**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 PoliMi.

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 saved in KeepYourDistanceProject.csc.

**TinyOS**

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

* KeepYourDistance.h: an 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 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.  
Each mote, every 500 [ms] will broadcast a message containg 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 gone out of range. This objective is achieved by checking for each mote’s counter 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. 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.

A javascript function 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 functionaliy 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.