

Contiki COOJA Hands-on Crash Course: Session Notes

Thiemo Voigt (thiemo@sics.se), based on previous versions by
Fredrik Österlind and Adam Dunkels
fros@sics.se, adam@sics.se
Swedish Institute of Computer Science

CONET Summer School, July 2009

1 Introduction

Welcome to this Contiki programming course! Contiki is a state-of-the-art, open source operating system for sensor networks and other networked embedded devices [3]. Contiki was the first operating system for sensor networks to provide TCP/IP communication (with the uIP stack) [1, 6], loadable modules [2], threading on top of its event-driven kernel [3], protothreads [5], protocol-independent radio networking (with the Rime stack) [7], cross-layer network simulation (with COOJA [9] and MSPSim [8]), and software-based power profiling [4]. Recent features include a networked shell and the memory efficient flash-based Coffee file system [11]. This course can be performed using the COOJA and MSPSim simulators as well as Tmote Sky boards. We will look at various cool Contiki and COOJA features such as the COOJA timeline, Contiki's Rime stack and protothreads, power profiling with Contiki and MSPSim, the Coffee file system and others.

The course ends with a challenging exercise: we will collaboratively build a neighbor discovery application based on RSSI measurements. These notes describe the practical, hands-on session with Tmote Sky boards and the COOJA and MSPSim simulators. The purpose of this course is to get experience with Contiki using actual hardware and simulation. We hope that this experience will be of help when later working with Contiki.

You can use the Instant Contiki development environment in the exercises. Instant Contiki is a single-file download that contains the Contiki source code and all necessary compilers and tools required for developing software for Contiki. The Instant Contiki environment is a Ubuntu Linux installation that runs within the VMware Player virtual machine execution envi-

ronment. VMware Player is available for free at the VMware website. If you have Contiki working on your native machine without Instant Contiki that is of course also fine.

In this course we use Instant Contiki 2.3, with the addition of a few Contiki programs used in the final exercise. The initial exercises, however, are based on standard Contiki example programs already included in Contiki 2.3.

These notes are structured as follows. Section 2 describes the pre-course requirements and Section 3 how to get started at the course. Section 4 contains instructions for the programming exercises for single nodes and introduces MSPSim. Section 5 contains information on how to simulate Contiki programs using MSPSim and COOJA. Section 6 has instructions for exercises using the Rime stack and multiple nodes. Section 7 contains instructions for the final neighbor exercise, which can be useful for debugging or understanding network behavior throughout the course. Section 7 presents the neighbour discovery exercise. Finally, Section 8 concludes this course.

In this hand-outs you may sometimes find siena09 since Fredrik gave a Contiki course in Siena earlier this month.

2 System Requirements

For this course, you need a PC that can run Instant Contiki (VMWare) and access to a Tmote Sky board [10] (Figure 1). We have confirmed that Instant Contiki runs in the VMWare Player, which is available free of charge from the VMWare website, under Windows and Linux, and with VMWare Fusion, which costs money, under Mac OS X. We have a few Tmote Sky boards for the



Figure 1: A Tmote Sky board is useful but working with the simulators also possible, except for the final exercise.

course in case you do not have any with you.

2.1 Download, Install Tools

Instant Contiki and other required tools can be downloaded from the Contiki website: <http://www.sics.se/contiki/instant-contiki.html> But they are also included on some memory sticks that we provide.

- VMWare Player
- FTDI Driver
- Instant Contiki

Note that the FTDI driver must be installed before plugging in any Tmote Sky boards.

Unfortunately, Thiemo installed this a long time ago and has does not really recall how things worked (might be more useful to ask someone who recently did it, e.g. your neighbour).

2.2 Exercise Software

The exercise programs used in the course are based on standard Contiki examples, so they are already in the provided Instant Contiki 2.3. The Contiki exercise programs are also available for download:

<http://www.sics.se/~thiemo/contiki-crash.zip>

The zip-file contains a single directory called `contiki-crash`. The contents should be extracted into the Contiki examples directory, thus creating the directory:

```
contiki-2.x/examples/contiki-crash
```

3 Getting Started

Before starting with the actual exercises, make sure that your development setup works by conducting the steps below.

3.1 Start Instant Contiki

Open the file

```
instant-contiki.vmx
```

to start VMware and Instant Contiki.

3.2 Log In

When the login screen appears, log in to Instant Contiki:

- Username: **user**
- Password: **user**

3.3 Open a Terminal Window

After logging in, click on the terminal icon to start a terminal window.

3.4 Verify that your system works

Try one of the examples below (or both):

3.4.1 Compile, Run Hello World

In the terminal window, go to the hello world example directory, and compile for the native platform:

```
cd contiki-2.x_siena09 // or do cd contiki-2.x
                        depending on your image
cd examples/hello-world
make TARGET=native
```

Wait for the compilation to finish. Run the Hello World program in Contiki:

```
./hello-world.native
```

The program should print the words “Hello, world” on the screen and then appear to hang. In reality, Contiki is still running correctly, but will not produce any output because the Hello World program has finished. Press Ctrl+C on the keyboard to quit.

3.4.2 Start COOJA

```
cd contiki-2.x_siena09
cd tools/coolja
ant run
```

This should start COOJA. You can exit COOJA again (Ctrl+X or menu).

3.5 Connect the Tmote Sky

Put a Tmote Sky in the computer's USB port. The Tmote Sky will appear in the top of the Instant Contiki (VMware Player) window with the name "Future Technologies Device". Click on the name to connect the Tmote Sky to Instant Contiki.

3.6 Run Hello World on the Tmote Sky

Unfortunately, there is a bug in instant-contiki (at least in earlier 2.3 versions). A fix is needed to enable download of code on Tmote Sky.

```
cd tools/sky
chmod u+x tmote-bsl-linux
```

Now you should be able to compile and upload the Hello World program on the Tmote Sky:

```
cd ../../examples/hello-world
make hello-world.upload TARGET=sky
```

Wait for the compilation and uploading procedure to finish. During the uploading the Tmote Sky should quickly flash the red LEDs next to the USB connector.

Connect to the USB port to view the program output:

```
make login TARGET=sky
```

Press the reset button on the Tmote Sky and a message similar to the following should appear:

```
Contiki 2.3 started. Node id is set to 1.
Rime started with address 1.0
MAC 00:12:75:00:11:6e:cd:fb X-MAC channel 26
Starting 'Hello world process'
Hello, world
```

The Contiki boot-up code prints the first four lines, and the Hello World program prints out the last line. Press Ctrl+C to quit.

Potential Problem: Node id is 0. That happens when the node does not have a (Contiki)-id burnt into

the flash. You can burn a node-id (e.g. 33, good idea to take another number) into the flash by:

```
make TARGET=sky burn-nodeid.upload nodeid=33
```

4 Single-Node Exercises

This section contains the exercises that are also on the slides. Most of the exercises can be performed with the simulators or with real nodes but some are targeted towards a specific tool, so that you at least see them once during the course. Use an editor that works for you, e.g. gedit or emacs.

4.1 Hello World on MSPSim

MSPsim is a Java-based instruction level emulator for the MSP430 series microprocessor and emulation of some sensor networking platforms. MSPSim supports loading of IHEX and ELF firmware files, and has some tools for monitoring stack, setting breakpoints, and profiling. It can run standalone or within COOJA where it emulates single nodes and COOJA provides the radio medium and other simulation features.

The standalone MSPSim version is useful for the development of code that runs on a single node. For example, for the development of the Coffee filesystem MSPSim (together with the shell) has proven to be very useful.

MSPSim can be started directly from the command line:

```
make TARGET=sky hello-world.mpsim
```

This should automatically start MSPSim. In one of the windows "USART1 Port Output", you should see the lines printed by the boot-up code. MSPSim also includes LEDs that can blink and one of the cool features is that you can actually press the buttons! Check what happens when you press the user or reset button.

As a first small task, extend the program to toggle the LEDs and print something. See code on the corresponding slide. Do not forget to include "dev/leds.h".

4.2 Extending Hello World I

The next exercise is to extend the Hello World program to let it print something (and/or toggle the leds)

every four seconds or when the button is pressed. On the slides you find some useful code samples. Use the `hello-world.c` in `examples/hello-world/` as a starting point.

First, you need some code for the timer:

```
static struct etimer et;
..
etimer_set(&et, CLOCK_SECOND*4);
..
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
..
```

For the button sensor you need to include some files (see also `examples/sky/test-button.c` from where you can copy code lines:

```
#include "dev/button-sensor.h"
#include "dev/leds.h"
#include <stdio.h> /* For printf() */
```

A button sensor needs to be activated:

```
button_sensor.activate();
```

To test if a button has been pressed or a timer has expired:

```
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et) ||
ev == sensors_event && data == &button_sensor);
if (ev == sensors_event) {
    printf("button pressed");
}
leds_toggle(LED_S_ALL);
```

Instead of using `PROCESS_WAIT_EVENT_UNTIL(..)`, you can also use `PROCESS_WAIT_EVENT(..)` and then check what type of event has occurred using `ev == sensors...` as above to check if the button has been pressed. For checking if the timer has expired you can use the same condition as in the `PROCESS_WAIT_EVENT_UNTIL(..)` statement, namely `if(etimer_expired(&et))`.

4.3 Extending Hello World II

The next task is to extend your Hello World program by adding a counter that counts the number of events that have occurred. For this, add a variable and see what happens.

Hopefully, you do not get the expected result because then you have experienced one tricky problem with pthreads (that has a simple solution, see the slides).

4.4 Downloading Hello World on Tmote Sky

Now you can download your program on your Tmote Sky and execute it there:

```
make hello-world.upload TARGET=sky
```

Wait for the compilation and uploading procedure to finish. During the uploading the Tmote Sky should quickly flash the red LEDs next to the USB connector.

Connect to the USB port to view the program output:

```
make login TARGET=sky
```

The behaviour should be similar as in MSPSim in that you can press the buttons.

5 Simulating Contiki with COOJA

This section explains how to simulate Contiki programs in the COOJA simulator. COOJA is a network simulator, and uses MSPSim to simulate networks of Tmote Sky nodes.

5.1 Start COOJA

Go to the COOJA directory, compile and start COOJA:

```
cd contiki-2.x_siena09
cd tools/coolja
ant run
```

COOJA compiles, and after a few seconds the simulator appears. All COOJA simulations are controlled using plugins: small Java programs that interact with simulations and simulated nodes. When COOJA is started, no simulation is loaded and no plugins are started.

5.2 Create a Simulation

A new simulation is created via the menu.

- Click menu item: **File, New Simulation.**

A number of configuration options are presented. Feel free to ask if you have any questions.

- Enter a **Simulation title**, and click **Create.**

We have now created our first simulation in COOJA. However, the simulation does not contain any nodes yet. To add nodes we need to first create a node type, and then add nodes to the simulation.

5.3 Create a Node Type

Any simulated node in COOJA belongs to a node type. The node type determines, among others, which Contiki applications to simulate. The node type also determines whether nodes are simulated or emulated.

- Click menu item: **Mote Types, Create mote type, Sky Mote Type**.

You have selected to emulate Tmote Sky nodes, and now need to select what Contiki program to simulate.

- Enter a **Description**
- Click **Browse**, and navigate to **examples/rime/example-abc.c**
- Click **Compile** to start compiling the Contiki program

When the compilation finishes:

- Click **Create**.

We have now created a simulation with a single node type. Before finally starting to simulate, we need to add nodes belonging to this node type.

5.4 Add Simulated Nodes

A dialog allowing you to add nodes has appeared. This dialog can later be accessed via:

- Menu item: **Motes, Add notes of type**, [your type description].

Add 5 nodes:

- Enter **5**, and press **Create and Add**.

Five nodes are added to the simulation, randomly located in the XY-plane.

Instead of using the **Random Position**, you can also use **Linear, Ellipse, or Manual Positioning**. Note that when you add nodes later, the visualiser plugin is rescaled in order to distribute the nodes all over the XY-plane. The same happens when you move nodes. Moving nodes is possible by left-clicking on a node and dragging it across the XY-plane.

5.5 Start Simulating

A number of plugins are automatically started. These, and more plugins, can be accessed via the plugins menu:

- Menu items: **Plugins, Log Listener** and **Plugins, Simulation visualizer**.

The Log Listener plugin listens to the serial ports of all nodes. Select the Log Listener and press the F1 key for help.

The Simulation Visualizer shows the positions of the simulated nodes as viewed from above. Enable the different visualizer skins by mouse clicking the upper part of the plugin. Mouse drag and drop nodes to change their positions.

In the Control Panel:

- Click **Start** to start the simulation.

Note the node serial data appearing in the Log Listener.

5.6 Save, Load and Reload

COOJA allows for saving and loading simulation configurations. When a simulation is saved, any active plugins are also stored with the configuration. The state of a current simulation is however not saved; all nodes are reset when the simulation is loaded again. To save your current simulation:

- Click menu item: **File, Save simulation**.

Simulations are stored with the file extensions “.csc”. To later load a simulation:

- Click menu item: **File, Open simulation, Browse....** Select a simulation configuration.

When a simulation is loaded, all simulated Contiki applications are recompiled.

A functionality similar to saving and loading simulations, is *reloading a simulation*. Reloading can be used to reset the simulation – to restart all nodes. More importantly, reloading a simulation will recompile all Contiki code, useful while developing Contiki programs. To reload your current simulation:

- Click menu item: **File, Reload simulation**, or press **Ctrl+R**

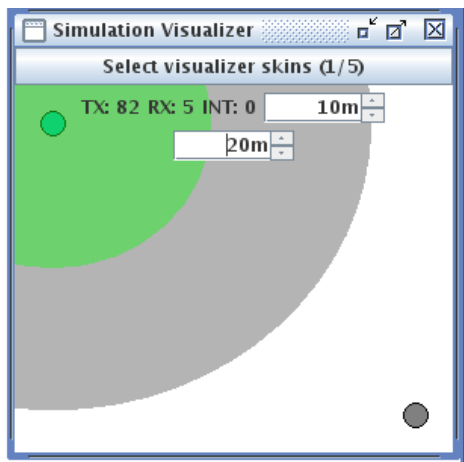


Figure 2: COOJA visualising Communication properties in Unit Disk Graph Model

5.7 Node Communication within COOJA

Once you have added nodes, you may want to check if they can communicate with each other. The following discussion applies to COOJA's default radio model, namely the unit disk graph model. For this, choose the **Simulation visualizer** plugin. Left-click on the **Select visualizer skin** textbox and tick UDGM in the text window that pops up. If you now left-click on one of the nodes, you will see a green circle around the selected node. The green circle presents the **transmission range** of the node, i.e. the selected node can communicate with all other nodes within that circle. You may also see a grey circle around the green circle. This circle represents the **interference range**. That is when the selected nodes transmits packets, a node in the grey area cannot receive packets correctly but it is interfered which means that it is not able to receive packets from other nodes when the selected node communicates simultaneously. The areas that are not covered by the green or grey circle are white; communication in that areas is not affected at all by transmission from the selected nodes.

When you want nodes to communicate, you have at least two options if they are not within the same green circle: you can move them, or you can change the transmission range. The latter is done by right-clicking in the XY-plane and choosing the action **change transmission ranges**.

5.7.1 Exercise: Node communication in COOJA

The goal of this small exercise is to create two nodes and make them communicate. For this, create two nodes of Contiki or Sky Type that run the example-abc program. This program is in **examples/rime/example-abc.c**. Once you have created the nodes, perform all necessary steps to make the nodes communicate. Once they communicate, change their transmission ranges to stop all communication between them.

An interesting plugin in that context is the **Radio Logger Plugin**. Turn it on and look what it displays. Note that you might not only see data packets but also smaller probe or strobe packets if you use a low power MAC protocol.

6 Exercises with Multiple Nodes

In the next part, we will use some networking features of Contiki. Contiki has two major communication stacks, mainly IP and Rime. In this course, we will concentrate on the Rime stack.

6.1 MAC layers

Contiki has three main MAC layers: X-MAC, LPP (low power probing) and NULLMAC. X-MAC and LPP are low power MACs that work best under low traffic loads, so they do not work well with 60 people in a room. The default MAC layer in Contiki on Tmote Sky is X-MAC.

Actually, MSPSim is a very useful tool for MAC layer "behaviour observation" as well. Start hello-world in MSPSim and look at the window called "Duty Cycle". The green line is the Duty Cycle for the radio in "listen mode". Also the boot-up code prints out the name of the MAC layer that is used.

Now let's change the MAC layer to NULLMAC (non-persistent CSMA). There are several ways to change the MAC layer to NULLMAC, e.g. using compile commands, modifying the Makefile etc. Some of these do not work with COOJA, so we will use a brute force method and modify platform/sky/contiki-conf.h. The MAC layer to be used is specified in the top of this file, use

```
#define MAC_CONF_DRIVER nullmac_driver
```

Add the line if it is not there and uncomment the other mac drivers (do not remove the lines, so you can switch to another MAC layer later).

6.2 Broadcast

The first exercise consists of sending your name to a base station with broadcast packets. This exercise shows how to use the broadcast communication primitive in the Rime stack and how to put data in packets.

The program to use is called `example-abc.c` and resides in the directory `examples/rime`. `abc` stands for anonymous broadcast, i.e. it adds no headers at all, not even the sender id. Go there directly by typing

```
cd ..
cd rime
```

The example program sends broadcast packets with a random interval between 2 and 4 seconds. All nearby Tmote Sky boards will receive the packet.

Open the file `example-abc.c` by typing:

```
gedit example-abc.c &
```

Go down to the line that contains

```
packetbuf_copyfrom("Hello", 6);
```

Change this line so that the string contains your name or email address. The second argument is the number of characters in the name string, including the terminator character.

Compile and upload the broadcast program to your Tmote Sky: in the terminal window, type:

```
make example-abc.upload TARGET=sky
```

This will compile and upload the program to the Tmote Sky connected to your PC. This takes some time the first time because the entire Contiki operating system is compiled.

When the compilation and uploading has finished, watch your name appear on the projector screen.

To see what broadcast messages your Tmote Sky sees, type the following in the terminal window:

```
make login TARGET=sky
```

This shows all serial output that the Tmote Sky is sending over the USB port. To stop, press `Ctrl+C`.

Go through the code in the `example-abc.c` file to see how it works. If you have any questions about how it works, don't be afraid to ask!

6.3 Unicast

The unicast program looks similar to the broadcast program, however, a destination address is needed.

Now start COOJA and create two nodes that run the program `example-unicast` in `examples/rime`.

Can the nodes communicate? Probably not. You might see the reason when you open the file `example-unicast.c`. Here, locate the lines that contain the following:

```
packetbuf_copyfrom("Hello", 5);
addr.u8[0] = 41;
addr.u8[1] = 41;
```

COOJA assigns node-ids starting from node-id 1. Therefore, you need to modify `example-unicast.c`. Can you get the two nodes to communicate?

6.4 COOJA's TimeLine

Now when you got two nodes communicate in COOJA, we can take a closer look at the timeline, a very cool and recent COOJA feature. As shown on the slide, when there is no colour, the radio is off. When the radio of a node is on, there is a grey line. On top of the grey line, you will sometimes see some colors: green when a packet is received, blue when a packet is sent and red indicates an interfered radio by e.g. collisions.

Now change the MAC layer and explore the behaviour. Either for unicast or broadcast.

6.5 Measuring Power Consumption

You can also measure the power consumption, e.g. to get an idea of the power consumption of the different MAC layers. Contiki has a software-based power profiler [4] that basically measures the times different components are turned on. This time can be multiplied with the voltage and a pre-measured current draw which gives power consumption. Using this mechanism requires only some extra lines of code as you can see on the slides.

You can also use MSPSim's power profiler that is based on the same principle (measure how long components are active) as Contiki's software-based profiler. To do this in COOJA, right-click on a node, open the mote interface and choose MSP Cli (command line interface). Type `duty 1 CC2420` (there is also a help command). There are columns with different values. Use different MAC layers (NULLMAC and one of the

low power MAC layers) and you should be able to find out what each column means.

6.6 Shell

The purpose of the shell exercise is to see how the Contiki shell works. The Contiki shell is a powerful application that makes it easy to interact with both individual nodes and a network of nodes.

Compile and upload the shell program:

```
cd ..
cd sky-shell
make sky-shell.upload
```

When compilation and uploading finishes, login to your Tmote Sky:

```
make login
```

Note that the file commands use the Coffee file system. You can now enter commands to the shell via the keyboard. `help` lists all installed commands, try them and play around:

```
help
sense | senseconv
ls
format
echo test | write file
ls
read file
nodeid
blink 10
reboot
repeat 2 2 { echo again } &
ps
```

Exit with `Ctrl+C`.

Another feature (pedagogically useful) that you can turn off a low power MAC protocol. For this you need to change to a low power MAC protocol in `platform/sky/contiki-conf.h` Use `MSPSim` again and do:

```
make sky-shell.mspsim TARGET=sky
```

When `MSPSim` is running, go to the bottom of the `USART1` window and type

```
mac 0
...
mac 1
```

Watch the effect of this on the duty cycle monitor.

6.7 COOJA Watchpoints with the Time-Line

The Contiki simulators have been developed to support fast development of Contiki programs, so they are not developed for people using simulation as their research methodology but for systems people that need development support. But they have also been used successfully for simulation studies.

One of the recent COOJA features is the TimeLine that we have seen previously. The TimeLine also supports so-called Watchpoints. These are similar to the usual breakpoints, but they do not stop the simulation. Instead they visualize the watchpoint and hence can be used to figure out if a certain statement of a sequence of statements is executed.

Normal breakpoints are set as follows: Right-click on a node in the visualizer plugin. Choose "open mote plugin→MSP code watcher". Select a source file, scroll to source code line and right-click "do add breakpoint". The simulation will stop at the breakpoint.

A Watchpoint is a breakpoint that is displayed in the TimeLine but the simulation does not stop. To convert the breakpoint into a watchpoint find the hidden pane in the left of the window. Open this pane by clicking on the left hand side of the window and drag the pane open. This pane lists all breakpoints. To turn the breakpoint into a watchpoint, deselect the "Stop" checkbox.

Next, chose a name and a color for the watchpoint. Do this by clicking in the empty "Info" field of the watchpoint. This opens a window in which one can enter a name and chose a color.

Finally, enable watchpoints in the TimeLine by selecting the "Watchpoints" checkbox. Start the simulation. The watchpoint for the selected node should now be visible as a tick in the TimeLine.

Exercise: Let's see how a watchpoint works. Modern low power radios such as the CC2420 on the Tmote Sky allow programmers to set the transmission power in order to change the transmission range (smaller transmission ranges enable more spatial reuse of frequencies and save some energy). On the CC2420, there are transmission power 32 levels available. The radio driver sets these and the code can be found in `core/dev/cc2420.c`. Find the right lines in the function `cc2420_send`. Search for `set_tx_power`. Now set one or two watchpoints there.

Now you can go to back to the example `example-abc.c` Modify the transmission power there by adding one line

```
packetbuf_set_attr(
    PACKETBUF_ATTR_RADIO_TXPOWER,
    txpower);
```

Note, that these line must be placed correctly, otherwise nothing happens. Check using watchpoints if you really modify the transmission power.

Once you have succeeded, download the code on your Tmote and execute the program. See what transmission power you need to reach the sink (remind Thiemo to run the right program on the sink). If you are really ambitious, you can do the following. Merge rime/example-abc.c with the hello world program that contains the button code (see Section 4.2) and increase the transmission power with one level whenever you press the button. Continue to increase until you see your name on the projector.

7 Neighbor Discovery Exercise

The neighbor discovery exercise consists of constructing a Contiki application that samples the on-board light sensor, and broadcasts the light measurements. In addition, it should also listen for light measurements from neighboring nodes, and sort all neighbors by their associated Received Signal Strength Indicator (RSSI). Finally, the three best neighbors (those with the highest RSSI values) should be reported to a sink node. The sink node is connected to a projector, and displays the overall progress on the projector screen as in Figure 3.

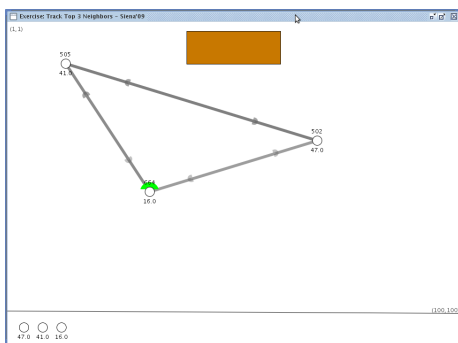


Figure 3: Screenshot of exercise with few nodes.

This exercise involves understanding existing examples, and combining them to build a complex Contiki application. In addition, it also illustrates how well radio RSSI measurements are for differentiating between good and bad neighbors.

Go to the contiki-crash directory with the exercise skeleton code and examples:

```
cd contiki-2.x_siena09
cd examples/contiki-crash
```

Below follows short descriptions about the provided Contiki code.

7.1 Skeleton: report-top3.c

A Contiki program with two processes: a light broadcasting process, and a neighbor reporting process. You may use this program as skeleton code when building the full application.

7.2 RSSI sampling: ex-broadcast-rssi.c

Example code that broadcasts packets using Rime, and prints the RSSI values of received radio packets.

7.3 Light sampling: ex-light.c

Example code reading from the Tmote Sky on-board light sensor.

7.4 Neighbor library: myneighbors.c

Simple neighbor library implementation that sorts neighbors according to their RSSI values. If you finish the exercise early, you may try to improve the implementation of this library, for example by keeping a moving average of RSSI values.

7.5 Neighbor library: ex-neighbors.c

Example code using the above neighbor library.

7.6 Packet structures: packets.h

Contains the packet structures to use, and the Rime channel info.

Note the simulation files (.csc) in the directory. These may be helpful for understanding and testing your programs. For example, to simulate the light sampling Contiki application:

```
make ex-light.csc TARGET=cooja
```

To solve the task you may need to help each other. Also, feel free to ask for help.

Note also that in the `Makefile.sky.defines` `NULL-MAC` is chosen as the MAC layer.

Good luck!

7.7 Quick-start Simulations

To quickly start simulating Contiki programs, both MSPSim and COOJA can be quickstarted from a Contiki working directory.

Go to the Rime examples directory:

```
cd contiki-2.x_siena09
cd examples/rime
```

To simulate `example-abc.c` in standalone MSPSim:

```
make example-abc.mspsim TARGET=sky
```

To create a COOJA simulation with `example-abc.c`:

```
make example-abc TARGET=cooja
```

Note that with this command, the nodes will be simulated at the operating system level, they will not be emulated using MSPSim.

To load a COOJA simulation configuration residing in the same directory:

```
make mysimulation.csc TARGET=cooja
```

8 Conclusions

This course is an introduction to Contiki programming – use it as a starting point for further exploration. Copy the examples and the exercise code in your own projects: it is a very good way to learn. Play around with COOJA to get a feeling for how it works.

Always feel free to ask us questions! Use the mailing list for best responses as more people will see the questions and have a chance to answer.

For more information, visit the Contiki web site: <http://www.sics.se/contiki/>

References

- [1] A. Dunkels. Full TCP/IP for 8-bit architectures. In *Proceedings of The First International Conference on Mobile Systems, Applications, and Services (MOBISYS '03)*, May 2003.
- [2] A. Dunkels, N. Finne, J. Eriksson, and T. Voigt. Runtime dynamic linking for reprogramming wireless sensor networks. In *ACM Conference on Networked Embedded Sensor Systems (SenSys 2006)*, Boulder, USA, November 2006.
- [3] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Workshop on Embedded Networked Sensors*, Tampa, Florida, USA, November 2004.
- [4] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He. Software-based on-line energy estimation for sensor nodes. In *Proceedings of the Fourth Workshop on Embedded Networked Sensors (Emnets IV)*, Cork, Ireland, June 2007.
- [5] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali. Prothreads: Simplifying event-driven programming of memory-constrained embedded systems. In *Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys 2006)*, Boulder, Colorado, USA, November 2006.
- [6] A. Dunkels, T. Voigt, and J. Alonso. Making TCP/IP Viable for Wireless Sensor Networks. In *Proc. EWSN'04 work-in-progress session*, January 2004.
- [7] A. Dunkels, F. Österlind, and Z. He. An adaptive communication architecture for wireless sensor networks. In *Proceedings of the Fifth ACM Conference on Networked Embedded Sensor Systems (SenSys 2007)*, Sydney, Australia, November 2007.
- [8] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, and T. Voigt. Accurate Network-Scale Power Profiling for Sensor Network Simulators. In *Proceedings of the Fifth European Conference on Wireless Sensor Networks (EWSN2009)*, Cork, Ireland, February 2009.
- [9] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *Proceedings of the First IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006)*, Tampa, Florida, USA, November 2006.
- [10] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling ultra-low power wireless research. In *Proc. IPSN/SPOTS'05*, Los Angeles, CA, USA, April 2005.
- [11] Nicolas Tsiftes, Adam Dunkels, Zhitao He, and Thiemo Voigt. Enabling Large-Scale Storage in Sensor Networks with the Coffee File System. In *Proceedings of the 8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2009)*, San Francisco, USA, April 2009.