Final Project Report

Project title: Keep your distance

Author:

Abednego WAMUHINDO KAMBALE 10740158
Muhammad Irfan Mas'udi 10672914

IMPLEMENTATION LOGIC

1. **TinyOS**

In TinyOs we have three files: FinalProject.h, FinalProjectAppC.nc and FinalProjectC.nc.
The header file FinalProject.h contain the message structure which is the following:

```
typedef nx_struct msg {

  nx_uint8_t mote_id;

} msg_t;
```

The FinalProjectAppC.nc contain the configuration and all the components and interfaces we used for the implementation.
The FinalProjectC.nc contains the implementation. Here is the logic we used to implement the solution.
We defined three arrays:
uint8_t ids_received [Mote_Number]: this is used to store the ids of the motes that are close to the actual mote.

uint8_t counters [Mote_Number]: this is used to store the counters of the mote that are close to the actual mote. The index of the id in ids_received is the same as the index of the counter in counters of the same mote.

uint32_t last_times [Mote_Number]: This is used to store the last time a particular mote sent it requests to the actual mote. This time is used to reset the counter in the case the mote is not in the range of the actual mote. The index of the id in ids_received is the same as the index of the last_time in last_times of the same mote.

We set the **Mote_Number** to 50.

The first thing we do after the radio module is active, is to initialize those arrays. And then start the timer. We followed the logic to initialize the arrays: we set all the id in **ids_received** to 60 since this

cannot be reached because we can support up to 50 motes for our case. All the **counters** and **last_times** are set to 0.

Every time the timer is fired, we broadcast the message to other motes.

When a mote receives a message from other motes. First, we check if the received ID is in the **ids_received** array. If not, we will have to register it and set the counter to 1 and set the last_time.

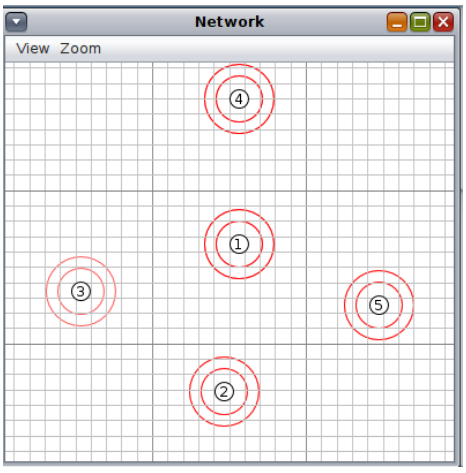
If the id is already present in the array, we retrieve the index of the **mote_id** from the array **ids_received** and we check if the difference between the current time and the saved last_time is less than 510ms. We add the 10ms to compensate some delay that can be due to processing. If yes increment the counter. Then we check if the counter is equal 10, if true, we reset the counter to 0 and set the last_time, finally we send the alarm to node-red with the printf function.

Suppose a message is received from a particular mote and the difference between the last time and the current time is greater than 510ms. In that case, the counter is reset to 1, so that the current message is counted as the first one and we will need other 9 successive message from the same mote to trigger the alarm. This is done for every mote since the id can be retrieved from the arrays that are set correctly.

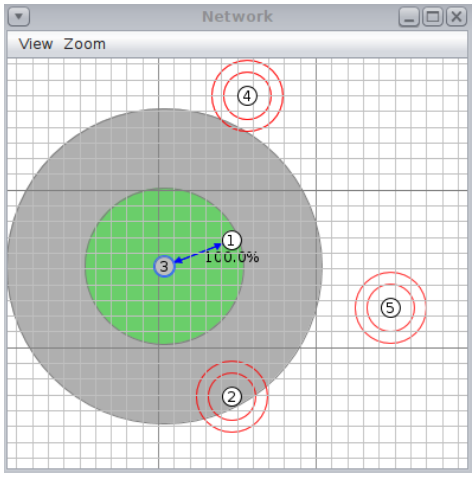
If the ids_received is full of the mote ids, we will erase the one with the minimum last time (that means the old id in the array), shift the array and register the new mote id in the array.

2. Cooja

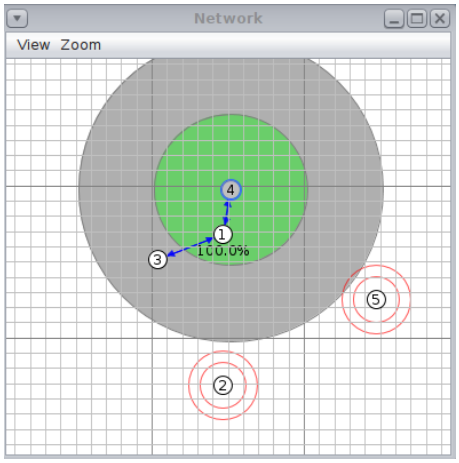
We made the simulation in Cooja with 5 motes. We moved the mote for them to be in range. We followed this pattern for the simulation.



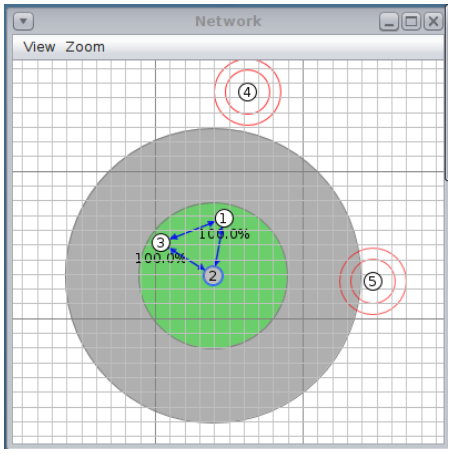
This is the initial position of the mote, there are not all in range each one to another



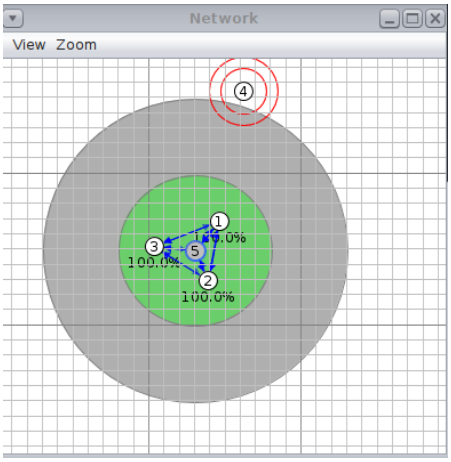
Then we move the mote 3 toward the mote 1, for them to be in range, after about 5s the alarm was triggered



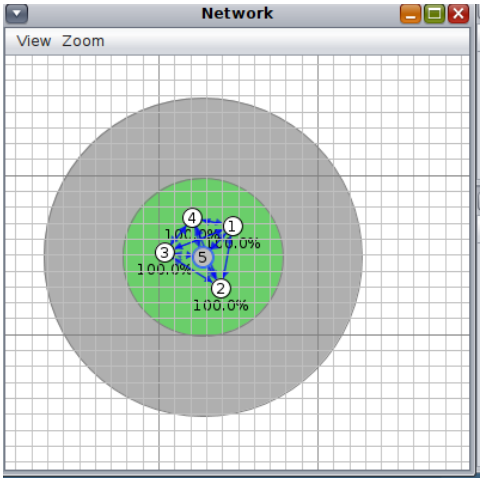
After we move the mote 4 toward the mote and before the 5s (10 counts) was expired we move it far from mote 1. In that case the alarm was not triggered



Then we move the mote 2 to be in range with mote 3 and mote 1, the alarms were triggered by the 3 motes.



After we took the mote 5 and move it to be in range with the motes 1,2 and 3, and move it far from these motes before the 10 successive counts and the alarm concerning mote 5 was not triggered



Finally, we put all the mote in range, and all the alarms were triggered successfully.

3. Tossim

We even used Tossim to make simulation. We set up 5 mote all of them connected two by two with -60dB. We modified also the RunSimulation.py accordingly.

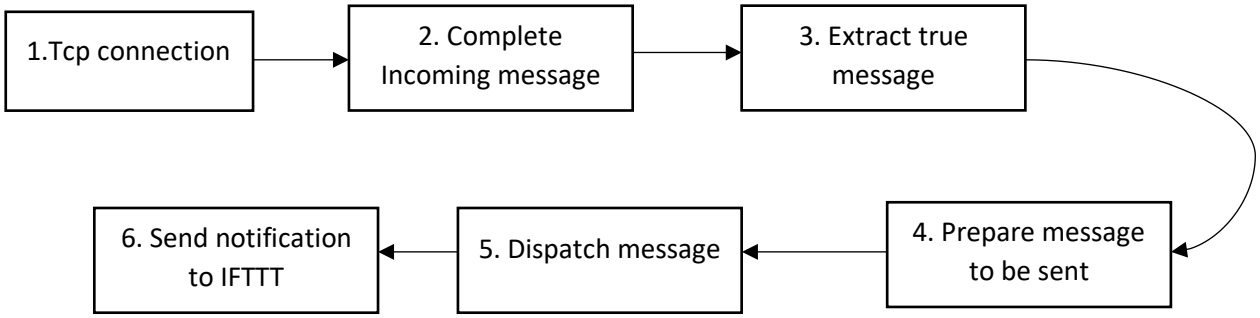
For the logfile “sim1” we had this configuration: Mote 1 starts at 0s, mote 2 at 5s, mote 3 at 3s, mote 4 at 6s and mote 5 at 7s. The steps were 2200.

For the log file “sim2” we had this configuration: Mote 1 starts at 0s, mote 2 at 1s, mote 3 at 2s, mote 4 at 3s and mote 5 at 4s. The steps were 2200

In theses log files we can see how the counter evolves. To run the Tossim simulation we should remove the printf usage in TinyOs to not have conflit.

4. Node-red

We send the alarm from the mote through Cooja using the socket on the motes. We open the port for each mote. Then in node red we used this pipeline to send notification to IFFT.



The Tcp connection block: is there to establish the connection with Cooja through the ports set of each mote. The port is 6000x, x being the mote id.

The complete incoming message block : is a function block that help complete the message with a user_id and port field in order to use it for further processing and for log.txt output.

The extract true message block: is there to extract the true message since with the printf function we have some strange characters so we used this function to extract the true message constituted by the id of the two motes.

The prepare message to be sent block: is used to prepared the message that is going to be send to the IFTTT server.

The dispatch message block: help dispatch the message in order to do the web request. The dispatching is made according to the user_id set in the the complete incoming message block

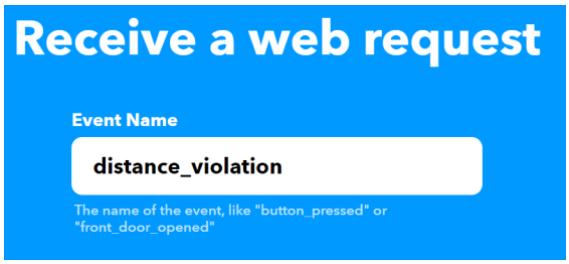
The send notification block: this is a web request block for each notification message. It will be sent to the IFTTT server that will display it to the user smartphone. Each user to be notified must have a specific API_KEY to put in the url of the web request block. In our case we just simulate using one API_KEY for mote 3 and 1 so we received the messages on the same smartphone.

On the output of the block 2,3,4,6 we used a specific function prepareLog for each of them to customize the message to display in the log file.

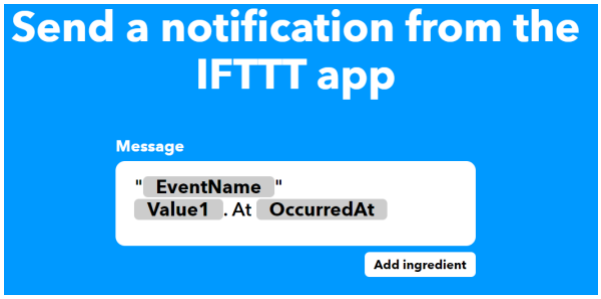
5. IFFT

To receive a notification on his phone, every user needs to install IFTTT app and configure a new applet online.

The above picture shows the name we gave to our applet



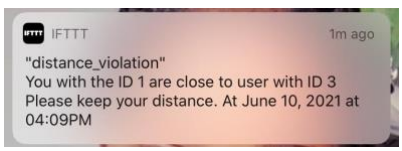
This picture shows the event name we setup

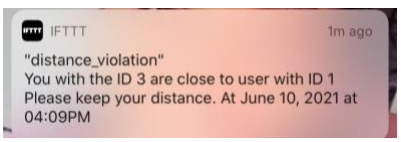


This picture shows the notification message to be sent to the user mobile phone. Value1 is the message that was set in node-red in the block that prepared the message to be sent.

Notification received

Here are some examples of notification we received while performing simulation.





The logs for the simulation are provided in the log files. For node-red we have the file log.txt, for Tossim all the file with the “sim” prefix, and for TinyOs the log_tiny.txt