ENSEIRB-MATMECA

CONCEPTION D'OBJETS CONNECTES PAR PROTOTYPAGE RAPIDE ET PROTOCOLE MQTT
- TP3 -

Léo Mendiboure

TABLE DES MATIERES

1. PRÉSENTATION DE MQTT

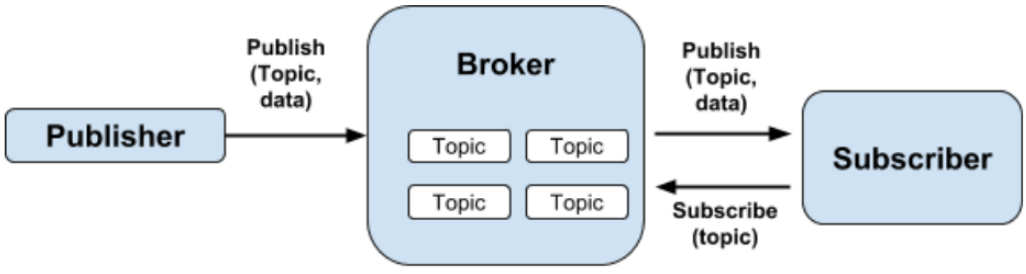
Ce TP doit vous permettre de découvrir le fonctionnement de MQTT ainsi que quelques une des utilisations de ce protocole.

MQTT (Message Queuing Telemetry Transport) est un protocole de messagerie léger basé sur TCP/IP. Celui-ci est très utilisé dans le domaine de l'IoT aussi bien pour des applications médicales et domotiques que de transport, de logistique ou de sécurité et de surveillance. Parmi les principaux avantages de ce protocole on peut citer la mise à l'échelle, la réduction de consommation de bande passante (lié au modèle Pub/Sub), ses capacités bidirectionnelles, la simplification des communications ou encore la réduction des temps de développement.

Ce protocole doit permettre d'assurer la transmission de données tant pour du sensing/crowdsensing que du contrôle à distance entre un serveur et des clients. Pour ce faire, MQTT suit un modèle Pub/Sub (Publish/Subscribe) visible ci dessous. Cinq notions principales sont nécessaires à la compréhension de ce modèle : publisher (ou producer), broker, subscriber (ou consumer), topic et data.

Le publisher et le subscriber sont tous les deux clients du broker MQTT (serveur intermédiaire pour tous les clients). Chaque publisher envoie au broker des données (data) sur un sujet donné (topic). Chaque subscriber s'abonne (subscribe) à certains topics et reçoit (du broker) en push toute donnée concernant ses abonnements. Dans notre cas ce broker MQTT va être utilisé pour le relevé d'informations des capteurs.

Différentes versions libres et open-source de brokers MQTT et de bibliothèques (C, C++, Java, Javascript, Python, etc.) permettant de programmer des clients MQTT existent. Dans le cadre de l'IoT et des réseaux M2M (Machine-to-Machine) on parle notamment pour les brokers d'ActiveMQ, JoramMQ et Mosquitto (utilisé dans ce TP). Pour ce qui est des bibliothèques destinées au client, la plus avancée est le projet Eclipse Paho qui offre des implémentations de différents protocoles de messagerie.



Fonctionnement de MQTT

Mots clés : objet connecté, Raspberry Pi, Sense HAT, MQTT, Raspbian, langage Python

2. TP 1 : MISE EN PRATIQUE

2.1. Introduction à MQTT

- Question 1 : Réexpliquez le fonctionnement de MQTT.

- Commencez par lancer le service MQTT :
  host % sudo service mosquitto start

- Maintenant que le broker est lancé, on va créer dans un premier terminal un client qui s'abonne au topic "*testMQTT"* et dans un autre terminal un second client qui publie le message "*Hello world !"* dans ce topic.

  Pour cela il vous faudra utiliser les commandes :
  host % mosquitto_sub
  host % mosquitto_pub

  Attention : Pour indiquer à quel topic s'abonner (*mosquitto_sub)* et sur quel topic publier "Hello world !" ces fonctions ont besoin d'arguments (notamment "-t") que vous pourrez trouver grâce aux aides (*mosquitto_sub --help)*.

  Lancez le subscriber dans un terminal et, une fois le message publié dans un autre terminal, vérifiez qu'il a bien été reçu par le subscriber.

- Maintenant que vous avez réussi, ouvrez un troisième terminal et créez un second client abonné au même topic. En publiant un nouveau message, vérifiez que les deux clients subscribers le reçoivent bien.

2.2. Paho Python MQTT Client

- Question 2 : Qu'est ce que Paho MQTT ?

- Créez dans le dossier courant deux fichiers *mqtt_subscriber.py* et *mqtt_publisher.py* que nous utiliserons dans la suite de ce TP.

- Le code ci-dessous correspond à une implémentation simple d'un subscriber MQTT, copiez le dans le fichier *mqtt_subscriber.py* en remplaçant les "???" par des données pertinentes. A noter que *MQTT_SERVER* correspond à l'adresse du serveur MQTT et *MQTT_PATH* au nom du topic.

```
import paho.mqtt.client as mqtt
MQTT_SERVER = "???" # MQTT server address
MQTT_PATH = "???" # Topic name

def on_connect(client, userdata, flags, rc):
  print("Connection code : "+str(rc))
  # Subscribe to the topic
  client.subscribe(MQTT_PATH)
# A publish message is received from the server
def on_message(client, userdata, msg):
  print("Sujet : "+msg.topic+" Message : "+str(msg.payload))

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect(MQTT_SERVER, 1883, 60)

client.loop_forever()
```

- De même, copiez le code ci-dessous dans le fichier *mqtt_publisher.py en remplaçant les "???".*

```
import paho.mqtt.publish as publish
MQTT_SERVER = "???"
MQTT_PATH = "???"
publish.single(MQTT_PATH, "Hello World!", hostname=MQTT_SERVER)
```

- Lancez le programme *mqtt_subscriber.py* puis le programme *mqtt_publisher.py* dans un autre terminal et vérifiez que le subscriber a bien reçu le message envoyé.

  Remarque :Dans le cas où la connexion/l'envoi de données serait impossible, accédez au fichier */etc/mosquitto/mosquitto.conf* et remplacez la ligne *allow_anonymous false* par *allow_anonymous true*. Une fois cette modification réalisée, relancez le serveur mqtt pour qu'elle soit prise en compte : *sudo service mosquitto restart*.

2.3. Ping (Keep alive)

- Modifiez le fichier *mqtt_subscriber.py* pour y ajouter une nouvelle fonction (à la fin de cette sous section 2.3 vous pourrez si vous le souhaitez retirer cette fonction, ce qui évitera de voir les logs s'afficher à l'écran) :

```
def on_log(client, userdata, level, buf):
   print("log: ",buf)

client.on_log=on_log # set client logging
```

Pour éviter d'avoir à attendre 60 secondes, vous pouvez également remplacer la valeur 60 par une valeur plus petite dans le *client.connect* (par 6 par exemple).

Relancez le client MQTT.

- Question 3 : Que constatez vous une fois que les *X* secondes indiquées dans le *client.connect* sont écoulées ? Quel est l'intérêt d'un tel mécanisme ? Que se passe-t-il si le *publisher* envoie un message au *subscriber* ? Que se passe-t-il à votre avis si le broker MQTT n'obtient pas de réponse du client ?

2.4. Sessions persistantes et QoS

- Dans le fichier *mqtt_subscriber.py* on va ajouter un nouveau paramètre au moment de la création du client MQTT : *clean_session = False*.
  Une fois ce paramètre ajouté, lancez le client MQTT puis arrêtez le. Lancez le publisher *mqtt_publisher.py* et relancez le client.

  Question 4.a : Qu'observez vous ?

- On va maintenant modifiez le paramètre *clean_session* et le faire passer à *True*. Essayez de relancer le client MQTT.

  Question 4.b : Que constatez vous ? Quel peut être à votre avis l'intérêt de définir l'ID du client pour la création d'une session persistante ?

- Pour tester l'intérêt de la session persistante, on va tout d'abord essayer de la tester en l'état.
  Après avoir défini un identifiant pour le client (*client_id="test"*) au moment de la création du client MQTT, répétez les mêmes opérations qu'au cours de la première partie de cette sous section : Lancez le client, arrêtez le, lancez le publisher, relancez le client.

  Vous devriez constater qu'aucun changement n'a eu lieu.

  En effet, de base avec Paho MQTT la QoS est de 0. Avec ce niveau de QoS les messages ne sont délivrés qu'a condition que le publisher et le subscriber soient actuellement connectés.

  Pour tester l'intérêt de ces sessions persistantes on va donc modifier la valeur de la QoS, et pour le publisher et pour le subscriber.

  Pour ce faire, on va ajouter le paramètre *qos=1* au moment où le publisher envoie un message et au moment où le subscriber souscrit à un topic.

  Une fois que c'est réalisé, répétez les opérations précédentes : lancez le client et arrêtez le, lancez le publisher (plusieurs fois!) et relancez le client MQTT.

- Question 4.c : Que constatez vous ? Quel est l'intérêt de ce type de sessions persistantes ? Cherchez (sur internet) quelle est la différence entre une QoS de 1 et 2. Cherchez également ce qui est stocké dans une session persistante. Enfin, cherchez dans quels cas il est intéressant d'utiliser ces niveaux de QoS tant pour le publisher que pour le subscriber.

2.5. Message enregistré (Retained Message)

- Dans le fichier *mqtt_publisher.py*, ajoutez maintenant un nouveau paramètre à la fonction *publish.single* :

  retain=True

- Question 5 : Relancez le client publisher puis lancez un nouveau client subscriber (**dans cet ordre !**), que constatez vous ? Quel pourrait être l'intérêt d'un tel mécanisme ?

2.6. Acte de dernière volonté (Last Will and Testament)

- On va maintenant modifier le publisher MQTT pour pouvoir expérimenter une autre fonctionnalité. Pour ce faire, remplacez le contenu du fichier *mqtt_publisher.py* par :

```
import paho.mqtt.client as mqtt
MQTT_PATH = "t_channel/test"
MQTT_SERVER = "localhost"


client = mqtt.Client()
client.will_set(MQTT_PATH, "Does it works ?",qos=0)
client.connect(MQTT_SERVER, 1883, 60)
client.loop_start()
client.publish(MQTT_PATH, "Hello World")
client.disconnect()
client.loop_stop()
```

  Lancez le subscriber puis lancez le publisher, observez ce qui se passe au niveau du subscriber.

- Répétez maintenant la même opération en enlevant la dernière ligne du fichier *mqtt_publisher.py* (*client.loop_stop()*).

  Question 6 : Que constatez vous ? On parle d'acte de dernière volonté, quel peut être à votre avis l'intérêt d'un tel mécanisme ? Dans quel cas constatez vous que ce mécanisme est enclenché ?

2.7. Remontée d'informations des capteurs et gestion de multiples topics

En utilisant les travaux menés au cours des TPs précédents (Sense Hat), on va maintenant créer deux nouveaux topics permettant de s'abonner à la température et à la pression.

- Dans le fichier *mqtt_publisher.py*, créez deux nouveaux topics "*Temperature*" et "*Pression*" qui renverront respectivement les valeurs de la température et de la pression récupérées de la carte *Sense Hat* (il vous faudra par conséquent récupérer des précédents TPs les lignes de codes permettant de récupérer ces données, *ie sense.get...*).

- Créez deux nouveaux clients subscribers, un premier qui s'abonnera au topic *Temperature* et un second qui s'abonnera au topic *Pression*.

- Vérifiez le bon fonctionnement du système en lançant les deux clients subscribers et le client publisher.

2.8. Arborescence et joker

Avec MQTT il existe également la notion de *wildcard*, qui doit permettre de s'abonner à plusieurs topics en même temps. Cette notion est directement rattachée à la notion d'arborescence. Ainsi un topic peut être séparés en différents niveaux séparés par des slashs (/), par exemple :

"myRasp / sensors/ temperature"
"myRasp / sensors / pression"

Il s'agit là de deux topics à trois niveaux, un premier niveau correspondant à "myRasp", un second correspondant à "sensors" et un troisième correspondant à "temperature" ou "pression". Il pourra être intéressant pour certains client de s'abonner à l'ensemble des informations produites par myRasp ou l'ensemble des sensors provenant de myRasp. Rapporté à un cas réel il pourra par exemple être intéressant pour certains clients de récupérer l'ensemble des informations correspondant à ma cuisine ou de manière plus vaste à l'ensemble des informations provenant du premier étage de ma maison.

Ce multi abonnement est possible grâce à des jokers : "+" et "#". Ici on va simplement s'intéresser au second "#", nommé joker multi-niveaux qui permet de s'abonner à l'ensemble des topics existant à un niveau et aux niveaux plus bas. Par exemple "myRasp/sensors/#" permettra à la fois de s'abonner aux informations provenant du capteur de température et aux informations provenant du capteur de pression.

- Modifiez le nom des topics présents dans le fichier *mqtt_publisher.py* pour qu'ils respectent l'arborescence présentée ci-dessus

- Créez un nouveau client MQTT pour qu'à l'aide d'un joker il s'abonne aux informations provenant des différents capteurs.

- Vérifiez le bon fonctionnement du système.

Considérant l'arborescence ci dessous :

myRasp / groundfloor / livingroom / temperature
myRasp / groundfloor / kitchen / temperature
myRasp / groundfloor / kitchen / pression
myRasp / firstfloor / kitchen / temperature
myRasp / groundfloor / kitchen / fridge / temperature

L'utilisation de "myRasp/groundfloor/+/temperature" permettra de s'abonner à :
myRasp / groundfloor / livingroom / temperature
myRasp / groundfloor / kitchen / temperature

Mais non aux autres, le deuxième ou le quatrième niveau ne correspondant pas à ce qui est attendu.

- Question 7 : Que semble donc permettre de faire ce second joker (+) ? Expliquez son fonctionnement et son intérêt.

- Question 8 : Que permettent de faire les jokers ? Donnez des exemples !

2.9. Authentification

Actuellement, tout client, même non identifié peut se connecter au broker mais également lire le contenu de l'ensemble des topics sans aucun contrôle. Dans cette partie nous allons configurer le broker MQTT pour que l'authentification d'un client avec un nom d'utilisateur et un mot de passe valides soit nécessaire pour qu'une connexion soit possible.

- Commencez par créer un fichier de stockage des mots de passe grâce à la commande (cette commande ne peut être utilisée sans que le nom d'un utilisateur soit entré !) :

  host % mosquitto_passwd -c <nom_du_fichier> <nom_du_user>

- Affichez le contenu du fichier que vous venez de créer. A quoi ressemble-t-il ? Ajoutez un second utilisateur à ce fichier avec la commande *mosquitto_passwd <nom_du_fichier> <nom_du_user>*

- On va maintenant utiliser ce fichier. Pour cela il va falloir le copier dans le dossier */etc/mosquitto/* et modifier le fichier */etc/mosquitto/mosquitto.conf* en y ajoutant les lignes suivantes :

  password_file /etc/mosquitto/<nom_du_fichier>
  allow_anonymous false

  A noter que la seconde ligne permet d'empêcher la connexion de clients ne possédant pas d'identifiants (de clients anonymes). Pour simplifier le développement on pourra à nouveau passer à true cette valeur dans les autres parties de ce TP.

- Après avoir relancé le service MQTT (*sudo service mosquitto restart*), essayez à nouveau de relancer le client subscriber python, que se passe-t-il ?

- Modifiez les fichiers subscriber et publisher pour que l'authentification soit prise en compte :
  - Dans le fichier *mqtt_subscriber.py*, ajoutez la ligne suivante (avant "*client.on_connect = on_connect*") en utilisant les données de l'utilisateur que vous venez de créer :

    client.username_pw_set("<username">,"<password">)

  - Faites de même dans le fichier *mqtt_publisher.py*.

- Vérifiez qu'il est à nouveau possible de publier et souscrire à des topics.

- Question 9 : Différents codes de retour (« 5 » par exemple) ont été renvoyés par le broker MQTT lors des tentatives de connexion du subscriber, quels peuvent être les différents codes renvoyés par le broker MQTT et à quoi correspondent ils ?

- Essayez maintenant de mettre en place des restrictions pour l'accès en écriture/lecture aux topics.
  Pour cela vous aurez besoin d'indiquer dans le fichier de configuration *mosquitto.conf* où trouver le fichier de gestion du contrôle d'accès (*acl_file*) ainsi que de définir le contenu de ce fichier.

  En sachant que la définition des droits dans le fichier *acl_file* revient à :
  user <username>
  topic [read|write|readwrite] <topic_name>

  Question 10 : A quoi devrait ressembler le contenu du fichier *acl_file* pour que le premier utilisateur *user1* puisse accéder au topic *myRasp / groundfloor / livingroom / temperature* en lecture et le second utilisateur *user2* en écriture ?

2.10. Conclusion

Une vision simple de MQTT est présentée dans ce TP et les capacités de ce protocole sont bien supérieures à ce qui est introduit ici (utilisation avec Python, authentification, retained messages, QoS, gestion de topics, arborescence et jokers).

Une utilisation étendue de MQTT consisterait à l'utiliser non sur un seul équipement mais sur un ensemble d'équipements formant un réseau local qui pourraient partager des informations ou les remonter à internet, rendant possible un contrôle à distance.

Une limite importante de MQTT est la sécurité, en effet actuellement pour qu'une utilisation sécurisée soit possible SSL/TLS doivent être mis en place, ce qui "alourdit" fortement le protocole.

Des alternatives existent à MQTT, notamment CoAP (Constrained Application Protocol) qui suit un modèle requête/réponse ou AMQP (Advanced Message Queuing Protocol) qui suit tout comme MQTT un modèle Pub/Sub. Toutefois, MQTT occupe et devrait occuper à l'avenir une place importante.

- Question 11 : Donnez des exemples concrets d'utilisation de MQTT.