IoT project - Keep your distance

Luca Ferraro (10748116), Fabio Losavio (10567493),
Bernardo Camajori Tedeschini (10584438)
https://github.com/LucaFerraro/IoT-HomeChallenge

A.Y. 2019/2020

Contents

List of Figures

Listings

1 Introduction

The aim of this project is designing and implementing a software prototype for a social distancing application. To do this, we have used:

- **TinyOS** for the development of the software that should run on the motes involved.
- **Node-Red** as IoT platform.
- **IFTTT** for providing a notification service to the users.
- **Cooja** for the simulation of our software.

In the following sections, we'll describe first of all how to setup the environment for simulating and testing the software we have developed. Then, we provide a brief description of our solution to the request of the project.
All the material we have used to develop our project along with all the home challenges we have done during the semester is available in this GitHub Repository: https://github.com/LucaFerraro/IoT-HomeChallenge

2 Setup

The first requirement to try this project is having an **IFTTT** account opened. This because in the project, when two nodes are too close one to the other, a notification is net via email through IFTTT. To make this possible, it's necessary to create an Applet in our IFTTT account. The steps to create it are (once registered to IFTTT):

- Creating a new applet.
- In the section **This**, select the voice *Webhooks* and to trigger the reception of a web request.
- Name the event *Proximity_alarm*; the name is fundamental for the proper working of the application.
- In the section **That**, choose *Email* and insert the email where you want to receive notifications.

Now we can start the simulation.

First of all, we need to compile the code. To do this, we need to navigate from the terminal to the directory of the project and run the command `make telosb`: the result is the creation of a folder (named **build**) containing the executable file that is needed to add the motes inside the simulation framework **Cooja**.

The next step is opening **Cooja**. We need to create a new simulation and add as *Skymotes* the number of motes we want to use in the simulation.
Pay attention that the number of nodes can't be greater than the parameter **NUMBER_OF_MOTES** that is specified in the file `Project.h` (it's been initialized to 5 from us, but it can be changed).
Also, the number of motes cannot be in general greater than 14, that is the number of TCP connections we have added in the **NodeRed** flow we have prepared. In case the simulation would require a greater number of motes, the file `flows.json` must be changed adding the needed *tcp in* blocks.
Once added the motes, it's necessary to start a socket on each on of them (each mote acts as a server); the ports on which the sockets must be opened are (if the file `flows.json` is not changed from the user of the program) the ones from 60001 to 60014.

At this point, we need to start **NodeRed**. To do this we need to:

- Open the terminal.
- Type the command `node-red`.
- Open a browser and connect to the local host (*http://127.0.0.1:1880/*).

- Import the file `flows.json`.
- Deploy the flow (check if the tcp blocks corresponding to the port on which the socket has been opened in Cooja is actually connected to that socket).

Note that, by default, the block *Req Params* is connected to the block *Web Req*, that sends a mail through **IFTTT** to one of ours mail. In order to let a user receive the notification as a mail, the block *Req Params* must instead be connected to the block *INSERT YOUR IFTTT KEY TO GET EMAILS*, which must be filled with the key of the mail registered in **IFTTT**. (All of this is explained also in the block *IFTTT instructions* inside the Nodered flow.)

Now everything is ready to start the simulation: just go again in Cooja and click on the *Start* button. By moving the motes around, it's possible to see the messages exchanged by the motes both in Cooja and in NodeRed.

3 Program description

3.1 TinyOS

The primary aim of the Project is to understand if two motes are too close each other; in order to obtain this, we have thought to exploit the *Receive Signal Strengh Indication (RSSI)* from the packets received.
Therefore, we include in the **ProjectAppC.nc** file all the components to extract the power received from the packet according to the specific radio used.
The temporal accuracy of the simulation was *500 ms*, so each mote will send a packet in broadcast at the expiration of a timer, that happens every *500 ms*.
The problem was then that if an *Alarm* was solicited, the very next *Alarm* should have been relaunched at least after 1 minute (to avoid useless repetition of warnings). So, we have created in the **ProjectC.nc** file a structure called *moteMemory[NUMBER_OF_MOTES][2]*: for each mote, this structure keeps memory of the time passed between the last *Alarm* with every other mote. The structure is initialized in the function **AMControl.startDone** with all zeros; as we can easily understand, it has as many rows as thew number of motes we want to consider in the simulation and two columns:

- The first column is a boolean, in particular the first column of row i is 1 at mote j if mote j has received a message from mote with ID = i less than a minute ago.
- The second column has a value different from 0 only if the first column of the same row is 1; in this case, it indicates the time elapsed from the last notification sent in multiple of the timer duration (so *500 ms*). Thus, this column of the row i is updated when the timer fires and the first column of the same row is 1.

At this point, we only need to define how it is updated the first column of each row. This is done looking at the *RSSI* of a received packet from mote i: if it's higher than the *POWER_THRESHOLD* and if the boolean value of the row i of the matrix is 0 (meaning that it has passed at least one minute from the last *Alarm* sent), we show off the motes that start the event and the RSSI, and send a message to the NodeRed application.

3.2 NodeRed

In the **NodeRed** application, we first connect to the **Cooja** simulation; this can be done using the *TCP_IN* block in which we listen for the TCP requests at the default ports (60001, 60002, 60003, ...); we have put up to 14 connections, but this can be easily upgraded increasing the number of blocks.
Then, the payload of a received packet is filtered in order to retrieve correctly the information about:

- The source and the destination motes, the ones that have triggered the the alarm since too close.

- The power of the message exchanged by them (that is a function of the distance between the two motes).

Subsequently, we adapt the format of the payload to be suitable for the IFTTT syntax.
Finally we send an HTTPS request to the IFTTT account specified, that will trigger a command to send an email with the data of the *Alarm*.