MQTT : MQ Telemetry/Transport
A protocol for the IoT

Ph. Truillet

October 2021 – v. 1.6

1. MQTT: how does it work?

MQTT is an open, simple, lightweight and easy-to-implement protocol. This protocol is ideal for the following needs:

- the use of very low bandwidth,
- use on wireless networks,
- energy consumption,
- speed with a response time superior to other current web standards,
- reliability,
- and the use of low processor and memory resources.

MQTT is an ISO standard (**ISO/IEC PRF 20922:2016**[1]). **TT** (the protocol's first name) was originally developed by Andy Stanford-Clark (IBM) and Arlen Nipper (Eurotech, now Cirrus Link) in 1999 for controlling a pipeline in the desert. The objective was to have a bandwidth efficient protocol using little energy at a time when devices could only be connected through satellites.
The MQTT protocol uses a "*publish/subscribe*" architecture in contrast to HTTP and its "*request/response*" architecture (see Figure 1). The central point of communication is the MQTT broker (which by default uses port 1883 for unencrypted connections and port 8883 for SSL/TLS encrypted connections), in charge of relaying messages from senders to clients. Each client subscribes via a message to the broker: the "topic" (a kind of routing information for the broker) which will allow the broker to re-transmit messages received from data producers to clients. Clients and producers do not have to know each other, communicating only through topics. This architecture allows for multi-scale solutions.

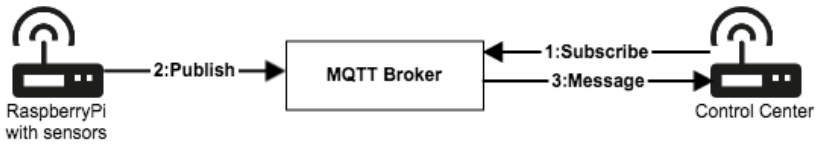


Figure 1: MQTT "Publish/Subscribe" architecture

Each MQTT client has a permanently open connection to the broker. If the connection goes down, the broker buffers the messages and sends them out as soon as the connection with the client is reconnected.
A "**MQTT topic**" is a string that can have a hierarchy of levels separated by the "**/**" character. For example, living room temperature information could be sent on the topic "**`house/living room/temperature`**" and kitchen temperature on "**`house/kitchen/temperature`**".
Note: Topics must contain at least one character (do not start with the "**$**" character, reserved for internal MQTT use), are case sensitive and the topic "**/**" is valid!

Wildcards

- The "**+**" sign is a *wildcard* character that allows arbitrary values for a particular level

E.g.: "**`house/+/temperature`**" allows you to subscribe to the temperatures of the kitchen and living room

- the "**#**" sign for more than one level (this sign must be at the end). The messages sent can be of any kind but cannot exceed 256 MB in size.

Ex: If a *publisher* subscribes to the *topic* "**`house/lounge/#`**", he will receive all data from the lounge. Similarly, if he subscribes to the *topic* "**`house/#`**", he will collect all data from the house sensors.

2. Security

The IoT data exchanged can be very critical, which is why it is also possible to secure the exchanges on several levels:

- Transport in SSL/TLS,
- Authentication with SSL/TLS certificates,
- Authentication by login/password.

3. QoS -Quality of Service

MQTT natively integrates the notion of QoS. Indeed, the *publisher* has the possibility to define the quality of its message. Three levels are possible:

- A **QoS level 0** "*At most once*" message will be delivered at most once. This means that the message is sent with no guarantee of receipt, (the broker does not inform the sender that it has received the message)
- A **QoS level 1** "*At least once*" message will be delivered at least once. The client will transmit several times if necessary until the Broker confirms that it has been transmitted on the network.
- A **QoS level 2** "*exactly once*" message will be compulsorily saved by the sender and will always be transmitted as long as the receiver does not confirm its sending on the network. The main difference is that the sender uses a more sophisticated recognition phase with the broker to avoid duplicate messages (slower but more secure).

4. Exercises

4.0 Preamble

In order to operate, you will need to use an MQTT broker. You can easily install one by going to `https://shiftr.io` and downloading the **Desktop App** (available on Linux, Windows and MacOS).

Note: if this does not work, you can go to: https://shiftr.io/try
Unzip and launch the shiftr.io *DesktopApp*: an MQTT broker is launched on your machine, ready to receive your messages! You should see a window open (see Figure 1)

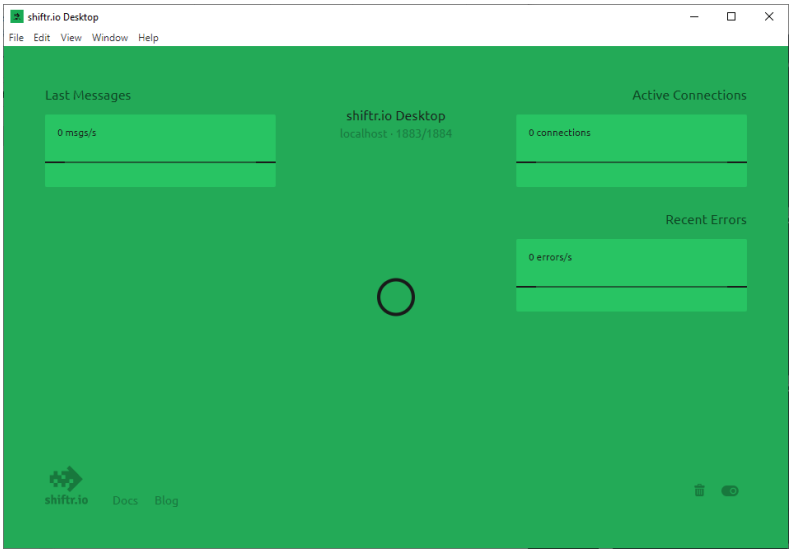
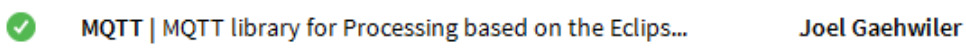


Figure 2: DesktopApp – shiftr.io

4.1 A first example with Processing.org

Open Processing, install the MQTT library (Menu "**Tools**" | "**Add a tool**", tab "**Libraries**" MQTT.



Open the example installed with the MQTT library ("**File**" menu | "**Examples...**", | "**Contributed Libraries**" | "**MQTT**", example *PublishSubscribe*.

Replace the url used in the `client.connect` statement line 20 "mqtt://try:try@broker.shiftr.io" with "`mqtt://localhost`".
Start the sketch. Press the space base in the Processing.org sketch and view the result on the DesktopApp (see Figure 3)

Figure 3: Viewing MQTT "Publish/Subscribe" messages

Exercise:

- Now write a Processing.org program from this example that generates random temperature values every second and sends them to the MQTT broker (**you must determine the topic**). Each instance sent will be considered a "*room*" in the house)
- Then write a program that will subscribe to all the temperatures emitted by each of the instances and display the overall average of the temperatures of the "*house*" (generated by each of the rooms) and the averages per "*room*".

4.2 Python (Paho)

MQTT is multi-language and you can use python as a support language (interesting for a Raspberry Pi for example). To use MQTT under Python, check the documentation here:
https://`pypi.python.org/pypi/paho-mqtt`

- Write the same data sender program in Python

4.3 Arduino

In the same way, it is possible to generate MQTT messages from an arduino compatible device connected to TCP/IP (via a wired network or WiFi)
First of all, you have to get the MQTT library by doing "**Sketch | Include a library | Manage libraries**" and look for the **MQTT** library (Arduino 1.8.7 and later)

In the "*Examples*" section, you will find some code to send data from arduino directly to MQTT.

Note: *By lack of ESP8266 or ESP32 type arduino devices, you may be able to skip this step.*

You can find a sample code for **ESP** here →
`https://github.com/truillet/upssitech/tree/master/Divers`

Note2: the code uses a *NodeMCU* and a *DHT22* humidity and temperature sensor

4.4 A MQTT broker

The best known and most used MQTT broker is **Mosquitto** (managed by the **eclipse** foundation)

You can download **Mosquitto** here: https://mosquitto.org
You also have access to the mosquitto MQTT broker online at the IP address **upssitech.co.uk** (ports 1883 and 9001 [websockets] are available with *login*: upssitech/ *password*: 2011) for your tests

5. Exercises

It is now a question of creating, managing and supervising a network of sensors from "**real local**" data and remotely (using, for example, data retrieved via a **REST API**). To do this :

1. create an MQTT client that retrieves local weather data from **OpenWeatherMap** and sends it to the broker.
2. create MQTT clients from **arduino** compatible microcontrollers and sensors for luminosity, temperature (or other) probes ... and generate values to the MQTT broker (you can simulate it if you don't have an available arduino).
3. Write a program capable of reading data from the broker, aggregating the values of the various sensors positioned in the environment and periodically sending these values back to the broker (determine the topic to be used).
4. Finally, propose a visualization (such as the evolution of values over time) of your data (for example, you can use the javascript dashboard `mqtt_index.html` which can be downloaded here →
   `https://github.com/truillet/upssitech/blob/master/Divers`