Lab Exercise 5 – Time Synchronisation and RF Ranging

Note: This exercise can be done in groups of two students.

Objectives:

- Coarse-grained time synchronisation using a global clock and coordinated universal time.
- Introduction to RF ranging.

Introduction:

This exercise introduces you to coarse-grained time synchronization and RF ranging for wireless sensor networks. Time synchronisation is an important feature of wireless sensor networks, particularly applications that collect extensive time series data from the field. Ranging is an important component of localization, which is important for most sensor network applications, such as tracking objects. The goals of this exercise are (i) to understand the basics of time synchronisation with a global clock and coordinated universal time and (ii) to understand the basics of ranging using RF signal strength on sensor tags.

Coordinated Universal Time (1.5 Marks)

You must implement a time synchronisation network using Coordinated Universal Time (UTC). Your network will consist of 2 Sensor Tags that must be synchronised using UTC with an rpl-board-router 'base' sensortag. Each sensortag has its own 'local' or system time (See Tutorial 4 for system time). A Python script (timesync.py) running on the PC will periodically transmit the current UTC timestamp (int32) in seconds using a UDP packet, via the base sensortag. You will have to modify the provided timesync.py script to connect to your sensortags. You must demonstrate that each sensortag (excluding the base) periodically (every 5 seconds) responds with the current UTC timestamp via UDP to the base sensortag.

You must complete the following functions and variables for each sensortag (not base):

- uint32 localutctime – contains current local UTC time, that is incremented every second.
- updateUtcTime(uint32 utctime) – update the sensortag's UTC time, using sensortag's system time.
- getUtcTimeFromLocalTime() – return the current sensortag's UTC time.

HINT:

1. Refer to Tutorial 4 for information on the system time.
2. Refer to udp-server example in contiki-examples/networking/udp-ipv6 (Tutorial 6). You may find below modifications useful because the provided example accepts UDP connections from 3001 source port only.

   ```
   // set NULL and 0 as IP address and port to accept packet from any node and any srcport.
   server_conn = udp_new(NULL, HTONS(0), NULL);
   ```

Synchronised Actuation (1 Marks)

Program sensortags to display synchronised LED blinking patterns. You can demonstrate the RG LEDs of all sensortags are changing their pattern once every few seconds at the same time. This exercise will show you the limitation of the coarse-grained time sync method above, as it only allows you to synchronize the time to about 1 second resolution. Please refer Tutorial 2 for Contiki LED interface.

RF Ranging (2.5 Marks)
You will implement simple linear ranging.

- Follow Tutorial 8, to understand how to measure and use the Received Signal Strength Indicator (RSSI).
- Create a pathloss propagation model between two sensor tags; measure RSSI (in dBm) over several distances using a provided rectangular grid (at least 10 distances), use numpy's least squares equation introduced in Tutorial 9 to estimate the unknown propagation model parameters.
- Display range between two nodes on a PC by using the propagation model to find range from RSSI.

HINT: One possible solution for this is using CoAP. The CoAP server (acts as the Beacon) provides RSSI information, while the CoAP client (acts as the sink node) GETs the RSSI information periodically from the server. For Server side, you may use the **er-rest-example** which we have used in Tutorial 7 as your building block. In case of the client, you can use the client implementation in the AioCoap (the Python CoAP library, downloadable from course website) to GET the RSSI from CoAP server. You may find ***clientGET_modified.py*** useful.

Marking Criteria (5 marks)

You must demonstrate the following criteria and be able to answer questions.

- For Coordinated Universal Time, you must demonstrate that each sensortag responds with the current UTC timestamp via UDP. You must complete the updateUtcTime(), getUtcTimeFromLocalTime() methods. – **1.5 marks**
- For Synchronized Actuation, you must demonstrate that nodes change their RG LED patterns in synchrony (1 second difference is ok). – **1 marks**
- For RF Ranging, you must demonstrate the calibrated pathloss model and be able to do simple linear ranging – **2.5 marks**

References:

AioCoap: https://aiocoap.readthedocs.io/en/latest/.

Notes:

- Please allocate radio channels as in Lab 4.
- All Python code should be run in the VM
- timesync.py transmits UTC timestamps to sensortag via UDP periodically, as well as listens for reply messages and prints them out. You should change "Address For Sensortag" in Line 44 to your sensorTag's IPV6 address.
- aiocoap-master is a Python CoAP library implementation you may use. Please read README.rst and doc/examples.rst. you may find clientGET-modified.py useful.