# Protocole de messagerie MQTT

## I. Introduction à MQTT

### A. Présentation de MQTT

Le protocole MQTT (*Message Queing Telemetry Transport*) est basé sur le principe de la publication (***publish***) de messages et l'abonnement (***subscribe***) à des sujets (***topic***). Plusieurs clients se connectent à un courtier (***broker***) et s'abonnent à des sujets. Les clients se connectent également au courtier et publient des messages sur des sujets.
Il a été conçu en 1999 pour être utilisé sur les satellites et, en tant que tel, est très léger, avec des exigences de faible bande passante. Depuis novembre 2014, la version 3.1.1 de MQTT est devenue un standard international pour la communication entre machines (M2M - *Machine To Machine*) et les objets connectés (IoT - *Internet Of Things*). Il est soutenu par la fondation OASIS qui regroupent des grandes entreprises de l'industrie informatique et des télécommunications.

### B. Publier / S'abonner (*Publish / Subscribe*)

Le modèle de publication/abonnement (souvent appelé *pub-sub*) est au cœur du MQTT. Il est basé sur un courtier de messages, avec d'autres nœuds disposés autour du courtier dans une topologie en étoile. Il s'agit d'un modèle très différent de l'approche client-serveur standard.
Les clients peuvent publier ou s'abonner à des sujets particuliers qui sont un peu comme des sujets de message. Ils sont utilisés par le courtier pour décider qui recevra un message. Les sujets de MQTT ont une syntaxe particulière. Ils sont disposés dans une hiérarchie à l'aide du caractère slash (/) en tant que séparateur. Ainsi, un capteur de température dans votre cuisine pourrait publier sur un sujet comme « capteurs/temperature/maison/cuisine ».

Voyons un exemple (figure 1): Imaginez un service météorologique qui dispose d'un réseau de capteurs de température connectés à Internet partout dans le monde. Tous ces capteurs entretiennent une connexion avec un courtier et toutes les dix minutes, ils rapportent la température actuelle. Ils publient sur un sujet particulier en fonction de leur emplacement dans le format suivant : «capteurs/temperature/{pays}/{ville}/{nom de la rue} »

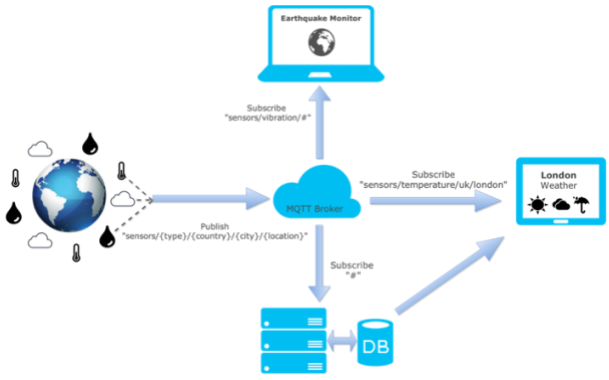


*Figure 1 : Principe du protocole MQTT*
(https://zoetrope.io/tech-blog/brief-practical-introduction-mqtt-protocol-and-its-application-iot)

Donc, un capteur de température sur la Rue de la Chebarde à Aubière publierait dans «capteurs/temperature/fr/aubiere/rue_chebarde» avec un message contenant la température actuelle.
Le service météorologique veut conserver une base de données à jour sur les températures historiques afin de créer un service de base de données pouvant s'abonner à un sujet MQTT. Le service de base de données sera alors notifié lorsqu'une nouvelle température sera reçue.

Il y a cependant un problème : le service de base de données doit connaître tous les capteurs de température du monde entier et il serait énorme de s'abonner à des sujets individuels pour chaque capteur. Heureusement, MQTT a une solution : les caractères génériques (*Wildcards*).

## C. Caractères génériques (Wildcards)

Il existe deux caractères génériques (*wildcards*) qui peuvent être utilisés dans MQTT, **+** et **#** :

- **+** correspond à tous les sujets à un niveau hiérarchique unique ;
- **#** correspond à un nombre quelconque de niveaux.

Ainsi, la base de données de température globale peut s'abonner aux «capteurs/temperature/**#**» et elle recevra les températures de chaque capteur dans le monde. Si toutefois le gouvernement Français voulait utiliser les données pour leur propre service météorologique, ils pourraient simplement s'abonner aux «capteurs/temperature/fr/**#**», limitant ainsi les lectures des capteurs à celles de la France. Et si un service voulait obtenir des données de tous les types de capteurs dans un emplacement particulier, il pourrait s'abonner à : «capteurs/**+**/fr/aubiere/rue_chebarde».

## D. Connexion avec le courtier (*broker*)

Comme nous l'avons vu, MQTT connecte un client (publication ou abonnement) à un courtier.

### 1. Client

Lorsqu'on parle d'un client MQTT, cela inclut la publication et l'abonnement. En général, un client MQTT peut à la fois publier et s'abonner en même temps. La mise en œuvre par le client du protocole MQTT est très simple. C'est pourquoi MQTT est idéal pour les petits appareils.

### 2. Courtier (*broker*)

Le courtier du MQTT est le cœur de ce protocole de publication / abonnement. Il est principalement responsable de recevoir tous les messages, de les filtrer, de décider qui l'intéresse et d'envoyer le message à tous les clients souscrits. Il tient également la session de tous les clients persistants (fonctionnalités *Clear Session*), y compris les abonnements et les messages manqués. Une autre responsabilité du courtier est l'authentification et l'autorisation des clients (*ACL - Access Control List*). La plupart du temps, un courtier est également extensible, ce qui permet d'intégrer facilement l'authentification, l'autorisation et l'intégration personnalisées dans les systèmes dashboards (figure 2). En particulier, l'intégration est un aspect important, car souvent le broker est le composant, directement exposé sur Internet et qui gère beaucoup de clients, puis transmet les messages aux systèmes d'analyse et de traitement en aval.

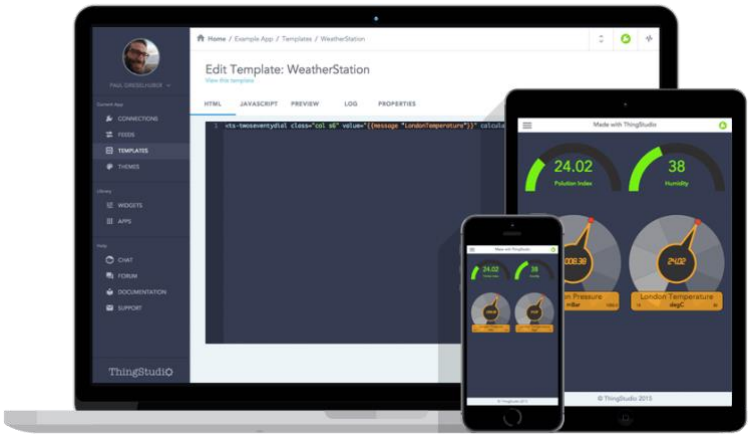


*Figure 2 : Exemple d'un dashboard sur le MQTT online : ThingsStudio*

Afin de réaliser des tests en développement on peut utiliser des courtiers MQTT en ligne avec des versions gratuites. Voici quelques exemples dans le tableau suivant :

| Broker | Server | Ports | Websocket |
|---|---|---|---|
| Mosquitto | iot.eclipse.org | 1883 / 8883 | n/a |
| HiveMQ | broker.hivemq.com | 1883 | 8000 |
| Mosquitto | test.mosquitto.org | 1883 / 8883 / 8884 | 8080 / 8081 |
| mosca | test.mosca.io | 1883 | 80 |
| HiveMQ | broker.mqttdashboard.com | 1883 | |

### 3. Connexion à un courtier

Lors d'une connexion à un courtier par un client cela exige la présence de plusieurs informations :

- Identité du client (*Client ID*)
  L'identifiant client est un identifiant de chaque client MQTT qui se connecte à un courtier MQTT et qui est unique par courtier. Le courtier l'utilise pour identifier le client et l'état actuel du client. Il est également possible d'envoyer un *Client Id* vide, ce qui entraîne une connexion sans aucun état. Une condition est que la session propre (*Clear Session*) est vraie, sinon la connexion sera rejetée.
- Séance propre (*Clear Session*)
  L'indicateur de la session propre indique au courtier, que le client souhaite ou non créer une session persistante. Une session persistante (Clean Session est fausse) signifie que le courtier stockera tous les abonnements pour le client et tous les messages manqués lors de l'abonnement avec Quality of Service (QoS) 1 ou 2. Si la session clean est définie sur true, le courtier n'a pas à stocker quoi que ce soit pour le client et purger toutes les informations d'une session persistante précédente.
- Identifiant - Mot de passe (*Username - password*)
  MQTT permet d'envoyer un nom d'utilisateur et un mot de passe pour authentifier le client. Cependant, le mot de passe est envoyé en texte clair, s'il n'est pas chiffré ou

haché par la mise en œuvre ou le TLS est utilisé en dessous. Nous recommandons fortement d'utiliser le nom d'utilisateur et le mot de passe avec un transport sécurisé de celui-ci.

- Dernière volonté et testament (*Last Will and Testament*)
  Le message de dernière volonté permet de notifier à d'autres clients, lorsqu'un client se déconnecte de manière inattendue. Un client à la connexion fournira son testament sous la forme d'un message MQTT et d'un sujet où le publier. Si ce client est déconnecté de manière inattendue, le courtier envoie ce message au nom du client.
- Inactivité de connexion (*Keep Alive*)
  Le maintien en vie (*keep alive*) est un intervalle de temps, auquel les clients s'engagent en envoyant régulièrement des messages de demande PING au courtier. La réponse du courtier avec PING Response et ce mécanisme permettra aux deux parties de déterminer si l'autre est encore vivant et accessible.

Ce sont essentiellement toutes les informations nécessaires pour se connecter à un courtier MQTT à partir d'un client MQTT. Souvent, chaque bibliothèque individuelle aura des options supplémentaires, qui peuvent être configurées. Dans ce cours nous utiliserons la librairie Python paho-mqtt.

### 4. Réponse à une connexion : CONNACK (*Connexion Acknowledge*)

Lorsqu'un courtier obtient un message de connexion, il est obligé de répondre avec un message CONNACK qui contient deux entrées de données : indicateur de la session et le code de retour.

- Indicateur de la session
  L'indicateur de la session indique si le courtier a déjà une session persistante du client à partir d'interactions antérieures. Si un client se connecte et a défini Clear Session sur true, ce drapeau est toujours faux car il n'y a pas de session disponible. Si le client a défini Clear Session sur false, le drapeau dépend de l'existence d'informations de session disponibles pour ce Client Id. Si les informations de session stockées existent, alors l'indicateur est vrai et sinon il est faux.
- Code de retour
  Le paramètre est un indicateur de validation de connexion. Il signale au client, si la tentative de connexion a été réussie et sinon, quel est le problème.

| Code de retour | Réponse du code de retour |
|---|---|
| 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 |

## E. Publier, S'abonner et se désabonner

### 1. Publier

Une fois qu'un client MQTT est connecté à un courtier, il peut publier des messages. Chaque message comporte généralement un message utile (*payload*) qui contient les données réelles à transmettre en format binaire. Un message de publication MQTT a également d'autres attributs:

- Nom du sujet (*topic*)
  Une chaîne simple, structurée hiérarchiquement avec des barres obliques en tant que délimiteurs. Un exemple serait " capteurs/temperature/fr/aubiere/rue_chebarde ".
- Qualité de Service (QOS – Quality Of Service)
  Un niveau de qualité de service (QoS) pour ce message (0,1 ou 2) détermine la garantie d'un message atteignant l'autre extrémité (client ou courtier).
- Indicateur de message retenu (Retained Flag)
  Cet indicateur détermine si le message sera enregistré par le courtier pour le sujet spécifié comme dernière valeur connue. Les nouveaux clients qui s'abonnent à ce sujet recevront le dernier message retenu sur ce sujet instantanément après l'abonnement. En savoir plus sur les messages conservés et les meilleures pratiques dans l'un des messages suivants.
- Message utile (payload)
  C'est le contenu réel du message. MQTT est totalement agnostique, il est possible d'envoyer des images, des textes dans n'importe quel encodage, des données chiffrées et pratiquement toutes les données en binaire.
- Identificateur de paquets (*client ID*)
  L'identifiant de paquet est un identifiant unique entre client et courtier pour identifier un message dans un flux de messages. Ceci n'est pertinent que pour QoS supérieur à zéro. La définition de cet identifiant interne MQTT incombe à la bibliothèque client et / ou au courtier.

Donc, lorsqu'un client envoie une publication à un courtier de MQTT, le courtier lira la publication, reconnaîtra la publication si nécessaire (selon le niveau QoS), puis le traitera. Le traitement comprend la détermination de quels clients ont souscrit sur ce sujet et ensuite l'envoi du message aux clients sélectionnés qui s'abonnent à ce sujet.
Le client, qui a d'abord publié le message, ne se préoccupe que de transmettre le message publicitaire au courtier. De là, il incombe au courtier de transmettre le message à tous les abonnés. Le client de publication ne reçoit aucun commentaire, si quelqu'un était intéressé par ce message publié et combien de clients ont reçu le message par le courtier.

### 2. S'abonner

La publication de messages n'a pas de sens si personne ne reçoit jamais le message ou, en d'autres termes, s'il n'y a pas de clients souscrits à un sujet. Un message d'abonnement est assez simple, il contient simplement un identifiant de paquet unique et une liste d'abonnements.

- Identificateur de paquets (*Client ID*)
  L'identifiant de paquet est un identifiant unique entre client et courtier pour identifier un message dans un flux de messages. La définition de cet identifiant interne MQTT incombe à la bibliothèque client et / ou au courtier.

- Liste des abonnements (*topics*)
  Chaque abonnement est une paire de sujet et de QoS. Le sujet dans le message d'abonnement peut également contenir des caractères génériques, ce qui permet de s'abonner à certains modèles de sujets. S'il y a des abonnements qui se chevauchent pour un client, le niveau de QoS le plus élevé pour ce sujet gagne et sera utilisé par le courtier pour la transmission du message.

#### 3. Réponse à un abonnement : SUBACK (*Subscribe Acknowledge*)

Chaque abonnement sera confirmé par le courtier en envoyant un accusé de réception au client sous la forme du message SUBACK. Ce message contient le même identifiant de paquet que le message d'inscription original (pour identifier le message) et une liste de codes de retour.

- Identificateur de paquets
  L'identifiant de paquet est un identifiant unique utilisé pour identifier un message. C'est le même que dans le message d'abonnement.
- Code de retour
  Le courtier envoie un code de retour pour chaque paire de questions / QoS qu'il a reçue dans le message d'abonnement. Donc, si le message d'abonnement comportait 5 abonnements, il y aura 5 codes de retour pour reconnaître chaque sujet avec le niveau QoS accordé par le courtier. Si l'abonnement a été interdit par le courtier (par exemple, si le client n'a pas été autorisé à souscrire à ce sujet en raison d'une autorisation insuffisante ou si le sujet a été mal formé), le courtier répondra avec un code de retour d'échec pour ce sujet spécifique.
  Réponse du code de retour du code de retour
  0 Succès - QoS maximale 0
  1 succès - QoS maximale 1
  2 succès - QoS maximum 2
  128 Échec

#### 4. Se désabonner

Il y a possibilité de supprimer les abonnements existants d'un client sur le courtier. Le message est similaire au message d'abonnement et possède également un identifiant de paquet et une liste de sujets. Le courtier reconnaîtra la demande de désabonnement avec le message UNSUBACK. Ce message ne contient qu'un identifiant de paquet.

## II. Caractéristiques de MQTT

### A. Niveaux de qualité de service

MQTT est conçu pour fonctionner correctement sur des réseaux peu fiables et définit trois niveaux de qualité de service (QoS). Ceux-ci permettent aux clients de spécifier la fiabilité qu'ils désirent.

### 1. QoS Niveau 0:

Le niveau le plus simple de QoS, il ne nécessite aucune reconnaissance du client et la fiabilité sera la même que celle de la couche réseau sous-jacente, TCP / IP. Le courtier / client livrera le message une fois, sans confirmation.

### 2. QoS Niveau 1:

Cela garantit que le message soit envoyé au client au moins une fois, mais il pourrait être envoyé plus d'une fois. Il s'appuie sur le client en envoyant un paquet ACK lorsqu'il reçoit un paquet. Ceci est couramment utilisé pour une livraison garantie, mais les développeurs doivent s'assurer que leurs systèmes peuvent gérer des paquets en double.

### 3. QoS Niveau 2:

C'est la QoS la moins courante, et assure qu'un message est envoyé une seule et même fois. Cette méthode nécessite un échange de quatre paquets et diminue les performances du courtier. Ce niveau est également souvent exclu des implémentations du MQTT en raison de sa complexité relative.

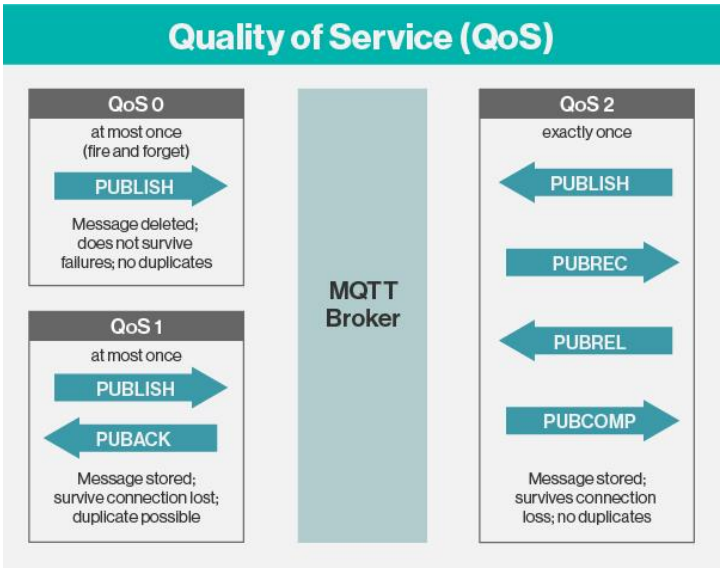


*Figure 3 : Qualité de service des échanges MQTT*
(http://www.lemagit.fr/conseil/Internet-des-Objets-bien-comprendre-MQTT )

La QoS définit la difficulté avec laquelle le courtier/client essaiera de s'assurer qu'un message soit reçu. Les messages peuvent être envoyés à n'importe quel niveau QoS et les clients peuvent tenter de s'abonner à des sujets à n'importe quel niveau QoS. Cela signifie que le client choisit la QoS maximale qu'elle recevra.
Par exemple, si un message est publié à QoS 2 et qu'un client est abonné à QoS 0, le message sera envoyé à ce client avec QoS 0. Si un second client est également abonné au même sujet, mais avec QoS 2, alors il recevra le même message mais avec QoS 2. Pour un deuxième exemple, si un client est abonné à QoS 2 et qu'un message est publié sur QoS 0, le client le recevra sur QoS 0.

### B. Messages persistants et la mise en file d'attente (*Clear Session*)

MQTT donne au client la possibilité de choisir au moment de la connexion s'il souhaite que sa session soit conservée par le serveur (*broker*) même une fois déconnecté. Dans ce cas, sa session sera restaurée lors d'une éventuelle reconnexion et le client recevra alors les messages mis en attente.

### C. Messages retenus (*Retained messages*)

Afin de pouvoir fournir aux nouveaux abonnés des données rapidement même lorsque le producteur des données ne publie pas à intervalles réguliers, il est possible d'indiquer au serveur de retenir une publication reçue comme la dernière publication connue pour un sujet donné, qui sera transmise aux nouveaux abonnées dès leur inscription. Ce sont les messages retenus (*retained messages*) qui entrent en jeu.

Un message retenu est un message MQTT normal avec l'indicateur retenu (*retained flag*) défini sur *true*. Le courtier conservera le dernier message retenu et la QoS correspondante pour cette rubrique. Chaque client qui souscrit à un modèle de sujet qui correspond au sujet du message retenu recevra le message immédiatement après son abonnement. Pour chaque sujet, seul un message retenu sera stocké par le courtier.
Ainsi, les messages retenus peuvent aider les nouveaux clients souscrits à obtenir une mise à jour d'état immédiatement après s'être abonné à un sujet et ne doivent pas attendre jusqu'à ce que les clients de publication envoient la prochaine mise à jour. En d'autres termes, un message retenu sur un sujet est la dernière bonne valeur connue, car il ne doit pas nécessairement être la dernière valeur, mais c'est certainement le dernier message avec l'indicateur retenu défini sur true.

#### 1. Envoyer un message retenu

L'envoi d'un message retenu est assez simple, il vous suffit de définir l'indicateur retenu (retained flag) d'un message de publication MQTT sur true. Chaque bibliothèque client offre généralement un moyen simple de le faire.

#### 2. Supprimer un message retenu

Pour supprimer un message retenu sur un sujet, il faut envoyez un message retenu vide sur ce. Le courtier supprime le message retenu et tous les abonnés ne recevront plus un message conservé pour ce sujet.

### A. Dernière volonté et testament (*Last Will and Testament*)

Les clients peuvent envoyer un message *Last Will and Testament* (LWT) au courtier à tout moment. Lorsque le courtier détecte que le client est hors ligne (sans fermer sa connexion), il enverra le message LWT enregistré sur un sujet spécifique, ce qui permet à d'autres clients de savoir qu'un nœud s'est mis hors ligne de manière inattendue.

### B. Inactivité de la connexion (*Keep Alive & Client Take-Over*)

### 1. Intervalle d'inactivité (*Keep Alive*)

Le protocole MQTT est basé sur TCP et qui assure une certaine garantie que les paquets soient transférés de façon «fiable ». Néanmoins, il se peut que l'une des parties communicantes soit désynchronisée avec l'autre, souvent en raison d'un accident d'un côté ou à cause d'erreurs de transmission. Cet état s'appelle une connexion à demi-ouverte. Le point important est que la fin du fonctionnement continu n'est pas informée de l'échec de l'autre côté et essaie toujours d'envoyer des messages et d'attendre des remerciements. Afin de contourner ce problème de connexion à demi ouverte ou du moins d'avoir la possibilité d'accéder si la connexion est encore ouverte, MQTT fournit la fonctionnalité keep alive.
La fonctionnalité keep alive garantit que la connexion est encore ouverte et que le courtier et le client sont connectés l'un à l'autre. Par conséquent, le client spécifie un intervalle de temps en secondes et le communique au courtier lors de l'établissement de la connexion. L'intervalle est la période de temps la plus longue possible, que le courtier et le client peuvent supporter sans envoyer de message.
Les spécifications MQTT signifie que tant que les messages sont échangés fréquemment et que l'intervalle keep alive n'est pas dépassé, il n'est pas nécessaire d'envoyer un message supplémentaire pour s'assurer que la connexion est encore ouverte.

Mais si le client n'émet aucun message pendant la période de conservation, il doit envoyer un paquet PINGREQ au courtier pour confirmer sa disponibilité et veiller à ce que le courtier soit toujours disponible. Le courtier doit déconnecter un client qui n'émet pas PINGREQ ou tout autre message en une fois et demi de l'intervalle de maintien en vie. De même, le client doit fermer la connexion si la réponse du courtier n'est pas reçue dans un délai raisonnable.
Le protocole fournit une méthode pour détecter lorsque les clients ferment leurs connexions incorrectement en utilisant des paquets de durée maximale de session (*keepalive*). Donc, lorsqu'un client manque de batterie, il se bloque ou son réseau diminue, le courtier peut agir.

### 2. Prise en charge (*Client Take-Over*)

Un client déconnecté tentera probablement de se connecter à nouveau. Il se pourrait que le courtier ait toujours une connexion à moitié ouvert pour le même client. Dans ce scénario, le MQTT effectuera une prise en charge du client. Le courtier fermera la connexion précédente au même client (déterminé par le même identifiant client) et établit la connexion avec le client nouvellement connecté. Ce comportement garantit que la connexion à moitié ouverte n'empêchera pas un nouvel établissement de connexion du même client.

## C. Sécurité

La sécurité pour MQTT (et les périphériques IoT en général) est un sujet assez important, dont nous aborderons plus loin dans le cours. Il existe deux principales fonctionnalités liées à la sécurité, l'authentification et le chiffrement.
L'authentification est fournie par l'envoi d'un nom d'utilisateur et d'un mot de passe avec le paquet de connexion MQTT. Presque tous les courtiers et les implémentations de clients soutiendront cela, mais il est vulnérable à l'interception. Pour éviter cela, TLS devrait être utilisé si possible.
Le protocole lui-même ne fournit aucune fonctionnalité de chiffrement, mais comme MQTT s'exécute sur TCP, nous pouvons facilement utiliser TLS et ainsi fournir une connexion chiffrée. Cela augmente toutefois la complexité informatique des messages d'envoi et de réception, ce

qui peut poser problème sur les systèmes contraints et influencer également les performances des courtiers. Nous verrons ce problème particulier plus tard.
Sur la figure 4, on observe des mauvaises utilisations de ce protocoles dans son utilisation dans la technologie IoT.

Caisse enregistreuse britannique ?

Compteur d'énergie (smartmeter) et sonde météo BLE ?

*Figure 4 : Exemples de mauvaises utilisations du protocole MQTT*

## III. Client MQTT sous le langage Python : paho-mqtt

La bibliothèque Eclipse Paho MQTT Python (https://pypi.org/project/paho-mqtt/) implémente les versions 3.1, 3.1.1 et 5.0 du protocole MQTT.
Ce code fournit une classe de client qui permet aux applications de se connecter à un courtier *(brocker)* MQTT pour publier (*publish*) des messages et s'abonner (*subscribe*) à des sujets (*topic*) et recevoir des messages publiés. Il fournit également des fonctions d'aide pour rendre la publication unique des messages sur un serveur MQTT très simple.
Il prend en charge Python 2.7.9+ ou 3.5+.

## IV. Exercices : Manipulations de MQTT avec Python

Nous allons réaliser plusieurs exercices autour du protocole MQTT à l'aide de la librairie Python paho-mqtt.

### A. Connexion à un broker public

Réaliser un programme sous Python afin de vous connecter au broker public : https://test.mosquitto.org. La connexion se fait de façon anonyme sur le port 1883.

Ressources : https://pypi.org/project/paho-mqtt/

*Correction : Connexion à un broker*

```
import paho.mqtt.client as mqtt

def on_connect(client, userdata, flags, rc):
        print("Connected with result code => " + mqtt.connack_string(rc))

client = mqtt.Client()
client.on_connect = on_connect

try:
```

```
        # Login et password si l'on est avec un broker avec authentification (cf. TP2)
        #client.username_pw_set(username="GX", password="isimaGX")

        client.connect("test.mosquitto.org", 1883)
except:
        print("Connection Failed")

client.loop_forever()
```

### B. S'abonner à un topic

Abonnez-vous au topic "isima/GX" afin de récolter les informations envoyées sur ce topic. Vous mettrez également en place le callback lors de la réception d'un message afin d'afficher ses caractéristiques (payload, retained message, qos) et le callback lors de votre abonnement afin d'afficher le retour du broker.

*Correction : S'abonner à un topic*

```
import paho.mqtt.client as mqtt

def on_connect(client, userdata, flags, rc):
        print("Connected with result code => "+mqtt.connack_string(rc))

def on_subscribe(client, userdata, mid, granted_qos):
        print("Subscribed: " + str(mid) + " " + str(granted_qos))

def on_message(client, userdata, msg):
        print("Received message '" + str(msg.payload) + "' on topic '" + msg.topic + "' with QoS
" + str(msg.qos))

client = mqtt.Client()

client.on_subscribe = on_subscribe
client.on_message = on_message
client.on_connect = on_connect

try:
        # Login et password si l'on est avec un broker avec authentification (cf. TP2)
        #client.username_pw_set(username="GX", password="isimaGX")

        client.connect("test.mosquitto.org", 1883)
except:
        print("Connection Failed")

client.subscribe("isima/GX", qos=0)

client.loop_forever()
```

### C. Publier sur un topic

Publier sur le topic "isima/GX" et vérifier le bon fonctionnement du système en mettant en place le callback lors de votre publication. Essayer de publier sur d'autres topic "isima/GY" (ou X!=Y), "isima/GX/sensor1", "isima", "my_topic".

*Correction : Publier sur un topic*

```
import paho.mqtt.client as mqtt
import time

def on_connect(client, userdata, flags, rc):
	print("Connected with result code => "+mqtt.connack_string(rc))

def on_subscribe(client, userdata, mid, granted_qos):
	print("Subscribed: " + str(mid) + " " + str(granted_qos))

def on_message(client, userdata, msg):
	print("Received message '" + str(msg.payload) + "' on topic '" + msg.topic + "' with QoS
" + str(msg.qos))
	if msg.retain == 1:
		print("This is a retained message")

def on_publish(client, userdata, mid):
	print("-- on_publish callback -- mid: " + str(mid) )

client = mqtt.Client()

client.on_subscribe = on_subscribe
client.on_message = on_message
client.on_connect = on_connect
client.on_publish = on_publish

try:
	# Login et password si l'on est avec un broker avec authentification (cf. TP2)
	#client.username_pw_set(username="GX", password="isimaGX")

	client.connect("test.mosquitto.org", 1883)
except:
	print("Connection Failed")

client.subscribe("isima/#", qos=0)

client.loop_start()

while True:
	(rc, mid) = client.publish(topic="isima/GX", payload="TEST_GX", qos=0)
```

```
        print("Error return from publish of mid = " + str(mid) +" : " + mqtt.error_string(rc))

        time.sleep(5)
```

### D. Gestion du message retenu

Lors de l'abonnement (subscribe) à un sujet (topic) vérifier si le message reçu est un message retenu (Retained Message).

Dans un deuxième temps, publier (publish) un message avec l'indicateur message retenu (Retained Message Flag). Vérifier son bon fonctionnement, puis supprimer ce message retenu. Vérifier à nouveau le bon fonctionnement du mécanisme.

*Correction : Gestion du message retenu*
*Afin de détecter si un message est un message retenu, on peut modifier la fonction de callback on_message et vérifier l'indicateur :*

```
def on_message(client, userdata, msg):
   print("Received message '" + str(msg.payload) + "' on topic '" + msg.topic + "' with QoS " +
str(msg.qos))
   if msg.retain == 1:
      print("This is a retained message")
      # Actions à réaliser si le message est un message retenu
```

Afin de publier un message retenu, il suffit de rajouter un paramètre lors de la publication:

```
(rc, mid) = client.publish(topic="isima/GX", payload=messageGX, retain=True)
```

Afin de supprimer un message retenu, il suffit de publier un message (payload) vide:

```
(rc, mid) = client.publish(topic="isima/GX", payload="", retain=True)
```

### E. Mise en place d'une dernière volonté

Mettre en place un message de dernière volonté et testament (Last Will and Testament Message). Vérifier son bon fonctionnement par un arrêt brutal de votre client en observant les messages publiés sur le broker public.

*Correction : Mise en place d'une dernière volonté*
*Après la création du client MQTT, il suffit d'utiliser la fonction will_set(). Lors d'une déconnexion du client, un message sera publié sur le topic spécifié.*

```
client = mqtt.Client()
client.will_set(topic = MQTT_LAST_WILL_TOPIC, payload = MQTT_LAST_WILL_MSG, qos = 0,
retain = False)
```

## F. Mise en place d'une session persistante

Mettre en place une session persistante et reconnectez-vous pour voir le bon fonctionnement du système (avez-vous récupéré les messages émis lors de votre déconnexion ?).

*Correction : Mise en place d'une session persistante*
*Lors de la création du client MQTT, il existe un paramètre pour spécifier que vous souhaitez une session persistante (Clean Session). Pour que cela fonctionne, vous devez préciser un client ID, sinon le broker ne vous connaît pas et ne sait pas les messages que vous n'avez pas reçu.*

```
MQTT_CLIENT_ID = "my_ID"
MQTT_CLEAR_SESSION = True # ou False
client = mqtt.Client(client_id=MQTT_CLIENT_ID,clean_session=MQTT_CLEAR_SESSION)
```

## G. Programme des "capteurs"

Pour information durant vos manipulations, le programme suivant tournait afin de publier sur les topics associés aux différents groupes de TP.

```
import paho.mqtt.client as mqtt
import random as ra
import time

# Define variable
MQTT_HOST_NAME      = "…" # Host of broker MQTT
MQTT_HOST_PORT      = ….    # Port of broker MQTT

MQTT_USERNAME       = "…."    # Username of user log (here PROF log)
MQTT_PASSWORD       = "…."    # Password of user log (here PROF log)

MQTT_CLIENT_ID      = "prof_ISIMA"      # Client ID
MQTT_LAST_WILL_TOPIC  = "isima/LWT"       # Topic of Last Will Message
MQTT_LAST_WILL_MSG    = "Lost Connection"   # Message of Last Will Message
MQTT_QOS            = 0               # Quality of service
MQTT_CLEAR_SESSION    = True            # Flag Clean Session
MQTT_RETAINED_MSG     = True            # Flag Retained Session

MQTT_CLEAR_RETAINED   = True

# The callback for when the client receives a CONNACK response from the server.
def on_connect(client, userdata, flags, rc):
    print("-- on_connect callback -- Connection returned result: " + mqtt.connack_string(rc))
    if rc == 0:
        global Connected  # Use global variable
        Connected = True  # Signal connection
```

```
# The callback for when the client publish on server.
def on_publish(client, userdata, mid):
   print("-- on_publish callback -- mid: " + str(mid) )

# The callback for when the client receives a message from server.
def on_message(client, userdata, msg):
   print("-- on_message callback -- Received message '" + str(msg.payload) + "' on topic '" +
msg.topic + "' with QoS " + str(msg.qos))
   if msg.retain == 1:
      print("This is a retained message")

Connected = False   #global variable for the state of the connection

client = mqtt.Client(MQTT_CLIENT_ID,MQTT_CLEAR_SESSION)
client.will_set(MQTT_LAST_WILL_TOPIC,MQTT_LAST_WILL_MSG, 0, False)

client.on_publish = on_publish
client.on_connect = on_connect
client.on_message = on_message

try:
   #client.username_pw_set(MQTT_USERNAME,MQTT_PASSWORD)
   client.connect(MQTT_HOST_NAME,MQTT_HOST_PORT)
except:
   print("Connection failed")

client.loop_start()

while Connected != True:  # Wait for connection
   time.sleep(0.1)

client.subscribe("$SYS/broker/version") # Sample of system topic

if MQTT_CLEAR_RETAINED:
   (rc, mid) = client.publish(topic="isima/G1", payload="", qos=MQTT_QOS, retain=True)
   (rc, mid) = client.publish(topic="isima/G2", payload="", qos=MQTT_QOS, retain=True)
   (rc, mid) = client.publish(topic="isima/G3", payload="", qos=MQTT_QOS, retain=True)
   (rc, mid) = client.publish(topic="isima/G4", payload="", qos=MQTT_QOS, retain=True)

while True:
   tempG1 = 15 + ra.uniform(-10, 10)
   (rc,   mid)   =   client.publish(topic="isima/G1",   payload=tempG1,   qos=MQTT_QOS,
retain=MQTT_RETAINED_MSG)
   print("Error return from publish of mid = " + str(mid) +" : " + mqtt.error_string(rc))

   tempG2 = 15 + ra.uniform(-10, 10)
```

```
    (rc, mid) = client.publish(topic="isima/G2", payload=tempG2, qos=MQTT_QOS, retain=MQTT_RETAINED_MSG)
    print("Error return from publish of mid = " + str(mid) +" : " + mqtt.error_string(rc))

    tempG3 = 15 + ra.uniform(-10, 10)
    (rc, mid) = client.publish(topic="isima/G3", payload=tempG3, qos=MQTT_QOS, retain=MQTT_RETAINED_MSG)
    print("Error return from publish of mid = " + str(mid) +" : " + mqtt.error_string(rc))

    tempG4 = 15 + ra.uniform(-10, 10)
    (rc, mid) = client.publish(topic="isima/G4", payload=tempG4, qos=MQTT_QOS, retain=MQTT_RETAINED_MSG)
    print("Error return from publish of mid = " + str(mid) +" : " + mqtt.error_string(rc))

    time.sleep(3)
```