Politecnico di Milano

FINAL IOT PROJECT REPORT

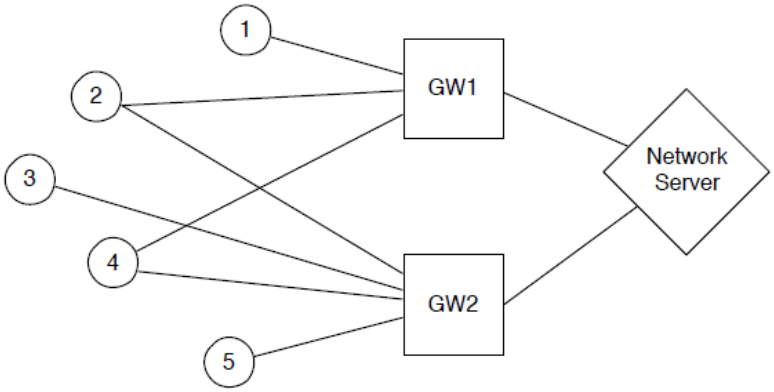
ANNO ACCADEMICO 2022/2023

Angelo Attivissimo
10667094 - 233484

DESCRIPTION

WeatherSensorsNetwork's goal is to simulate a weather monitoring system that reports the temperature, humidity, wind, and air quality of the city of Milan on a spring day. As a result, the temperature is always between 20 and 30 degrees Celsius, the humidity between 0% and 100%, the wind between 0 and 20 kilometers per hour, and the PM2.5 concentration between 0 and 40 $\mu g/m^3$.

Five sensor nodes in the simulation randomly choose to measure one of the aforementioned attributes every 10 seconds and transmit a message with the measurement to one or more gateways. The sensor nodes' messages are forwarded by two gateways to the network server, which responds with an ACK to the sending sensor node and then elaborates on the messages it has received. The network server is connected to Node-Red, and it delivers meteorological data to Thingspeak over the MQTT protocol. Four charts concerning the weather are displayed by Thingspeak.



IMPLEMENTATION

The project is articulated in three parts: Tossim (and Cooja for simulation), Node-Red and Thingspeak.

Tossim

The sensor nodes, gateways, and network server activities are all carried out by Tossim, which serves as the node's operating system. The code is divided into three components: NetworkServerC, SensoreNodeC, and WeatherSensorsNetworkC. A message that is equal for each simulation component is defined in other and is explained in WeatherSensorsNetwork.h.

Sensor Node

This part refers to the code written in SensorNodeP and WeatherSensorsNetworkC.

A sensor node uses the *getUnitMeasure* and *getValue* functions of the SensorNode interface once every 10 seconds to randomly obtain a unit measure and a value that agrees with it and the above-mentioned ranges, then sends a message containing the value measured with its unit measure to its gateway/s and start a timer of one second. Once the timer has expired and the ACK has not yet been received, the message is sent again and the timer is re-started.
I accomplished this using a linked list, where the sensor node appends each message delivered and deletes it when the ACK comes in. When the timer expires, it checks to see if the message appended is still in the list and resends the message if it is.
Every sensor node uses a message_id count to identify each message; it is initialized at 1 and increases by 1

with each message transmitted (for example, two messages sent by sensor node 2 to gateways 1 and 2 share the same message id).

Gateway

This part refers to the code written in WeatherSensorsNetworkC.
A gateway receives a message from a sensor node and forward it to the network server and vice versa.

Network Server

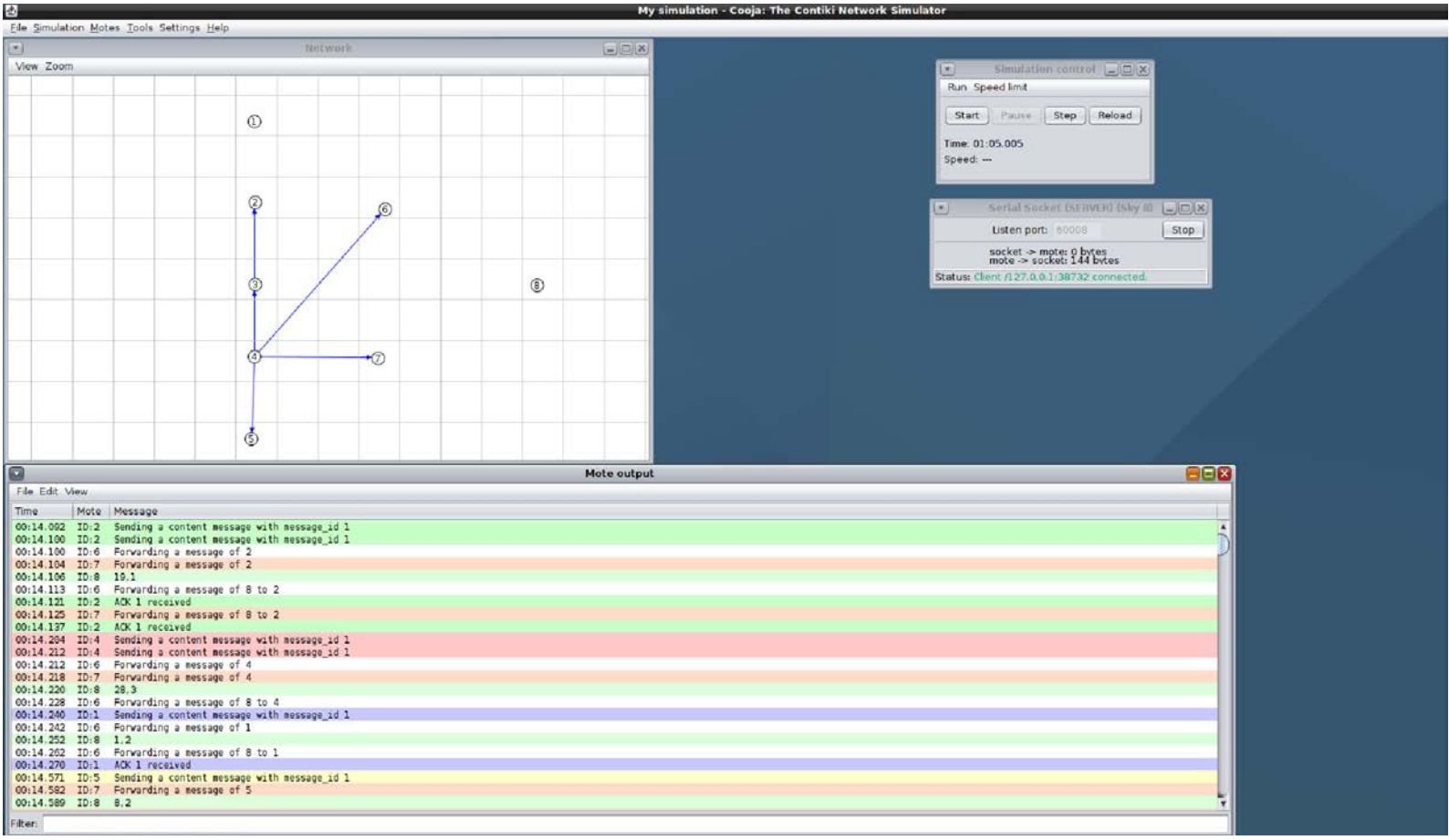
This part refers to the code written in NetworkServerP and WeatherSensorsNetworkC.
When a network server gets a message from a sensor node, it sends an acknowledgement with the same message_id to the gateway that delivered the message. Then, using the NetworkServer interface's *check* method, determine whether the message received is a duplicate; if so, discard it; if not, print the value and the unit measure received. Cooja will then transmit the message to localhost via TCP.

Message

Either a content message or an ACK is identified by the message's type field. The other parameters are concerned with the message's path and measure.

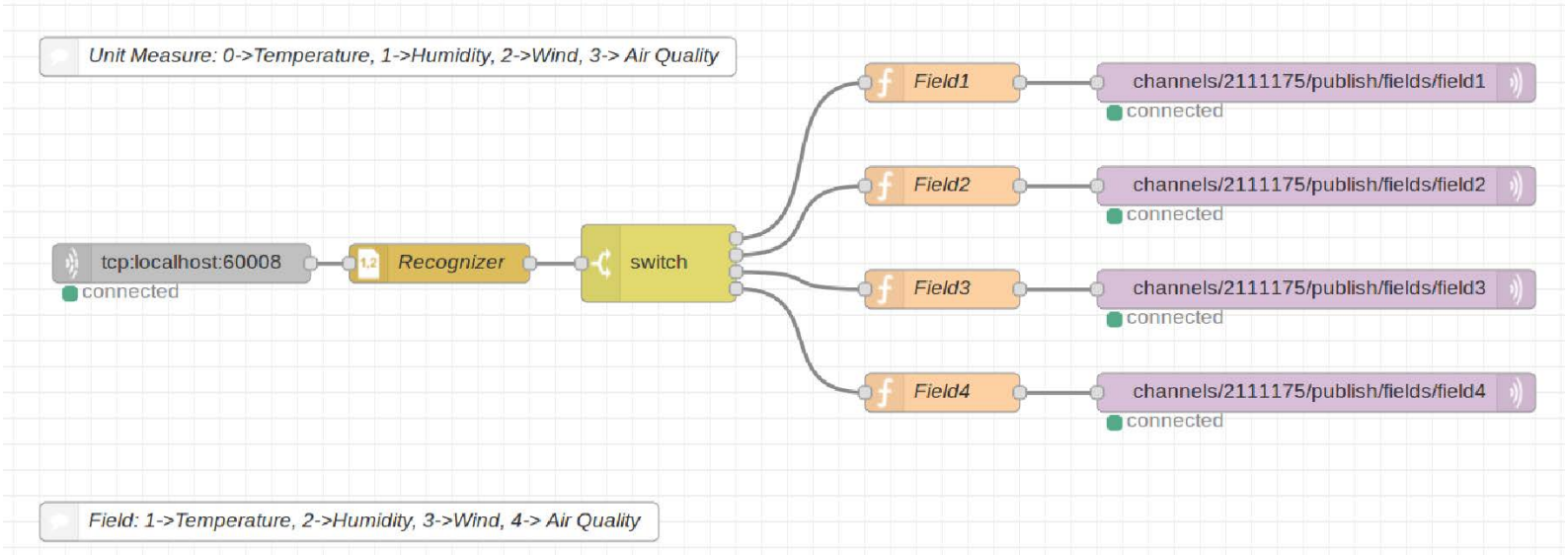
Cooja

I used Cooja for the simulation (remember to compile the code with Telosb), so I can send the value and its unit measure to node-red by utilizing the printf function in the network server portion and setting up a TCP server on the simulation's mote 8 (on localhost and port 60008). The arrangement of the motes is seen in the image below.



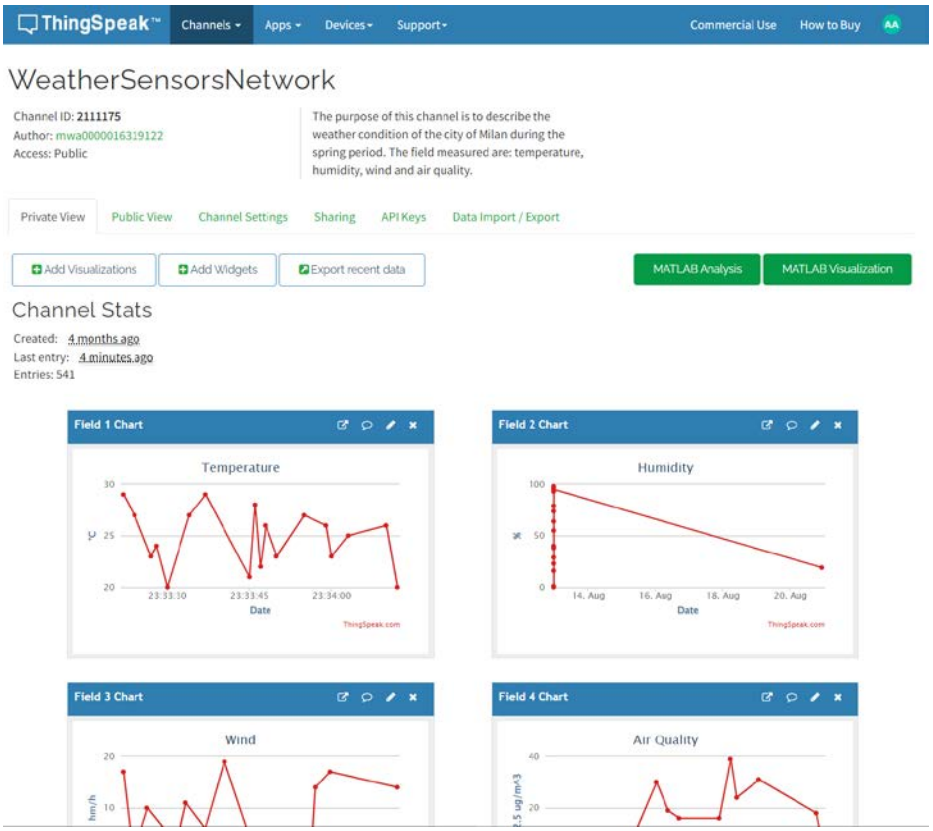
Node-Red

The flow begins with a TCP block that listens on localhost at port 60008, detects the unit of measurement, and selects the appropriate MQTT channel to publish the value measured in it. There is a separate MQTT channel for each of the four attributes, with each channel containing only one type of meteorological characteristic.



Thingspeak

The channel is called WeatherSensorsNetwork, and it features four charts of the measured properties as well as a widget that sends out alerts when the PM2.5 level surpasses the safe level. By subscribing to the corresponding MQTT channel, the charts are updated.



HOW TO USE

Open in a terminal the folder of the project, then insert the command:

- make telosb
- node-red

Go to the URL http://localhost:1880, import the flow from the *node-red* folder's *node-red-source* file, and then deploy.

Open Cooja and arrange the eight Sky motes in a manner similar to the network simulation shown in the network's description image (the first one of the report). Afterward, open a Serial Socket (SERVER) on port 60008 on localhost on mote 8.
You can also use my saved simulation that is located in the *cooja* folder, but you will need to update all of the Contiki firmware with the right location of your *main.exe* file. Go to *File > Open Simulation > Open and Reconfigure > Browse* and select the file from the folder *cooja > my_simulation.csc*. Next, select SkyMoteType as the mote type and contiki firmware from the folder *build > telosb > main.exe* for each mote.

Open the Thingspeak channel page at https://thingspeak.com/channels/2111175.

Finally start the simulation on Cooja.