Ewan Mackay - Abir Bennazzouz - Octobre 2021

Middleware For IoT
TP1 MQTT

What is the typical architecture of an IoT system based on the MQTT protocol?
MQTT has a client/server model, where every sensor is a client and connects to a server, known as a broker, over TCP. MQTT is message oriented. Every message is a discrete chunk of data, opaque to the broker.

What is the IP protocol under MQTT?
MQTT relies on the TCP protocol for data transmission. A variant, MQTT-SN, is used over other transports such as UDP or Bluetooth.

What does it mean in terms of bandwidth usage, type of communication, etc ?
Both the client and the broker need to have a TCP/IP stack. We can say that it is connection-oriented devices first establish a connection with each other before they send data), bidirectional, multiply-connected, reliable (checked for data integrity), acknowledged, stream-oriented (TCP allows applications to send it a continuous stream of data for transmission) and flow-managed.

What are the different versions of MQTT?
The different versions of MQTT are :
- MQTT v3.1
- MQTT v 3.1.1 (in common use)
- MQTT v5.0 (Currently limited use)
- MQTT-SN (SN stands for Sensor Networks)

What kind of security/authentication/encryption are used in MQTT?
MQTT uses Transport Layer Security (TLS) encryption with user name, password protected connections, and optional certifications that requires clients to provide a certificate file that matches with the server's. With MQTT broker architecture, the devices and applications become decoupled and more secure. MQTT eliminates vulnerable and insecure client connections, manages and tracks all client connection states, including security credentials and certificates. MQTT reduces network strain without compromising the security (cellular or satellite network).

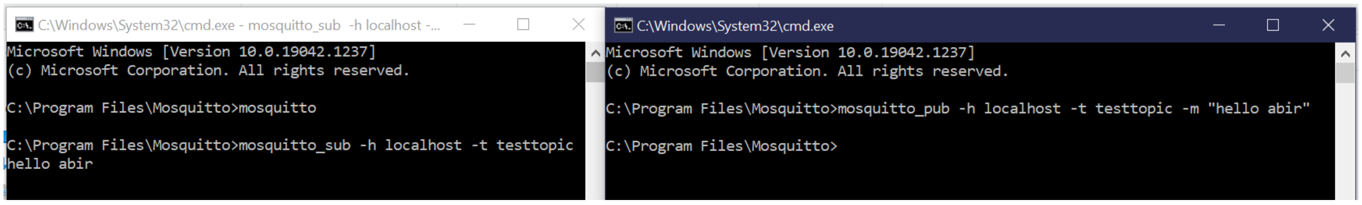
Suppose you have devices that include one button, one light and luminosity sensor. You would like to create a smart system for your house with this behavior:
- you would like to be able to switch on the light manually with the button
- the light is automatically switched on when the luminosity is under a certain value

What different topics will be necessary to get this behavior and what will the connection be in terms of publishing or subscribing?
Ideally, in this case we would use two topics, one for the switch and one for the light sensor.

Ewan Mackay - Abir Bennazzouz - Octobre 2021

Installing and testing the broker

Below is the test of the mosquitto broker, with the creation of the topic, subscription to that topic and publishing of a message to that topic.



C:\Windows\System32\cmd.exe - mosquitto_sub -h localhost -...

```
Microsoft Windows [Version 10.0.19042.1237]
(c) Microsoft Corporation. All rights reserved.

C:\Program Files\Mosquitto>mosquitto

C:\Program Files\Mosquitto>mosquitto_sub -h localhost -t testtopic
hello abir
```

C:\Windows\System32\cmd.exe

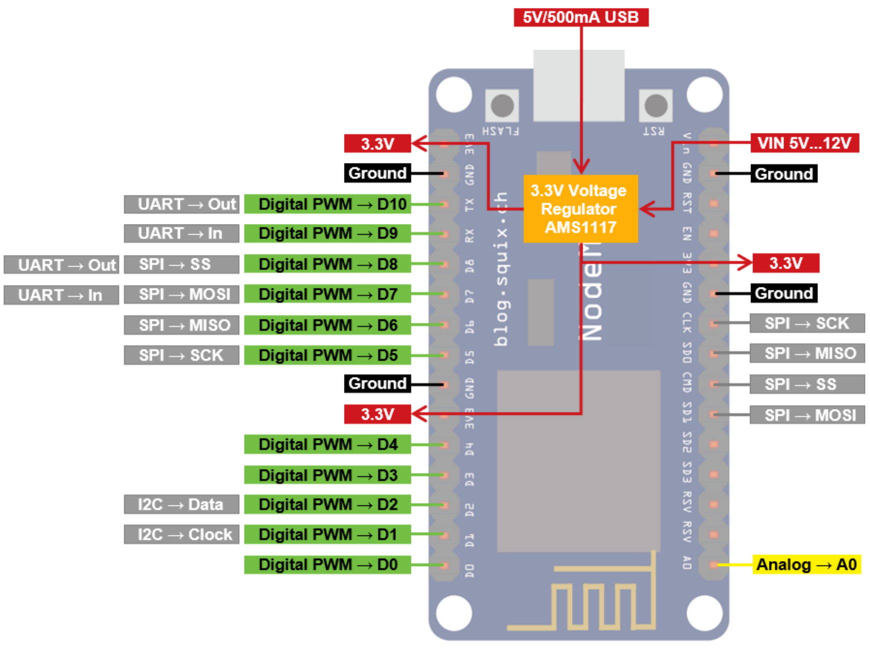
```
Microsoft Windows [Version 10.0.19042.1237]
(c) Microsoft Corporation. All rights reserved.

C:\Program Files\Mosquitto>mosquitto_pub -h localhost -t testtopic -m "hello abir"

C:\Program Files\Mosquitto>
```

As we can see, once we are subscribed to a topic, any message published to that topic is shown.

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

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

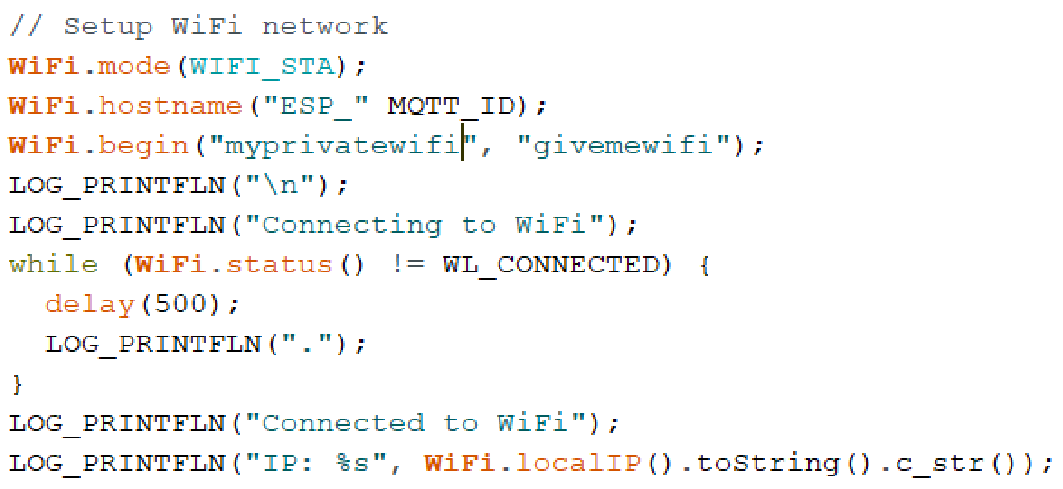
The ESP8266 NodeMCU microcontroller’s firmware can be programmed using the LUA language and code can be flashed onto the µController in C.
The ESP8266 NodeMCU V2 has in total 11 digital input / output pins. All these 11 pins can create PWM signals and have a maximum output current of 12mA. The ESP8266 has only 1 analog input A0 that is connected internally with a 10-bit analog-to-digital converter (ADC) to convert the analog voltage into 1024 digital values between 0 and 1023.



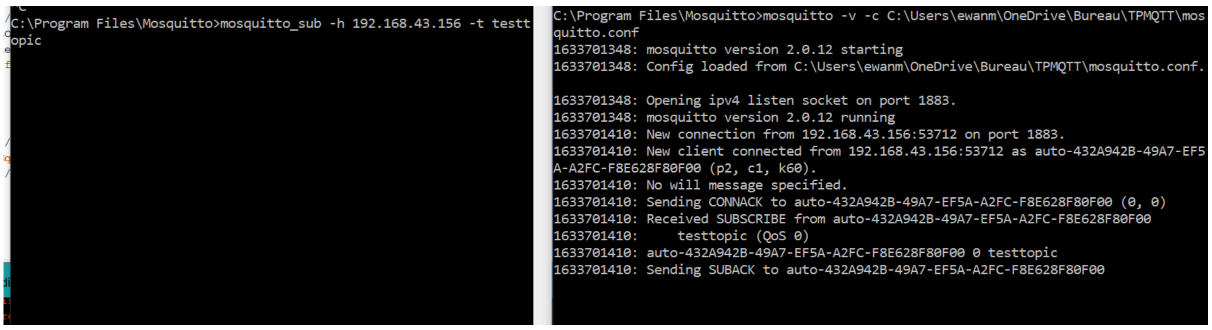
ESP8266 NodeMCU Input/Outputs

Code example

2 sections in the Arduino code: setup and loop on the Arduino IDE
Setup : Setup WiFi network, Setup MqttClient (system, logger and network), Make 128 bytes of send and receive buffer, configure client and finally Make client object
Loop : Re-establish TCP connection with MQTT broker, connect to new MQTT connection, sub and or publish.
We modify the address of the MQTT server to be able to connect to the broker on the PC through a local network hotspot.



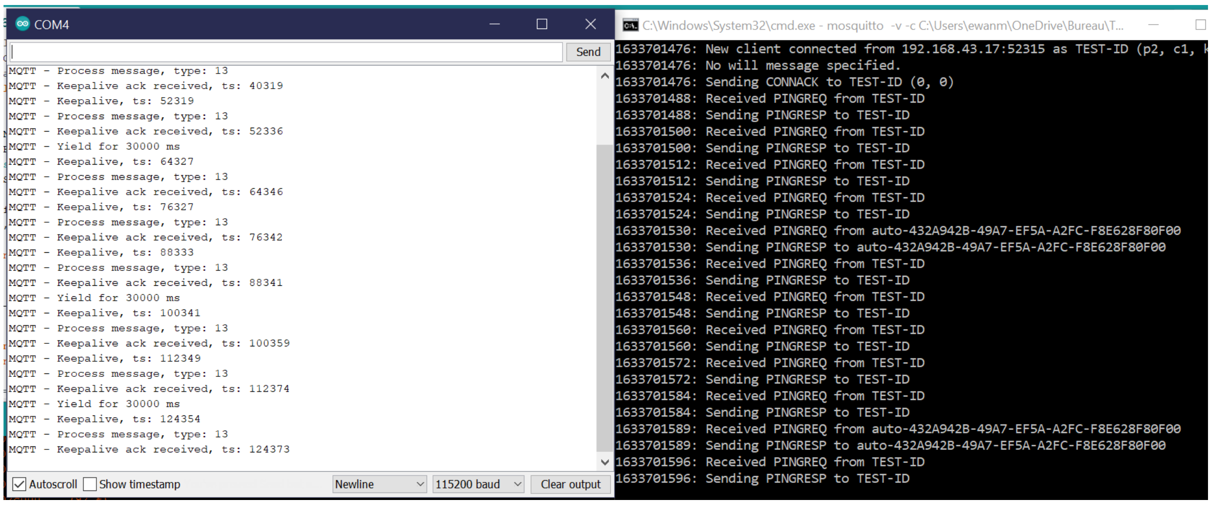
```
// Setup WiFi network
WiFi.mode(WIFI_STA);
WiFi.hostname("ESP_" MQTT_ID);
WiFi.begin("myprivatewifi", "givemewifi");
LOG_PRINTFLN("\n");
LOG_PRINTFLN("Connecting to WiFi");
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  LOG_PRINTFLN(".");
}
LOG_PRINTFLN("Connected to WiFi");
LOG_PRINTFLN("IP: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



```
C:\Program Files\Mosquitto>mosquitto_sub -h 192.168.43.156 -t testt
opic
```

```
C:\Program Files\Mosquitto>mosquitto -v -c C:\Users\ewanm\OneDrive\Bureau\TPMQTT\mos
quitto.conf
1633701348: mosquitto version 2.0.12 starting
1633701348: Config loaded from C:\Users\ewanm\OneDrive\Bureau\TPMQTT\mosquitto.conf.

1633701348: Opening ipv4 listen socket on port 1883.
1633701348: mosquitto version 2.0.12 running
1633701410: New connection from 192.168.43.156:53712 on port 1883.
1633701410: New client connected from 192.168.43.156:53712 as auto-432A942B-49A7-EF5
A-A2FC-F8E628F80F00 (p2, c1, k60).
1633701410: No will message specified.
1633701410: Sending CONNACK to auto-432A942B-49A7-EF5A-A2FC-F8E628F80F00 (0, 0)
1633701410: Received SUBSCRIBE from auto-432A942B-49A7-EF5A-A2FC-F8E628F80F00
1633701410:     testtopic (QoS 0)
1633701410: auto-432A942B-49A7-EF5A-A2FC-F8E628F80F00 0 testtopic
1633701410: Sending SUBACK to auto-432A942B-49A7-EF5A-A2FC-F8E628F80F00
```

Connection between esp and broker validated. The MQTT connection is done between you device and the broker by looking through the serial port of the Arduino IDE:

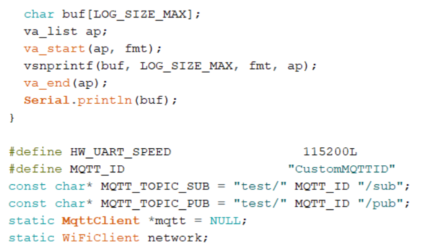

COM4
Send
Autoscroll
Show timestamp
Newline
115200 baud
Clear output
C:\Windows\System32\cmd.exe - mosquitto -v -c C:\Users\ewanm\OneDrive\Bureau\T...

```
MQTT - Process message, type: 13
MQTT - Keepalive ack received, ts: 40319
MQTT - Keepalive, ts: 52319
MQTT - Process message, type: 13
MQTT - Keepalive ack received, ts: 52336
MQTT - Yield for 30000 ms
MQTT - Keepalive, ts: 64327
MQTT - Process message, type: 13
MQTT - Keepalive ack received, ts: 64346
MQTT - Keepalive, ts: 76327
MQTT - Process message, type: 13
MQTT - Keepalive ack received, ts: 76342
MQTT - Keepalive, ts: 88333
MQTT - Process message, type: 13
MQTT - Keepalive ack received, ts: 88341
MQTT - Yield for 30000 ms
MQTT - Keepalive, ts: 100341
MQTT - Process message, type: 13
MQTT - Keepalive ack received, ts: 100359
MQTT - Keepalive, ts: 112349
MQTT - Process message, type: 13
MQTT - Keepalive ack received, ts: 112374
MQTT - Yield for 30000 ms
MQTT - Keepalive, ts: 124354
MQTT - Process message, type: 13
MQTT - Keepalive ack received, ts: 124373
```

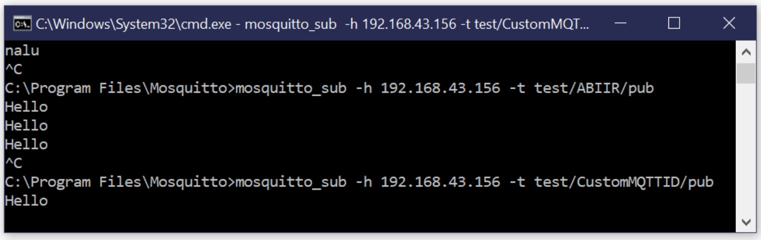
```
1633701476: New client connected from 192.168.43.17:52315 as TEST-ID (p2, c1,
1633701476: No will message specified.
1633701476: Sending CONNACK to TEST-ID (0, 0)
1633701488: Received PINGREQ from TEST-ID
1633701488: Sending PINGRESP to TEST-ID
1633701500: Received PINGREQ from TEST-ID
1633701500: Sending PINGRESP to TEST-ID
1633701512: Received PINGREQ from TEST-ID
1633701512: Sending PINGRESP to TEST-ID
1633701524: Received PINGREQ from TEST-ID
1633701524: Sending PINGRESP to TEST-ID
1633701530: Received PINGREQ from auto-432A942B-49A7-EF5A-A2FC-F8E628F80F00
1633701530: Sending PINGRESP to auto-432A942B-49A7-EF5A-A2FC-F8E628F80F00
1633701536: Received PINGREQ from TEST-ID
1633701536: Sending PINGRESP to TEST-ID
1633701548: Received PINGREQ from TEST-ID
1633701548: Sending PINGRESP to TEST-ID
1633701560: Received PINGREQ from TEST-ID
1633701560: Sending PINGRESP to TEST-ID
1633701572: Received PINGREQ from TEST-ID
1633701572: Sending PINGRESP to TEST-ID
1633701584: Received PINGREQ from TEST-ID
1633701584: Sending PINGRESP to TEST-ID
1633701589: Received PINGREQ from auto-432A942B-49A7-EF5A-A2FC-F8E628F80F00
1633701589: Sending PINGRESP to auto-432A942B-49A7-EF5A-A2FC-F8E628F80F00
1633701596: Received PINGREQ from TEST-ID
1633701596: Sending PINGRESP to TEST-ID
```

Ewan Mackay - Abir Bennazzouz - Octobre 2021

Adding a publish/subscribe behavior in our device :
By copying the publish and subscribe and necessary declaration and put that in the previous example we can check that our custom publish/subscribe behaviour works correctly.



```
  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                        "CustomMQTTID"
const char* MQTT_TOPIC_SUB = "test/" MQTT_ID "/sub";
const char* MQTT_TOPIC_PUB = "test/" MQTT_ID "/pub";
static MqttClient *mqtt = NULL;
static WiFiClient network;
```



C:\Windows\System32\cmd.exe - mosquitto_sub -h 192.168.43.156 -t test/CustomMQT...

```
nalu
^C
C:\Program Files\Mosquitto>mosquitto_sub -h 192.168.43.156 -t test/ABIIR/pub
Hello
Hello
Hello
^C
C:\Program Files\Mosquitto>mosquitto_sub -h 192.168.43.156 -t test/CustomMQTTID/pub
Hello
```

Creation of the application

Creation of two topics :

- one where the lux meter will publish it’s analog value, with a callback translating this value into a digital value and acting upon it
- one for communication with the switch and acting upon it

Below is the callback function that is called upon every message being sent to the LuxMeter Topic. This allows us to follow the value of the luxmeter and either turn on/turn off the led accordingly, at a regular interval.

```
// ============== Subscription callback ========================================
void processMessage(MqttClient::MessageData& md) {
  static int payload_int;
  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
  );
  payload_int = atoi(payload); //casting to an int
  if (payload_int <100) {
      digitalWrite(BUILTIN_LED, 1);
    }
    else {
      digitalWrite(BUILTIN_LED, 0);
  }
}

 //Définition des pins
 const int analogInPin = A0;  // Analog input pin that the luxmeter is attached to
 const int digitalInPin = D2; // Digital input pin that the switch is attached to
```

Below is the subscribe/publish behavior to deal with the luxmeter. We sent the value of the luxmeter within each message that will later be decoded, it’s type adapted to be read and acted upon.

```
  // Subscribe
  {
    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 {
  // Publish
    lightSensorValue = analogRead(ANALOG_PIN);
    char cstr[16];
    itoa(lightSensorValue, cstr, 10);   //int to char cast
      const char* buf = "Hello";
    MqttClient::Message message;
    message.qos = MqttClient::QOS0;
    message.retained = false;
    message.dup = false;
    message.payload = cstr;
    message.payloadLen = strlen(cstr);
    mqtt->publish(MQTT_TOPIC_PUB, message);  //send the lux meter value
    //LOG_PRINTFLN("message publie :%s",buf);
    LOG_PRINTFLN("light = %i",lightSensorValue);
  }
  // Idle for 2 seconds
  mqtt->yield(2000L);
```

Conclusion

Through this practical, we have learnt how MQTT functions, how to create topics, how to publish and subscribe to these different topics. These notions were then applied to a real-life practical example. This practical, though not finished due to lack of time, was very instructive, and gave us a solid beginner’s approach to using MQTT in IoT applications.