**Report for the project activity:**

**Keep your distance**

**For the course of Internet of Things at Politecnico di Milano**

A.A. 2019-2020

**Group members:**

Marco Ceran ID: 952673

Urszula Szczepanska ID: 950021

The files that we delivered are:

* TinyOS source files (KeepYDC.nc , KeepYDAppC.nc and the Makefile)
* Node-Red source file (KeepYD json file)
* A log file showing the simulation of the motes’ behaviour in Cooja (Main.exe.csc)
* A zip archive containing various screenshot of the IFTTT application and notification

The goal of the project is to design and implement an applications that alerts you when two TinyOS motes are close to each other by using Cooja to simulate the motion of the motes, and Node-Red and IFTTT to process the information sent by the motes and consequentially alerting the user. The motes are sending messages in broadcast at a frequency of 2 Hz, so that every mote sends messages continuously, but messages are only received if two (or more) motes are in the proximity (reception) zone. When a mote receives a message it stores the value of the mote that sent the message in a variable called ‘last’ that is used for this purpose, before writing into the ‘last’ variable, its content is written in the ‘secondlast’ variable; in this way each mote can store at each time the ID of the last two motes that it communicated with (if needed this <messages from the same mote the ‘last’ and ‘secondlast’ variables are not rewritten. The ‘last’ and ‘secondlast’ variables are initialized to 0 in order to have a value that is different from each mote ID, so that they always get written at the first reception of a message, regardless of the ID of the sender mote. The structure of the messages is simple, the only field is a 8 bit unsigned integer variable called ‘topic’ that stores the value of the sending mote’s ID.

The interfaces used in the source code are:

* Boot, to boot the system;
* Receive, to receive messages;
* AMSend, to send messages;
* SplitControl (renamed AMControl with ‘as’), to control the ActiveMessageC component;
* Packet, to handle packets;
* Timer<TMilli>, to time the transmission of packets;

So when the device is booted the two storage variables are initialized to 0, and then the timer is set to fire periodically each 500ms, when the timer fires a new message containing the TOS\_NODE\_ID is sent in broadcast. This operation repeats periodically, and nothing else happens until a mote enters the proximity area of another one. When this happens both the motes that are now close to each other store the other node’s ID in the ‘last’ variable, and its previous content gets written in ‘secondlast’, after that each mote prints a string message that says ”Mote #X in proximity area”, with the number of the other mote’s ID in place of the X, and this string gets forwarded to Node-Red. As the minimum number of motes that need to get in touch to have a successful reception of a message is two, two is also the minimum number of strings that are forwarded to Node-Red every time two or more motes get in touch.

Before starting work with Node-Red in order to present our application its needed to add in Cooja six motes and for each one create a separate Serial Socket (SERVER). Using TCP request node all output from simulation in Cooja are transmitted to Node-Red to begin the data processing. For the purpose of this task, the possibility of installing to the palette additional nodes was used. The first added one is a function node called deduplicate and second one is an output node called ifttt out. Each of outputs that come from tcp block is a stream of string delimited by \n. The first function block is removing bad characters which arises from compatibility problem which appear if the printf is use between TinyOS and Cooja. Then deduplicate function has been applied which is responsible for filtering duplicate messages and shows only the first instance, and until the set time expires (in this case 30 seconds) it stops informing about repeated messages. At the same time, this function does not interrupt notifications in the case of another mote approaching the proximity area. The second function block is used to set the Event Name needed to set the right URL for IFTTT Webhook and also change the message payload dividing it into three variables values 1-3 displayed in the IFTTT notification. At the end the filtered messages are shown in the debug sidebar tab and in the IFTTT application via webhook and notification in app using API key. The Output IFTTT node will trigger an event with the msg.payload data. A separate event has been created for each mote to increase the transparency of information transmission.

The nodes used in the Node-Red are:

* TCP In, node that will send a message into flow for each packet it receives from a tcp port connection
* Function, node to write JavaScript function to run against the messages being received by the node.
* Deduplicate, node that removes duplicate messages.
* IFTTT Out, node to connect to ifttt Webhooks channel
* Debug, node that displays selected message properties in the debug sidebar tab

Moving on to the If This Then That app to build an applet at first its needed to create the conditional part of the script to select what needs to happen for this script to be executed. For this app a webhook is used as a trigger service, which is basically just a web address that Node-RED installation needs to contact to trigger the IFTTT applet every time the Maker service receives a web request to notify it of an event. With the trigger set up next its needed to assign an IFTTT service to trigger. When the URL (webhook) for this applet is contacted an notification in the IFTTT app is received . The Value 1-3 are variables that were define in Node-RED and send them whenever the event is triggered in our flow.