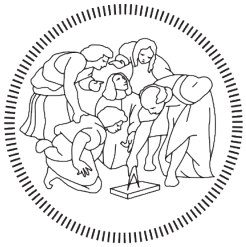


POLITECNICO
MILANO 1863

Internet of Things

IoT Project 2022-2023

Lightweight Publish-subscribe Application Protocol

Mauro Famà - 10631287
Alessandra De Stefano - 10..

August 25, 2023

Table of Contents

1 Introduction

The project aims to design a lightweight publish-subscribe application and deploy it, with a simulator, in a star-shaped network topology. TinyOS is the platform used to implement the sensor's logic, basic components and interfaces were used to implement a fully functional application, where Timers played a fundamental role in making the system work properly. In order to test and evaluate the application, Cooja has been used to simulate a sensor system, exploiting motes' output prints to verify the correctness of the developed project. In order to make the system communicate with the external world, we connected the PAN (Personal Area Network) Coordinator with Node-Red through a TCP socket, and successively transmitted data to the ThingSpeak visualization service using the MQTT communication protocol.

2 Methodology

2.1 Firmware Development

2.1.1 Data Structures

In our application, we used a `struct` to define the structure of the messages that are to be exchanged in the sensor network, it includes only the essential fields needed for a publish-subscribe communication protocol in order to maintain the system lightweight. We used integers to represent the message types and the topics, the table in Fig. 2.1 summarizes the message structure, showing the enumeration of the fields.

Message Fields
Type
Sender
Destination
Topic
Payload

Type	
0	CONNECT
1	CONNACK
2	SUBSCRIBE
3	SUBACK
4	PUBLISH

Topic	
0	TEMPERATURE
1	HUMIDITY
2	LUMINOSITY

Figure 2.1: Message structure

To manage the logic of the application, we used simple data structures to manipulate data, only arrays were used, without creating `structs` other than the one used for messages.

- **Connection**: an array sized as the number of nodes has been used to verify if a node is connected to the PAN Coordinator, the element indexed as the `node ID` is set to 1 if the node is successfully connected, 0 otherwise;
- **Subscriptions**: three similar arrays have been used to verify if a node is subscribed to a specific topic, following the same boolean logic described before, we created an array for each topic;
- **Messages queue**: an array of messages is used to enqueue them before being sent, size can be defined before the execution of the application.

2.1.2 Timers

Timers have been used to implement the essential application procedures: they impose a delay between connection and subscribe procedures, they manage timeouts for retransmission and they mark the time for sending messages eventually present in the queue.

Retransmission

Every node waits for the same amount of time before starting the connection procedure, we decided to do that because, in case of receiving failure from the PAN Coordinator due to concurrency, the retransmission mechanism is enabled to make all the nodes connect successfully. The same logic is applied to subscriptions, enabled after a fixed amount of time right after the connection procedure is completed. The PAN Coordinator does not handle CON and SUB messages with the queue because the retransmission mechanism guarantees that, at a certain point, all the nodes will be successfully connected and subscribed.

Transmission Queue

A different logic has been implemented for the publish procedure, in this case, the PAN Coordinator has to send multiple messages when receiving a PUB message from a node, so a queue is needed in order to not lose packets: the transmitter remains busy for a certain amount of time when sending a packet, so if we call the send function multiple times without waiting in between, packets will not be sent due to concurrency. In our application, when a PUB message needs to be sent, it is first put into a queue, then a periodic timer will check if the queue is empty, if not, a message is popped out and sent. With this design, the transmitter will not be called when busy.

2.2 Simulation

When creating the simulation with the Cooja platform, we have positioned the nodes in such a way that the sensor groups subscribed to the same topic are able to communicate only with the PAN Coordinator, by doing so, we demonstrate that the only way messages published by the concerned node can be received only through the Coordinator. In Fig. 2.2 the topology of the network used for experimental evaluation is shown. We compiled the TinyOS code to be installed on a TelosB device, simulating it with Cooja's Sky Motes.

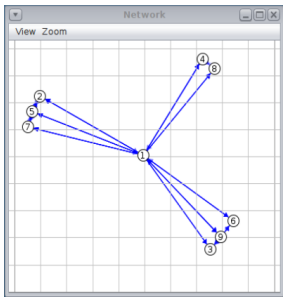


Figure 2.2: Star-shaped network topology

2.3 Data Visualization

To establish a connection between the sensor network and Node-Red, we opened a socket serial port on the PAN Coordinator accessible via TCP. In our Node-Red flow (Fig. 2.3), we use a TCP-in component to collect all the information printed out from the PAN Coordinator, we then pass all these strings to two function blocks, one to filter only PUB messages received from the other motes, and another one to format the information inside the messages to be published on our ThingSpeak channel through a MQTT-out component.

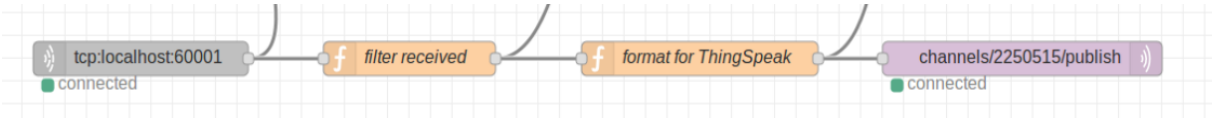
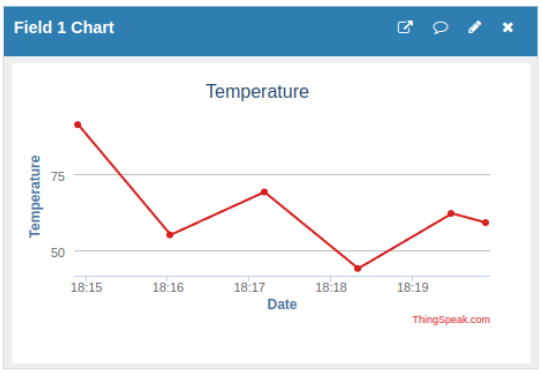
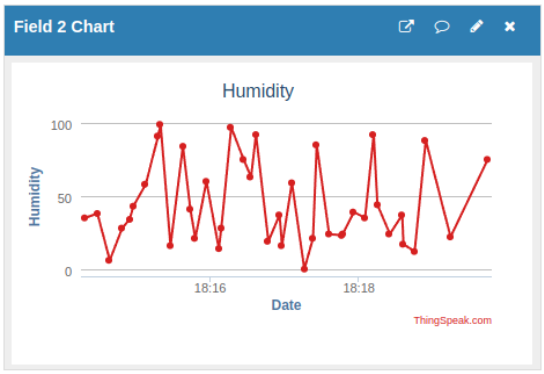


Figure 2.3: Node-Red flow

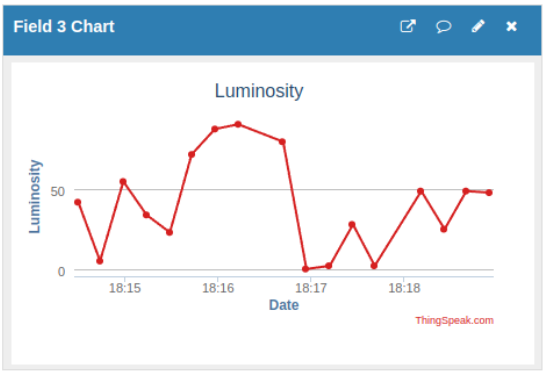
Our flow publish data by processing one message at a time, indeed, in the payload we send through MQTT to ThingSpeak, we update one field per publish. In Fig. 3.1 we show a screenshot of the charts of our channel some moments after the motes start communicating between each other.



(a) Temperature



(b) Humidity



(c) Luminosity

Figure 2.4: ThingSpeak channel charts

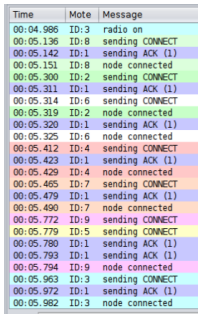
3 Experimental Evaluation

3.1 Assumptions

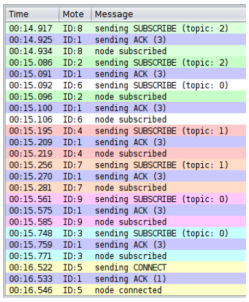
1. The PAN Coordinator is the node with `ID` equal to 1;
2. Nodes will publish and subscribe to a topic based on their `ID`, considering integer topics, the subscribe topic is `node ID` mod 3 and publish topic is (`node ID` - 1) mod 3;
3. In order to have at least three nodes subscribed to more than one topic, every node subscribed to topic 0 (Temperature) is subscribed to topic 1 (Humidity);
4. The publish rate depends on `node ID`: the lower, the faster.

3.2 Results and Analysis

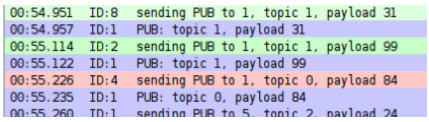
We evaluated the correctness of the developed application with debug messages printed out by the motes and by the ON/OFF switch of the mote LEDs: every time a PUB is received the red LED toggles, after connection green LED is on and after the first subscription blue LED is on. The following screenshots are taken from a single simulation.
In Fig. 3.1a, it is possible to notice the 3-step connection correctness, after a node sends the CONNECT message, it declare himself connected only after receiving the ACK of type 1 (CONNACK) from the PANC. In Fig. 3.1b the subscribe procedure and the retransmission of CONNECT message from Mote 5 due to lost CONNACK is shown. In Fig. 3.1c Mote 2 is publishing on topic 1 (payload 99) and, in Fig. 3.1d, all the subscribed nodes receive the PUB message from the PANC.



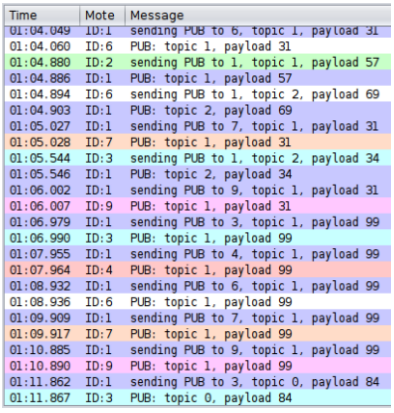
(a) Connection



(b) Subscritpion and retransmission



(c) Publish from Mote 1



(d) Publish received from subscribed motes

Figure 3.1: Cooja Mote Output console