ENSEIRB-MATMECA

CONCEPTION D’OBJETS CONNECTES PAR PROTOTYPAGE RAPIDE
- TP2 -

Léo Mendiboure

TABLE DES MATIERES

1. PRÉSENTATION DE MQTT

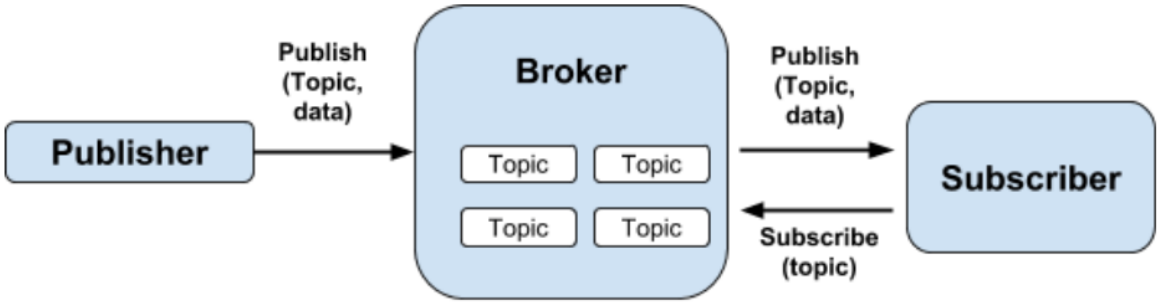
Ce TP doit vous permettre de découvrir le fonctionnement de MQTT ainsi que quelques une des utilisations de ce protocole.

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 ou encore la réduction des temps de développement.

Ce protocole doit permettre d'assurer la transmission de données tant pour du sensing 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 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

- 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

La librairie Eclipse Paho MQTT Python Client permet d'utiliser le broker MQTT à l'aide de scripts Python, permettant en particulier l'implémentation de clients 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.

2.3. 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 ?
  (Si vous le souhaitez vous pouvez ajouter un utilisateur à ce fichier : *mosquitto_passwd -d <nom_du_fichier> <nom_du_user> <mot_de_passe>* ou en supprimer un utilisateur : *mosquitto_passwd -D <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 start*), 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">)

  - Dans le fichier *mqtt_publisher.py*, modifiez la commande *publish.single* en y ajoutant le paramètre suivant en utilisant les données de l'utilisateur que vous venez de créer :

    auth={'username':'<username>', 'password' : '<password>'}

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

2.4. Message enregistré (Retained Message)

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

  retain=True

  - 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 ?

2.5. 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.

2.6. 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.

Pour information le joker "+" est quand à lui un joker agissant à un seul niveau. Par exemple, si l'on utilise l'arborescence suivante :

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.

- S'il vous reste du temps, vous pouvez expérimenter l'utilisation de ce second (+) joker dans une arborescence que vous aurez à définir.

2.7. Ouverture et 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, gestion de topics, arborescence et jokers). Des niveaux de qualité de service peuvent par exemple être définis, trois au total, le niveau QoS0 ("At most one") , le niveau QoS1 ("At least one") et le niveau QoS2 ("Exactly once").

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, et réaliser du contrôle à distance, ou les remonter à internet.

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.