Middleware for IoT
MQTT

5A ISS
2024-2025

Noël Jumin
Clément Gauché

Noël Jumin
Clément Gauché

Noël Jumin
Clément Gauché

Introduction

In the context of the Machine-to-Machine (M2M) course, this lab spanned two sessions and aimed to provide hands-on experience with MQTT, a lightweight messaging protocol designed for machine-to-machine communication and IoT applications. Through this lab, we gained a deeper understanding of MQTT's architecture, its practical implementation, and its potential use cases in IoT.

Theoretical Part

These two lab sessions mainly focused on the use of MQTT on an ESP8266. MQTT, which stands for Message Queuing Telemetry Transport, is primarily a lightweight messaging protocol designed for machine-to-machine communication and the Internet of Things. Unlike other protocols like CoAP, which operates on a request-response system, MQTT uses a publish-subscribe system. Each device on the network can subscribe to one or more topics and publish to others.

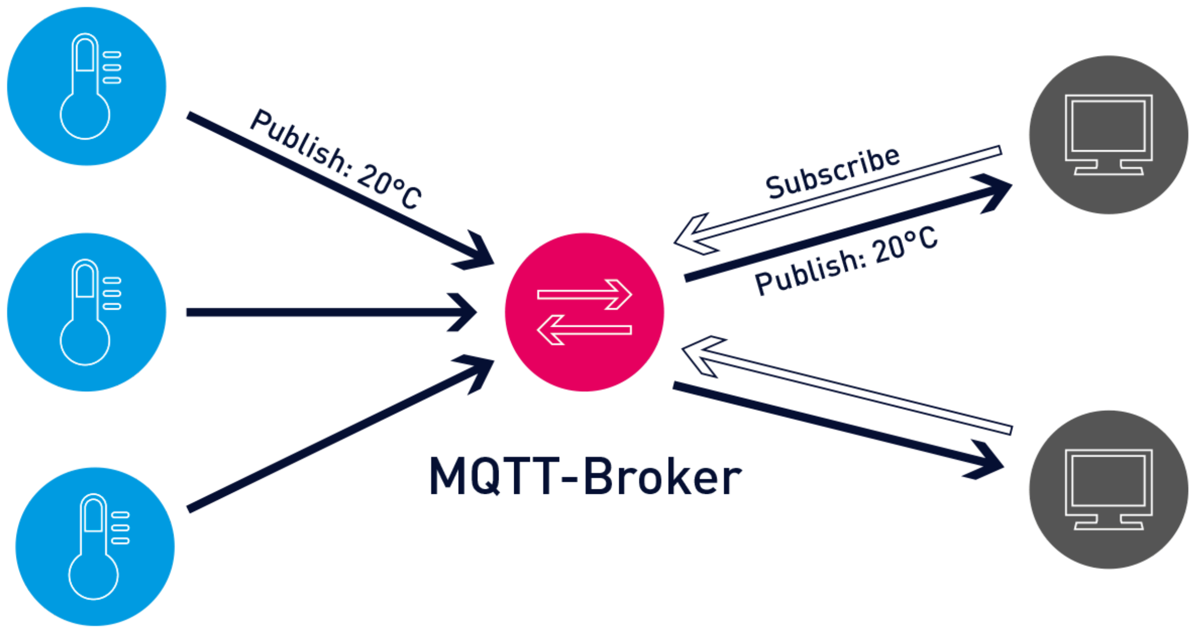


Figure 1. Example of a MQTT architecture

MQTT is based on a broker and client architecture where the broker acts as a server to which all clients communicate. Through the broker, clients can publish their topics and offer the possibility for other clients using the same broker to subscribe to these topics. Additionally, MQTT provides high reliability in data transmission since it is based on TCP/IP. However, this consumes more bandwidth as the connection must remain active during the exchange. To counter this, MQTT remains very lightweight aside from this detail, allowing it to be used on low-bandwidth devices.

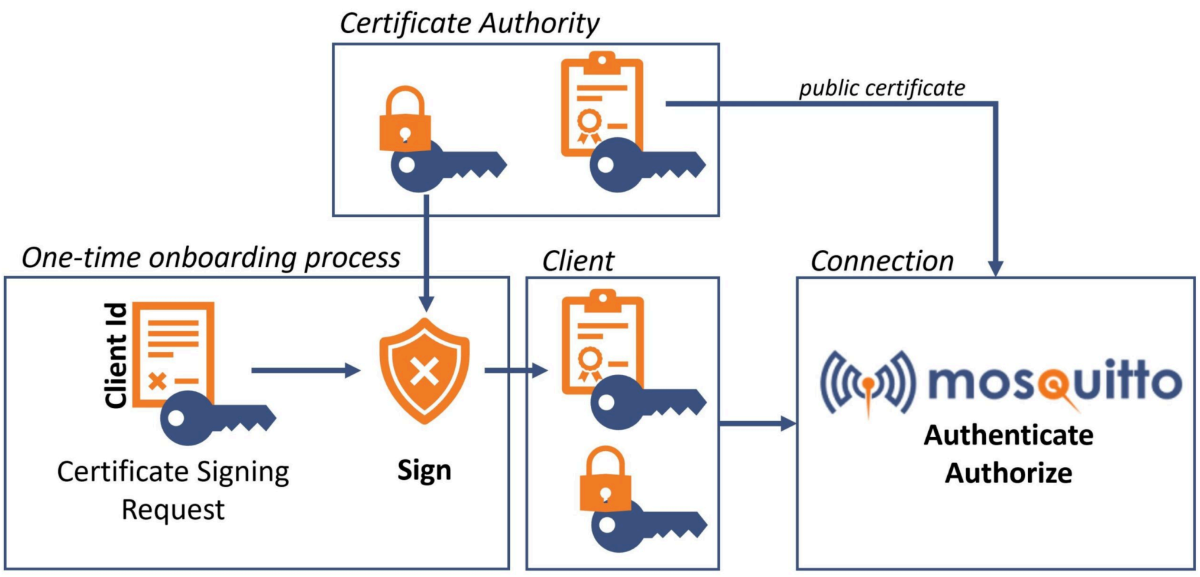


Figure 2. Usage of Mosquitto with authentication

MQTT can also integrate various security systems such as encryption using TLS/SSL to secure communication between clients and the broker. However, this requires clients to communicate regularly via “pings” to keep the connection active, as it could be rejected after a certain time otherwise. Another solution is client authentication via a username and password to subscribe to a topic or publish.

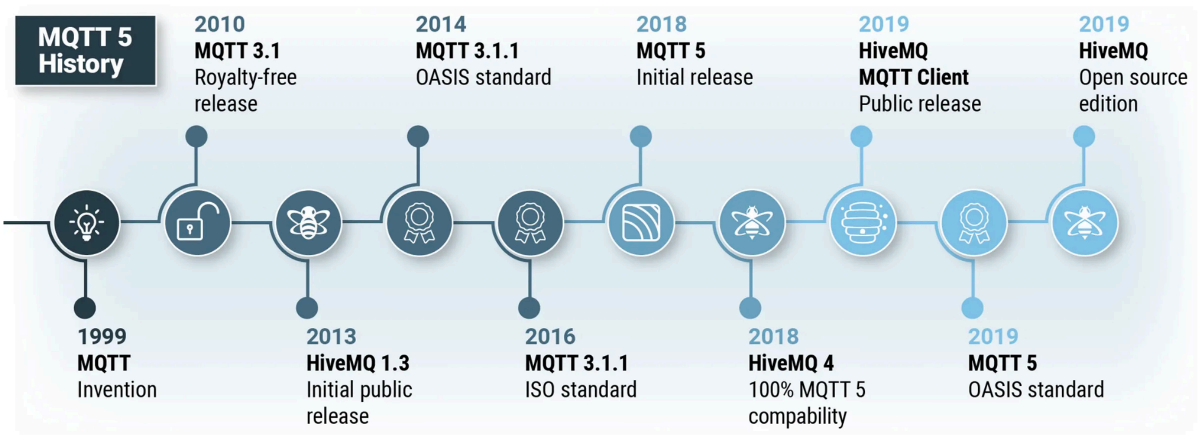

MQTT 5 History
2010 MQTT 3.1 Royalty-free release
2014 MQTT 3.1.1 OASIS standard
2018 MQTT 5 Initial release
2019 HiveMQ MQTT Client Public release
2019 HiveMQ Open source edition
1999 MQTT Invention
2013 HiveMQ 1.3 Initial public release
2016 MQTT 3.1.1 ISO standard
2018 HiveMQ 4 100% MQTT 5 compability
2019 MQTT 5 OASIS standard

Figure 3.Principal versions of MQTT

Finally, MQTT has evolved significantly. Initially created in 1999 by Eurotech and IBM, it will only be Royalty-free in 2010 with version 3.1, OASIS later standardized this version with 3.1.1 in 2014. Today, we use version 5.0, released in 2019.

Noël Jumin
Clément Gauché

But what can MQTT be used for in the end? Let's take a concrete example: I would like the light in my room to turn on when the brightness falls below a certain threshold. However, I still want to be able to manually turn the light on or off via a button. I have at my disposal a Raspberry Pi, four ESPs, an LED, a button, and a light sensor. To address the problem, I will simply need to install my broker on my Raspberry Pi and connect the four ESPs to this broker. Then, here is what each ESP will do:

- ESP_Light: This ESP will be connected to the LED and will subscribe to the light topic, which will be “On” or “Off,” and will turn the LED on or off accordingly.
- ESP_Sensor: This ESP will be connected to the light sensor and will simply publish a percentage of the brightness on the brightness topic.
- ESP_Button: This ESP will be connected to the button and will publish a message “turn on” or “turn off” on the button topic each time it is pressed.
- ESP_Choice: Finally, this last ESP will subscribe to the brightness and button topics to decide whether to publish “On” or “Off” on the light topic accordingly.

Here is a use case based on what we’ve just explained: with a brightness level of 80%, there is no need to turn on the light. However, if the user presses the button, the light will turn on. A possible scenario is that if the user turned off the light a long time ago, it would be useful to detect if the user is in the room or not. If the user is not present, there is no need to turn on the light, but if they are, it would be necessary.

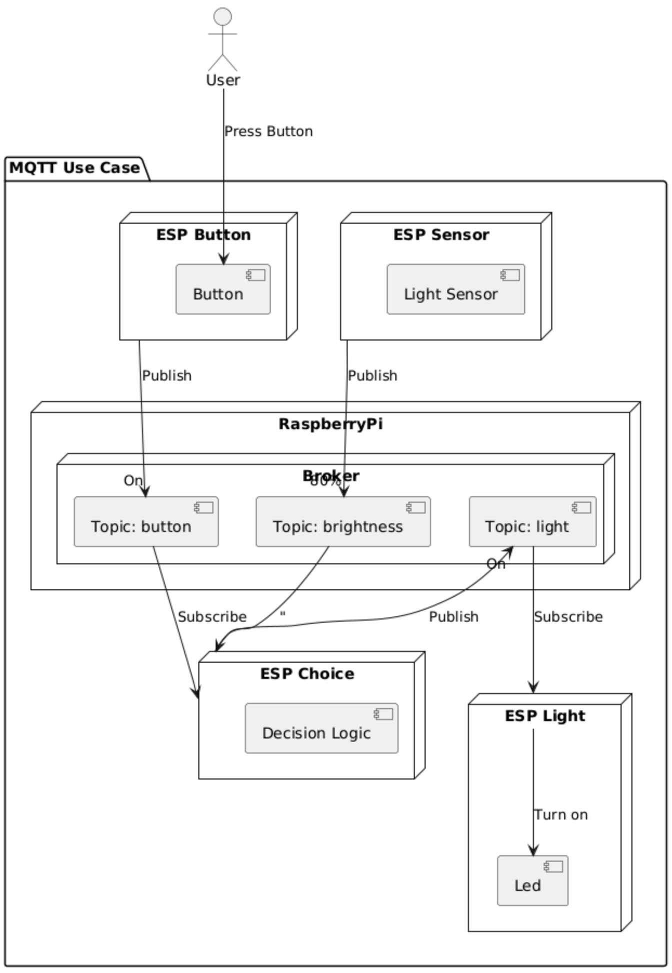


Figure 4. UML representing the Use Case described

In this example, we have assigned one ESP per function, but in reality, it would be possible to combine all functions into a single ESP. However, using MQTT in this case would be unnecessary.

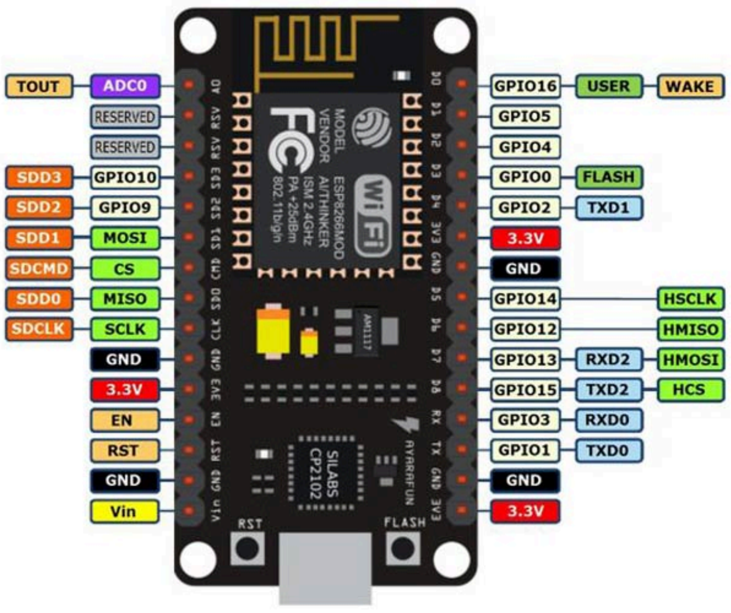


Figure 5. ESP8266 Pinout

During this lab session, we used the ESP8266 development board, which is programmable via USB on any IDE like Arduino, for example. This board can operate up to 160 MHz and can accommodate up to 4 MB of flash memory. Like all development boards, it integrates UART, I2C, SPI communication as well as PWM and an ADC. Its main advantage, which sets it apart from its competitors, is the integration of a Wi-Fi module, whereas an Arduino or STM would need an additional module communicating via I2C or SPI. Finally, its price is very attractive since on sites like Mouser, it can be found for less than 10 euros (less than 5 euros on Chinese websites). Its only real problem is its power management since it consumes a lot and is difficult to program for more advanced power management compared to STM.

Practical part

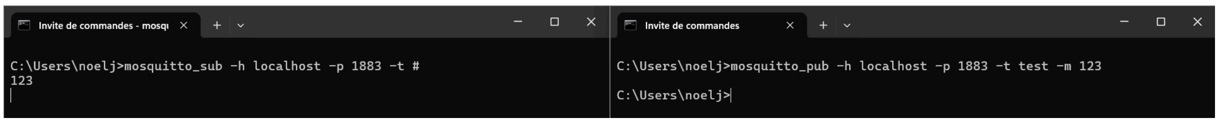
It was the first time we tested MQTT with Mosquitto. To verify its functionality without having to consider potential issues with the ESP boards, we tested directly on our local PCs.

Before launching Mosquitto, you first need to modify its configuration file located at /etc/mosquitto/conf.d/mosquitto.conf and change two lines: the first to allow anyone to access our broker, and the second to define the listening port.

```
allow_anonymous true
listener 1883 0.0.0.0
```

Finally, on Windows, you will need to disable all firewalls. After that, simply start Mosquitto on your PC as a service with net start mosquitto.

Now, the MQTT broker is available at the Wi-Fi IP address of our PC. You can also access it from the same PC via localhost.



```
C:\Users\noelj>mosquitto_sub -h localhost -p 1883 -t #
123
```

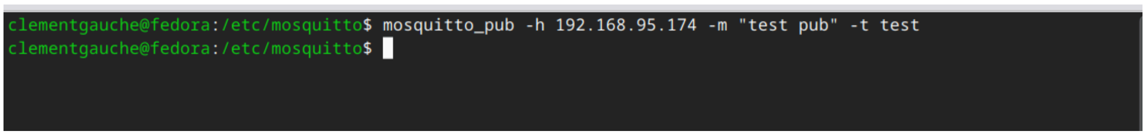
```
C:\Users\noelj>mosquitto_pub -h localhost -p 1883 -t test -m 123

C:\Users\noelj>
```

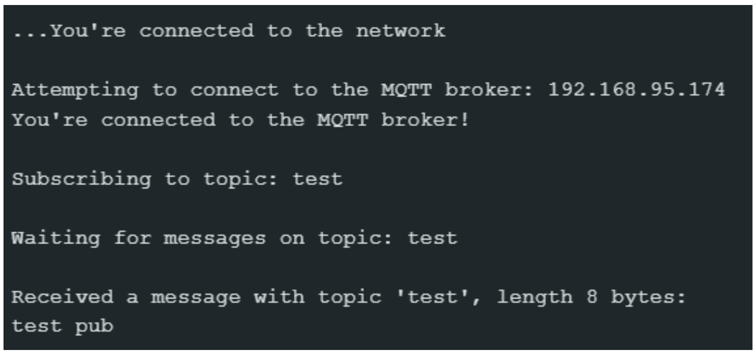
Figure 6. Publication and Subscription test on the PC

On the left, we have a subscriber to all topics, symbolized by “#”, and on the right, we have a publisher on a test topic that sends the data 123. As you can see above, our subscriber successfully receives the message.

Once this is tested, we can now write a small program for our ESP8266 that will continuously listen to the test topic and display the received messages.



```
clementgauche@fedora:/etc/mosquitto$ mosquitto_pub -h 192.168.95.174 -m "test pub" -t test
clementgauche@fedora:/etc/mosquitto$ 
```



```
...You're connected to the network

Attempting to connect to the MQTT broker: 192.168.95.174
You're connected to the MQTT broker!

Subscribing to topic: test

Waiting for messages on topic: test

Received a message with topic 'test', length 8 bytes:
test pub
```

Figure 7 and 8. Publication from the PC and Subscription from the ESP

As we can see, the ESP, connected to the same Wi-Fi and broker as my PC, receives the message sent by my PC. You can also note that this time we did not specify the port when publishing. Mosquitto, if no port is specified, automatically chooses port 1883, which is the one our broker listens on.
To go further, we installed a button on our ESP8266 to allow physical interaction with it. The goal is to turn on an LED on our ESP via this button. When the button is pressed, the LED turns on; otherwise, it turns off. For this, we created a topic “state_led”. The ESP publishes 1 on this topic if the button is pressed, 0 otherwise. The ESP is also subscribed to this same topic and turns off the LED if it sees a 0 and turns it on otherwise.

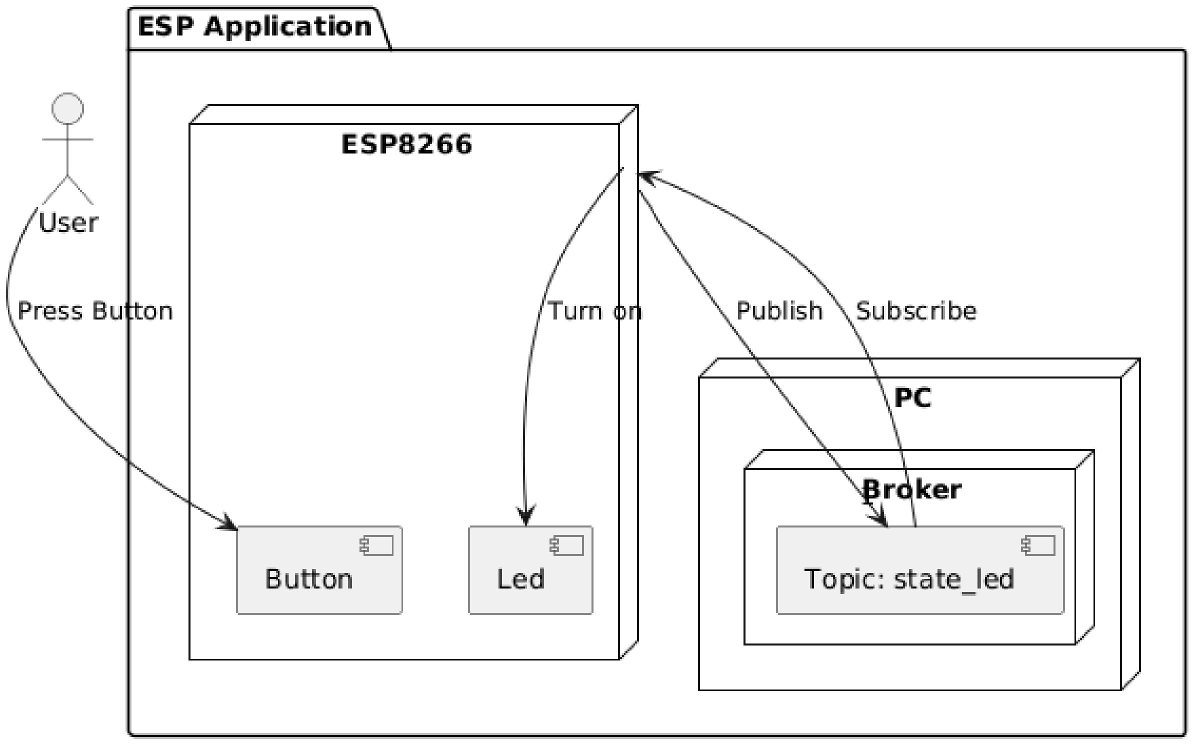


Figure 10. UML representing the solution implemented

Obviously, this approach on a single ESP is a bit silly, but the code can easily be split into two distinct codes running on two separate ESPs connected to the same Wi-Fi network and broker.

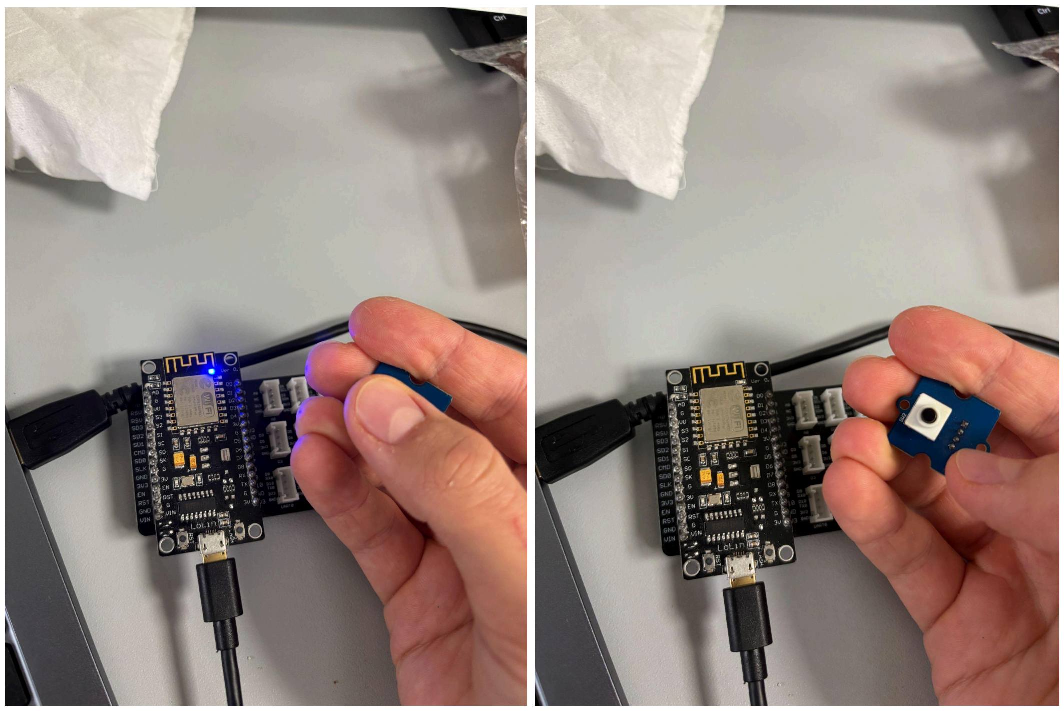
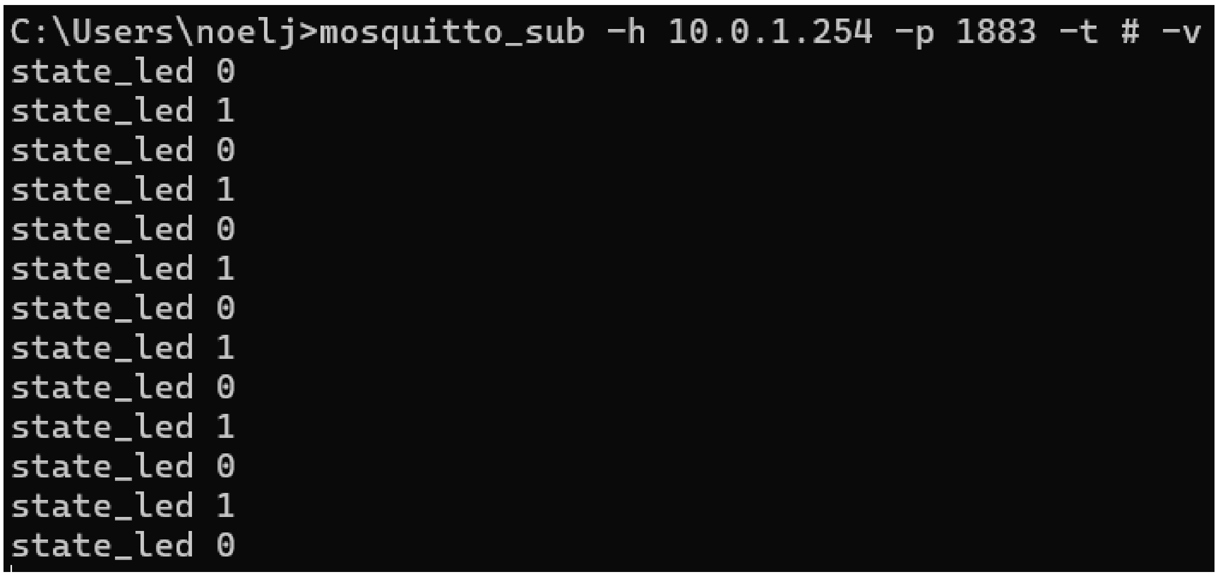


Figure 11. Viewer's point of view

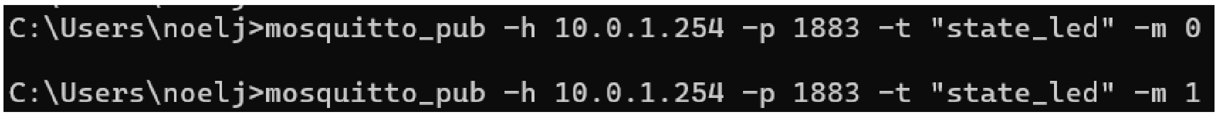
As we can see, this works well, and by listening to the topic, we can see a new message sent only at each change.



```
C:\Users\noelj>mosquitto_sub -h 10.0.1.254 -p 1883 -t # -v
state_led 0
state_led 1
state_led 0
state_led 1
state_led 0
state_led 1
state_led 0
state_led 1
state_led 0
state_led 1
state_led 0
state_led 1
state_led 0
```

Figure 12. Topic subscription from any device

However, a security flaw is quickly found. Indeed, any device connected to the same Wi-Fi and broker can listen to all topics and see the name of our topic as well as the possible values. This poses significant security issues where it is possible to simply turn off the LED or turn it on from another device.



```
C:\Users\noelj>mosquitto_pub -h 10.0.1.254 -p 1883 -t "state_led" -m 0

C:\Users\noelj>mosquitto_pub -h 10.0.1.254 -p 1883 -t "state_led" -m 1
```

Figure 13. Possibility to publish from any device

One solution would be to add authentication between our ESP and the broker.

Noël Jumin
Clément Gauché

Conclusion

In conclusion, our exploration of MQTT with the ESP8266 over two lab sessions has demonstrated the protocol's effectiveness in enabling seamless communication between IoT devices. Through both theoretical understanding and practical implementation, we have seen how MQTT's lightweight nature and publish-subscribe model make it suitable for various IoT applications. The hands-on lab sessions provided valuable insights into setting up and configuring an MQTT broker using Mosquitto, as well as developing a functional application involving an ESP8266, a button, and an LED. Despite some security concerns, which can be mitigated through proper authentication mechanisms, MQTT remains a robust and versatile solution for IoT communication. As IoT continues to evolve, protocols like MQTT will undoubtedly play a pivotal role in shaping the future of connected devices.