IoT Final project
project 1. Lightweight publish-subscribe application protocol

Marco Barbieri (10684602), Matteo Bianchi (10698168)

July 24, 2023

POLITECNICO
MILANO 1863

1 Objective

The project's objective is to create a publish-subscribe application protocol, similar in essence to MQTT. In our project there are two types of motes arranged in a star topology. Node 1 is the PAN coordinator and the other nodes are both publishers and subscribers, these nodes will be called motes from now on and node 1 will only be called PANC.

The tools we are using are: TinyOS to create the code of the nodes, Cooja to simulate and test our code, Thingspeak to output our data on graphs and Node-RED to link Cooja with Thingspeak.

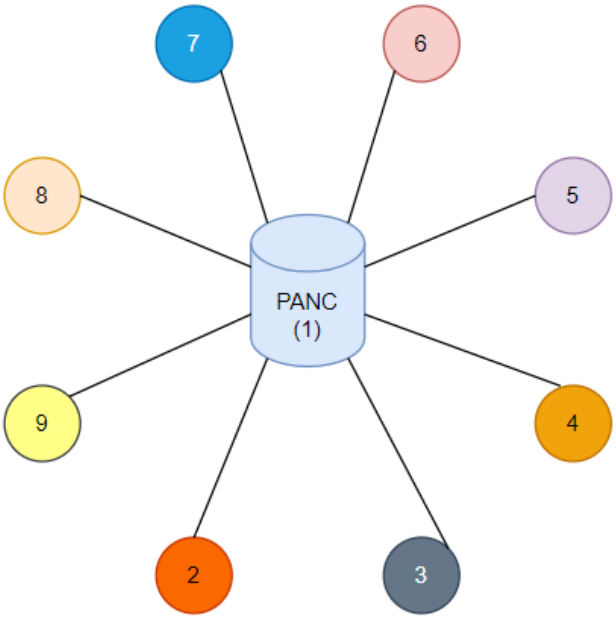

Figure 1: topology of the network

2 Message

The message interface is unique, this means that to differentiate the type of message we have an attribute inside the message interface.

attributes:

- id, used to identificate the sender of the message
- type, can be CONNECT, SUBSCRIBE, PUBLISH, CONNACK, SUBACK
- topic, used for subscription and publishing
- payload, used only for publishing

3 PANC

The PANC is the central node of the network, it is similar to an MQTT broker. the PANC can communicate with the motes using three types of messages: CONNACK, SUBACK and PUBLISH:

- The CONNACK messages are used to confirm that we received a connection request by one mote. If a mote is already connected to the PANC but it sends a CONNECT message, we still send the CONNACK because it could be that one of our previous CONNACK was lost. The PANC uses a connection array to know which motes are connected.

- The SUBACK messages are used to confirm a subscription request. As for the CONNACK messages, they are also sent again if the mote sends a SUBSCRIBE message with a topic they are already subscribed to (in case a previous SUBACK was lost). The PANC uses a subscription matrix to update and check the subscription topic of the motes.

- The PUBLISH messages are used to inform all the subscriber motes that another mote published a message on the topic they are subscribed on.

When the PANC receives a PUBLISH message, it checks its topic and loops through all the motes to check whether they are subscribed to this topic, if so the PANC will send them a PUBLISH message with the same topic as the one received. To make this loop we used a timer so that the PANC wouldn't send too many packets too fast.

4 Motes

The motes can communicate only to the PANC using three types of messages: CONNECT, SUBSCRIBE and PUBLISH:

- The CONNECT messages are used to establish a connection to the PANC. When a mote sends a CONNECT message, it starts a timer in which it expects to receive a CONNACK message from the PANC, if the mote doesn't receive the CONNACK when the timer expires it tries again to connect to the PANC (QoS=1).

- The SUBSCRIBE messages are used to specify the topic a mote wants to receive informations on. A mote can subscribe to three topics: TEMPERATURE, HUMIDITY and LUMINOSITY. Just like the CONNECT message, when a mote sends a SUBSCRIBE message, it starts a timer in which it can receive a SUBACK message from the PANC, if the mote doesn't receive the SUBACK in time, it tries to resubscribe to the same topic (QoS=1).

- The PUBLISH messages are used to give new informations to the other motes about a certain topic. Differently from the other message types, the PUBLISH message does not expect an ACK message (QoS=0). We chose to set the payload of the PUBLISH message between 20-100 for the sake of the simulation.

5 Simulation

To make our simulation, we used 5 phases for each mote: conn, sub1, sub2, pub1, pub2. In the conn phase, the mote sends a CONNECT message to the PANC. To proceed to the next phase it is necessary that the mote is connected to the PANC, so if there were errors connecting, the simulation won't skip to the next phase until we receive a CONNACK.

Once a CONNACK is received, we go on with phases sub1 and sub2. In these phases the mote subscribes to a random topic, it could happen that a mote tries to subscribe two times to the same topic, in this case the PANC still sends a SUBACK message (even though the sub2 phase is rendered useless).

After the two sub phases, we have two phases in which the mote generates a PUBLISH message with random topic and payload (between 20-100) and sends it to the PANC. The simulation phases are handled by a timer which is activated every time we detect that a phase is finished. When the timer runs out, we proceed with a new phase.

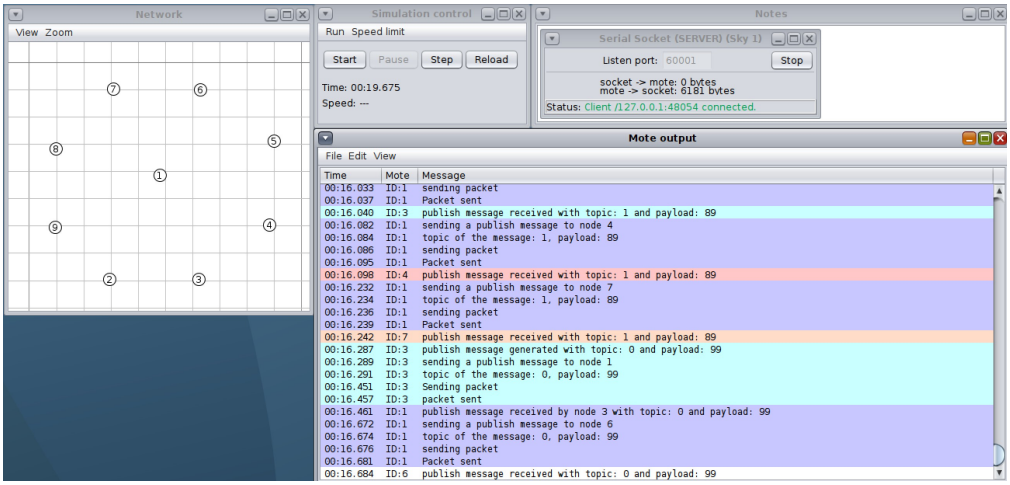

Figure 2: screenshot of Cooja showing the topology as well as some debugging informations

To actually run the simulation we used Cooja, creating a PANC node and 8 mote nodes. In this simulation we print many useful informations like whether a node is trying to send a packet, the packet type, the payload of the message.
The PANC was set so that every message it prints would be sent through TCP at port 60001 in our local machine. this enables the communication between Cooja and Node-RED

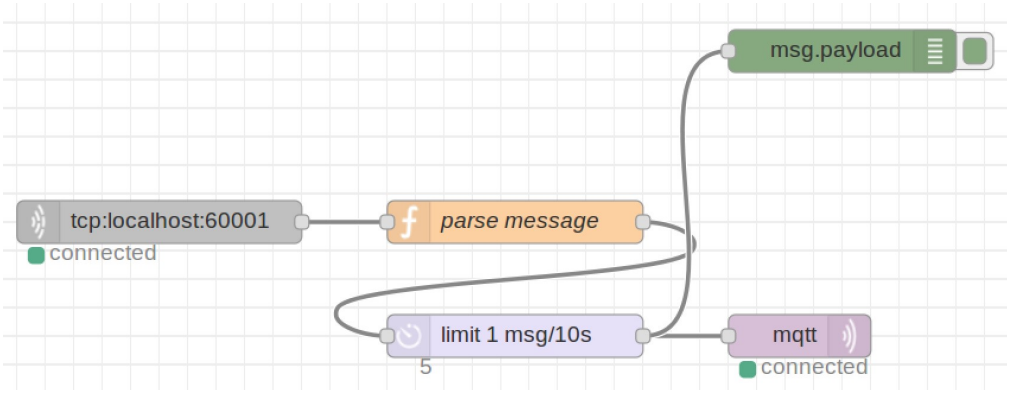

Figure 3: node-RED project diagram

In Node-RED we used a TCP node which listens at port 60001 for messages. The messages are then parsed, because the only messages we are interested in are the informations about the publish received by the PANC, we checked whether the first word of the message was “publish”.

After that we extracted the topic and payload of the PUBLISH message associated with the message and proceeded to send it to an MQTT node. This node was connected to the Thingspeak server in which it updated the graphs made for temperature, humidity and luminosity.

The delay node between the parser block and the mqtt block is used because we noticed that many packets were being dropped in the mqtt node. This was probably because the mqtt node is too slow compared to the arrival rate of the messages.
The link for the Thingspeak channel is: `https://thingspeak.com/channels/2224975`

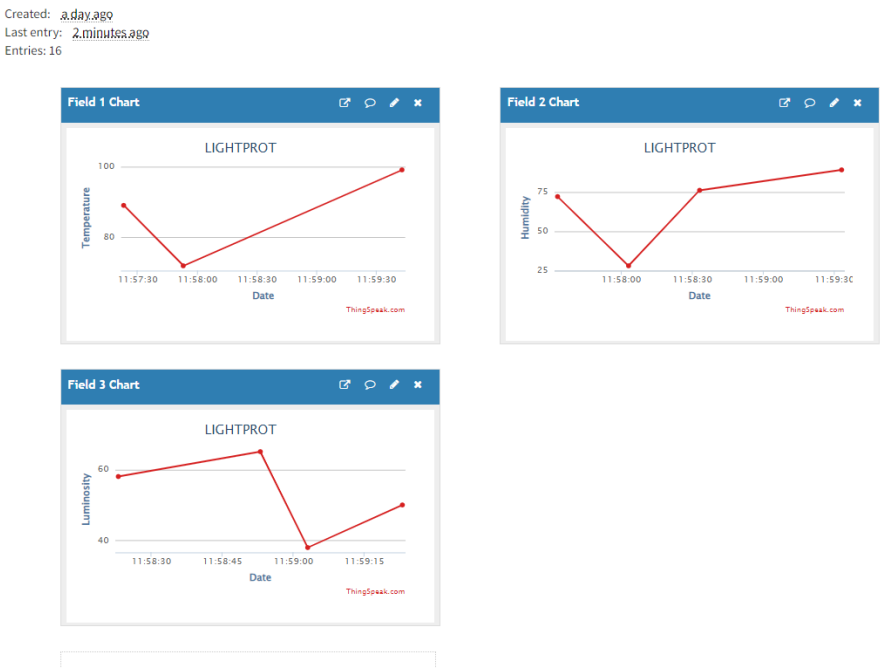

Figure 4: Thingspeak channel graphs

6 Conclusion

This project was a great opportunity to gain hands-on experience with the practical implementation of an IoT application. We learned a lot about IoT, TinyOS and Cooja. We consider a lot of factors when making design decisions and we have a better understanding of the trade-offs that are involved.
We also gained a deeper understanding and found interesting to explore the different problematic situations in which an IoT application could get stuck. We learned how to design our application to handle these situations gracefully.
Overall, we are very happy with the outcome of this project: we are confident that this experience will be valuable to us in our future career.