



# MQ Télémétrie Transport

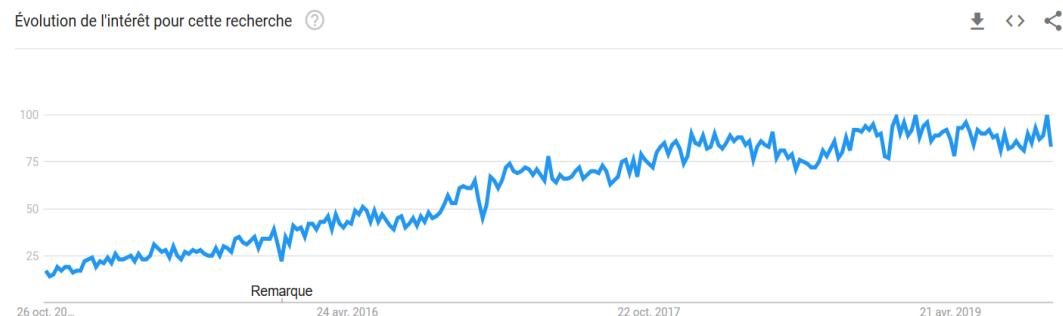
# Introduction:

*MQTT est un protocole M2M open source . C'est un protocole de messagerie basé sur le publish/subscribe à la fois extrêmement simple et léger idéal pour l'Internet des objets.*

*Il a été inventé par Andy Stanford-Clark (IBM) et Arlen Nipper (Arcom, maintenant Cirrus Link) en 1999 avec les objectifs suivants:*

- 1. Simple à mettre en œuvre.*
- 2. Fournir une qualité de service de livraison de données.*
- 3. Léger et faible consommation de la bande passante.*

*Le protocole est standardisé par l'OASIS en 2014 dans sa version 3.1.1. Il devient un standard ISO en 2016.*



# Fonctionnement

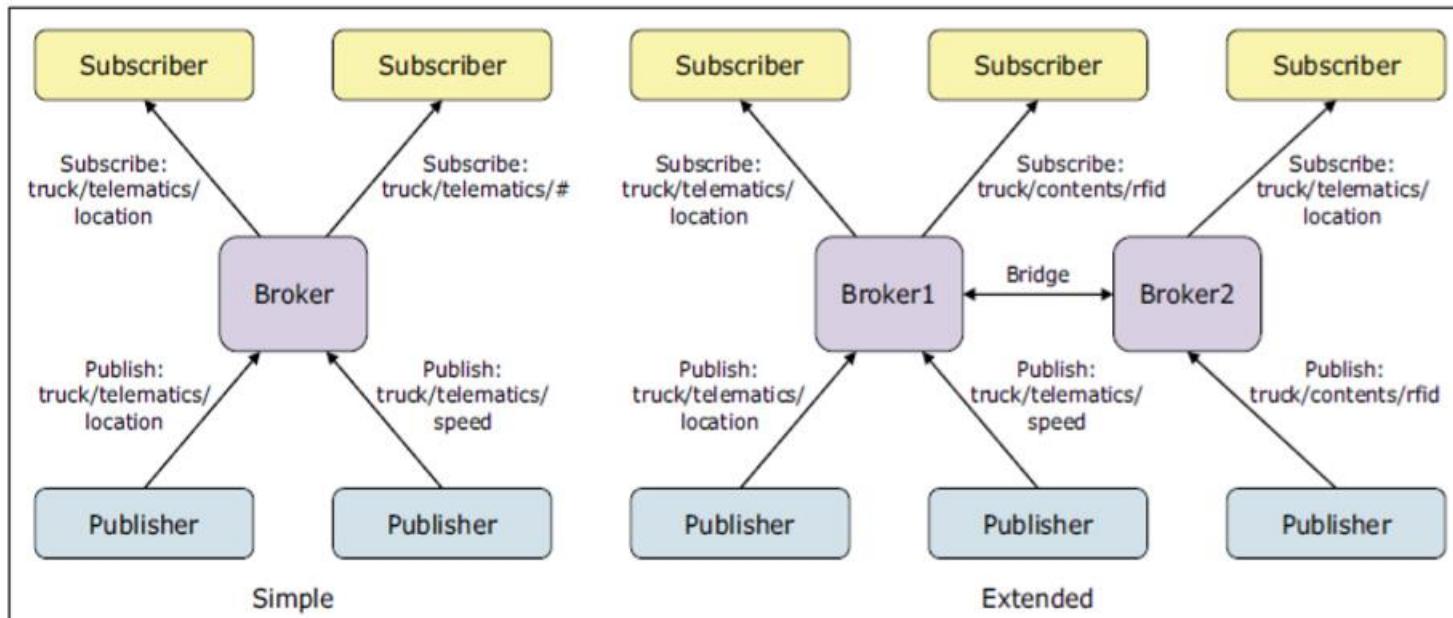
*MQTT est un **protocole de publication et de souscription** de message. Les **clients** ne communiquent pas **directement** entre eux, ils publient des messages sur un **broker**, les messages sont composés d'un **contenu** et d'un **Topic**. Le broker garde en mémoire le **dernier message pour chaque sujet**. Les clients qui sont intéressés par les messages d'un **Topic** peuvent les récupérer en se connectant au broker. L'avantage de cette solution:*

- *Permet à plusieurs clients de communiquer même s'ils ne sont jamais connectés en même temps au broker.*

# Fonctionnement: Topologies

*Le standard ne définit pas de topologie particulière d'utilisation, la plus simple étant celle avec un seul broker et un certain nombre de clients.*

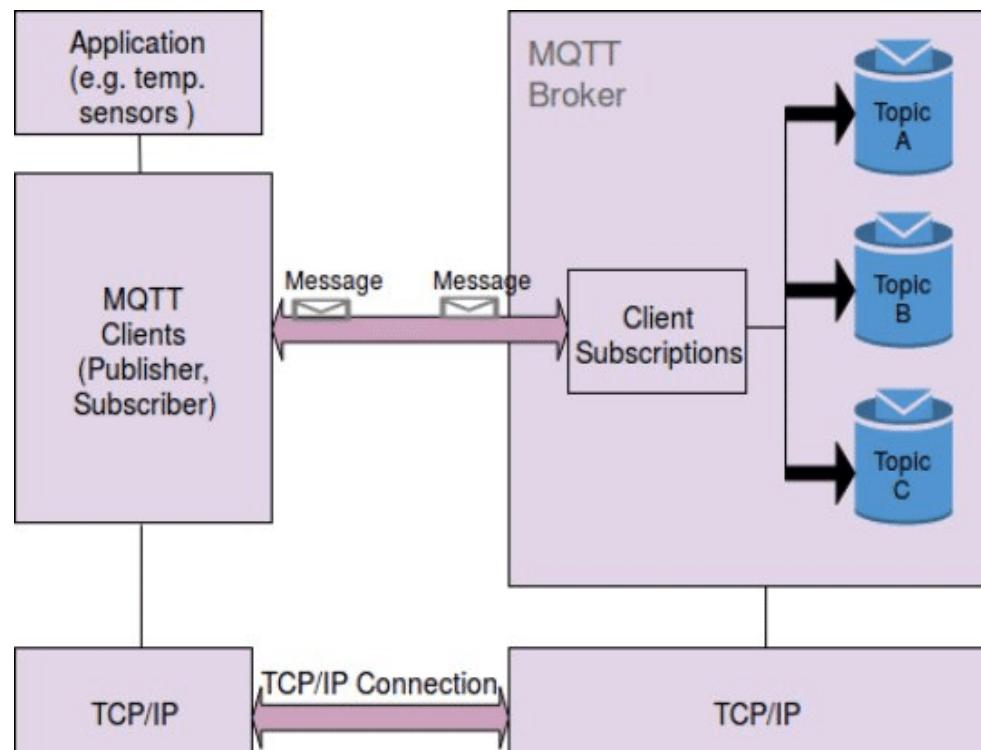
*D'autres organisations restent évidemment possible, avec en particulier les systèmes à base de bridge dans lesquels plusieurs brokers sont utilisés, chacun étant client des autres brokers.*



# Fonctionnement: modelé TCP/IP

*MQTT est un protocole qui repose sur TCP. L'encapsulation des trames sur le réseau est donc la suivante :*

**MQTT se sert de connexions persistantes entre les clients et le broker, et exploite pour cela les protocoles de transport garantissant un bon niveau de fiabilité :TCP**



*le port TCP utilisé pour le protocole MQTT (non chiffré) est le 1883*

- > Ethernet II, Src: Raspberry\_f3:03:41 (b8:27:eb:f3:03:41), Dst: Dell\_7d:b5:7e (10:65:30:7d:b5:7e)
- > Internet Protocol Version 4, Src: 192.168.0.200, Dst: 192.168.0.11
- > Transmission Control Protocol, Src Port: 1883, Dst Port: 62454, Seq: 15, Ack: 5, Len: 4
- > MQ Telemetry Transport Protocol, Publish Release

# Fonctionnement: paquet MQTT Entête fixe

*Tous les packets MQTT commencent par un format d'entête fixe, contenant le code de commande MQTT (sur 4 bits), un drapeau de 4 bits et la remaining length.*

| Bit    | 7                        | 6 | 5 | 4 | 3   | 2 | 1 | 0 |
|--------|--------------------------|---|---|---|---|---|---|---|
| byte 1 | MQTT Control Packet type |   |   |   | Flags specific to each MQTT Control Packet type |   |   |   |
| byte 2 |                          |   |   |   | Remaining Length                                |   |   |   |

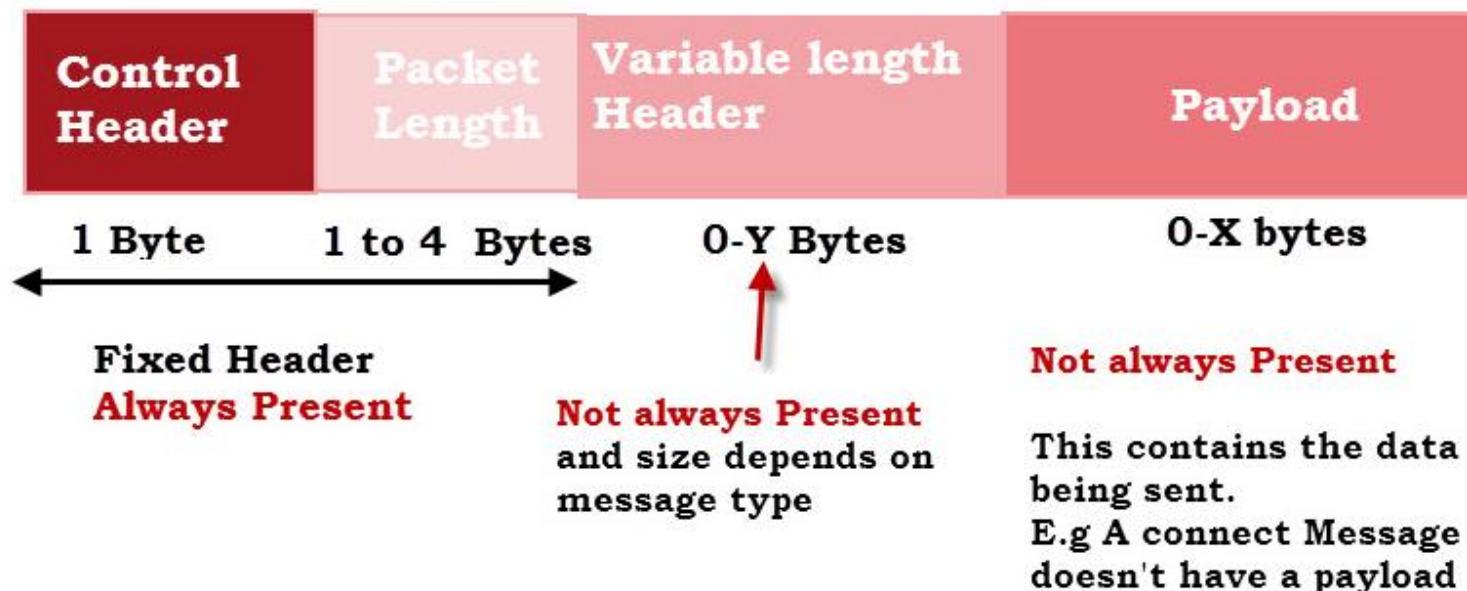
**RemainingLength** : longueur en octets de la trame MQTT restante.  
Longueur maxi d'un paquet MQTT et de 256 Mo. Donc sa taille peut varier de 1 à 4 octets.

| Name        | Value | Direction of flow                 | Description                                |
|-------------|-------|-----------------------------------|--|
| Reserved    | 0     | Forbidden                         | Reserved                                   |
| CONNECT     | 1     | Client to Server                  | Client request to connect to Server        |
| CONNACK     | 2     | Server to Client                  | Connect acknowledgment                     |
| PUBLISH     | 3     | Client to Server/Server to Client | Publish message                            |
| PUBACK      | 4     | Client to Server/Server to Client | Publish acknowledgment                     |
| PUBREC      | 5     | Client to Server/Server to Client | Publish received (assured delivery part 1) |
| PUBREL      | 6     | Client to Server/Server to Client | Publish release (assured delivery part 2)  |
| PUBCOMP     | 7     | Client to Server/Server to Client | Publish complete (assured delivery part 3) |
| SUBSCRIBE   | 8     | Client to Server                  | Client subscribe request                   |
| SUBACK      | 9     | Server to Client                  | Subscribe acknowledgment                   |
| UNSUBSCRIBE | 10    | Client to Server                  | Unsubscribe request                        |
| UNSUBACK    | 11    | Server to Client                  | Unsubscribe acknowledgment                 |
| PINGREQ     | 12    | Client to Server                  | PING request                               |
| PINGRESP    | 13    | Server to Client                  | PING response                              |
| DISCONNECT  | 14    | Client to Server                  | Client is 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 <sup>1</sup> | QoS <sup>2</sup> | QoS <sup>2</sup> | RETAIN <sup>3</sup> |
| 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                   |

# Fonctionnement: paquet MQTT complet

Tous les paquets MQTT consiste en un en-tête fixe de 2 à 5 octets (toujours présent) + en-tête de variable (pas toujours présent) + payload (pas toujours présente).



MQTT Standard Packet Structure

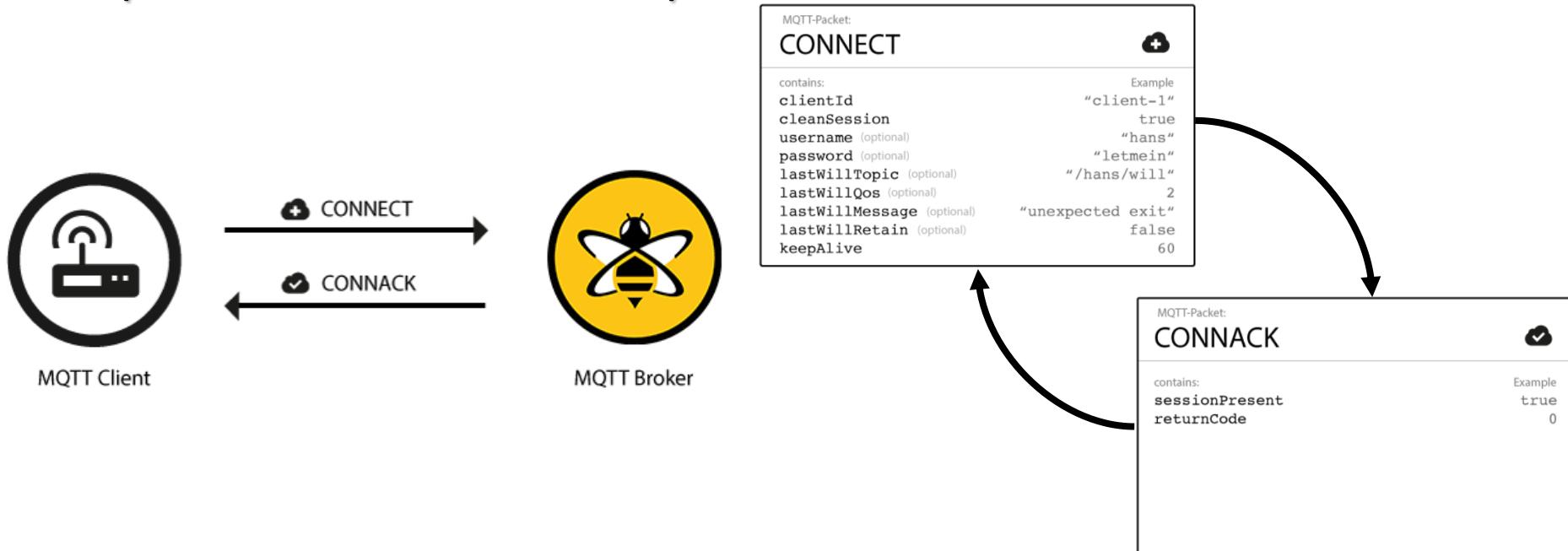
# Fonctionnement: connexion 1/6

*Pour établir une connexion, le client envoie un message **CONNECT** au courtier.*

*Le courtier répond avec un message **CONNACK** et un code d'état.*

*Une fois la connexion établie, le courtier la maintient ouverte:*

- jusqu'à ce que le client envoie une commande de déconnexion*
- que la connexion soit rompue.*



# Fonctionnement: connexion 2/6

MQTT-Packet:

## CONNECT

contains:

- clientId**
- cleanSession**
- username** (optional)
- password** (optional)
- lastWillTopic** (optional)
- lastWillQos** (optional)
- lastWillMessage** (optional)
- lastWillRetain** (optional)
- keepAlive**



Example

|                   |      |
|-------------------|------|
| "client-1"        | true |
| "hans"            |      |
| "letmein"         |      |
| "/hans/will"      | 2    |
| "unexpected exit" |      |
| false             | 60   |

**clientId:** identifie chaque client MQTT cet ID doit être unique par client et par courtier. Dans MQTT 3.1.1 (le standard actuel), vous pouvez envoyer un ClientId vide, si vous n'avez pas besoin qu'un état soit détenu par le courtierSession propre

**optionnel:** MQTT peut envoyer **username** et **password** pour l'autorisation du client. Attention il faut les cryptées ou hachées (par implémentation ou TLS).

**LastWillMessage** fait partie de la fonction de testament (**LWT**) de MQTT. Ce message informe les autres clients lorsqu'un client se déconnecte de manière incohérente. Lorsqu'un client se connecte, il peut fournir au courtier une dernière volonté sous la forme d'un message MQTT et d'une rubrique dans le message CONNECT. Si le client se déconnecte anormalement, le courtier envoie le message **LWT** au nom du client.

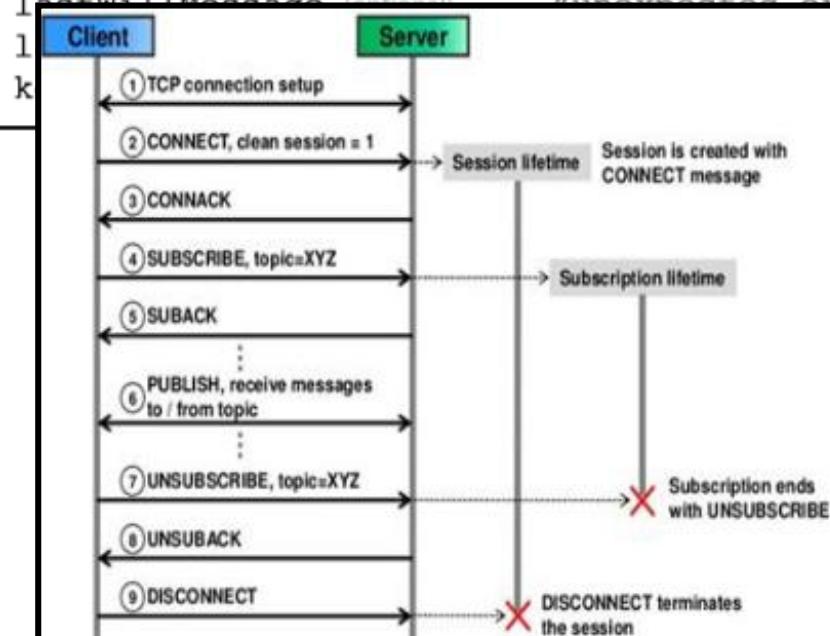
# Fonctionnement: connexion 3/6

MQTT-Packet:

## CONNECT

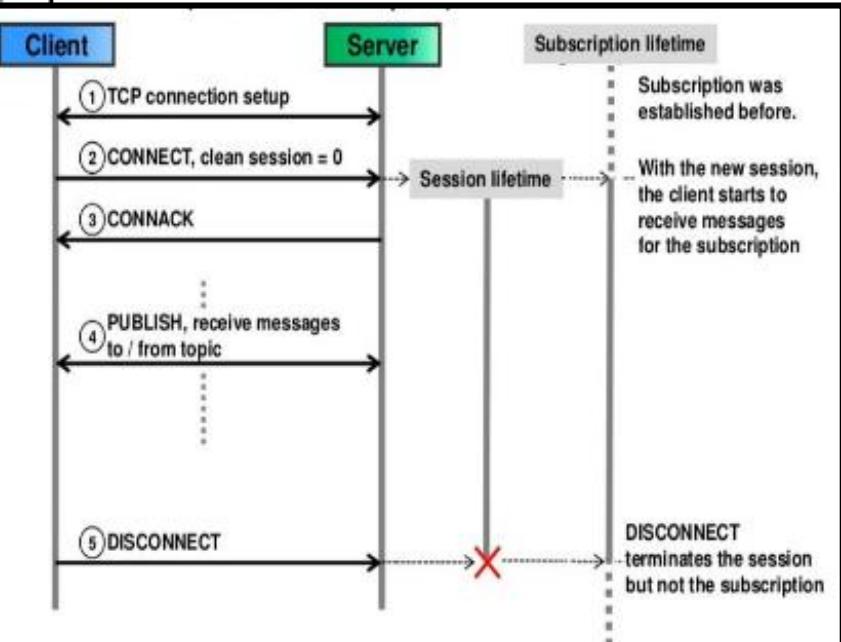
contains:  
**clientId**  
**cleanSession**  
**username** (optional)  
**password** (optional)  
**lastWillTopic** (optional)  
**lastWillQos** (optional)

Example  
"client-1"  
true  
"hans"  
"letmein"  
"/hans/will"  
2



**cleanSession** établir une session persistante ou non:

- Persistante (**cleanSession=false**), le courtier stocke tous les abonnements du client et tous les messages manqués du client abonné si une qualité de service (QoS) et de niveau 1 ou 2.
- Non persistante (**cleanSession=true** ), le courtier ne stocke rien pour le client et purge toutes les informations de toute session persistante précédente.



# Fonctionnement: connexion 4/6

MQTT-Packet:

## CONNECT

contains:  
**clientId**  
**cleanSession**  
**username** (optional)  
**password** (optional)  
**lastWillTopic** (optional)  
**lastWillQos** (optional)  
**lastWillMessage** (optional)  
**lastWillRetain** (optional)  
**keepAlive**

Example  
"client-1"  
true  
"hans"  
"letmein"  
"/hans/will"  
2  
"unexpected exit"  
false  
60

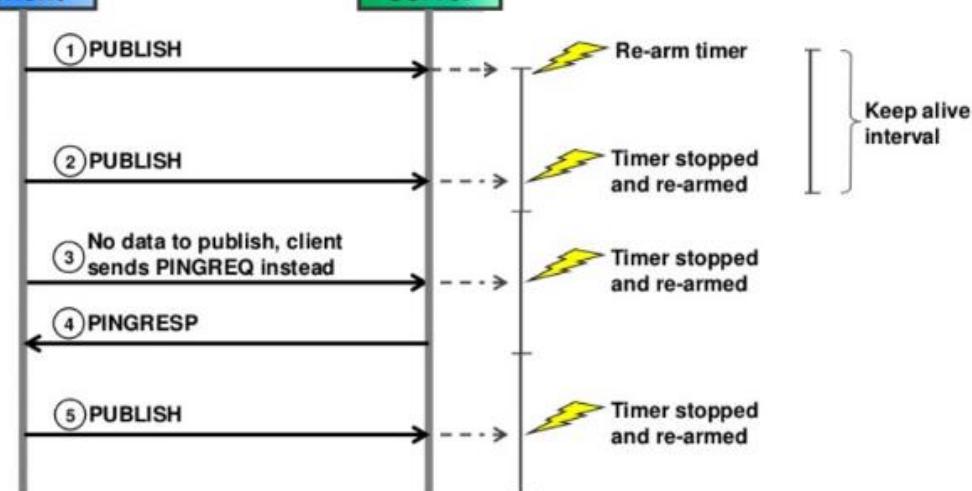


Client

Server

**keepAlive:** C'est un intervalle de temps en secondes que le client spécifie et communique au courtier lors de l'établissement de la connexion. Cet intervalle définit la plus longue période de temps que le courtier et le client peuvent supporter sans envoyer de message. Le client s'engage à envoyer des messages de requête **PING** au courtier. Le courtier répond avec une réponse **PING**.

Cette méthode permet aux deux parties de déterminer si l'autre est toujours disponible.



# Fonctionnement: connexion 5/6

MQTT-Packet:

**CONNACK**



Example  
true  
0

contains:

sessionPresent  
returnCode

**sessionPresent** : *indique au client si le courtier a déjà une session persistante disponible précédentes avec le client.*

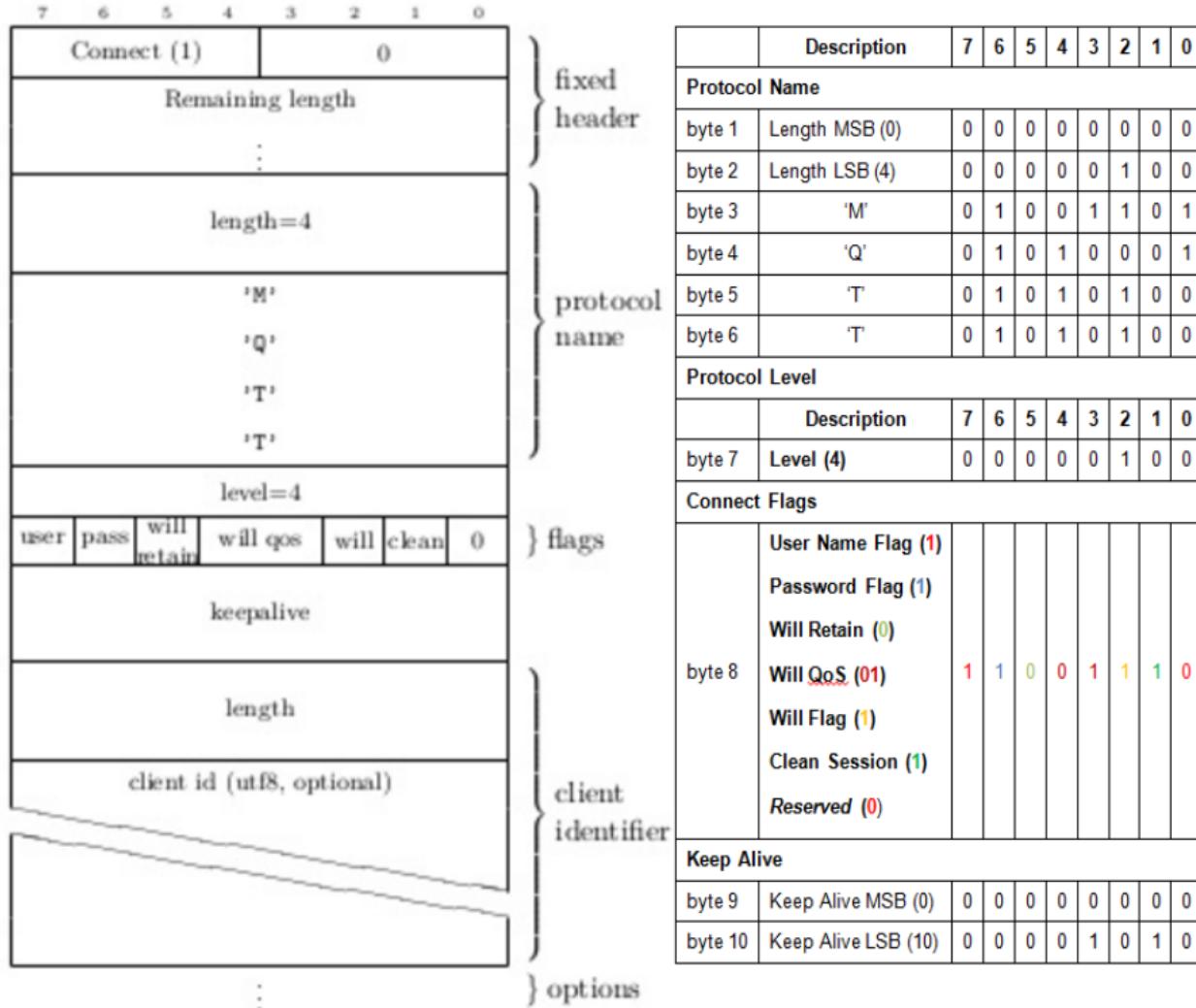
*Cet indicateur a été ajouté dans MQTT 3.1.1 pour aider les clients à déterminer s'ils doivent s'abonner à des sujets ou s'ils sont toujours stockés dans une session persistante.*

**returnCode**: *Le deuxième drapeau du message CONNACK est le drapeau d'accusé de réception de connexion. Cet indicateur contient un code de retour indiquant au client si la tentative de connexion a abouti.*

| Return Code | Return Code Response                              |
|-------------|---|
| 0           | Connection accepted                               |
| 1           | Connection refused, unacceptable protocol version |
| 2           | Connection refused, identifier rejected           |
| 3           | Connection refused, server unavailable            |
| 4           | Connection refused, bad user name or password     |
| 5           | Connection refused, not authorized                |

# Fonctionnement: connexion 6/6

## Tous les Entête de CONNECT (MQTT 3.1.1)



**Protocol Name** : nom du protocole (préfixé par sa taille). « MQTT »

**Protocol Level** : version du protocole. Pour la version 3.1.1 la valeur est de **0x04**.

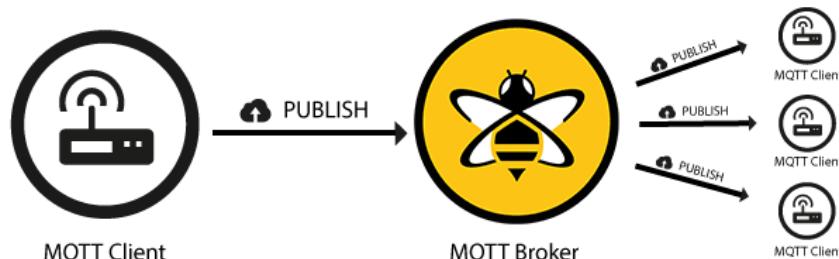
**Connect Flags...**

**Keep Alive...**

# Fonctionnement: publier 1/4

*Un client MQTT peut publier des messages dès qu'il est connecté à un courtier.*

- *MQTT utilise le filtrage par thème des messages sur le courtier. Chaque message doit contenir un sujet.*
- *En général, chaque message a un **payload** contenant les données à transmettre sous forme d'octets.*
- *Le client expéditeur (l'éditeur) décide s'il souhaite envoyer des données binaires, des données texte ou même du code XML ou JSON*



# Fonctionnement: publier 2/4

MQTT-Packet:

**PUBLISH**

contains:

**packetId** (always 0 for qos 0)

**topicName**

**qos**

**retainFlag**

**payload**

**dupFlag**



Example

4314

"topic/1"

1

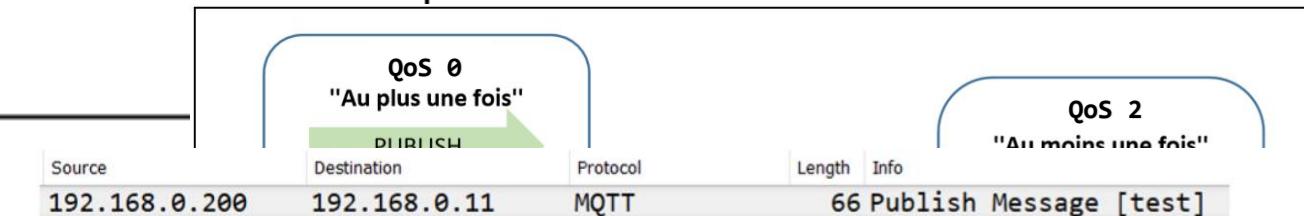
false

"temperature:32.5"

false

**packetId**: identifie de manière unique un message lorsqu'il circule entre le client et le courtier. L'identificateur de paquet n'est pertinent que pour les niveaux de qualité de service supérieurs à zéro.

**topicName** c'est le nom du topic c'est une simple chaîne structurée hiérarchiquement avec des barres obliques comme séparateurs. Par exemple, «astier/salle\_1\_bts\_snlr/temperature».



**qos**: Cette valeur indique le niveau de qualité de service (QoS) du message. Il existe trois niveaux: 0,1 et 2. Le niveau de service détermine le type de garantie qu'un message a pour atteindre le destinataire souhaité (client ou courtier).

| Source        | Destination   | Protocol | Length | Info                          |
|---------------|---------------|----------|--------|-------------------------------|
| 192.168.0.200 | 192.168.0.11  | MQTT     | 68     | Publish Message (id=4) [test] |
| 192.168.0.11  | 192.168.0.200 | MQTT     | 58     | Publish Ack (id=4)            |

| Source        | Destination   | Protocol | Length | Info                          |
|---------------|---------------|----------|--------|-------------------------------|
| 192.168.0.200 | 192.168.0.11  | MQTT     | 68     | Publish Message (id=5) [test] |
| 192.168.0.11  | 192.168.0.200 | MQTT     | 58     | Publish Received (id=5)       |
| 192.168.0.200 | 192.168.0.11  | MQTT     | 60     | Publish Release (id=5)        |
| 192.168.0.11  | 192.168.0.200 | MQTT     | 58     | Publish Complete (id=5)       |

# Fonctionnement: publier 3/4

MQTT-Packet:

**PUBLISH**



contains:

**packetId** (always 0 for qos 0)

**topicName**

**qos**

**retainFlag**

**payload**

**dupFlag**

Example

4314

"topic/1"

1

false

"temperature:32.5"

false

**payload** : c'est le contenu réel du message.

On peut théoriquement envoyer n'importe quel type d'informations binaire, texte, image..

**dupFlag:** indicateur de duplication.

Si l'indicateur DUP est réglé sur 0, cela indique que c'est la première fois que le Client ou le Broker a tenté d'envoyer ce

**retainFlag** : se drapeau définit si le message est enregistré par le courtier en tant que dernière valeur correcte connue pour un sujet spécifié. Lorsqu'un nouveau client s'abonne à un sujet, il reçoit le dernier message retenu sur ce sujet.

# Fonctionnement: publier 4/4

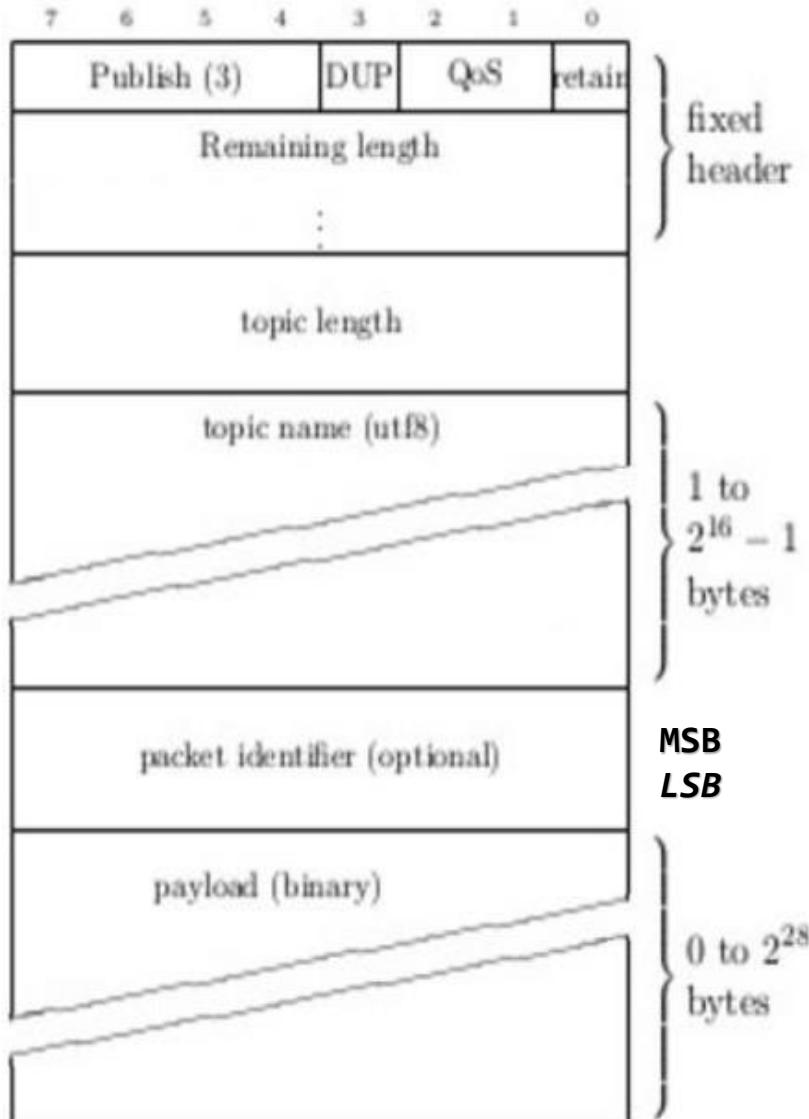
MQTT-Packet:

## PUBLISH



contains:  
**packetId** (always 0 for qos 0)  
**topicName**  
**qos**  
**retainFlag**  
**payload**  
**dupFlag**

Example  
4314  
"topic/1"  
1  
false  
"temperature:32.5"  
false

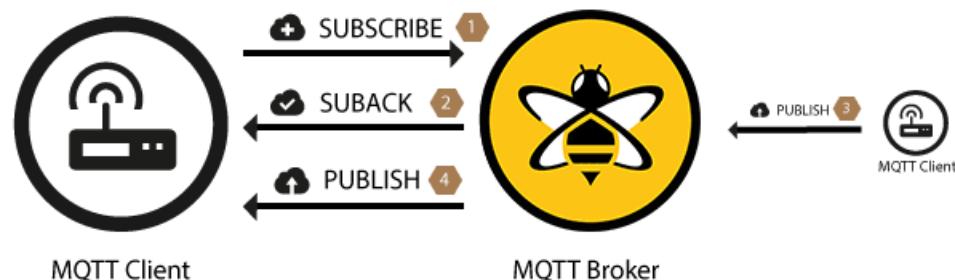


# Fonctionnement: abonnement 1/3

Pour recevoir des messages d'un topic, le client envoie un message **SUBSCRIBE** au courtier MQTT. Ce message d'abonnement est très simple, il contient:

- un identifiant de paquet unique
- une liste d'abonnements.

Lorsqu'un client envoie avec succès le message **SUBSCRIBE** et reçoit le message **SUBACK**, il récupère chaque message publié correspondant à un sujet dans les abonnements contenus dans le message **SUBSCRIBE**.





# Fonctionnement: abonnement 2/3

| MQTT-Packet:                        |   |
|-------------------------------------|---|
| <b>SUBSCRIBE</b>                    |  |
| contains:                           | Example   |
| <b>packetId</b>                     | 4312  |
| <b>qos1</b> } (list of topic + qos) | 1   |
| <b>topic1</b>                       | "topic/1"   |
| <b>qos2</b> }                       | 0   |
| <b>topic2</b>                       | "topic/2"   |
| ...                                 | ...   |

**packetId** : identifie de manière unique un message lorsqu'il circule entre le client et le broker.

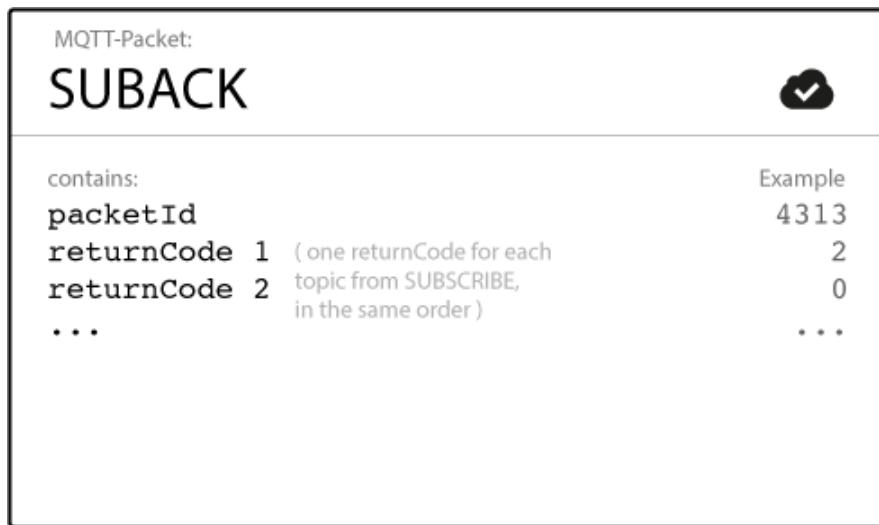
**Qosx/topicx**: un message SUBSCRIBE peut contenir plusieurs abonnements pour un client. Chaque abonnement est composé d'un sujet et d'un niveau de QoS. Le sujet du message d'abonnement peut contenir des caractères jokers '+', '#' permettant de s'abonner à un motif de topic plutôt qu'à un topic spécifique. Si des abonnements se chevauchent pour un client, le courtier transmet le message qui a le niveau de qualité de service le plus élevé pour ce sujet.

## jokers :

+ : sujet unique. Par exemple : a/+c sera abonné à a/b/c et a/x/c.

# : multi-sujets. Par exemple : a/# sera abonné à a/<tout>

# Fonctionnement: abonnement 3/3



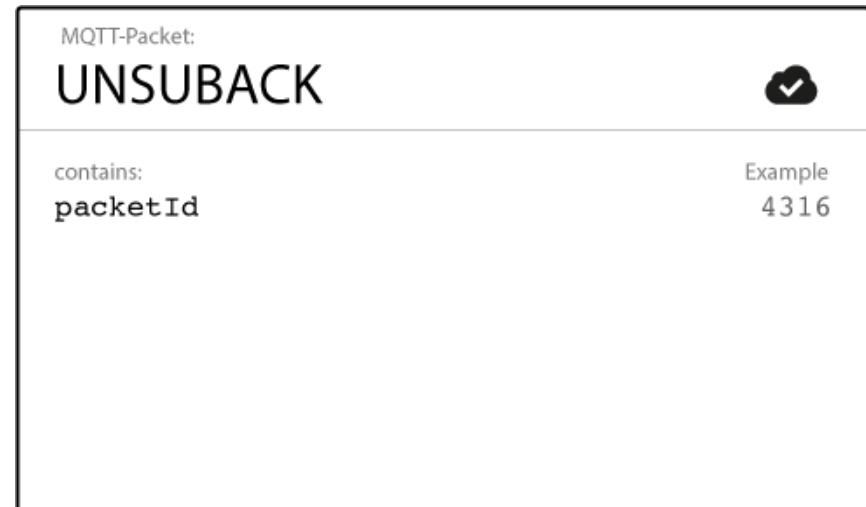
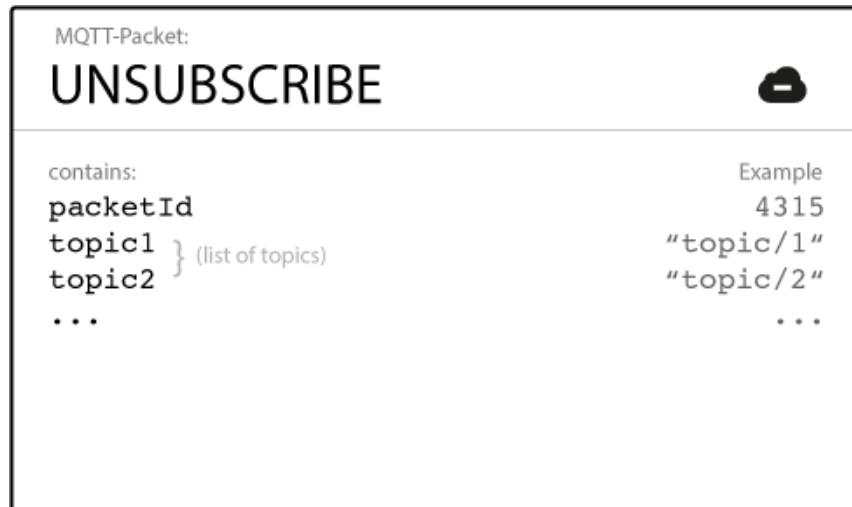
**packetId** l'identifiant de paquet du message Subscribe original (pour identifier clairement le message).

**returnCode x**: le courtier envoie un code de retour pour chaque paire sujet / QoS qu'il reçoit dans le message SUBSCRIBE.

| Code de retour | Réponse du code de retour |
|----------------|---------------------------|
| 0              | Succès - QoS maximum 0    |
| 1              | Succès - QoS maximum 1    |
| 2              | Succès - QoS 2 maximum    |
| 128            | Échec                     |

Lorsqu'un client envoie avec succès le message SUBSCRIBE et reçoit le message SUBACK, il reçoit chaque message publié correspondant à un sujet dans les abonnements contenus dans le message SUBSCRIBE

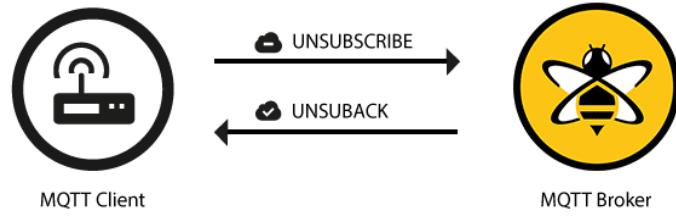
# Fonctionnement: désabonnement 1/3



**packetId** : identifie de manière unique un message lorsqu'il circule entre le client et le broker.

**topic X**: Il suffit d'envoyer le sujet (sans QoS). Le courtier désabonne le sujet, quel que soit le niveau de QoS auquel il était abonné à l'origine..

**packetId** : l'identifiant de paquet du message Unsubscribe original (pour identifier clairement le message)



Lorsqu'un client envoie avec succès le message **UNSUBSCRIBE** et reçoit le message **UNSUBACK**, le client ne reçoit plus les messages liés aux topics **UNSUBSCRIBE**.

