Conception et implémentation des objets connectés

Dr. Ing. Chiheb Ameur ABID

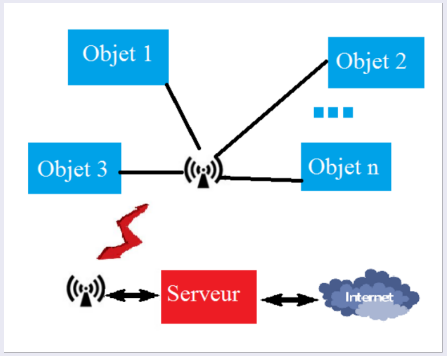
Contact: chiheb.abid@fst.utm.tn

Année universitaire : 2021 - 2022

Plan

Architecture d'une application IOT

Architecture d'une application IOT



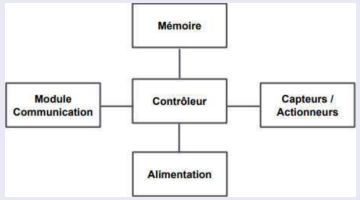
Protocole de communication MQTT

MQTT en bref

- MQTT (Message Queue Telemetry Transport)
- Un protocole de messagerie publish-subscribe basé sur le protocole TCP/IP utilisé pour la communication M2M
- Possibilité de sécuriser le transport de messages en SSL/TLS et par identification de l'utilisateur
- Taille maximale d'un message : 256 Mo

Protocole de communication MQTT

MQTT en bref



- Les messages sont envoyés par des publieurs (les publishers) sur un canal appelé Topic
- Les messages peuvent être lus par les abonnés (les subscribers)
- Les Topics (ou les canaux d'informations) peuvent avoir une hiérarchie qui permet de sélectionner finement les informations que l'on désire.
- Corps des messages : format JSON

Protocole de communication MQTT

Qualité de service (QoS)

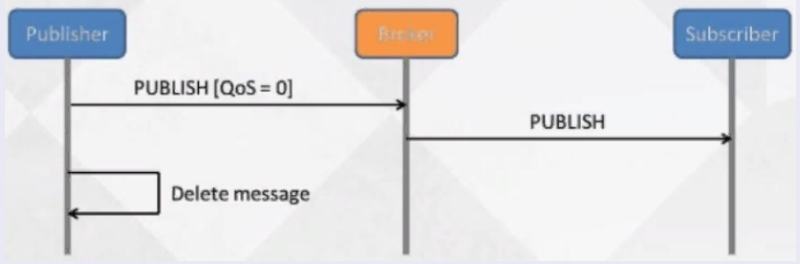
- La Qualité de Service (QoS) est un accord qui définit la garantie de livraison d'un message spécifique et le type d'authentification utilisé
- Il y a trois niveaux de QoS dans `MQTT`

❶ `QoS 0` : Au plus une fois

- Le message est distribué une fois au plus, ou n'est pas distribué du tout.
- Sa distribution via le réseau n'est pas accompagnée d'un accusé de réception.
- Le message n'est pas stocké. Le message peut être perdu si le client est déconnecté ou si le serveur échoue.
- Le mode de transfert le plus rapide. Il est parfois appelé "autonome après diffusion".

Protocole de communication MQTT

QoS 0 : At most once (fire and forget)



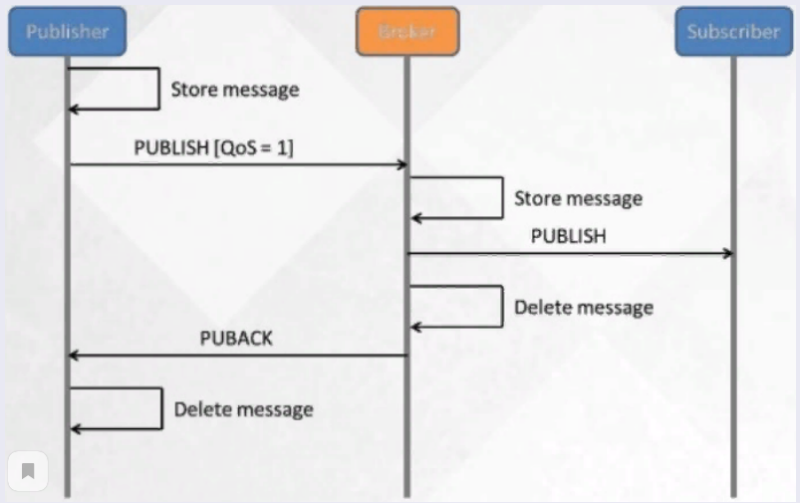
Protocole de communication MQTT

❷ QoS 1 : Au moins une fois

- Le message est toujours distribué au moins une fois.
- Si l'expéditeur ne reçoit pas d'accusé de réception, le message est renvoyé avec l'indicateur DUP défini jusqu'à la réception de l'accusé de réception
- Le message doit être stocké localement au niveau de l'expéditeur et du destinataire jusqu'à ce qu'il soit traité.
- Le message est supprimé de l'expéditeur une fois qu'il a reçu un accusé de réception de la part du destinataire.

Protocole de communication MQTT

QoS 1 : At least once



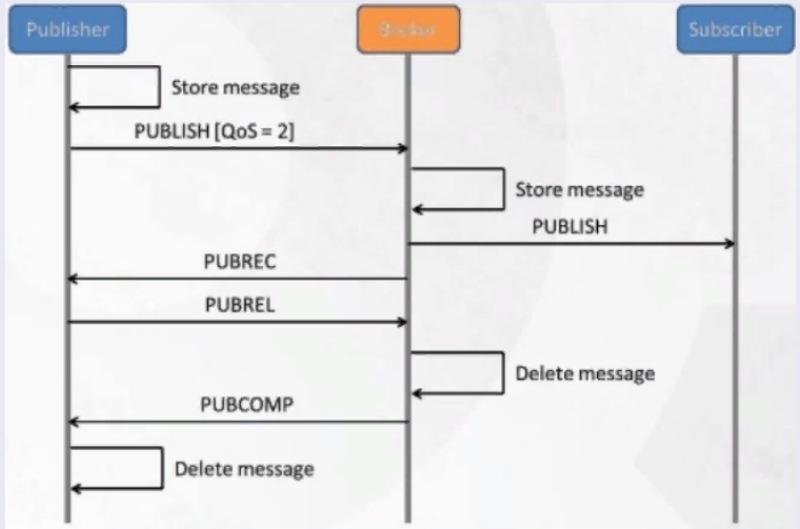
Protocole de communication MQTT

❸ QoS 2 : Une seule fois

- Le message est toujours distribué une seule fois
- Le message doit être stocké localement au niveau de l'expéditeur et du destinataire jusqu'à ce qu'il soit traité
- Le mode de transfert le plus sûr, mais le plus lent
- Il faut au moins deux paires de transmissions entre l'expéditeur et le destinataire
  1. L'expéditeur transmet le message et reçoit un accusé de réception du destinataire indiquant qu'il a stocké le message. Si l'expéditeur ne reçoit pas d'accusé de réception, le message est renvoyé avec l'indicateur DUP défini jusqu'à la réception de l'accusé de réception.
  2. L'expéditeur dit au destinataire qu'il peut terminer le traitement du message, `PUBREL`. Si l'expéditeur ne reçoit pas d'accusé de réception du message `PUBREL`, le message `PUBREL` est renvoyé jusqu'à la réception de l'accusé de réception. L'expéditeur supprime le message qu'il a enregistré lors de la réception de l'accusé de réception au message `PUBREL`

Protocole de communication MQTT

QoS 2 : Exactly once



Principales versions du protocole MQTT

Principales versions du protocole MQTT

- MQTT-SN v1.2, standardisé par IBM, 14-11-2013.
  - SN : Sensor Network
  - Variation destinée aux réseaux qui ne sont pas basés sur TCP/IP tel que le `Zigbee`
- MQTT v3.1, standardisé par Eurotech and IBM, 2010.
- MQTT v3.1.1, standardisé par OASIS (Organization for the Advancement of Structured Information Standards), 10-12-2015.
- MQTT v5.0, standardisé par OASIS, 07-03-2019.
  - Ajout de date d'expiration pour les messages
  - Association des alias pour les topics
  - etc.

Brokers en ligne gratuits

Brokers MQTT en ligne gratuits

Broker	Serveur	Port
Mosquitto	iot.eclipse.org test.mosquitto.org	1883 / 8883 1883 / 8883 / 8884
Mosca	test.mosca.io	1883
HiveMQ	broker.hivemq.com	1883
emqx.io	broker.emqx.io	1883 ou 8883 (TCP / TLS)
MoQiatto	maqiatto.com Nécessite l'ouverture d'un compte	1883
flespi	mqtt.flespi.io	443 (SSL) ou 80 (non-SSL)

Les topics et les jokers (wildcards)

Les topics et les jokers (wildcards)

- Les topics sont organisés de manière hiérarchique (comme un système de fichiers)
  - Chaque sub-topic est séparé par le cractère /
- Les noms des topics sont sensibles à la casse
- **Exemples**
  - `<country>/<region>/<town>/<house>/energyConsumption`
  - `<country>/<region>/<town>/<house>/solarEnergy`
  - `<country>/<region>/<town>/<building>/alarmState`

Les topics et les jokers (wildcards)

Les jokers (wildcards)

- Les jokers permettent l'abonnement à plusieurs topics à la fois
  - Ils ne sont utilisés que pour l'abonnement
- + : joker à un niveau remplace un niveau de sujet
  - `<country>/<region>/<town>/<house>/+` remplace `<country>/<region>/<town>/<house>/energyConsumption` et `<country>/<region>/<town>/<house>/solarEnergy`
- # : joker multi-niveaux qui permet de s'abonner à l'ensemble des topics existant à un niveau et aux niveaux plus bas
  - `<country>/<region>/<town>/#` remplace `<country>/<region>/<town>/<house>/energyConsumption`, `<country>/<region>/<town>/<house>/solarEnergy` et `<country>/<region>/<town>/<building>/alarmState`

Les topics et les jokers (wildcards)

Sujets commençant par $

- Les sujets commençant par le symbole $ sont réservés aux statistiques internes du broker MQTT
- Les clients ne peuvent pas publier de messages sur ces sujets
- Pour l'instant, il n'existe pas de normalisation officielle pour ces sujets.
  - Dépend du broker
- **Exemple de sujets**

```
$SYS/broker/clients/connected
$SYS/broker/clients/disconnected
$SYS/broker/clients/total
$SYS/broker/messages/sent
$SYS/broker/uptime
```

Les topics et les jokers (wildcards)

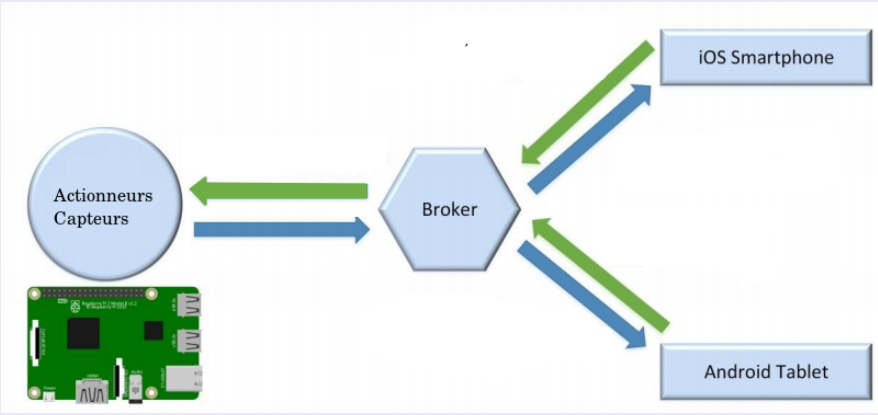
Bonnes pratiques

- Ne jamais utiliser un slash avant (/)
  - Le slash avant introduit un niveau de sujet inutile avec un caractère zéro au début
- N'utilisez jamais d'espaces dans un sujet
- Gardez le sujet court et concis
- Utilisez uniquement des caractères ASCII, évitez les caractères non imprimables
- Intégrer un identifiant unique ou le Client Id dans le sujet
- Ne vous abonnez pas à #

Plan

Objectif

Architecture générale d'une solution IOT



Présentation du projet `Paho MQTT`

Paho MQTT en bref

- Projet opensource développé par la fondation `Eclipse`
- Bibliothèque client Paho MQTT qui implémente les versions 3.1, 3.1.1.1 et 5.0 du protocole MQTT
- API fournit en `C, C ++, Java, Javascript, Python, Go, C#`
- Site officiel : `http://www.eclipse.org/paho`
- Documentation C++ : `https://www.eclipse.org/paho/files/cppdoc/index.html`

paho

Installation de la bibliothèque `Paho MQTT C++`

Installation de la bibliothèque `Paho MQTT C++` sur RPI

- Compilation et installation depuis le code source
- Elle nécessite la compilation et l'installation de la bibliothèque de `Paho C` version 3.1.8 ou supérieure
- Installer les paquets de dépendances
  - Outils de compilation : `build-essential gcc make cmake`
  - Utilisation de SSL : `libssl-dev`
  - Construction de la documentation : `doxygen`

Compilation et installation de la bibliothèque `Paho MQTT C`

- Télécharger le code source
  ```
  git clone https://github.com/eclipse/paho.mqtt.c ; cd
  paho.mqtt.c
  ```
- Compilation avec `cmake`
  ```
  cmake -Bbuild -H. -DPAHO_ENABLE_TESTING=OFF
  -DPAHO_BUILD_STATIC=ON -DPAHO_WITH_SSL=ON
  -DPAHO_HIGH_PERFORMANCE=ON
  ```
- Installation
  ```
  sudo cmake --build build/ --target install
  sudo ldconfig
  ```


Installation de la bibliothèque `Paho MQTT C++`

Compilation et installation de la bibliothèque `Paho MQTT C++`

- Télécharger le code source
  ```
  git clone https://github.com/eclipse/paho.mqtt.cpp
  cd paho.mqtt.cpp
  ```
- Compilation avec `cmake`
  ```
  cmake -Bbuild -H. -DPAHO_BUILD_STATIC=ON
  -DPAHO_BUILD_DOCUMENTATION=TRUE -DPAHO_BUILD_SAMPLES=TRUE
  ```
- Installation
  ```
  sudo cmake --build build/ --target install
  sudo ldconfig
  ```

Utilisation de la Bibliothèque `Paho MQTT`

Connexion : la classe `mqtt::async_client`

- `#include <async_client.h>`
- La classe `mqtt::async_client` permet de communiquer avec un broker en utilisant des méthodes non bloquantes
- Constructeur `async_client (const string &serverURI, const string &clientId, iclient_persistence *persistence=nullptr)`
  - `serverURI` : spécifie l'adresse et le port du broker
  - `clienId` : une chaine choisie pour identifier le client
- La méthode `token_ptr connect()` permet d'établir la connexion. Elle renvoie un pointeur sur un jeton permettant de surveiller l'état de la connexion
- La méthode `token_ptr disconnect()` permet de se déconnecter
- La méthode `bool is_connected()` indique si une connexion est bien établie

Utilisation de la Bibliothèque `Paho MQTT`

Connexion : la classe `mqtt::token`

- `#include <token.h>`
- La classe `mqtt::token` fournit un mécanisme pour surveiller les actions asynchrones
- La méthode `wait()` permet de bloquer la tâche courante jusqu'à que l'action associée au jeton soit terminée

Utilisation de la Bibliothèque `Paho MQTT`

Exemple : connexion à un broker

```
const std::string SERVER_ADDRESS("tcp://localhost:1883");
const std::string CLIENT_ID("paho_cpp_async_subcribe");

    mqtt::async_client cli(SERVER_ADDRESS, CLIENT_ID);

    // Connect to the server
    cout << "Connecting to the MQTT server..." << flush;
    cli.connect()->wait();

...
    std::cout << "\nDisconnecting from the MQTT server..." << std::flush;
        cli.disconnect()->wait();
        std::cout << "OK" << std::endl;
```

Utilisation de la Bibliothèque `Paho MQTT`

Inscription à un sujet

- Pour récevoir les messages d'un sujet, il faut d'abord s'inscrire

```
token_ptr mqtt::async_client::subscribe(const string &topicFilter)
```

  - ☞ `topicFilter` : le sujet avec la possibilité d'inclure des jokers
- Se désinscrire d'un sujet

```
token_ptr mqtt::async_client::unsubscribe(const string &topicFilter)
```

Utilisation de la Bibliothèque `Paho MQTT`

Démarrer la réception des messages

- Démarrer la réception asynchrone des messages et les stocker dans une file

```
void mqtt::async_client::start_consuming()
```

- Arrêter la réception des messages et ignorer tout message non lu

```
void mqtt::async_client::stop_consuming()
```


Utilisation de la Bibliothèque `Paho MQTT`

Lecture des messages

- La lecture bloquante des messages se fait à partir d'une file

```
const_message_ptr mqtt::async_client::consume_message()
```

  - `const_message_ptr` : un pointeur sur l'objet encapsulant le message. Si `nullptr` : échec de lecture

- Lecture non bloquante

```
bool mqtt::async_client::try_consume_message(const_message_ptr * msg)
```

La classe `message`

- Récupérer le nom du sujet d'un message reçu

```
const string& mqtt::message::get_topic() const
```

- Renvoyer sous forme d'une chaine de caractère la charge utile d'un message reçu

```
string mqtt::message::to_string() const
```

Utilisation de la Bibliothèque `Paho MQTT`

Exemple : Réception asynchrone et lecture synchrone des messages

```
const string SERVER_ADDRESS     { "tcp://localhost:1883" };
const string CLIENT_ID          { "paho_cpp_async_consume" };
const string TOPIC                      { "#" };

int main(int argc, char* argv[])   {
        mqtt::async_client cli(SERVER_ADDRESS, CLIENT_ID);
        cli.start_consuming();
        cli.connect()->wait();
        cli.subscribe(TOPIC)->wait();
        const_msg_ptr msg;
        while (true) { // Loop to consume read messages
            bool reception = cli.try_consume_message(&msg);
            if (reception==true) break;
                  cout << msg->get_topic() << ": " << msg->to_string() << endl;

            cli.unsubscribe(TOPIC)->wait();
            cli.stop_consuming();
            cli.disconnect()->wait();
       }

       return 0;
}
```


Utilisation de la Bibliothèque `Paho MQTT`

Envoie de messages

- On peut créer une instance de la classe `message` pour le message à envoyer

```
message_ptr mqtt::make_message(string_ref topic,binary_ref payload)
```

  - `topic` : le sujet
  - `payload` : la charge utile du message

- Publier un message

```
delivery_token_ptr mqtt::async_client::publish(message_ptr msg)
delivery_token_ptr mqtt::async_client::publish(string_ref topic, binary_ref payload)
```

  - `topic` : le sujet
  - `payload` : la charge utile du message

Utilisation de la Bibliothèque `Paho MQTT`

Exemple : Réception asynchrone et lecture synchrone des messages

```
const string SERVER_ADDRESS {"tcp://localhost:1883"};
const string CLIENT_ID {"paho_cpp_async_publish"};
const string TOPIC { "hello" };
const char* PAYLOAD1 = "Hello World!";
const char* PAYLOAD2 = "Hi there!";

int main(int argc, char* argv[])
{
    mqtt::async_client client(SERVER_ADDRESS, CLIENT_ID);
    client.connect()->wait();
    mqtt::message_ptr pubmsg = mqtt::make_message(TOPIC, PAYLOAD1);

    client.publish(pubmsg)->wait_for(TIMEOUT);
    cout << "First message was sent..." << endl;
    // Now try with itemized publish
    mqtt::delivery_token_ptr pubtok;
    pubtok = client.publish(TOPIC, PAYLOAD2);
    pubtok->wait();
    client.disconnect()->wait();
    cout << "\nDisconnected" << endl;

    return 0;
}
```


MERCI POUR VOTRE ATTENTION

Questions ?