Computer science engineering

Internet Of Things

Project n°1

Waste management system

Fabio Codiglioni
919897 - 10484720

Alessandro Nichelini
949880 - 10497404

Professor:
Matteo Cesana

Teaching assistant:
Edoardo Longo

Academic year 2018/2019

POLITECNICO
MILANO 1863

Contents

1 Abstract

This document describes the implementation of our project for the IoT course.

We decided to implement the project number 1: *Waste management system*. We used Contiki as the operating system for the IoT devices and Cooja as the simulator. We also used Node-RED to make data available to the external world and to build a light dashboard to display the system status and the transmitted messages.

We implemented all requested requirements by writing two different firmwares: one for the bins and one for the truck. We manually set up in Cooja both the requested and our own node topology.

Full source code is available at
`https://github.com/fabiocody/CodiglioniNicheliniIoT`

2 Implementation design choices

2.1 Architecture

Each one of the two firmwares lives almost exclusively in a single `.c` source file (except for some utility functions in `toolkit.c`), and can be summed up by the following figures.

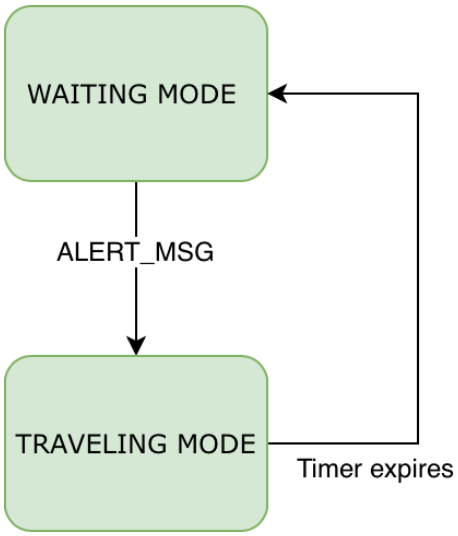

Figure 1: Truck FSM.

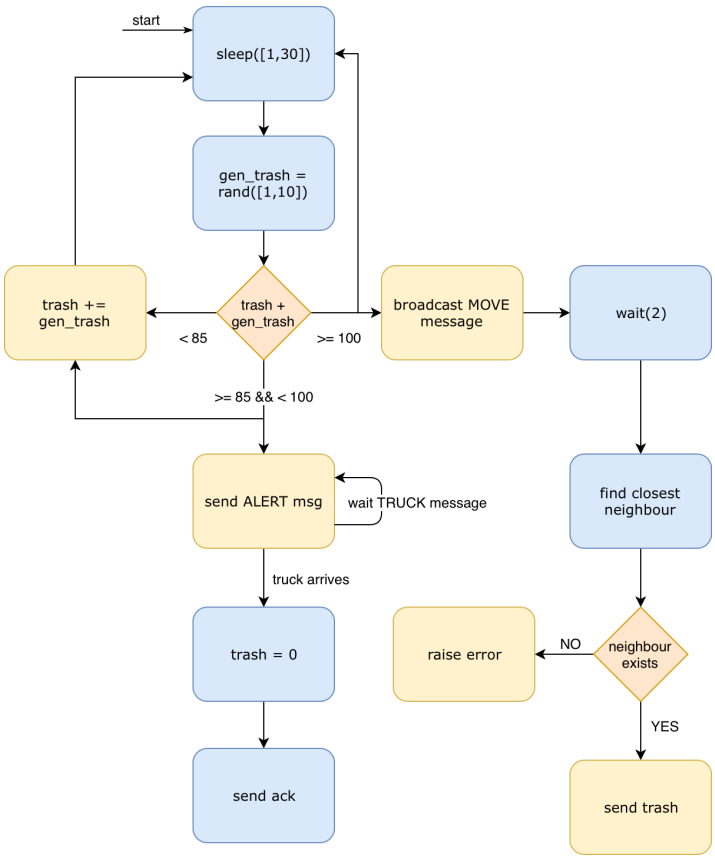

Figure 2: Bin flowchart.

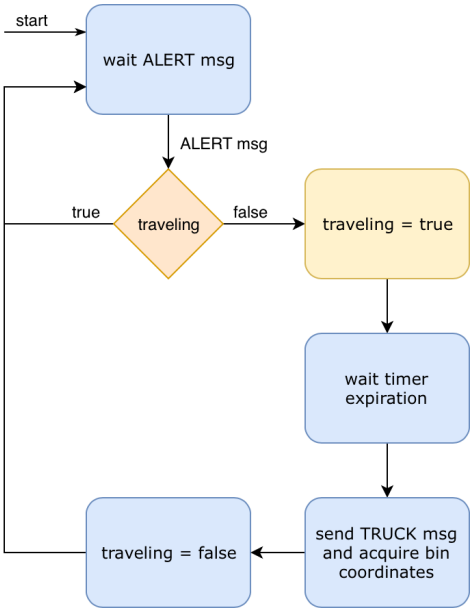

Figure 3: Truck flowchart.

2.2 Bin firmware

The bin firmware is composed of three main processes, one for each of the three operation modes:

- `trash_proc` – handles the periodic generation of the trash;
- `alert_mode_proc` – handles the period transmission of `ALERT` messages.
- `full_mode_proc` – handles the neighbor mode, in particular the broadcast of the `MOVE` message, the selection of the closest bin and the transmission of the exceeding trash to it.

There actually is a fourth process in the firmware, which is only used to reply to a `MOVE` message. We chose to place this routine in a separate process and not to leave it in the unicast callback in order not to lock the communication processes.

2.3 Truck firmware

The truck firmware is composed on a single process, which handles the traveling simulation (it sets up a timer and waits for its expiration) and the transmission of the `TRUCK` message.

2.4 Other implementation details

- We adopted two different transmission methods, both of them part of the native Rime communication stack:
  - **Broadcast** – The broadcast module sends packets to all local area neighbors with an a header that identifies the sender. No retransmission and acknowledgement implemented.
  - **RUnicast** – The runicast primitive uses acknowledgements and retransmissions to ensure that the neighbor successfully receives the packet, thus we were able to avoid implementing an explicit acknowledgement system (e.g., the ACK sent by the bin upon setting its trash to 0).
- The random coordinates generated by each node at startup are limited between 0 (inclusive) and 50 (exclusive).
- The time interval between two consecutive `ALERT` messages sent out by the same node is statically set to 10 seconds.
- $\alpha_{\text{bin}-\text{bin}} = 0.01$ and $\alpha_{\text{truck}-\text{bin}} = 0.75$
- For Node-RED dashboard we used some external plugins that have to be manually installed: `ui_led`, `node-red-dashboard` and `python3-function`.
- We used Cooja's success TX and RX ratio to simulate noise over the radio channel.

3 Simulation details

In both simulation, the truck is represented by node 8.

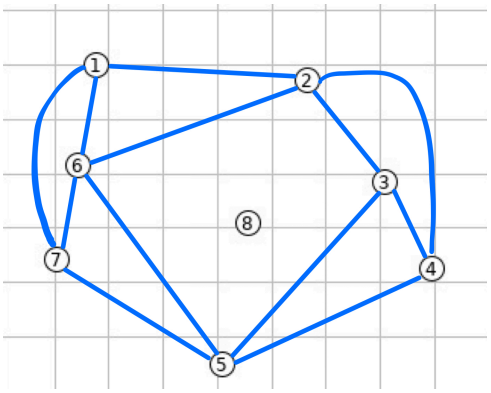

Figure 4: Simulation 1 topology. This is the *vanilla* simulation that implements the given topology. No noise trace is added.

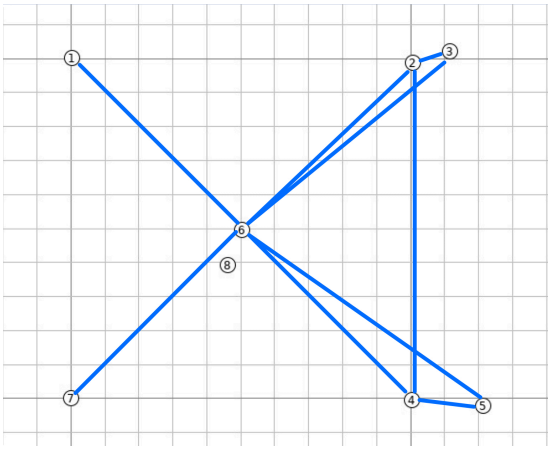

Figure 5: Simulation 2 topology. The truck can still communicate with all nodes. The other links are described in the figure. We also added a maximum TX/RX success ratio of 95% to simulate noise.