Project 1. Keep your distance

Authors:
Andrea Falanti(codice persona: 10568850, matricola: 944954)
Andrea Huang(codice persona: 10582949, matricola: 963091)

---

0. Introduction

The project consists in sending an alarm from the local network (simulated in Cooja) to a device over the Internet using nodeRED and IFTTT. A mote broadcasts messages periodically every 500ms, signaling its presence to other nearby motes. The alarm is triggered when a mote in the local network receives 10 consecutive messages from another mote, by sending an HTTP request via nodeRED to the IFTTT webhook service.

1.TinyOS

The motes are implemented using TinyOS.

1.1 KeepYourDistance.h

It provides 2 data structures, 1 for communication and 1 for the state of the mote:

- **radio_id_msg**: this is the broadcast message. It contains:
    - *sender_id* (nx_uint16_t)*:* the id of the sender
    - *counter* (nx_uint16_t)*:* the id of the message. It is used to keep track of consecutive messages on the receiving mote.
- **bcast_map_t**: a data structure to keep track of the consecutive messages from other motes. It is used in a list to dynamically handle the messages from other motes. It contains:
    - *sender_id* (uint16_t)*:* id of the sender
    - *last_counter* (uint16_t): counter of the last message it received from mote {*sender_id*}
    - *consecutive_counter* (uint16_t): the number of consecutive messages received from the mote {*sender_id*}. When it reaches 10, the alarm is triggered, and the counter is reset to 0. It is defined as an *uint16_t*, so when it reaches the value 65535, the next message will be 0, but this case is properly handled by application logic. Thus, two messages are consecutive based on their counter value *X mod 65535,* but, for the *consecutive_counter* to be reset to 0 by overflow and match this condition, the sending mote needs to be active for about 22 days and reconnect in this specific time frame, so it is extremely unlikely.
    - *next:* pointer to next *bcast_map_t* (unidirectional list).

1.2 KeepYourDistanceAppC.nc

It's the configuration file and defines the components used by the app. Contains components required for radio communication (AMSenderC, AMReceiverC, ActiveMessageC) extended with

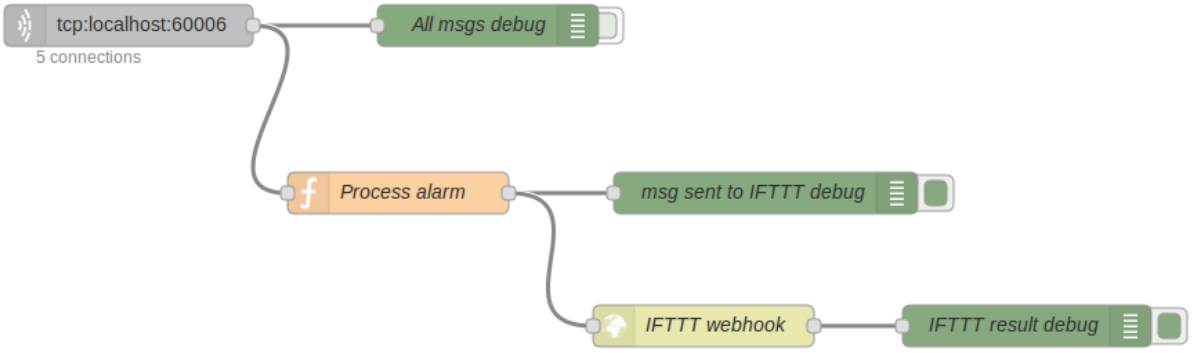
the components required to print log messages (SerialPrintfC, SerialStartC). TimerMilliC is used to send broadcast messages at the right frequency.

1.3 KeepYourDistanceC.nc

It contains the application logic of a mote.

- Every mote broadcasts a *radio_id_msg* every 500ms.
- Upon receiving a broadcast message from another mote, the list of *bcast_map_t* is updated as follows:
  - if the sender was not present in the list, it is inserted in the head of the list and all its fields are initialized, in particular, *consecutive_counter* is set to 1.
  - otherwise, if the sender id is already present in the list:
    - if the new message has a *counter* that is subsequent to *last_counter,* then the *consecutive_counter* value is increased by 1, and *last_counter* is updated to *counter.* If the number of consecutive messages is 10 then an alarm message is logged (and therefore sent to nodeRED via a serial port, see next chapter) with a *mote_id* (id of the alarm sender) and a *proximity_mote_id* (id of the mote from which 10 messages were received) values and the *consecutive_counter* is set to 0. Otherwise, no message is logged.
    - otherwise, if the new message is not a consecutive message, the *last_counter* is updated and the *consecutive_counter* is reset to 1.

2. NodeRED



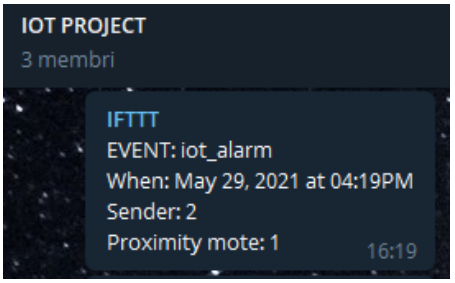
Node-red flow is pretty simple. A TCP server socket starts the flow, Cooja motes can connect to it with this procedure:

- right-click on the Cooja mote in network tab > More tools for Sky # > Serial Socket (CLIENT)
- set host = localhost and port = 60006, then click connect

All printf messages are sent to the socket and are processed by the node-red flow (in a real deployment, a specific message should be sent directly to node-red, instead of using serial port). “Process alarm” is a simple function to generate a message for an IFTTT webhook, that processes only printf messages with the tag “Alarm”. Msg.event is set to the event name given in IFTTT webhook configuration, while in the payload two fields are set: value1 is the id of the mote that generates the alarm and value2 is instead the id of the mote which sent 10 consecutive messages.
The message is then sent to IFTTT with an HTTP request node, whose URL depends on the IFTTT applet API_KEY and the IFTTT webhook target event (set in msg.event).

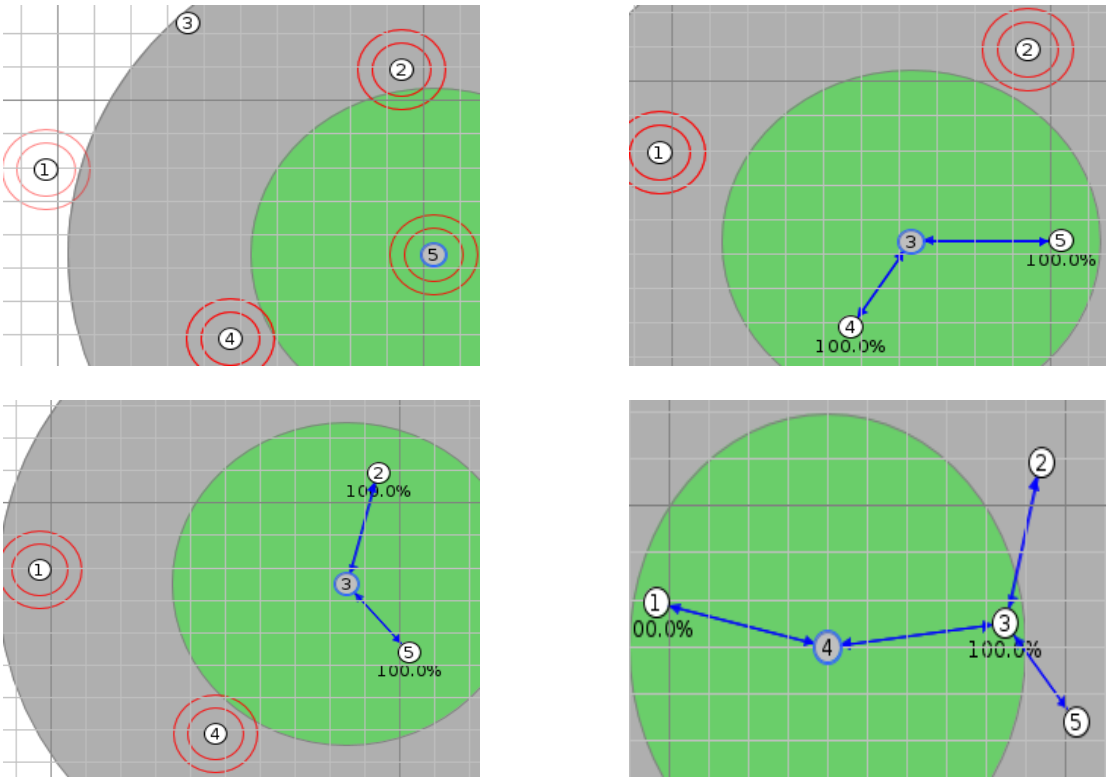
3. IFTTT

An IFTTT applet must be up and running for sending notifications on alarm events. ‘IF’ block must be configured with a webhook, so that node-red can send a request to it. ‘THEN’ block is instead flexible, in our case we decided to send the notifications to a telegram group (Telegram > Send message), with this format:
EVENT: {{EventName}}
When: {{OccurredAt}}
Sender: {{Value1}}
Proximity mote: {{Value2}}



Remember that API_KEY and msg.event must be updated in the node-red flow based on your applet one.

4. Cooja simulation example



Example of Cooja simulation, with the following steps:

1. (top-left) At start motes are distant, no communication happens
2. (top-right) Mote 3 communicates with 4 and 5
3. (bottom-left) Mote 3 now communicates with 2 but lost connection to 4
4. (bottom-right) Mote 4 now communicates again with 3 and also with 1

This short example tests all the logic of the application. When communication starts a list element with the counters is allocated. In step 4, we can see that mote 3 and 4 reset their counters properly when they start communicating again (log line 106 and 116). Motes that remain in range update the consecutive counter properly and send the alarm when 10 is reached (first one at line 160). Log file of this example is provided in the zip file.