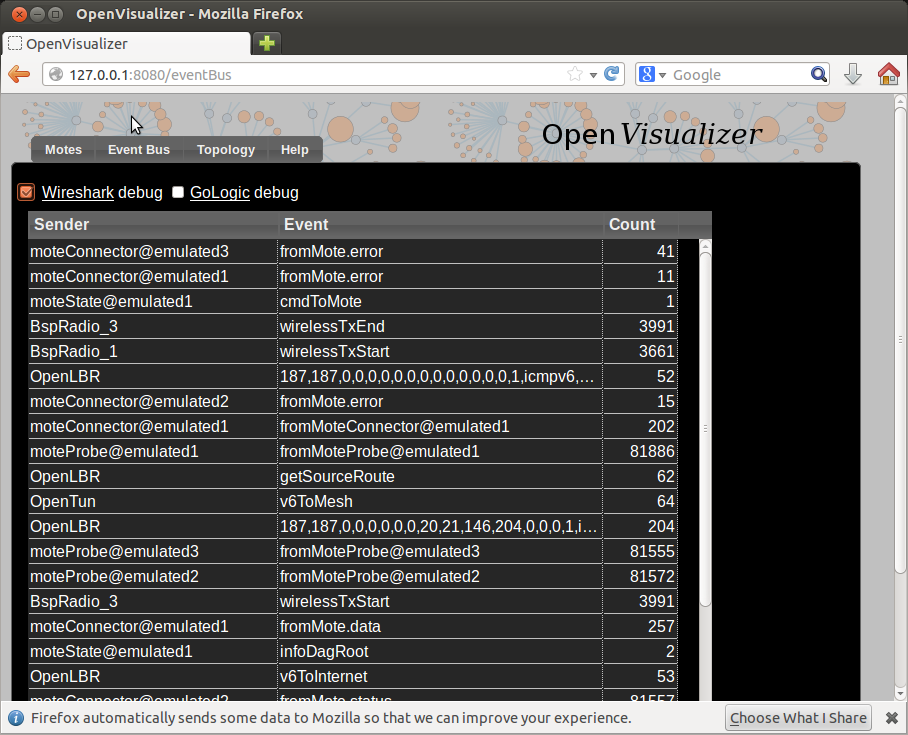
You need to start Wireshark with "sudo":

|  |
| --- |
| sudo wireshark |

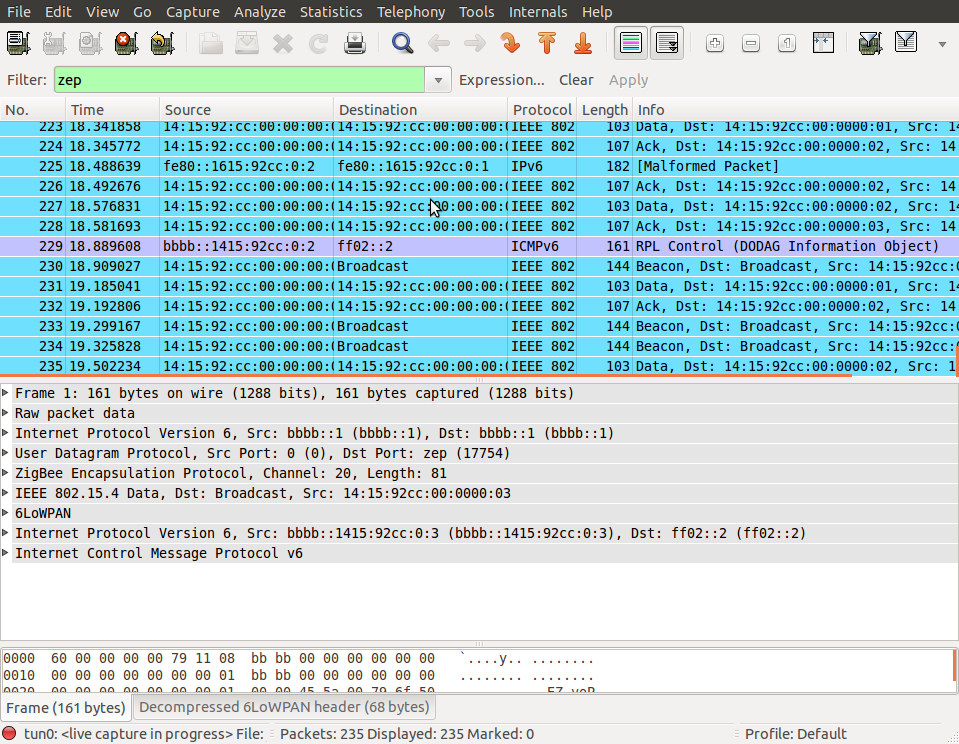
For now, you don't see any packets, that's normal:



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

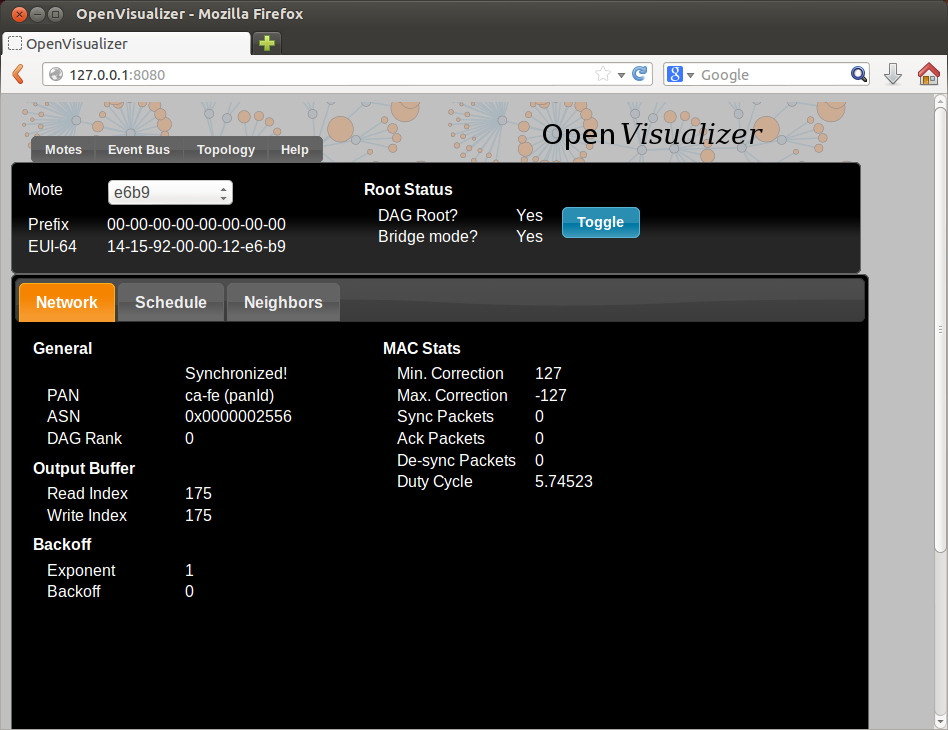
## Start a network

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

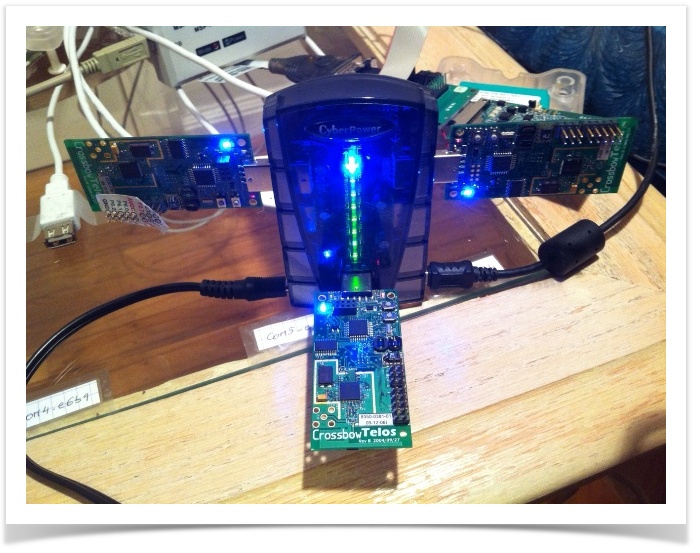
First, start the OpenVisualizer:

|  |
| --- |
| 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. **Install Source Insight for Checking and Modifying**   **Code**   * Install Source Insight   sudo apt-get install wine   * Download Source Insight   <http://www.sourceinsight.com/>   * Install   cd [directory storing Source Insight exe file]  wine SourceInsigh.exe // Note: don’t use root permission  (Use Serial : SI3US-925532-03991)  Ref :  <http://www.programering.com/a/MDMzUTNwATA.html>  <http://www.programmershare.com/3751677/>   * How to use   <https://www.youtube.com/watch?v=pBgNXJoZnW4>  <https://www.youtube.com/watch?v=1ui4s5VQ3JY>   * How to use Source Insight for Python   🡪 <http://www.programering.com/a/MTN3cDNwATI.html>  <http://pjack1981.blogspot.kr/2011/10/how-to-see-python-code-in-source.html>   * Change font size in Source Insight: options->Document options   Note: When add and remove project files, you should disable “*Show only known document types*”   1. **How to access Ubuntu remotely**   Ref: <http://c-nergy.be/blog/?p=5305> or <https://www.youtube.com/watch?v=sDZ6zmuYsho>  ***sudo apt-get update***  ***sudo apt-get install xrdp***  ***sudo apt-get install xfce4***  ***echo xfce4-session >~/.xsession***  ***sudo service xrdp restart***  *Sau khi cài đặt xong, trên máy tính Windows các bạn chỉ cần bấm phím* ***Windows + R****và gõ****mstsc****để bật chương trình Remote Desktop Connection lên. Tiếp theo bạn gõ IP của Raspberry Pi và kết nối đến như bình thường*   1. **Install Doxygen for Checking Code**   [**http://xmodulo.com/how-to-generate-documentation-from-source-code-in-linux.html**](http://xmodulo.com/how-to-generate-documentation-from-source-code-in-linux.html)  $ sudo apt-get install doxygen  $ sudo apt-get install graphviz  $ doxygen -g my\_proj.conf  $ vim my\_proj.conf  # document all entities in the project.  EXTRACT\_ALL = YES  # document all static members of a file.  EXTRACT\_STATIC = YES  # specify the root directory that contains the project's source files.  INPUT = /home/xmodulo/source  # search sub-directories for all source files.  RECURSIVE = YES  # include the body of functions and classes in the documentation.  INLINE\_SOURCES = YES  # generate visualization graph by using dot program (part of graphviz package).  HAVE\_DOT = YES  $ doxygen my\_proj.conf  $ cd html  $ firefox index.html |

1. **Simulation on Windows (Todo: How to run)**

[**https://bitbucket.org/6tisch/simulator/src/e2326c62268e48f18acdd21714076f5c782c62f7?at=master**](https://bitbucket.org/6tisch/simulator/src/e2326c62268e48f18acdd21714076f5c782c62f7?at=master)

[**https://gist.github.com/changtengfei/128236288fc4e96f3753#file-installation-of-6tisch-simulator**](https://gist.github.com/changtengfei/128236288fc4e96f3753#file-installation-of-6tisch-simulator)

[**http://stackoverflow.com/questions/4750806/how-to-install-pip-on-windows**](http://stackoverflow.com/questions/4750806/how-to-install-pip-on-windows)

Environment: window 7 ultimate 64-bit, python 2.7.9 32-bit.

Installation:

1. python 2.7, url: <https://www.python.org/ftp/python/2.7.9/python-2.7.9.msi>
2. Install environment variables (**Path** link) :

C:\Python27\Scripts

C:\Python27

1. matplotlib, url: <http://jaist.dl.sourceforge.net/project/matplotlib/matplotlib/matplotlib-1.4.0/matplotlib-1.4.0.win32-py2.7.exe>
2. scipy, url: <http://jaist.dl.sourceforge.net/project/scipy/scipy/0.14.0/scipy-0.14.0-win32-superpack-python2.7.exe>
3. numpy, url: <http://jaist.dl.sourceforge.net/project/numpy/NumPy/1.9.0/numpy-1.9.0-win32-superpack-python2.7.exe>
4. natgrid (natgrid-0.2.1-cp27-none-win32.whl), url: <http://www.lfd.uci.edu/~gohlke/pythonlibs/>

pip install natgrid-0.2.1-cp27-none-win32.whl

or

pip install natgrid (err)

1. pip install numpy six pyparsing pytz python-dateutil

**Note from cmd:**

How to install .whl file

* cd [directory containing natgrid-0.2.1-cp27-none-win32.whl file ]
* pip install natgrid-0.2.1-cp27-none-win32.whl

How to run .py file

* cd [directory .py file ]
* python file.py

One useful link: Unofficial Windows Binaries for Python Extension Packages, url: <http://www.lfd.uci.edu/~gohlke/pythonlibs/>

**Code Organization**

* bin/: the script for you to run
* SimEngine/: the simulator
  + - * Mote.py: Models a 6TiSCH mote running the different standards listed above.
      * Propagation.py: Wireless propagation model.
      * SimEngine.py: Event-driven simulation engine at the core of this simulator.
      * SimSettings.py: Data store for all simulation settings.
      * SimStats.py: Periodically collects statistics and writes those to a file.
      * Topology.py: creates a topology of the motes in the network.
* SimGui/: the graphical user interface to the simulator

**IDE Tool** Eclipse PyDev, pyCharm

**How To Run Simulation**

Step 1 : Edit Setting in **runSimOneCPU.py**, **runSimAllCPUs.py** and **Mote.py**

Step 2 : Run **runSimOneCPU.py** or **runSimAllCPUs.py** (faster) 🡪 generate data statistic log files

Step 3 : Run **plotStuff.py**, **plotStatsVsParameter.py (err)** and **plotAveStatsVsCycles.py (err)** 🡪 generate graphs

**Debug**: **Error with case X-threshold (plotStuff.py)**: Disable *pkPeriod = 'NA'*

*if m:*

*pkPeriod = float(m.group(1))*

*#else:*

*#* *pkPeriod = 'NA'*

**Note**: “topology.py” to calculate the average/max depth in network

1. **Questions**
2. **How is the medium accessed in a timeslot?**

*The IEEE 802.15.4e standard specifies that in a timeslot access to the medium can be based on request (CSMA-CA) for shared timeslots or guaranteed for guaranteed timeslots. Does OpenWSN provide both mechanisms? How is a guaranteed timeslot requested and used?*

->

You are right, the IEEE802.15.4e standard supports both:

* in "dedicated" slot, only two nodes are talking (in one direction), so there are no collisions possible. This means that no CSMA/CA access is needed.
* in "shared" slot, the sender uses a backoff algorithm to calculate when to retry in case it receives no acknowledgment from the neighbor it is talking to.

In the TSCH schedule used by the motes, a slot is "shared" when the "shared" bit is set; it is dedicated otherwise.

OpenWSN supports both. You can specify what you through the shared flag in the following function signature.

owerror\_t schedule\_addActiveSlot(

slotOffset\_t slotOffset,

cellType\_t type,

bool shared,

channelOffset\_t channelOffset,

open\_addr\_t\* neighbor

)

1. **How can I control channel hopping?**

*How can I control channel hopping?*

*When I change calculate Frequency function to control return value or change asnoffset, motes can not establish a connection.*

->

It should work if both tx and rx sides are in the same channel. Would you show what value you returned?

If only enabled the channel hopping (CH), it may take a while to detect the EB packet to join the network, since the mote only listen on synchronization channel (20). EB packet will be sent on different channel after enabling CH.

->

You can control the CH sequence with function: \_\_calculateFrequency \_\_.

To enable channel hopping, you can just uncomment [this line|https://github.com/opeSYnwsn-berkeley/openwsn-fw/blob/develop/openstack/02a-MAClow/IEEE802154E.c#L1937] and comment the previous line.

I have tried it with 3 motes in opensim and it works. I can see the dagroot received the DAO from mote 2 and 3 (mote 1 is dagroot)

1. **How do I define the communication channel?**

*Hi there,*

*I want to change the communication-channel of my OpenWSN network. I have many networks in some test environment and need to choose a special channel number. How I can change the number on before starting communication or compiling the resources?*

->

Okay, I selfanswered this question this searching in FW of OpenWSN.

The correct setting need to change is in the header of the MAC layer (sounds logical):

/openwsn/openwsn-fw/firmware/openos/openwsn/02a-MAClow/ in file IEEE802154.h

Just change the SYNCHRONIZING\_CHANNEL to the channel you wish

1. **Channel Hopping Working on Any Platform?**

*I've noticed that radio channel hopping appears to be disabled by default in the latest OpenWSN source code. I tried to enable the channel hopping (as described in this question ) for CC2538 but found that radio communications stopped working.*

*Is there any platform, other than the simulation, where channel hopping works for OpenWSN 1.8.0? If not, does anyone know of some OpenWSN fork/branch where channel hopping is working on some platform?*

*Thanks!*

*->*

Strangely, it seems that everything is now working with channel hopping enabled.

I really don't know if I simply made a mistake or if I encountered some other problem.

->

I would like to clarify the calculation for worst case.

The channel hopping sequence depends the ASN, not the packet. As we knew the calculation for used channel is: (channelOffset+ASN)%16, if all 16 channels were used:

If every slotframe there will be one ADV sent, and also channel hopping is enabled, it may takes 16 x slotframe\_length long to hear one ADV at worst case. Since the ADV is sent every 30 seconds and 30 seconds is an average value (randomly around 30 seconds https://github.com/openwsn-berkeley/openwsnfw/blob/develop/openstack/02b-MAChigh/sixtop.c#L120)

it's hard to knew at which slot, the ADV will be sent at synchronization channel (20 In OpenWSN).

Theoretically, you may never hear one ADV

1. **Wrong CRC in input buffer AND busy receiving when stop of serial activity**

->

I see from your log that you are running OpenWSN on the wsn430v13b, a board which is not fully supported yet. What might be happening is that your computer is too busy and doesn't answer the mote's "request" serial frame in time (I believe the max supported delay is 45ms). It looks like your computer answers, but only the first 3 bytes of the answer are received by the mote before that 45ms window is over.

There are a couple of things you can do:

* make sure your computer answers "fast". That is, don't run any virtual machine, trying running only the OpenVisualizer with a single mote connected
* try with a fully supported board, e.g. TelosB to convince yourself everything is OK
* as a last resort, increase the windows for the mote to receive commands by increasing the following #defines in schedule.c by the same amount:

NUMSERIALRX

SUPERFRAME\_LENGTH

Let us know how things go!

Thomas

1. **Objective Functions in RPL**

*Hello,*

*I am searching in the source code to see what objective functions for RPL are implemented and what routing metrics are they using, but I do not find. I am not sure I am looking where I have to. Are the "icmpv6rpl.c" and "forwarding.c" the only files with RPL code?*

*Thanks,*

*Oana*

->

I think you are looking for neighbors.c and link cost.c; by default the ETX metric is used in openwsn. Is it useful?

1. **Problems adding a new CoAP app**

*Hi to all, I'm facing some issues trying to play a bit with the CoAP module. I point out that I followed step-by-step the tutorials: Interact over CoAP and Adding a CoAP app. I ran some simulation, I don't use real motes.*

*I have already noticed the unfixed bug SW-140 - enabling rex application causes simulation to crash OPEN , and I'm facing the same problem, but I tried to add a new CoAP app (called rfx), with a very similar source code to rex, in order to verify my ability in adding a new app following the tutorials. But when I run a simulation, I have this:*

*created following virtual interface:*

*11: tun4: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UNKNOWN group default 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*

*Traceback (most recent call last):*

*File "bin/openVisualizerApp/openVisualizerWeb.py", line 343, in <module>*

*app = openVisualizerApp.main(parser)*

*File "/home/giorgio/Desktop/openwsn/openwsn-sw/software/openvisualizer/bin/openVisualizerApp/openVisualizerApp.py", line 190, in main*

*argspace.trace, argspace.debug*

*File "/home/giorgio/Desktop/openwsn/openwsn-sw/software/openvisualizer/bin/openVisualizerApp/openVisualizerApp.py", line 69, in \_\_init\_\_*

*import oos\_openwsn*

*ImportError: /home/giorgio/Desktop/openwsn/openwsn-sw/software/openvisualizer/build/runui/sim\_files/oos\_openwsn.so: undefined symbol: rfx\_init*

*where rfx\_init() is obviously the initialization function of my app, called in openwsn\_init().*

*I thought that could be necessary to modify the SConscript, but I'd like to have suggestions or explanations about this behavior.*

*Thank you in advance for your help!*

*Giorgio*

*->*

Hello Giorgio/Victoria

When in simulation, we have additional modifications to export our functions to the python´s bindings. Remember that each mote is a python object in simulation. In this case, a new parameter (self) is added to each function that we want to export, it is required for creating the object.

This process is done by a python script that modifies all required functions and files.

Please, take a look in this file:

openwsn-fw\firmware\openos\projects\python\SConscript.env

List your functions in the list "functionsToChange".

List your header files in the list "headerFiles".

Recompile the firmware code. Each file in the current compilation will be scanned, searching for these functions and headers. After recompiling, run again openvisualizer.

1. **Ask about "Adding a CoAP app" at version 1.8**

*Hello*

*Currently, i am using OpenWSN 1.8. But in your tutorial , topic "Adding a CoAP app" is for version 1.6*

*So, what is new configure i can add a CoAP app at version 1.8 ?*

*Hope get helps from you.*

*->*

In new version instead of openwsn.h, you should modify opendefs.h in openwsn\_fw/Inc/opendefs.h path.

1. **NOTE**

* Keep SuperFrame Length = 11
* Keep MaxactiveSlot = 9
* Schedule init should :

Slot0 : ADV – Slot1: TXRX Shared (Minimum to RPL mechanism) -> for nomal nodes

Slot0 : ADV – Slot1: TXRX Shared (Minimum to RPL mechanism) – Slot2 SERIALRX (Minimum to send data to serial port) -> for Root node

* Setup length of slotframe : Schedule.h
* Set prefix : openTun.py -19
* Enable Channel Hopping : IEEE802154E.c – 1937
* Setup time standards : IEEE802154E.h – 20->30

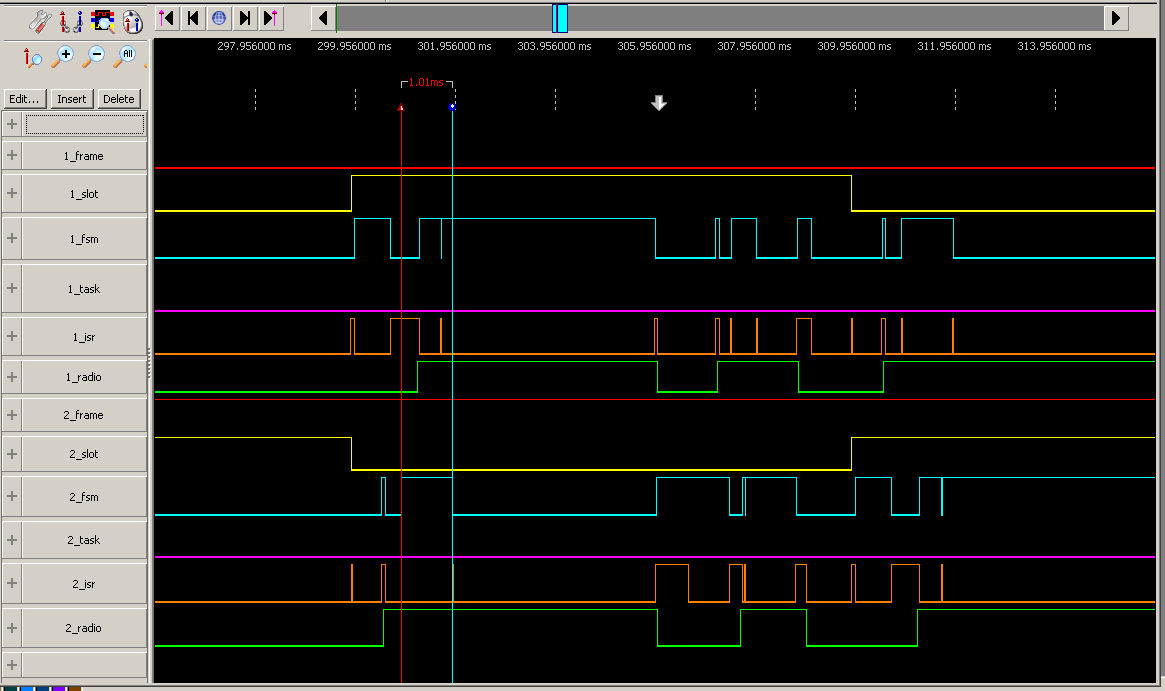
Some Parameter is related in type of HW:

openwsn-fw\bsp\boards\telosb\Board\_info.h - 56 ->71

* Việc truyền ADV packets thông thường được thiết lập tại slot đầu của slotframe của Mote. (Không quan tâm thời gian - tần suất bao nhiêu truyền lại một lần). ADV có thể được truyền bởi Root Mote hoặc tất cả các Mote trong mạng
* Guard Time

The Timeslot template indicate that we are using a guard time of 1ms. That is, the receiver starts listening for a packet 1ms before its supposed to (TsTxOffset) and keeps listening for 1ms after. This means that the sender can transmit up to 1ms late or early.

The figure below shows a slot where mote 1 is listening, but mote 2 is not sending. Mote 1 starts listening 1ms into the slot, then listens for 2\*1ms, and goes back to sleep since nothing is received.



The guard time indicates how often one needs to resynchronize. At 10ppm, it takes 100s for mote to desynchronize by 1ms. With a security factor of 3, this means that motes need to resynchronize every 30s.

Clock drift, guard time and resynchronization periods are linked. A smaller clock drift allows the guard time to be smaller, or the resynchronization period to be larger. A shorter guard time means that motes need to resynchronize more frequently, etc.

* Keep alive

As per the calculation above, master and slave need to communicate at least every 30s to keep synchronized. If they are exchanging data more frequently than that, synchronization is kept "for free". If that is not the case, the slave has to send a data packet with an empty payload just to receive the ACK to resynchronize. These packets are called keep-alive packets. Handling those packets is done exactly the same way as real data packets. However, since the payload is empty, the packet is simply not handed to the higher layer.

* On the counter

The width of the counter (i.e. the number of bits it contains) specifies the maximum duration is can count up. For the IEEE802.15.4e implementation, we need it to be able to count a slot length. With a 10ms slot and counter clocked by a 32kHz crystal, the counter needs to count up to 328.

We also need the counter to be able to count to a certain value, then roll over and generate an interrupt.

* uRES

<https://openwsn.atlassian.net/wiki/display/OW/uRES>

Information Elements (IE)

* Version of firmware only is used Openvisualizer having equivalent version
* Normally, each sensor has a channel-offset itself when add TX cell. TXRX cell for all node should be same channel-offset
* When enable Channel hopping at question 4, note that SYNCHRONIZING\_CHANNEL = return channel hopping at channel-offset of ADV.

Some cases: you can enable channel hopping + set SYNCHRONIZING\_CHANNEL = 11 are ok. Other cases can not.

* Check how to get Channel-offset by schedule\_getChannelOffset() *(returns The channel offset of current schedule entry)* -> **Can be Channel-offset of the first slot of schedule**
* Function : *activity\_synchronize\_newSlot(void) -> Listen EB (ieee802154e.c)*
* Enable Channel Hopping : IEEE802154E.c#L1937
* Schedule -> ADV+TXRX+TXRX+SERIAL+SERIAL is control plane form in Schedule (Control message sent via TXRX slots and data is given from SERIAL slots after that send it to parent node through next slots )
* The link layer consists of two sublayers:
* "02a-MAC" is the "lower MAC". It consists of the IEEE802.15.4e implementation and the associated timer. IEEE802.15.4e feeds from the queue of packets to be sent (the OpenQueue module), and the schedule (in the schedule schedule). It runs entirely in ISR (interrupt service routine) mode and "executes" the schedule without modifying it
* "02b-RES" is the "upper MAC" which is responsible for managing the schedule. It runs entirely in task mode (same goes for everything in the stack except IEEE802.15.4e)
* When FFDs device sends EB, joining mote will scan to listen EB on one channel offset for synchronization (*Thực ra ban đầu khi joining mote chưa join vào mạng nó ko biết giá trị ASN -> ASN = 0 -> Có thể set một channel listening bất kì cho nó hoặc set a random channel* ). After that schedule setups -> RPL routing starts -> some other policies setup -> Send Data

The best way is only Sink node sending EB but with a wide scale network, the signal wave can’t reach to all of joining node if these nodes are so far from sink node.

The worse way is same OpenWSN: All FFDs send EB same timeslot and channel offset but will happen collision.

* 6top send ADV each ADVTIMEOUT (30s)
* KA each 1s -> payload packet is sent per 1s
* **For RPL** :

+ OpenWSN uses TX/RX slots to broadcast DIOs, numTx and numTxACK values can collected from these slots (checked on OpenWSN-activity\_ti1ORri1())

+ On emulator numTx and numTxACK values only can collected from TX slots and don’t have TX/RX slots for broadcast DIOs (broadcasting DIOs is default )

+ numTx and numTxACK values displayed on simulation windows is gotten by schedule\_indicateTx() and not by neighbors\_indicateTx() for ETX value in RPL

+ Each node collests neighbors set by function neighbors\_indicateRx() based on packets coming (DIO-broadcast,DAO, KA, Data)

+ numTx and numTxACK values are collected from dedicated slots (Simulator) or dedicated slots + TX/RX slot (openWSN) bw 2 nodes. This can’t be applied with broadcast packet (ADVs, DIOs)

+ neighbors\_indicateRxDIO() indicates DIOs received and update DAGrank for node based on rank of neighbors in DIOs

# Information Elements

[**https://openwsn.atlassian.net/wiki/display/OW/uRES**](https://openwsn.atlassian.net/wiki/display/OW/uRES)[**https://openwsn.atlassian.net/wiki/display/OW/The+Big+Picture**](https://openwsn.atlassian.net/wiki/display/OW/The+Big+Picture)

//------------------------------------------------------------------------------------------------------------------------/

**OpenSerial Notes:**

[***https://openwsn.atlassian.net/wiki/display/OW/Serial-To-Mote***](https://openwsn.atlassian.net/wiki/display/OW/Serial-To-Mote)

[***https://openwsn.atlassian.net/wiki/display/OW/Serial+Format#SerialFormat-DATAFrame***](https://openwsn.atlassian.net/wiki/display/OW/Serial+Format#SerialFormat-DATAFrame)

<https://openwsn.atlassian.net/wiki/questions/34308186>

<https://openwsn.atlassian.net/wiki/questions/users?username=vpimentel>

<https://openwsn.atlassian.net/wiki/questions/33620079>

**How can I implement SerialTester.py in the openvisualizer?**

[**https://openwsn.atlassian.net/wiki/questions/33620079**](https://openwsn.atlassian.net/wiki/questions/33620079)

serialTester is used solely to test the serial communication between a computer and a mote. The mote can be running any application which contains openserial; oos\_openwsn is one of them.

The serial protocol between the mote and the computer is documented in [1]. Per the serial format, the computer can send a number of commands to the mote. One of them is SERFRAME\_PC2MOTE\_TRIGGERSERIALECHO [2]. When sending this command, the mote simply echoes back the bytes.

serialTester is just a little script which you run on the computer and which tests this serial communication, by calling the SERFRAME\_PC2MOTE\_TRIGGERSERIALECHO command over and over. It is composed of two files in the SW repo:

**the serialTester module [3],**

**the serialTesterCli application [4].**

To start a serial test:

**connect a single TelosB mote to your computer**

**double click on serialTesterCli application**

**follow the command line help to pick a number of packets to transmit, their size, etc.**

I hope this answers your question,

Thomas

*[1] Serial Format*

*[2] https://github.com/openwsn-berkeley/openwsn-fw/blob/develop/firmware/openos/drivers/common/openserial.h#L59*

*[3] https://github.com/openwsn-berkeley/openwsn-sw/blob/develop/software/openvisualizer/openvisualizer/moteConnector/SerialTester.py*

*[4] https://github.com/openwsn-berkeley/openwsn-sw/blob/develop/**software/openvisualizer/bin/serialTesterCli/serialTesterCli.py*

- PC2MOTE through serial port:

*\_cmdToMote\_Handler(self,sender,signal,data)*

*\_sendToMoteProbe(self,datatoSend)*

* Check output serial data :  *software/openvisualizer/bin/serialTesterCli /**serialTesterCli.py*
* ERR\_INPUTBUFFER\_LENGTH (Note : Max 60 bytes UDP)
* openserial\_printData((uint8\_t\*) (request->payload), request->length) // Print Data to PC
* reply = openqueue\_getFreePacketBuffer(COMPONENT\_UECHO);

packetfunctions\_reserveHeaderSize(reply,request->length); // add payload-header

memcpy(&reply->payload[0],&request->payload[0],request->length); // having many bytes from payload[0]

* packetfunctions\_reserveHeaderSize(msg,sizeof(uint8\_t)); // add payload-header \*((uint8\_t\*)(msg->payload)) = value\_nextHeader; // having 1 byte
* **Bug Function**: Check error or measure time stamp.

openserial\_printError(COMPONENT\_SIXTOP,ERR\_NO\_FREE\_PACKET\_BUFFER,

(errorparameter\_t)XX,

(errorparameter\_t)0);

* Set fixed topology at *topology.c*
* Openbridge.c is only used for DAGroot for bridging data which is sent from PC to Mote.
* Data (Payload or cmd) is sent from PC to DAGroot through SerialRX slot (IEEE802154E.c#880) by using function: *openserial\_stop() – openserial\_startInput().*

Data from Mote to PC (IEEE802154E.c#805) by using function: *openserial\_stop() – openserial\_startOutput().*

Maximum data sent from PC to DAGroot through a SerialRX slot is 128B + 8B address next hop.

Ref: <https://openwsn.atlassian.net/wiki/questions/46235660>

* Normally, OpenWSN setup (SUPERFRAME\_LENGTH - MAXACTIVESLOTS) slots for SerialTX (IEEE80215.4.E#802) using *openserial\_startOutput().* Furthermore, we add one more type of slot (CONTROL) for sending data from Mote to PC.
* When set up Slotframe length, you should choose 11. Other numbers maybe make syn error.
* **Edit log format at**:

+ /home/thang/openwsn-1.8/openwsn-sw/software/openvisualizer/bin/openVisualizerApp/logging.conf

//---------------------------------------------------------

*[formatter\_console]*

*format=%(asctime)s %(levelname)s %(message)s*

*datefmt=*

*#datefmt=%H:%M:%S*

//---------------------------------------------------------

+ StackDefines.py

+ opendefs.h

//------------------------------------------------------------------------------------------------------------/

//---------------------------------- How to add new App------------------------------------------

With OpenWSN 1.9

* openwsn-fw/openapps/SConscript

*ex: add* ***uinject*** *at defaultAppsInit structure: https://github.com/openwsn-berkeley/openwsn-fw/blob/develop/openapps/SConscript*

* openwsn-fw/openstack/SConscript
* openwsn-fw/projects/python/SConscript.env 🡪 need for only python simulation
* openwsn-fw/openapps/openapps.c 🡪 add ‘.h’ and ‘\_init()’ function

//-----------------------------------How to add new module--------------------------------------

With OpenWSN 1.9

* openwsn-fw/openstack/SConscript: add ‘.c’ and ‘.h’ file
* openwsn-fw/openstack/openstack.c: add ‘.h’ file and ‘\_init()’ function 🡪 Adding depends on \_init() having info or not. If \_init() is empty (ie. otf\_init() ) , you can’t need add into.
* openwsn-fw/projects/python/SConscript.env 🡪 add functions (need for only python simulation)

//----------------------------Note when add new function in existing .c file----------------------

* openwsn-fw/projects/python/SConscript.env 🡪 need for only python simulation
  + - * 1. **Main Commands**

cd openwsn/openwsn-fw

cd openwsn/openwsn-sw/software/openvisualizer

cd openwsn/openwsn-fw/openapps/cleds

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

scons board=python toolchain=gcc oos\_openwsn

sudo scons runweb

scons runweb --sim --simCount=2

scons rungui --sim --simCount=2

scons runweb --sim --pathTopo ../../../topology/5motes.json

scons runweb --sim --pathTopo=../../../topology/5motes.json (for Window?????)

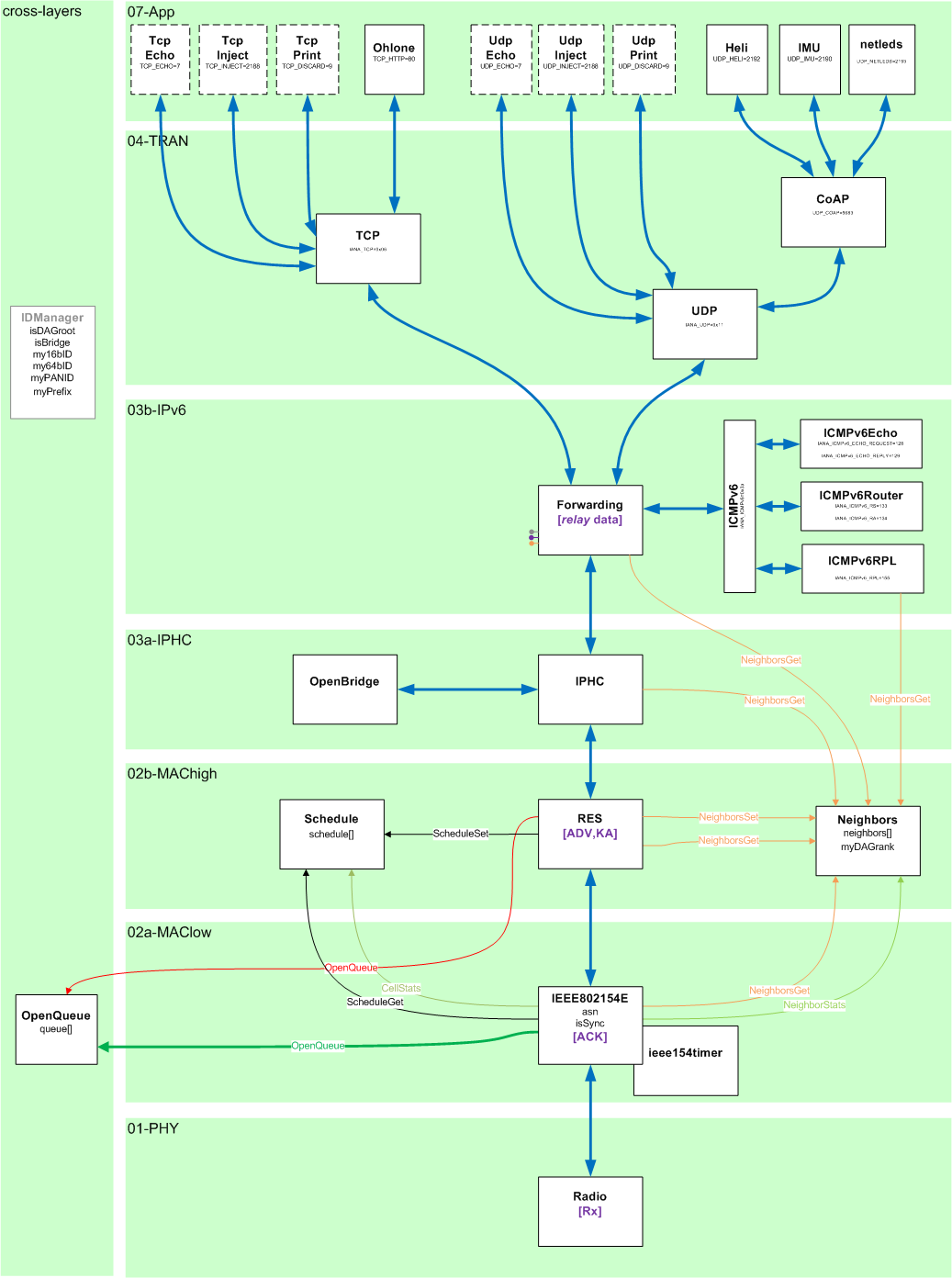
dmesg

bbbb::1415:92cc:0:2

bbbb::1415:9200:13:e63b

Simulation mote PORT: emulatedXX

* + - * 1. **Component Pictures**



**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

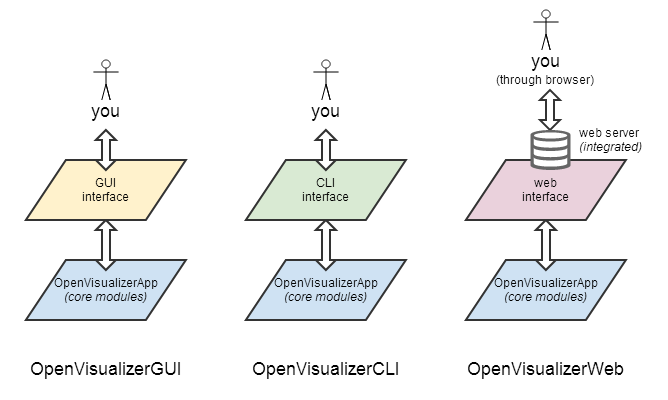
**bool l1\_crc;** // did received packet pass CRC check?

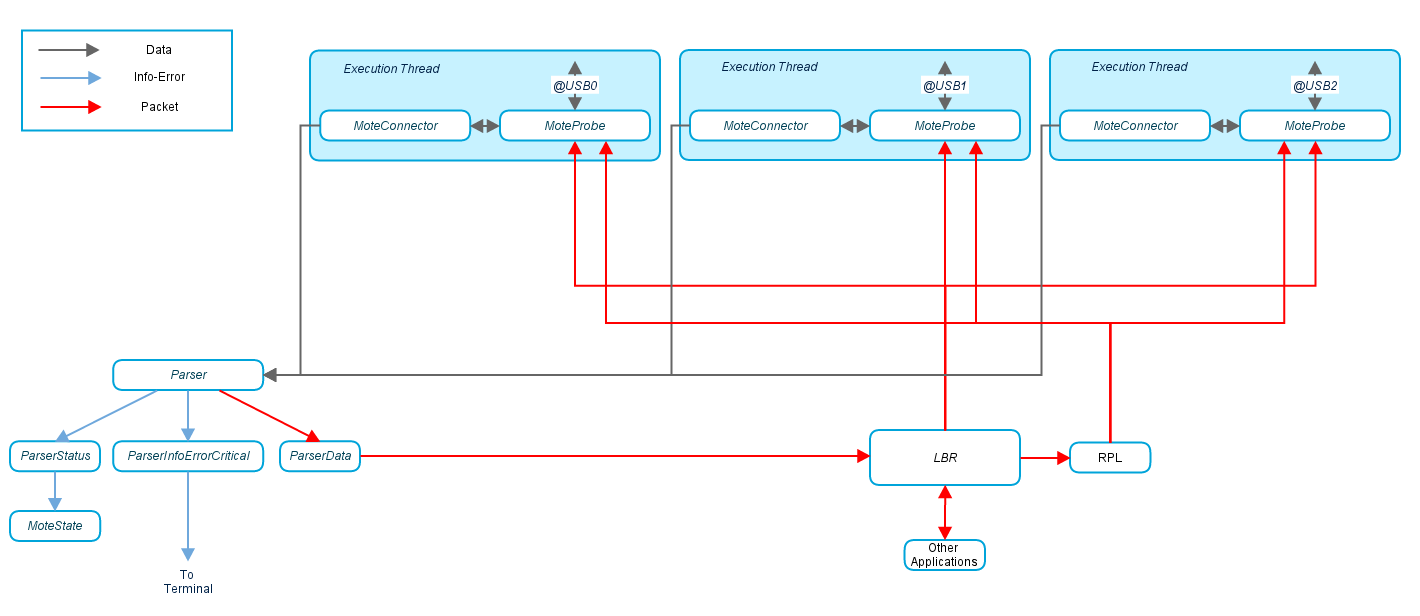
//the packet

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

**} OpenQueueEntry\_t;**

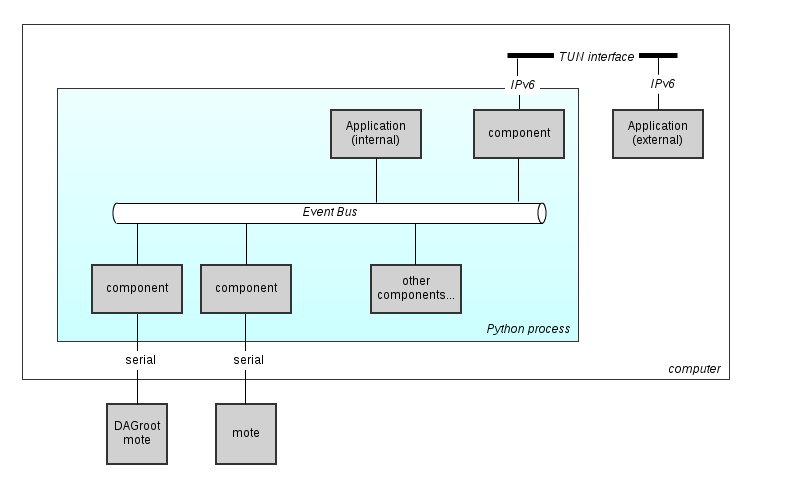
**OpenVisualizer Code Organization**

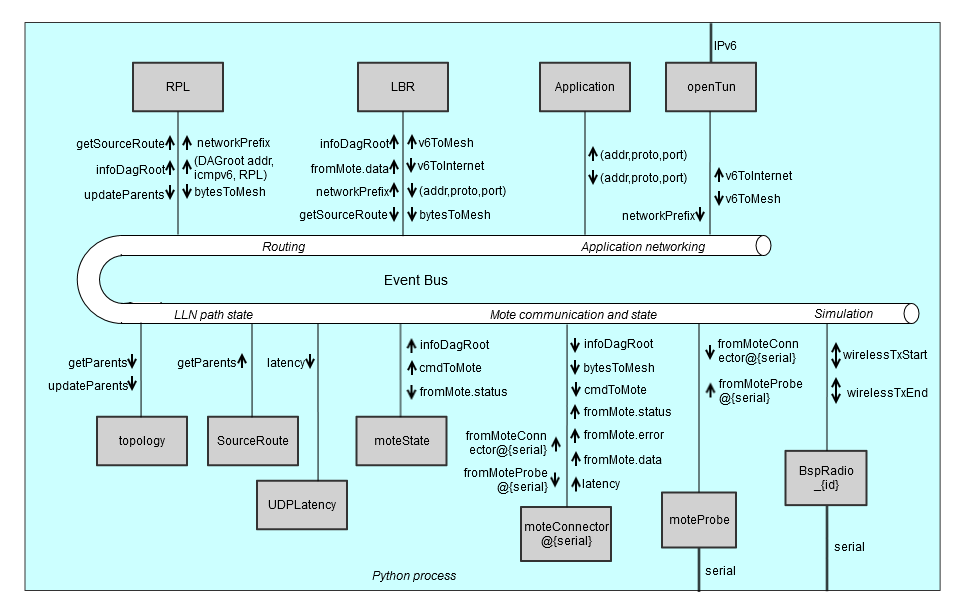
****

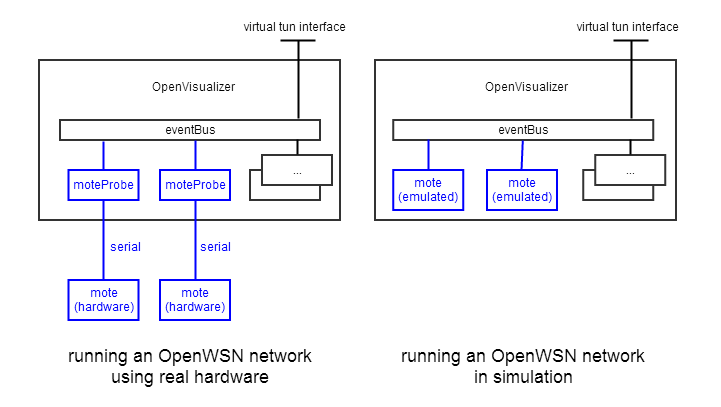
****

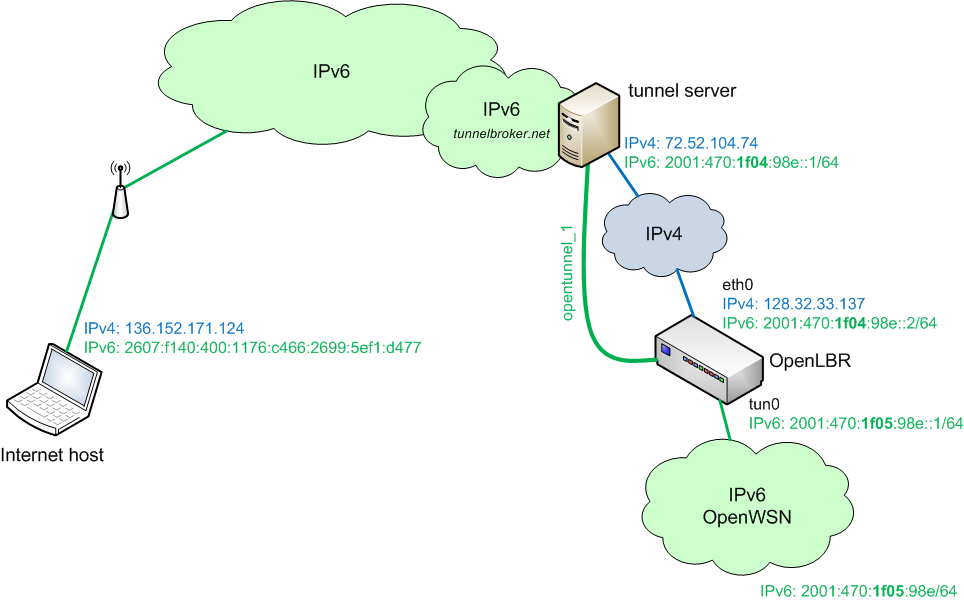
Modules

* [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. **DeTAS Deployment**

1. Debug binary loading when use openwsn 1.4 and python 2.7

<https://openwsn.atlassian.net/wiki/questions/67403783/loading-binary-error-when-build-openwsn-1.4>

2. Collect data statistics 🡪 cannot run due to non-compatibility

<https://openwsn.atlassian.net/wiki/display/OW/How+to+gather+your+app+data+from+your+mote+on+your+own>

* DeTAS (openwsn-fw 1.4) only is used with openwsn-sw in zip
* How to print info statistics to a log file:
  + Use function openserial\_printError
  + Modify on StackDefines.py
  + Modify on ParserInfoErrorCritical.py: \_translateErrorDescription(..,..,..,..)
* Ref : <https://github.com/openwsn-berkeley/openwsn-sw/blob/develop/software/openvisualizer/openvisualizer/moteState/moteState.py>
* How to print xls log
  + Install xlwt : sudo pip install xlwt
  + Sample

<http://www.blog.pythonlibrary.org/2014/03/24/creating-microsoft-excel-spreadsheets-with-python-and-xlwt/>

**import** xlwt

*#----------------------------------------------------------------------*

**def** main():

""""""

book = xlwt.Workbook()

sheet1 = book.add\_sheet("PySheet1")

row = sheet1.row(1) # row 1

row.write(0, ‘A’) # col 1.1

row.write(1, ‘B’) # col 1.2

row = sheet1.row(1) # row 2

row.write(0, ‘A’) # col 2.1

row.write(1, ‘B’) # col 2.2

book.save("test.xls")

*#----------------------------------------------------------------------*

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

main()

* + - * 1. **WirelessHART and 802.15.4e in NS2**
        2. **Hacking Raspberry Pi**

<http://raspberrypi.vn/setup-raspberrypi/huong-dan-cai-dat-va-truy-cap-raspberry-pi-thong-qua-remote-desktop-1527.pi>