Contiki bootcamp

Serial Shell, MSPSim, COOJA

Marcus Lundén, marcus.lunden@csiro.au

This document runs through some of the stuff I talked about and showed on the Contiki Bootcamp at CSIRO QCAT 2012 feb 17. It is more like a walkthrough of an example how to use them rather than descriptive and is written for a broad audience in mind so please don't be offended if you feel it is too simple. Focus is on how to use COOJA for debugging. For more information refer to the link in the end of the document. For corrections please email me and I'll update it.

Serial Shell

The Contiki Serial Shell is a UNIX-style shell that allows for text-based interaction, including features such as piping data, run in background etc. Having shell running on a mote allows for simple interaction and testing. The drawback is that it uses precious ROM/RAM space; hence it is not used in Contiki per default but must be added in the following way.

In project sourcefile

```
#include "shell.h"
#include "serial-shell.h"
...
serial_shell_init();
shell_sky_init();
shell_text_init();
shell_file_init();
shell_time_init();
shell_ps_init();
shell_coffee_init();
```

In Makefile

```
APPS=serial-shell
```

If you get an error like 'no rule for make all' when compiling, try moving this line up or down in the Makefile

Each shell_X_init() adds a set of features to shell, hence if you run out of space on the mote try removing these. They (above are examples, there are more) are located in `contiki-2.x/apps/shell/` in files like `shell-file.c`. For what features they come with, check the source file.

Basic shell commands and usage, using `shell_coffee` (the Coffee file system), `shell-time` and `shell-sky` as examples:

- List available commands
  ```
  ?
  ```
- List running processes
  ```
  ps
  ```
- Read sensors and convert to human readable format
  ```
  sense | senseconv
  ```

- Same, write to file
  ```
  sense | senseconv | write logfile.txt
  ```
- Periodically (every 2 seconds), forever (0), read sensors and append to logfile.txt, in background.
  ```
  repeat 0 2 {Sense | senseconv | append logfile.txt} &
  ```

Create your own shell commands like this

```
PROCESS(utoggle_process, "Toggle lines");
SHELL_COMMAND(utog_command,
      "ut",
      "ut: toggle lines",
      &utoggle_process);
/* --------------------------------- */
PROCESS_THREAD(utoggle_process, ev, data)
{
  PROCESS_BEGIN();
  // ...
  PROCESS_END();
}
...
shell_register_command(&utog_command);
```

MSPSim

MSPSim is a sensor board and MSP430 instruction simulator written in Java. Main author is Joakim Eriksson at SICS. It is cycle accurate and can simulate the Tmote Sky sensor mote (including MSP430 mcu, LEDs, CC2420 radio transceiver, M25P80 flash etc) and has recently gotten support for the MSP430x mcu that has 20-bit address space. MSPSim can run with GUI or CLI. Start the GUI by compiling with following syntax

```
make my-project-file.mspsim TARGET=sky
```

There, the buttons can be pressed (on the picture) and LEDs will reflect on their corresponding state. The mcu can be monitored and controlled from the control box and CLI. Another available platform is ESB: `TARGET=esb`

COOJA

COOJA is a network simulator that can create motes either from pre-compiled code (allows TinyOS-motes) or from source code (will be compiled for eg Sky motes or directly as x86-motes). COOJA is very extendable and provides many ways for doing so. Anyone can extend it and use their own radio propagation model, simulated mote (just as it per default uses MSPSim for Sky motes), add visualizers, plugins etc.

Start COOJA GUI by navigating to the following folder and running

```
cd contiki-2.x/tools/cooja/
ant run
```

If you are running a large simulation you can instead run

```
ant run_bigmem
```

Create a new simulation

```
File/New Simulation
```

Choose a radio propagation model (*UDGM* being probably the most simplistic as it only depends on distance and transmission power), random startup time (if you have several nodes they will start within this interval) and a seed for the random number generator.

Add a Tmote Sky mote:

```
Mote Types/Create/Sky mote type
```

Name the type of mote you are creating (eg 'Data sink', 'Sensor node'). **Browse** to find your source file (or precompiled binary), **Compile** and press **Create**. Enter how many you would like to have, press **Create**. Note that it/they now show as circles in Simulation Visualizer. For this, create one mote with '**silent.c**'.

Explore the **Control Panel**; use the slider to set simulation speed: full, real time or with delay. Now set it to '**Real time**' (one click on the slider when at the left).

In **Simulation Visualizer**, press 'Set visualizer skins' and choose eg Mote IDs, LEDs, and 10m background grid. You can zoom in and out by holding <Ctrl> and clicking + dragging (on Mac). To view all nodes again, right click and choose 'Reset viewpoint'.

Explore the **Timeline**. This is a very useful tool. Timeline can show when the radio is on/listening (grey) or off, transmissions (blue is transmission, green is receiving, red is interference or collision (not received but "sensed")), LEDs and watchpoints and breakpoints. Right click to zoom in/out.

The **log listener** will contain all UART output from the motes. You can filter on ID or contents. Right click and choose '**Mote specific coloring**'.

Start the simulation by pressing Start on the Control Panel. The mote should start periodic toggling of LEDs. Right click on it (in Simulation Visualizer) and choose '**Show serial port on Sky**'. This is where you access the Shell. Try typing '**?**' to see available commands. Try the commands mentioned above under Shell.

Use Timeline to zoom in to millisecond level to see how the power saving MAC protocol turns the radio on and off.

<u>Radio traffic and some debugging</u>

Now we will create a small network with a subset of transmitters among receivers.

Either start a new simulation (Ctrl + S) or add more motes to the already existing one. Create a total of **3** nodes with '**broadcaster.c**' and **7** with '**silent.c**'.

Make sure you have zoomed out Timeline, chosen 'Mote specific coloring' on Log listener and set time to 'Real time' in Control Panel. Start the simulation and observe in Timeline how three motes are transmitting broadcasts. Zoom in to see how it looks on a smaller time scale.

On any mote of your chosing, open the serial port and issue the command 'ut'. This shows a COOJA feature useful in visualizing networks: by outputting a certain string, lines will be drawn in the Visualizer. Issue 'ut' again to remove them. For drawing a line between a node and one with ID 9, the software on the mote should do this:

```
printf("#L 9 1\n");
```

To remove the line again:

```
printf("#L 9 0\n");
```

Right click in Timeline, choose '**Print statistics to the console**'. Switch to the console; now you can see for how long time the radio has been on/transmitting/receiving, red/green/blue LED on/off etc. Using LEDs is a good debugging aid in time critical applications such as debugging code in a interrupt service routine as printf's will both take too long time and also have a delay as it is printed out over UART.

More debugging tools

In Simulation Visualizer, right click, 'Open mote plugin' contains some useful tools.

- **Stack Watcher** will show a graph of the stack and can help telling if a memory leak occurs.
- **Variable watcher** will let you read or write any variable, including the hardware registers in the MSP430.
- **MSP CLI** is a command line interface to MSPSim. From there, `reset` will reset the CPU, `logcalls > log.txt` will save the function calls to text file, `exec pwd` will show where that file is located, `stacktrace` will show the call stack.
- **MSP Code Watcher** will open a window where you can view currently executing assembler code, its corresponding c file and position. Expand the divider on the left to show these hidden windows. Under 'Browse:' you can choose a c-file to view and set **watchpoints** and **breakpoints**.

There are a number of COOJA plugins available that are very useful as well. They are available through the main window menu 'Plugins'.

- **Radio Logger** will log and show all radio messages transmitted: their content, sender and receivers. If there are a set, commonly occurring packet, you can assign it an alias so it will be more easily identified among all other packets.
- **Buffer Listener** is Variable Watcher on steroids. Set it (right click) to display the packet buffer (buffer/*packetbufptr) or any other memory space on a node. For example, declare a char array and print debug messages to it (much faster than to do a regular printf as it does not have to wait for the UART to complete) and monitor that.

  ```
  static char cooja_debug_string[100] ;
  snprintf(cooja_debug_string , 100, "My debuginfo: %u\n", var);
  ```

  and set Buffer Listener to monitor 'Custom pointer', enter name 'cooja_debug_string' and Parser to 'Terminated string'. Monitoring a

variable with Parser 'Graphical: Grayscale' is good for seeing how a value changes over execution time.

Adding the Mobility plugin

COOJA is very extendable: you can add plugins, visualizers, packet analyzers etc. Mobility is a plugin currently residing in the Contiki Projects repository. Mobility accepts a text file with position data for the motes in the simulation. There are a number of such position data generators on the Internet, or use data gathered from real deployments or measurements. At the end of this document is a link to small position data files I have generated using the Random Waypoint Mobility Model. NB that Mobility itself do not generate any position data, just handles the movement itself.

To install it, download the from the following repository:

```
sourceforge.net/projects/contikiprojects
```

then look under sics.se, mobility.

Install process

- Put the mobility folder under your cooja/apps/ (eg `contiki-2.x/tools/cooja/apps/mobility`)
- Start COOJA
- Menu: Settings/COOJA Projects
- In file browser pane on the left, locate the mobility folder and click the box next to it so it becomes green
- Press Save as default
- Exit COOJA
- Run ant clean

Usage

- Start COOJA, create your simulation
- Start mobility; menu: Plugins/Mobility
- It will prompt you for a file with position data, browse and select it
- The motes will now move when simulation is running and there is position data for them

Note

- When/if the provided position data runs out and simulation is still running, it will start over again (overflow)
- If you close the Mobility window, all movement will be stopped
- Mobility can be started at any time, even after simulation has run some time. It will always start from the top of the position data file however, so temporary stopping movement is not possible.

Format of position data file

The position data file is a space separated value file with each line indicating a COOJA node ID, time [s], X position [m] and Y position [m]. Comments are preceded with #.

Example of four nodes moving a little bit, from their starting positions to new positions 0.2 seconds later.

```
# Comment
0 0.0 44.0413538236 135.737294054
1 0.0 39.1822871509 109.741972953
2 0.0 117.418994575 123.906576338
3 0.0 78.382316355 87.7137366549
0 0.2 44.3447495247 135.598526325
1 0.2 39.9107771733 109.470414563
2 0.2 116.784111194 123.715365587
3 0.2 77.6188236041 87.4762144466
```

Additional resources and links

Contiki

www.contiki-os.org

MSPSim

Main author, Joakim Eriksson, SICS

http://www.sics.se/node/126

http://www.sics.se/project/mspsim

COOJA

Main author, Fredrik Österlind, SICS

https://www.sics.se/people/fros

Contiki projects; Mobility

Repository address

sourceforge.net/projects/contikiprojects

Position data (ordinary text files, see header).

http://www.filedropper.com/positionsdat

Files used in this document

Broadcaster.c

http://pastebin.com/xCn3PFx5

Silent.c

http://pastebin.com/uPC1qEYz

Makefile

http://pastebin.com/vP3EeAJj

Mobility data (see above)