Keep your distance Report

Alessandro Dangelo, 10524044
Tancredi Covioli, 10498705

1 Project proposal

Design and implement a software prototype for a social distancing application using TinyOS and Node-Red and test it with Cooja. The application is meant to understand and alert you when two people (motes) are close to each other, and it is developed in three steps:

- Each mote broadcasts its presence every 500ms with a message containing the ID number.
- When a mote is in the proximity area of another mote and receives 10 consecutive messages from that mote, it triggers an alarm. Such alarm contains the ID number of the two motes. It is shown in Cooja and forwarded to Node-Red via socket.
- Upon the reception of the alert, Node-Red sends a notification through IFTTT to your mobile phone.

2 Assumptions and implementation choices

Some assumptions and implementation choices that have been made are:

- **Distance range**: the distance range corresponds to the Transmission range of the mote. It is used the default value given by Cooja, which is an order of magnitude greater than a real social distancing application, but makes the management of the motes easier. Transmission ranges are the same for every mote.
- **Population size**: the population size is set to 10, since a greater number would have been not helpful for testing purposes. It can be increased by editing the header file (`MAX_NODES`).
- **Alarm message**: the alarm message is sent each time a mote receives 10 consecutive messages from the same corresponding mote, and it is sent again once 10 more messages are received. This approach was chosen to avoid a flood of incoming messages, expecially in crowded use cases in which social distancing may not be easily met.
- **Testing**: the system is tested with Cooja, since it gives the possibility to move motes in real time. Some tests have been made also in TOSSIM, but they have not been included in the documentation because they have been easily replicated in Cooja.
- **IFTTT notification**: the alarm message is displayed on the user's smartphone by a notification that appears in the notification menu. The Telegram message notification method was also tested, but it resulted in a less polished experience.

3 Mote's overwiew

All motes are equal and they implement the same code:
a mote has its own id, a personal counter and stores the status of the counters received from the other motes around.

When a mote starts its radio, the counter is set to 0. Once the message containing its own Id and the value of the counter is broadcasted, then the mote increments its counter.
Upon reception of a new message, the mote updates its status by checking the sender ID, this way the corresponding value in the status is updated or reset, depending on the value stored and the new value received.
If one of the counters in the status becomes a multiple of 10, then the alarm procedure is triggered and a message is sent. This message contains the IDs of the two motes (the one of the neighboring mote and its own) and the number of consecutive messages that either one received from the other.
As soon as the two neighboring motes go away from each other, the alarm procedure is stopped.

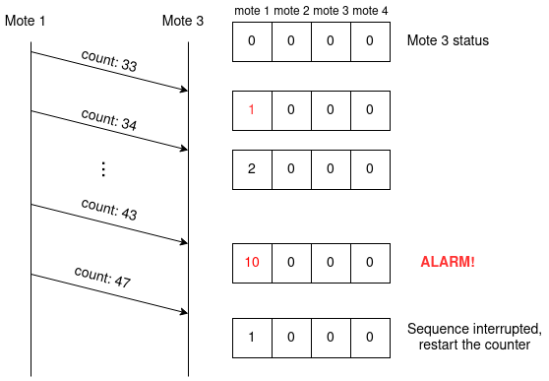

Figure 1: evolution of the status of a mote

4 Node-Red flow overview

The flow (shown in figure 2) begins with some TCP input nodes. These nodes are each connected to a specific port, that corresponds to the port which a Cooja's mote is listening on. Each mote may be listening on a specific port (in figure 2 the flow is shown connected only to one port to save space, but in the attached flow it is connected to 10 ports).
All incoming messages are filtered in the next step, this way only the alarm messages are forwarded. All these messages are then turned into sentences, without losing information.
As last step, an IFTTT service is triggered by sending an URL request to the IFTTT API, containing the message.

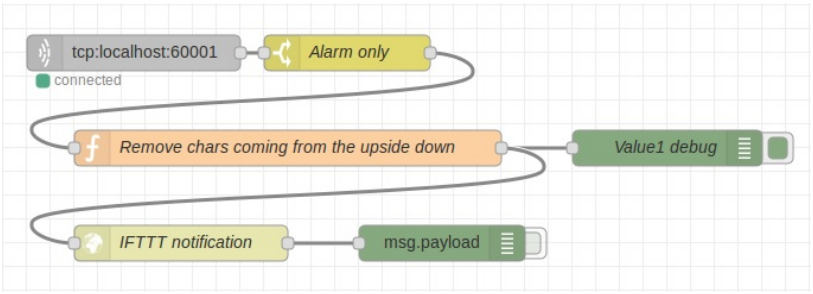

Figure 2: Node-Red flow

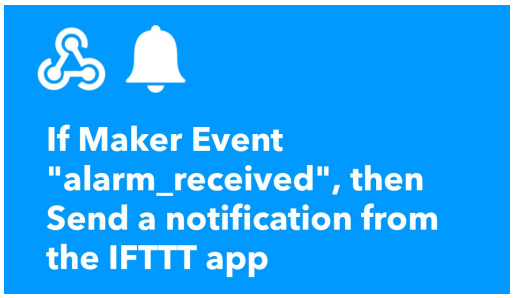

Figure 3: IFTTT applet

5 IFTTT overview

The IFTTT applet used works this way (as shown in figure 3):
IF the event "alarm_received" is triggered
THEN a notification is sent from the app.

6 Simulation and testing

The project was tested with several topologies; in this documentation a 10-mote simulation is described, since it seemed like a good middleground between stressing the system and discovering edge cases.
During the simulation different scenarios were tested in sequence:

1. All motes are far enough from each other not to violate the distantiation rule (figure 4).
2. The motes form four groups with different number of members (figure 5).
3. A group is dismantled and its members are assigned to different groups (figure 6).

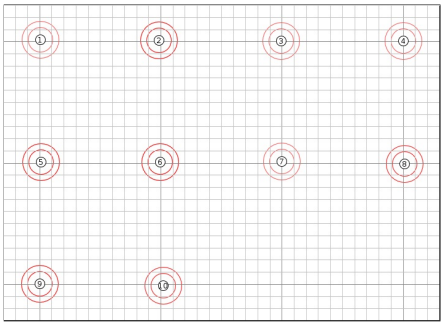

Figure 4:
motes follow distantiation rule

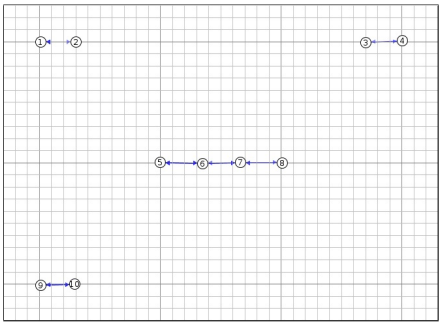

Figure 5:
some small groups are created

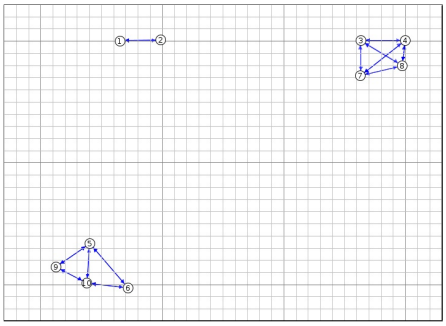

Figure 6:
few crowded groups are created

These scenarios were chosen in order to test the motes' behavior in a 1-to-1 situation and in a crowded environment that dynamically changes.

6.1 Results

Simulations had results in line with what it was expected, but two peculiar behaviors were encountered:

- In a crowded environment, when many motes are near each other, the packets sent among the motes may collide and result in malformed messages being received. These messages are automatically discarded by the receiver, which will lose the sequence of incoming messages from its neighbours.
- In the remote case in which a mote is receiving packets while it is ready to transmit, the message reception is stopped and the transmission takes place. If this happens, the mote will lose the incoming message.

Figure 7: timeline of the two behaviors

An example of the two behaviors can be noticed in figure 6, at timestamp 59s (see the log provided with this documentation, `log10motesDoc`). In figure 7 we can notice that mote 6 starts the transmittion of a packet and a few milliseconds later mote 10 does the same. Mote 5, which is in range of both, receives corrupted data as soon as mote 10 starts its transmission because of the collision.
At the same time, as mote 10 is receiving the packet from mote 6, its transmission timer fires and the reception is stopped.
The result is that motes 5, 6 and 10 miss a message and, in turn, reset their counter (as shown in figure 8).

```
00:59.080   ID:8    ALARM! m8 <- m7, n:30
00:59.486   ID:7    ALARM! m7 <- m4, n:30
00:59.505   ID:5    m5: m6 got near me
00:59.506   ID:10   m10: m6 got near me
00:59.511   ID:6    m6: m10 got near me
00:59.511   ID:9    m9: m10 got near me
00:59.511   ID:5    m5: m10 got near me
01:00.509   ID:1    ALARM! m1 <- m2, n:100
01:00.574   ID:2    ALARM! m2 <- m1, n:100
```

Figure 8: log section where the two behaviors take place