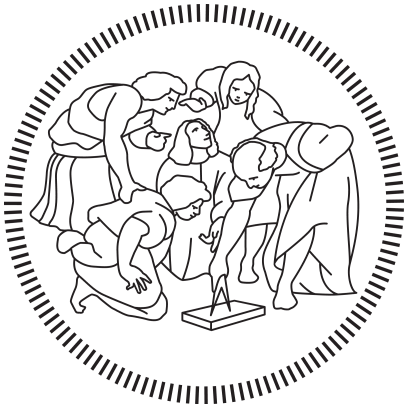
AY 2023/2024



Politecnico di Milano

Internet of Things Challenge - 3

Rishabh Tiwari - rishabh.tiwari@mail.polimi.it - 10987397
Marcos V. Firmino P. - marcosvinicius.firmino@mail.polimi.it - 10914211
Alexander Stephan - alexander.stephan@mail.polimi.it - 10932707

Professor
Redondi Alessandro Enrico Cesare

Version 1.1
April 22, 2024

1 Introduction

In this exercise, we'll be designing a Node-RED flow to manage MQTT message exchange with a local Mosquitto broker. The flow encompasses various tasks, including message publishing, subscription, processing, logging, and interaction with external services like ThingSpeak.

The tasks are outlined as follows:

- Periodically publishing MQTT messages containing randomly generated IDs and timestamps, with logging in a CSV file.
- Subscribing to a specific MQTT topic to receive messages, calculating a value based on the received ID, and processing a CSV file accordingly.
- Sending MQTT publish messages based on certain conditions, with a rate limit on publication, and formatting the payload accordingly.
- Plotting temperature values from publish messages in a Node-RED chart and saving relevant data to a CSV file.
- Handling MQTT ACK messages, logging them in a CSV file, and updating a global counter, followed by updating a ThingSpeak channel.
- Managing message processing to ensure a limit of 80 ID messages is not exceeded.

This exercise will demonstrate proficiency in MQTT communication, data manipulation, integration with external platforms, and flow control within Node-RED.

2 Node-red flow

This is a print-screen of our node-red flow

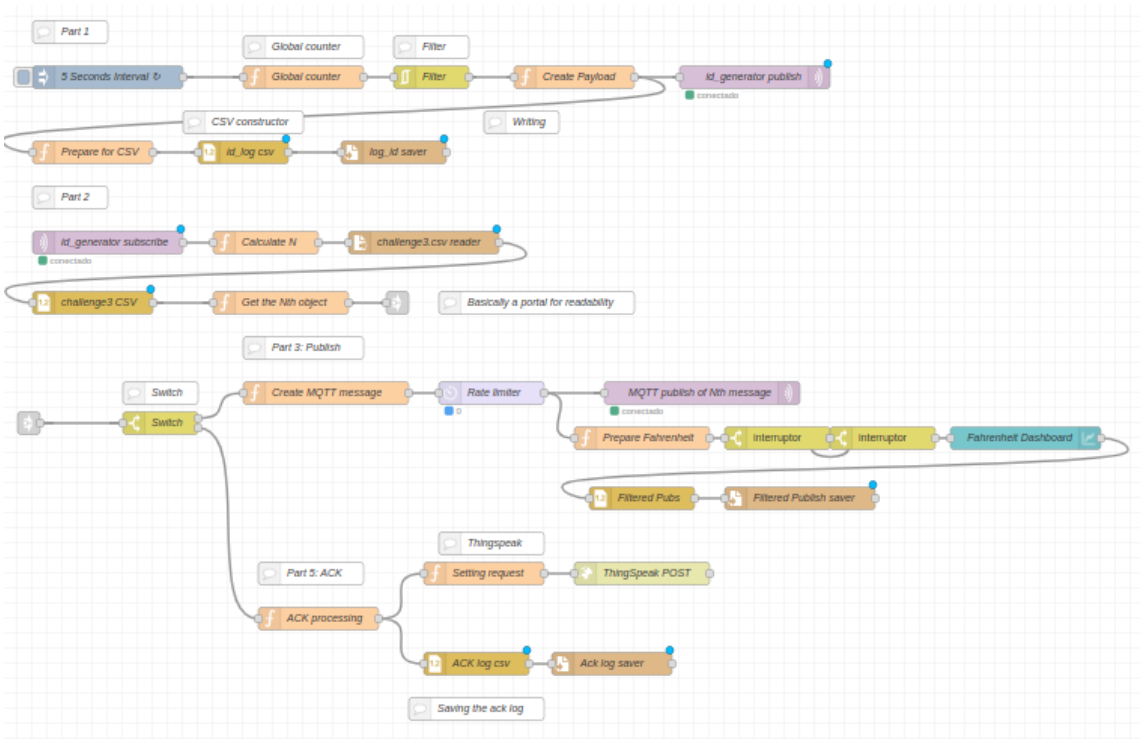


Figure 1: Node-red flow

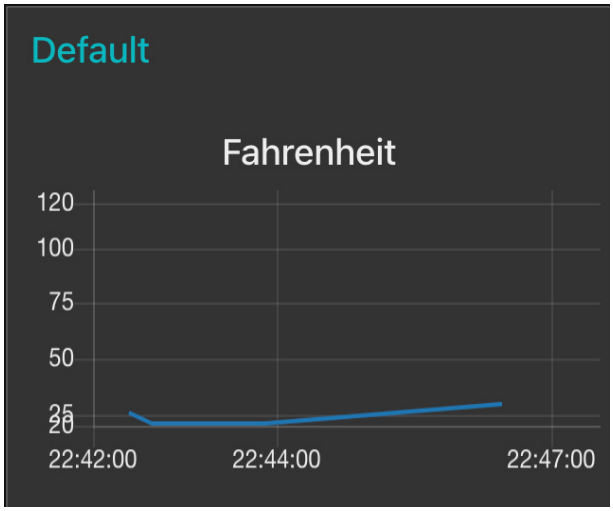


Figure 2: Obtained Fahrenheit graph for the first couple of minutes

3 General Overview

We labelled our node red flow to separate it into different sections. Here is a general overview:

Part 1: We inject a timestamp node to limit the messages sent to the local broker to 5 s. Then, we have some counter logic in place to make sure we don't send any messages after 80 received messages. This is done in the following function node. We save this counter in the message and the flow context. Then, we have a filter node in place. It is set to "block unless value" changes. It inspects if the counter changes, which is capped at 80. Then, we create a JSON object for message payload in the next function node, containing a random ID and a timestamp. The following MQTT out node publishes it to the topic "challenge3/id_generator". In the "Prepare for CSV" function node, we get the counter from the flow scope to use it as No. and also create the other columns (TIMESTAMP and ID). The following CSVs node converts the created object to a CSV in Node-RED. The following "write file node" materializes the changes.

Part 2: In the first node, we receive the subscription from the local broker. In the next function node, we retrieve the ID from the payload and calculate it mod 7711 to get N. Then, we read the challenge3.csv file provided via Webeep. We parse it as a csv, using a csv-node. Then, we wrote a function to retrieve the n-th element in the CSV. We subtract -1 from in the index to account for the 0 based indexing. Then, we retrieve the object using `JSON.stringify`. Then, we use a link for readability that forwards the message to the next part.

Part 3: First, we perform string matching in the stringified JSON object for the previously retrieved row. The switch node is looking for either "ACK" or "Publish message" as the type. If we didn't receive an ACK, we create a new MQTT message. The following function node creates the actual message content. Then, we use a delay node as a rate limiter, setting the rate to 4 messages per 1 minute. Then, we publish the message using a "mqtt out node". Using this data, we create the actual plot. See the explanation of each node.

If we received an ACK node, we again process the message in a function node. Essentially, it creates a message of the right type, according to the values found in the received n-th value. Then, we create a CSV file with the "ack_log.csv" to track the received acks. The following "write file node" materializes the CSV on our hard drive. Additionally, after receiving the ACK, we also publish it to ThingSpeak using another function node. It essentially sets the API key and the API URL. Finally, we have a "http request node" that sends the just created message via an HTTP post request.

4 Explaining each node

Below there is the list of all nodes and a quick description of its purpose:

- 5 seconds interval: Sends the timestamp value for the next node each 5 seconds

- Global counter: Counts the amount of messages sent (only 80 messages allowed according to description)
- Filter: Filters the messages out after the 80th, thus let by only the first 80
- Create Payload: Create the payload of the random value generated (ID) and current timestamp
- id_generator publish: MQTT node that publishes the ID and timestamp produced before
- Prepare for CSV: Prepares the information (ID,timestamp) to be printed into CSV
- id_log CSV: Sets up the CSV file of the id_log, for example the header
- log_id saver: Actually access the file system and updates/saves the information of the log in the CSV file
- id_generator subscribe: MQTT node that subscribes to the id_generator, receiving the information (ID and timestamp)
- Calculate N: Given the received ID, calculates the corresponding N
- challenge3.csv reader: Read the file challenge3.csv
- challenge3 CSV: Receives the challenge3.csv data in CSV format, then convert it to JSON
- Get the Nth object: With the information of challenge3.csv in JSON, access the corresponding Nth message
- Switch: Receives the Nth object and redirect the flow: if it contains an "ACK" or contains a "Publish Message"
- Create MQTT message: In case of a Pulish Message, this node reads the Nth message and extract all publish info out of it. It created the correct pair of topic-payload for each one (or more) publish messages of the Nth entry
- Rate limiter: Limits the rate of publication for 1 message per 4 minutes
- MQTT publish of Nth message: Publish each message received from the Nth object into the corresponding topic
- Prepare Fahrenheit: Reads the information of the message and extract the Fahrenheit value and its average, as well of preparing the message with the Fahrenheit value (if any)
- Interrupter: Filters out the message, and passes only if the message has a temperature in Fahrenheit
- Fahrenheit Dashboard: The dashboard showing the Fahrenheit information
- Filtered Pubs: Prepares to save the filtered message with Fahrenheit value and average
- Filtered Publish saver Access the file system and updates the CSV file with the temperature informatiion
- ACK processing: In case of an ACK, extract the type of the ACK out of the body of the message
- Setting request: Sets up the HTTP request to be sent to ThingSpeak
- ThingSpeak POST: Perform the HTTP GET request to ThingSpeak
- ACK log csv: Prepares the ACK information to be saved into CSV (header for e. g.)
- Ack log saver: Actually access the file system and updates/saves the information of the ACKs in the CSV file