

I5SSIL11 - Middleware For the IoT

Tom Lassalle | Florant Miranville

Louis Rousset

Table des matières

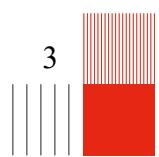
1.	Introduction.....	3
1.1.	Contexte général.....	3
2.	TP 1/2 : Middleware For the IoT	4
2.1.	Objectif de ce TP	4
2.2.	Objectif de ce TP	4
2.3	Creation of an IoT device with the nodeMCU board that uses MQTT communication	
	6	
2.3	Creation of a simple application.....	10
2.4	Creation of a complex application.....	10
3.	TP 3 : Fast application prototyping for IoT	13
3.1.	Mise en place du dispositif IoT	13
3.2.	Développement de l'application avec Node-RED.....	15
4.	TP 4 : Middleware spécialisé	16

1. Introduction

1.1. Contexte général

Les technologies de l'Internet des objets (IoT) sont omniprésentes dans notre quotidien, avec de nombreuses applications, notamment dans le domaine de la domotique. Par exemple, les ampoules connectées (comme Philips Hue ou LIFX) permettent de contrôler l'intensité lumineuse, de s'adapter automatiquement à la luminosité ambiante ou encore à la présence des occupants dans une pièce.

Dans ce contexte, le protocole MQTT se distingue pour sa capacité à gérer efficacement la communication entre ces dispositifs. Utilisé fréquemment dans des environnements à faible bande passante, il s'avère idéal pour les systèmes IoT, où la réactivité et la simplicité d'échange sont possibles.



2. TP 1/2 : Middleware For the IoT

2.1. Objectif de ce TP

Dans le cadre de ce TP, nous allons explorer les capacités du protocole MQTT pour la communication entre objets dans un système IoT. MQTT (Message Queuing Telemetry Transport) est un protocole léger, orienté message, conçu pour des réseaux à faible bande passante et pour des dispositifs avec des ressources limitées, comme ceux utilisés dans l'IoT.

L'objectif principal de ce TP est de comprendre comment MQTT peut être utilisé pour interconnecter des dispositifs IoT nous utiliserons pour ce faire des cartes NodeMCU/ESP8266 et un serveur central via un broker MQTT.

Au cours de ce TP nous allons dans un premier temps analyser les caractéristiques de MQTT, ses versions, son fonctionnement, ainsi que les questions de sécurité et d'authentification. Puis nous mettrons en place un broker MQTT via le logiciel Mosquitto pour tester les mécanismes de publish/subscribe (publication/abonnement). Et enfin nous développerons une application IoT simple sur la carte NodeMCU (ESP8266) pour simuler la gestion d'un système d'éclairage intelligent contrôlé par un capteur de luminosité et un bouton.

2.2. Objectif de ce TP

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

L'architecture générale d'un système IoT basé sur le protocole MQTT est le modèle Broker/client

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

Le protocole MQTT est basé sur TCP/IP. Cela signifie que les communications sont asynchrones, la bande passante est aménagée pour les capteurs IoT, et la garantie de l'envoi du message, le contrôle de congestion sont offertes par ce protocole TCP/IP

3. What are the different versions of MQTT?

Le protocole MQTT et sa spécification sont disponibles depuis la fin des années 1990. On retrouve 3 principales versions MQTT :

- 3.1 qui introduit notamment le modèle du pub/sub
- 3.1.1 qui améliore la gestion des erreurs de connexion et des déconnexions.
- 5.0 qui ajoute des principes comme : la propriétés utilisateur ; le partage de souscriptions et les alias de topic

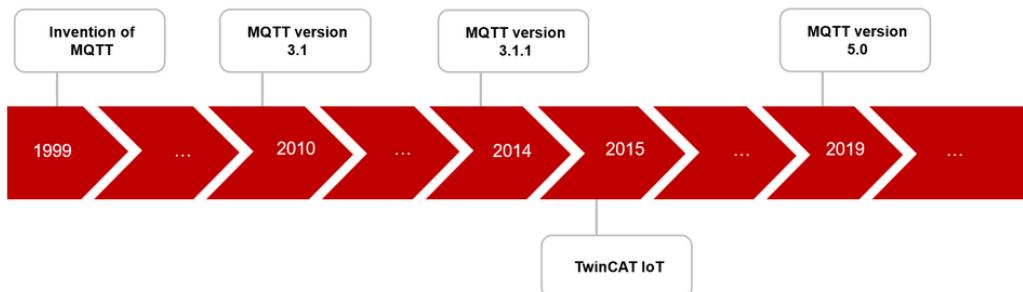


Figure 1 : Protocol versions

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

Nativement, MQTT n'inclut pas de solution de sécurité, il est possible de le coupler avec des solutions extérieures comme TLS (Transport Layer Security) qui va assurer le chiffrement des messages entre les clients MQTT et le broker, l'intégrité de ces messages, et l'authentification pour que les clients MQTT et le broker peuvent se reconnaître entre eux.

Voir : <https://www.emqx.com/en/blog/fortifying-mqtt-communication-security-with-ssl-tls>

5. Suppose you have devices that include one button, one light and luminosity sensor.

You would like to create a smart system for your house with this behavior:

- You would like to be able to switch on the light manually with the button
- The light is automatically switched on when the luminosity is under a certain value

6. What different topics will be necessary to get this behavior and what will the connection be in terms of publishing or subscribing?

On repère les éléments et actions de la mise en situation :

- Un bouton, une lumière, et un capteur de luminosité.
- Switch manuel et l'état de la lampe (On/Off)

Nous avons les publications suivantes :

Switch : publication de On ou Off quand le switch est activé (bouton appuyé)

Valeur de luminosité : publication de la valeur de la luminosité

Etat de la lumière : publication de l'état actuel de la lampe (On ou Off)

Nous avons les relations suivantes avec les clients :

1. **Le capteur** publie la valeur de luminosité
2. **Le bouton** publie l'état après activation du switch
3. **Le système interne** s'abonne à la valeur de luminosité et à l'état actuel de la lampe et publie l'état après switch

Ainsi, quand la valeur de luminosité est sous un certain seuil, la lumière est switch.

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

Pour ce travail pratique, nous avons choisi de mettre en place notre propre broker Mosquitto, hébergé sur notre PC et accessible via notre réseau mobile en configurant en téléphone en modem Wifi.

1. Give the main characteristics of nodeMCU board in term of communication, programming language, Inputs/outputs capabilities

Le nodeMCU (ou encore ESP8266) offre du WIFI 802.11b/g/n avec une antenne intégrée, et supporte le WPA/WPA2. En termes de communication, le nodeMCU permet UART, SDIO, SPI, I2C, I2S, et il possède une télécommande IR. Concernant les langages de programmation, l'ESP8266 est basé sur le langage Lua et il supporte également le C/C++ via Arduino et il est possible d'utiliser MicroPython également.

Nous utilisons le code exemple suivant, disponible à partir de PubSubClient :

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

// Update these with values suitable for your network.

const char* ssid = "Pctom";
const char* password = "12345678";
const char* mqtt_server = "10.178.100.251";

WiFiClient espClient;
PubSubClient client(espClient);
unsigned long lastMsg = 0;
#define MSG_BUFFER_SIZE (50)
char msg[MSG_BUFFER_SIZE];
int value = 0;
```

```

void setup_wifi() {

    delay(10);
    // We start by connecting to a WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    randomSeed(micros());

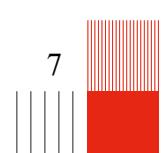
    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
}

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(BUILTIN_LED, LOW);      // 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(BUILTIN_LED, HIGH);     // Turn the LED off by making the
voltage HIGH
    }
}

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("outTopic", "hello world");
        // ... and resubscribe
        client.subscribe("inTopic");
    } else {
        Serial.print("failed, rc=");
        Serial.print(client.state());
        Serial.println(" try again in 5 seconds");
        // Wait 5 seconds before retrying
        delay(5000);
    }
}

void setup() {
    pinMode(BUILTIN_LED, OUTPUT);          // Initialize the BUILTIN_LED pin
as an output
    Serial.begin(115200);
    setup_wifi();
    client.setServer(mqtt_server, 1883);
    client.setCallback(callback);
}

void loop() {

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

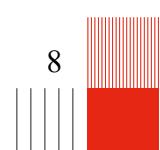
    unsigned long now = millis();
    if (now - lastMsg > 2000) {
        lastMsg = now;
        ++value;
        snprintf (msg, MSG_BUFFER_SIZE, "hello world #%ld", value);
        Serial.print("Publish message: ");
        Serial.println(msg);
        client.publish("outTopic", msg);
    }
}

```

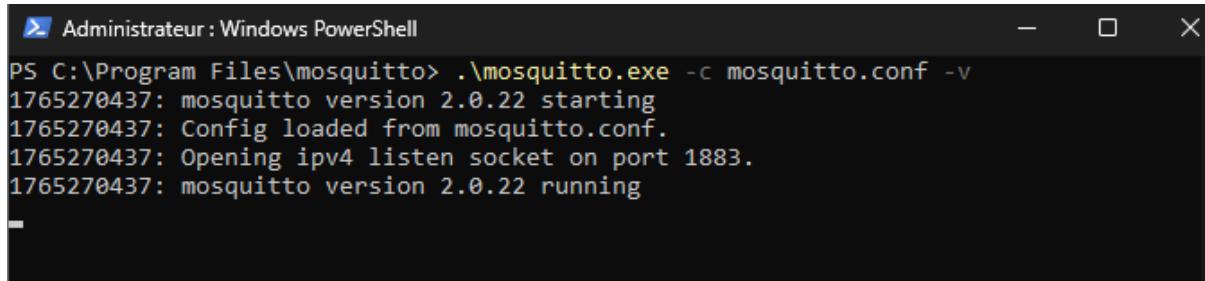
Ce code permet à un ESP8266 de se connecter à un réseau Wi-Fi et de communiquer via le protocole MQTT. Il se connecte à un serveur MQTT à l'adresse spécifiée, publie un message toutes les 2 secondes sur le sujet outTopic, et s'abonne au sujet inTopic pour recevoir des messages. Si le message reçu est "1", il allume la LED intégrée, sinon il l'éteint. Si la connexion MQTT est perdue, le code tente de se reconnecter automatiquement.

Une fois la configuration du serveur mosquitto édité nous pouvons lancer via la commande suivante :

`./mosquitto.exe -c mosquitto.conf -v`



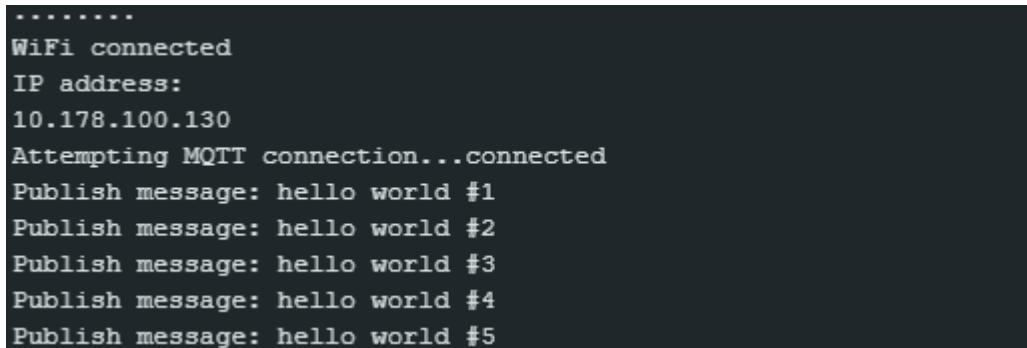
Afin de lancer mosquitto avec la configuration créée et en mode verbose pour avoir les différentes informations.



```
PS C:\Program Files\mosquitto> ./mosquitto.exe -c mosquitto.conf -v
1765270437: mosquitto version 2.0.22 starting
1765270437: Config loaded from mosquitto.conf.
1765270437: Opening ipv4 listen socket on port 1883.
1765270437: mosquitto version 2.0.22 running
```

Figure 2 : Mosquitto lancement

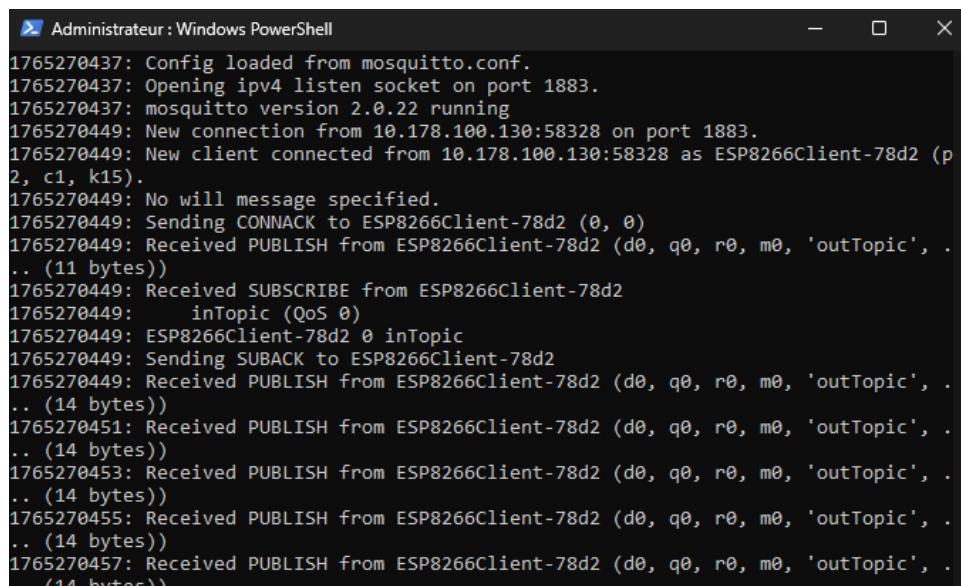
Une fois mosquitto lancé nous pouvons nous connecter via le protocole MQTT, avec l'esp82.



```
.....
WiFi connected
IP address:
10.178.100.130
Attempting MQTT connection...connected
Publish message: hello world #1
Publish message: hello world #2
Publish message: hello world #3
Publish message: hello world #4
Publish message: hello world #5
```

Figure 3 : ESP82 MQTT

Le code par défaut envoi simplement hello world sur le topic outTopic.



```
1765270437: Config loaded from mosquitto.conf.
1765270437: Opening ipv4 listen socket on port 1883.
1765270437: mosquitto version 2.0.22 running
1765270449: New connection from 10.178.100.130:58328 on port 1883.
1765270449: New client connected from 10.178.100.130:58328 as ESP8266Client-78d2 (p
2, c1, k15).
1765270449: No will message specified.
1765270449: Sending CONNACK to ESP8266Client-78d2 (0, 0)
1765270449: Received PUBLISH from ESP8266Client-78d2 (d0, q0, r0, m0, 'outTopic', .
. . (11 bytes))
1765270449: Received SUBSCRIBE from ESP8266Client-78d2
1765270449:     inTopic (QoS 0)
1765270449: ESP8266Client-78d2 0 inTopic
1765270449: Sending SUBACK to ESP8266Client-78d2
1765270449: Received PUBLISH from ESP8266Client-78d2 (d0, q0, r0, m0, 'outTopic', .
. . (14 bytes))
1765270451: Received PUBLISH from ESP8266Client-78d2 (d0, q0, r0, m0, 'outTopic', .
. . (14 bytes))
1765270453: Received PUBLISH from ESP8266Client-78d2 (d0, q0, r0, m0, 'outTopic', .
. . (14 bytes))
1765270455: Received PUBLISH from ESP8266Client-78d2 (d0, q0, r0, m0, 'outTopic', .
. . (14 bytes))
1765270457: Received PUBLISH from ESP8266Client-78d2 (d0, q0, r0, m0, 'outTopic', .
. . (14 bytes))
```

Figure 4 : Mosquitto en mode verbose

Avec mosquitto en mode verbose on remarque que l'esp82 établie bien une connexion avec mosquitto.

```
1765270520: No will message specified.
1765270520: Sending CONNACK to auto-26FCC00D-3AE0-AB8A-8E04-891318C263FD (0, 0)
1765270520: Received SUBSCRIBE from auto-26FCC00D-3AE0-AB8A-8E04-891318C263FD
1765270520:     outTopic (QoS 0)
1765270520: auto-26FCC00D-3AE0-AB8A-8E04-891318C263FD 0 outTopic
1765270520: Sending SUBACK to auto-26FCC00D-3AE0-AB8A-8E04-891318C263FD
1765270521: Received PUBLISH from ESP8266Client-78d2 (d0, q0, r0, m0, 'outTopic', .
.. (15 bytes))
1765270521: Sending PUBLISH to auto-26FCC00D-3AE0-AB8A-8E04-891318C263FD (d0, q0, r
0, m0, 'outTopic', ... (15 bytes))
1765270523: Received PUBLISH from ESP8266Client-78d2 (d0, q0, r0, m0, 'outTopic', .
.. (15 bytes))
1765270523: Sending PUBLISH to auto-26FCC00D-3AE0-AB8A-8E04-891318C263FD (d0, q0, r
0, m0, 'outTopic', ... (15 bytes))
1765270524: Received PINGREQ from ESP8266Client-78d2
1765270524: Sending PINGRESP to ESP8266Client-78d2
1765270525: Received PUBLISH from ESP8266Client-78d2 (d0, q0, r0, m0, 'outTopic', .
.. (15 bytes))
1765270525: Sending PUBLISH to auto-26FCC00D-3AE0-AB8A-8E04-891318C263FD (d0, q0, r
0, m0, 'outTopic', ... (15 bytes))
-
```

Figure 5 : Mosquitto – v

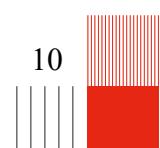
Nous pouvons ainsi écouter sur le topic : outTopic :

```
PS C:\Program Files\mosquitto> .\mosquitto_sub.exe -t outTopic
hello world #37
hello world #38
hello world #39
hello world #40
hello world #41
hello world #42
hello world #43
hello world #44
hello world #45
hello world #46
hello world #47
hello world #48
hello world #49
```

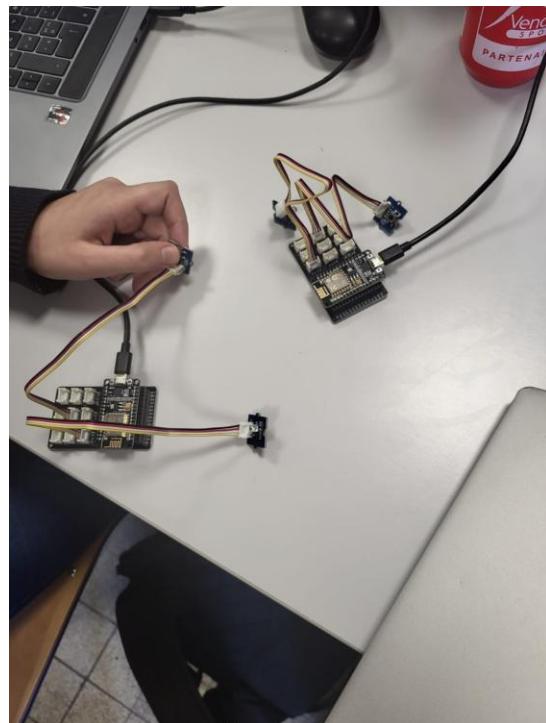
Nous recevons bien les « hello world envoyé » par l'ESP82.

2.3 Creation of a simple application

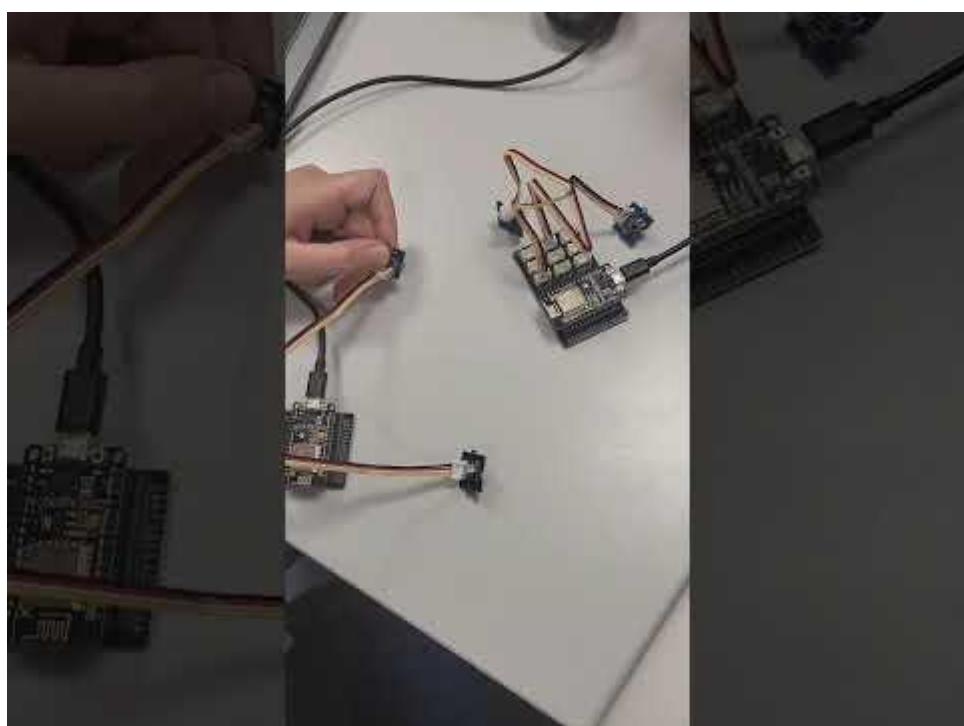
2.4 Creation of a complex application



Nous avons développé une application collaborative en collaboration avec le groupe de TP de Julie et Amalia. Dans cette application, lorsque l'une d'entre elles appuie sur le bouton de leur ESP82, la LED de notre propre ESP82 s'allume en réponse. Ce système permet ainsi une interaction entre les différents modules ESP82, illustrant le partage d'événements entre plusieurs dispositifs dans un réseau.



Vidéo de démonstration :



Pour ce faire, nous nous abonons au Topic bouton :

```
void reconnect() {
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");

        String clientId = "ESP8266Client-";
        clientId += String(random(0xffff), HEX);

        if (client.connect(clientId.c_str())) {
            Serial.println("connected");

            // Abonnement au topic bouton
            client.subscribe("bouton");
            Serial.println("Subscribed to: bouton");

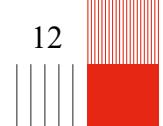
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            delay(5000);
        }
    }
}
```

Et lorsque nous recevons un message « ON » nous allumons la LED, et lorsque que nous recevons un « OFF » on éteint la LED.

```
void callback(char* topic, byte* payload, unsigned int length) {
    Serial.print("Message arrived [ ");
    Serial.print(topic);
    Serial.print("] ");

    // Convert payload en String
    String message = "";
    for (int i = 0; i < length; i++) {
        message += (char)payload[i];
    }
    Serial.println(message);

    if (message == "ON") {
        digitalWrite(BUILTIN_LED, LOW);
        Serial.println("LED ON");
    }
    else if (message == "OFF") {
        digitalWrite(BUILTIN_LED, HIGH);
    }
}
```



```
    Serial.println("LED OFF");
}
}
```

3. TP 3 : Fast application prototyping for IoT

Ce TP a pour objectif de concevoir une application IoT complète et fonctionnelle. Nous poursuivons le travail commencé au cours des TP1 et TP2, au cours desquels nous avons mis en place un broker MQTT (Mosquitto) ainsi que le protocole de communication MQTT, et assuré son déploiement sur une carte ESP8266.

Dans ce troisième TP, l'objectif est de développer rapidement une application de haut niveau en s'appuyant sur l'outil Node-RED, afin de simplifier l'interconnexion des différents composants. L'application développée permet de relier des capteurs et des actionneurs, de visualiser en temps réel les données collectées, et de déclencher automatiquement des actions en fonction de règles et de seuils prédéfinis.

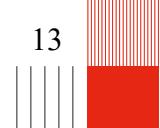
Ce travail met en évidence l'intérêt des outils de prototypage rapide pour le développement d'applications IoT, permettant ainsi de réduire significativement le temps de développement.

3.1. Mise en place du dispositif IoT

L'architecture est composée d'une carte ESP8266 ; un capteur de luminosité et un bouton.



Afin de pouvoir mettre en place l'application, nous avons structuré le programme embarqué sur l'ESP8266 de la manière suivante.



Nous utilisons le protocole MQTT pour assurer la communication entre l'ESP8266 et le broker. Deux topics sont dédiés à la publication des données issues des capteurs :

- ***sensor/button*** : état du bouton (appuyé ou relâché)
- ***sensor/light*** : valeur du capteur de luminosité

Ces topics permettent de transmettre périodiquement les informations des capteurs vers les clients abonnés.

Par ailleurs, l'ESP8266 s'abonne au topic ***led/control***, qui permet de recevoir des commandes distantes afin d'allumer ou d'éteindre la LED intégrée à l'esp.

Abonnement au topic de commande

L'abonnement au topic ***led/control*** est réalisé dans la fonction ***reconnect()***, laquelle est automatiquement appelée en cas de déconnexion du broker MQTT. Cela garantit que l'ESP8266 se réabonne correctement après une perte de connexion.

```
client.subscribe("led/control");
```

Traitement des messages reçus

Dans la fonction de callback, nous avons ajouté le code suivant afin de mettre à jour l'état de la LED à la réception d'un message sur le topic dédié ***led/control***.

```
// Commande de la LED
if (String(topic) == "led/control") {

    if (message == "ON" || message == "1") {
        digitalWrite(BUILTIN_LED, LOW);    // LED ON (active LOW)
        Serial.println("LED ALLUMÉE");
    }
    else if (message == "OFF" || message == "0") {
        digitalWrite(BUILTIN_LED, HIGH);   // LED OFF
        Serial.println("LED ÉTEINTE");
    }
}
```

Ainsi, la LED peut être contrôlée à distance via des messages MQTT simples (ON/OFF ou 1/0).

Publication des données des capteurs

Enfin, dans la boucle principale (***loop()***), l'état des capteurs est mis à jour et publié sur les topics associés toutes les 2 secondes. Cette temporisation permet d'éviter une surcharge du réseau tout en conservant une mise à jour régulière des données.

```

if (now - lastMsg > 2000) {
    lastMsg = now;

    // Lecture du capteur de lumière
   轻光值 = analogRead(LIGHT_PIN);

    // Lecture du bouton (0 = appuyé, 1 = relâché)
    按钮值 = digitalRead(BUTTON_PIN);

    // Conversion en chaîne
    sprintf(光Msg, 10, "Lum : %d", 轻光值);
    sprintf(按钮Msg, 10, "Butt : %d", 按钮值);

    // Publication MQTT
    client.publish("sensor/light", 光Msg);
    client.publish("sensor/button", 按钮Msg);

    // Debug
    Serial.print("Lumière : ");
    Serial.print(轻光值);
    Serial.print(" | Bouton : ");
    Serial.println(按钮值);
}

```

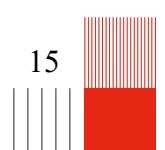
Ce mécanisme permet à tout client MQTT abonné aux topics correspondants de recevoir en temps réel l'état du bouton et la valeur du capteur de luminosité.

3.2. Développement de l'application avec Node-RED

Après avoir installé Node-RED nous avons dans un premier temps effectué un test simple sur Node-RED.



Nous regardons simplement les éléments écrit sur le topic associé au capteur. Et nous recevons bien les messages générés par l'esp.





```

[Tous les noeuds] [Tout]
16/12/2025 18:09:40 noeud: debug 1
sensor/button : msg.payload : string[8]
"Butt : 0"
16/12/2025 18:09:40 noeud: debug 1
sensor/light : msg.payload : string[9]
"Lum : 736"
16/12/2025 18:09:40 noeud: debug 1
sensor/button : msg.payload : string[8]
"Butt : 0"
16/12/2025 18:09:40 noeud: debug 1
sensor/light : msg.payload : string[9]
"Lum : 735"
16/12/2025 18:09:40 noeud: debug 1
sensor/button : msg.payload : string[8]
"Butt : 0"
16/12/2025 18:09:40 noeud: debug 1
sensor/light : msg.payload : string[9]
"Lum : 736"
16/12/2025 18:09:40 noeud: debug 1

```

4. TP 4 : Middleware spécialisé

1. Architecture : Quel est le rôle de MQTT et pourquoi ne pas utiliser HTTP ?

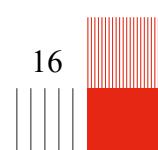
Le protocole MQTT permet la communication entre des objets connectés. Le protocole MQTT repose sur 2 principales composantes, un broker MQTT, et un client MQTT (n'importe quel appareil utilisant MQTT et se connectant à un broker). Le broker est chargé de recevoir tous les messages, et les envoyer aux clients abonnés. Il permet de gérer l'authentification et l'autorisation des clients.

MQTT repose sur le principe de publication / abonnement géré par le broker. Designé spécialement pour l'IoT, un des principaux objectifs de MQTT est de minimiser le data overhead de chaque paquet MQTT.

On peut ainsi voir une simple comparaison disponible sur :

<https://www.hivemq.com/blog/mqtt-vs-http-protocols-in-iot-iiot/>

	MQTT Bytes	HTTP Bytes
Establish connection	5572	2261
Disconnect	376 (optional)	0



For each message published	388	3285
Sum for 1 message	6336	5546
Sum for 10 messages	9829	55,460
Sum for 100 messages	44,748	554,600

Ainsi dans le cadre de l'IoT représente un réel avantage par rapport à http parce qu'il est moins lourd car http possède des headers importants et sur un modèle de requête/réponse. De plus HTTP nécessite que le client initie chaque communication, alors que MQTT permet au broker d'envoyer directement vers les abonnés.

2. Sécurité : Quelles mesures pour sécuriser la communication ESP8266 ↔ Home Assistant ?

Une des principales mesures de sécurité à mettre en place est l'utilisation de l'authentification à l'aide d'un nom d'utilisateur et d'un mot de passe pour éviter les connexions anonymes (connexion réalisable dans Home Assistant).

Un autre aspect de sécurité qui peut être mis en place est l'utilisation de permission afin de restreindre les autorisations à certain topic pour chaque device. Et interdire tout connexion anonyme.

Il est également possible de sécuriser chiffrer les communications via TLS/SSL sur le broker MQTT. Ce mécanisme permet ainsi de protéger la communication contre l'interception ou modification des messages.

<https://franck-ladriere.com/comment-chiffrer-efficacement-les-flux-de-donnees-en-mqtt-dans-un-environnement-smarthome/>

3. Scalabilité : Comment gérer 50 ESP8266 dans un bâtiment ?

MQQT se basant sur le principe Topic il est alors possible d'opté pour une organisation cohérente correspondant à la topologie du bâtiment. Nous pouvons ainsi obtenir une organisation comme suit :

batiment/etageX/salleY/capteurZ

Ou

batiment/etageX/salleY/actionneurZ

Ou

batiment/etageX/couloireY/capteurZ

Il serait également intéressant de répartir les différents nœuds sur plusieurs points d'accès Wi-Fi afin d'éviter qu'il y ait comme en TP une saturation.

4. Optimisation : Réduire la consommation énergétique de l'ESP8266

Le mécanisme le plus évident et efficace à mettre en œuvre est l'utilisation du *mode sleep* sur l'ESP82. L'ESP82 possède notamment un mode deepsleep, permettant d'endormir le CPU, le Wi-Fi et la majorité des circuits. Il faut pour cela relier la broche RST à GPIO16 pour que le timer interne puisse provoquer un reset.

On peut ensuite utiliser la commande suivante : `ESP.deepSleep(Time);`

Ainsi dans le cas de l'utilisation d'un ESP en tant que GateWay à chaque cycle l'ESP se réveille, il se connecte au Wi-Fi, il lit ensuite la valeur de ces capteurs et renvoie ensuite les valeurs via MQTT, et enfin il se rendort.

Ainsi plus la période d'endormissement sera grande moins l'ESP82 consommera.

5. Extension : Exemple de cas d'usage réel

On peut imaginer une serre connectée entièrement automatisée, dont l'objectif est de maintenir des conditions optimales de croissance pour différentes plantes. Grâce à un réseau de capteurs et d'actionneurs pilotés via Home Assistant et MQTT, la serre devient capable de s'autoréguler.

La serre peut être équipée de plusieurs types de capteurs pour monitorer les différentes données :

- Capteur de luminosité : pour mesurer l'intensité lumineuse pour ajuster l'éclairage.
- Capteurs de température : pour surveiller le climat interne et déclencher le chauffage ou la ventilation.
- Capteurs d'humidité : pour déclencher automatiquement l'arrosage.
- Capteurs de croissance : pour suivre la taille des plantes

Et chaque capteur nœud de capteur peut publier régulièrement par l'intermédiaire d'une Gateway (par exemple un esp82) ses données sur des topics MQTT.