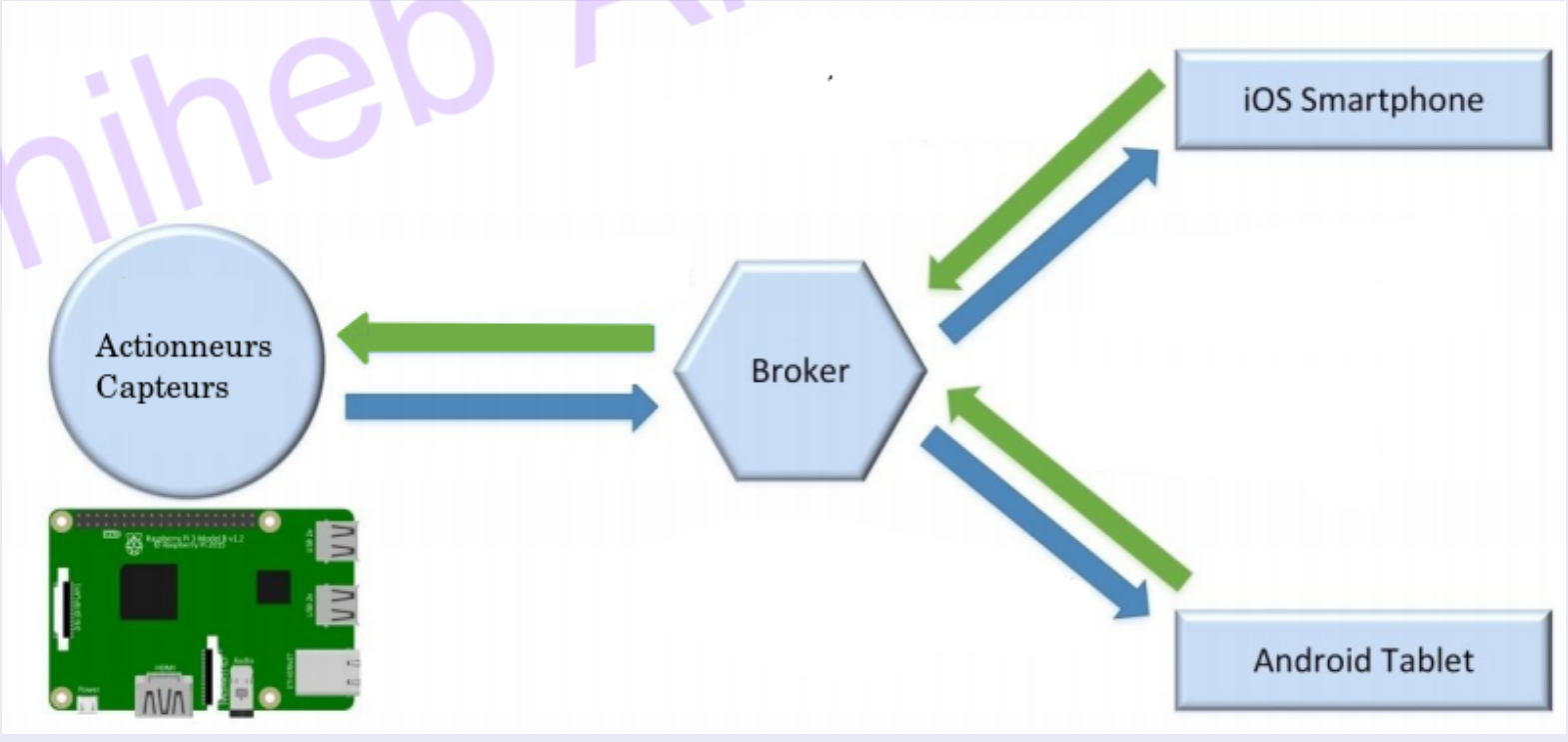
Plan

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`

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
- La méthode `connect_response get_connect_response() const` renvoie la réponse de l'opération de connexion. Si l'opération n'est pas encore terminée, la tâche en cours se bloquera jusqu'à ce que le résultat soit disponible.

Connexion : la classe `mqtt::connect_response`

- `#include <server_response.h>`
- Elle fournit des informations sur une connexion établie
- Vérifier si une session existait déjà pour un client sur le serveur : le serveur a une session persistante stockée pour le client, étant donné l'identifiant du client spécifié dans le message de connexion.

```
bool mqtt::connect_response::is_session_present () const
```

- Renvoyer la version MQTT utilisée pour la connexion

```
int mqtt::connect_response::get_mqtt_version() const
```

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);
try {
    // Connect to the server
    cout << "Connecting to the MQTT server..." << flush;
    cli.connect()->wait();
} catch(const mqtt::exception &exc) {
    cerr << "\n  " << exc << endl;
    return 1;
}
...
try {
    std::cout << "\nDisconnecting from the MQTT server..." << std::flush;
        cli.disconnect()->wait();
        std::cout << "OK" << std::endl;
}
catch (const mqtt::exception& exc) {
        std::cerr << exc << std::endl;
        return 1;
}
```

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,int qos,const
      subscribe_options &opts=subscribe_options(),const properties &props=properties())
```

  - `topicFilter` : le sujet avec la possibilité d'inclure des jokers
  - `qos` : le degré de la qualité du service
  - `opts` : des options d'inscription avec MQTT v5
  - `props` : les propriétés de MQTT v5

- Se désinscrire d'un sujet

```
token_ptr mqtt::async_client::unsubscribe(const string &topicFilter,const properties &props=
      properties())
```

  - `topicFilter` : le sujet avec la possibilité d'inclure des jokers
  - `props` : les propriétés de MQTT v5

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                      { "#" };
const int  QOS = 1;
int main(int argc, char* argv[])   {
        mqtt::async_client cli(SERVER_ADDRESS, CLIENT_ID);
        try {   // Start consumer before connecting to make sure to not miss messages
                cli.start_consuming();
                auto rsp = cli.connect()->get_connect_response();
                // If there is no session present, then we need to subscribe, but if
                // there is a session, then the server remembers us and our subscriptions.
                if (!rsp.is_session_present()) cli.subscribe(TOPIC, QOS)->wait();
                while (true) { // Loop to consume read messages
                        auto msg = cli.consume_message();
                        if (!msg) break;
                        cout << msg->get_topic() << ": " << msg->to_string() << endl;
                }
                if (cli.is_connected()) {
                        cli.unsubscribe(TOPIC)->wait();
                        cli.stop_consuming();
                        cli.disconnect()->wait();
                }
                else cout << "\nClient was disconnected" << endl;
       }
        catch (const mqtt::exception& exc) {
                cerr << "\n  " << exc << endl;
                return 1;
        }
        return 0;
}
```

Utilisation de la Bibliothèque `Paho MQTT`

Envoie de messages

- Il faut créer une instance de la classe `message` pour le message à envoyer

```
message_ptr mqtt::make_message(string_ref topic,binary_ref payload)
message_ptr mqtt::make_message(string_ref topic,const void *payload,size_t len)
```

  - `topic` : le sujet
  - `payload` : la charge utile du message
  - `len` : taille de message en nombre d'octets

- Se désinscrire d'un sujet

```
delivery_token_ptr mqtt::async_client::publish(string_ref topic, binary_ref payload)
delivery_token_ptr mqtt::async_client::publish(string_ref topic, binary_ref payload,int qos,bool
       retained)
```

  - `topicFilter` : le sujet avec la possibilité d'inclure des jokers
  - `props` : les propriétés de MQTT v5

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!";
const int QOS=1;
const auto TIMEOUT = std::chrono::seconds(10);
int main(int argc, char* argv[])
{
    mqtt::async_client client(SERVER_ADDRESS, CLIENT_ID);
    try {
        client.connect()->wait();
        mqtt::message_ptr pubmsg = mqtt::make_message(TOPIC, PAYLOAD1);
        pubmsg->set_qos(QOS);
        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, strlen(PAYLOAD2), QOS, false);
        cout << "  ...with token: " << pubtok->get_message_id() << endl;
        cout << "  ...for message with " << pubtok->get_message()->get_payload().size()<< " bytes" <<
              endl;
        pubtok->wait_for(TIMEOUT);
        client.disconnect()->wait();
        cout << "\nDisconnected" << endl;
    }
    catch (const mqtt::exception& exc) {
        cerr << exc.what() << endl;
        return 1;
    }
    return 0;
}
```