

# Le Protocole MQTT

Nicolas Vadkerti

November 21, 2019

## Abstract

[https://github.com/SlaynPool/CR\\_MQTT/](https://github.com/SlaynPool/CR_MQTT/)

Le but de ce document est de concentrer une journée de recherche à propos du protocole MQTT. Pour tous lecteurs, je ne garantis en rien l'exactitude des informations présent dans ce document.

## 1 Présentation MQTT

Comme vu en cours, MQTT est à la base développé pour permettre à des capteurs en plein milieu du désert puissent remonter des données via des liaisons satellites. L'idée de MQTT était donc d'avoir un protocole qui soit tolérant à la latence, économe en bande passante.

MQTT (Message Queuing Telemetry Transport) est donc très intéressant pour l'IOT. La légèreté du protocole permet l'implémenter sur quasiment n'importe quoi (L'exemple donné par oasis est un contrôleur avec 256KB de Ram)

Le protocole MQTT a pour architecture un mode client serveur. Les clients seront nos Objets, et le serveur (communément appelé Broker) aura pour rôle de recevoir les informations envoyées par chacun de nos objets ainsi que de s'assurer que les informations demandées par les objets seront bien envoyées.

### 1.1 Un client MQTT

Comme vu dans l'introduction, un client MQTT a pour objectif de soit envoyer des données, soit en recevoir. Pour cela, MQTT utilise un système de TOPIC. L'idée est donc d'avoir différents Topics sur le Broker, où l'on va pouvoir parler avec notre objet, écouter, ou faire les deux en même temps.

Pour mieux comprendre, on peut imaginer la situation suivante:

On dispose d'objets qui sont capables de mesurer la température et tous se connectent au même Broker MQTT. On décide d'une logique dans nos topics et ainsi:

- L'objet présent sur le toit de l'IUT écrira sur le topic `/capteurs/temperatures/toit`
- L'objet dans le hall, sur le topic `/capteurs/temperatures/hall`

Maintenant, imaginons un système de Climatisation qui utilise nos capteurs pour réguler la température. Pour qu'il puisse récupérer les informations de nos capteurs, il lui suffira de s'abonner à nos topics. Dès que le topic sera mis à jour, il recevra le nouveau message. Ici, on a que deux capteurs, et s'abonner

un a un à ceux ci est concevable. Mais imaginons que nous avons 2000 capteurs, cela les beaucoup moins. MQTT étant pensé dans une optique de Scalabilité, il est possible de :

- D'écrire sur des topics qui n'existe pas encore et il sera crée. (exemple: Je rajoute un capteur dans la BU, il pourra écrire sans configuration sur le broker dans /capteurs/temperatures/bu
- S'abonner a des groupes de topics. Si on a aussi des capteurs de pression par exemple qui écrive dans /capteurs/pression/lieux , on peut s'abonner a tous les topics qui parlent pression comme ceci : /capteurs/pression/+ ou si l'on veut tous les capteurs du hall, /capteurs/+ /hall

## 1.2 Le Broker

Important: Le rôle du Broker n'est pas de stocker les données. On pourra utiliser une base de données pour ceci. Cela n'est pas un problème. On peut considérer que le Broker ne sera pas une machine avec des capacités de calculs réduites contrairement a un Objets. Pour sauvegarder les messages, il nous suffira d'avoir un client MQTT abonné a tous les topics qui remplira une BDD

Le rôle du Broker est de s'assurer de recevoir les messages publiés, ainsi que les transmettre aux abonnés. Pour cela, la notion de QoS est importante pour déterminer son comportement.

- Le message est envoyé avec pour QoS voulu 0  
Quand le broker reçoit un message avec pour QoS 0, le mode par défaut, il ne doit renvoyer qu'une fois le message aux abonnés du topic. Il ne s'attend pas a recevoir d'accusé de réception. On parle aussi de Best-Effort. On peut donc utiliser ce mode quand notre lien entre le client et le Broker est stable (Par exemple un câble ethernet)
- Le message est envoyé avec pour QoS voulu 1  
Le broker va envoyer le message tant que il n'aura pas reçu d'accusé de réception. Evidemment, le problème est que l'application doit être programmée avec la possibilité de recevoir de fois le même message, et donc être capable de le détecter. Ce mode est privilégié lorsque nous voulons être sûr de recevoir au moins une fois notre message.
- Le message est envoyé avec pour QoS voulu 2 C'est le mode qui s'assure que le destinataire reçoit qu'une seule fois le message. L'expéditeur du message va attendre en premier lieu un accusé de réception, ce paquet lui indique qu'il peut détruire la donnée qu'il a envoyée. Il envoie donc un message au receveur pour confirmer la destruction du paquet et l'émetteur fait de même.

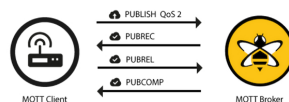


Figure 1: QoS 3

Evidement, le soucis est que ce niveau de QoS est plus couteux, en temps, bande passante, etc ... Ce niveau de QoS est donc a utilisé lors de cas très particulier, où l'on ne peut pas se permettre de a la fois perdre un paquet, ou le reexpédier.

Nous n'en avons pas encore parler, cependant, il y a une notion de connection dans le protocole MQTT. En effet, un client, qui est abonné a un topic, mais qui n'est pas connecté va quand meme recevoir les messages noter avec pour QoS 1 et 2. Le Broker va bufferiser les messages a destination du client le temps qu'il soit connecté.

## 2 Echange de trames Elementaires

Pour cette partie, nous allons analyser quelques trames communes du protocole MQTT. Voici tous d'abord

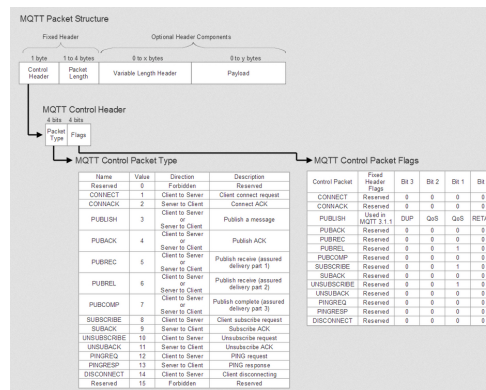


Figure 2: QoS 3

### 2.1 Une demande de connection d'un client à un broker

## 3 Sources

- [https://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=mqtt](https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt)
- <https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/>
- [http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html#\\_Toc442180841](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html#_Toc442180841)
- Hands-On Internet of Things with MQTT Author(s): Pulver, Tim Publisher: Packt Publishing Pub. Date: 2019