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

Lab Session 8: Sensor Communications

Lecturers: Dr Mohammad Shojafar, Dr Chuan H Foh, Dr Ahmed Elzanaty

8.1 Introduction

The goal of this lab session is to understand how to achieve communication using two motes. We shall focus on using RIME package in Contiki.

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

8.2 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;
- How to read sensor data and print the output to the terminal;

In this lab session, you will learn:

- How to program in asynchronous code structure for communications.
- How to implement a simple communication handshake.

8.3 Exercise 1 – Understanding the Code

You task is to understand the structure of a communication client and test the communication with a server mote. You can navigate to "/home/user/contiki-2.6/surrey/S5" where you will find the following source codes. They are ready to be deployed.

- client1.c: it contains a client implementation
- client2.c: it contains another client implementation
- server.c: it contains a server implementation

We strongly recommend you to run the experiment in Cooja simulation first to see the expected outcomes before trying with XM1000 (if you have collected one). For this exercise, you should aim to develop full understanding of the code, and be familiar with coding in asynchronous style. This is particularly important to develop programs involving in communications.

8.4 Exercise 2 – Programming a Simple Handshake

Your task is to develop a simple communication handshake. We also recommend you to use Cooja simulation for the development and testing. You may deploy your code to two motes after you have tested your code in Cooja. You need to first do the following:

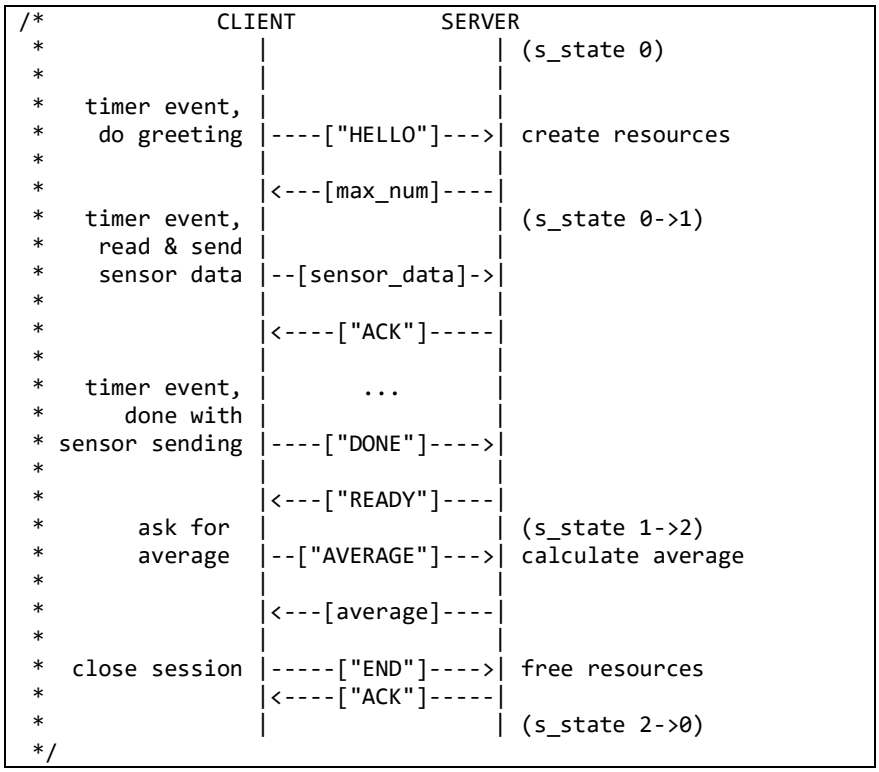
- server-extra.c: it contains the server-side implementation. To run Cooja, you should rename the file to server.c for Cooja to work. Alternatively, you can copy-n-paste the source code to the existing server.c source file.
- client1.c: you need to re-implement this client to describe the handshake. In our solution (to release later), client1 will submit five random numbers to the server.
- client2.c: you need to re-implement this client to describe the handshake. In our solution (to release later), client2 will submit five light sensor readings to the server.

The handshake is described as follows. The server constantly listens to a greeting message.

- When a greeting message arrives, the server creates resources and replies with a number specifying the maximum number of accepted samples.
- After which, the client periodically sends sensor data reading to the server. The server acknowledges each reception of data accordingly.
- When the client deems there is no further sample needed, the client issues a “DONE” message. When the server is ready to take further command, it replies with “READY”.
- The client can now request for a data processing service from the server. For illustration purpose, we assume that the server only offers calculating the average value of the submitted readings. In this case, the client issues a “AVERAGE” request message, and the server replies with the result.
- Finally, the client closes the session by issuing an “END” message, and an acknowledgement from the server confirms closing of the session.

The data structure used in the communication is given in the server-side code (see the provided source code in ‘server-extra.c’). Your task is to implement the client. Please also refer to the lecture slides for this scenario and techniques covered to do this exercise.

If your implementation is correct, your client should complete the handshake with the server and obtain the average value of the readings your client program submitted.



8.5 Working on XM1000 (skip this section if you cannot collect a mote)

You need two motes to test the communication. We encourage you to work in a pair. Please pay attention to the configuration of the communication. The channel, port number, and server address must be specified correctly.

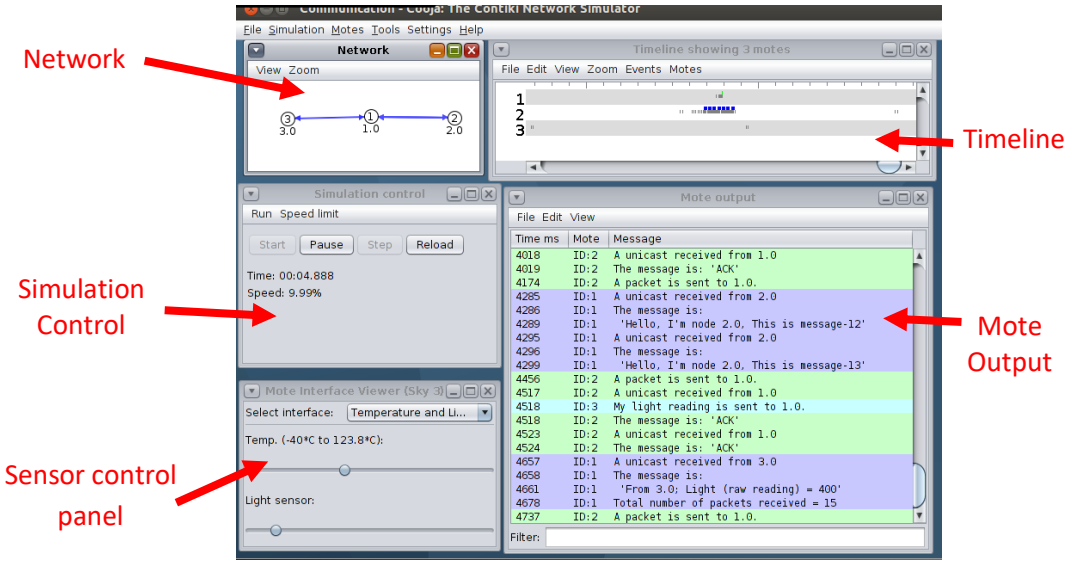
For a large scale application, it is more convenient to work in a simulation environment. You can create as many motes as possible with some running different codes. You are encouraged to use Cooja for this experiment.

8.6 Working with Cooja Simulator

You can also find the simulation environment "cooja_communication.csc" in the folder. Launch Cooja and load "cooja_communication.csc" to see the actions.

The simulation environment contains 5 panels:

- **Network**: it shows the network setup. As you can see, we have 3 motes, 2 as a client, the one in the middle is the server.
- **Timeline**: it shows the events happening during a simulation run. You can see communication activities from this panel.
- **Simulation Control**: it provides several buttons for you to start, pause, run step-by-step, or reload to restart the simulation. Notice that we reduce the simulation speed to 10% so that we can see the actions in the timeline easily.
- **Sensor Control Panel**: it provides two sliders for users to control the sensor inputs for Mote 3. You can use the slider to change the light intensity of the environment, and you should see the value being reported to the server.
- **Mote Output** (new!): it consolidates all console outputs from all motes to this panel. We set it such that the outputs from different motes are shown with different colors.



For Exercise 1, the codes are already provided, you can run Cooja to see the actions and understand the code.

For Exercise 2, you need to replace 'server.c' with the provided code given in 'server-extra.c' by simply renaming the files. You also need to implement a client to communicate with the server. You should then copy the client-side code to both 'client1.c' and 'client2.c'.