POLITECNICO
MILANO 1863

Polytechnic University of Milan

Internet of Things - 054323

Project 1. Lightweight publish-subscribe application protocol

Author	*Person code*
BOVING Alexandre (leader)	*10601768*
BICCARI Stefano	*10652373*

Telecommunication Engineering - Internet Engineering
Date : June 3, 2023.

Contents

1 Problem Solving Approach

To solve this project, we took inspiration from the challenges, particularly the third one, as well as the various tutorials available to us to tackle the **TinyOS** part of the project. Before starting the project, we carried out an in-depth analysis of the objectives to be achieved, as set out in the project.pdf file, which we also consider to be our specifications. The first part of the project deals with connecting the various nodes to the PANC and managing any malfunctions that may occur. Next comes the subscription of the so-called client nodes to the various categories of data on offer:

- Temperature
- Humidity
- Luminosity

Any malfunctions must also be managed. After this, comes the publication of data by certain client nodes. Not only are they subscribed to pre-defined subjects, but they are also responsible for sending the data for the various categories. They first send the data to the PANC, which is responsible for sending the data back to all the client nodes that have subscribed to the correct data. This process is simulated in an event simulator called **Tossim**, which we use. Eventually, the PANC transmits the data received to **Node-red**, from which the same data is sent to **ThingSpeak** in order to construct the graphs related to the temperature, humidity and the luminosity.

1.1 Connection

The part dealing with the connection of client nodes to the PANC is inspired by the third and final challenge proposed this year. From the challenge, we take the idea that the nodes have different transmission times. Using the "*generate-send()*" function, we can prepare the type of message to be sent and its associated delay. In the case of a connection message, we send a type 1 message **(CONNECT = 1)** so that the PANC can subsequently detect the type of message received. The PANC will send a type 2 message **(CONNACK = 2)** to inform the originating node that its message has been received. In the inconvenient case of loss, deletion or waiting too long for a message to be sent, a flow control protocol named Stop & Wait is implemented to maintain the correct operation of message sending. This means that a Timer0 is implemented acting as the Stop & Wait protocol timeout and its value is doubled each time it is not respected. In order to avoid overloading the system by sending messages an infinite amount of times, we also implemented a Stop & Wait usage limit using the "*retransmission-con*" variable. Finally, when the CONNACK message is received, the "*connack-flag*" variable is incremented by 1 to stop the execution of the Stop & Wait protocol and indicates that we are connected.

1.2 Subscription

Very similar to the section regarding the connection aspect, we are implementing solely a modifiable list of values named "*sub-type*" associated with different data categories. The values are indicated as follows:

- Temperature

- Humidity
- Luminosity

Within the function "*generate-send()*", we explicitly specify that the message for subscription is of type 3 **(SUBSCRIPTION = 3)**, along with its confirmation as type 4 **(SUBACK = 4)**. For these messages, we incorporate the respective sub-type from the pre-defined "*sub-type*" list for each of the client nodes. We employ the same recovery mechanism of Stop & Wait in case of a glitch, albeit with a distinct timer: Timer1.

1.3 Publish

For the section concerning the message publication along with their associated values, we drew inspiration from an assignment provided during the audio-visual readings of the TinyOS laboratories, titled *TempHumSensor*. We initiate timers specific to distinct client nodes, totaling three in number, to commence the data retrieval procedure. We retained the same logical approach as presented in the solution provided in that assignment. To adapt it to our code, we now incorporate the data into the message to be transmitted. As a reminder, the data category name included in the message is referenced by a numerical value. Subsequently, the PANC is responsible for transmitting the messages to Node-RED through **Cooja**, enabling us to further forward them from Node-RED to ThingSpeak. We hereby present the received data below.

2 Data and results

2.1 Stop and Wait protocol

Sending and receiving data had been considered established knowledge since the previous challenge, as well as through the exercises. However, the flow control system using the Stop and Wait protocol warrants specific attention. Presented below is an illustration depicting the initiation of Timer0 associated with the re-transmission of a connection-type message.

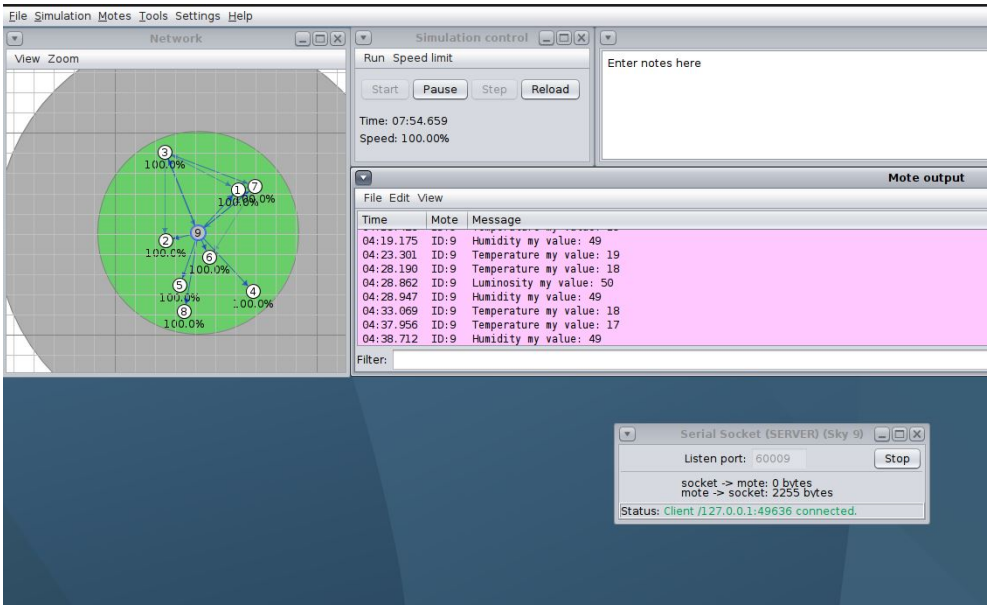
```
451 DEBUG (8): Timer0 for handling 'CONNECTION' re-transmissions fired at 0:0:0.976562510.
452 DEBUG (2): Temperature timer fired at 0:0:0.976562510.
453 DEBUG (2): temp read done 20.425727
```

```
DEBUG (9): Received packet at time 0:0:1.757186874
DEBUG (9):  Payload length 8
DEBUG (9): >>>Pack
         Payload Received
         type: 1
         subtype: 0
         source: 8
         destination: 9
         data: 0
 DEBUG (9): Received connect message! 0:0:1.757186874
DEBUG (9): Timer2 for handling the sending of the packets fired at 0:0:1.757186884.
DEBUG (9): Preparing the message...
DEBUG (9): Packet passed to lower layer successfully!
DEBUG (9): >>>Pack
     Payload length 8
     Payload Sent
DEBUG (8): Packet sent... at time 0:0:1.757354720
DEBUG (8): Received packet at time 0:0:1.759201033
DEBUG (8):  Payload length 8
DEBUG (8): >>>Pack
         Payload Received
         type: 2
         subtype: 0
         source: 9
         destination: 8
         data: 20
 DEBUG (8): Received connack message! 0:0:1.759201033
```

From the images, it is evident that upon receiving the CONNACK following the initiation of an initial re-transmission, the initiation of a subsequent re-transmission is prevented. Furthermore, the re-sending of the subscription message is effectively managed.

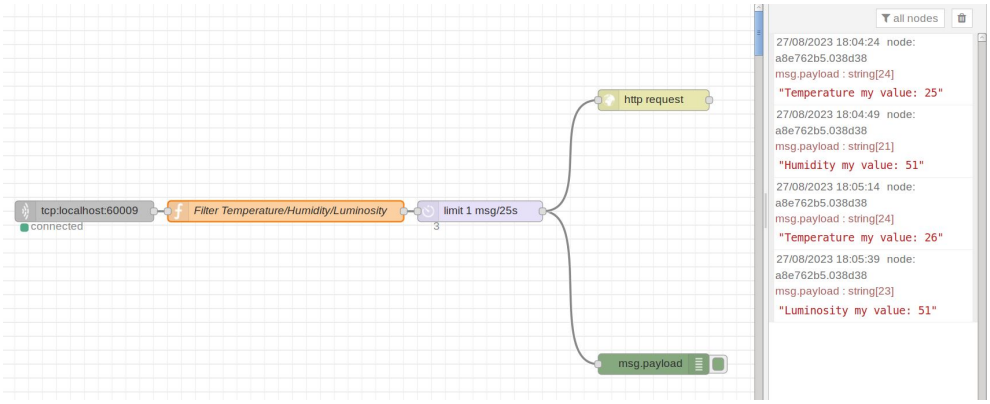
2.2 Cooja

We employ 'Cooja' to transmit our data to Node-RED. Presented below is a screenshot of Cooja illustrating the topology of our network. The message printing is facilitated using the 'printf' function, which we obtained from one of the tutorials. It is important to emphasize that we comment out the lines for printing when utilizing Tossim, as it leads to compilation errors. There are no noteworthy compilation issues encountered when using Cooja.



2.3 Node-red and ThingSpeak

For sending information to Thingspeak, we did not use the **MQTT** node due to several malfunctions with the node. Instead, we resorted to the 'HTTP POST' method. Even after following the instructions provided during the Node-red lab, including inserting the 'ClientID', 'UsernameID', password, and the pattern: 'channels/<channelID>/publish', the node indicates that it is 'connected', but the information does not reach the graphs on Thingspeak. This is why we opted for the alternative method using API keys to write the information from the URL. Below is the diagram of our flow on Node-red and our tables collected from Thingspeak.



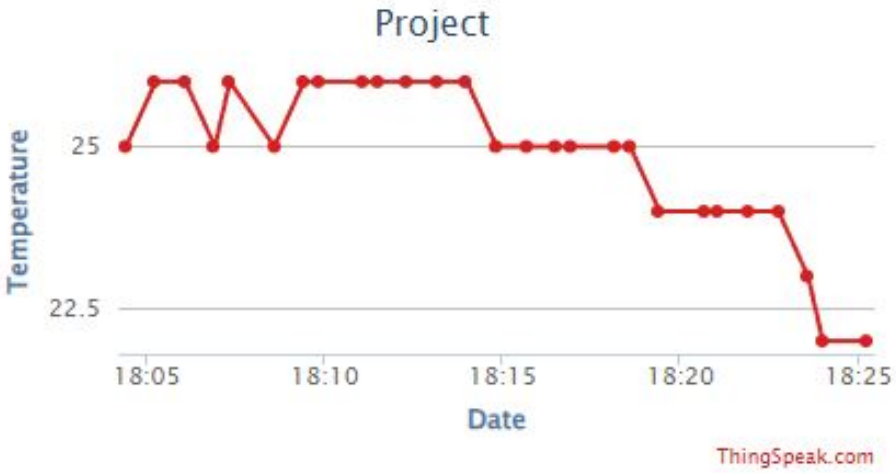
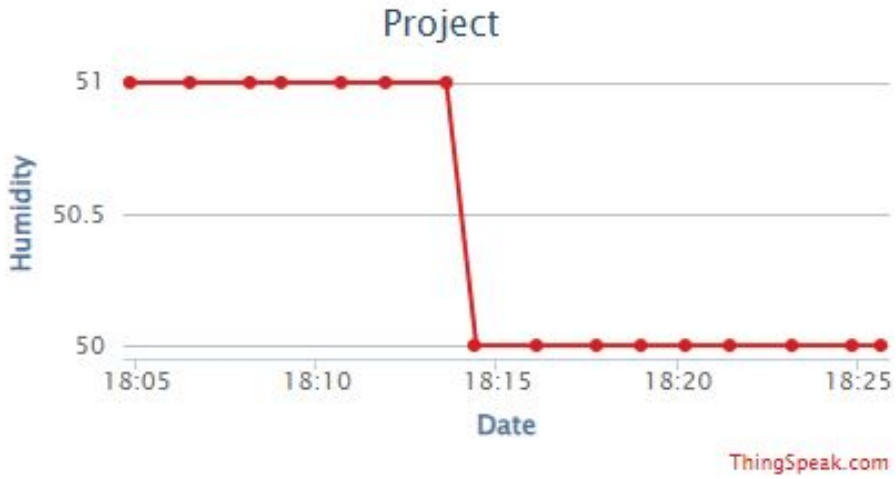
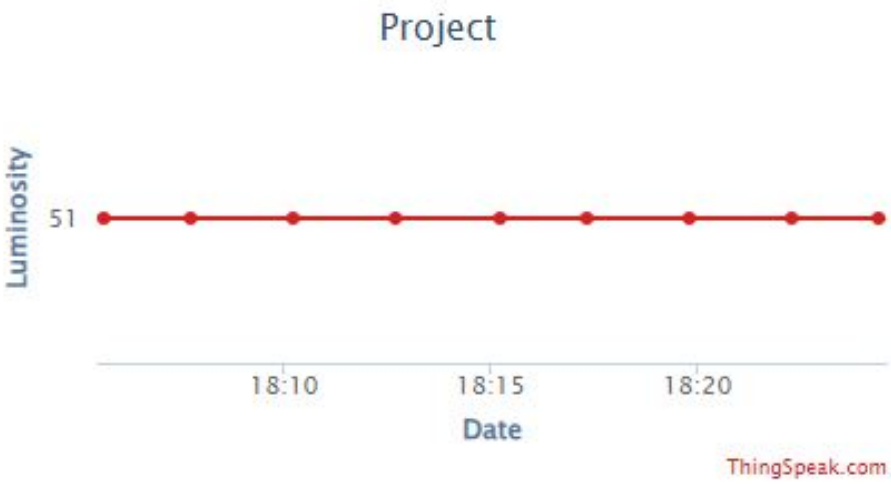


Figure 1: Temeperature





We indicate that the values for luminosity are the same as humidity. We just delayed the recuperation of the data for luminosity by 30seconds compared to humidity.