Le protocole MQTT
Message Queuing Telemetry Transport

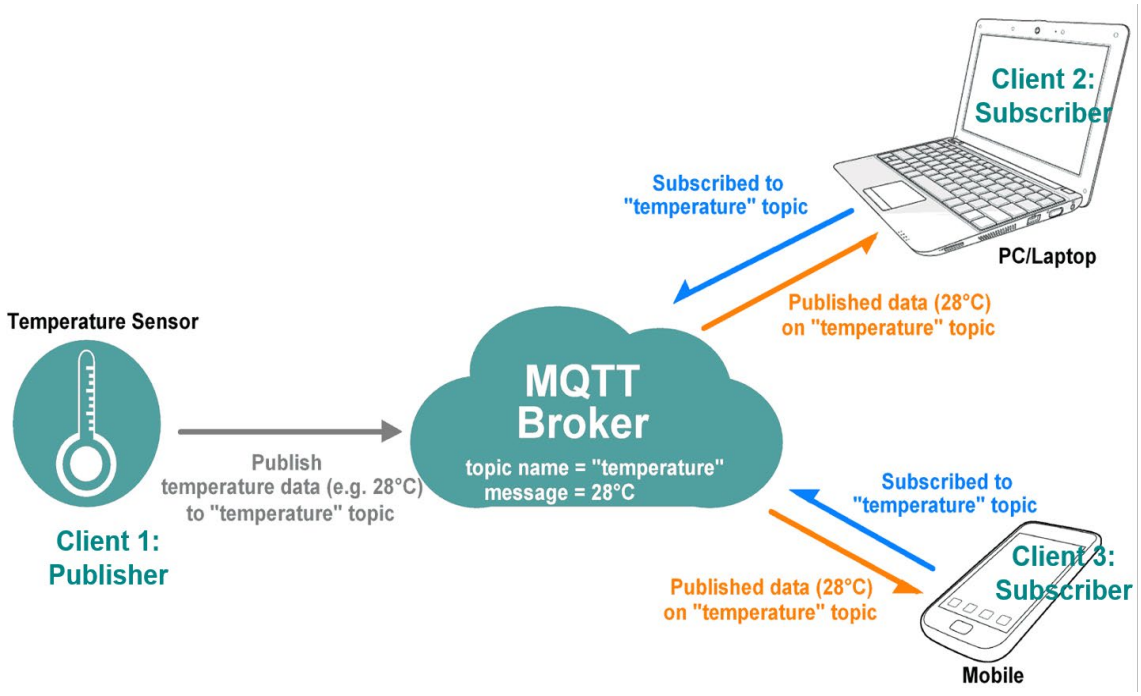
Le protocole MQTT à été développé par IBM et Eurotech en 1999 pour le contrôle d'un pipeline situé dans le desert. L'objectif été d'avoir un protocole fiable avec une faible bande passante et utilisant peu d'énergie et peu de ressources processeurs et mémoire. MQTT est très bien adapté au monde de l'IoT.

Communication client - broker

Une session MQTT est divisée en quatre étapes :

- Connexion,
- Authentification,
- Communication
- Fin de connexion.

Un client commence par créer une connexion TCP/IP vers le broker.
MQTT utilise le port standard 1883 et le port 8883 dans le cas d'une communication chiffrée utilisant SSL/TLS.



Sur cet exemple, on dispose de trois clients et d'un serveur (MQTT).

- Le client 1 est un capteur de température connecté qui envoie au serveur la valeur de la température (par exemple toutes les 30 secondes). On parle de "**publication**" (publisher).
  Sur une structure IoT il peut y avoir de nombreux capteurs donc plusieurs "Client 1". Pour s'y retrouver au niveau des valeurs envoyées, le client va définir un "Topic" c'est-à-dire un sujet de discussion unique avec le serveur.
  Dans notre exemple, le "**topic**" se nomme "température". Le serveur sauvegardera toutes les valeurs venant de "Client 1" dans un endroit réservé au sujet "température".
  Bien sûr, les autres clients qui doivent transmettre des valeurs devront utiliser un autre nom pour leurs topics. **Le nom du topic est unique**.

- Les clients2 et 3 souhaiteraient recevoir les données. Pour cela, ils doivent s'abonner à un topic auprès du serveur.

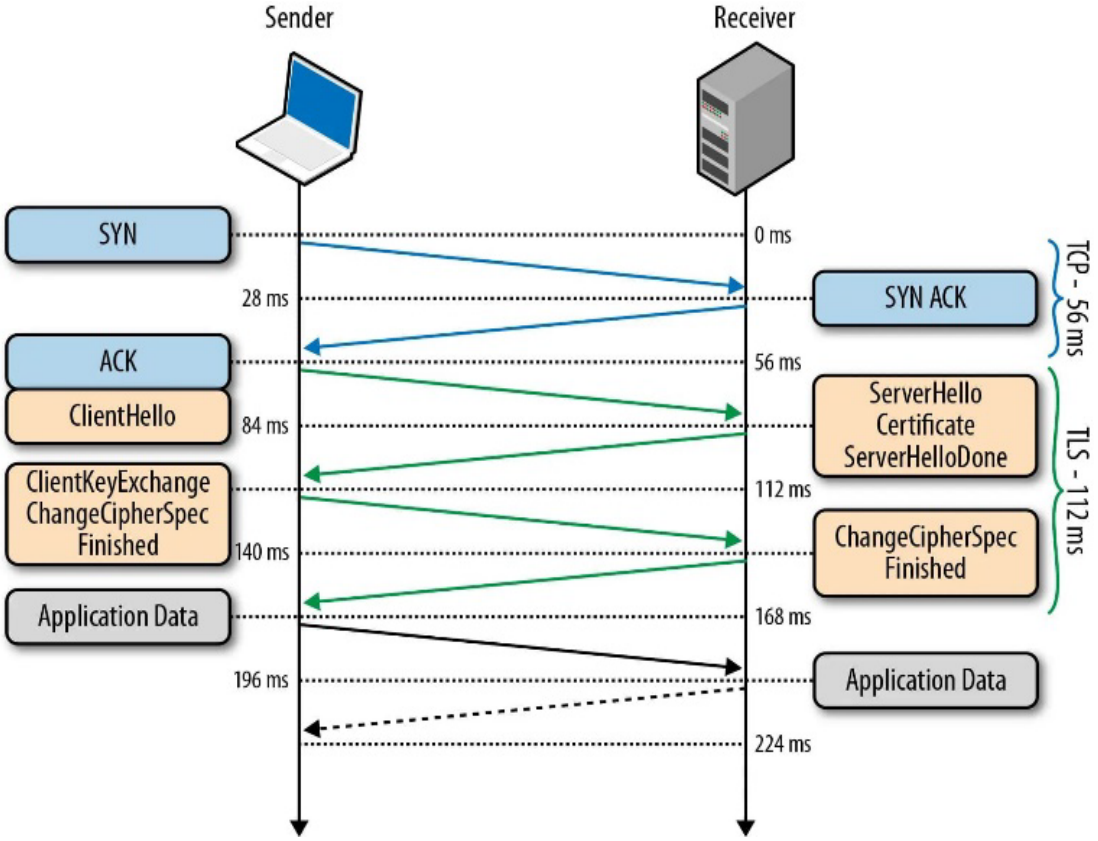
- Le serveur se nomme "**Broker**" (courtier), il gère tous les topics des clients de type "capteur". Quand une nouvelle valeur arrive au Broker alors il la transmet aux clients abonnés au topic concerné. Il joue le rôle d'intermédiaire, donc d'un courtier.

Communication chiffrée

Les données IoT échangées peuvent s'avérer très critiques, c'est pourquoi il est aussi possible de sécuriser les échanges à plusieurs niveaux :

- Transport en SSL et authentification par certificats TLS,
- Authentification par login/mot de passe.

Voici les différentes étapes d'une connexion TLS dites "**poignée de main**" (handbrake).
Cela consiste à se mettre d'accord sur les méthodes à utiliser pour l'échange de données (version du protocole, le type de chiffrement, …).



1. Etablissement d'une connexion TCP

2. **ClientHello**. Le client envoie au serveur : la version du protocole qu'il souhaite utiliser, les méthodes de chiffrement prises en charge, etc.

3. **ServerHelloDone**. Le serveur approuve la version du protocole, sélectionne la méthode de chiffrement dans la liste fournie. Puis il attache son certificat et envoie une réponse au client. Le serveur peut également demander un certificat client.

4. **ClientKeyExchange et ChangeCipherSpec**. La version du protocole et la méthode de chiffrement sont considérées comme approuvées à ce moment.
   Le client vérifie le certificat envoyé. Il génère une clé symétrique qu'il chiffre avec la clé publique du certificat du serveur. Enfin il l'envoie au serveur.

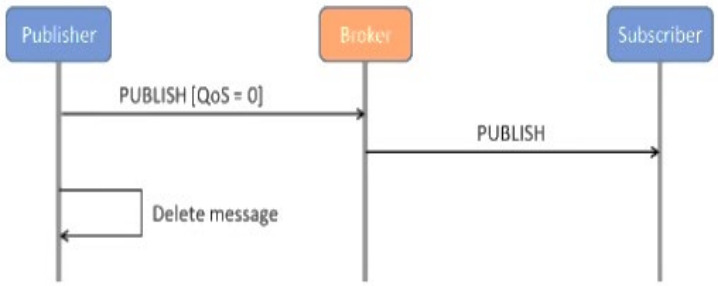
5. **ChangeCipherSpec**. Le serveur traite le message envoyé par le client, vérifie le MAC et envoie au client un message final "Terminé" sous forme chiffrée avec la clé symétrique.

6. Le client déchiffre le message reçu, vérifie le MAC et si tout va bien, la connexion est considérée comme établie et l'échange des données d'application commence. La session TLS s'effectue.

QoS – Qualité de service

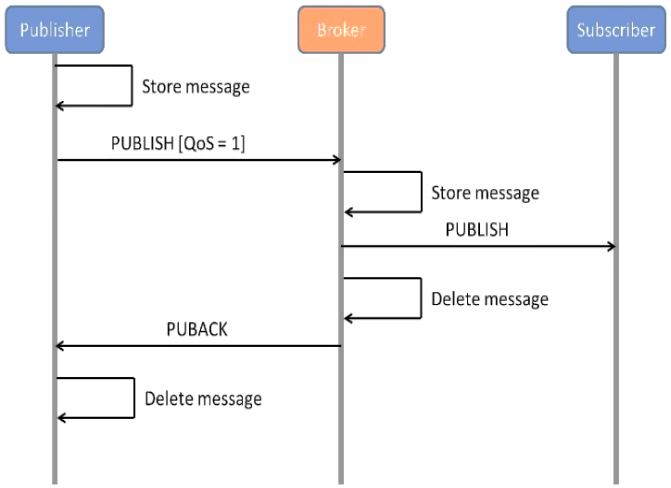
Le "**publisher**" a la possibilité de définir trois niveaux pour la qualité de service.

QoS niveau 0 (au plus une fois)



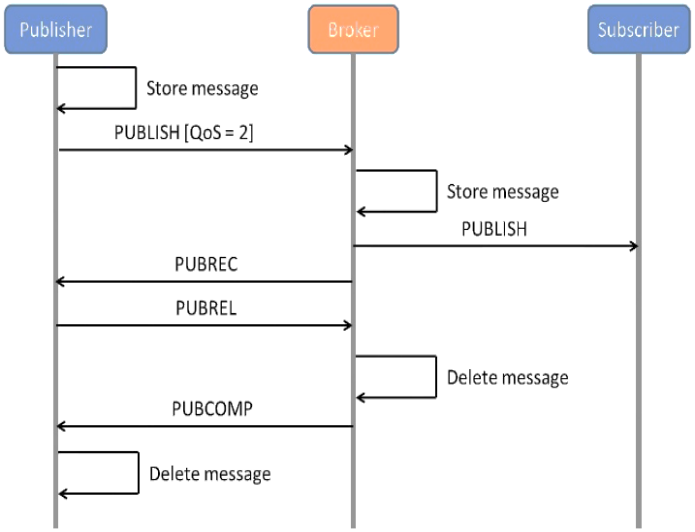
"**At most once**" : Au niveau 0, un message envoyé (`PUBLISH`) ne peut être reçu **qu'une seule fois**. L'expéditeur ne stocke pas le message pour les futures transmissions. Par conséquent, si un message est envoyé et que le destinataire n'est pas disponible au moment de l'envoi, la charge utile ne lui parviendra pas. De plus, **aucune confirmation de réception ne sera fournie**.

QoS niveau 1 (au moins une fois)



"**At least once**" : ce niveau garantit qu'un message envoyé (`PUBLISH`) sera reçu **au moins une fois**. Dans ce cas, si l'expéditeur ne reçoit pas d'accusé de réception (`PUBACK`) dans le délai prévu, le message peut être envoyé plusieurs fois jusqu'à recevoir une confirmation. La confirmation est envoyée une seule fois par le destinataire en tant que réponse au message d'origine. Par conséquent, si l'expéditeur d'origine d'un message est indisponible au moment de l'acheminement de la réponse, il ne recevra pas de confirmation de réception et continuera à envoyer le même message jusqu'à en recevoir une. Cependant, le destinataire ne comprendra pas qu'il s'agit d'un message dupliqué et l'interprétera comme une nouvelle information.

QoS niveau 2 (Exactement une fois)



"**Exactly once**" : ce niveau de service garantit que le message envoyé sera reçu précisément une fois. Il comprend :

PUBLISH : Le message est envoyé.
PUBREC : Réception confirmée par le destinataire.
PUBREL : L'expéditeur affirme la réception, à son tour, de la confirmation.
PUBCOMP : Une dernière confirmation du PUBREL envoyée par le destinataire.

Le flux QoS 2 requiert qu'une quantité plus importante de données soit véhiculée entre l'expéditeur et le destinataire, ce qui n'est pas vraiment approprié lorsqu'il s'agit de clients dont la capacité de mémoire est faible.

MQTT et le modèle OSI.

Layer	Dénomination	Protocole	Description
7	Application		Acquisition de données/contrôle processus
6	Présentation	Binaire, JSON, ASCII	Codage du message : binaire, JSON, ASCII …
5	Session	MQTT	Gestion du message MQTT
4	Transport	TCP	Transmission Control Protocol
3	Réseau	IP	Routage du message
2	Liaison	802.11	Ajout @MAC, gestion WiFi, …
1	Physique	802.11	Génération de l'ondes radio (Wi-Fi)

Structure d'un message MQTT (couche 5 - OSI)			
Control Header	**Packet Lenght**	**Variable length header**	**Payload**
Type de message	Longueur du message	Dépend du type de message	Données
1 octet	1 à 4 octets	0 à x octets	0 à 256 Mo

Control Header

On précise le type de message et certains indicateurs :

Bit 7-4 : Type de message (ex : 0x30 pour PUBLISH)
Bit 3 : Indicateur DUP (0 ou 1)
Bit 2-1 : QoS (0, 1 ou 2)
Bit 0 : Indicateur RETAIN (0 ou 1)

Type de message (bit 7 à 4)			
Nom	**Valeur**	**Direction du flux**	**Description**
Réservé	0	----	Réservé
CONNECT	1	Client vers serveur	Connexion demandée
CONNACK	2	Serveur vers client	Connexion validée (Accusé de reception)
PUBLISH	3	Client vers serveur ou serveur vers client	Publication d'un message
PUBACK	4	Client vers serveur ou serveur vers client	Si QoS = 1 - Publication validée (Accusé de reception)
PUBREC	5	Client vers serveur ou serveur vers client	Si QoS = 2 – Publication reçue
PUBREL	6	Client vers serveur ou serveur vers client	Si QoS = 2 – Publication effectuée
PUBCOMP	7	Client vers serveur ou serveur vers client	Si QoS = 2 – Publication terminé

DUP (0 ou 1) : L'indicateur DUP est utilisé pour indiquer si un message est un doublon d'un message précédemment envoyé. Si un expéditeur ne reçoit pas d'accusé de réception pour un message, il renverra le message avec l'indicateur DUP défini sur 1.

RETAIN (0 ou 1) : L'indicateur RETAIN est utilisé pour demander au courtier MQTT de stocker le message et de le renvoyer aux futurs abonnés qui correspondent au sujet du message.

Packet Lenght (Longueur restante - MSB) : Il indique combien d'octets restent à lire après ce champ, pour obtenir la totalité du paquet.
La longueur restante est encodée sur 1 à 4 octets, selon la valeur du MSB (Most Significant Byte) dans l'en-tête fixe.

- Si la valeur du MSB est comprise entre 0 et 127, la longueur restante est encodée sur 1 octet.
- Si la valeur du MSB est 128, la longueur restante est encodée sur 2 octets suivants.
- Si la valeur du MSB est entre 129 et 253, la longueur restante est encodée sur 3 octets suivants.
- Si la valeur du MSB est 254, la longueur restante est encodée sur 4 octets suivants.
- La valeur 255 est réservée et ne doit pas être utilisée.

En résumé, l'en-tête fixe d'un message MQTT a toujours une taille de 2 octets, mais la longueur restante, qui fait partie de l'en-tête variable, peut s'étendre sur plusieurs octets en fonction de la taille du message.

Variable length header (En-tête variable) **:** Présente dans certains types de messages, comme CONNECT, PUBLISH, etc. On peut retrouver : le nom du protocole, la version, le nom du topic, …

Payload (Charge utile)
Données du message.

Exemples

Exemple 1
Voici le code hexadécimal d'une trame MQTT :

10 12 00 04 4D 51 54 54 04 02 3C 00 07 63 6C 69 65 6E 74 31

Le premier chiffre "1" indique que l'on analyse un message CONNECT. Voici la structure typique :

- Octet fixe (1octet) : Type de message et indicateurs
- Longueur variable (1, 2, 3 ou 4 octets)
- Protocole Name Length (2 octets) : Précise la longueur du champ suivant : nom du protocole
- Protocole Name (4 octets)
- Protocole Level (1 octet) : version du protocole.
- Connexion Flags (1 octet)
- Keep Alive (2 octets) : vérifier que la connexion entre le client et le broker est toujours active en envoyant un ping toutes les x secondes (x = keep Alive).
- Client Identifier Length (2 octets) : Taille du champs suivant : Identifiant du client
- Payload (Client Identifier, Username, Password)

Analysez la trame.
Solution

1. **Octet fixe** :
2. **Longueur variable** :
3. **Protocole Name Length** :
4. **Protocole Name** :
5. **Protocole Level** :
6. **Connexion Flags** :
7. **Keep Alive** :
8. **Client Identifier Length** :
9. **Client Identifier** :

Exemple 2
Proposez la trame correspondant à un message MQTT PUBLISH, sans DUP ni RETAIN, et un sujet "test/topic" avec la charge utile "Hello, World!".
Code hexadécimal :