Internet of Things

Final Project (2020/2021)

Mattia Siriani (10571322) - Matteo Visotto (10608623)
@ Politecnico di Milano

Contents

Informations

Github repository: https://github.com/matteovisotto/IoT2021-FinalProject
Telegram channel: https://t.me/IoT_IFTTT
Logs: file loglistener.txt in the repository. It contains messages received by motes and alarms.

1 Introduction

For the final project we decided to choose "Keep your distance" which request to implement a software for social distancing, using TinyOS, Node-Red, Cooja and IFTTT.
In our implementation we provide the TinyOS code for 5 motes, everyone with a frequency of 2 Hz.
The message, which is broadcasted to report a mote presence, contains the mote ID (as requested) and also a progressive message number used for the synchronization between received messages.

The Node-Red flow is configured to receive data in single socket and relay the two motes ID to the IFTTT platform.

In our project we decided also to create a Telegram channel where notifications managed by Node-Red are relayed using both telegram official API and IFTTT telegram bot, more details are provided in Node-Red and Conclusion sections.

2 TinyOS

In our implementation the number of motes is defined by the global constant: "N_MOTES", which is used to set up the following arrays length:

- uint8_t motes: which contains the number of consecutive messages received by each mote.
- uint16_t packets: which contains the last received packet number of each mote.

The message, sent in broadcast by the motes, contains the mote ID (equals to TOS_NODE_ID) used also as index for the arrays (subtracting 1 to it) and the packet ID which is the value of an internal counter that represents the number of broadcasted messages. In order to reduce the probability of false alarm during the mote life, we decide to use the counter as uint16_t data type instead of uint8_t because in this way we could have a false alarm each $(2^{16} - 1)$ messages.
When a mote receives a message, the received moteID is used firstly to access the "packets" array, where the saved value is compared with the received packet ID; then if the difference between the two values is exactly 1, it means that this is a consecutive message and so the corresponding value memorized in "motes" array is increased by 1, otherwise the corresponding value in motes array is set to 1 (because in this case the message is not consecutive and we have to start to count from 0 again plus the message just received). In both cases the value in "packets" array is updated with the received one.
When the counter in "motes" array is equals to 10 it means that 10 consecutive messages have been received by the mote, so an alert is printed out and the counter itself is set to 0 again in order to start a new cycle.
The value printed by the mote is a JSON string with the following format:

$$\{"my\_id" : X, "other\_id" : Y\}$$

where X is the ID of the mote that triggered the alarm, while Y is the ID of the mote that sent the 10 consecutive messages.
The choice of using a JSON string allow us to perform a parsing of the data once it is sent to Node-Red.

3 Cooja

In cooja we have created 5 Sky Motes to perform our simulation.
Before starting it, we have placed all the motes far away in order to avoid any massages exchange (Figure 3.1) and then we started our simulation.

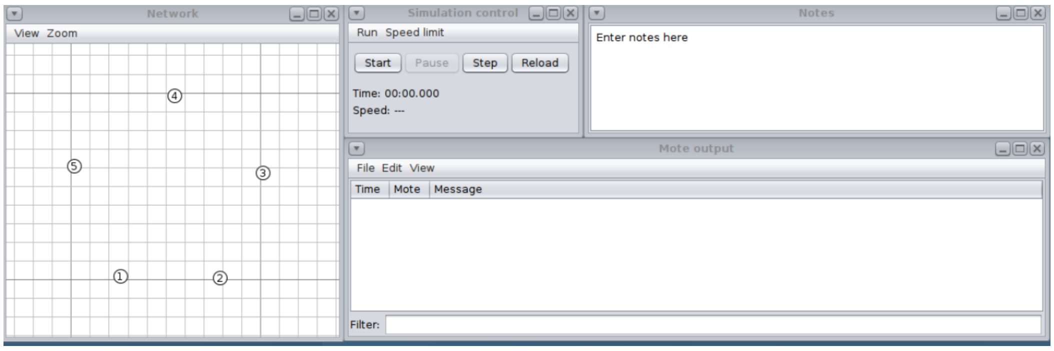
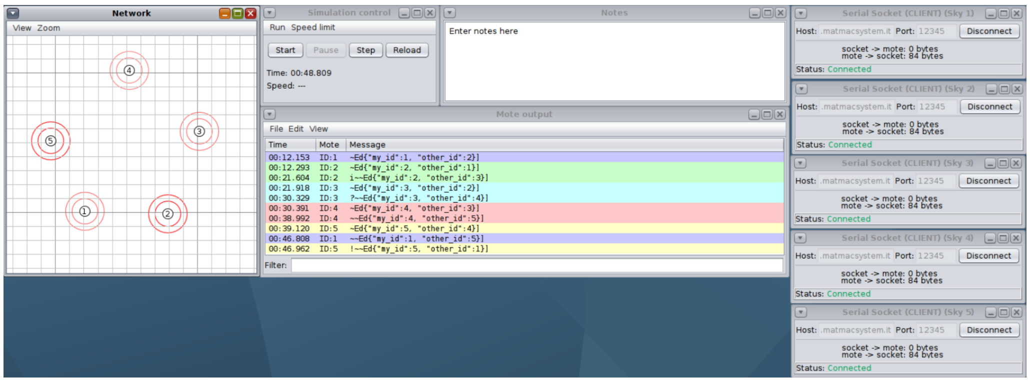


Figure 3.1: Initial Cooja configuration

First of all we have opened only one TCP socket connection to our Node-Red flow in order to test the output of a single mote approaching one by one the other four motes and checking the output provided both directly in Cooja and in Node-Red.
Then, as a second test, we have opened a TCP connection for each mote to the same TCP port (12345) and we have tested each mote with the nearest, approaching the motes involved to let them start communicate (Figure 3.2). In this test, as we expected, we have received two different messages every time, because both the motes are communicating so they both send information through the TCP port.

As a third test we have left a TCP connection open for each mote to the same TCP port, in this way the Node-Red flow takes care of all the notification relaying them to our IFTTT account and Telegram channel, placing all the motes one over the other (Figure 3.3).

As a final test, we have checked that the counter will be reset if we move away a

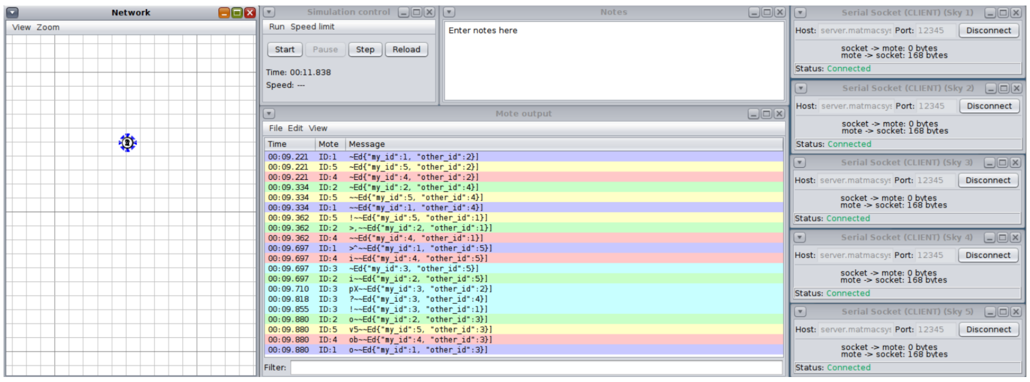


```
Time: 00:48.809
Speed: ---

Time       Mote  Message
00:12.153  ID:1  ~Ed{"my_id":1, "other_id":2}]
00:12.293  ID:2  ~Ed{"my_id":2, "other_id":1}]
00:21.604  ID:2  i~~Ed{"my_id":2, "other_id":3}]
00:21.918  ID:3  ~Ed{"my_id":3, "other_id":2}]
00:30.329  ID:3  ?~~Ed{"my_id":3, "other_id":4}]
00:30.391  ID:4  ~Ed{"my_id":4, "other_id":3}]
00:38.992  ID:4  ~~Ed{"my_id":4, "other_id":5}]
00:39.120  ID:5  ~Ed{"my_id":5, "other_id":4}]
00:46.808  ID:1  ~~Ed{"my_id":1, "other_id":5}]
00:46.962  ID:5  !~~Ed{"my_id":5, "other_id":1}]
```

```
Serial Socket (CLIENT) (Sky 1) … (Sky 5)
Host: matmacsystem.it  Port: 12345
socket -> mote: 0 bytes
mote -> socket: 84 bytes
Status: Connected
```

Figure 3.2: All communicating, single approach



```
Time: 00:11.838
Speed: ---

Time       Mote  Message
00:09.221  ID:1  ~Ed{"my_id":1, "other_id":2}]
00:09.221  ID:5  ~Ed{"my_id":5, "other_id":2}]
00:09.221  ID:4  ~Ed{"my_id":4, "other_id":2}]
00:09.334  ID:2  ~Ed{"my_id":2, "other_id":4}]
00:09.334  ID:5  ~~Ed{"my_id":5, "other_id":4}]
00:09.334  ID:1  ~~Ed{"my_id":1, "other_id":4}]
00:09.362  ID:5  !~~Ed{"my_id":5, "other_id":1}]
00:09.362  ID:2  >,~~Ed{"my_id":2, "other_id":1}]
00:09.362  ID:4  ~~Ed{"my_id":4, "other_id":1}]
00:09.697  ID:1  >^~~Ed{"my_id":1, "other_id":5}]
00:09.697  ID:4  i~~Ed{"my_id":4, "other_id":5}]
00:09.697  ID:3  ~Ed{"my_id":3, "other_id":5}]
00:09.697  ID:2  i~~Ed{"my_id":2, "other_id":5}]
00:09.710  ID:3  pX~~Ed{"my_id":3, "other_id":2}]
00:09.818  ID:3  ?~~Ed{"my_id":3, "other_id":4}]
00:09.855  ID:3  !~~Ed{"my_id":3, "other_id":1}]
00:09.880  ID:2  o~~Ed{"my_id":2, "other_id":3}]
00:09.880  ID:5  v5~~Ed{"my_id":5, "other_id":3}]
00:09.880  ID:4  ob~~Ed{"my_id":4, "other_id":3}]
00:09.880  ID:1  o~~Ed{"my_id":1, "other_id":3}]
```

```
Serial Socket (CLIENT) (Sky 1) … (Sky 5)
Host: server.matmacsy  Port: 12345
socket -> mote: 0 bytes
mote -> socket: 168 bytes
Status: Connected
```

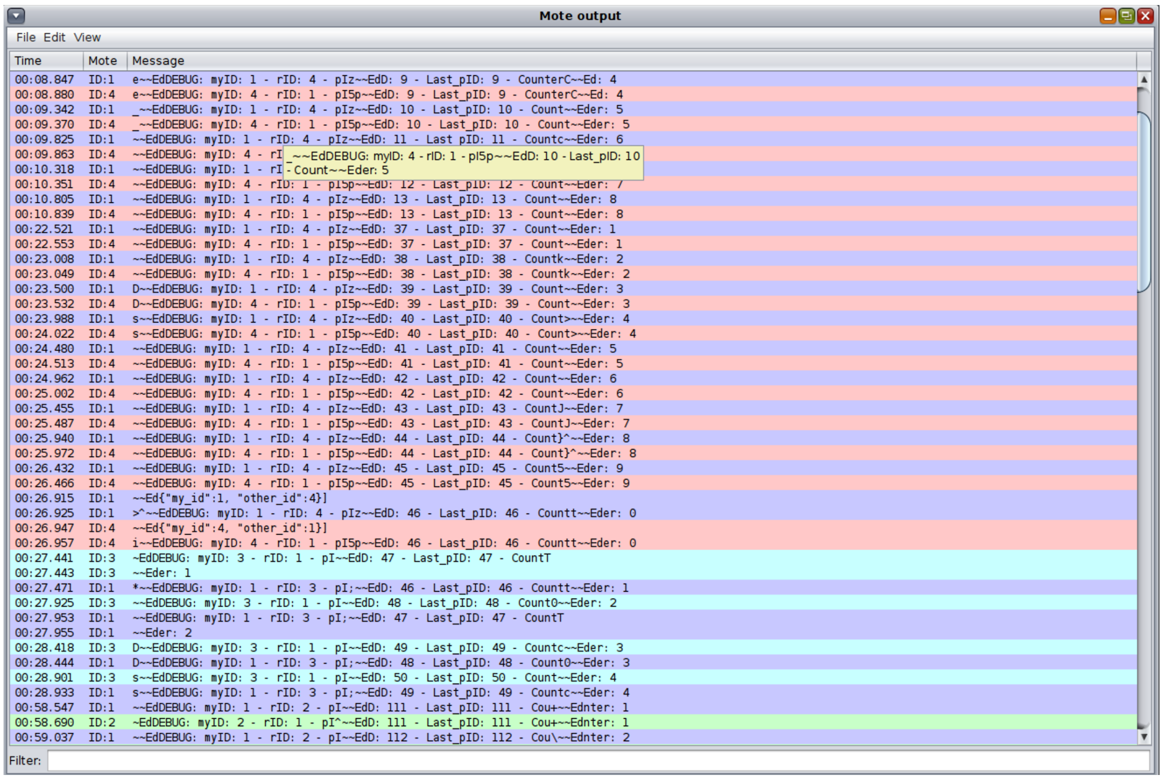
Figure 3.3: All motes sending at the same time

mote during the period in which it's receiving the 10 messages (as example if mote 1 has received only 7 messages). As expected the mote will restart counting messages from 1 (0 + the message just received) when the motes come closer again and will trigger the alarm only after 10 consecutive messages (Figure 3.4).
In the figure 3.4 we can also notice that:

- The counter start from 0 (which means 0 message received) and trigger the alarm when it reaches the value 10 (the 10^{th} value is not available in the logs because the counter is reset)
- When an alarm is triggered the counter will be immediately reset to 0
- When a mote goes away before reaching 10 consecutive messages, the counter is correctly reset.

images are also provided as logs in our repository.

In the next section we will describe different types of possible configuration in Node-Red to relay the traffic of each mote to a different mobile.



```
Time       Mote  Message
00:08.847  ID:1  e~~EdDEBUG: myID: 1 - rID: 4 - pIz~~EdD: 9 - Last_pID: 9 - CounterC~~Ed: 4
00:08.880  ID:4  e~~EdDEBUG: myID: 4 - rID: 1 - pI5p~~EdD: 9 - Last_pID: 9 - CounterC~~Ed: 4
00:09.342  ID:1  _~~EdDEBUG: myID: 1 - rID: 4 - pIz~~EdD: 10 - Last_pID: 10 - Count~~Eder: 5
00:09.370  ID:4  _~~EdDEBUG: myID: 4 - rID: 1 - pI5p~~EdD: 10 - Last_pID: 10 - Count~~Eder: 5
00:09.825  ID:1  ~~EdDEBUG: myID: 1 - rID: 4 - pIz~~EdD: 11 - Last pID: 11 - Countc~~Eder: 6
00:09.863  ID:4  ~~EdDEBUG: myID: 4 - rI _~~EdDEBUG: myID: 4 - rID: 1 - pI5p~~EdD: 10 - Last_pID: 10
00:10.318  ID:1  ~~EdDEBUG: myID: 1 - rI - Count~~Eder: 5
00:10.351  ID:4  ~~EdDEBUG: myID: 4 - rID: 1 - pI5p~~EdD: 12 - Last_pID: 12 - Count~~Eder: 7
00:10.805  ID:1  ~~EdDEBUG: myID: 1 - rID: 4 - pIz~~EdD: 13 - Last_pID: 13 - Count~~Eder: 8
00:10.839  ID:4  ~~EdDEBUG: myID: 4 - rID: 1 - pI5p~~EdD: 13 - Last_pID: 13 - Count~~Eder: 8
00:22.521  ID:1  ~~EdDEBUG: myID: 1 - rID: 4 - pIz~~EdD: 37 - Last_pID: 37 - Count~~Eder: 1
00:22.553  ID:4  ~~EdDEBUG: myID: 4 - rID: 1 - pI5p~~EdD: 37 - Last_pID: 37 - Count~~Eder: 1
00:23.008  ID:1  ~~EdDEBUG: myID: 1 - rID: 4 - pIz~~EdD: 38 - Last_pID: 38 - Countk~~Eder: 2
00:23.049  ID:4  ~~EdDEBUG: myID: 4 - rID: 1 - pI5p~~EdD: 38 - Last_pID: 38 - Countk~~Eder: 2
00:23.500  ID:1  D~~EdDEBUG: myID: 1 - rID: 4 - pIz~~EdD: 39 - Last_pID: 39 - Count~~Eder: 3
00:23.532  ID:4  D~~EdDEBUG: myID: 4 - rID: 1 - pI5p~~EdD: 39 - Last_pID: 39 - Count~~Eder: 3
00:23.988  ID:1  s~~EdDEBUG: myID: 1 - rID: 4 - pIz~~EdD: 40 - Last_pID: 40 - Count>~~Eder: 4
00:24.022  ID:4  s~~EdDEBUG: myID: 4 - rID: 1 - pI5p~~EdD: 40 - Last_pID: 40 - Count>~~Eder: 4
00:24.480  ID:1  ~~EdDEBUG: myID: 1 - rID: 4 - pIz~~EdD: 41 - Last_pID: 41 - Count~~Eder: 5
00:24.513  ID:4  ~~EdDEBUG: myID: 4 - rID: 1 - pI5p~~EdD: 41 - Last_pID: 41 - Count~~Eder: 5
00:24.962  ID:1  ~~EdDEBUG: myID: 1 - rID: 4 - pIz~~EdD: 42 - Last_pID: 42 - Count~~Eder: 6
00:25.002  ID:4  ~~EdDEBUG: myID: 4 - rID: 1 - pI5p~~EdD: 42 - Last_pID: 42 - Count~~Eder: 6
00:25.455  ID:1  ~~EdDEBUG: myID: 1 - rID: 4 - pIz~~EdD: 43 - Last_pID: 43 - CountJ~~Eder: 7
00:25.487  ID:4  ~~EdDEBUG: myID: 4 - rID: 1 - pI5p~~EdD: 43 - Last_pID: 43 - CountJ~~Eder: 7
00:25.940  ID:1  ~~EdDEBUG: myID: 1 - rID: 4 - pIz~~EdD: 44 - Last_pID: 44 - Count}^~~Eder: 8
00:25.972  ID:4  ~~EdDEBUG: myID: 4 - rID: 1 - pI5p~~EdD: 44 - Last_pID: 44 - Count}^~~Eder: 8
00:26.432  ID:1  ~~EdDEBUG: myID: 1 - rID: 4 - pIz~~EdD: 45 - Last_pID: 45 - Count5~~Eder: 9
00:26.466  ID:4  ~~EdDEBUG: myID: 4 - rID: 1 - pI5p~~EdD: 45 - Last_pID: 45 - Count5~~Eder: 9
00:26.915  ID:1  ~~Ed{"my_id":1, "other_id":4}]
00:26.925  ID:1  >^~~EdDEBUG: myID: 1 - rID: 4 - pIz~~EdD: 46 - Last_pID: 46 - Countt~~Eder: 0
00:26.947  ID:4  ~~Ed{"my_id":4, "other_id":1}]
00:26.957  ID:4  i~~EdDEBUG: myID: 4 - rID: 1 - pI5p~~EdD: 46 - Last_pID: 46 - Countt~~Eder: 0
00:27.441  ID:3  ~EdDEBUG: myID: 3 - rID: 1 - pI~~EdD: 47 - Last_pID: 47 - CountT
00:27.443  ID:3  ~~Eder: 1
00:27.471  ID:1  *~~EdDEBUG: myID: 1 - rID: 3 - pI;~~EdD: 46 - Last_pID: 46 - Countt~~Eder: 1
00:27.925  ID:3  ~~EdDEBUG: myID: 3 - rID: 1 - pI~~EdD: 48 - Last_pID: 48 - Count0~~Eder: 2
00:27.953  ID:1  ~~EdDEBUG: myID: 1 - rID: 3 - pI;~~EdD: 47 - Last_pID: 47 - CountT
00:27.955  ID:1  ~~Eder: 2
00:28.418  ID:3  D~~EdDEBUG: myID: 3 - rID: 1 - pI~~EdD: 49 - Last_pID: 49 - Countc~~Eder: 3
00:28.444  ID:1  D~~EdDEBUG: myID: 1 - rID: 3 - pI;~~EdD: 48 - Last_pID: 48 - Count0~~Eder: 3
00:28.901  ID:3  s~~EdDEBUG: myID: 3 - rID: 1 - pI~~EdD: 50 - Last_pID: 50 - Count~~Eder: 4
00:28.933  ID:1  s~~EdDEBUG: myID: 1 - rID: 3 - pI;~~EdD: 49 - Last_pID: 49 - Countc~~Eder: 4
00:58.547  ID:1  ~~EdDEBUG: myID: 1 - rID: 2 - pI~~EdD: 111 - Last_pID: 111 - Cou+~~Ednter: 1
00:58.690  ID:2  ~EdDEBUG: myID: 2 - rID: 1 - pI^~~EdD: 111 - Last_pID: 111 - Cou+~~Ednter: 1
00:59.037  ID:1  ~~EdDEBUG: myID: 1 - rID: 2 - pI~~EdD: 112 - Last_pID: 112 - Cou\~~Ednter: 2
```

Figure 3.4: Counter check logs

4 Node-Red

In the Node-Red part we receive the JSON string through socket from the Cooja simulator, containing the ID of the mote that triggers the alarm and the ID of the mote which has generated the event, this is done using the TCP socket in block.
The other blocks we have used in our flow are:

- A function block used to sanitize the incoming string from the random chars added by the Cooja's printf.
- A JSON block to parse the string into a JSON object, in the message payload.
- A function block used to prepare the message for posting data in the IFTTT platform. We added in msg.event the IFTTT applet id, while in the payload the motes ids as value1 and value2.
- A http post request block used to send data to IFTTT platform using the URL: *https://maker.ifttt.com/trigger/{{event}}/with/key/API_KEY*, where the event is directly retrieved from msg.event variable and the API_KEY substituted with the one provided in our account.

This type of configuration relay alarms on a unique account and it can receive data from one or more motes. If we want to split the notification for each mote involved in the simulation we just have to use multiple node-red flow or adding a split block connected to multiple function + http post blocks (one for each mote/account pair).

In order to manage Telegram notifications the following blocks have been added:

- A function block connected between json and IFTTT http post has been used to prepare data for a second IFTTT applet using the same structure as before but changing the event name.
- A function and a http request blocks are used to prepare and send the same data to Telegram API endpoint. However, a new http request block is needed

due to the fact that the URL structure is different requiring parameters like API Key, chat id and text previously prepared in the function block.

The final flow is the following (Figure 4.1)

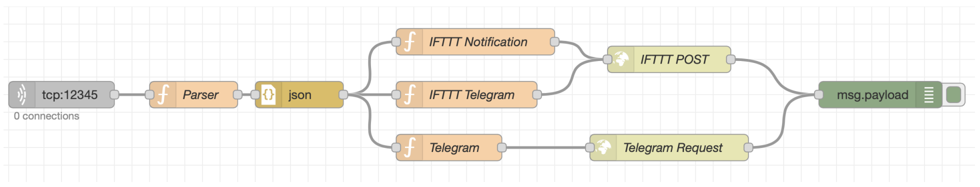


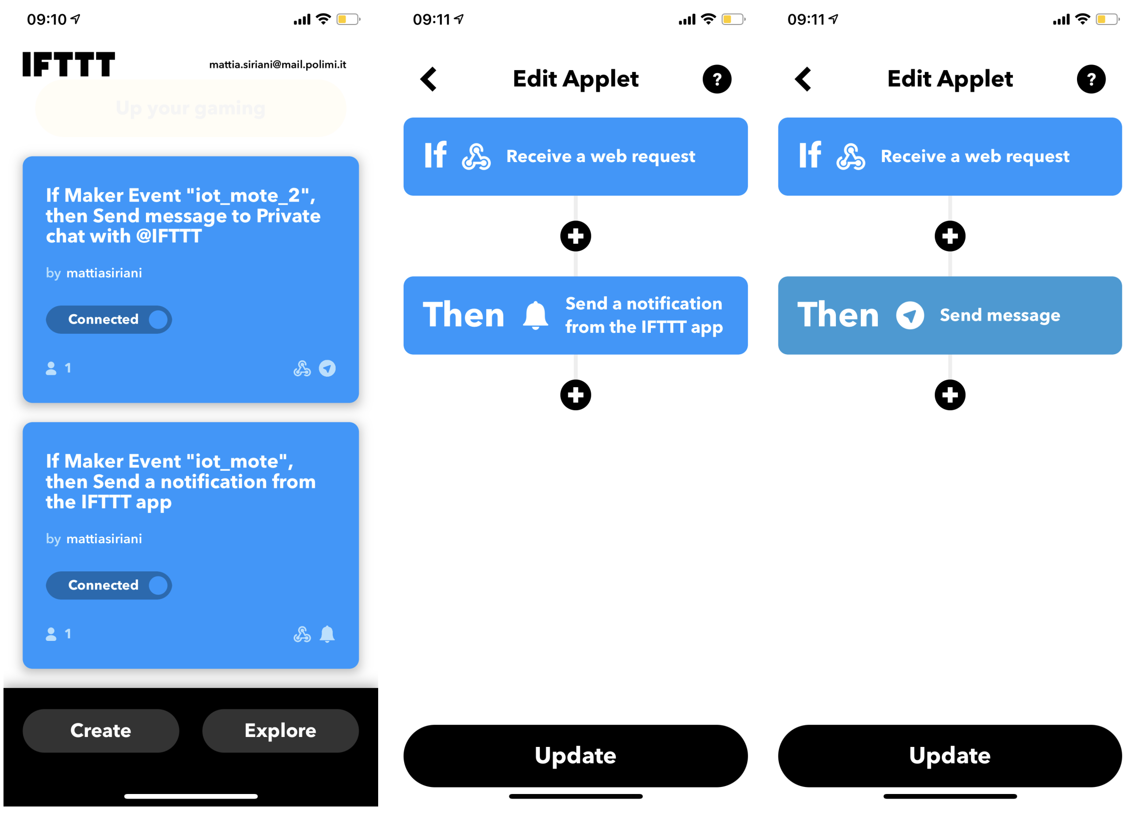
Figure 4.1: Node-Red Flow

The final debug node, that receives all the payloads from requests, is used just to check if the HTTP Request is successful.

5 IFTTT

We created two applets, the first one triggering a notification to the phone, while the second sends a message to our telegram channel. For the first applet we have used the first two trigger services below, while for the second applet we have used the first and the third trigger service below:

- The Webhooks trigger service to acquire the two motes ids from Node-red through an http request.
- The Notification trigger (Figure 5.1b) service to send a notification to our mobiles when an event is triggered (an event corresponds to a http request, generated by an alarm triggered by a mote).
- The Telegram trigger service (Figure 5.1c) to send a message to our Telegram channel.



If Maker Event "iot_mote_2", then Send message to Private chat with @IFTTT

by mattiasiriani

Connected

1

If Maker Event "iot_mote", then Send a notification from the IFTTT app

by mattiasiriani

Connected

1

If Receive a web request

Then Send a notification from the IFTTT app

If Receive a web request

Then Send message

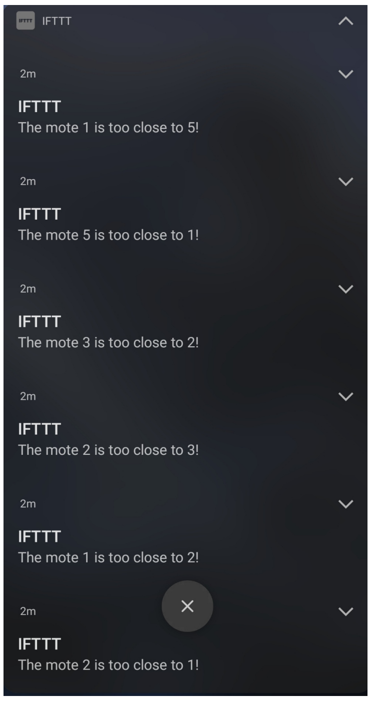
(a) Applets (b) Notification (c) Telegram

Figure 5.1: IFTTT configurations

6 Conclusion

We tested our project with different number of motes and different configurations and it works reliably.
Furthermore, also Telegram notifications are working both using direct API and IFTTT applet, for that reason the messages received are always doubled but we decided to keep both configurations.

We finally provide some images about notification on our mobile


IFTTT

2m
IFTTT
The mote 1 is too close to 5!

2m
IFTTT
The mote 5 is too close to 1!

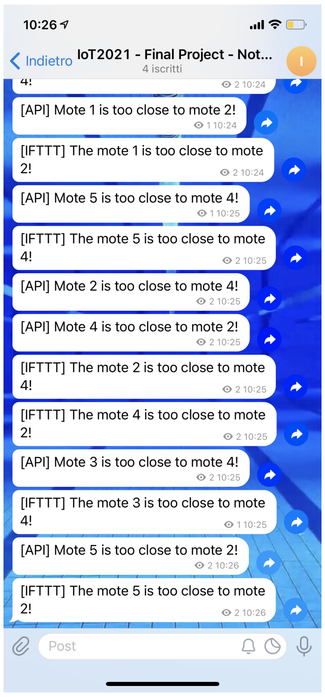
2m
IFTTT
The mote 3 is too close to 2!

2m
IFTTT
The mote 2 is too close to 3!

2m
IFTTT
The mote 1 is too close to 2!

2m
IFTTT
The mote 2 is too close to 1!

(a) IFTTT App notification


10:26

Indietro IoT2021 - Final Project - Not...
4 iscritti

4! 2 10:24
[API] Mote 1 is too close to mote 2! 1 10:24
[IFTTT] The mote 1 is too close to mote 2! 2 10:24
[API] Mote 5 is too close to mote 4! 1 10:25
[IFTTT] The mote 5 is too close to mote 4! 2 10:25
[API] Mote 2 is too close to mote 4! 2 10:25
[API] Mote 4 is too close to mote 2! 2 10:25
[IFTTT] The mote 2 is too close to mote 4! 2 10:25
[IFTTT] The mote 4 is too close to mote 2! 2 10:25
[API] Mote 3 is too close to mote 4! 2 10:25
[IFTTT] The mote 3 is too close to mote 4! 1 10:25
[API] Mote 5 is too close to mote 2! 2 10:26
[IFTTT] The mote 5 is too close to mote 2! 2 10:26

Post

(b) Telegram Notification

Figure 6.1: Notifications

6.1 Final notes

Due to the bug in the printf function the final code include only the printing of the alert while the printf used to generate the log file has been commented in the source code. We have done this to simplify the Node-Red flow because if we also send the debug messages through the socket we would need a lot of JavaScript code to clear the input from the useless part and also, more important, sending logs through socket, in a real situation, produces an extra, useless, energy consumption while only the important data must be sent. Obviously in the simulation it doesn't matter but a good solution would be to split data produced by motes into logs (to save locally) and alarm to send through the network, however this is not possible in Cooja.