IOT Challenge #3

Francesco Spangaro - Luca Tosetti

27 April 2025

POLITECNICO
MILANO 1863

Academic Year 2024 - 2025

Contents

1 Introduction

We were tasked with creating a Node-Red flow to periodically publish MQTT messages to the local mosquitto broker (127.0.0.1:1884), to the topic ***challenge3/id_generator***. Messages should be sent with a rate of 1 message every 5 seconds. Each message should contain in the payload a JSON string with a random **ID** in the range $[0, 30000]$ and the timestamp of message generation.

Message payload example: {"id": 12345, "timestamp": 1710930219}

When sending the message, we need to save its payload in a CSV file named ***id_log.csv***. The file needs to be formatted as: ***No., ID, Timestamp***, with ***No.*** being an auto-incremental index indicating the number of the row.
In another branch of the same Node-Red flow, we have to subscribe to the topic ***challenge3/id_generator*** in the local broker (127.0.0.1:1884). After receiving a message from the subscription, we need to take the message's ID and compute the modulo of the division: $N = ID \bmod 7711$. At every received message, we need to process the ***challenge3.csv*** file and take the message M with frame number equal to N.

- If M is an MQTT PUBLISH message we have to send a publish message to the local broker, with the same topic and payload as M, but with the timestamp of the new message generation and id the id we used to evaluate N. The rate of published messages in this step has to be limited to four per minute.
  Additionally, if the publish message contains in its payload a temperature in Fahrenheit, we need to plot this value in a Node-Red chart. We are also requested to save the payload of these messages in a csv file named ***filtered_pubs.csv***.
- If M is an MQTT ACK message, we need to increment a global ACK counter, then save the message into a CSV file named ***ack_log.csv***, in which we save the ack type of M and its id. After having saved the ack, we need to send the value of the updated ACK counter to our thingspeak channel using HTTP API. The link is: `https://thingspeak.mathworks.com/channels/2931739`.
- If M is not a PUBLISH nor an ACK message, M is discarded.

Our flow needs to be programmed to stop after receiving 80 id messages from the subscriptions.

2 Implementation

The final obtained flow is:

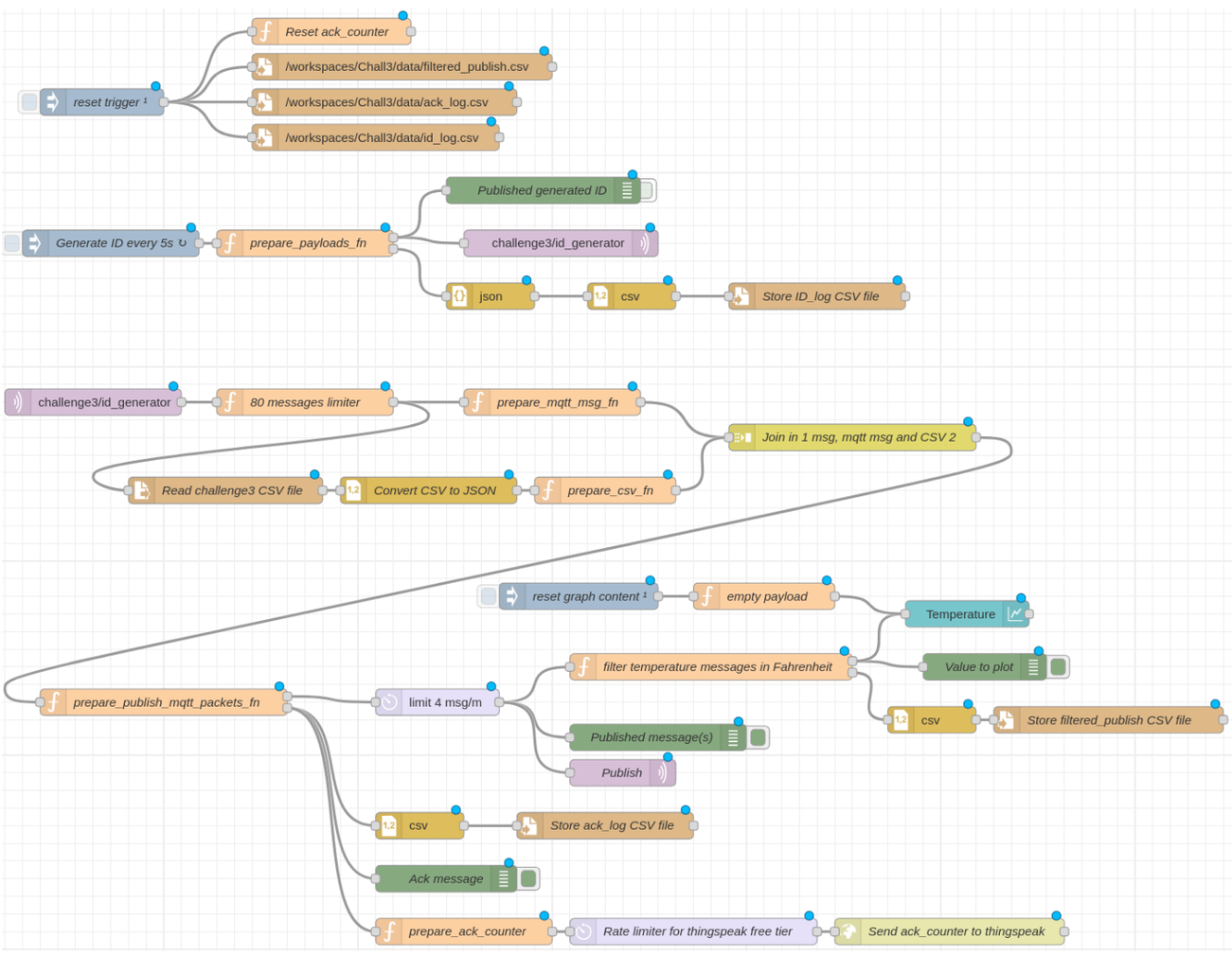

Figure 1: Complete flow

The flow can be divided into three main parts.

2.1 Message generation

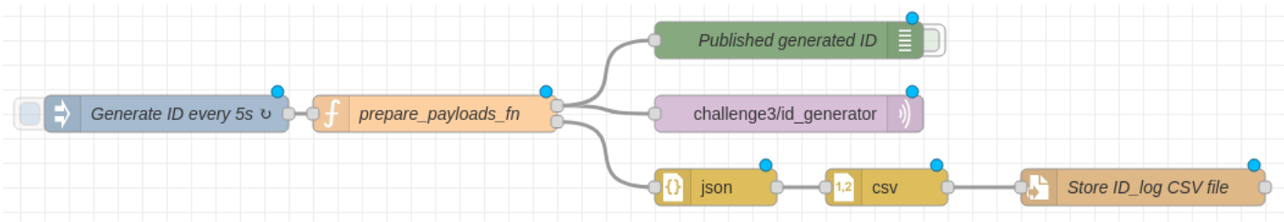

Figure 2: Generate-message flow branch

This flow branch is in charge of generating the messages that will be published to the topic ***challenge3/id_generator***. Every 5 seconds it generates a message using the function:

```
// Max ID value
const MAX = 30000;
// Taking a random number
var random = Math.floor(Math.random() * MAX);
// Taking current timestamp
var timestamp = Date.now();

// Updating the id message counter to save it in the CSV file
var counter = context.get("counter");
counter += 1;

// Payload to send to the MQTT broker
var payload_to_broker = {
    "id": random,
    "timestamp": timestamp
};
// Payload to send to the CSV
var payload_to_csv = {
    "No.": counter,
    "ID": random,
    "TIMESTAMP": timestamp
};

context.set("counter", counter);
// Message to send to the MQTT broker
var msg1 = {
    payload: payload_to_broker
};
// Message to send to the CSV file
var msg2 = {
    payload: payload_to_csv
};

return [msg1, msg2];
```

The message is then published on the topic, and in parallel saved in the ***id_log.csv*** file.

2.2 Message reception

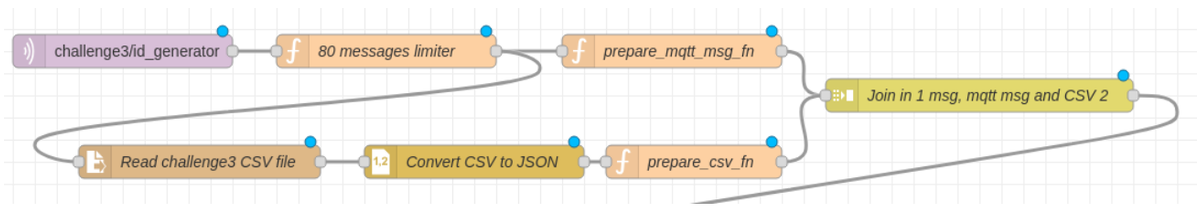

Figure 3: Receive-message flow branch

This flow branch is in charge of receiving messages published on the topic ***challenge3/id_generator***. After receiving the messages, it reads from the ***challenge3.csv*** file the corresponding message M and prepares the csv. In parallel, the MQTT message is prepared. These two are then merged in a single message and sent to be parsed. If we have already received 80 messages on the topic, a null message is returned, stopping the evaluation of the message, and, therefore, stopping the flow.

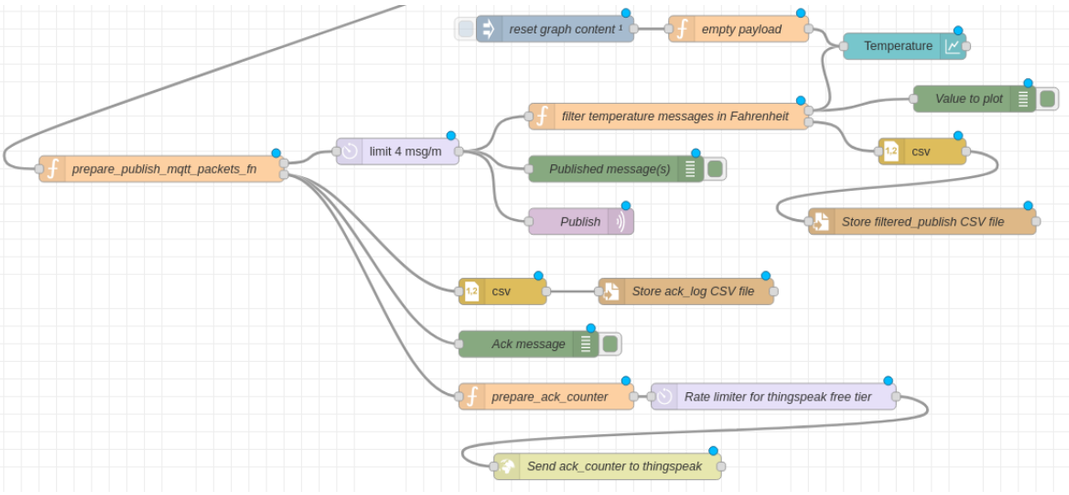

Figure 4: Parse-message flow branch

The messages are parsed and those that are MQTT PUBLISH messages are prepared in order to be published. The messages here are limited to a rate of 4 messages per minute. The ***prepare_publish_mqtt_packets_fn*** node does this with the function:

```
// Gather data from input msg
const csvData = msg.payload.csv
const mqtt_mess = msg.payload.mqtt_message

// Retrieve msg ID from the MQTT msg received
const id = mqtt_mess.id;

// Compute N and retrieve the corresponding msg from the csv
var N = id % 7711;
var msg_from_csv = csvData.filter((row) => row["No."]==N)[0];

// Initialize the messages to return
var msgs_to_publish = null;
var ack_msg = null;

// If the computed N is 0, ignore the messages
if (N == 0) {
    return [null, null];
}
// Filtering the MQTT messages of type PUBLISH
if (msg_from_csv["Info"].includes("Publish Message")) {
    msgs_to_publish = [];
    // Splitting the different PUBLISH messages in the same MQTT packet
    var messages = msg_from_csv["Info"].split(",");
    // Necessary to split for "[", because some Publish msg has
    // (id=<number>) between "Publish Message" and the topic
    var topics = messages.map((el)=> el.split("[")[1].slice(0, -1));
    // Handling malformed messages' payload
    try {
        var payloads = JSON.parse(`[${msg_from_csv["Payload"]}]`);
    } catch (e) {
        return [null, null];
    }

    // Building the messages to publish
    for (var i = 0; i<topics.length;i++) {
        msgs_to_publish.push({
                "timestamp": Date.now(),
                "id": id,
                "topic": topics[i],
                "payload": (i < payloads.length
                    ? (payloads[i]==null
                        ? {}
                        : payloads[i])
                    : {}),
        });
    }
}
// Filtering the MQTT messages of type ACK
if (msg_from_csv["Info"].includes("Ack")) {
    // Obtaining the type of the message without its ID
    var type = msg_from_csv["Info"].split(" (")[0];
    // Updating the global ACK message counter
    var ack_counter = global.get("ack_counter") + 1
    global.set("ack_counter", ack_counter);

    // Building the ACK message to store
    ack_msg = {
        "payload": {
            "No.": ack_counter,
            "TIMESTAMP": Date.now(),
            "SUB_ID": id,
            "MSG_TYPE": type
            }
    };
```

```
}
return [msgs_to_publish, ack_msg];
```

The flow then parses the publish messages containing Fahrenheit temperatures and sends the values to a Node-Red plot with the function:

```
// Filter the PUBLISH messages that contain a temperature in fahrenheit
if (msg.payload["type"]=="temperature" && msg.payload["unit"]=="F") {
    // Computing the average temperature
    var temps = msg.payload["range"];
    var avg_temp = Math.round((temps[0] + temps[1])/2);

    // Creating the message to send to the chart node to plot the temp value
    var value_to_plot = {
        "payload": avg_temp
    };

    // Updating the counter for filtered publish messages
    var count = context.get("filtered_pubs_counter");
    count += 1;
    context.set("filtered_pubs_counter", count);

    // Building the message to store
    var msg_to_save = {
        "payload": {
            "No.": count,
            "LONG": msg.payload.long,
            "LAT": msg.payload.lat,
            "MEAN_VALUE": avg_temp,
            "TYPE": msg.payload.type,
            "UNIT": msg.payload.unit,
            "DESCRIPTION": msg.payload.description
    }};
    return [value_to_plot, msg_to_save];
}

return [null, null];
```

In parallel, the MQTT PUBLISH messages are published on the respective topics, the MQTT ACK messages are saved in the ***ack_log.csv*** file and the global ack counter is incremented, then the new value is sent to thingspeak.

2.3 Reset trigger

This flow branch is used to reset all graphs and the ack counter. Is called on deploy, asserts that all files and global counters are reset.

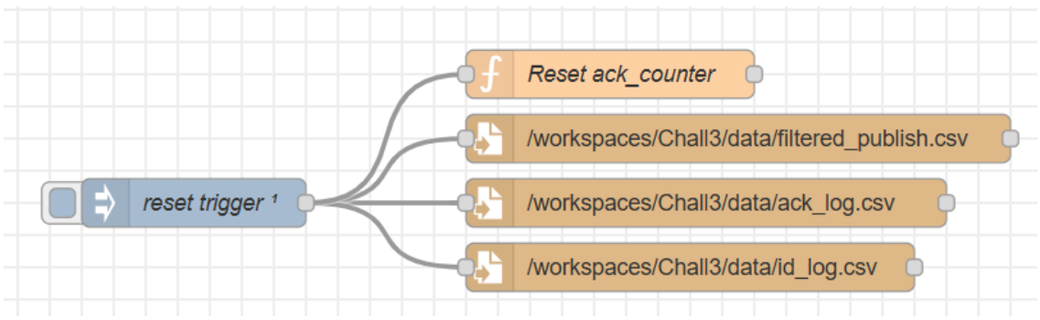

Figure 5: Reset trigger flow branch

3 Charts

The resulting graphs obtained for the temperature reading and ack counters are as follows:

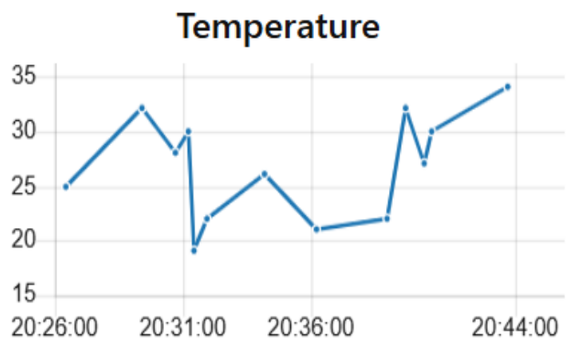

Figure 6: Temperature graph

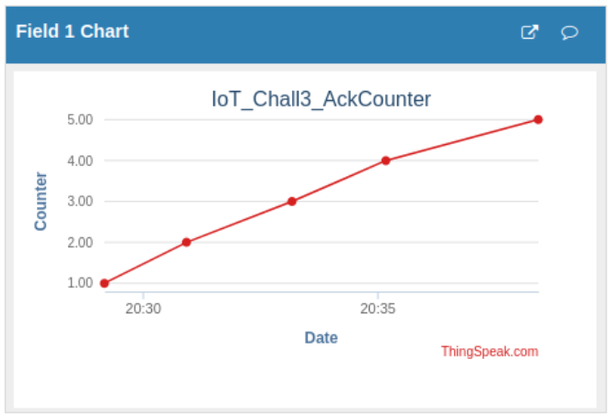

Figure 7: Ack counter graph