

MiddleWare for the IoT

TP N°1

BERRADA El Ghali
BERTA Pauline
CONCEICAO NUNES Joao

THEORICAL PART

Based on web resources and the previous course respond to those questions:

- **What is the typical architecture of an IoT system based on the MQTT protocol?**

The typical architecture of a system based on the MQTT protocol is a system with multiple sensors with variable IP addresses, because they move in space, for example, and change networks often. The architecture also englobes multiple clients and servers, and the MQTT protocol allows everything to be managed and centralised by a “broker” that coordinates every connection and information exchange.

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

The IP protocol under MQTT normally is TCP. But every protocol that usually provides ordered, lossless, bi-directional connection can support MQTT. It's designed for connection with remote locations where a “small code footprint” is required and also where the network bandwidth is limited, so it can be used on low power and low bandwidth networks.

- **What are the different versions of MQTT?**

- MQTT v3.1.0
- MQTT v3.1.1 - In common Use
- MQTT Version 5, last version. Currently limited Use
- MQTT-SN, designed to run over UDP, ZigBee etc. Not very popular but will certainly become more useful now that IoT is taking off.

- **What kind of security/authentication/encryption are used in MQTT?**

MQTT supports x509 client certificates, that is the most secure method of client authentication.

For data security, TLS and SSL security is supported, as well as Payload encryption.

The data is encrypted at the application level, and not by the brocker, so it means that the data is encrypted “end to end” and not just between the brocker and the client.

- Suppose you have devices that include one button, one light and luminosity sensor. You would like to create a smart system for you 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

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

The topics :

- Button state
- Lum limit state

The connections will be between the light sensor and the lum limit state, the button and the button state for the publishing type of connection. The light would be the client and would subscribe to these two topics. That way it will receive every message and will update accordingly.

PRACTICAL PART

STEP n°1 :

Installation of Mosquitto.

Run the broker with the file.exe.

Test the command *mosquitto_pub* and *mosquitto_sub*.

Command *mosquitto_sub*:

mosquitto_sub -t button/# -v

We use the command to subscribe and listen if a message is arriving.

Command *mosquitto_pub*:

mosquitto_pub -m "test message" -t button/jaune

We use the command to send a message in a topic "jaune" contained in the button.

Send :

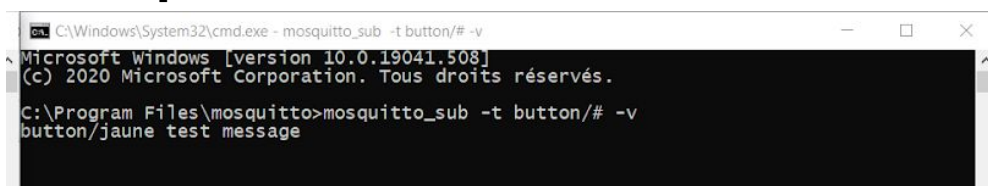


```
C:\Windows\System32\cmd.exe
Microsoft Windows [version 10.0.19041.508]
(c) 2020 Microsoft Corporation. Tous droits réservés.

C:\Program Files\mosquitto> mosquitto_pub -m "test message" -t button/jaune

C:\Program Files\mosquitto>
```

Reception :



```
C:\Windows\System32\cmd.exe - mosquitto_sub -t button/# -v
Microsoft Windows [version 10.0.19041.508]
(c) 2020 Microsoft Corporation. Tous droits réservés.

C:\Program Files\mosquitto> mosquitto_sub -t button/# -v
button/jaune test message
```

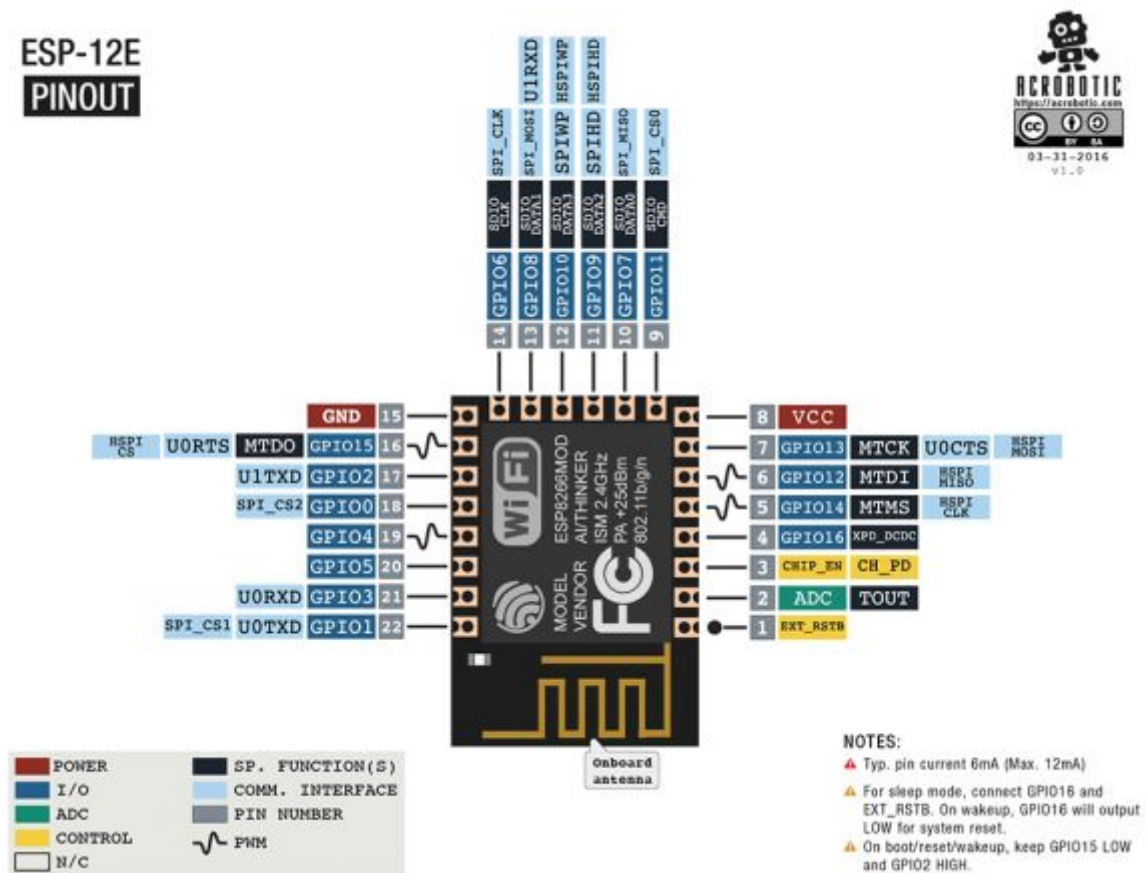
STEP n°2 :

We installed the Arduino IDE and opened an example that we can find on the library arduinoMqtt, which is named connectESP8266wificlient.

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

The ESP8266 is a microcontroller 32 bits, which integrates the IEEE 802.11 b/g/n Wi-Fi standard. The nodeMCU board needs between 3.0V and 3.6V to operate, there are 16 inputs/outputs (GPIO).

We have here an image retracing the inputs/outputs capabilities :



About configuration, we can program the module with several languages :

- Commandes AT
- ESP8266 SDK
- Lua (NodeMCU)
- C/C++ (Arduino)
- MicroPython
- Javascript

MQTT information :

We send a message from the wireless router to the electronic card, then the message is sent to the broker that we have installed on the computer.

The mosquitto broker manages a set of topics. A topic can be described as a queue of messages.

Basically the broker centralises the information sent by the publishers so that everything is available for every subscriber. The publisher is a sensor and the subscriber a client, like a server on the cloud.

STEP n°3 :

Creation of an application

Blocs that we have modified from the example.

Bloc Configuration

```
// Setup WiFi network
WiFi.mode(WIFI_STA);
WiFi.hostname("ESP_" MQTT_ID);
WiFi.begin("Cisco38658", "");
LOG_PRINTFLN("\n");
LOG_PRINTFLN("Connecting to WiFi");
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    LOG_PRINTFLN(".");
}
LOG_PRINTFLN("Connected to WiFi");
LOG_PRINTFLN("IP: %s", WiFi.localIP().toString().c_str());
```

Bloc Sub

```
{
    MqttClient::Error::type rc = mqtt->subscribe(
        MQTT_TOPIC_SUB, MqttClient::QOS0, processMessage
    );
    if (rc != MqttClient::Error::SUCCESS) {
        LOG_PRINTFLN("Subscribe error: %i", rc);
        LOG_PRINTFLN("Drop connection");
        mqtt->disconnect();
        return;
    }
}
```

Bloc Pub

```
{  
    const char* buf = "Hello";  
    MqttClient::Message message;  
    message.qos = MqttClient::QOS0;  
    message.retained = false;  
    message.dup = false;  
    message.payload = (void*) buf;  
    message.payloadLen = strlen(buf);  
    mqtt->publish(MQTT_TOPIC_PUB, message);  
}
```

Results that we can observe in the broker Mosquitto.

Mosquitto_sub

```
C:\Program Files\mosquitto>mosquitto_pub -m "test_message" -t test/TEST-ID/sub
```

COM3

```
MQTT - Process message, type: 3  
MQTT - Publish received, qos: 0  
MQTT - Deliver message for: test/TEST-ID/sub  
Message arrived: qos 0, retained 0, dup 0, packetid 0, payload:[test_message]  
MQTT - Keepalive, ts: 326287  
MQTT - Process message, type: 13  
MQTT - Keepalive ack received, ts: 326561  
MQTT - Publish, to: test/TEST-ID/pub, size: 5  
MQTT - Yield for 29999 ms  
MQTT - Keepalive, ts: 344282  
MQTT - Process message, type: 13  
MQTT - Keepalive ack received, ts: 344438  
MQTT - Keepalive, ts: 356288  
MQTT - Process message, type: 13  
MQTT - Keepalive ack received, ts: 356533
```

☒ Défilement automatique ☐ Afficher l'horodatage

Mosquitto_pub

```
C:\Program Files\mosquitto>mosquitto_sub -t test/# -v  
test/TEST-ID/pub Hello  
test/TEST-ID/pub Hello  
test/TEST-ID/pub Hello  
test/TEST-ID/pub Hello
```

The core of the entire program :

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

// Enable MqttClient logs
#define MQTT_LOG_ENABLED 1
// Include library
#include <MqttClient.h>

#define LOG_PRINTFLN(fmt, ...) logfln(fmt, ##__VA_ARGS__)
#define LOG_SIZE_MAX 128
#define BUTTON D1
#define SENSOR A0
#define LED D0

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;
const char* MQTT_TOPIC_PUB = "test/" MQTT_ID "/pub";
const char* MQTT_TOPIC_SUB = "test/" MQTT_ID "/sub";
const char* MQTT_TOPIC_BUTTON_PUB = "button/" MQTT_ID "/pub";

int buttonOutput = 0;
// ===== Object to supply system functions =====
class System: public MqttClient::System {
public:

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

```

        void yield(void) {
            ::yield();
        }
};

// ===== Setup all objects =====

void setup() {
    // Setup hardware serial for logging
    Serial.begin(HW_UART_SPEED);
    while (!Serial);

    // Setup WiFi network
    WiFi.mode(WIFI_STA);
    WiFi.hostname("ESP_" MQTT_ID);
    WiFi.begin("Cisco38658", "");
    LOG_PRINTFLN("\n");
    LOG_PRINTFLN("Connecting to WiFi");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        LOG_PRINTFLN(".");
    }
    LOG_PRINTFLN("Connected to WiFi");
    LOG_PRINTFLN("IP: %s", WiFi.localIP().toString().c_str());

    // Setup MqttClient
    MqttClient::System *mqttSystem = new System;
    MqttClient::Logger *mqttLogger = new MqttClient::LoggerImpl<HardwareSerial>(Serial);
    MqttClient::Network *mqttNetwork = new
MqttClient::NetworkClientImpl<WiFiClient>(network, *mqttSystem);
    //// Make 128 bytes send buffer
    MqttClient::Buffer *mqttSendBuffer = new MqttClient::ArrayBuffer<128>();
    //// Make 128 bytes receive buffer
    MqttClient::Buffer *mqttRecvBuffer = new MqttClient::ArrayBuffer<128>();
    //// Allow up to 2 subscriptions simultaneously
    MqttClient::MessageHandlers *mqttMessageHandlers = new
MqttClient::MessageHandlersImpl<2>();
    //// Configure client options
    MqttClient::Options mqttOptions;
    ///// Set command timeout to 10 seconds
    mqttOptions.commandTimeoutMs = 10000;
    //// Make client object
    mqtt = new MqttClient(
        mqttOptions, *mqttLogger, *mqttSystem, *mqttNetwork, *mqttSendBuffer,
        *mqttRecvBuffer, *mqttMessageHandlers
    );
};

```



```

// Pin Configuration
pinMode(BUTTON,INPUT);
pinMode(SENSOR, INPUT);
pinMode(LED, OUTPUT);
digitalWrite(LED,HIGH);
}

// ===== Subscription callback =====
void processMessage(MqttClient::MessageData& md) {
    const MqttClient::Message& msg = md.message;
    char payload[msg.payloadLen + 1];
    memcpy(payload, msg.payload, msg.payloadLen);
    payload[msg.payloadLen] = '\0';
    LOG_PRINTFLN(
        "Message arrived: qos %d, retained %d, dup %d, packetid %d, payload:[%s]",
        msg.qos, msg.retained, msg.dup, msg.id, payload
    );
}

// ===== Main loop =====
void loop() {
    // Check connection status
    if (!mqtt->isConnected()) {
        // Close connection if exists
        network.stop();
        // Re-establish TCP connection with MQTT broker
        LOG_PRINTFLN("Connecting");
        network.connect("192.168.1.110", 1883);
        if (!network.connected()) {
            LOG_PRINTFLN("Can't establish the TCP connection");
            delay(5000);
            ESP.reset();
        }
        // Start new MQTT connection
        MqttClient::ConnectResult connectResult;
        // Connect
        {
            MQTTPacket_connectData options = MQTTPacket_connectData_initializer;
            options.MQTTVersion = 4;
            options.clientID.cstring = (char*)MQTT_ID;
            options.cleansession = true;
            options.keepAliveInterval = 15; // 15 seconds
            MqttClient::Error::type rc = mqtt->connect(options, connectResult);
            if (rc != MqttClient::Error::SUCCESS) {
                LOG_PRINTFLN("Connection error: %i", rc);
                return;
            }
        }
    }
}

```



```

        }
    }
    {
        MqttClient::Error::type rc = mqtt->subscribe(
MQTT_TOPIC_SUB, MqttClient::QOS0, processMessage
);
if (rc != MqttClient::Error::SUCCESS) {
    LOG_PRINTFLN("Subscribe error: %i", rc);
    LOG_PRINTFLN("Drop connection");
    mqtt->disconnect();
    return;
}
    }
    } else {
        {
            const char* buf = "Hello";

MqttClient::Message message;
message.qos = MqttClient::QOS0;
message.retained = false;
message.dup = false;
message.payload = (void*) buf;
message.payloadLen = strlen(buf);
mqtt->publish(MQTT_TOPIC_PUB, message);
        }
        // Idle for 3 seconds
        mqtt->yield(3000L);
    }
buttonOutput = digitalRead(BUTTON);
if (buttonOutput){
    const char* buf = "Button_Pressed";
    MqttClient::Message message;
    message.qos = MqttClient::QOS0;
    message.retained = false;
    message.dup = false;
    message.payload = (void*) buf;
    message.payloadLen = strlen(buf);
    mqtt->publish(MQTT_TOPIC_BUTTON_PUB, message);
    //LED control
    digitalWrite(LED,LOW);
}
else{
    digitalWrite(LED,HIGH);
}
}
}

```

Sources :

Wikipedia

<http://www.steves-internet-guide.com/mqtt/>

<https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/#:~:text=In%20MQTT%2C%20the%20word%20topic,MQTT%20topics%20are%20very%20lightweight>

Links use during the TP :

https://mosquitto.org/man/mosquitto_sub-1.html

http://www.steves-internet-guide.com/mosquitto_pub-sub-clients/

<http://www.steves-internet-guide.com/understanding-mqtt-topics/#:~:text=%20Understanding%20MQTT%20Topics%20%201%20The%20%24SYS.publish%20to%20an%20individual%20topic.%20That...%20More%20>