Keep Your Distance

Di Cesare 10529764, Giacometti 10524482

TinyOS implementation

keepYourDistance.h

In this file there's the specification of the messages exchanged by the motes, the structure is the following:

```
nx_struct keepyourdistance_msg {

	// the id of the mote which is propagating the message
	nx_uint8_t id;
	nx_uint8_t seq_n;

} keepyourdistance_msg_t;
```

Where id is the ID of the sending mote, and sequence number is the value of the internal counter of the mote.

keepYourDistanceC.nc

In this file we have the core of the project as it describes the behaviour of the motes.

By default we support the existence of 5 motes, but this hard limit can be changed by modifying the appropriate global variable in this file (MAX_MOTES).

The mote signals its presence every 500ms to each of its neighbours, meanwhile it receives signals from other motes indicating their presence. Every time the timer is fired, we increment by one an internal counter, which starts from 0.

The mote keeps track of two values for each one of the other motes, we have:

```
uint8_t last_counter[MAX_MOTES];
uint8_t start_counter[MAX_MOTES];
```

Last_counter is the value of the last sequence number received from mote i, where i is the index in the array. Similarly start_counter is the first sequence number received from mote i. We update every time the value of last_counter.

If we receive a sequence number which is not the successor of last_counter, we reset start_counter and last_counter to the sequence number received.

If last_counter reaches a value of start_counter+10, we reset both values and we trigger an alert which states that the mote is too close to the triggering one.

keepYourDistanceAppC.nc

Here is the configuration of the components of the mote, note that we used SerialPrintf in order to get rid of the nasty garbage characters which appeared when using the standard Printf.

Node-Red

From Cooja we connected each mote to a different tcp socket, the port is set to 6000X where X is the ID of the mote.

Each message is forwarded to Node-Red via these sockets and will processed by various function blocks.

setUserPort()

This node checks if the message is an “interesting” one and not random log. Moreover is adds two fields to the msg containing the source port and the source ID.

dataParsing()

This block parses the payload of the message and gets the source ID (the mote who sent the alert) and the target ID (the mote too close to the source).

settingParameters()

This node prepares the aforementioned values for the IFTTT interface.

Switch Node

This node routes the message to the correct IFTTT interface.

POST to IFTTT

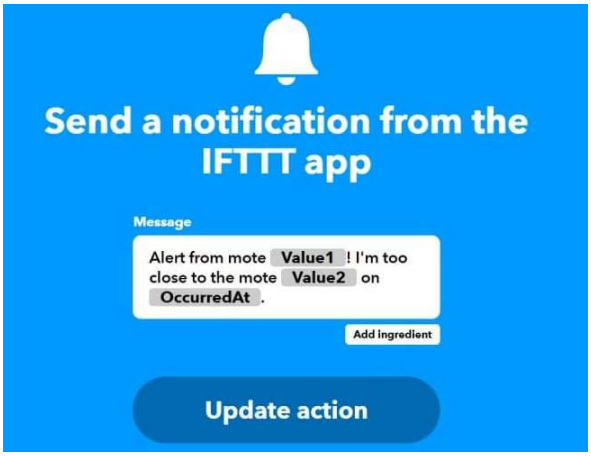
There are 5 similar blocks, each of them emulates the interface to the mobile phone of the users (the owners of the motes). In our practical implementation they all forward the messages to the same IFTTT applet, but in theory each node should sent the alert to each different physical person (with different IFTT keys).

Logging

Along the processing path there are some logging nodes which will work together to produce a log file which keeps track of what’s happening.

IFTTT

Each user should configure its own Webhook IFTTT applet and the corresponding key will be used in Node-Red. Note that we provide two values, the first one is the source mote and the last one (the user) and the second one is the other mote which is too close.



LOG details

We provide two different logs, one comes from the Cooja simulation and contains all the events happening in the motes, the other log comes from Node-Red and it’s relative to the alerts sent by the motes.

Cooja simulation

An example of this log can be found in the file `cooja-log.txt`, each message contains the timestamp, the ID of the sender mote and a log text. We log every time a mote fires its timer, whenever a mote sends a message and whenever a mote receives a message (together with the content of the message).

Node-Red simulation

An example of this log can be found in the file `NR-log.txt`. Each message contains the ID of the source mote and a brief text. We log every time an alert reaches Node-Red, the content of the data parsed, the

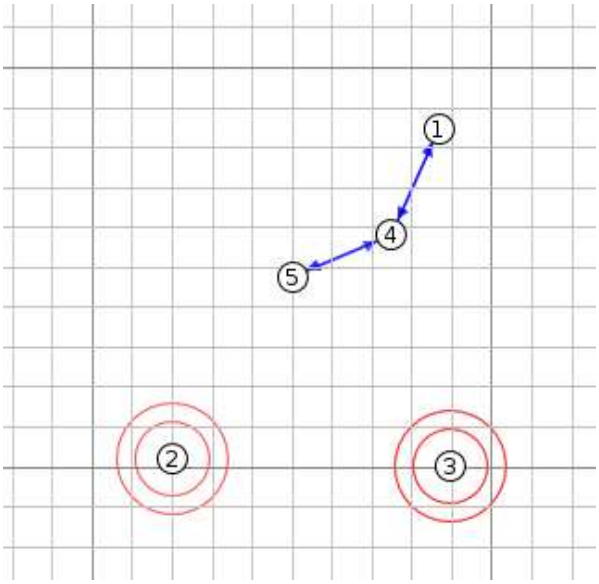
actual values for the IFTTT interface and whenever we send an alert to IFTTT. We had to manage the synchronization between log messages as Node-Red messed up with the order of the output to file.

Experiments

In order to prove the correctness of the project, here we provide some possible scenarios. We will indicate with square brackets a group of close motes, for example [1, 2] [3] means that motes 1 and 2 are close, while mote 3 is alone.

[1, 4, 5] [2] [3]

In this scenario we expect no alerts from motes 2 and 3, while motes 1, 4 and 5 will trigger some alarms. As we can see from the Cooja diagram, only mote 4 can "hear" both motes 1 and 5.



We also provide here a picture of a smartphone receiving all the notifications from IFTTT applet.

The logs are available in the folder experiments/exp1

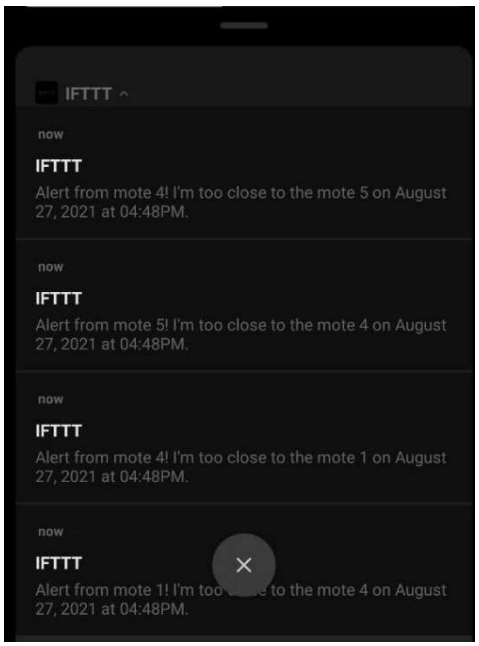
[2, 5] [1, 3] [4]

The logs of this simulation can be found in the folder experiments/exp2.

Everything works as expected. Mote 2 signals that it is too close to 5 and viceversa, same applies for motes 3 and 4.

[1, 2, 3, 4, 5]

Every mote is in range of the others, so we can see a total of 5x4=20 alerts sent to IFTTT, confirmed by the NR-log.



Also in this case, everything works as expected.

The logs can be found in the folder experiments/exp3.

Network pictures

Missing pictures of the network used in the experiments can be found as a png in the respective experiment folders.