Report Middleware for IoT

Middleware for IoT

BRUNETTO Marie
BIENDOU Brian
5 ISS
Année universitaire 2024/2025

INSA Toulouse
135, Avenue de Rangueil
31077 Toulouse Cedex 4 - France
www.insa-toulouse.fr

Report Middleware for IoT

Middleware for IoT

BRUNETTO Marie
BIENDOU Brian
5 ISS
Année universitaire 2024/2025

INSA Toulouse
135, Avenue de Rangueil
31077 Toulouse Cedex 4 - France
www.insa-toulouse.fr

Table des matières

Introduction

The Internet of Objects (IoT) thrives on seamless communication between devices, sensors, and systems. At the heart of this interconnectivity lies MQTT (Message Queuing Telemetry Transport), a lightweight, efficient protocol designed specifically for IoT environments. Developed for constrained networks and low-bandwidth scenarios, MQTT enables reliable communication between devices by using a publish-subscribe model. In our labs session, we had the opportunity to try out the MQTT services provided by the software Mosquitto, and to create a small application that we will describe through this report.

1 Definition of MQTT

The MQTT (Message Queuing telemetry Transport) protocol is a lightweight communications solution used primarily in IoT systems. Bases on the IP protocol, MQTT uses a publish/subscribe model for efficient, asynchronous communication between connected devices. It operates over TCP/IP, which guarantees reliable message transport with minimal bandwidth consumption, making it ideal for low-speed resources-constrained environments. Various versions of MQTT are available, including 3.11 3.1.1 and 5.0, each with improvements in terms of flexibility, error control and support for complex use cases. In terms of security, MQTT can incorporate mechanisms such as TLS (Transport Layer Security) for data encryption and authentication using certificates or user IDs/passwords.

In a domestic context, a system using MQTT and a Mosquitto server could, for example, control a light using a button and a brightness sensor. The topics required would include :

— home/button to publish the state of the button (manually on or off).
— home/sensor/luminosity to publish the values of the luminosity sensor.
— home/light to control the state of the light (on/off).

The luminosity sensor will publish its data on the associated topic, while a client will subscribe to this topic to automatically activate the light if the threshold is reached. At the same time, the button will publish its status on its own topic, to which the light controller will also subscribe to enable manual control. This architecture ensures fluid, responsive communication between the system components.[1, 2]

2 NodeMCU

NodeMCU is an open-source firmware used for ESP32 and ESP8266 boards. It is based in the Lua language, with a programming model similar to NodeJS. However, NodeMCU is also able to run C or C++ binary, convenient for IDE such as Arduino IDE.[3]

NodeMCU had C modules for ESP8266, amongts which we can find the GPIO module, abbreviation of General Purpose Input Output. The module is composed of 12 pins, with the following mapping :

0	ESP8266 pin
0	GPIO16
1	GPIO5
2	GPIO4
3	GPIO0
4	GPIO2
5	GPIO14
6	GPIO12
7	GPIO13
8	GPIO15
9	GPIO3
10	GPIO1
11	GPIO9
12	GPIO10

TABLE 1 – Technologies de sécurité utilisées dans différents protocoles MACs.

3 Development of a small application

The goal of this exercise was to create a way to subscribe the light, so it turns on or off depending on the message received on the topics. In a similar way, we want to be able to publish on a topic using the button. To develop this small application, we need to follow a few step :

1. Get familiar with the board and modules,
2. Define the variables of the system,
3. Program and upload the program on the board
4. Test the application

3.1 Getting familiar

The first step was to get familiar with the board and module. To develop this small application, we had a development board ESP8266, a button module. For the light, we will use the built-in light embedded on the ESP board.

For the development, we used Arduino IDE, as it is convenient for ESP boards. Some example of basic programs were provided and useful to practice. We were also able to install a library called pubsubclient by Nick O'Leary, which give us examples to work with the MQTT technology.

Finally, for the network, we connected to the teacher's network "asni", on which he hosted the Mosquitto broker open to publication and subscription. The address of the device hosting the broker is 10.0.1.254.

3.2 Define the variable of the system

A small step that we needed to take was to define the variable of our system.

First of all, we need to define a topic on which the board will be able to publish and subscribe. We decided to set on "insa/toulouse/gei/105/Brian_Marie".

It was also important to define how the communication will be done between the button and light, what will be the message. We decided to define that when the light received 1, it turned on. Otherwise, if it receives 0, it turns off.

3.3 Program and upload

Using Arduino IDE, we were proposed a few examples that allowed us to see the structure of a subscription and publication using MQTT. A few minor problems were encountered, mainly through typos in the identification and unmatch baud rates, but nothing we were not able t overcome quite fast. After adapting the code to match our requirements, we were ready to test out the application.

3.4 Test the application

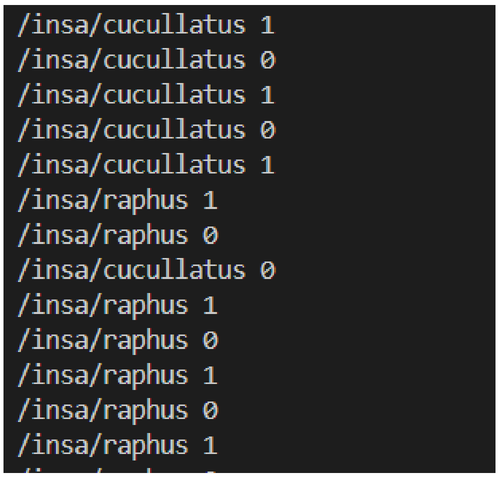
For the test, we were able to work with other students to try the wireless communication possible with MQTT. To proceed, we first had to agree on a topic and the message. For the topic, we decided to choose two different topic depending on the sense of the communication, so we could both test that our button and light were properly working with MQTT. The names on the topic we decided were "/insa/raphus" and "/insa/cucullatus", coming from the binomial name of the dodo bird, Raphus Cucullatus.

For the light, we decided to keep the previous way of operating, meaning turning on when 1 is received, turning off when 0 is received. In the Figure 1, we used the following command to see all the publication that we were interested in :

```
./mosquitto\_sub.exe -h 10.0.1.254 -p 1883 -t "/insa/#" -v
```

We can see that pressing the buttons publishes both 0 and 1 commanding the lights for both topics. We were successfully able to communicate between the device using MQTT.

A more complex application using the provided light sensor could have been interesting to work in, using the sensor as a publisher to see if a threshold of light is exceeded.



```
/insa/cucullatus 1
/insa/cucullatus 0
/insa/cucullatus 1
/insa/cucullatus 0
/insa/cucullatus 1
/insa/raphus 1
/insa/raphus 0
/insa/cucullatus 0
/insa/raphus 1
/insa/raphus 0
/insa/raphus 1
/insa/raphus 0
/insa/raphus 1
```

Figure 1 – Observation of the publications of our systems

When limit set to container for contentinstance, those can be created, but will not be attached

ty=2 = ae ty=3 : container

import sys import requests import json def handleResponse(r) : print (r.status$_{c}ode)print(r.headers)$

—————————Functions Definition————————— def createObject(payload, ty, url) : payload = payload $_{h}eaders = "X-M2M-RVI" : "3", RVIversion" X-M2M-Origin"$

$False), indent = 4)r = requests.post(url, data = payload, headers =_h eaders)returnr$

def createAE() : return createObject('"m2m :ae" : "rn" : "Notebook-AE","api" : "NnotebookAE","rr" : true,"srv" : ["3"]', 2, "http ://localhost :8080/cse-in")

def createCT() : return createObject('"m2m :cnt" : "rn" : "Container"', 3, "http ://localhost :8080/cse-in/Notebook-AE")

def createCI() : return createObject('"m2m :cin" : "cnf" : "text/plain :0", "con" : "Hello, World !"', 4, "http ://localhost :8080/cse-in/Notebook-AE/Container")

————————MAIN———————— print("Creation of the AE") result = createAE() print("Result : ") handleResponse(result) print("") print("Creation of the Container") result = createCT() print("Result : ") handleResponse(result) print("") print("Creation of ContentInstance") result = createCI() print("Result : ") handleResponse(result) print("") print("Creation of ContentInstance") result = createCI() print("Result : ") handleResponse(result)

————————————————————

Creation of the AE Result : 403 'Server' : 'Werkzeug/3.0.3 Python/3.12.4, ACME 0.10.2', 'Date' : 'Mon, 09 Dec 2024 10 :48 :05 GMT', 'X-M2M-RSC' : '4117', 'X-M2M-RI' : '123', 'X-M2M-RVI' : '3', 'X-M2M-OT' : '20241209T104805,284559', 'Content-Type' : 'application/json', 'Content-Length' : '57', 'Connection' : 'close' "m2m :dbg" : "Originator has already registered : Cmyself"

Creation of the Container Result : 409 'Server' : 'Werkzeug/3.0.3 Python/3.12.4, ACME 0.10.2', 'Date' : 'Mon, 09 Dec 2024 10 :48 :05 GMT', 'X-M2M-RSC' : '4105', 'X-M2M-RI' : '123', 'X-M2M-RVI' : '3', 'X-M2M-OT' : '20241209T104805,291690', 'Content-Type' : 'application/json', 'Content-Length' : '38', 'Connection' : 'close' "m2m :dbg" : "resource already exists"

Creation of ContentInstance Result : 201 'Server' : 'Werkzeug/3.0.3 Python/3.12.4, ACME 0.10.2', 'Date' : 'Mon, 09 Dec 2024 10 :48 :05 GMT', 'X-M2M-RSC' : '2001', 'X-M2M-RI' : '123', 'X-M2M-RVI' : '3', 'X-M2M-OT' : '20241209T104805,296372', 'Content-Type' : 'application/json', 'Content-Length' : '272', 'Connection' : 'close' "m2m :cin" : "cnf" : "text/plain :0", "con" : "Hello, World !", "ri" : "cin5310188860178817342", "pi" : "cnt1891837642757666682", "rn" : "cin$_gMxIYKNbhN", "ct" : "20241209T104805, 295391", "lt" : "20241209T104805, 295391", "et$
$"20251209T104805, 295414", "ty" : 4, "cs" : 13, "st" : 3$

Creation of ContentInstance Result : 201 'Server' : 'Werkzeug/3.0.3 Python/3.12.4, ACME 0.10.2', 'Date' : 'Mon, 09 Dec 2024 10 :48 :05 GMT', 'X-M2M-RSC' : '2001', 'X-M2M-RI' : '123', 'X-M2M-RVI' : '3', 'X-M2M-OT' : '20241209T104805,300545', 'Content-Type' : 'application/json', 'Content-Length' : '272', 'Connection' : 'close' "m2m :cin" : "cnf" : "text/plain :0", "con" : "Hello, World !", "ri" : "cin6707426738440962861", "pi" : "cnt1891837642757666682", "rn" : "cin$_eYDAko9IQg", "ct" : "20241209T104805, 299560", "lt" : "20241209T104805, 299560", "et" :$
$"20251209T104805, 299593", "ty" : 4, "cs" : 13, "st" : 4$

Conclusion

After these 5 hours of lab session, we got to familiarize ourselves with MQTT technologies, especially with the principle of publishing and subscribing using an intermediate MQTT Broker on a private network. We were able to develop a small application using Mosquitto to communicate in a wireless manner the command of a publishing button to a subscribed light. The research as well as the application helped us to deepen our understanding of this technology, and the diversity that offers the Internet of Objects, and we hope to be able to apply this type of services in our future projects or work.

Références

[1] *Essential MQTT Architecture Considerations for IoT Use Cases.* URL : `https://www.hivemq.com/mqtt-architecture` (visité le 26/11/2024).

[2] *MQTT in IoT : An Architecture for Connected Devices.* URL : `https://blog.paessler.com/mqtt-in-iot-an-architecture-for-connected-devices` (visité le 26/11/2024).

[3] *NodeMCU Documentation.* URL : `https://nodemcu.readthedocs.io/en/release/` (visité le 26/11/2024).