Contiki: Open Source OS for the Internet of Things

Adapted by Hongwei Zhang from Materials Online

Why Choose Contiki?

- Contiki is an open source operating system for the Internet of Things.
- Contiki connects tiny low-cost, low-power microcontrollers to the Internet

The Contiki Operating System

- Contiki is an open source operating system that
  - runs on tiny low-power microcontrollers
  - makes it possible to develop applications that make efficient use of the hardware while providing standardized low-power wireless communication for a range of hardware platforms
- Contiki is used in numerous commercial and non-commercial systems
  - city sound monitoring, street lights
  - networked electrical power meters, industrial monitoring
  - radiation monitoring, construction site monitoring
  - alarm systems, remote house monitoring, etc
- http://www.contiki-os.org/

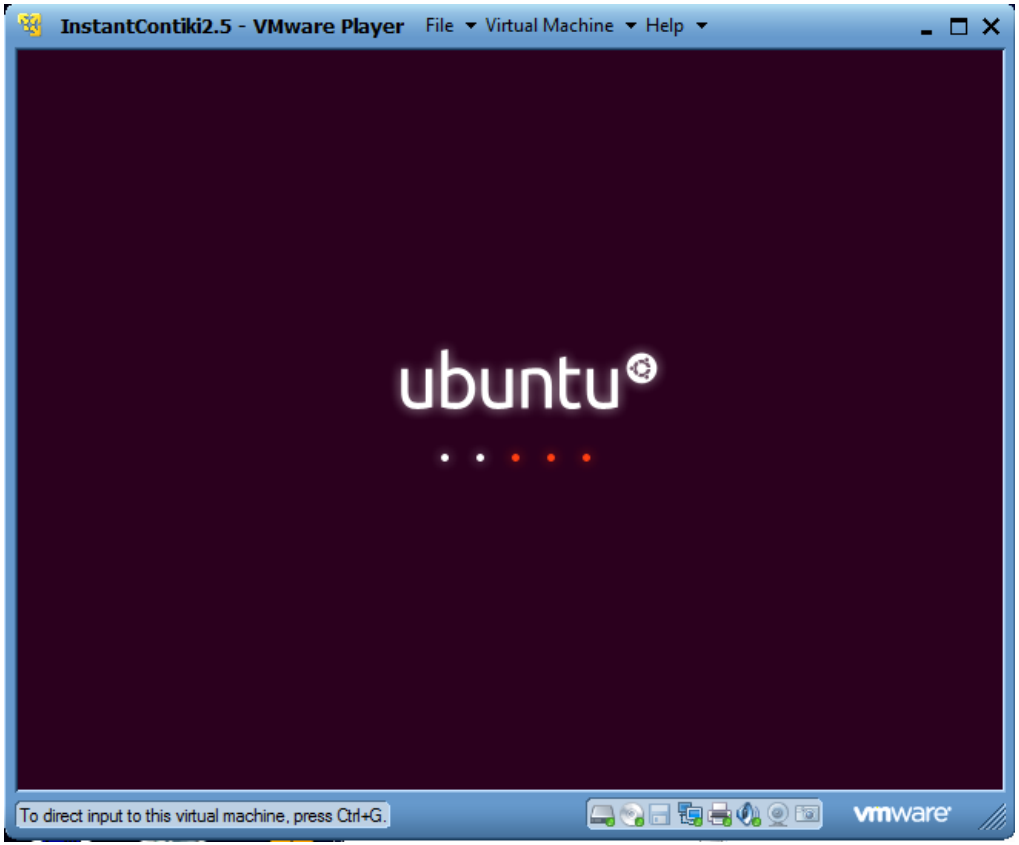
Get Started with Contiki

http://www.contiki-os.org/start.html

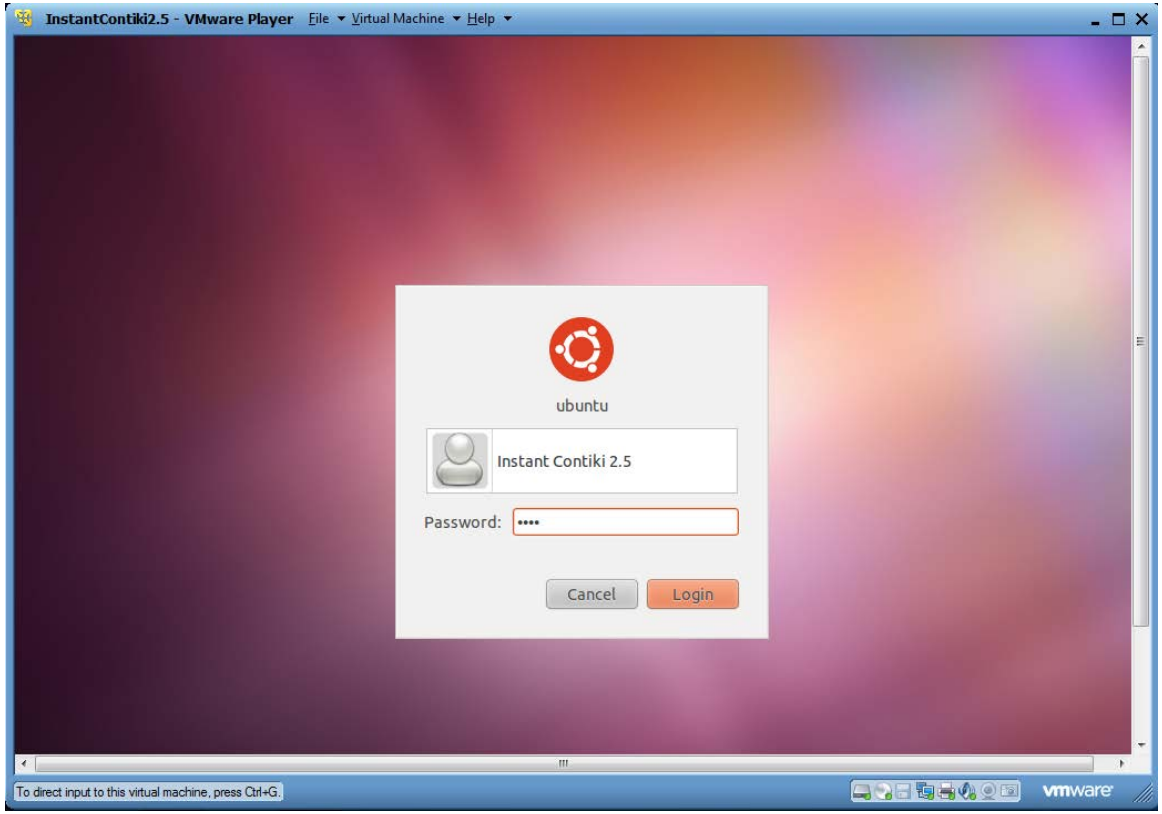
Firstly

- Download Instant Contiki
  - http://sourceforge.net/projects/contiki/files/Instant%20Contiki/
- Install VMWare Player
  - https://my.vmware.com/web/vmware/free#desktop_end_user_computing/vmware_player/6_0
  - Register and reboot
- Start Instant Contiki
- Start Instant Contiki by running InstantContiki2.6.vmx. Wait for the virtual Ubuntu Linux boot up.
- Log into Instant Contiki. The password is **user**.

Boot Ubuntu


InstantContiki2.5 - VMware Player
File
Virtual Machine
Help
ubuntu
To direct input to this virtual machine, press Ctrl+G.
vmware

Log in

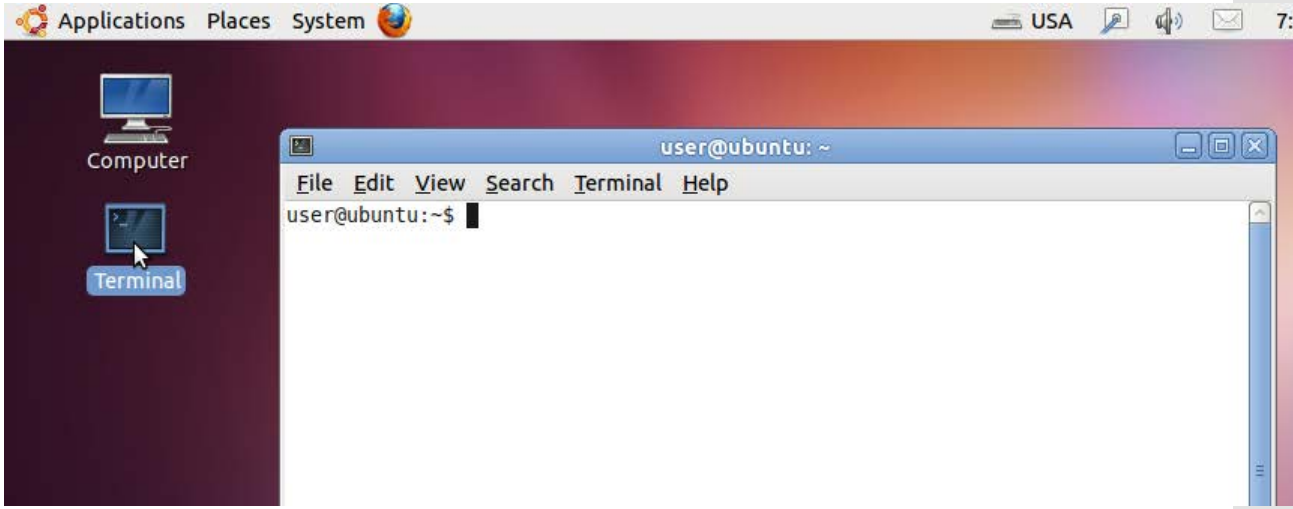

InstantContiki2.5 - VMware Player
File
Virtual Machine
Help
ubuntu
Instant Contiki 2.5
Password:
Cancel
Login
To direct input to this virtual machine, press Ctrl+G.
vmware

Step 2: Start Cooja

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

Open a terminal window

- We will now compile and start Cooja, the Contiki network simulator.
- Starting the terminal



Start Cooja

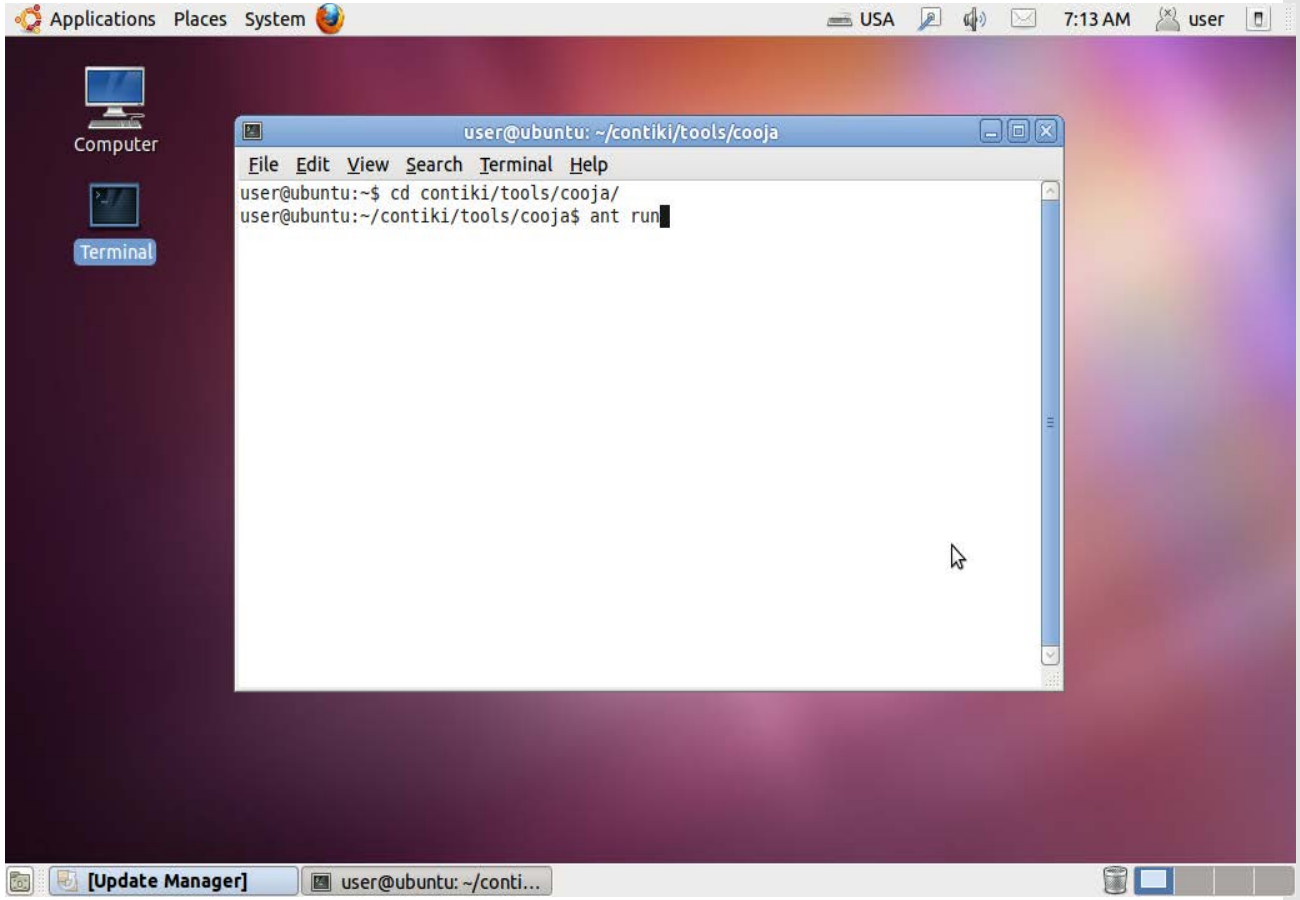
- Start Cooja by
- In the terminal window, go to the Cooja directory:

```
cd contiki/tools/cooja
```

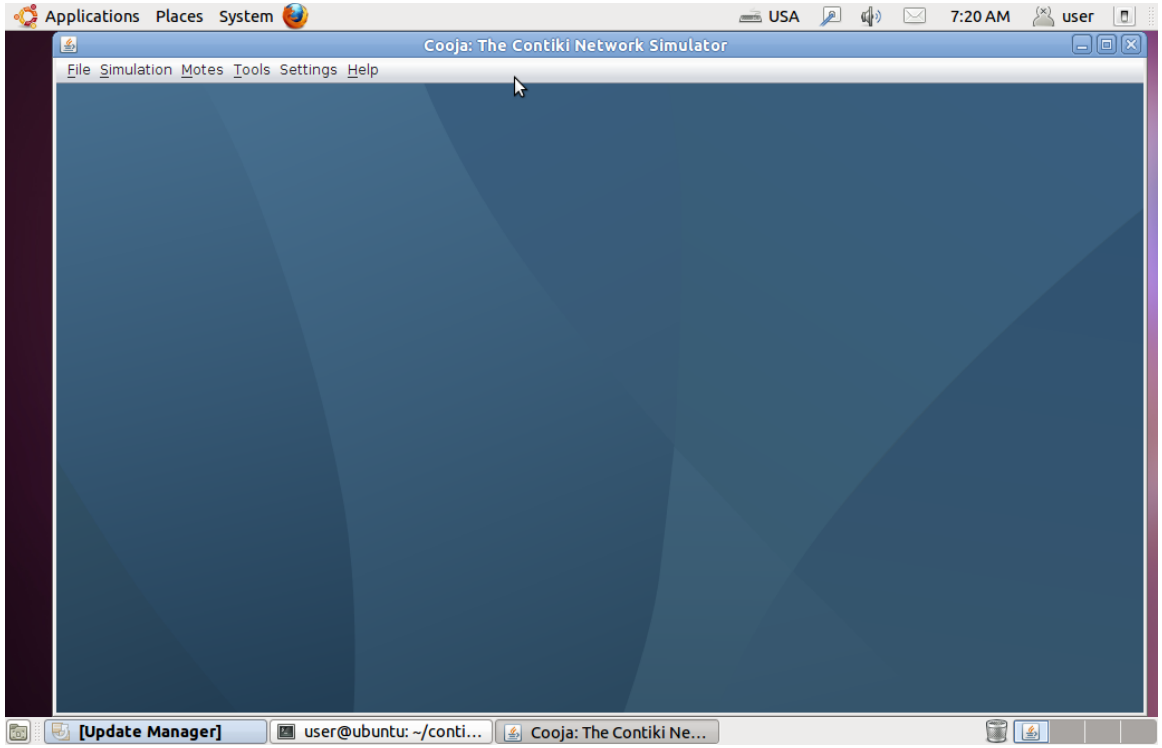
- Start Cooja with the command:
-

```
ant run
```

Running Cooja



Cooja UI

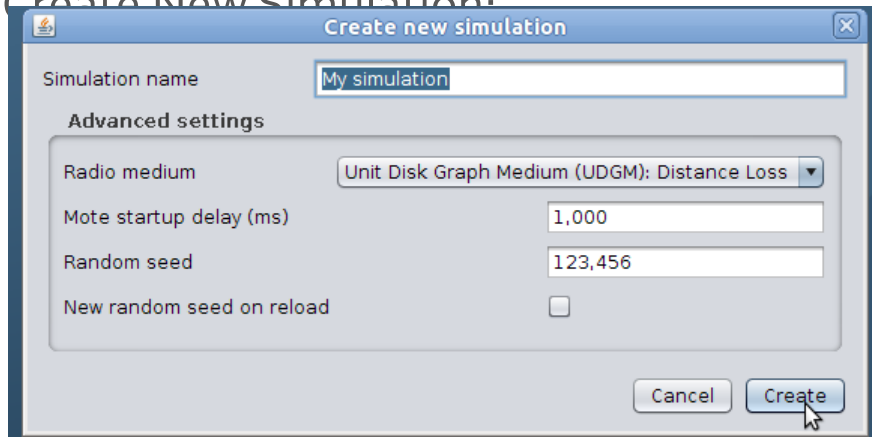

Applications
Places
System
USA
7:20 AM
user
Cooja: The Contiki Network Simulator
File Simulation Motes Tools Settings Help
[Update Manager]
user@ubuntu: ~/conti...
Cooja: The Contiki Ne...

Step 3: Run Contiki in simulation

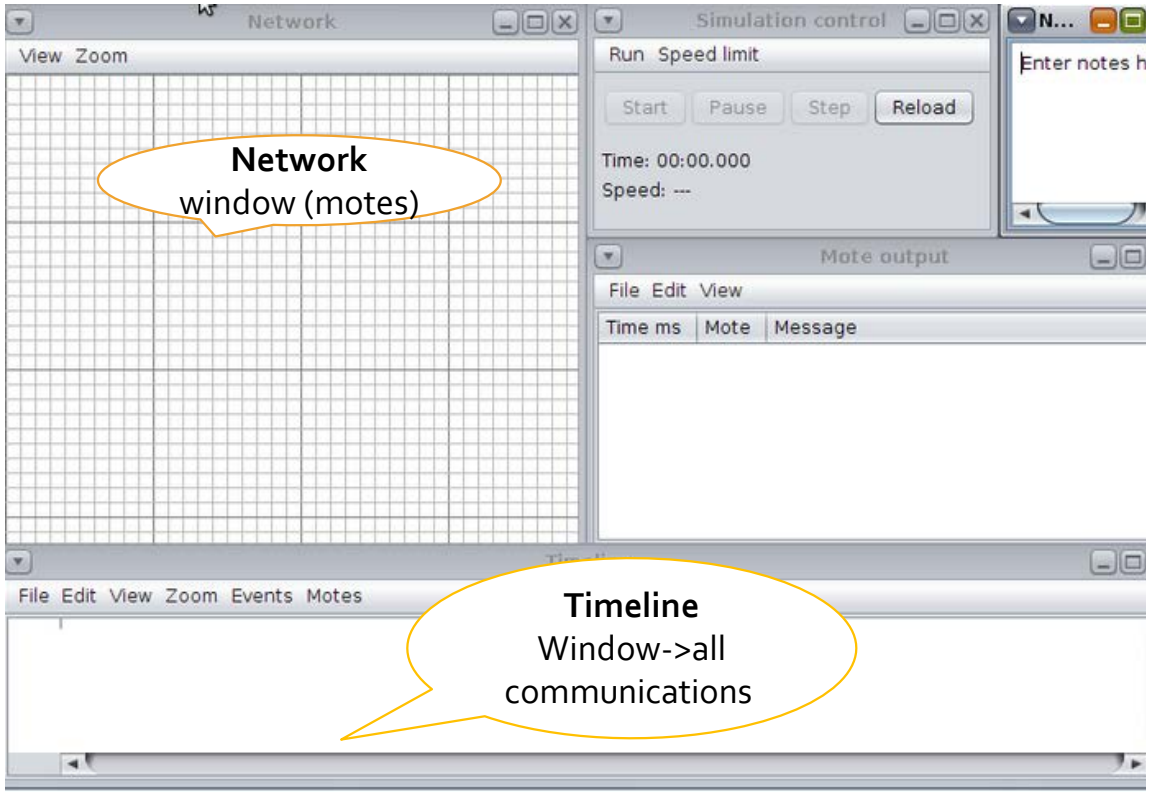
Creating Simulation

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

Create New Simulation:



Simulation Window



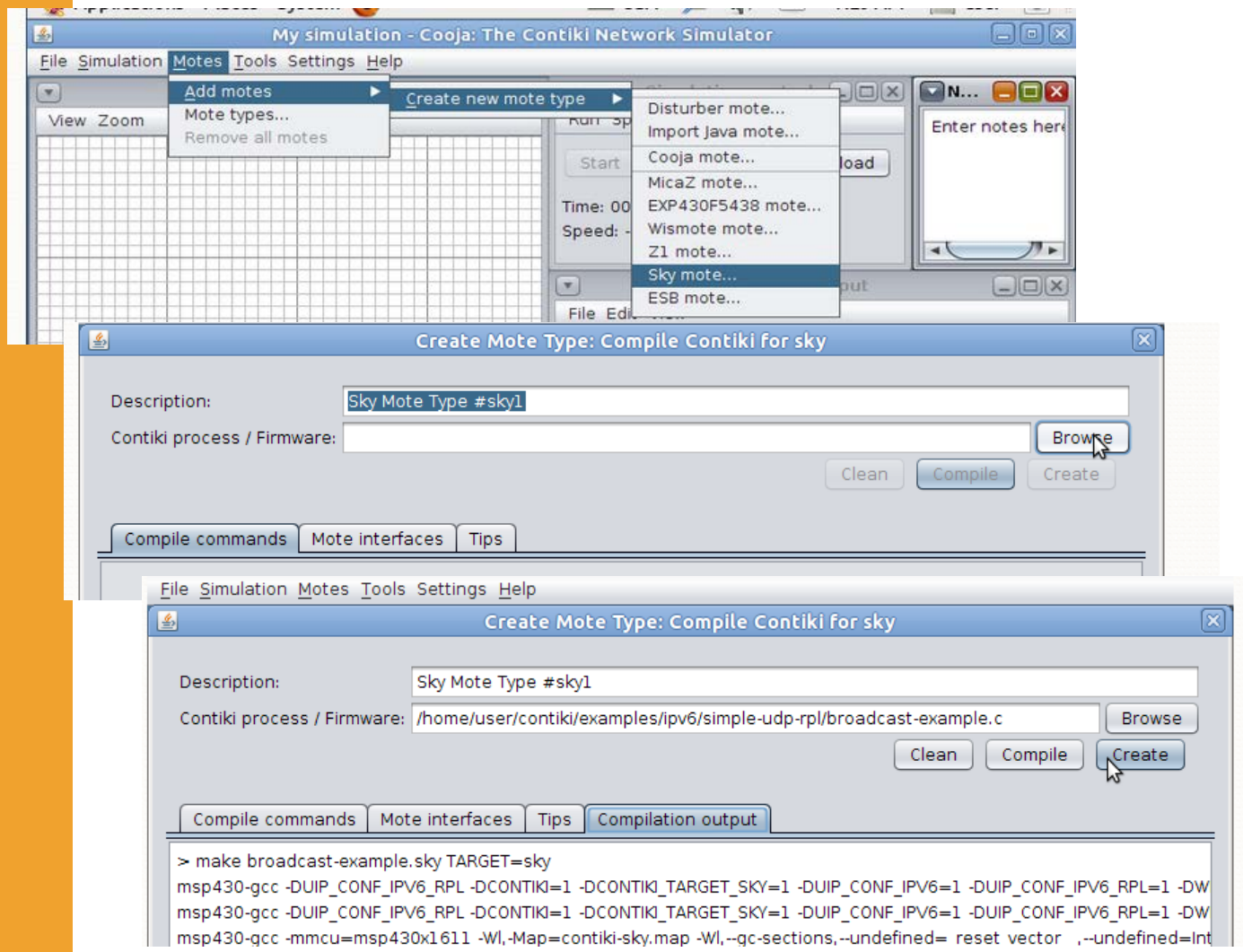
Simulation windows

Network window	-Top left -shows all motes in the simulated area
Timeline window	- At the bottom of screen - Shows all communication events in the simulation over time - very handy for understanding what goes on in the network
Notes window	- On the top right is where we can put notes for our simulation.
Mote output window	- on the right side of the screen - shows all serial port printouts from all the motes.
Simulation control	-window is where we start, pause, and reload our simulation

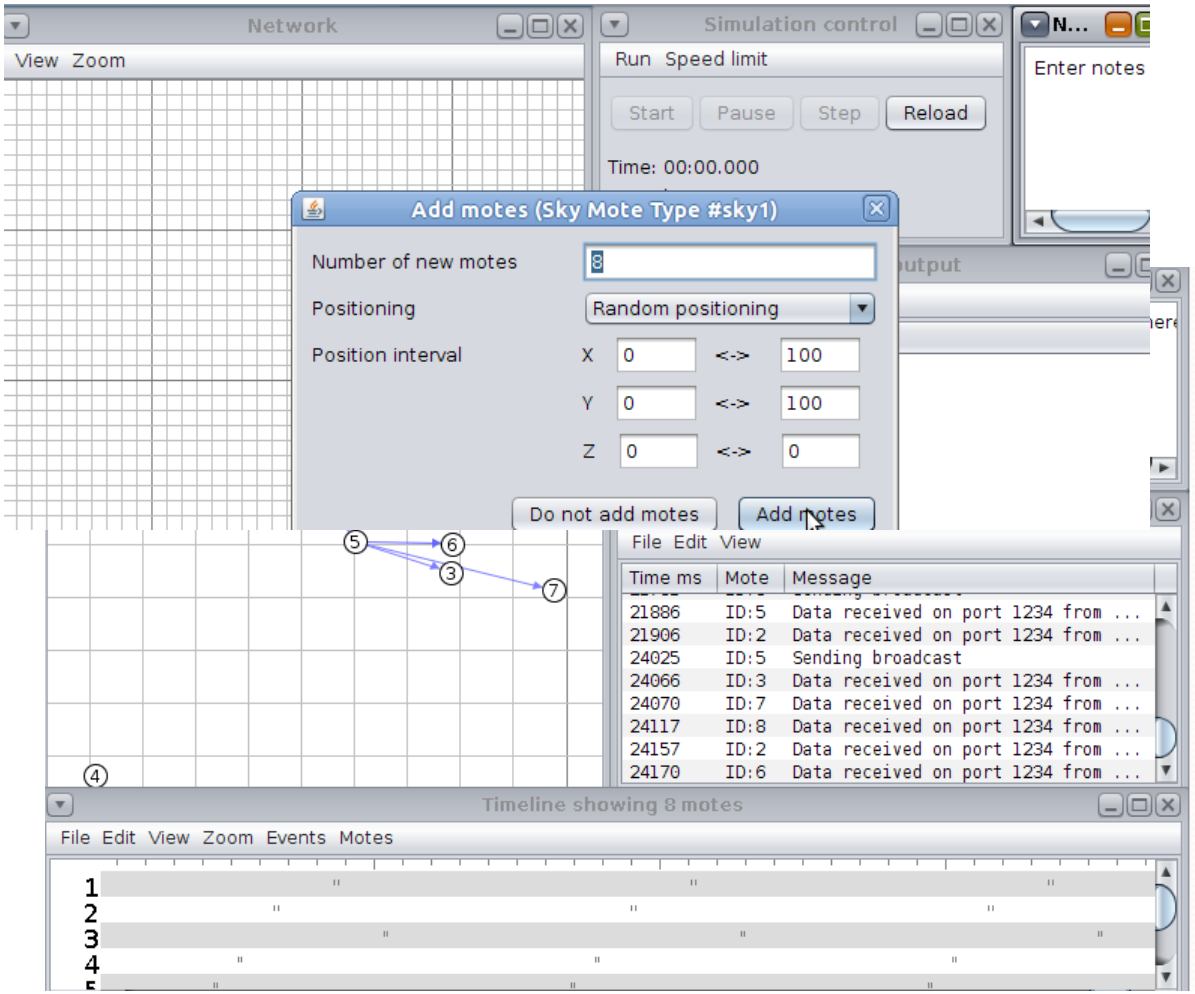
Add mote to the interface

- Before we can simulate our network, we must add one or more motes.
- We do this via the **Motes** menu, where we click on
  - **Add motes…**.
- Since this is the first mote we add, we must first create a mote type to add.
- Click **Create new mote type…** and select one of the available mote types.
- For this example, we click **Sky mote…** to create an emulated Tmote Sky mote type.
- Choose the hardware then install the firmware with any functionality that you desire

Add mote to the interface



Simulation window



Blink Application example

-Code
-explanation
-Cooja runnin’
-video

Blink example code

-Code

```
#include "contiki.h"
#include "dev/leds.h"
/*---------------------------------------------
PROCESS(blink_process, "Blink");
AUTOSTART_PROCESSES(&blink_process);
/*---------------------------------------------
PROCESS_THREAD(blink_process, ev, data)
{
  PROCESS_EXITHANDLER(goto exit;)
  PROCESS_BEGIN();

  while(1) {
    static struct etimer et;
    etimer_set(&et, CLOCK_SECOND);
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
    leds_on(LEDS_ALL);
    etimer_set(&et, CLOCK_SECOND);
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
    leds_off(LEDS_ALL);
  }

 exit:
  leds_off(LEDS_ALL);
  PROCESS_END();
}
/*---------------------------------------------
```

Code explanation

PROCESS(name, strname)	**Declare a process.** **PROCESS(blink_process, "Blink");**
AUTOSTART_PROCESSES(&blink_process);	Starting a process automatically &blink_process
PROCESS_THREAD(name, ev,data)	-Define the body of a process -The process is called whenever an event occurs in the system -Start with the PROCESS_BEGIN() macro - Ends with the PROCESS_END() macro.
PROCESS_EXITHANDLER(handler)	-Specify an action when a process exits. -Comes before declaring PROCESS_BEGIN()
PROCESS_BEGIN( ) ,PROCESS_END()	-specify the beginning and the End of a process

Code explanation

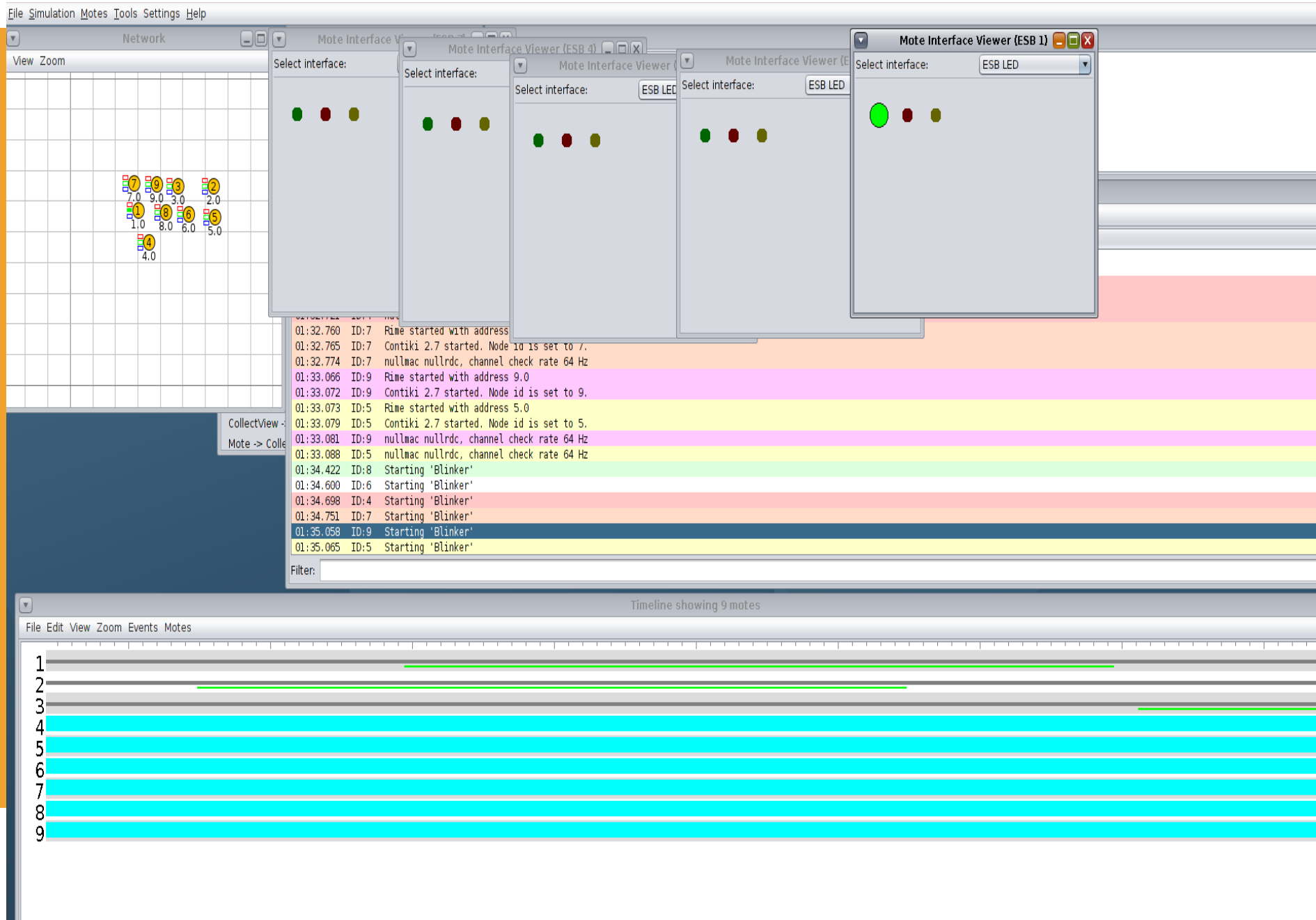
etimer et;	**-This structure is used for declaring a timer. The timer must be set with <u>etimer_set()</u> before it can be used.**
PROCESS_WAIT_EVENT_UNTIL(c)	Wait for an event to be posted to the process, with an extra condition. -This macro is similar to PROCESS_WAIT_EVENT() in that it blocks the currently running process until the process receives an event.
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));	-Check if an event timer has expired. -Parameters: et :a pointer to the event timer. -return true if the time expires
leds_on(LEDS_ALL);	On the LEDS
leds_off(LEDS_ALL);	Off the LEDs

LED API

- Simple and important to communicate with the user
- The platform startup code initializes the LEDs library by calling leds_init() initializes the API
- ledv: is LED vector platform independent
  - #define LEDS_GREEN 1
  - #define LEDS_YELLOW 2
  - #define LEDS_RED 4
  - #define LEDS_ALL 7
- leds_on() takes a LEDs vector argument, ledv, and switches on the LEDs set in the vector.
- Led_off() takes the LED in ledv and switch it off.
- leds_invert() inverts the current status of the LEDs set in the argument ledv.

```
void leds_init(void)s  : Initialize  the LEDs driver.
unsigned char leds_get(void)  : Get the status of a LED.
void leds_on(unsigned char ledv) : Turn on a set of LEDs.
void leds_off(unsigned char ledv) : Turn off a set of LEDs.
void leds_toggle(unsigned char ledv) : Toggle a set of LEDs.
void leds_invert(unsigned char ledv) : Toggle a set of LEDs.
```

Cooja capture



Youtube Demo

- http://www.youtube.com/watch?v=9WohGp8udOQ

APIs

process

PROCESS_BEGIN()	-Define the beginning of a process
PROCESS_END()	-Define the end of a process.
PROCESS_WAIT_EVENT()	-Wait for an event to be posted to the process. -blocks the currently running process until the process receives an event.
PROCESS_WAIT_EVENT_UNTIL(c)	Wait for an event to be posted to the process, with an extra condition.
PROCESS_WAIT_UNTIL(c)	Wait for a condition to occur.
PROCESS_EXIT()	Exit the currently running process.

LED functions

- Basic LED functions
- leds_on() - turn LEDs on
- leds_off() - turn LEDs off
- leds_invert() - invert LEDs
- **leds_blink()** - blink all LEDs

Timer functions

The Contiki kernel does not provide support for timed events. Rather, an application that wants to use timers needs to explicitly use the timer library.

Event Timer	**Callback timer**	**Simple timer**
-generates an event when the timer expires	-call a function when the timer expires	-have to be actively queried to check when they have expired
•etimer_expired() •etimer_reset •etimer_set() •etimer_restart()	•ctimer_expired() •ctimer_reset •ctimer_set() •ctimer_restart()	•timer_expired() •timer_reset •timer_set() •timer_restart()

References

- Contiki home: http://www.contiki-os.org/
- Gett started: http://www.contiki-os.org/start.html
- Resources: http://www.contiki-os.org/support.html
  - "Hands on Contiki OS and Cooja Simulator: Exercises (Part II)": https://george.autonomic-networks.ele.tue.nl/files/exercise_partII.pdf
  - Article: Adam Dunkels et al., "An adaptive communication architecture for wireless sensor networks", ACM SenSys'07 (http://dl.acm.org/citation.cfm?id=1322295)
    - Directory "examples\rime": example-abc.c, example-unicast.c, example-runicast.c etc
    - Directory "core\net\rime"
      - Single-hop: abc.h/abc.c (anonymous broadcast), unicast.h/unicast.c (unicast), runicast.h/runicast.c (reliable unicast), stunicast.h/stunicast.c
      - Multi-hop: multihop.c, rmh.c, collect.c, mesh.c, netflood.c etc
  - IETF standards-conforming implementation
    - core\net\mac\tsch, core\net\rpl, core\net\ipv6
  - Doxgen documentation: http://contiki.sourceforge.net/docs/2.6/
    - Rime communication stack: http://contiki.sourceforge.net/docs/2.6/a01798.html
    - uIP TCP/IP stack: http://contiki.sourceforge.net/docs/2.6/a01793.html
- Contiki Tutorials: http://anrg.usc.edu/contiki/index.php/Contiki_tutorials
- Troubleshooting: http://anrg.usc.edu/contiki/index.php/Troubleshooting
- Community: http://www.contiki-os.org/community.html

Exercise

- Download and install Contiki
  - Gett started: http://www.contiki-os.org/start.html
- Read the following and execute relevant simulation experiments
  - "Hands on Contiki OS and Cooja Simulator: Exercises (Part II)": https://george.autonomic-networks.ele.tue.nl/files/exercise_partII.pdf
  - Article: Adam Dunkels et al., "An adaptive communication architecture for wireless sensor networks", ACM SenSys'07 (http://dl.acm.org/citation.cfm?id=1322295)
    - Directory "examples\rime": example-abc.c, example-unicast.c, example-runicast.c etc
    - Directory "core\net\rime"
      - Single-hop: abc.h/abc.c (anonymous broadcast), unicast.h/unicast.c (unicast), runicast.h/runicast.c (reliable unicast), stunicast.h/stunicast.c
      - Multi-hop: multihop.c, rmh.c, collect.c, mesh.c, netflood.c etc
  - IETF standards-conforming implementation
    - core\net\mac\tsch, core\net\rpl, core\net\ipv6