IOT Project- Keep Your Distance

Cristian Moioli- 10560811
Mariano Pelagatti- 10522120

Tiny-OS development

For what concern the development of the telosb firmware with TinyOs, we firstly start thinking about the probe message that each mote must send. To recognize the continuity of ten messages, the probe needs to include the sender ID and a sequence number of the message, which is incremented by the sender each time a new probe is sent. So, each mote keeps sending a new probe when the timer (500ms) fires and listen to incoming messages.
Moreover, each mote stores three arrays, which are used in this way:

1. **probeCounters[i]:** we store how many continuous probes the mote has received from device i.
2. **lastProbeCounters[i]:** snapshot of the same counters but delayed by 850ms, to know whether motes are still sending probes, thus, to infer whether a mote is still near or not.
3. **lastIncrementalId[i]:** we store the last sequence number received from mote i.

(Note = the array index i is equal to MOTE ID-1, since arrays start from 0, while mote enumeration from 1)

Once a new probe is received, the sender ID is retrieved. We check whether the new sequence number is exactly the last one we received, incremented by 1 (*lastIncrementalId[senderID] + 1*). If not, that means we find out a discontinuity, so that counter needs to be zeroed.
Then, we just increment by one the probe counter related to that sender ID. If we have a number which is greater then 10, an alarm is sent via the serial port. The alarm is composed by a list of the mote near to the source, separated by a comma. The first mote in the list is set to the source device.

An issued to be solved now is related to when a mote goes out of the transmission range of another mote. Since no probes will be received, we set up a timer (850ms) to check whether motes are still near or not. It periodically checks whether the probe counters have increased in a that period.
For a given mote *i*, if the **last** probe counter is equal to the **current** counter related to the same mote *i*, that means this mote has not received from *i* any probes in 850ms. If this happens, we can zero out the counter for mote *i*, since this period is greater than the sending period of the probes. It means mote *i* is now considered far away from the source mote.
At the end of this routine, we just update the state by copying the probe counters into the lastProbeCounters array.

A last note regards the frequency for which we send alarms. Let assume we have 3 motes which are all near and they do not move. We should avoid sending new alarm for each probe received. To deal with this problem, we decide to send the alarm only when the list of near devices changes. In this way, if (1, 2, 3) are all near, only one alarm (for each mote) is sent until a fourth device is found or one of the three goes away.

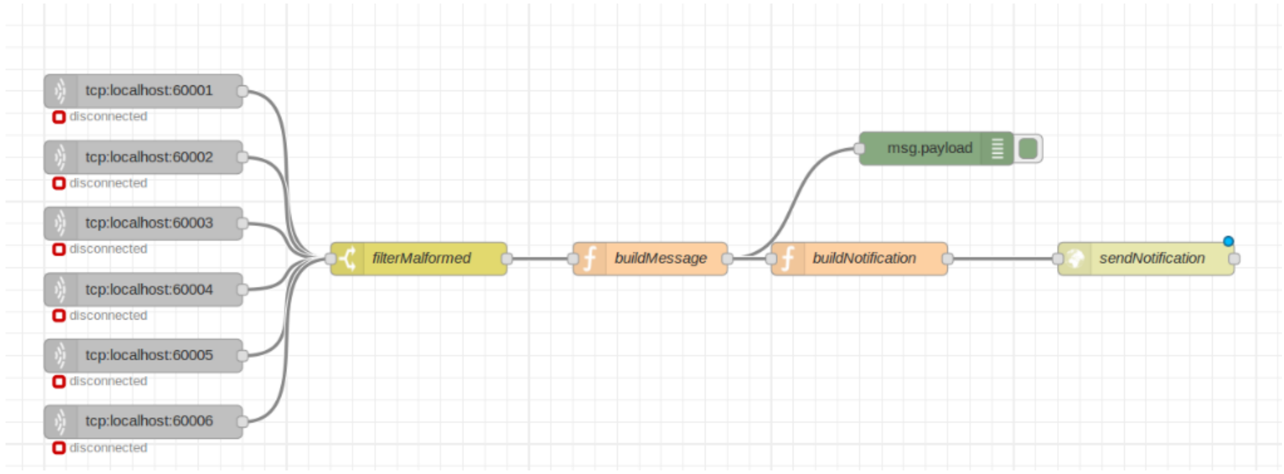
Simulation Environment

We performed the simulation with 6 motes. The printf sends the message to the serial port of the telosb mote. Cooja allows to capture each serial communication and forward it to a server socket, one for each mote. Then, node-red will create 6 socket and connect to Cooja server sockets. All the incoming messages are then filtered and readapted to be sent over IFTTT. In a real deployment scenario, the mote should be linked to a device that replaces the Cooja environment and forward this message to the node-red socket. Instead of using the serial port, a completely different scenario may include a sink node which gathers all the messages provided by the motes with a long-ranged communication technology, then forwards it to the backhauling network.

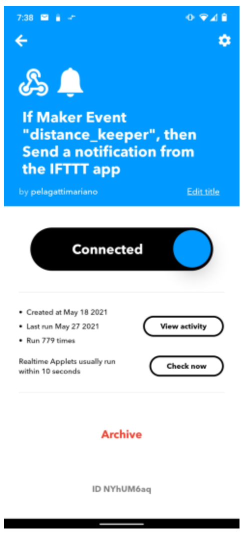
Node-RED flow:

tcp input: receives messages from mote x on port 6000x, which was set in Cooja as *Serial Socket (SERVER)*
filterMalformed: messages are received as an array of values, the first being the id of the mote sending it, and the following ones the motes near it. So, this function allows only messages matching to following regex to pass through → *^([1-6],)*[1-6]$*
buildMessage: retreives sender id and neighbour ids from the message received, and uses them to set msg.payload for debug purposes
buildNotification: this function sets
- msg.event to *distance_keeper*, which is the event which triggers the webhook in IFTTT
- value1 to the id of the user who needs to receive the notification
- value2 to the id of his "neighbours"

then it sends value1 and value2 as msg.payload to the following node
sendNotification: sends an HTTP POST to the URL composed with our API key, in order for IFTTT to get the request.



For a realistic deployment, all users would need to register to IFTTT and install an app on their phone. For simplicity, we used a single account, with a single applet, which is acting as a "local exchange", receiving notifications for all the users involved.


7:38

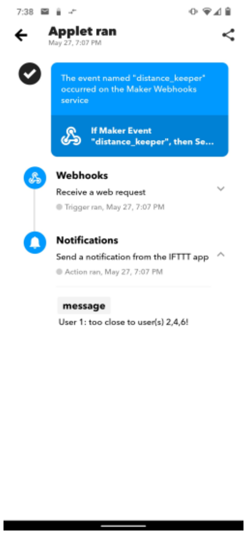
If Maker Event "distance_keeper", then Send a notification from the IFTTT app

by pelagattimariano

- Created at May 18 2021
- Last run May 27 2021
- Run 779 times

Realtime Applets usually run within 10 seconds

ID NYhUM6aq


7:38

Applet ran
May 27, 7:07 PM

The event named "distance_keeper" occurred on the Maker Webhooks service

If Maker Event "distance_keeper", then Se...

Webhooks
Receive a web request
Trigger ran, May 27, 7:07 PM

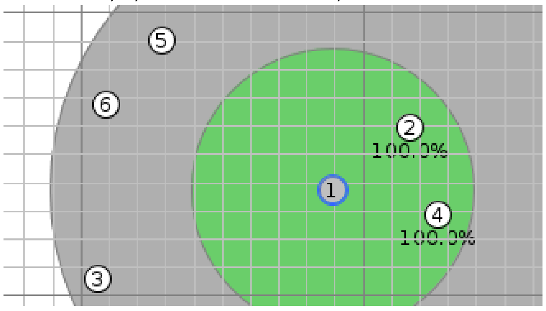
Notifications
Send a notification from the IFTTT app
Action ran, May 27, 7:07 PM

message
User 1: too close to user(s) 2,4,6!

Simulation Example

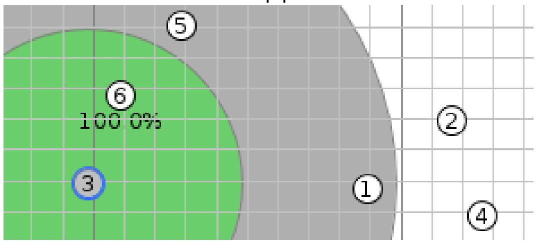
In files "cooja_log.txt", "node_red_log.txt" you can find the log regarding a simulation example with these movements:

0. Motes 1,2,4 starts all near, the same for mote 5 and 6. Mote 3 is not near to any motes.



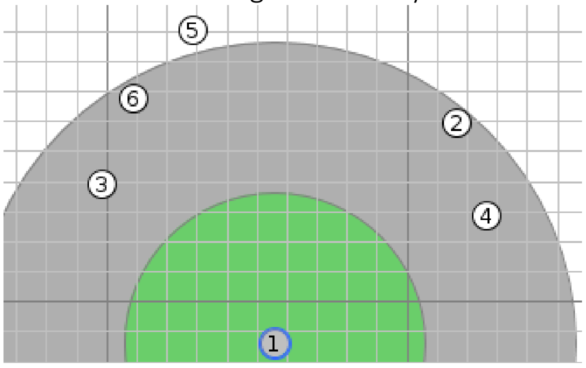
At stationary time (about 9s) the two cluster (4,1,2) and (6,5) has been detected.

1. 00:14.275: Mote 3 approaches mote 6.



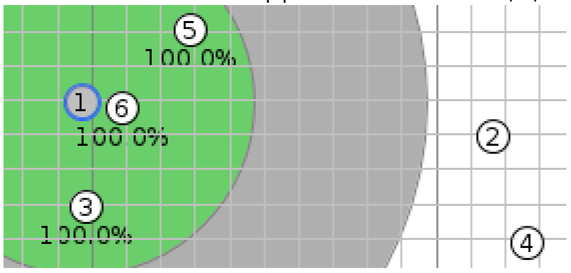
Mote 6 detects another new mote (3) once 10 probes have been exchanged.

2. 00:37:639: Mote 1 goes far away from all motes.



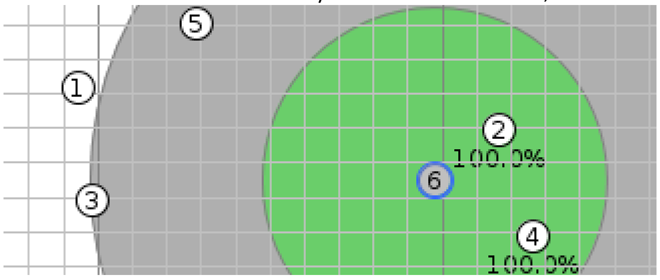
Once the 850ms timeout fires, motes 2 and 4 detects mote 1 is no more near

3. 00:52.657: Mote 1 approaches motes 3,6,5



In an asynchronous way, motes 3,6,5 detects mote 1 is now near to them.

4. 01:12.222: Mote 6 slowly moves near to 2,4



As before, both motes 2 and 4 detects mote 6, which is no more near to (1,3,5)