Internet of Things – A.Y. 2020/2021

Home Challenge #2 – Diego Savoia (Personal Code: 10535515)

Analysing the "traffic.csv" file, I understood that the type of each MQTT message (like "Publish Message" or "Subscribe Ack" and so on) is repeated twice, and the payloads are all at the end of the CSV line. Assuming that the order of the type of the MQTT messages and the order of the payloads are the same, I associated the first message with the first payload, the second message with the second payload, and so on.

Here, a brief explanation of the Node-RED flow (the ThingSpeak channel is at this page):

- Start – "inject" node to start the flow.
- Read CSV line – "file in" node to read the "traffic.csv" file line-by-line (one message per line).
- Convert CSV line into JS object – "csv" node to convert each line read by the previous node into a JS object, useful to work with it in an easier way.
- Publish Messages and hex payloads – "function" node that scans the items of the JS object one by one, which is equal to scanning the "traffic.csv" columns one by one. If an item contains the string "Publish Message", then it sets a Boolean variable "found" to true and it goes on to the next item. If an item contains the string "7b" (which corresponds to the term "{" in the hexadecimal MQTT payloads) and at least a Publish Message was found (so if the Boolean variable "found" was set to true) then it goes on to the next item. Instead, if none of these two conditions is satisfied, then the item in the JS object is deleted, since it is useless for our analysis. It returns a message containing the (possibly updated) JS object.
- Useful topics and corresponding hex payloads – "function" node that scans the JS object and creates an output message containing an array, which is composed (in each position) of a string containing the topic of the publish messages to be considered, followed by the corresponding hexadecimal payload. In particular, each time it finds one of the four topics (the ones we consider for the challenge) in an item of the JS object, it scans again the object to find a payload (which starts with "7b"). Then, it saves them and it replaces them with empty strings in the JS object, so that at the next iteration they will not be there. This is useful so that the first topic will be associated with the first payload; then, they are both replaced with empty strings, so the second topic will be associated with the second payload, and so on.
- Topic+Hex payload – "split" node which splits the previously obtained array: hence, each output is a string composed by the topic and the corresponding hexadecimal payload. This is useful since, in principle, in the original JS object we could have multiple MQTT publish messages with a useful topic, so the array could be longer than one.
- Topic division – "switch" node that checks which of the four topics is contained in the input message, and routes it accordingly.
- Take and convert hex payload – Four "function" nodes (one per topic) useful to isolate the hexadecimal payload and to convert it into a readable ASCII string.
- Prepare MQTT message for field* - Two "function" nodes (one per field: one for the two "plc" topics and one for the two "hydraulic_valve" topics) useful to extract the "value" field of the ASCII string and to prepare the MQTT message properly for the publication on ThingSpeak.
- Rate limiter – "delay" node to rate-limit the sending of the MQTT messages. This is useful because of ThingSpeak limitations, to assure that every message is actually received.
- Send MQTT message – "mqtt out" node to send the MQTT messages to ThingSpeak.