Keep Your Distance: IoT Project

Francesco Cecchetti 10497762
Daniel Caputo 10488023

Overview

In the following brief report, we will illustrate the work performed, the achieved results and the steps needed to develop "Keep Your Distance" project for the IoT course of 2020/2021 at Polytechnic of Milan.

The wireless network of motes has been developed using TinyOS, with Node-RED we wired the motes output with an IFTTT web hook in order to deliver an e-mail notification service.

The system has been simulated with Cooja, implementing 7 motes in the network.

An already configured simulation has been provided (KeepYourDistanceProject.csc).

The source code is commented to allow a better comprehension of the code sections.

TinyOS

In TinyOS we mainly developed the source code dividing it in three files, required for our network to properly work:

- KeepYourDistance.h: a header used to specify the structure of the messages exchanged between the motes of the network, the only field is the identifier of the mote;
- KeepYourDistanceAppC.nc: a NesC code that act as a configuration file to wire KeepYourDistance.nc with the components provided by TinyOS;
- KeepYourDistanceC.nc: a NesC code that implements the logic of each mote.

In KeepYourDistanceAppC.nc we defined the interfaces of the components used by the application:

- AMSenderC, AMReceiverC, ActiveMessageC: used to provide radio functionalities to motes allowing them to send and receive messages;
- Two TimerMilliC: the first timer (Timer1) is the "broadcaster" timer, with a frequency of 2 [Hz], it makes the mote broadcast the message containing the mote ID to every other mote that is in range of reception.
  The second timer (Timer2) has the task of checking if another mote, once in the range, has now left, going out of range. We decided to set 800 [ms] of absence of messages reception from another mote as an appropriate time interval to consider it out of range;
- SerialPrintfC, SerialStartC: used to printf debug and log messages.

In KeepYourDistanceC.nc we implemented the logic. The adopted approach is the following:

We setup two arrays of length 7, the first one, *neighbouring_motes*, represents the status of messages received count from the i-th mote of the network; the second array, *neighbouring_motes_prev*, represents the state of the array previously to a single or multiple messages arrival. The two arrays are both initialized to zero and is self-evident that the position corresponding to the current-execution mote will never be used or updated. Every 500 [ms] a message will be broadcasted from each mote containing its ID.

Upon message receival, the application will update the neighbouring_motes status, incrementing the counter of the related index of the array and logging the new counter value.

When a counter for the messages received from a specific mote reach 10, it means that the two motes have been in each other range for too long (5 [s]): a single alert message is logged from the mote with the smaller ID, and the counter value logging for the neighbour mote stops even if, under the hood, the actual counter keep increasing and changing its values.

The latter is an important aspect for the purpose of the *Timer2*, which has the job of detecting if a previously near mote is now out of range. This objective is achieved by checking for every other motes counters in *neighbouring_motes* if each value remained the same with respect to the *neighbouring_motes_prev*, which represents the array status during the last check. If the value of a counter changed (increased) it means that the mote i corresponding to the counter at i-th position of the array is still in range and sending messages.

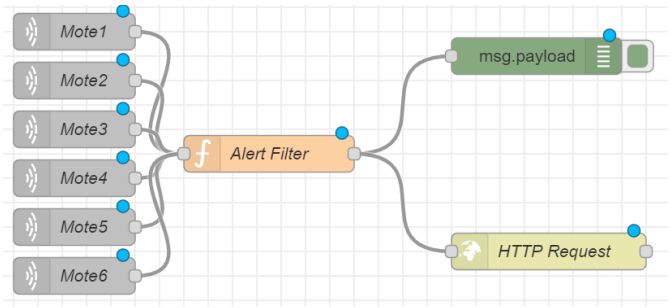
Otherwise, if the value remained the same during the 800 [ms] time interval, it means that mote i is out of range, and its counter can be reset to 0.

Then, *neighbouring_motes_prev* is updated to the current *neighbouring_motes*.

Node-RED

We then proceeded writing up a Node-RED flow that allowed us to redirect messages from the Cooja debugging, receiving them through 6 *tcp in* components. Being that only motes with ID from 1 to 6 will end up sending alerts, we only needed 6 *tcp in* modules.

A *javascript function* component will then filter out only the Alert messages parsing the two motes involved in the alert and reconstructing an appropriate message to be forwarded to an *http request* module. The latter will perform a POST request to a web hook on an IFTTT application that will deliver an e-mail to a configured address. This functionality will provide notifications of alert directly on a smartphone with an e-mail client.



Cooja Simulation

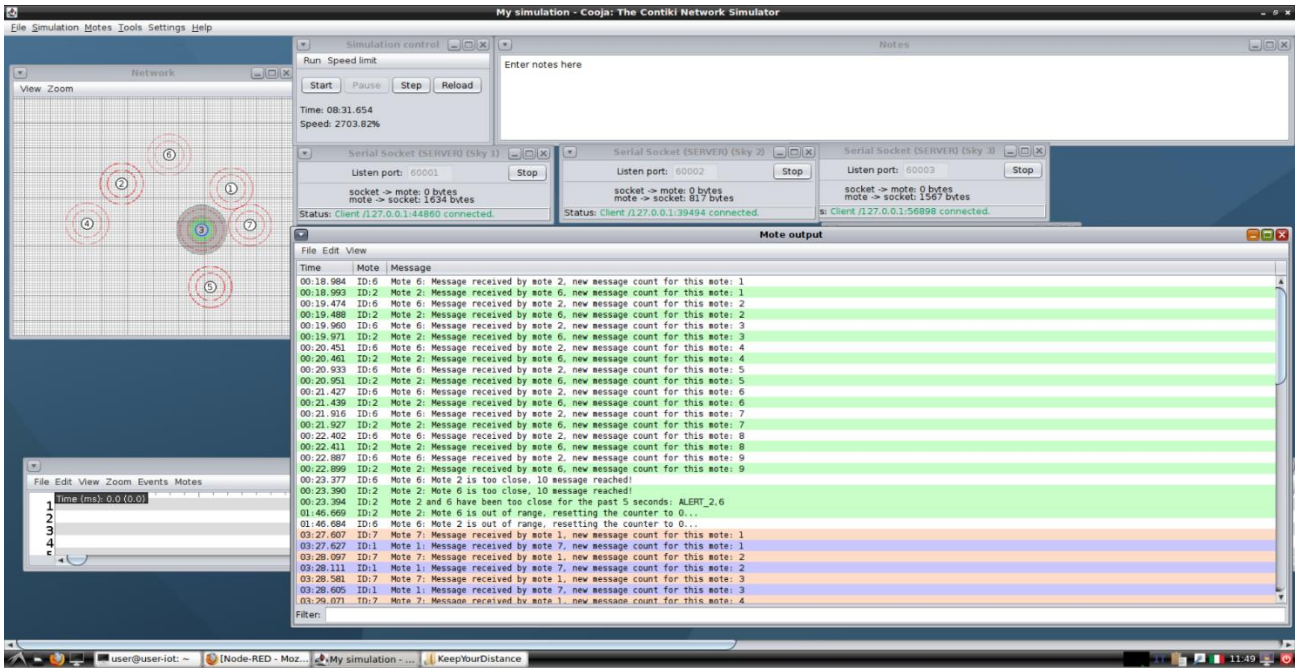
The simulation is run with 7 motes, each one setup with a server socket configured to communicate with the tcp in modules of the Node-RED previously described.

We run multiple simulations and took the liberty to make and provide a short screen recording of a simulation (Simulation.mp4) for which the log file has been exported and screenshots of the IFTTT notifications have been taken.

From the simulation log file provided, it is clearly visible how the application works: when 2 motes get in the range of each other, they both start counting how many messages they are sensing between them and log the increasing counter.

When 10 messages are reached both log it and stop counting messages. The mote with the lower ID also log an alert message, that can be forwarded to Node-RED and pass the Alert Filter module.

The same simulation also tested the case in which 3 motes get in each other range: the log shows that they all act accordingly to the expected behaviour we described so far.



Other interesting screenshots from the same simulation, including the IFTTT mail notifications and Node-RED debug are included in the provided material.