Internet of Things Projects - 2018/2019

Edoardo Longo
edoardo.longo@polimi.it

May 13, 2019

1 General Rules, Grading and Deadlines

Grade composition of the entire course:

- 26 out of 30 points are assigned based on the written exams
- up to 8α out of 30 points are assigned based on projects (parameter α depends on project delivery date, see below).

General rules:

- Projects are NOT mandatory. One student can decide not to take any project; in this case, the maximum grade he/she can get will be 26/30.
- Projects can be developed in groups of maximum 2 people.

Projects delivery deadlines and grading:

- September 10, 2019: $\alpha = 1$, this means that if you deliver your project by the 10th of September included, you can get the full points.
- December 31, 2019: $\alpha = 0.5$, this means that if your deliver the project after September 10, 2019, but before December 31, 2019 included you can get up to half of the points of the project.
- after December 31, 2019: $\alpha = 0$, this means that you don't get any additional points after this deadline.

The students willing to take the project assignment must choose among the project proposals listed in the following section or propose an original project, using the online form available at `https://forms.gle/jtFfD1peSa8bNMCu8` by **May 20, 2019**.

IMPORTANT 1: ONLY THE PROJECTS REGISTERED ON THE ONLINE FORM WILL BE CONSIDERED. LATE REGISTRATION WILL NOT BE ACCEPTED.

IMPORTANT 2: THE DELIVERY IS ONE AND ONLY ONE. WHEN YOU DELIVER THE PROJECT YOU CANNOT RE-DELIVER THE PROJECT FOR A BETTER GRADE.

IMPORTANT 3: REGISTRATION IS NOT BINDING. IF YOU REGISTER FOR A PROJECT AND THEN DECIDE AFTERWARDS YOU DON'T WANT TO DELIVER IT, THAT'S OK.

For original projects, write to `edoardo.longo@polimi.it` describing the project you intend to implement by **May 20, 2019**.

2 Proposed Projects

The following projects proposal are thought for being implemented with the tools seen during the hands-on lectures (TinyOS, Contiki, Node-Red, etc..). For projects related to WSN software programming, you are free to choose which O.S. to use.

You have to deliver the following items:

- Complete source code of the project preferably commented.
- Self-explanatory log file, showing that your project works. Try to be as detailed as possible when preparing the log file (i.e., use debug/print statements in all the crucial phases of your project).
- Project report (max. 3 pages), which summarizes your approach in solving the problem, including figures when needed. Don't include source code in the project report.

Projects will be evaluated based on the rationale and technical depth of the design choices, correctness of the source code and organization and clarity of the project report.

2.1 Project 1. Waste management system - up to 7 points

The waste management system is a network of smart bins connected to a garbage truck which picks up the trash.
You are requested to design, implement and simulate the waste management system using the following topology:

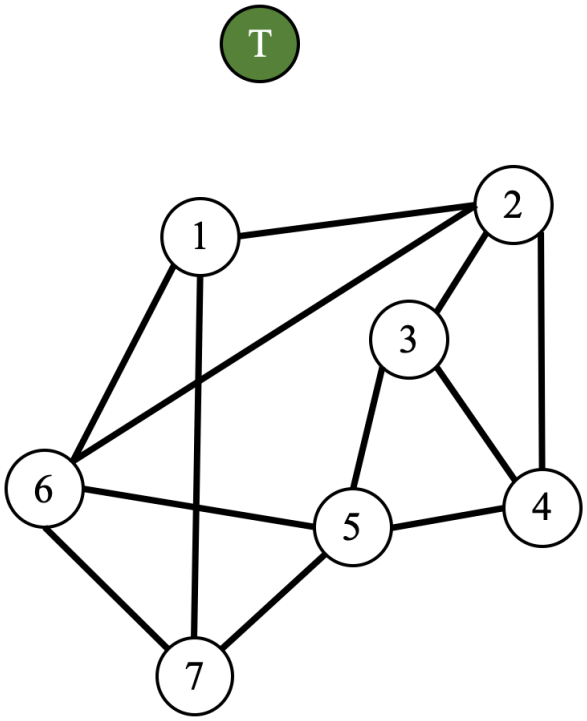

Figure 1: Network topology.

There are two different types of motes, the bins (white) and the garbage truck (green): the former has a fixed storable capacity of trash (100 units), the latter is assumed as infinite capacity. The bins are connected according the topology in figure 1, while the truck is connected with all the bins nodes even though the link are not shown in figure or the sake of simplicity. Moreover, all the motes have a pair of X, Y coordinates that are randomly loaded at booting.

The operation of the waste management system is as follows:

- **Trash phase**: this is a simulation of the action of taking out the garbage and it is performed only by the bin motes. Periodically, in a random interval between 1 and 30 seconds, a new trash is thrown in the bin. Such trash is still a random value, between 1 and 10 units. If the condition of the following point is fulfilled the new trash value is added to the current value.

- **Sensing phase**: After every action described above the bin checks its filling level. Three are three possible cases:

  - $level < 85 \rightarrow$ **NORMAL STATUS**: the garbage is correctly collected by the bin.
  - $level \geq 85 \wedge level < 100 \rightarrow$ **CRITICAL STATUS**: the garbage is correctly gathered, but the bin is almost full. Go on ALERT MODE.
  - $level \geq 100 \rightarrow$ **FULL STATUS**: the current trash cannot be collected by the bin and it must be moved to the closest bin. Go on FULL MODE.

- **Alert mode**: when the status of the bin is critical, it starts to transmit periodical *ALERT* messages to the garbage truck. That messages contain the position X, Y of the bin and its ID. When the truck receives the message it *travels* to the bin in a *travel time* directly proportional to the distance between the two motes. In order to simulate the trip time a timer is used. The truck computes the distance truck-bin and, after a t_{truck} time, it sends back to the bin a *TRUCK* message that simulate the arrival of the truck. Upon the receiving of the *TRUCK* message, the bin empties its filling level (set it to 0) and sends back an ACK.

- **Neighbor mode**: when the bin is completely full, it cannot store the trash anymore. The garbage that is still being generated must be sent somewhere else. For this purpose, the bin broadcasts a *MOVE* message, this special type of message can be read only by the other bin motes. When a neighbor receives a *MOVE* message, if its status is *NORMAL*, it replies with its X, Y coordinates after a t_{bin} time. The requester bin waits 2 seconds for collecting all the messages, then computes the

distance with the neighbors that have replied and sends the exceeding trash to its closest neighbor with a message. If no neighbor replies, it raises an error and delete that piece of garbage.

Some details of the implementation of the garbage truck:

- ★ the trash capacity is infinite;
- ★ if the garbage truck receives another message during the *travel time* it simply drops that message;
- ★ the coordinates at boot time are random;
- ★ when the garbage truck *moves* to a bin it acquire the coordinates of the bin itself.

Requirements

- Implement the prototype with the O.S. of your choice (including application logic, message formats, etc). The X, Y coordinates are random numbers generated at booting. The *travel time* (that is actually only a delay) between the motes is:

$$t = \alpha \times (Euclidean\ distance\ between\ the\ two\ motes) \tag{1}$$

  where

$$\alpha_{bin-bin} \ll \alpha_{bin-truck} \tag{2}$$

  Basically, the messages between the two bins are faster than the ones between the bins and the truck. The two alphas are of your choice.

- Simulate your implementation with two different topologies and two different noise traces. The first topology is the one in figure 1 and the second one is up to you, with at least 7 motes. As regards noise trace, one could be the topology seen at lesson, the other one is of your choice. You have to simulate your implementation with TOSSIM or with Cooja.

- Attach your simulation to TOSSIM-Live or Node-RED: the significant messages should be transmitted on the serial port and their output should be readable on the terminal or on the Node-RED dashboard.

2.2 Project 2. Smart Thermostat - up to 8 points

You are requested to design and implement a prototype of a smart thermostat using *Contiki (2.7)* operative system and simulating it in Cooja. The system is composed by a set of sensors, ideally placed in each room of a house, which manage and monitor the temperature. For controlling purposes, the thermostat communicates using *CoAP* with a local webpage based on the Node-RED dashboard.
The smart thermostat is composed by 4 or more simulated motes, plus a border router. This topology reproduces the one of the RPL border router example seen at lesson.

The thermostat is meant to:

- turn on/off the air conditioning, the heating and the ventilation unit;
- capture, store and visualize the temperature of each room;
- send an alert when the average temperature is above/below a certain threshold.

The operations of the thermostat are as follows:

- each node has by three LEDs. The blue LED represents the air conditioning, the red one represents the heating and the green one represents the ventilation. Turning on and off the LEDs activate/deactivate the specific status of the thermostat. The activation/deactivation is performed in CoAP using a POST request; the request sends back a confirmation response.
- the air conditioning and the heating are mutual exclusively, thus when the air conditioning is active the heating must be inactive and vice versa. Instead, the ventilation can be always active. If you want to turn on the heating and the air conditioning at the same time the POST sends back an error response.
- the temperature increases/decreases according the following rules:
  - after 20 seconds of operation, if the air conditioning is active the temperature decreases of 1 Celsius degree;

    - after 20 seconds of operation, if the heating is active the temperature increases of 1 Celsius degree;
    - the ventilation is a x2 multiplier of the current status, so the temperature increases/decreases of 2 Celsius degrees;

- each mote senses the temperature every 5 seconds and notify the value to the system;
- the client can request the current status of the thermostat (i.ie heating/air conditioning + ventilation) using a GET;
- the temperature at boot time is a random value between 10 and 30 Celsius degrees.

Client side, a Node-RED flow has to be implemented. The Node-RED flow is meant to:

- on the dashboard, visualize the current temperature of each room value observing the temperature resource;
- on the dashboard, visualize the current state of the thermostat in each room;
- turn on/off the different status of the thermostat using three buttons on the dashboard for each room and for all the house;
- send email alerts when the temperature is not in the desired range;
- publish to a *ThingSpeak*'s channel the average per minute of the temperature of the house via MQTT and visualize the temperature in a chart;
- publish to another *ThingSpeak*'s channel the average per minute of the temperature via MQTT, using a filed for each room;
- subscribe to the aforementioned *ThingSpeak*'s channel and visualize the same chart on the Node-RED dashboard.

Please include screens or a video of ThingSpeak and Node-RED into the technical report to show that everything is working correctly.

2.3 Project 3. Data collection with ThingSpeak - up to 4 points

In this project, you are required to implement a system for data collection using TinyOS or Contiki, Node-RED and ThingSpeak.

The requirements of this project are:

1. Create TWO simulated Wireless Sensors Networks (WSNs), each one with a SINK node and two sensor nodes. Both equipped with a temperature sensor with a humidity sensor (for TinyOS you can use the provided `TempHumSensorC.nc` component, whose operation is demonstrated in the `SensorTestAppC.nc` application or you can generate random, but realistic, values in Contiki). The two sensor nodes read the simulated data and forward it to the SINK node. Simulate the two WSNs in Cooja attaching each SINK node to a different Node-RED socket (use the SERVER serial socket tool in Cooja). Data should now be available in Node-RED.

2. Transmit the data from the simulation to ThingSpeak via Node-RED. Use *HTTP* for the WSN number 1 and *MQTT* for the second one. Each WSN should be linked to a different channel, and each sensor node in each WSN should be logged on a different Field.

3. Use Node-Red functions to read the data from ThingSpeak (use *HTTP* for WSN number 1 and *MQTT* for WSN number 2) and write a function to send an alert email when the average value of the temperature from the two WSNs exceed a predefined threshold.

Please include screens or a video of ThingSpeak and Node-RED into the technical report to show that everything is working correctly.