TOSSIM

Răzvan Musăloiu-E.

Hi, I'm Răzvan Musăloiu.
I'm the maintainer of the Deluge T2 but I care about several other things in TinyOS.
Today I'm here to talk about TOSSIM.

What is TOSSIM?

Discrete event simulator

ns2

TOSSIM is the TinyOS simulator. It's a discrete event simulator. This means that it maintains a sorted queue of events. At each step the event with the oldest timestamp is extracted and executed. Each event can add new events to the queue.

A popular simulator that works in the same way is ns2.

Alternatives

Cycle-accurate simulators

Avrora, MSPSim

Another different type of simulators are the cycle-accurate ones which simulate the MCU at the instruction level. A consequence of this is the fact that they take as input a binary image (and ELF file) which makes them agnostic to the way the binary was generated (TinyOS, Contiki, LiteOS, plain C, etc).

Such a simulator for the MicaZ mote is Avrora and the one for TelosB is MSPSim.

Two directions

Port

make PC a supported platform

TOSSIM
in tinyos-1.x

Virtualize

simulate one of the supported platforms

TOSSIM
in tinyos-2.x

Back to TOSSIM now. There are two ways in which we can achieve the goal of being able to run TinyOS on a PC. We can either make the PC a supported platform in TinyOS or we can virtualize one of the already existing ones. The first solution is implemented in TinyOS 1.x while the second one is implemented in TinyOS 2.x.

Features

- Simulates a MicaZ mote
  - ATmega128L (128KB ROM, 4KB RAM)
  - CC2420
- Uses CPM to model the radio noise
- Supports two programming interfaces:
  - Python
  - C++

The mote that TOSSIM simulates is MicaZ. This is based on a Atmel ARmega128L, an 8-bit MCI with 128KB of ROM and 4KB of RAM, and a CC2420, a 802.15.4 radio.

One important thing in TOSSIM is the way the radio communication is simulated. The main role is played by the CPM model of the radio noise. We'll talk more about this later.

TOSSIM can be controlled using two languages: Python and C++. In the rest of the talk I'm going to talk about Python.

Anatomy

TOSSIM

```
tos/lib/tossim
tos/chips/atm128/sim
tos/chips/atm128/pins/sim
tos/chips/atm128/timer/sim
tos/chips/atm128/spi/sim
tos/platforms/mica/sim
tos/platforms/micaz/sim
tos/platforms/micaz/chips/cc2420/sim
```

Application

```
Makefile
*.nc
*.h
```

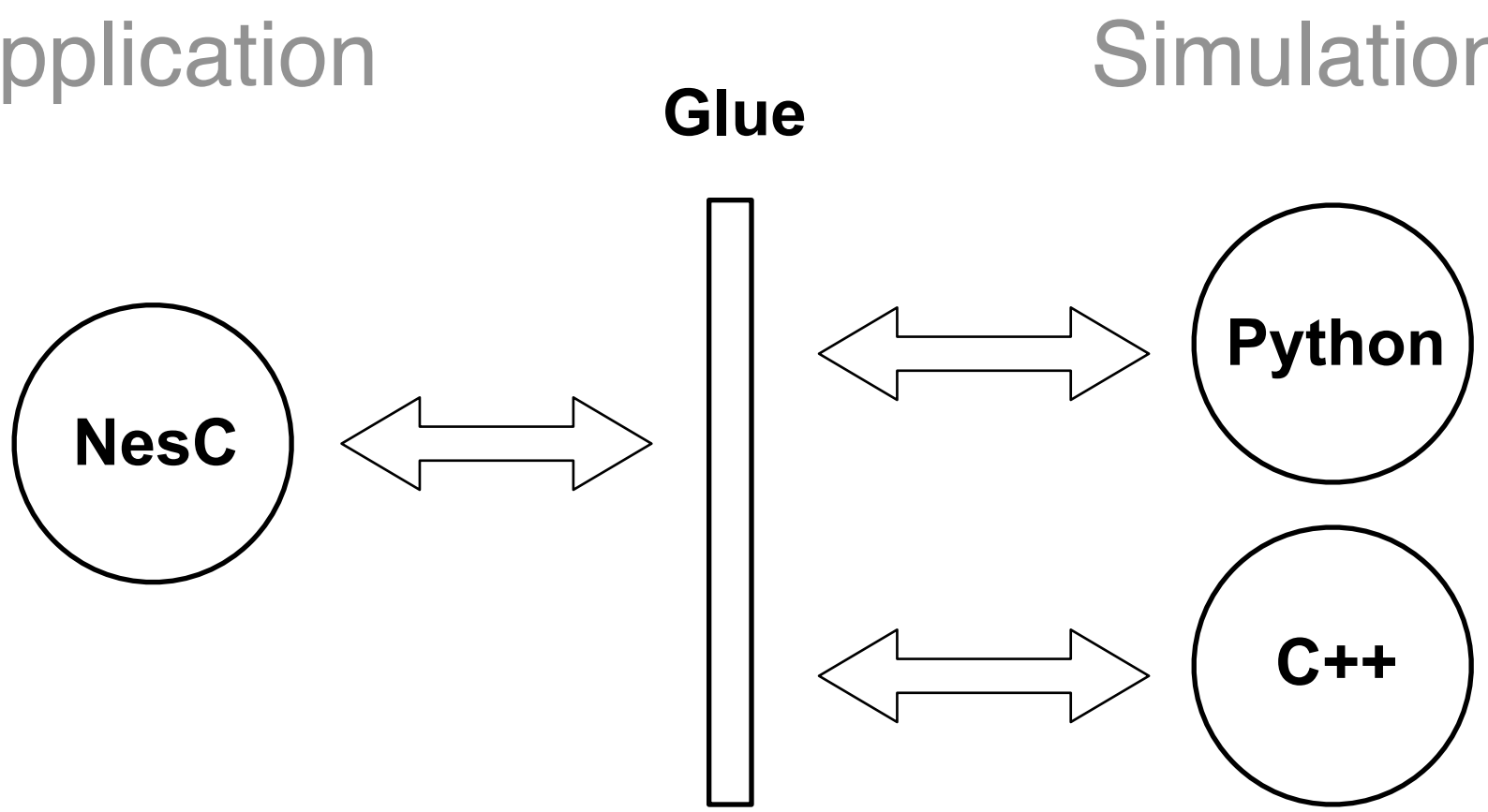
Simulation Driver

```
*.py | *.cc
```


The bulk of the code related to TOSSIM is in tos/lib/tossim. The code that virtualize the MicaZ is in several sim/ folders in tos/chips/atm128, tos/platforms/mica and tos/platforms/micaz.

To do a simulation we also need an application which has a Makefile, several .nc files and potentially some .h ones. One last piece we need is a simulation driver which is either a Python script or a C++ program.

Quick Overview



To summarize, the main components of a TOSSIM simulation are: the application, a simulation script and some glue that links this two things together. As I mentioned before, beside Python we can also use C++ to drive the simulation.

The Building Process

```
$ make micaz sim
```

1. Generate an XML schema — *app.xml*
2. Compile the application — *sim.o*
3. Compile the Python support — *pytossim.o*, *tossim.o*, *c-support.o*
4. Build a share object — *_TOSSIMmodule.o*
5. Copying the Python support — *TOSSIM.py*

```
$ ./sim.py
```

Here is how a typical building process looks like. The make command is “make micaz sim” and there are 5 things that take places: (1) an XML description of the wirings is generated (app.xml), (2) the application is compiled (sim.o) together with (3) the glue necessary for Python/C++ (pytossim.o, tossim.o, c-support.o), (4) all the object files are merge in one big share object file called _TOSSIMmodule.o and finally (5) the TOSSIM.py is copied from tos/lib/tossim to the application directory. After this we can run you simulation script which in this case is called sim.py

TOSSIM.py

Tossim

Radio

Mote

Packet

Mac

The TOSSIM.py offers 5 important classes. The main one is called Tossim and it is used as a factory object for all the other 4: Radio, Mote, Packet and Mac.

TOSSIM.Tossim

.getNode() → TOSSIM.Mote

.radio() → TOSSIM.Radio

.newPacket() → TOSSIM.Packet

.mac() → TOSSIM.Mac

.runNextEvent()

.ticksPerSecond()

.time()

Here are some useful TOSSIM.Tossim functions.

The .getNode() returns a TOSSIM.Mote object. The function has one parameter which is the number of the node.

The .radio() is used to obtain the TOSSIM.Radio and .mac() is used to obtain the TOSSIM.Mac.

The .newPacket() is used to construct a new packet which can then be injected in the radio stack of a node. I'm not going to demonstrate this but there is an example on the wiki page: http://docs.tinyos.net/index.php/TOSSIM

The .runNextEvent() is the one responsible for executing the oldest event in the event queue. We'll see on the next slide how to use it.

As the name indicate the .ticksPerSecond() returns the number of ticks per second. The .time() returns the current virtual time. This is also expressed in ticks.

10 seconds

```
from TOSSIM import *

t = Tossim([])

   ...

while t.time() < 10*t.ticksPerSecond():
    t.runNextEvent()
```

Here is the Python code to simulate 10 seconds.

dbg

Syntax

dbg(*tag, format, arg1, arg2, ...*);

Example

dbg(“Trickle”, “Starting time with time %u.\n”, timerVal);

Python

```
t = Tossim([])
t.addChannel(“Trickle”, sys.stdout)
```

When we use TOSSIM a nice feature is that we can add in the nesC code debug messages. This is done using a function called dbg which is similar with printf but it takes one additional text parameter: a tag. This tag can allow us to selectively enable and disable the debug messages that are recorded. This is done using the a function the addChannel from TOSSIM.Tossim. By default the debug messages are disabled.

Useful Functions

*char**	sim_time_string()
sim_time_t	sim_time()
int	sim_random()
sim_time_t	sim_ticks_per_sec()

typedef *long long int sim_time_t*;

Some other useful functions that available in the nesC code under TOSSIM are the following: sim_time_string() that returns a string with description of the current time (this is useful in dbg messages), sim_time() that returns the time in number of ticks, sim_random that returns a random integer, sim_ticks_per_sec() that returns the number of ticks for a virtual second.

Note that the time is expressed as a 64-bit number.

Radio Model

Closest-fit Pattern Matching (CPM)

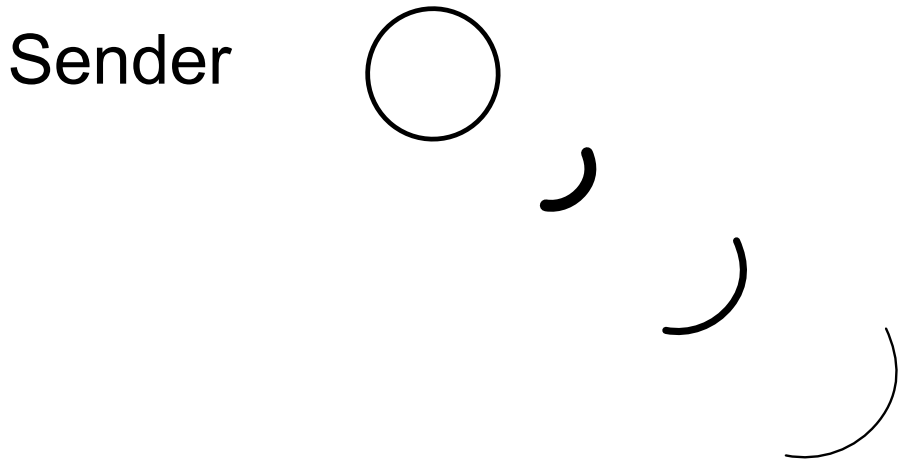
Improving Wireless Simulation Through Noise Modeling

HyungJune Lee, Alberto Cerpa, and Philip Levis

IPSN 2007

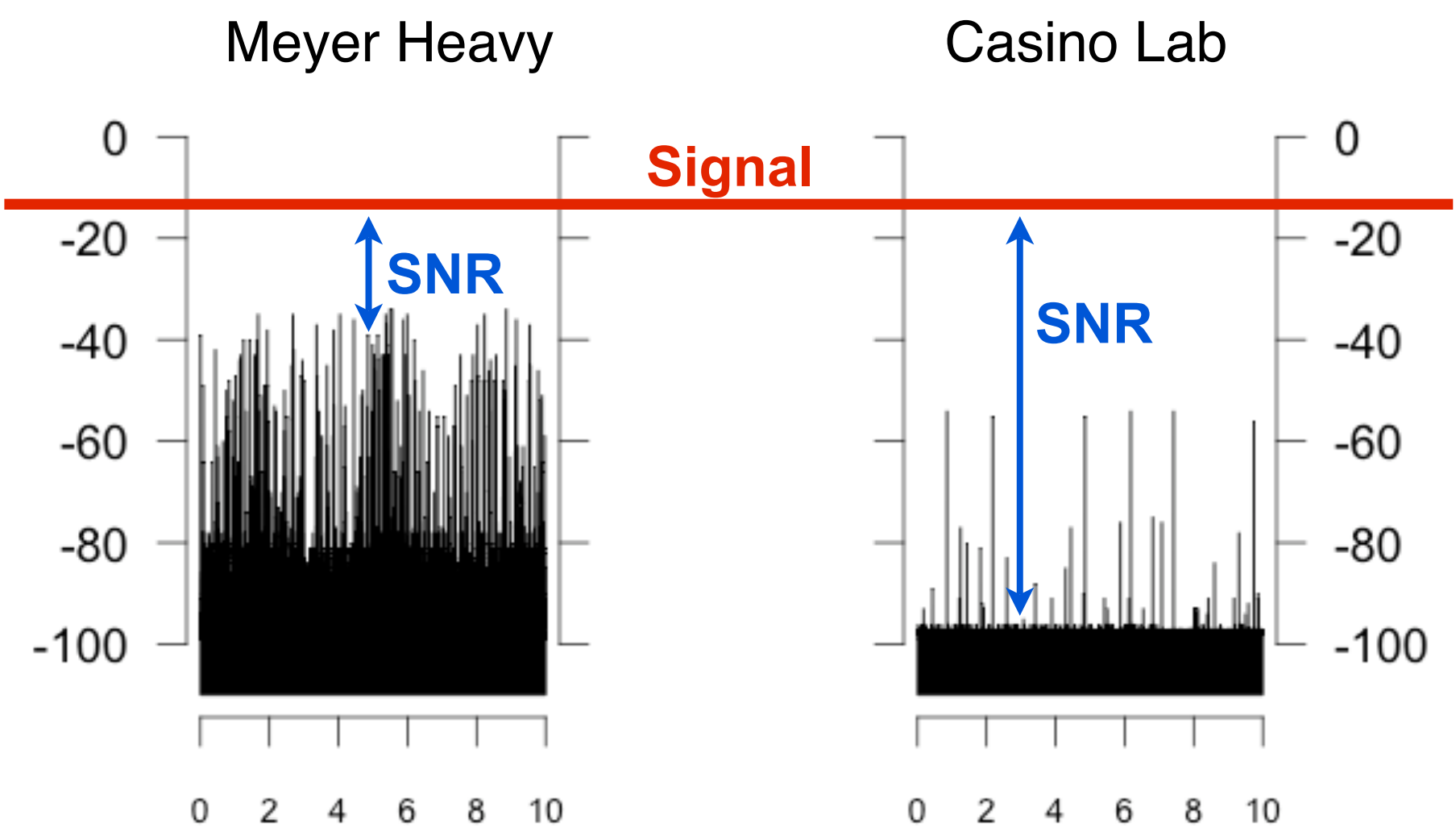
Let’s now talk a little about the radio model. As I said, the most important role is played by CPM, a way to model the noise that was published in IPSN 2007.

Radio Model



Let’s start with a sender and a receiver. The packet leaves from sender with a certain strength (1mW, 0 dBm for CC2420). By the time it reaches the receiver the strength is much lower. The receiver is also surrounded by noise.

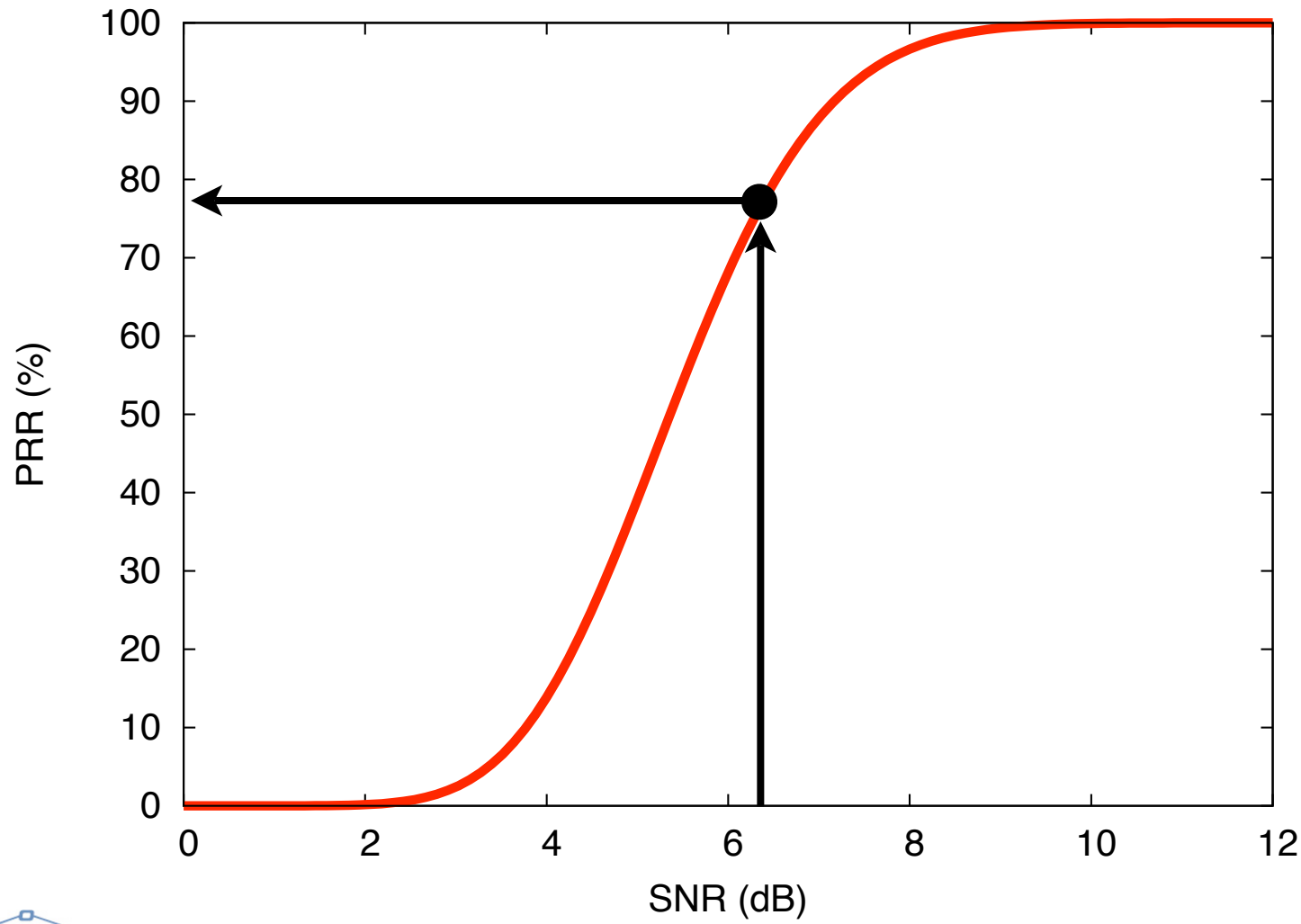
Noise Level



Here are two example of how the noise looks like. The trace from the left is shows the very noisy environment from the Meyer library from Stanford while the more quiet one from right is from the Casino Lab from Colorado Schools of Mines.

The strength of the signal is represented by the red line and the distance between it and the noise is what we called SNR, Signal to Noise Ratio.

CC2420 SNR/PRR



The Packet Reception Rate (PRR) of the CC2420 depends on the SNR in the way depicted in this figure. This is an idealized curve and is hard-coded in TOSSIM. We can see when the strength of the packet is approaching the noise floor (SNR is lower) the PRR is going to zero. When the signal is significantly stronger than the noise (higher SNR) then the PRR is 100%.

TOSSIM decides if a packet is drop in the following ways: it uses CPMS to find the noise level for the current moment in time, then it computes the SNR and finds the PRR p of the current packet using the SNR/PRR curve. In the last step a random number between 0 and 100 is draw and if the value is less than PRR p then packet is considered received.

TOSSIM.Radio

.add(*source, destination, gain*)

.connected(*source, destination*) → True/False

.gain(*source, destination*)

The most important function from TOSSIM.Radio object is the .add() function which is used to describe the path attenuation between nodes.

Two other handy functions are the .connected() which can indicate if two nodes have a link between them of not and the .gain, which return the value indicated by the .add() function.

TOSSIM.Mote

.bootAtTime(*time*)

.addNoiseTraceReading(*noise*)

.createNoiseModel()

.isOn() → True/False

.turnOn()/.turnOff()

The TOSSIM.Mote objects are used to indicate when the mote is booted using the .bootAtTime() function. The time is expressed in ticks. In order to avoid unreal behavior the nodes should always be started at different times.

This object is also used to describe the noise around the mote. This is done using two functions: .addNoiseTraceReading() and .createNoiseModel(). The first one is used to feed to the model a particular noise trace. The number of samples needs to be at least 100. The second function instructs the CPM to finalize the creation of the noise model.

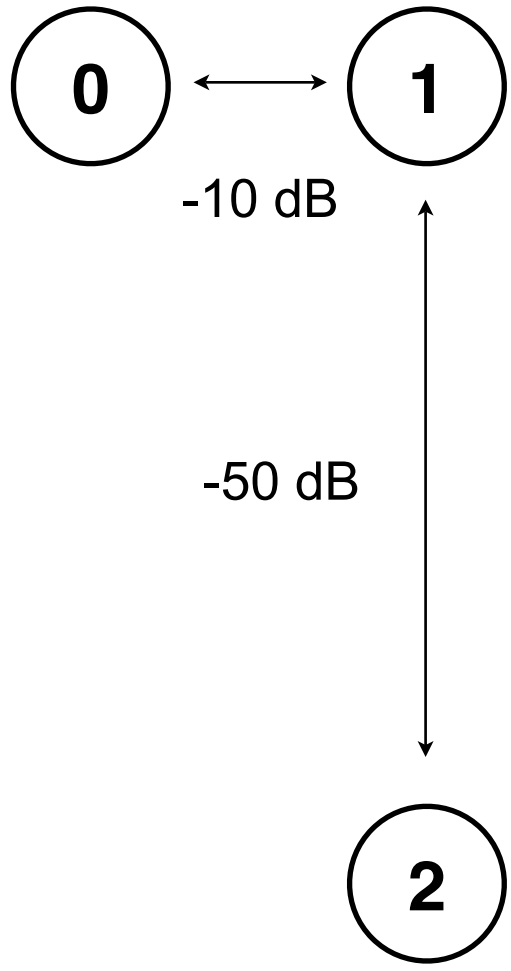
Some other useful functions are: isOn() to find if the mote is up or not and .turnOn() and .turnOff() to force the mote in and out the on state.

Example

```
from TOSSIM import *
t = Tossim([])
r = t.Radio()

mote0 = t.getNode(0)
mote1 = t.getNode(1)
mote2 = t.getNode(2)

r.add(0, 1, -10)
r.add(1, 0, -10)
r.add(1, 2, -50)
r.add(2, 1, -50)
```



Let's now look at an example of how to describe a network of 3 nodes. We only have two links and both are symmetric. The one between mote 0 and mote 1 is short and the attenuation is only -10 dB; for the one between 1 and 2 is -50 dB.

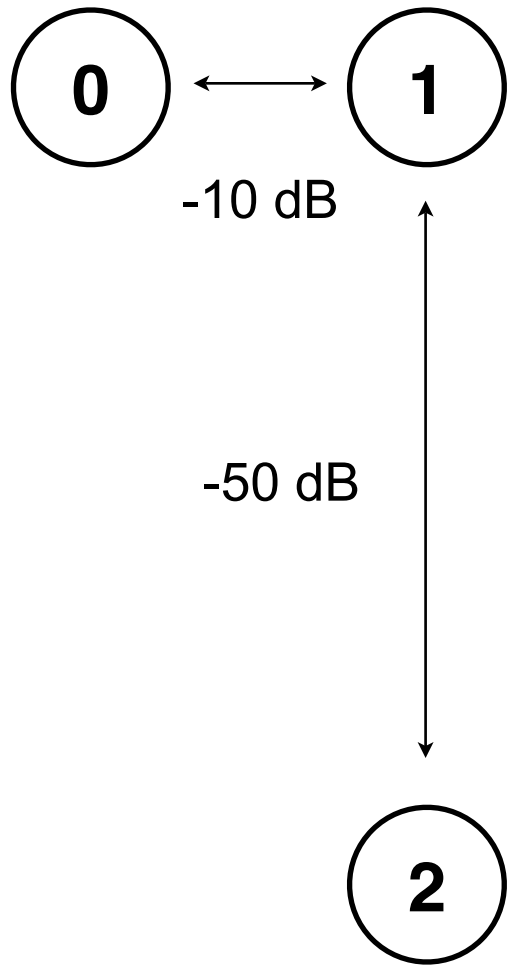
The first we do in our script is to import everything from the TOSSIM module (the TOSSIM.py file I mentioned in the building process). Then be construct a Tossim object called t and use it to get the radio in a variable called r.

What we do next is to construct three Mote objects using the .getNode function. Then we construct the two links using r.

Example (cont)

```
noise = file("meyer-short.txt")
lines = noise.readlines()
for line in lines:
  str = line.strip()
  if (str != ""):
    val = int(str)
    for m in [mote0, mote1, mote2]:
      m.addNoiseTraceReading(val)

for m in [mote0, mote1, mote2]:
    m.createNoiseModel()
```



Now we are going to read a noise trace from the meyer-short.txt file and feed it to all the three motes. Each line of the noise trace file contains one noise reading.

The last thing we need to do is to call the .createNoiseModel function for all the motes.

After this a similar loop as the one presented in the “10 seconds” slide can be used.

Other Features

- Injecting packets
- Inspecting internal variables
- C++ interface
- Debuging using gdb

Here are some other features that I didn't have time to talk about.

Improvements

- **TossimLive**
  - SerialActiveMessageC
- **CC2420sim**
  - Multiple channels
  - PacketLink
  - CC2420Packet: .getRSSI(), .getLQI()
  - ReadRssi()
  - Flash support

The TossimLive is an enhancement that add support for SerialActiveMessageC, the serial port used by a mote to talk to the PC (when is connected). The code for this is already in the CVS.

The CC2420sim is replacing the generic radio used by TOSSIM with a simulation of the CC2420, the 802.15.4 radio from the MicaZ mote. Beside this support for flash is also added. This code is not yet in CVS and is maintain in a cc2420sim heads from http://hinrg.cs.jhu.edu/git/.

Future

Parametrized the PRR/SNR curve based on packet size *(in progress)*

Support for multiple binary images *(harder)*

Here are a few things about the future. First the PRR/SNR curve can be improved by taking in consideration the packet size. This is quite easy to do and we are going to have it available soon.

A more complicated thing that could be fix is the fact that TOSSIM can only run one application at a time. This means that in order to simulate two applications they need to first be merge into one. This is cumbersome when the same codebase is used for both simulation and real deployment. We are investigating ways to solve this but it will take some time.

Next

Safe TinyOS