Module: Internet of Things **2024/25**

Lab Session 7: Power Consumption

*Lecturers: Dr Moh***ammad Shojafar, Dr Chuan H Foh, Dr Ahmed Elzanaty**

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

7.1 Introduction

In this session we are going to analyse the power consumption of a sensor node. We run a powertrace as a background process on the sensor node.

The powertrace reports the power consumption and resource utilisation of a node, and it prints the statistics to the console.

Equipment and materials:

- ❑ Lab PC or your own computer
- ❑ An XM1000 mote (optional, collect it during the lab)
- ❑ InstantContiki-2.6 image file, see "Lab Session 1" for the download link

7.2 Learning Objectives

We expect you to know the following from the previous lab sessions:

- How to launch Contiki-2.6 in the lab (if you are attending the lab) or from your own PC (if you are working remotely);
- How to compile and upload your code onto XM1000 mote, and debug your code by logging debugging messages onto the console (if you have a mote);
- How to load a Cooja simulation environment, start and stop the simulation, and debug your code by logging debugging messages onto the console;
- Be familiar with Cooja environment and Contiki programming structure;

In this lab session, you will learn:

- How to activate powertrace on Contiki;
- How to read the statistics reported by powertrace;

7.3 Powertrace Application[1]

The following source code demonstrates the use of powertrace.

```
#include "contiki.h"
#include "random.h"
#include "powertrace.h"

#include <stdio.h>

PROCESS(power, "powertrace example");
AUTOSTART_PROCESSES(&power);

PROCESS_THREAD(power, ev, data)
{
  static struct etimer et;
  static float t;

  PROCESS_BEGIN();

  /* Start powertracing */
  int n = 1; // 1 second reporting cycle

  powertrace_start(CLOCK_SECOND * n);
  printf("Ticks per second: %u\n", RTIMER_SECOND);

  while(1)
  {
    /* Delay 2-4 seconds and then perform some tasks */
    t = 2*((float)random_rand()/RANDOM_RAND_MAX) + 2;
    etimer_set(&et, CLOCK_SECOND*t);
     PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

    // Put your tasks here...
  }
  PROCESS_END();
}
```

To compile the source code, you need to activate powertrace app in the Makefile (see line 2 below).

```
1 CONTIKI_PROJECT = power
2 APPS += powertrace
3 all: $(CONTIKI_PROJECT)
4
5 CONTIKI = ../..
6 include $(CONTIKI)/Makefile.include
```

When running the program, the console will show power consumption statistics on the screen every second. An example of the output and the explanation are given as follow:

```
 2408 P 193.250 17 11309 578542 0 2574 0 2574 689 32078 0 144 0 144
 (radio 0.43% / 0.43% tx 0.00% / 0.00% listen 0.43% / 0.43%)
```

Parameter and value	Explanation
time = 2408 (or 2.408s)	clock time
rimeaddr = 193.250	rime address
seqno = 17	sequence number
all_cpu = 11309	accumulated CPU energy consumption

[1] See also: A. Dunkels, J. Eriksson, N. Finne, N. Tsiftes, “Powertrace: Network-level Power Profiling for Low-power Wireless Networks,” *SICS Technical Report T2011:05*, March 2011.

all_lpm = 578542	accumulated Low Power Mode energy consumption
all_transmit = 0	accumulated transmission energy consumption
all_listen = 2574	accumulated listen energy consumption
all_idle_transmit = 0	accumulated idle transmission energy consumption
all_idle_listen = 2574	accumulated idle listen energy consumption
cpu = 689	CPU energy consumption for this cycle
lpm = 32078	LPM energy consumption for this cycle
transmit = 0	transmission energy consumption for this cycle
listen = 144	listen energy consumption for this cycle
idle_transmit = 0	idle transmission energy consumption for this cycle
idle_listen = 144	idle listen energy consumption for this cycle

Powertrace tracks the duration of activities of a node being in each power state. In other words, the outputs show the fraction of time that a node remains in a particular power state. In the above example, there are 6 defined power states: CPU, LPM, TRANSMIT, LISTEN, IDLE_TRANSMIT, IDLE_LISTEN. We can group the record of a single trace into 4 groups:

- Clock time, Rime address, sequence number (they describe trace information)
- ALL_CPU, ALL_LPM, ALL_TRANSMIT, ALL_LISTEN, ALL_IDLE_TRANSMIT, ALL_IDLE_LISTEN (they show the accumulated number of ticks since the mote was booted)
- CPU, LPM, TRANSMIT, LISTEN, IDLE_TRANSMIT, IDLE_LISTEN (they show the number of ticks of this cycle which is between the last trace record and this trace record)
- Radio activities in percentages

We repeat the above example in the following:

```
2408 P 193.250 17 11309 578542 0 2574 0 2574 689 32078 0 144 0 144
(radio 0.43% / 0.43% tx 0.00% / 0.00% listen 0.43% / 0.43%)
```

Power states are measured with a hardware timer. The clock frequency is defined in `RTIMER_SECOND` (or 32,768 Hz for XM1000). In our program, we set the reporting cycle to be 1 second. With the 32,768 Hz of internal hardware clock speed, we expect 32,768 ticks. We can check that by summing all power states in the above example (i.e. those in green showing ticks per cycle) giving 33,055 ticks which are close to 32,768.

For radio activities, the record shows 3 pairs of percentages:

- Overall radio: ALL_TRANSMIT+ALL_LISTEN / ALL_IDLE_TRANSMIT+ALL_IDLE_LISTEN
- Transmit: ALL_TRANSMIT / ALL_IDLE_TRANSMIT
- Listen: ALL_LISTEN / ALL_IDLE_LISTEN

The percentage is calculated by taking the appropriate quantity divided by the sum of accumulated ticks of power states. For example, ALL_LISTEN percentage is calculated by:

$$\text{ALL\_LISTEN\%} = \frac{2574}{11309 + 578542 + 2574 + 2574} = 0.43\%$$

You may want to capture the output to a file. The following is how (for both running from a mote and running from Cooja):

- If you are **working on an XM1000 mote**, you may use the following command to divert the output to a file.

  ```
  $ make TARGET=xm1000 login > power.csv
  ```

  With the above command, instead of showing the statistics on the screen, the statistics are now saved into a file named 'power.csv'. As usual, to stop the process, press CTRL+C. The directory should now contain a file named 'power.csv' with the power consumption statistics. Using the saved data, you may even create a plot showing the power consumption of the node.

- If you are **working in Cooja simulator**, the “Mote Output” panel provides a feature for you to save the output data to a file. The feature can be found on the File menu of the panel.

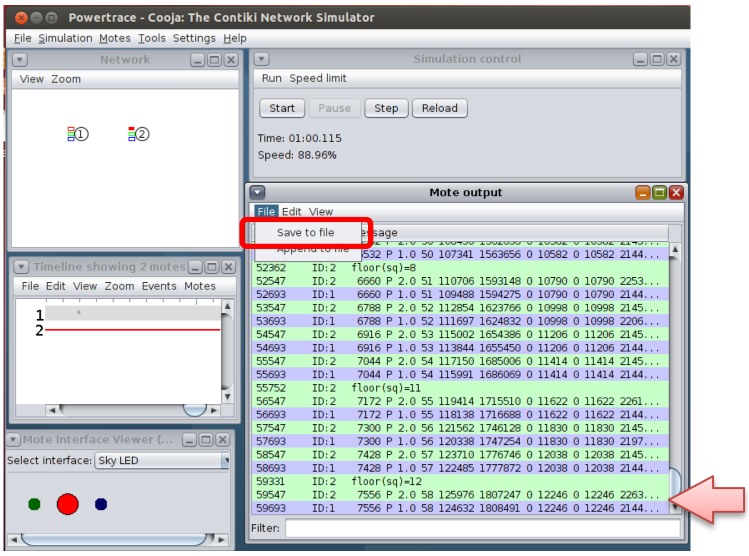
7.4 Exercise

The assignment of this lab is to implement some tasks and analyse the power consumption. First, navigate to “/home/user/contiki-2.6/surrey/S7”, you will find two source files: test1.c and test2.c.

- test1.c contains the skeleton code shown in the earlier section.
- test2.c extends the skeleton code by adding two tasks. You may replace them or add more tasks for your investigation.

You may now run each program, collect the powertrace statistics, and analyse the differences to understand the power consumption of a task. If you are working with a mote, you need to run the program separately. For Cooja simulation, we have prepared for you a simulation environment named “cooja_powertrace.csc”. In the simulation environment, we have created two motes running the two programs simultaneously. The “Mote Output” panel will contain the powertrace statistics of two motes. You can then save the output to a file for analysis.

The following screenshot shows the simulation running for about 1 minute:



```
Time: 01:00.115
Speed: 88.96%
```

```
       532 P 1.0 50 107341 1563656 0 10582 0 10582 2144...
52362  ID:2  floor(sq)=8
52547  ID:2   6660 P 2.0 51 110706 1593148 0 10790 0 10790 2253...
52693  ID:1   6660 P 1.0 51 109488 1594275 0 10790 0 10790 2144...
53547  ID:2   6788 P 2.0 52 112854 1623766 0 10998 0 10998 2145...
53693  ID:1   6788 P 1.0 52 111697 1624832 0 10998 0 10998 2206...
54547  ID:2   6916 P 2.0 53 115002 1654386 0 11206 0 11206 2145...
54693  ID:1   6916 P 1.0 53 113844 1655450 0 11206 0 11206 2144...
55547  ID:2   7044 P 2.0 54 117150 1685006 0 11414 0 11414 2145...
55693  ID:1   7044 P 1.0 54 115991 1686069 0 11414 0 11414 2144...
55752  ID:2  floor(sq)=11
56547  ID:2   7172 P 2.0 55 119414 1715510 0 11622 0 11622 2261...
56693  ID:1   7172 P 1.0 55 118138 1716688 0 11622 0 11622 2144...
57547  ID:2   7300 P 2.0 56 121562 1746128 0 11830 0 11830 2145...
57693  ID:1   7300 P 1.0 56 120338 1747254 0 11830 0 11830 2197...
58547  ID:2   7428 P 2.0 57 123710 1776746 0 12038 0 12038 2145...
58693  ID:1   7428 P 1.0 57 122485 1777872 0 12038 0 12038 2144...
59331  ID:2  floor(sq)=12
59547  ID:2   7556 P 2.0 58 125976 1807247 0 12246 0 12246 2263...
59693  ID:1   7556 P 1.0 58 124632 1808491 0 12246 0 12246 2144...
```

You can save the output traces to a file by using the “Save to file” option. The last two traces (pointed by the red arrow) show the powertrace report at the end of 1 minute of running time for motes 1 and 2 (we omit the radio activities):

```
59547 ID:2: 7556 P 2.0 58 125976 1807247 0 12246 0 12246 2263 30501 0 208 0 208 (radio … )
59693 ID:1 7556 P 1.0 58 124632 1808491 0 12246 0 12246 2144 30619 0 208 0 208 (radio … )
```

We shall focus on the accumulated values:

- Mote_ID2 recorded overall of 1,957,715 ticks, with 1,807,247 in low power mode.
- Mote_ID1 recorded overall of 1,957,615 ticks, with 1,808,491 in low power mode.

We can immediately compare the low power mode (LPM) of both motes:

$$\text{LPM\_MOTE}_2 = \frac{1807247}{1957715} = 92.31\%$$

$$\text{LPM\_MOTE}_1 = \frac{1808491}{1957615} = 92.38\%$$

Since Mote_ID2 has two additional tasks, its LPM is lower due to the need to run the tasks. However, since the two additional tasks are not intensive, the difference is very small. In this case, the difference between the two motes is 0.07%. Can you add some tasks to make the difference more obvious?

Pay attention that as you have learned earlier, if your application holds the CPU for over a second in a single process, the built-in watchdog will trigger a soft reset of the mote. You will see the mote keep booting (showing the initial message again and again). This usually happens when you run a loop that takes more than a second to complete. You can reduce the number of iterations to solve the problem.

Note: There is no standard solution for this exercise. You may implement any task of your choice to see the effect of power consumption.