IoT Project - Keep Your Distance

Edoardo Saputelli – Alessandro Passoni
Politecnico di Milano

July 3, 2021

Code Implementation

The files of the delivered folder are the following ones: KeepYourDistance.h, KeepYourDistanceC.nc, KeepYourDistanceAppC.nc.

- **KeepYourDistance.h:** it is the actual structure of the message sent between the motes during their interaction.
  It is composed of 2 fields, contained in a struct: the id of the sender mote and the timestamp of the message;
- **KeepYourDistanceAppC.nc:** here the components needed by the application are declared and wired. The inserted components are divided in the ones for the boot, for the radio communications, for the timer and for the printing tasks.
- **KeepYourDistanceC.nc:** this file is divided in two parts. The first part is the *module* one, where all the interfaces are declared. The second part is the *implementation* one, where we actually implemented the interfaces. In this part, after having declared some variables (described below), we specified how the application has to behave when certain events are triggered, which are:
  - the boot of the application;
  - the start of the radio communication, where each mote starts a timer that fires every 500 ms;
  - the stop of the radio communication;
  - the expiration of the timer, where each mote sends a message in order to notify its presence;

- the reception of a packet, where the node updates its variables to keep track of the number of consecutive messages received by each other mote. In the situation of 10 consecutive messages, it raises an alert.

At the beginning of the *implementation* section, we declared some variables:

- *timestamp*: a unique value for each message sent by a certain mote, which is incremented after each timer expiration;
- *action_timestamps[]*: an array to store the last received timestamps from each other mote;
- *num_of_msg[]*: an array to store the number of consecutive messages received from each other mote.

The underlying idea is that, upon the reception of a message, first of all the mote checks if it is the first message at all or if it is a message immediately after an alert, in both cases sent by a certain mote: in this situation the counter for the sender mote is set to 1, since it could be the first of the 10 possible consecutive messages. Moreover, the timestamp of the message is updated in the specific cell of *action_timestamps[]* array.

Alternatively, there are two possibilities: in the first case, after verifying that the just received message has not a consecutive timestamp with respect to the relative stored one, the timestamp is updated and the counter is reset to 1, because it can be the first of a possible series of 10 consecutive messages.
Instead, in the second situation, since the just received message has a consecutive timestamp with respect to the previous stored one, the timestamp is updated and the relative counter is incremented by one.

Forward to NodeRed

On NodeRed, we used the following nodes (in order):

- *tcp node*: one for each mote. It is used to establish a communication between the mote on Cooja and the socket on NodeRed, through a specific port for each mote;

- *switch node*: it is used to filter all the messages sent by the motes, keeping just the proximity alert ones;
- *function node*: in this node we extract the identifier of the close mote, in order to set up the parameter for the web request on IFTTT;
- *http request node*: this node is used to trigger the event through an url that contains the related event's name, which is directly specified, and the parameter set up in the previous node.
- *debug node*: it is used to show on NodeRed if the http request was successful.

In order to verify the correct work of the mote identifier parameter extraction, we inserted a parallel path with a *debug* node. In addition, the output forwarded to *debug* nodes are sent also to a *file* node in order to generate the log of the execution.

Notification delivery

With the aim to send a notification to mobile phones, we implemented an Applet on the IFTTT website.

- IF: whenever the Applet receives a request for a *proximity_event* due to a mote alert,
- THEN: there is the delivery of a notification, with the following format:

{{EventName}}: you are too close to mote {{Value1}}!

where EventName is *proximity_event* and Value1 is the parameter representing the ID of the close mote.

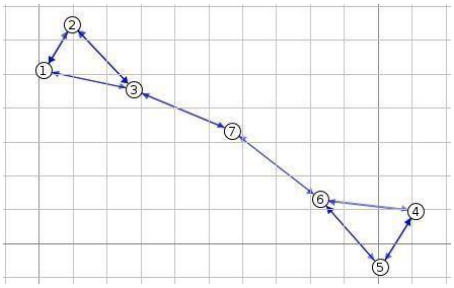
Tested configurations

We simulated the following two configurations (both using 7 motes):

- All motes start far away from each other, without reaching other motes with their messages. Then they are organized in a ring-like structure, where every mote reaches just two other motes (one on its left and one on its right) and they are kept in this configuration until an alert between the

mote and the two adjacent ones is sent as expected. Then, all of them are put in the range of all the other motes and they all exchange an alert with each other. In the end, the last disposition consists in moving separately away two of the motes (2 and 6 in the log file), showing the interruption of the communication between these two single motes and the others, which keep communicating as before.

- All motes start far away from each other, without reaching other motes with their messages. Then they are organized in two sub-networks (as presented below), with a mote placed between the two sub-networks, in order to simulate a possible real disposition.



After having exchanged the alert messages, motes keep exchanging messages for another little period of time, and then the sub-networks are broken, leaving just two couples of motes exchanging their messages. Then, in order to test the nonconsecutive exchange of messages among nodes, we recompose one of the two sub-networks. Finally, in order to test a completely new communication, a mote from one of the two initial subnetworks is placed in the range of a mote of the other one, with the purpose to let them exchanging messages for the first time.

All these dispositions were useful to evaluate all the interesting scenarios, like real possible configurations, motes that don't exchange messages, motes that exchange alerts multiple times in a row, motes that resume the communication after having interrupted it, clearly in addition to the simple exchange of messages between two or more motes.