IOT EMULATION WITH COOJA

BA BAGULA & ZENVILLE ERASMUS

ISAT LABORATORY

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF THE WESTERN CAPE (UWC)

CAPE TOWN – SOUTH AFRICA

ICTP-IoT Workshop, Trieste 16-27 March 2015

Outline

- What is Cooja
  - Emulator vs Simulator
  - Main steps
  - Hello-world
- More examples
  - UDP-RPL/broadcast
  - Z1 sensors
  - Sense/Send/Blink
  - UDP-RPL/Unicast
- Energy monitoring
  - Timeline
  - Energest
  - Powertrace
  - PowerTracker
- Networking protocols
  - RPL
  - LIBP
  - Multi-sink

Cooja is an emulator

- According to different sources, an emulator is:
  - a hardware or software system that enables one computer system (called the *host*) to *behave like another* computer system (called the *guest*): e.g. Cooja enabling your laptop to behave like a Z1 mote.
  - a system that typically *enables* the host system *to run software or use peripheral devices* designed for the guest system: e.g. Cooja enabling your laptop to run the RPL protocol, LIBP and/or other IoT protocols of interest .

Emulator

Cooja is an emulator

- According to different sources, an emulator is:
  - a system that *behaves exactly like* the guest system, and abides by all of the rules of the system being emulated, but operating in a different environment to the environment of the original emulated system.
  - a *complete replication* of the guest system, right down to being binary compatible with the emulated system's inputs and outputs.

Cooja is not a simulator

- According to different sources, a simulator is:
  - a hardware or software that that *enables* one computer system (called the *host*) *to behave like another* computer system (called the *guest*), *but* is implemented in an entirely *different way* : e.g. A flight simulator gives you the feeling of flying an airplane, but you are completely disconnected from the reality of flying the plane, and you can bend or break those rules as you see fit. e.g. Fly an Airbus A380 upside down between London and Sydney without breaking it.

Simulator

Cooja is not a simulator

- According to different sources, a simulator is:
  - a system that *provides the basic behaviour* of a system *but may not* necessarily *abide by all of the rules* of the system being simulated.
  - A system designed to *recreate the operation or behaviour* of the guest system. The underlying principles can be the same as the original or different.

What is Cooja?

- Cooja is a Contiki network emulator
  - An extensible Java-based simulator capable of emulating Tmote Sky (and other) nodes
- The code to be executed by the node is the exact same firmware you may upload to physical nodes
- Allows large and small networks of motes to be simulated
- Motes can be emulated at the hardware level
  - Slower but allows for precise inspection of system behaviour
- Motes can also be emulated at a less detailed level
  - Faster and allows simulation of larger networks

Cooja (continued)

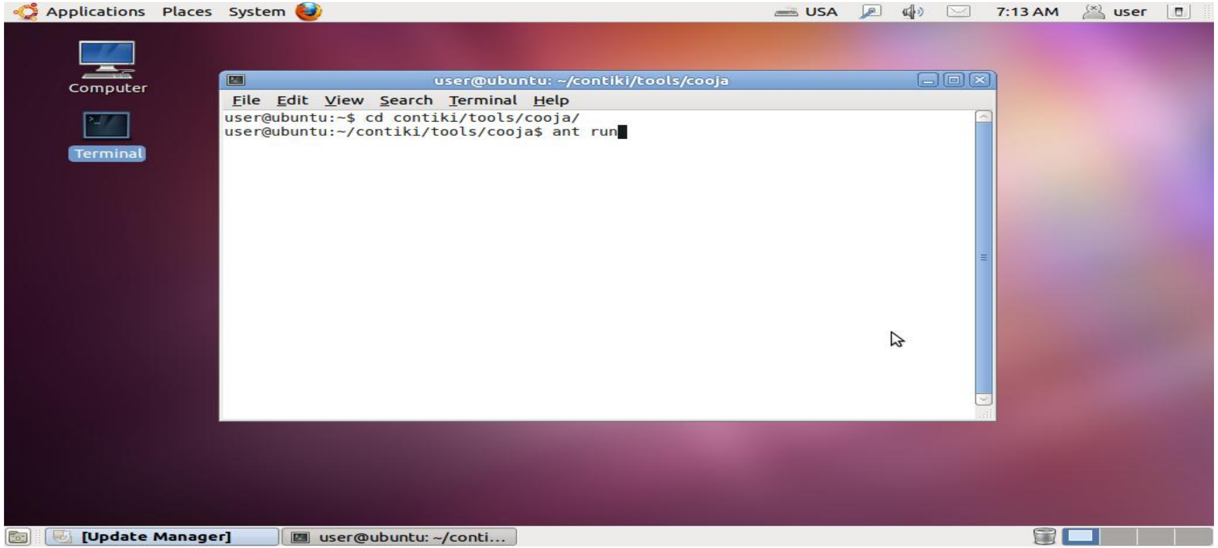
- Cooja is a highly useful tool for Contiki development
  - It allows developers to test their code and systems long before running it on the target hardware
  - Developers regularly set up new simulations to
    - debug their software
    - to verify the behaviour of their systems

Main steps

1. Open a terminal window to start Cooja
2. Create a new simualtion to run Contiki in simulation and wait for Cooja to start and compile itself
3. Set simulation options
4. Create a new mote type
5. Add motes to the simulation
6. Open a terminal Cooja is a highly useful tool for Contiki development
    1. It allows developers to test their code and systems long before
        1. running it on the target hardware
    2. Developers regularly set up new simulations to
        1. debug their software

1. Starting Cooja

- Open a terminal window

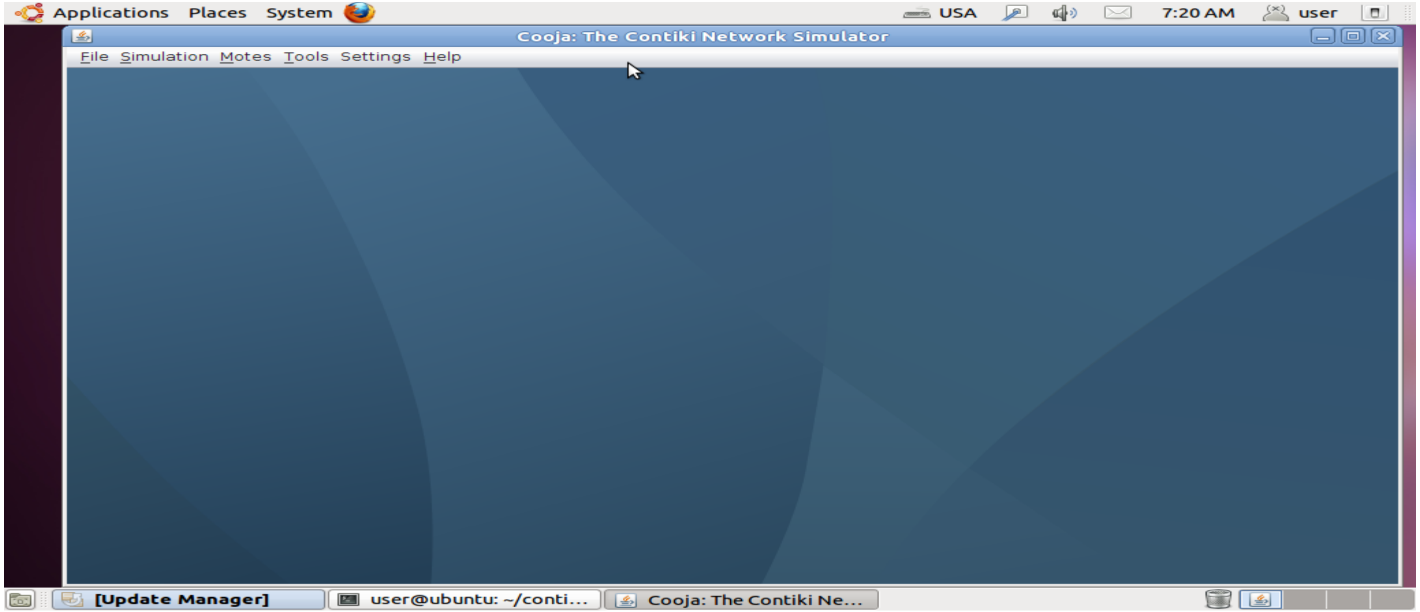


To start Cooja, first **open a terminal window.**

- ✓ **cd contiki/tools/cooja** (**Cooja directory**)
- ✓ start cooja by issuing **ant run**

Waiting for Cooja to start

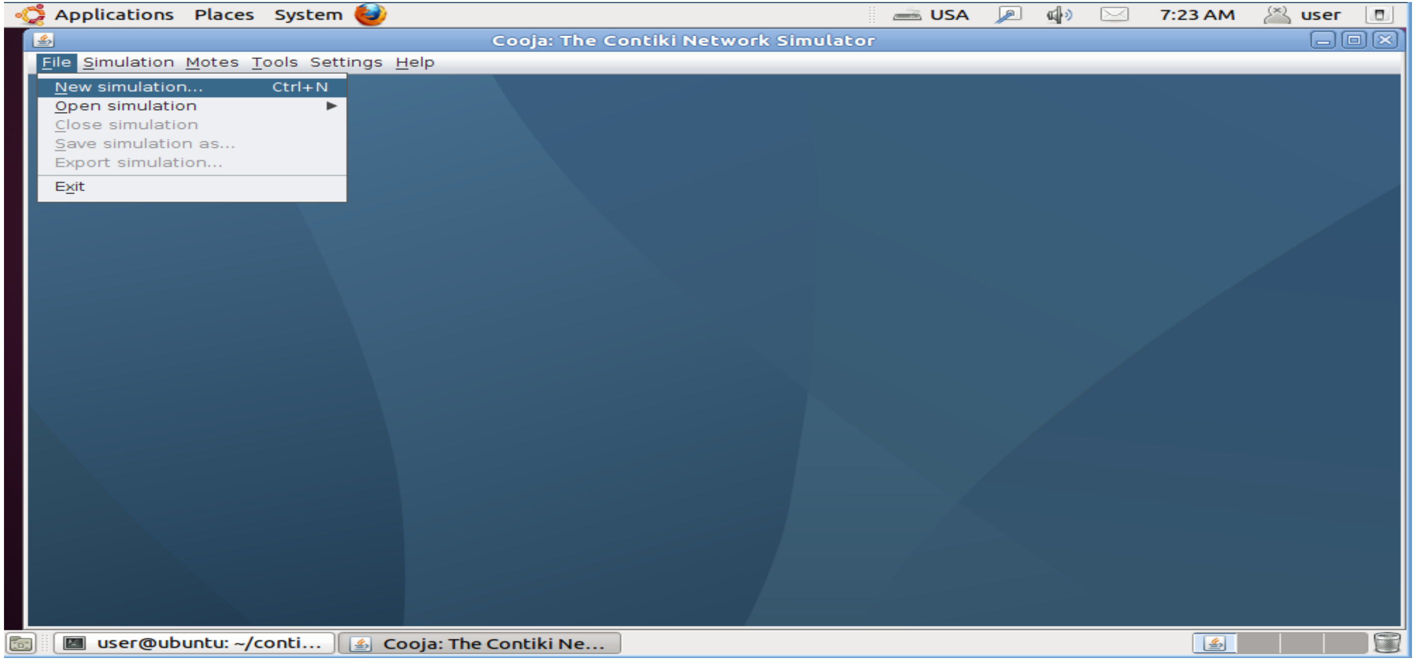
- When Cooja first starts, it will compile itself. This may take some time



When Cooja is compiled, it will start with a blue empty window.

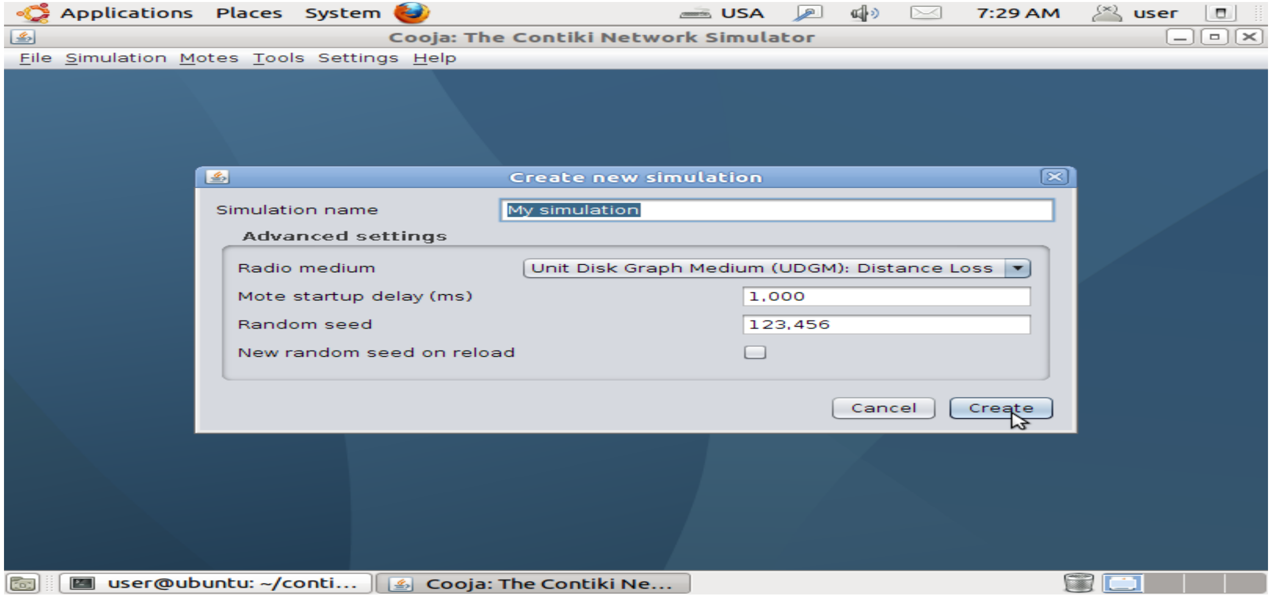
2. Create a new simulation

- Click the **File** menu and click **New simulation...**



3. Set simulation options

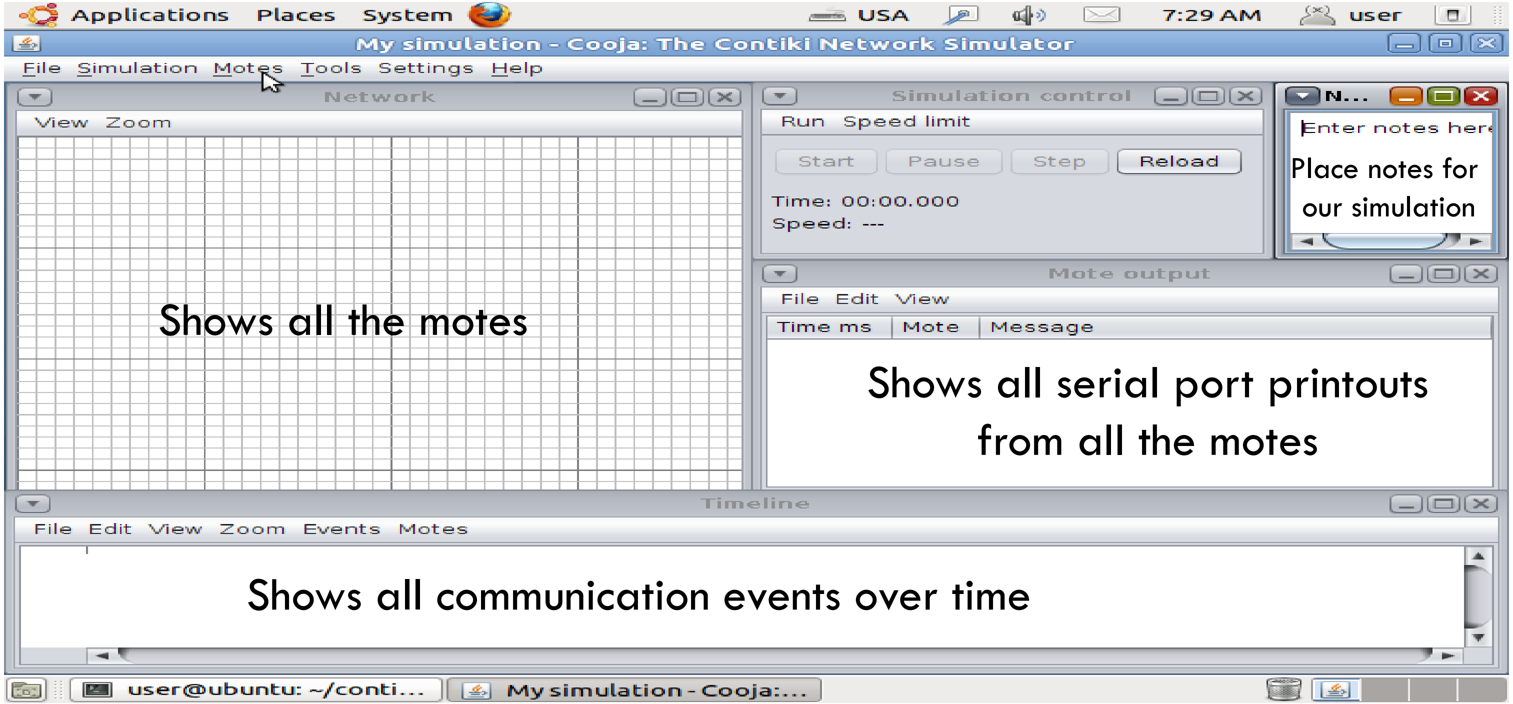
- Cooja now opens up the **Create new simulation** dialog.
  Either change the dialog name or stick with My simulation.



- ✓ Click the **Create** button.

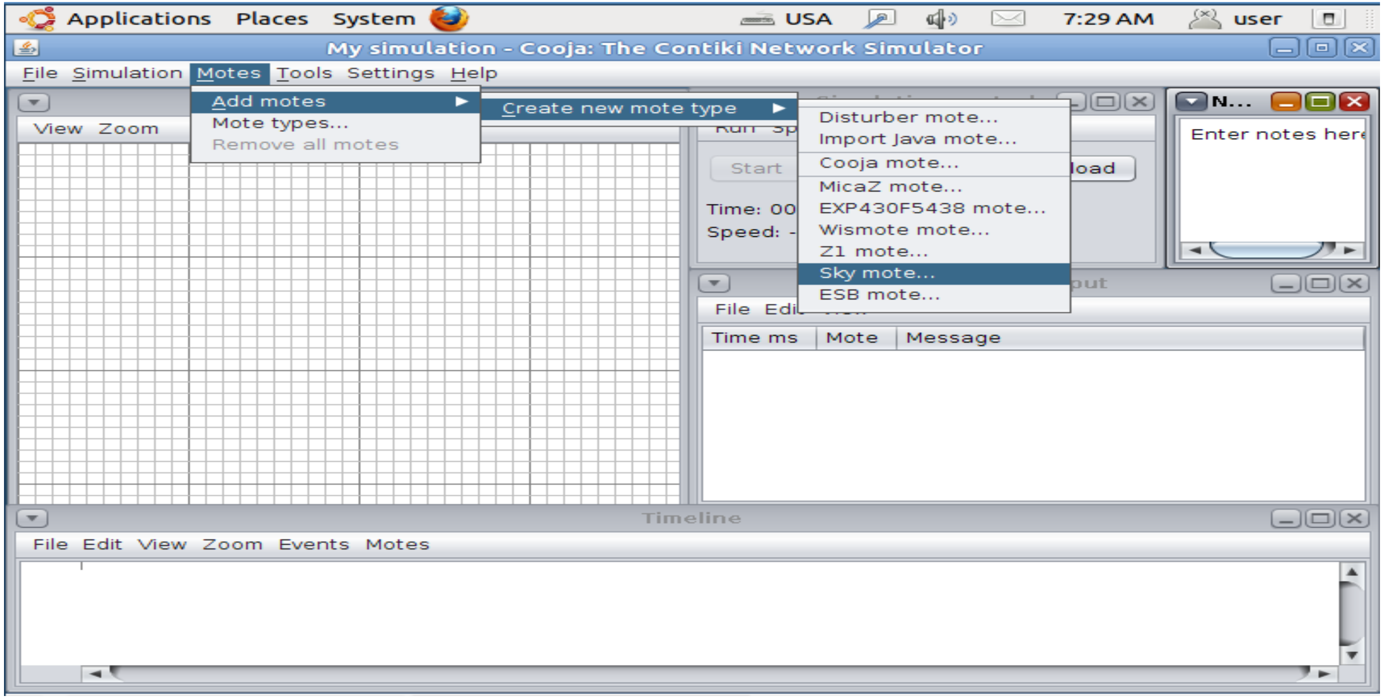
Simulation windows

- Cooja brings up the new simulation.



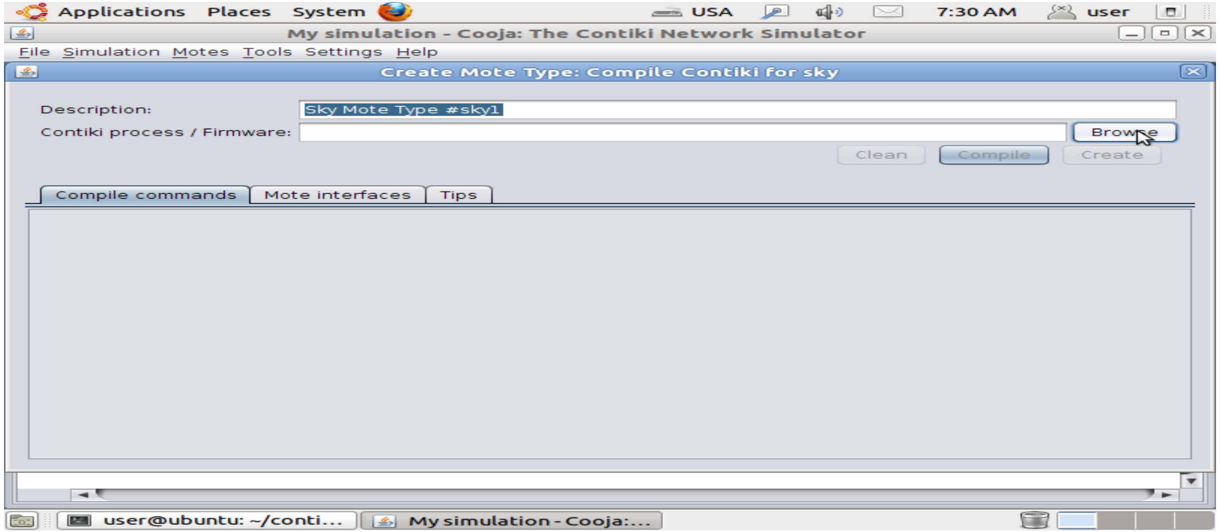
4. Add motes to the simulation

- Add motes



Create a new mote type

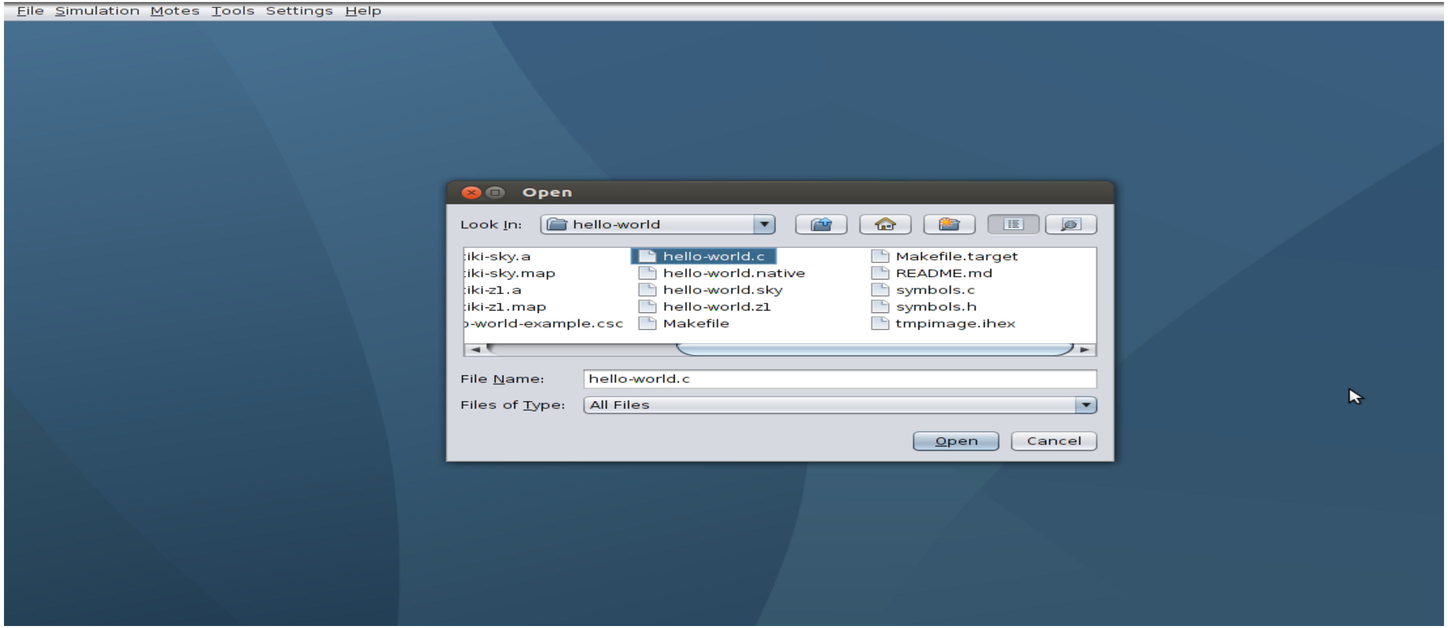
- Cooja opens up the **Create Mote Type** dialog



- ✓ choose a name for our mote type
- ✓ choose the Contiki application that our mote type will run

5. Find Contiki Application

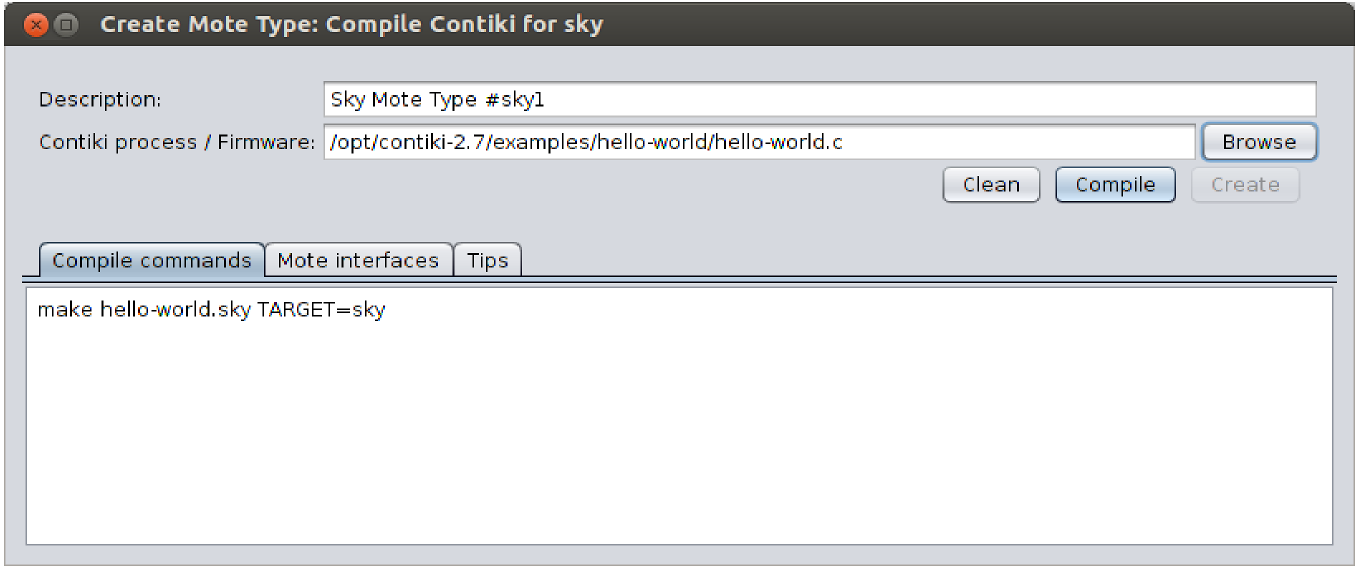
- Hello World /opt/contiki-2.7/examples/hello-world



✓ Specify application C source file → **Open**

6. Compile the Contiki application

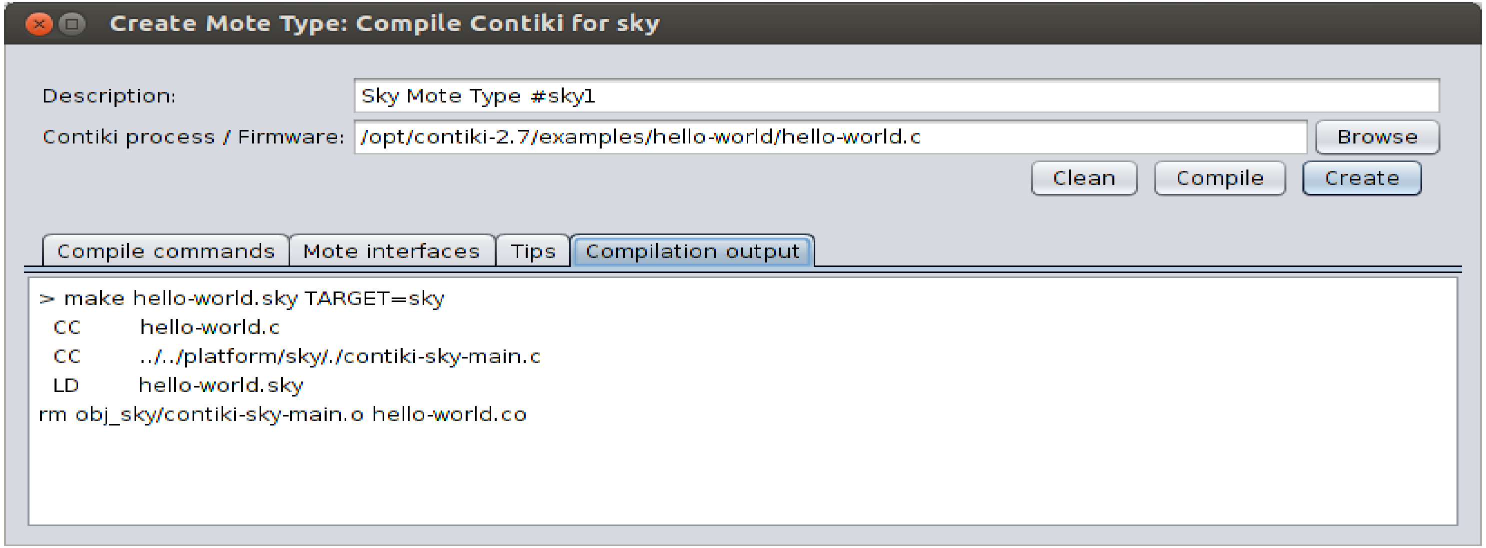
- Cooja will verify that the selected Contiki application compiles for the platform that we have selected



- ✓ Click on the **Compile** button. This will take some time…
- ✓ Compilation output will show up in the bottom white panel.

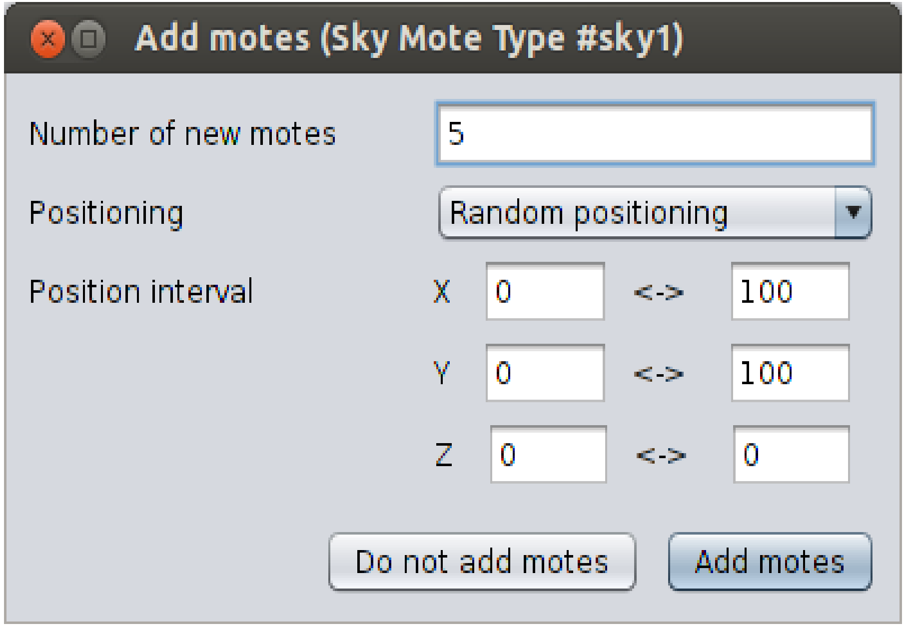
7. Create the mote type

- Click on the **Create** button to create the mote type. The window will close.



8. Add motes to simulation

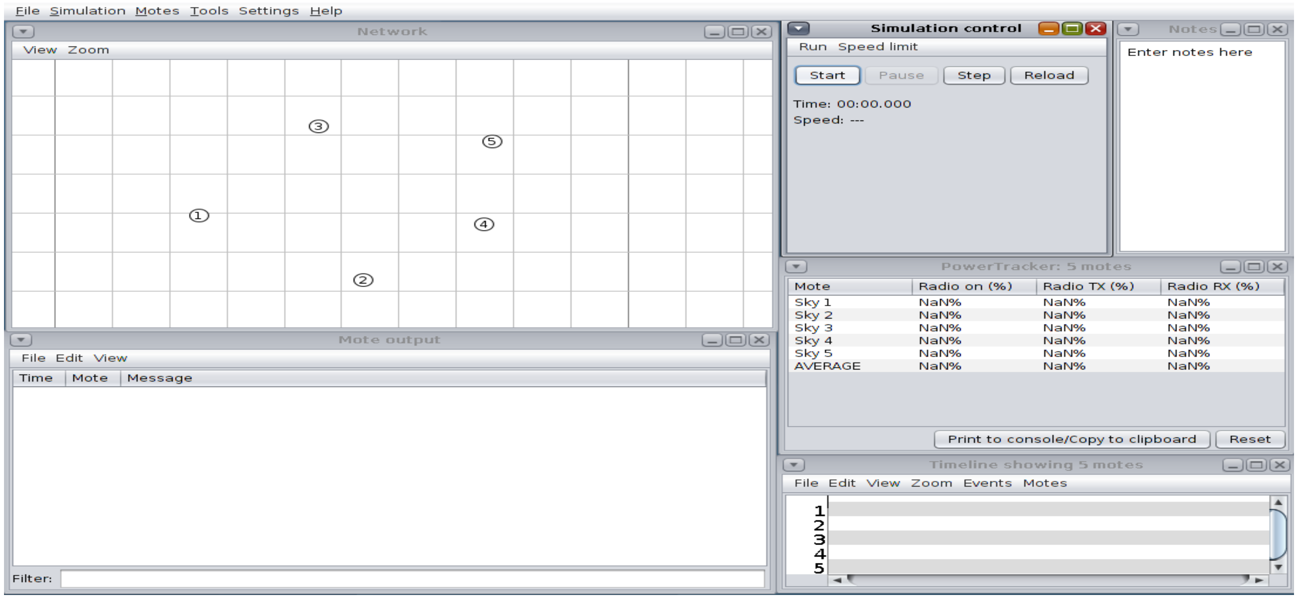
- Add motes by changing the number of motes in the **Number of motes** field to 5.



✓ Click on **Add motes** to add motes to the simulation

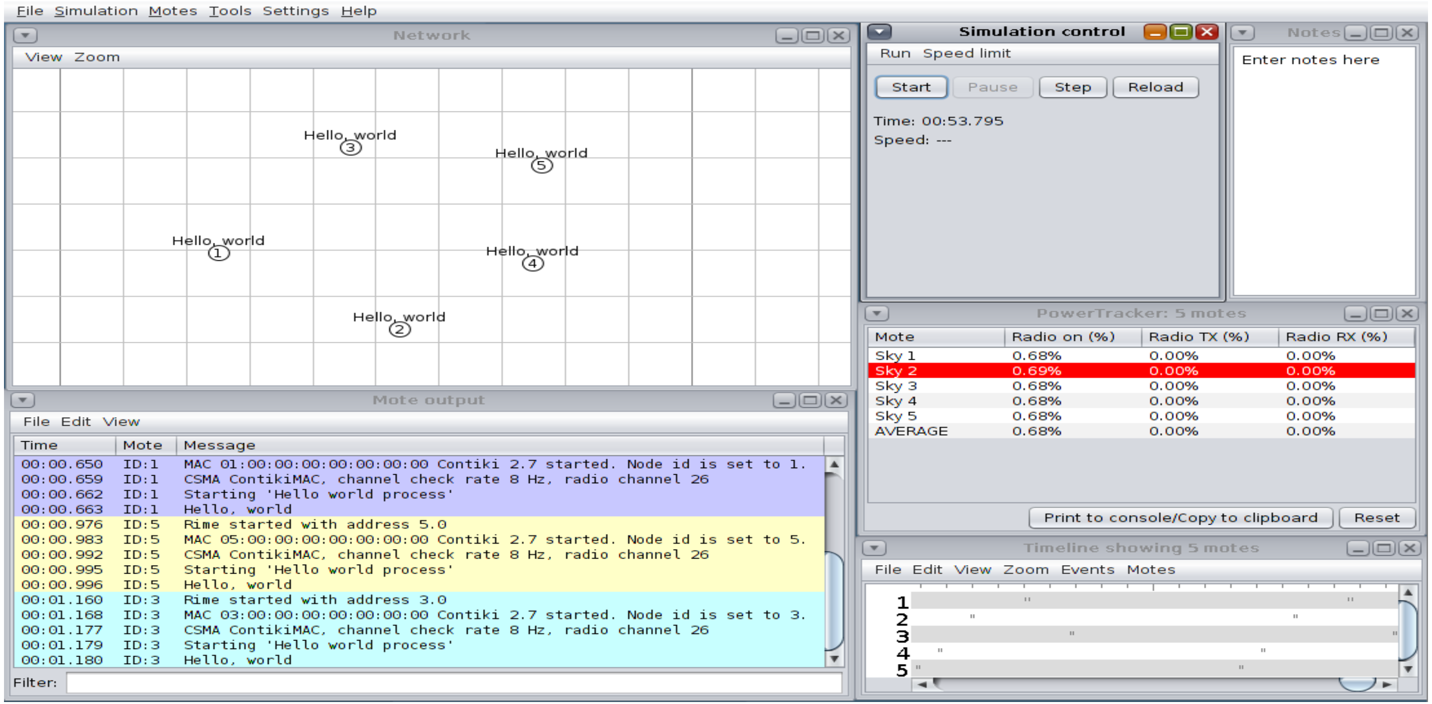
9. Start the simulation

- The 5 added motes are now seen in the simulation window.



✓ Click the **Start** button to start the simulation.

10. Pause the simulation



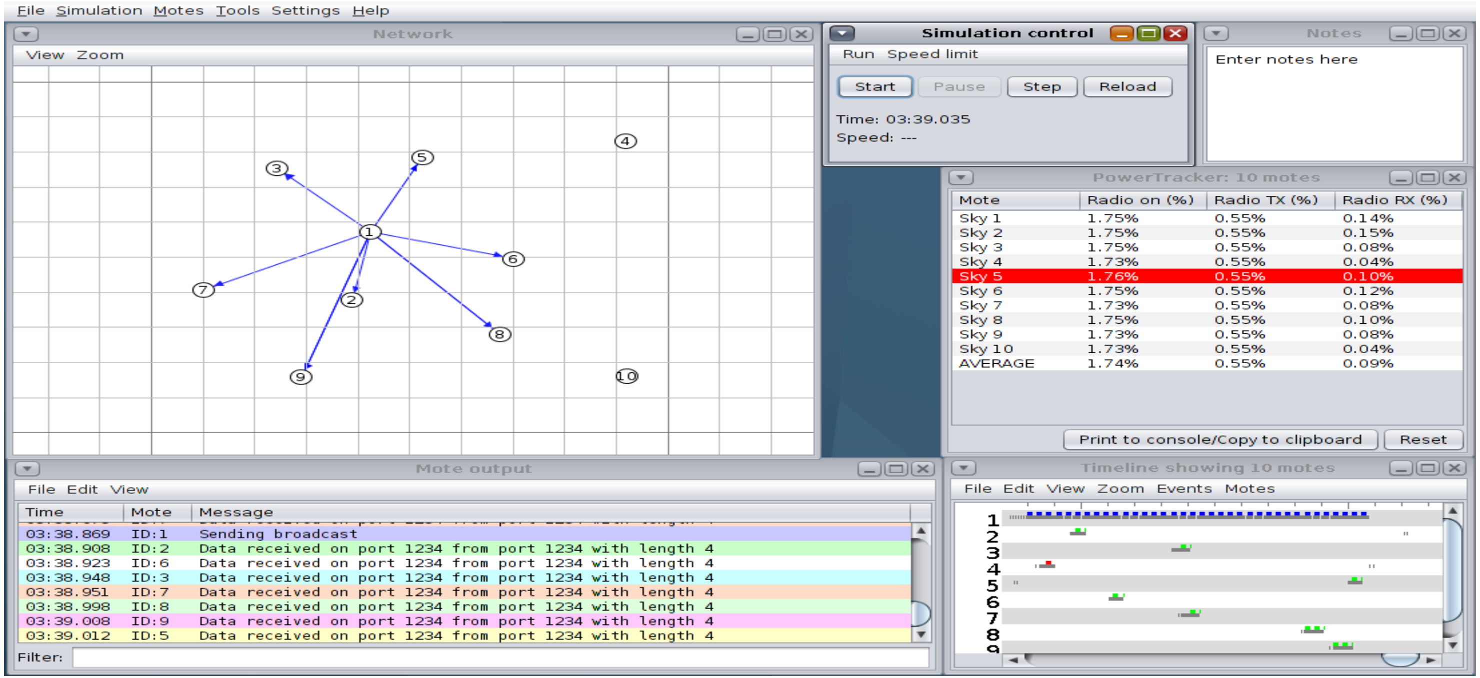
- ✓ **View** → Select Log output: printf()'s

11. Get some statistics

- Mote output window
  - Printouts from the simulated motes
- Network window
  - Shows ongoing network communication
- Timeline
  - Shows communication and radio events over time
  - The small gray lines are ContikiMAC periodically waking up the radio
- Pause
  - Click the **Pause** button to pause the simulation

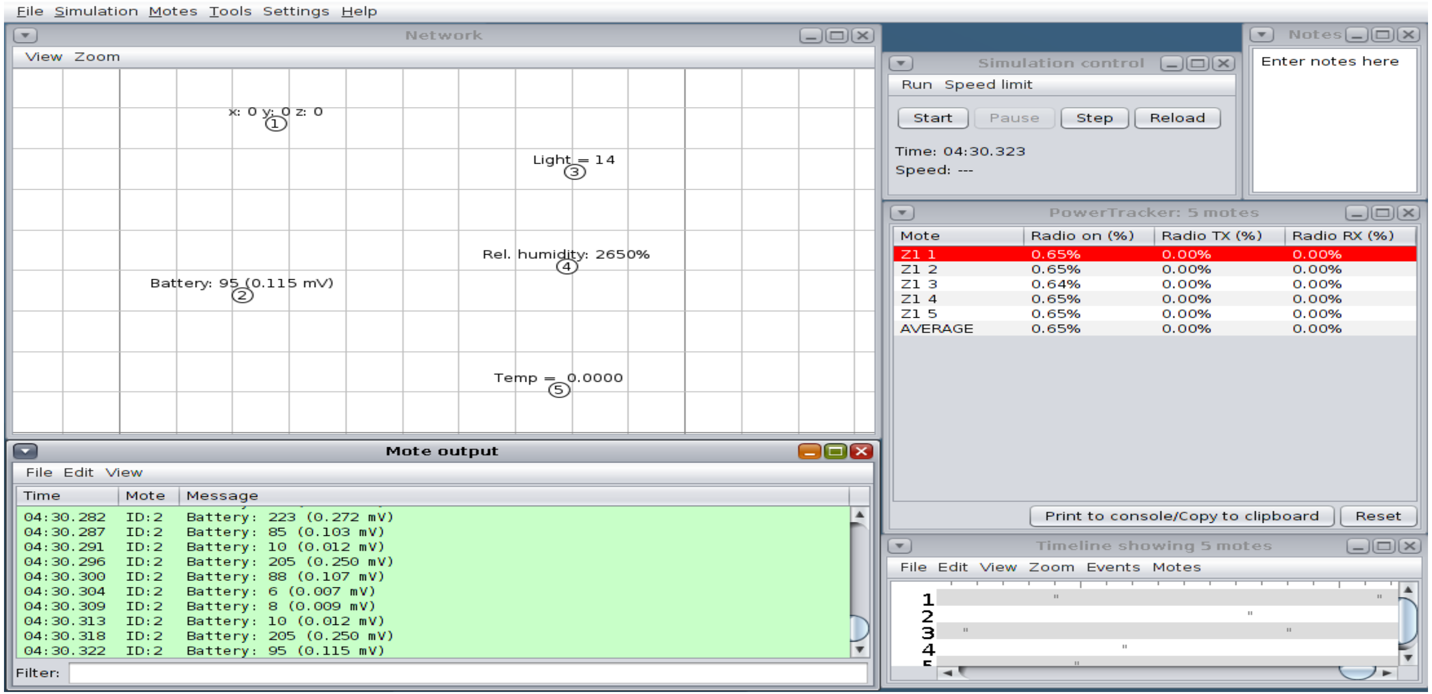
More examples

- simple-udp-rpl/broadcast-example.c



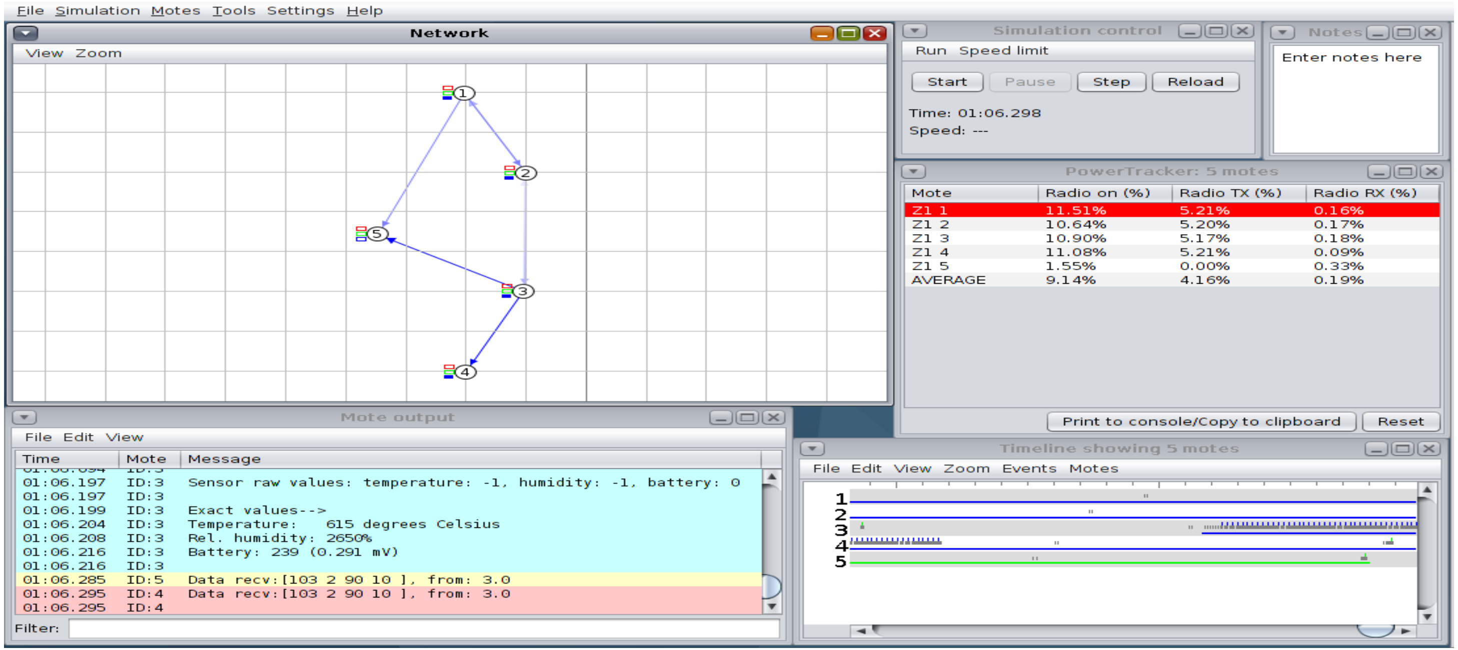
More examples

- Z1 sensors



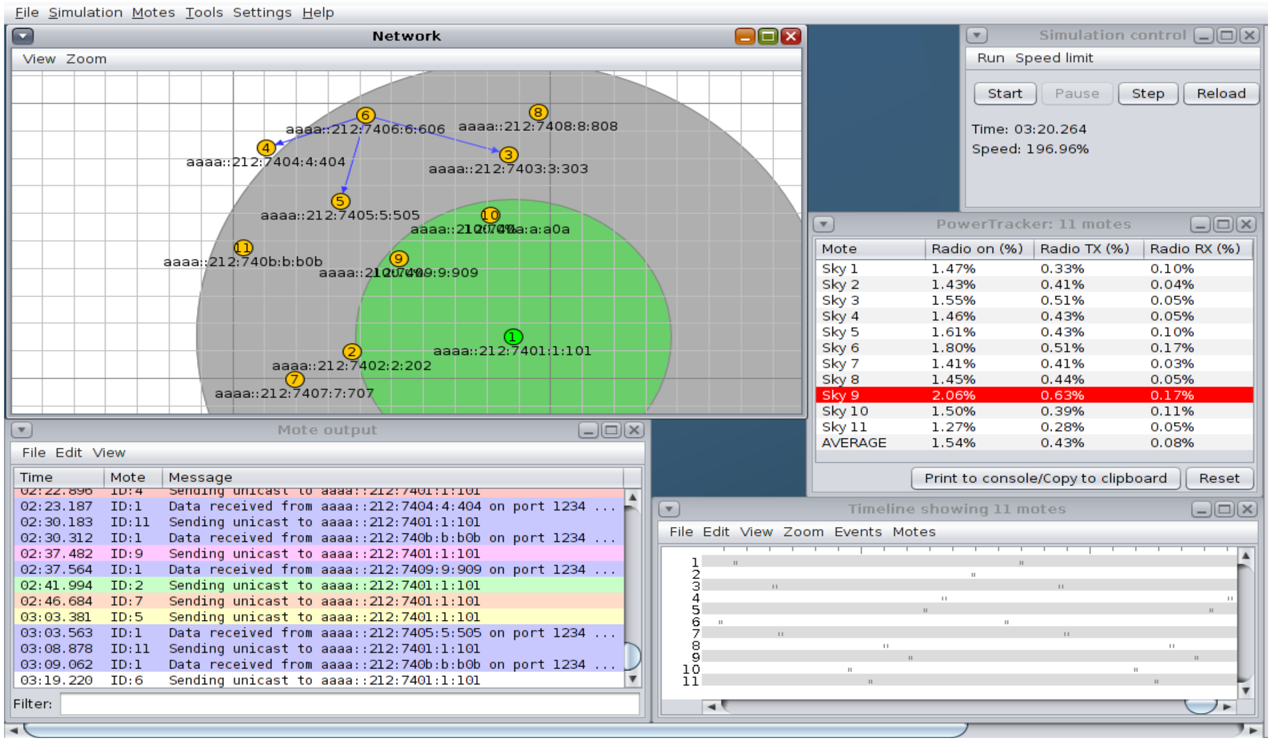
More examples

- Sense, send and blink with receive and blink



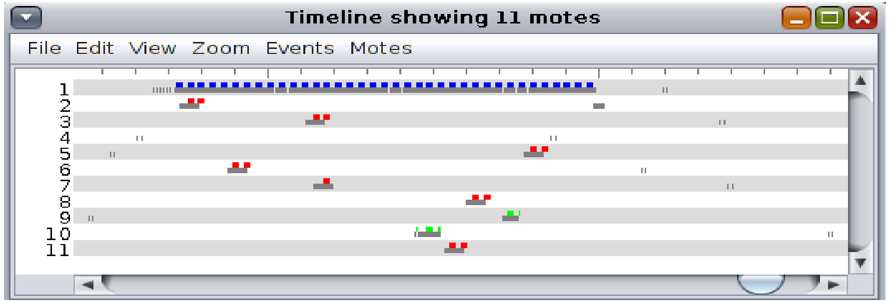
More examples

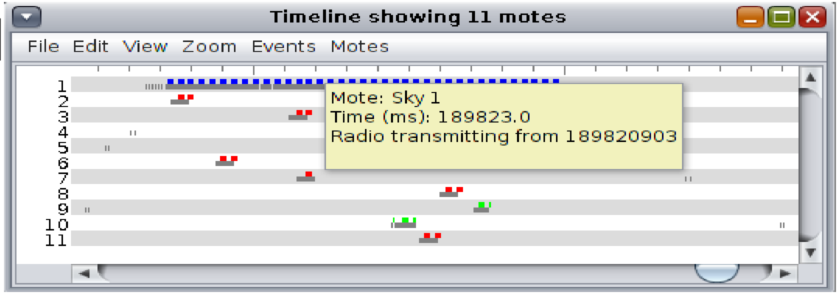
- unicast-example.c
  - ipv6
  - simple-udp-rpl



Timeline in COOJA

- Radio ON/OFF
  - No colour: radio off
  - Grey: radio on
- Radio RX/TX
  - Green: received a packet
  - Blue: packet sent
  - Red: interfered radio (collisions etc.)
- Right-clicking will reveal additional info.[2]



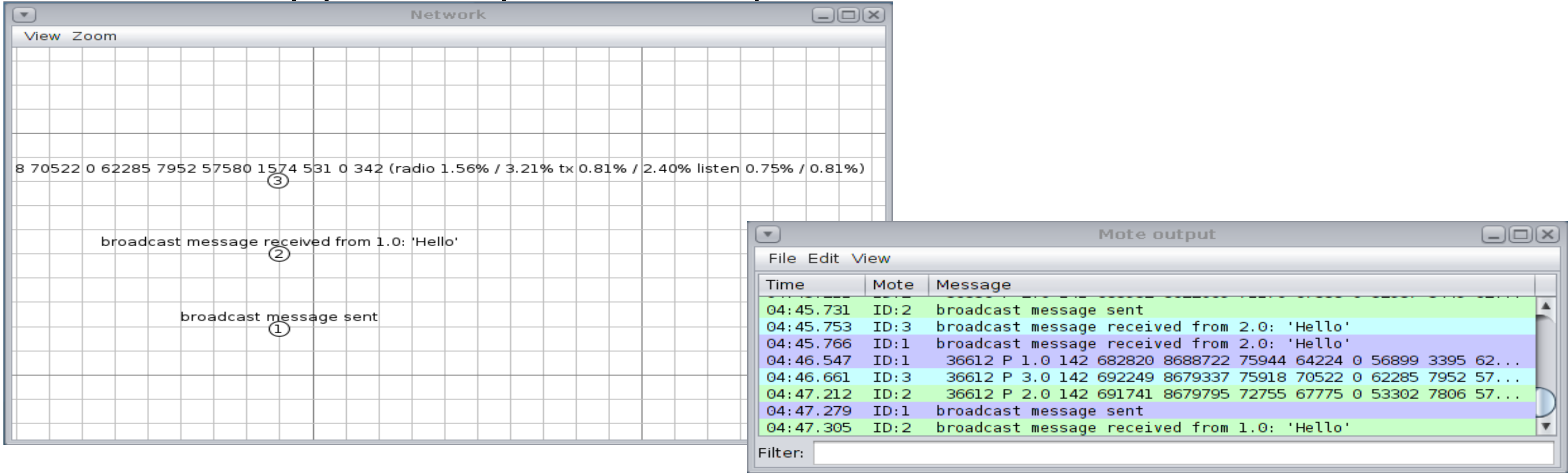


Measure Power Consumption with Energest

- Can be used for obtaining per-component power consumption on Contiki.
  - (cpu_ON, LPM, TX, RX)
  - i.e. the time the radio was in RX mode (rxon)
- For RX:
  - Power(mW) = (rxend – rxstart) * 20mA * 3V / 4096 / runtime(seconds)
  - If you do not divide by runtime you get the energy consumption during runtime.

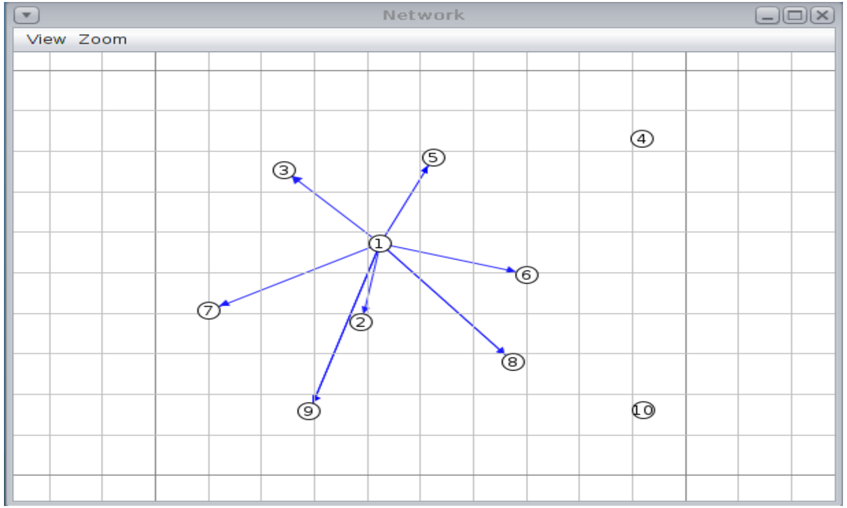
Measure Power Consumption: Powertrace

- Uses Energest along with a periodic difference of the rtimer ticks to get average power over a shorter period of time or for particular network modes[3].
- Periodically prints out power consumption



Measure Power Consumption: PowerTracker

- A COOJA plugin that measures the average simulated radio duty cycles.
  - ✓ simple-udp-rpl/broadcast-example.c



PowerTracker: 10 motes

Mote	Radio on (%)	Radio TX (%)	Radio RX (%)
Sky 1	1.75%	0.55%	0.14%
Sky 2	1.75%	0.55%	0.15%
Sky 3	1.75%	0.55%	0.08%
Sky 4	1.73%	0.55%	0.04%
Sky 5	1.76%	0.55%	0.10%
Sky 6	1.75%	0.55%	0.12%
Sky 7	1.73%	0.55%	0.08%
Sky 8	1.75%	0.55%	0.10%
Sky 9	1.73%	0.55%	0.08%
Sky 10	1.73%	0.55%	0.04%
AVERAGE	1.74%	0.55%	0.09%

Print to console/Copy to clipboard | Reset

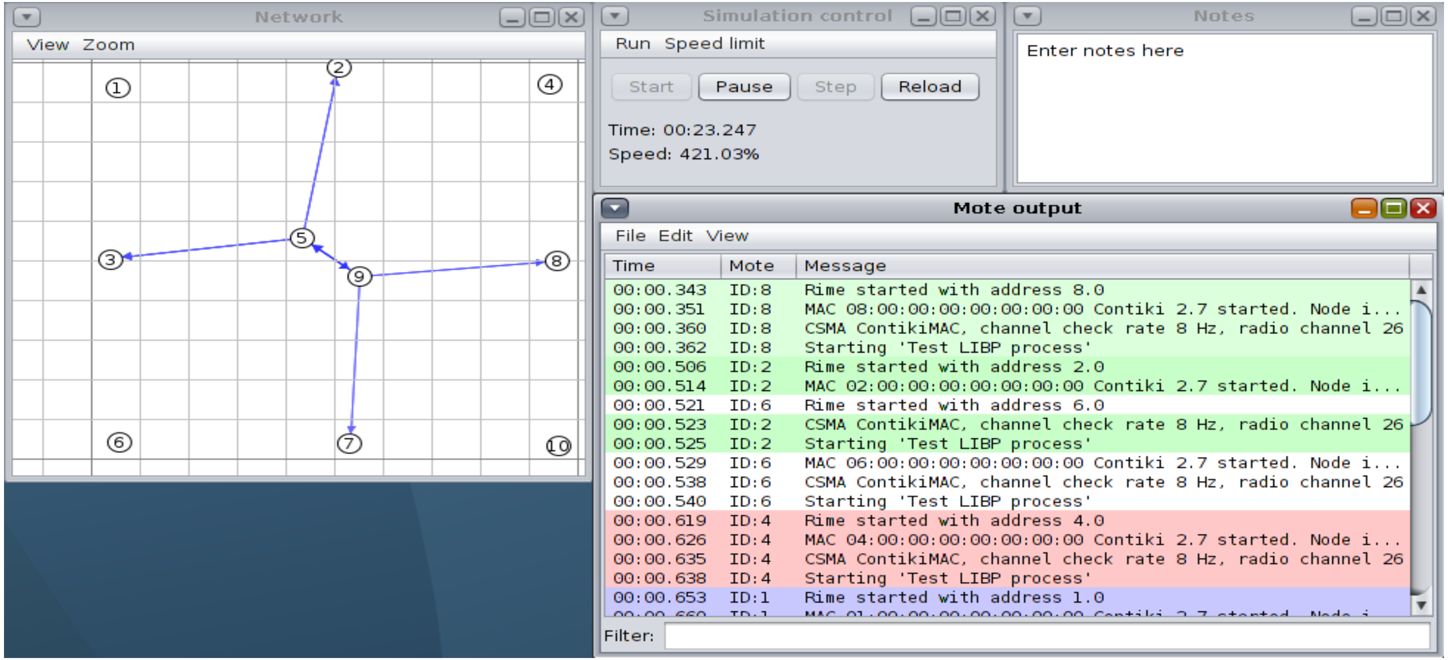
Network Protocols

- COOJA has 2 stacks: uIP and Rime
- Protocol stacks may be interconnected
  - uIP data can be transmitted over Rime and vice versa
- Cooja can be used to emulate network protocols:
  - RPL
  - LIBP

Introduction to LIBP

- LIBP, known as the **Least Interference Beaconing Protocol**, is the implementation of the **Least Interference Beaconing Algorithm, LIBA.**
- LIBP extends the beaconing process widely used by collection protocols with load balancing to improve the Ubiquitous Sensor Network (USN) energy efficiency[4].
- The process involving the least interference paradigm allows the selection of a parent node that has the smallest number of children. This is a point of least traffic flow interference.
- The parent selection model chooses the first parent node heard from, whereby the sensor nodes hear from a set of neighbours and select the least burdened (in number of children) as the parent node.

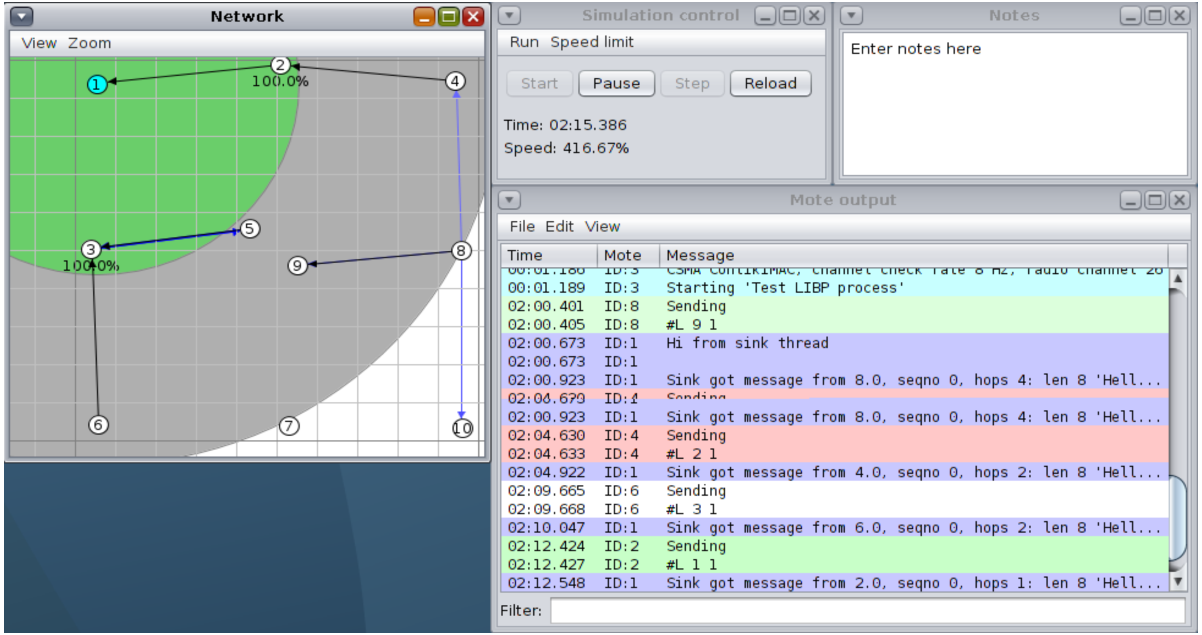
LIBP – Rime startup



Upon network startup, Rime started with address 8.0 for Node ID 8.

The image details the radio channel as 26 and the channel check rate of 8 Hz.

LIBP – sink mote



Sink mote with ID 1 – After 2 minutes, 0 seconds and 673 milliseconds, ID 1 broad-casted to the network that it is sink by sending "**Hi from sink thread**".

LIBP - PowerTracker

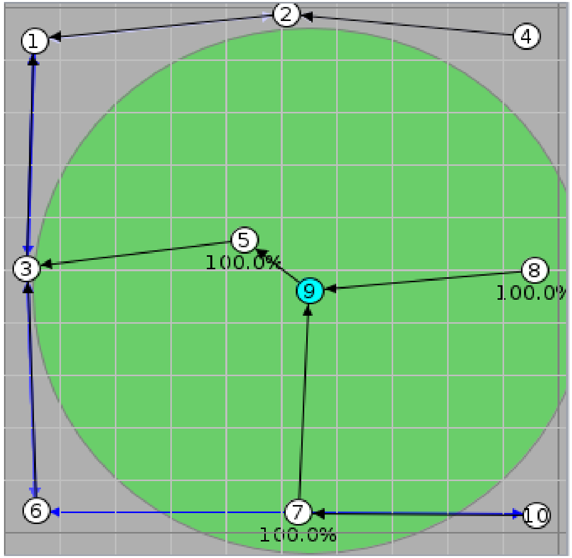
PowerTracker: 10 motes

Mote	Radio on (%)	Radio TX (%)	Radio RX (%)
Sky 1	2.22%	0.62%	0.08%
Sky 2	2.13%	0.60%	0.06%
Sky 3	2.48%	0.73%	0.09%
Sky 4	1.89%	0.53%	0.03%
Sky 5	2.17%	0.60%	0.08%
Sky 6	2.03%	0.52%	0.11%
Sky 7	2.14%	0.59%	0.05%
Sky 8	1.93%	0.54%	0.03%
Sky 9	2.30%	0.68%	0.05%
Sky 10	1.73%	0.47%	0.02%
AVERAGE	2.10%	0.59%	0.06%

PowerTracker after **5 minutes.**

Sky 3 used the most power by being on most of the time.
It's Radio TX is also the highest with a value of **0.73%.**
It has the second highest Radio RX of **0.09%.**

LIBP – Parent with children



Node 3 has 2 children, namely nodes 5 and 6.
These nodes also have children.
Sky 10 used the **least power** with its
Radio on at 1.73%
Radio TX at 0.47% and
Radio RX of 0.02% (least percentage).

Sky 10 is ranked along the bottom of the tree, has no children and is only active when it has to send its data, unlike the other motes.

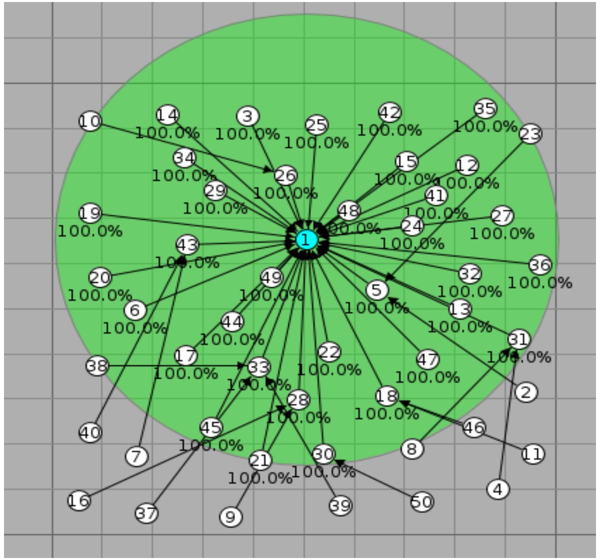
PowerTracker: 10 motes

Mote	Radio on (%)	Radio TX (%)	Radio RX (%)
Sky 1	2.22%	0.62%	0.08%
Sky 2	2.13%	0.60%	0.06%
Sky 3	2.48%	0.73%	0.09%
Sky 4	1.89%	0.53%	0.03%
Sky 5	2.17%	0.60%	0.08%
Sky 6	2.03%	0.52%	0.11%
Sky 7	2.14%	0.59%	0.05%
Sky 8	1.93%	0.54%	0.03%
Sky 9	2.30%	0.68%	0.05%
Sky 10	1.73%	0.47%	0.02%
AVERAGE	2.10%	0.59%	0.06%

LIBP – large network

- Starting COOJA with "ant run" will give you the default Java maximum memory
  - 5 – 10 emulated nodes
- If you use "ant run_bigmem" you will be able to simulate/emulate larger networks.

LIBP – large network



PowerTracker: 50 motes

Mote	Radio on (%)	Radio TX (%)	Radio RX (%)
Contiki 1	100.00%	0.21%	0.37%
Contiki 2	100.00%	0.01%	0.22%
Contiki 3	100.00%	0.01%	0.38%
Contiki 4	100.00%	0.01%	0.17%
Contiki 5	100.00%	0.03%	0.57%
Contiki 6	100.00%	0.01%	0.47%
Contiki 7	100.00%	0.01%	0.19%
Contiki 8	100.00%	0.01%	0.28%
Contiki 9	100.00%	0.01%	0.20%
Contiki 10	100.00%	0.01%	0.28%
Contiki 11	100.00%	0.01%	0.20%
Contiki 12	100.00%	0.01%	0.38%
Contiki 13	100.00%	0.01%	0.53%
Contiki 14	100.00%	0.01%	0.33%
Contiki 15	100.00%	0.01%	0.43%
Contiki 16	100.00%	0.01%	0.14%
Contiki 17	100.00%	0.01%	0.52%
Contiki 18	100.00%	0.04%	0.48%
Contiki 19	100.00%	0.01%	0.36%
Contiki 20	100.00%	0.01%	0.40%

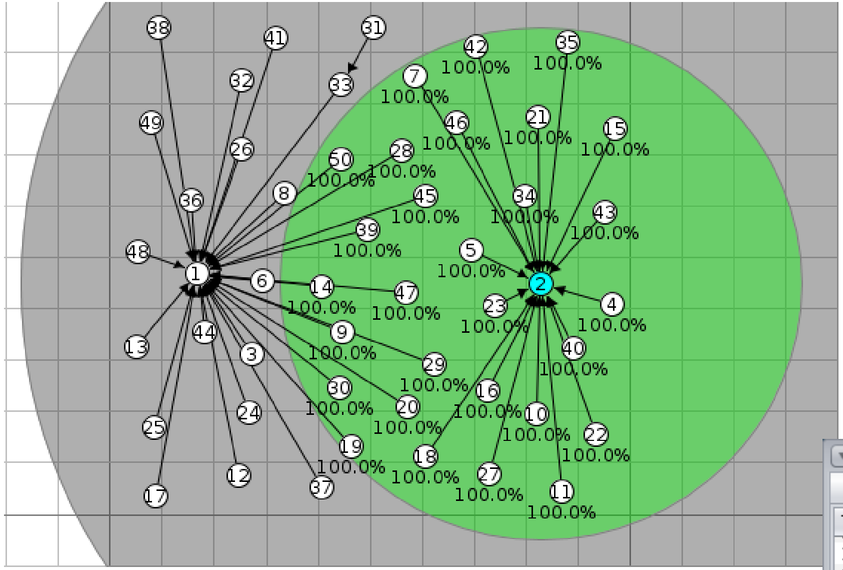
Print to console/Copy to clipboard | Reset

Compiled **50 Cooja Motes**

(Simulated motes – run as native java code)

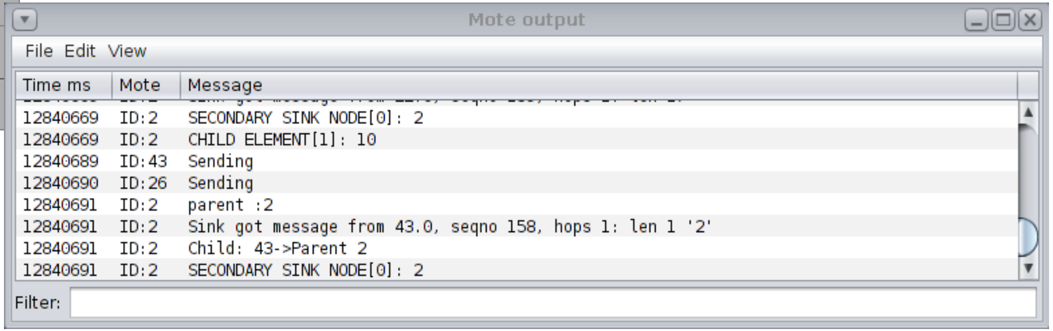
Downside: Cannot do any power profiling

LIBP – 2 sinks



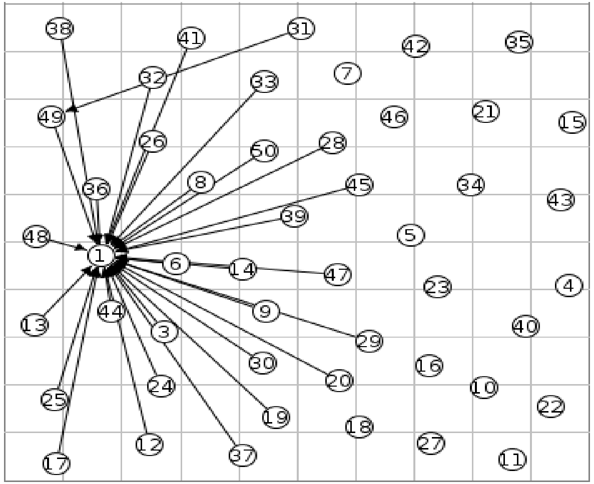
Node ID 1 : Primary Sink
Node ID 2 : Secondary Sink

Load balanced network with 2 sink nodes

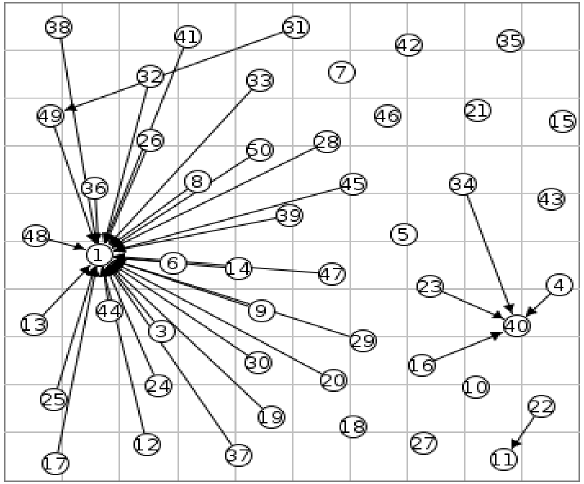


LIBP – 2 sinks

- The 2 sink nodes help with load balancing and network recovery
  - e.g. when one node goes offline, it's children (orphaned nodes) attempt to connect to the other sink

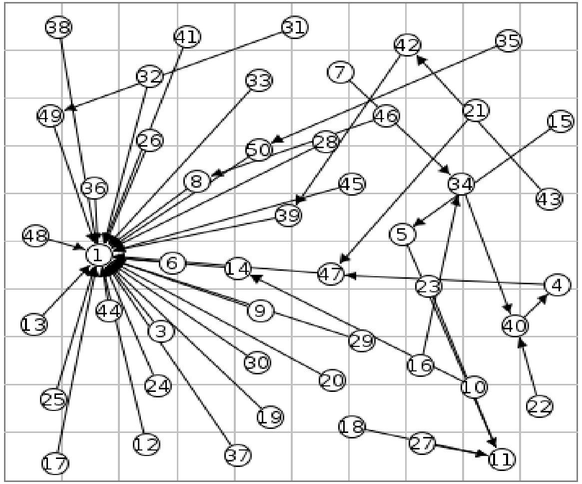


Secondary sink - offline

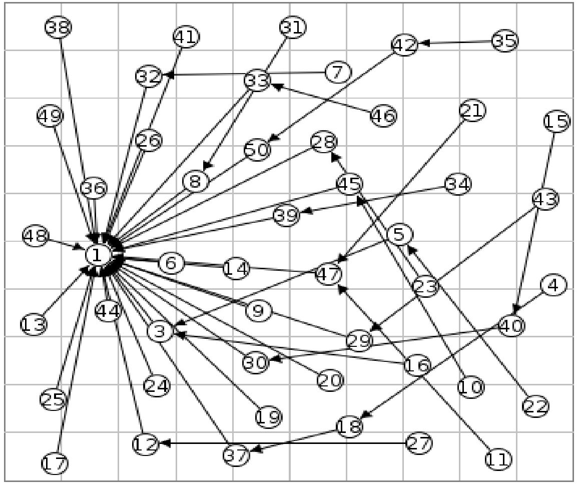


Recovery in process

LIBP – 2 sinks



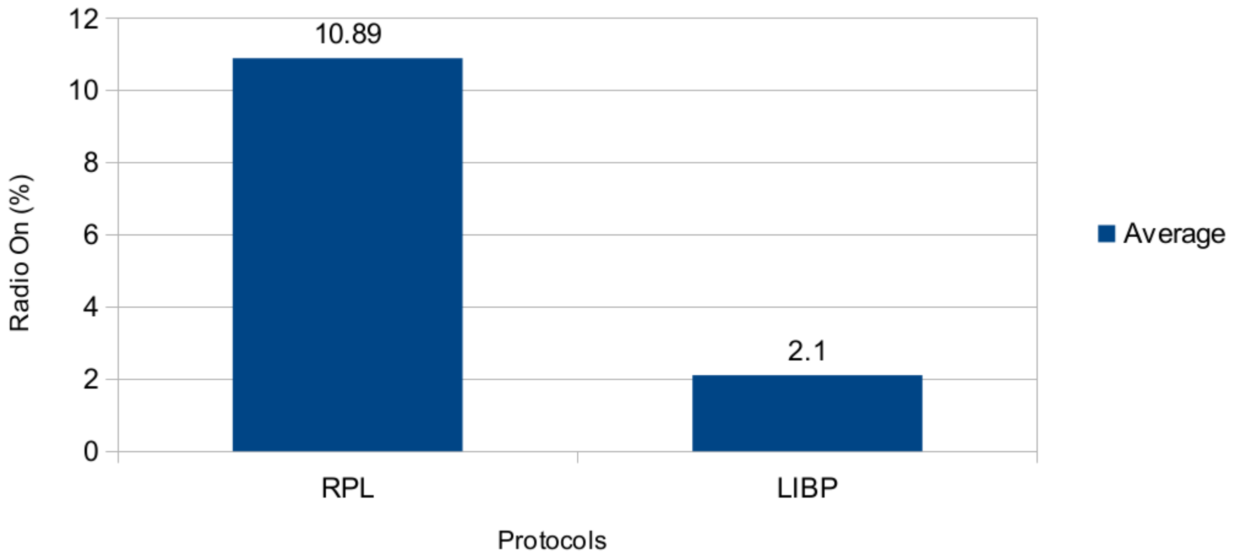
Orphaned nodes also attempt to connect with each other



Network has recovered and every node is making use of Node 1 as its sink node

Recovery in process

Comparison of RPL and LIBP Radio On Averages



LIBP uses less power amongst **10 Skymotes** in relation to keeping their radios on, thus creating a more energy efficient network.

References

[1] Alan, A., & Pritsker, B. (n.d.). Why Simulation Works. *Proceedings of the 1989 Winter Simulation Conference*. Retrieved from http://www.sfu.ca/~vdabbagh/p1-pritsker.pdf

[2] Voigt, T. *Contiki COOJA Crash Course*. Swedish Institute of Computer Science. Retrieved from https://www.sics.se/~thiemo/seniot09cccc-slides.pdf

[3] Kopf, D. *What is Difference Between The Energest and PowerTrace*. Retrieved from http://contiki-developers.narkive.com/VMpBTquh/what-is-difference-between-the-energest-and-powertrace

[4] Bagula, A., Djenouri, D., Karbab, E. *Ubiquitous Sensor Network Management: The Least Interference Beaconing Model*. In Proceedings of PIMRC 2013, Pages 2352-2356, 2013.

[5] Lutando Ngqakaza & Antoine Bagula, "**Least Path Interference Beaconing Protocol (LIBP): A Frugal Routing Protocol for The Internet-of-Things**", in proceedings of the IFIP Wired/Wireless Internet Communications WWIC 2014, Lecture Notes in Computer Science Volume 8458, 2014, pp 148-161, 2014.