INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Middleware Partie 1

by JAULHIAC Paul
VASSEUR Cyril

19/11/2024
Year 2024/2025
5A Innovative Smart Systems

19/11/24

1/Objectives

The goal of this lab is to explore the capabilities of the MQTT protocol for IoT. To do that you will first make a little state of art about the main characteristics of MQTT, then you will install several software on your laptop to manipulate MQTT. Finally, you will develop a simple application with an IoT device (ESP8266) using the MQTT protocol to communicate with a server on your laptop.

2/MQTT

1. What is the typical architecture of an IoT system based on the MQTT protocol?

The typical architecture of an IoT system based on MQTT follows the **publish/subscribe** model and consists of:

- **Devices (Clients):** Sensors, actuators, or user interfaces that publish data or subscribe to topics.
- **MQTT Broker:** A central server that manages message distribution, decoupling publishers from subscribers.
- **Network Infrastructure:** Uses TCP/IP to enable reliable communication between devices and the broker.

2. What is the IP protocol under MQTT? What does it mean in terms of bandwidth usage, type of communication, etc.?

MQTT operates over **TCP/IP**, which ensures reliable, ordered, and error-checked delivery of messages.

- **Bandwidth Usage:** MQTT is lightweight, with minimal headers, optimizing bandwidth even with the overhead of TCP.
- **Type of Communication:** Reliable and stateful communication ensures consistency, suitable for constrained networks and devices.

3. What are the different versions of MQTT?

- **MQTT 3.1:** The original version by Eurotech and IBM.
- **MQTT 3.1.1:** Standardized by OASIS, offering minor improvements over 3.1.
- **MQTT 5.0:** Introduced advanced features like reason codes, shared subscriptions, message expiry, and topic aliases to enhance flexibility and efficiency.

4. What kind of security/authentication/encryption are used in MQTT?

MQTT security relies on external measures:

- **Encryption:** Transport Layer Security (TLS) ensures data confidentiality and integrity.
- **Authentication:** Uses username/password pairs or client certificates to verify identities.
- **Authorization:** Access controls to limit topic permissions.
- **Network Security:** Firewalls and VPNs protect the MQTT infrastructure from unauthorized access.

5. Suppose you have devices that include one button, one light, and a luminosity sensor. You would like to create a smart system for your house with the following behavior:

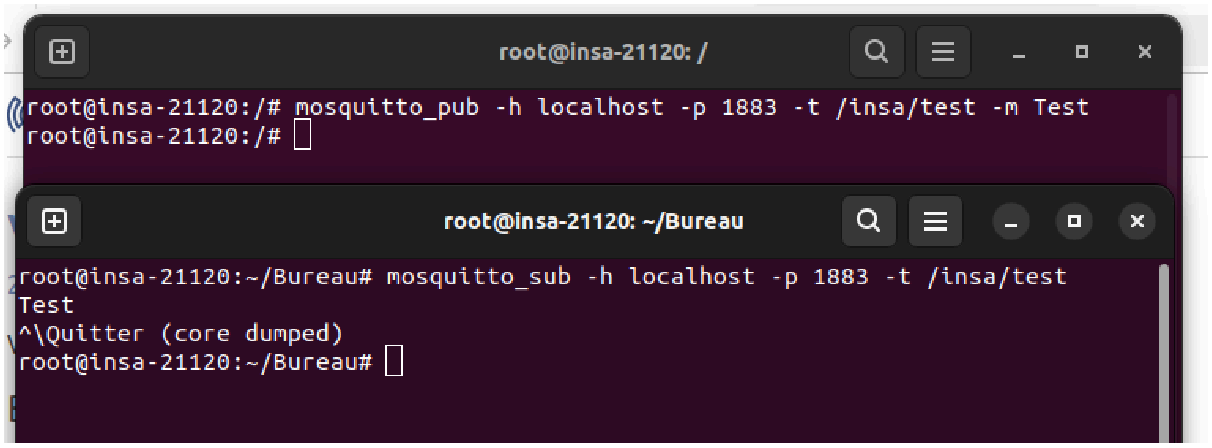
- **Switch on the light manually with the button.**
- **Automatically switch on the light when luminosity is under a certain value.**

Necessary Topics and Connections:

1. **Topics:**
   - `home/light/control`: Commands to the light (e.g., "ON" or "OFF").
   - `home/light/status`: Current status of the light.
   - `home/sensor/luminosity`: Luminosity readings from the sensor.
   - `home/button/press`: Button press events.
2. **Device Roles and Connections:**
   - **Button:**
     - **Publish:** Sends "press" messages to `home/button/press`.
   - **Luminosity Sensor:**
     - **Publish:** Sends periodic readings to `home/sensor/luminosity`.
   - **Light:**
     - **Subscribe:** Listens to `home/light/control` for on/off commands.
     - **Publish:** Updates `home/light/status` when its state changes.
   - **Controller (Logic):**
     - **Subscribe:**
       - `home/button/press` to detect manual activation.
       - `home/sensor/luminosity` to monitor ambient light.
     - **Publish:** Sends "ON" or "OFF" commands to `home/light/control` based on logic:
       - Turns on the light if the button is pressed.
       - Turns on the light if luminosity is below a threshold.
       - Turns off the light if luminosity exceeds the threshold and it was automatically turned on.

This architecture achieves both manual and automatic control seamlessly.

3/Install and test the broker



```
root@insa-21120:/# mosquitto_pub -h localhost -p 1883 -t /insa/test -m Test
root@insa-21120:/# 
```

```
root@insa-21120:~/Bureau# mosquitto_sub -h localhost -p 1883 -t /insa/test
Test
^\Quitter (core dumped)
root@insa-21120:~/Bureau# 
```

As you can see above, we managed to start a subscription on the terminal below, listening for topic “/insa/test”, on port 1883 of the host. From the terminal above, we published a message “Test” on the same port, and we managed to receive it on the subscriber’s terminal.

4/Creation of an IoT device with the nodeMCU board that uses MQTT communication

a.) **Main Characteristics of the NodeMCU Board (Based on ESP8266)**

1. Communication Capabilities

- **Wi-Fi Module (ESP8266)**:
    - Integrated Wi-Fi for 802.11 b/g/n standards.
    - Can function as both a client (STA) and an access point (AP).
    - Supports TCP, UDP, HTTP, MQTT, and other protocols for IoT communication.
    - Ideal for MQTT-based IoT systems due to lightweight and efficient communication.
- **Serial Communication**:
    - Equipped with a UART interface for debugging and data transfer.
- **GPIO (General-Purpose Input/Output)**:
    - Can be configured for digital communication (e.g., I2C, SPI).

2. Programming Language

- **Lua Scripting**:
  - Default firmware supports Lua, a lightweight scripting language.
  - Simplifies programming for IoT applications.
- **Arduino IDE**:
  - Fully compatible with Arduino IDE, enabling C/C++ programming.
  - Large community support with a rich set of libraries for IoT, sensors, and actuators.
- **MicroPython**:
  - Supports Python-based scripting for easy prototyping.
- **AT Commands**:
  - Can be programmed with AT command sets for low-level control.

3. Input/Output Capabilities

- **GPIO Pins**:
  - Provides up to 11 GPIO pins.
  - Configurable for input or output.
  - Support PWM, I2C, SPI, and UART protocols.
- **Analog Input (ADC)**:
  - Single ADC pin with a resolution of 10 bits (0–1023).
  - Used for reading analog sensors (e.g., luminosity sensors).
- **PWM Output**:
  - Can drive actuators like LEDs and motors.
- **I2C and SPI Interfaces**:
  - Communicates with external peripherals like sensors, displays, and other modules.
- **Power Supply and Consumption**:
  - Operates at 3.3V.
  - Equipped with a built-in voltage regulator for 5V USB power input.

4. Memory and Processing

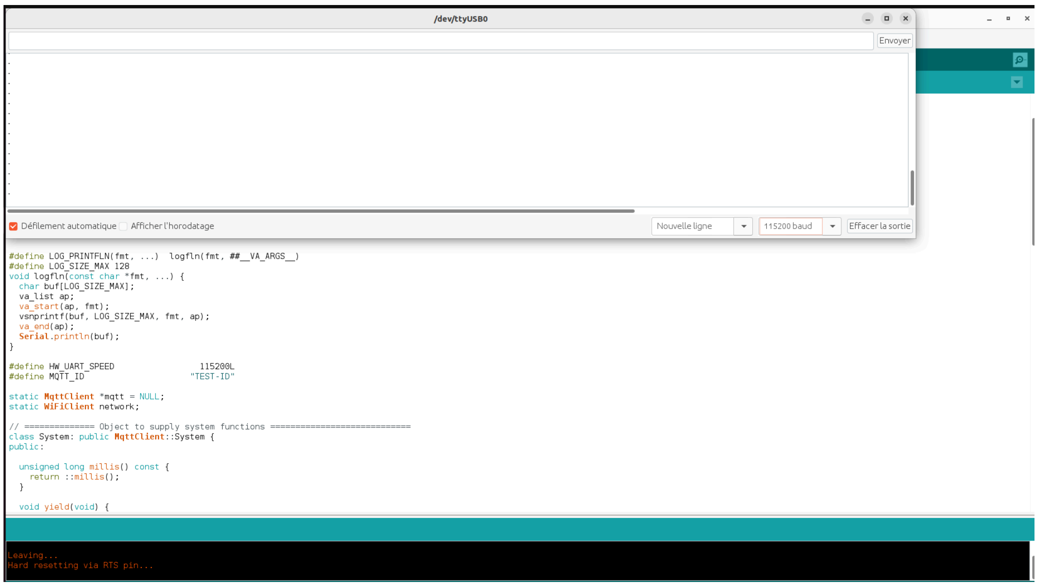
- **Flash Memory**:
  - Typically 4MB (depending on the version).
- **RAM**:
  - 50 KB of usable RAM for runtime applications.
- **Processing Power**:
  - Runs on a 32-bit Tensilica L106 processor at 80 MHz (configurable up to 160 MHz).

Advantages for IoT Applications

- Compact and cost-effective for IoT projects.
- Integrated Wi-Fi eliminates the need for external modules.
- Large support community ensures robust troubleshooting and resource availability.

This board is particularly suited for lightweight IoT protocols like MQTT and applications involving simple automation, sensor data collection, and control systems.

Application :
e.)
At first, the program opens the serial port, then initializes the WiFi network and sets up the MQTT client.
Following this, in the loop of the program, he tests whether a connection has been established or not. If there is, it is closed.
Then a new MQTT client connection is started by setting up the client structure (MqttClient::ConnectResult connectResult;).
The subscribing and publishing part of the code are the last two parts before the end of the loop.

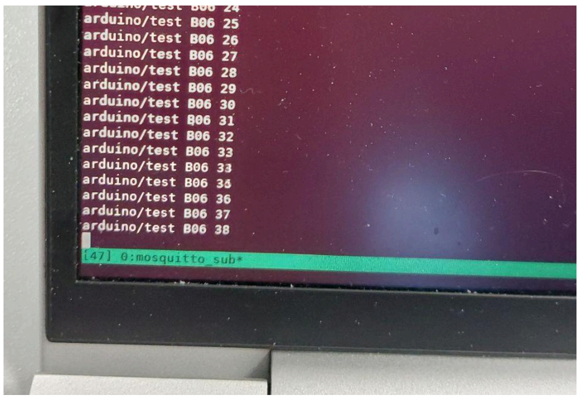


```
.
.
.
.
.
.
.
.
.
.
.
.
.
.
```

```
#define LOG_PRINTFLN(fmt, ...)  logfln(fmt, ##__VA_ARGS__)
#define LOG_SIZE_MAX 128
void logfln(const char *fmt, ...) {
  char buf[LOG_SIZE_MAX];
  va_list ap;
  va_start(ap, fmt);
  vsnprintf(buf, LOG_SIZE_MAX, fmt, ap);
  va_end(ap);
  Serial.println(buf);
}

#define HW_UART_SPEED                    115200L
#define MQTT_ID                        "TEST-ID"

static MqttClient *mqtt = NULL;
static WiFiClient network;

// ============= Object to supply system functions ============================
class System: public MqttClient::System {
public:

  unsigned long millis() const {
    return ::millis();
  }

  void yield(void) {
```

```
Leaving...
Hard resetting via RTS pin...
```

f.) As we can see below, the broker is indeed receiving our “BO6” message.



```
arduino/test B06 24
arduino/test B06 25
arduino/test B06 26
arduino/test B06 27
arduino/test B06 28
arduino/test B06 29
arduino/test B06 30
arduino/test B06 31
arduino/test B06 32
arduino/test B06 33
arduino/test B06 33
arduino/test B06 35
arduino/test B06 36
arduino/test B06 37
arduino/test B06 38

[47] 0:mosquitto_sub*
```

```
    Serial.print("Sending message to topic: ");
    Serial.println(topic);
    Serial.print("B06 ");
    Serial.println(count);

    // send message, the Print interface can be used to set the message contents
    mqttClient.beginMessage(topic);
    mqttClient.print("B06 ");
    mqttClient.print(count);
    mqttClient.endMessage();
```

```
Sending message to topic: arduino/test
B06 36

Sending message to topic: arduino/test
B06 37

Sending message to topic: arduino/test
B06 38

Sending message to topic: arduino/test
B06 39
```

5/ Creation of a simple application

To do the application we first connected a button and a light on the arduino with the line code :

```
  // Config pins motherboard
  pinMode(D5,INPUT_PULLUP);
  pinMode(D4, OUTPUT);     // Initialize the LED pin as an output
```

then we have to publish the button state with :

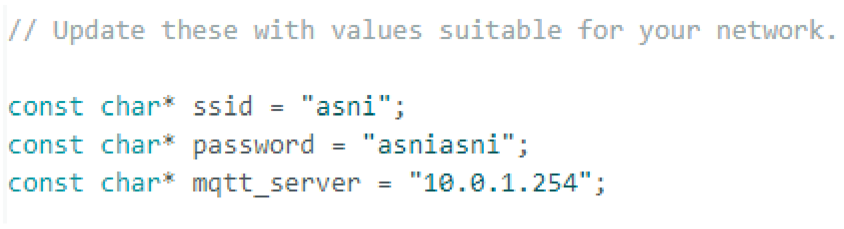
```
client.publish("ry/light", &StateButton);
```

after that we just have to subscribe on the same topic as the button state to listen when the button is activated or not with our function callback :

```
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();

  // Switch on the LED if an 1 was received as first character
  if ((char)payload[0] == '1') {
    digitalWrite(D4, HIGH);   // Turn the LED on (Note that LOW is the voltage level
    // but actually the LED is on; this is because
    // it is active low on the ESP-01)
  } else {
    digitalWrite(D4, LOW);  // Turn the LED off by making the voltage HIGH
  }

}
```

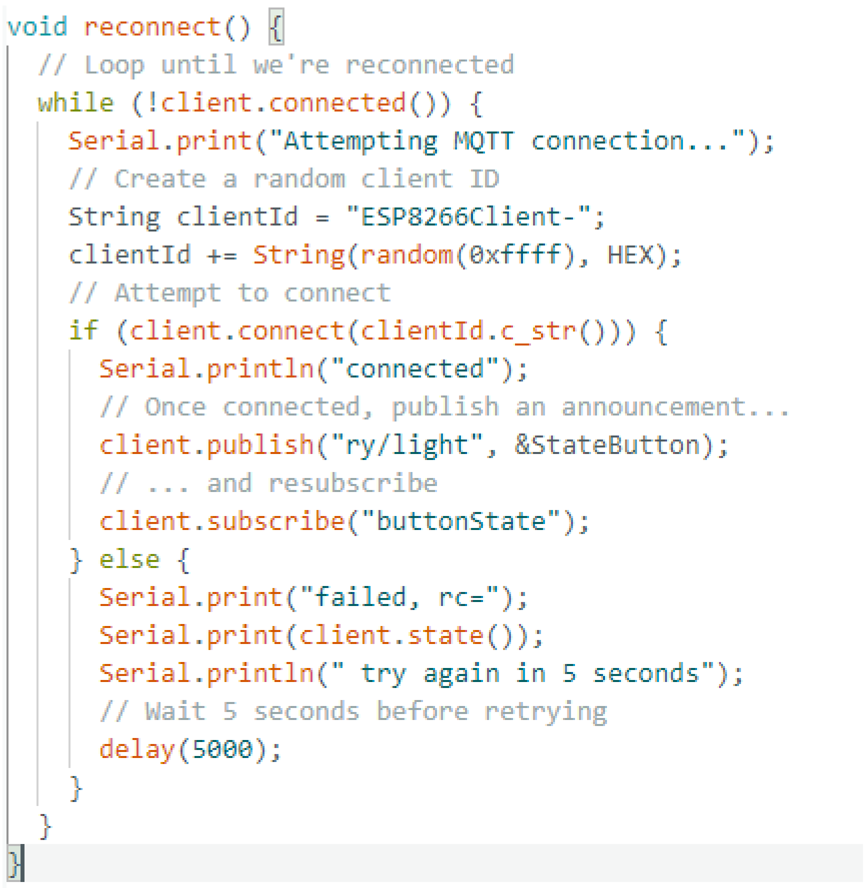
This application showed us how to connect and send data over a MQTT service.

6/ Creation of a complex application



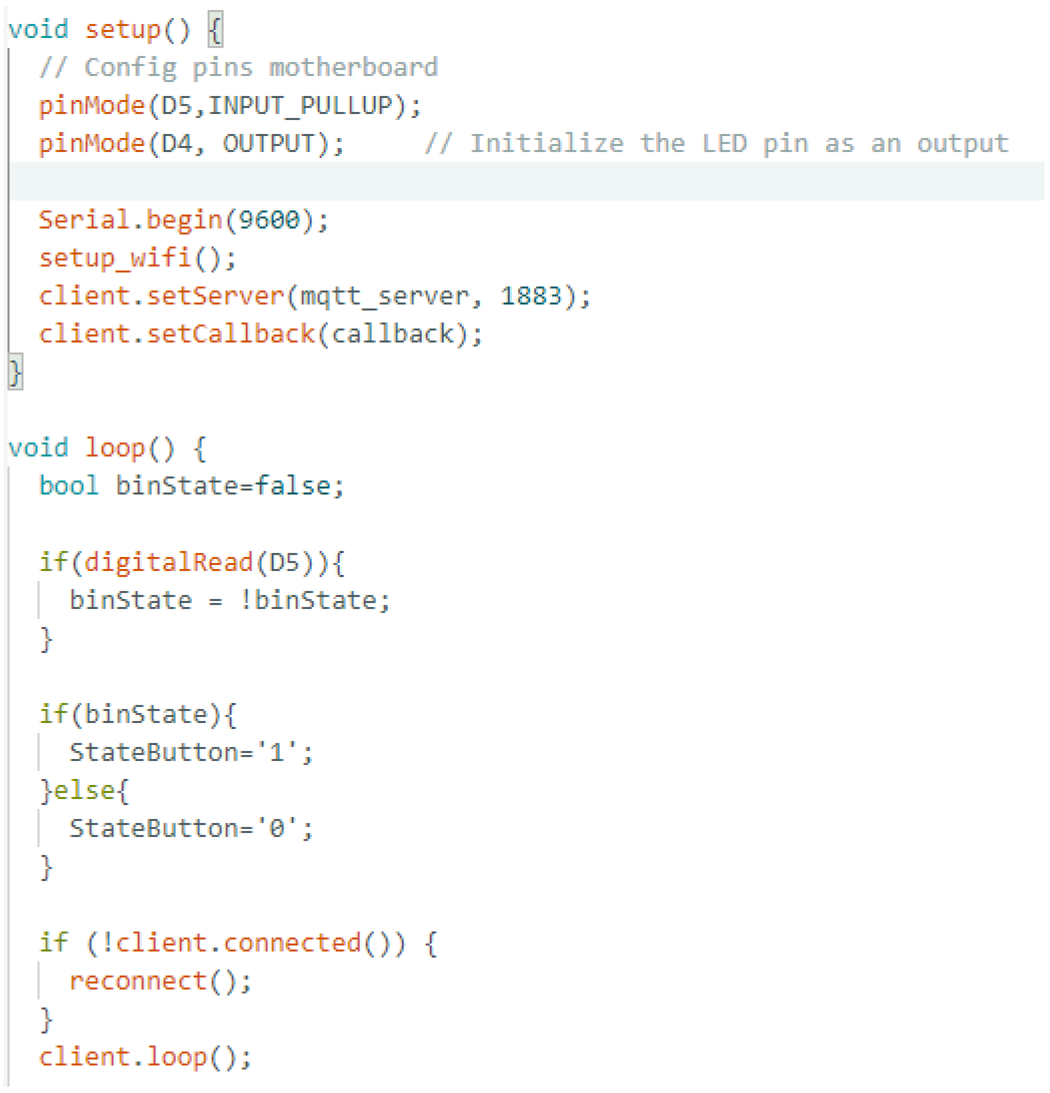
```
// Update these with values suitable for your network.

const char* ssid = "asni";
const char* password = "asniasni";
const char* mqtt_server = "10.0.1.254";
```

Network configuration with IP address of the server



```
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "ESP8266Client-";
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
      // Once connected, publish an announcement...
      client.publish("ry/light", &StateButton);
      // ... and resubscribe
      client.subscribe("buttonState");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}
```

Reconnect function, with publishing of the button state



```
void setup() {
  // Config pins motherboard
  pinMode(D5,INPUT_PULLUP);
  pinMode(D4, OUTPUT);     // Initialize the LED pin as an output

  Serial.begin(9600);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}

void loop() {
  bool binState=false;

  if(digitalRead(D5)){
    binState = !binState;
  }

  if(binState){
    StateButton='1';
  }else{
    StateButton='0';
  }

  if (!client.connected()) {
    reconnect();
  }
  client.loop();
```

PIN configuration during setup, plus reading of the button state and publishing with reconnect function.

To create this application, we started from the previous simple application, except that this time, we were publishing to the topic “ry/light”, from the group of students next to us, on the MQTT server. Both groups were connected to the “asni” network, of course. We were sending the state of our button, so they could listen to it and turn their LED on or off, according to it.

7/ Conclusion

This project allowed us to explore the MQTT protocol, its publish/subscribe architecture, and its application in IoT systems. After studying its theoretical aspects, including its lightweight design and TLS-based security, we implemented practical solutions by setting up a broker and developing applications using the NodeMCU board. From controlling an LED to collaborating with other groups, these experiments showcased MQTT's efficiency and simplicity for building connected systems.

The full code and implementation details are available on the project's GitHub repository: https://github.com/Cyril-vsr/Middleware_of_IOT/tree/Dev_Windows_Paul