Le Protocole MQTT

Nicolas Vadkerti

November 21, 2019

Abstract

`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 garantie en rien l'exactitude des informations present dans ce docuement.

1 Presentation MQTT

Comme vu en cours, MQTT est a la base dévelloppé pour permettre a des capteurs en plein millieu du desert puissent remonter des données via des laisons sattellites. L'idée de MQTT etait donc d'avoir un protocole qui soit tolérant a la latence, économe en bande passante.
MQTT (Message Queuing Telemetry Transport) est donc très intéresant pour l'IOT. La légerté du protocole permet de l'implémenter sur quasiment n'importe quoi (L'exemple donné par oasis est un controlleur avec 256KB de Ram)
Le protocole MQTT a pour architecture un mode client serveur. Les clients seront nos Objets, et le serveur (communement appellé Broker) aura pour role de recevoir les informations envoyées par chacun de nos objets ainsi que de s'assurer que les informations demandé par les objets seront bien envoyés.

1.1 Un client MQTT

Comme vu dans l'introduction, un client MQTT a pour objectif de soit envoyé des données, soit en recevoir. Pour cela, MQTT utilise un systeme de TOPIC. L'idée est donc d'avoir differents Topics sur le Broker, où l'on va pouvoir parler avec notre objet, ecouter, ou faire les deux en meme temps.
Pour mieux comprendre, on peut imaginer la situation suivante:
On dispose d'objet qui sont capable de mesurer la temperatures et tous connecter au meme Broker MQTT. On decide d'une logique dans nos topics et ainsi:

- L'objet present sur le toit de l'IUT ecrira sur le topic /capteurs/temperatures/toit
- L'objet dans le hall, sur le topic /capteurs/temperatures/hall

Maintenant, imaginons un systeme de Climatisation qui utilise nos capteurs pour réguler la temperatures. Pour qu'il puisse récuperrer les informations de nos capteurs, il lui suffira de s'abonner a nos topics. Dès que le topic sera mis a 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 etant 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 ecrire 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 ecrive 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 role du Broker n'est pas de stocker les données. On pourra utilisé une base de donné 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 à un Objets. Pour sauvegardé les mesages, il nous suffira d'avoir un client MQTT abonné à tous les topics qui remplira une BDD
Le rôle du Broker est de s'assurer de recevoir les messages publiés, ainsi que les transmettres aux abonnés. Pour cela, la notion de QoS est important pour déterminer sont comportement.

- Le message est envoyé avec pour QoS voulu 0
  Quand le broker recoit un message avec pour QoS 0, le mode par default, il ne doit renvoyé qu'une fois le message aux abonnés du topic. Il ne s'attend pas a recevoir d'accusé de récéption. On parle aussi de Best-Effort. On peut donc utilisé ce mode quand notre lien entre le client et le Broker est stable (Par exemple un cable ethernet)
- Le message est envoyé avec pour QoS voulu 1
  Le broker va envoyé le message tant que il n'aura pas recu d'accusé de récéption. Evidement, le probleme est que l'application doit etre programmé avec la possibilité de recevoir de fois le meme message, et donc etre capable de le détécter. Ce mode est à priviligié lorsque nous voulons etre sur 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 recoit qu'une seul fois le message. L'expediteur du message va attendre en premier lieux un accusé de récéption, ce paquet lui indique qu'il peut detruire la donné qu'il a envoyé. Il envoie donc un message au receveur pour confirmé la destruction du paquet et l'emeteur fais de meme.

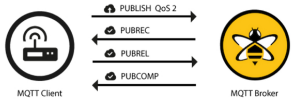


Figure 1: QoS 3

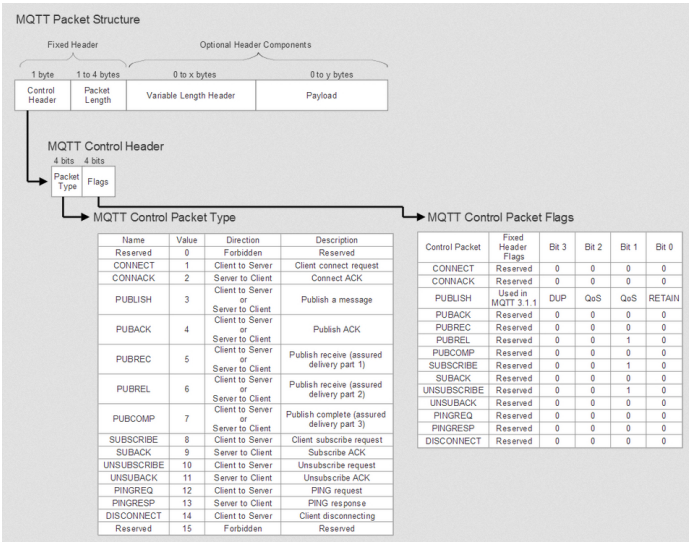
Evidement, le soucis est que ce niveau de QoS est plus couteux, en temps, bande passante, etc ... Ce niveau de QoS est donc a utilisé lors de cas très

particulier, où l'on ne peut pas ce permettre de a la fois perdre un paquet, ou le reexpedier.

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. Pour cela, le Broker va bufferiser les messages à 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. Le protocole MQTT est transporté grâce à TCP sur le port 1883. Voici tous d'abord la structure de toutes les trames MQTT



Name	Value	Direction	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client connect request
CONNACK	2	Server to Client	Connect ACK
PUBLISH	3	Client to Server or Server to Client	Publish a message
PUBACK	4	Client to Server or Server to Client	Publish ACK
PUBREC	5	Client to Server or Server to Client	Publish receive (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish receive (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Server to Client	Subscribe ACK
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe ACK
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server	Client disconnecting
Reserved	15	Forbidden	Reserved

Control Packet	Fixed Header Flags	Bit 3	Bit 2	Bit 1	Bit 0
CONNECT	Reserved	0	0	0	0
CONNACK	Reserved	0	0	0	0
PUBLISH	Used in MQTT 3.1.1	DUP	QoS	QoS	RETAIN
PUBACK	Reserved	0	0	0	0
PUBREC	Reserved	0	0	0	0
PUBREL	Reserved	0	0	1	0
PUBCOMP	Reserved	0	0	0	0
SUBSCRIBE	Reserved	0	0	1	0
SUBACK	Reserved	0	0	0	0
UNSUBSCRIBE	Reserved	0	0	1	0
UNSUBACK	Reserved	0	0	0	0
PINGREQ	Reserved	0	0	0	0
PINGRESP	Reserved	0	0	0	0
DISCONNECT	Reserved	0	0	0	0

Figure 2: Un paquet MQTT

Comme on peut le voir, seul les deux premieres parties d'un paquet sont obligatoire pour qu'une trame MQTT soit valide.

- La partie Control Header sur 1 octets qui a pour fonction d'annocer le but du paquet.

  - Par exemple, l'on veut se connecter a un broker il prendra la valeur 1, en binaire le champs type de paquet sera :
    —0001—
  - Le flags sera donc —0000— pour signaler que c'est une tentative de connection.

  Le contenu du paquet sera donc —0001 0000— pour une tentative de connection. Si on suit correctement le shema, le serveur va donc lui repondre par une trame CONNACK (acceptation de connection) —0010 0000—

- Le champs Taille du paquet permet d'annoncer la taille des champs suivants. Cette taille peut etre codé sur 4 octets maximun. L'utilité de faire ceci est simplement pour que dès que l'on recoit le paquet, on puisse savoir quelle taile on doit allouer de memoire pour recevoir le message.

Comme on l'a vu, une trame MQTT peut etre très courte, 2 octets, ce qui répond parfaitement à la problèmatique d'économie de bande passante. De plus, il est très léger en termes de calculs démandés. Il convient donc parfaitement pour les micro controleurs. A noter que toutes les trames reseaux sont echangé en claire sur le réseau. Cependant, il est possible d'ajouter une surcouche SSL/TLS pour sécuriser les echanges. Cela n'est pas forcement possible, notre controleur n'a pas forcement les capacités de calculs nécessaire pour gérer des clés etc.

3 Sources

- 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
- Hands-On Internet of Things with MQTT Author(s): Pulver, Tim Publisher: Packt Publishing Pub. Date: 2019
- https://www.oreilly.com/library/view/internet-of-things/9781788470599/6078c828-0f58-4cdc-8275-fadb0eae40f4.xhtml