Faire communiquer son Arduino avec un appareil Android

Cet article est un article "invité", rédigé par Sébastien (vous pourrez retrouver d'autres de ces réalisations à la fin de l'article).

Dans ce tutoriel porté sur Android et Arduino, nous allons connecter ces deux mondes géniaux (n'est-ce pas ?). Nous allons illustrer ce tutoriel par une application concrète ! A la fin, vous aurez une application Android avec deux boutons. Ces deux boutons permettront soit d'allumer une led connecté à l'Arduino, soit de l'éteindre. Pour cela, vous allez avoir besoin du matériel suivant :

1. Un Arduino avec un shield Ethernet (ou wifi) ainsi qu'une led et sa résistance
2. Un appareil sous Android
3. Un ordinateur

L'ordinateur jouera le rôle du serveur : il recevra les messages envoyés par l'application Android et les renverra ensuite à l'Arduino. Sachez qu'il est tout à fait possible d'envoyer des messages de l'Arduino à Android.

Au niveau des compétences requises, des connaissances basiques en programmation Android et en Arduino suffiront !

Si vous voulez voir le résultat que vous obtiendrez à la fin de ce tutoriel, je vous invite à vous rendre à la section "Résultat" où vous trouverez une vidéo !

Le protocole MQTT

Pour faire communiquer l'appareil Android avec l'Arduino, nous allons utiliser un langage commun : le protocole MQTT. L'avantage de ce protocole, c'est qu'il existe une librairie pour Android et une librairie pour Arduino. Ainsi nos deux mondes communiqueront de la même façon !

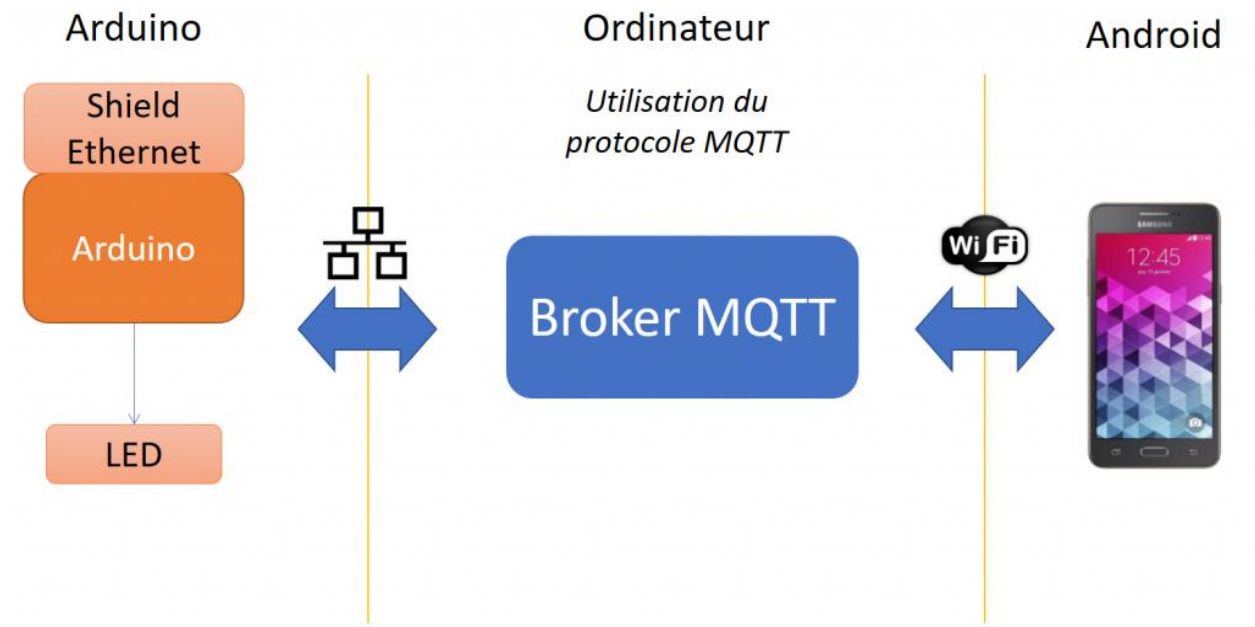
Le protocole MQTT est un protocole de messagerie de type Publish-Subscribe basé sur le protocole TCP/IP (d'où la nécessité du shield Ethernet). Le protocole se décompose donc en trois grandes parties :

- Les « publishers » : ils envoient un ou des message(s) sur un ou plusieurs « Topic »
- Le « broker » MQTT : il fait le lien entre les « publishers » et les « subscribers »
- Les « subscribers » : ils s'abonnent à un ou plusieurs « Topic ». Lorsque qu'un message est publié sur un topic, tous les subscribers de ce topic reçoivent le message

Le broker MQTT permet de faire le lien entre le publisher (appareil Android) et le subscriber (Arduino). Dans notre exemple il n'y a qu'un seul publisher et qu'un seul subscriber, mais il pourrait y en avoir plus.

Le broker MQTT : le nœud central

Le broker MQTT est le cœur de notre architecture (voir image ci-dessous). Il va permettre de faire communiquer l'appareil Android avec l'Arduino.



Architecture

Rassurez-vous, vous n'aurez pas besoin de développer votre propre broker, il en existe déjà. Mais si cela peut satisfaire votre soif de développement, ça fera un bon exercice !

Celui que je vais utiliser tout au long de ce tuto, et que je vous conseille, est le broker « Mosquitto ». C'est un broker open source. Il suffit de télécharger l'exécutable sur le site officiel et de l'installer. Pensez à lire le fichier readme.txt qui se trouve à la racine du dossier d'installation, car il vous précise les dépendances à installer. Il vous faudra notamment installer pthread, openSSL (les liens vous sont fournis dans le readme) ainsi que l'ajout de quelques dll. Pour information, il vous faudra placer les dll dans le dossier de l'installation.

Pour le lancer, une simple ligne de commande suffit. Dans mon cas ça donne (sachant que je suis sous Windows) :

```
"C:\Program Files (x86)\mosquitto\mosquitto.exe" -v -p 1883
```

Il suffit donc de lancer l'exécutable qui a été installé, avec les options :

- `-v` : mode verbose
- `-p 1883` : on va utiliser le port de l'ordinateur 1883

Vous voilà avec un broker qui tourne et qui est prêt à recevoir et envoyer des messages !

Connectez votre appareil Android au broker MQTT

On va à présent créer notre application Android ! Elle sera composée de deux boutons : un pour allumer la LED connectée à notre Arduino, et l'autre pour l'éteindre.

Initialisez votre projet Android

On va dans un premier temps initialiser notre projet pour qu'il puisse communiquer avec le protocole MQTT. Pour cela, il faut :

- Ajouter les dépendances dans le fichier gradle:
  - Au tout début de votre fichier gradle :

```
    repositories {
        maven { url 'https://repo.eclipse.org/content/repositories/paho-snapshots/' }
    }
```

```
- Dans la partie *dependencies* de votre fichier gradle:
```

```
    compile('org.eclipse.paho:org.eclipse.paho.android.service:1.0.3-SNAPSHOT') {
        exclude module: 'support-v4'
    }
    compile 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.0.3-SNAPSHOT'
```

- Ajouter le service dans le Manifest entre les balises `<Application>` (si vous l'oubliez, vous n'aurez aucune erreur mais il vous sera impossible de vous connecter au broker et d'envoyer/recevoir des messages. Si je vous dis ça c'est que j'ai passé quelques heures à débogger mon application pour ce simple oublie) :

```
<service android:name="org.eclipse.paho.android.service.MqttService" ></service>
```

Ajouter les permissions dans le Manifest (en dessous de la balise fermante : </application>) :

```
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

Connectez vous au broker MQTT

Votre projet est enfin prêt ! Nous allons maintenant créer la méthode qui va nous permettre de nous connecter au broker. Voici son prototype :

```
public void connect(String address, String port)
```

- Le premier argument `address` , correspond à l'adresse IP où se situe le broker. Pour connaitre l'adresse IP de votre ordinateur (sous windows), ouvrez une console et tapez la commande `ipconfig`et cherchez la ligne *Adresse IPv4* . L'adresse est de la forme : `192.168.1.xxx`
- Le second argument `port` correspond au port utilisé par votre broker MQTT.

Dans la partie précédente, nous avons lancé notre broker MQTT sur le port 1883.

Le corps de la méthode est assez simple. Je vous laisse le découvrir par vous même. Si la connexion est réussi, la méthode `onSuccess` est appelée, sinon la méthode `onFailure` sera appelée. **Attention !** Vous aurez une erreur sur la dernière ligne, quand on appelle la méthode `setCallback` , ainsi que sur la ligne`subscribe(topic);` . Commentez les, nous y reviendrons plus tard.

```
private MqttAndroidClient client = null;

public void connect(String address, String port) {
    String clientId = MqttClient.generateClientId(); // génère un ID
    client = new MqttAndroidClient(getApplicationContext(), "tcp://" + address +
":" + port, clientId);

    try {
        IMqttToken token = client.connect(); // on tente de se connecter
        token.setActionCallback(new IMqttActionListener() {
            @Override
            public void onSuccess(IMqttToken asyncActionToken) {
                // Nous sommes connecté
                System.out.println("On est connecté !");
                subscribe(topic); // ligne à commenter pour le moment
```

```
            }

            @Override
            public void onFailure(IMqttToken asyncActionToken, Throwable
exception) {
                // Erreur de connexion : temps de connexion trop long ou problème
de pare-feu
                System.err.println("Echec de connection !");
            }
        });
    } catch (MqttException e) {
        e.printStackTrace();
    }

    client.setCallback(new MqttCallbackHandler()); // ligne à commenter pour le
moment
}
```

Il ne vous reste plus qu'à appeler cette méthode dans la méthode `onResume` et vous serez connecté à votre broker ! Pensez à mettre votre propre adresse IP et à avoir le broker qui tourne sur votre ordinateur pour que la connexion puisse se faire.

```
@Override
public void onResume() {
    super.onResume();
    connect("192.168.1.17", "1883");
}
```

Bien sûr, si vous vous connectez au broker dans la méthode `onResume` , il faut penser à vous déconnecter dans la méthode `onPause` . Je ne détaille pas le code mais je vous le donne (il est assez simple) :

```
@Override
public void onPause() {
    super.onPause();
    disconnect();
}

public void disconnect() {
    if (client == null) {
        return;
    }
    try {
        IMqttToken disconToken = client.disconnect();
        disconToken.setActionCallback(new IMqttActionListener() {
            @Override
            public void onSuccess(IMqttToken asyncActionToken) {
                // Nous nous sommes correctement déconnecté
            }

            @Override
```

```
            public void onFailure(IMqttToken asyncActionToken,
                                  Throwable exception) {
                // Quelque chose c'est mal passé, mais on est probablement
déconnecté malgré tout
            }
        });
    } catch (MqttException e) {
        e.printStackTrace();
    }
}
```

Envoyez un message sur un topic

Pour envoyer un message, nous aurons besoin de deux choses : le topic sur lequel envoyer le message et le message lui-même. Pour ce faire, nous allons utiliser la méthode `publish` de notre client (de classe`MqttAndroidClient`). Cette méthode prend en paramètre un topic et un message, ça tombe bien ! Pour notre exemple, nous allons publier un message sur le topic *LEDArduino* . Ainsi, tous les subscribers abonnés à ce topic receveront le message. Voici le code :

```
private final String topic = "LEDArduino";

public void sendMsg(String msg) {
    MqttMessage message = new MqttMessage();
    message.setPayload(msg.getBytes());
    try {
        client.publish(topic, message);
    } catch (MqttException e) {
        e.printStackTrace();
    }
}
```

Recevoir un message

Pour notre exemple, notre application Android n'aura pas besoin de recevoir de message. Vous pouvez passer cette partie si vous le souhaitez. Sachez que c'est dans cette partie que nous allons pouvoir décommenter le code précédemment commenté dans notre méthode `connect` .

Nous allons avoir besoin de deux choses pour recevoir des messages : souscrire à des topics et avoir un callback qui sera appelé automatiquement quand un message aura été reçu par l'application.

Pour souscrire à un topic, nous allons faire appel à la méthode `subscribe` de notre client. Cette méthode prend deux arguments :

- topic : le topic sur lequel on veut s'abonner
- QOS (quality of service) : peut prendre trois valeurs :
    - 0 : Le message sera délivré qu'une seule fois, sans confirmation
    - 1 : Le message sera délivré au moins une fois, avec confirmation

- 2 : Le message sera délivré exactement une fois, avec vérification en quatre étapes

Voici ce que donne notre méthode `subscribe` . Une fois inséré dans votre code, vous pouvez décommenter le code qui l'appele dans la méthode `connect` .

```
private static int QOS = 0;

public void subscribe(final String topic) {
    try {
        IMqttToken subToken = client.subscribe(topic, QOS);
        subToken.setActionCallback(new IMqttActionListener() {
            @Override
            public void onSuccess(IMqttToken asyncActionToken) {
                // On a bien souscrit au topic
                System.out.println("onSuccess subscribe topic " + topic);
            }
            @Override
            public void onFailure(IMqttToken asyncActionToken,
                                  Throwable exception) {
                // La souscription n'a pas pu se faire, peut être que
l'utilisateur n'a pas
                // l'autorisation de souscrire à ce topic
            }
        });
    } catch (MqttException e) {
        e.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Passons maintenant à la classe qui va permettre d'avoir nos callback ! Elle doit implémenter l'interface`MqttCallback` . Une fois cette classe implémenter, vous allez pouvoir décommenter la méthode `setCallback`de la méthode `connect` . Dans notre exemple, la classe qui implémente cette interface se nomme`MqttCallbackHandler` . Je vous donne le code basique de la classe. Encore une fois, il est facile à comprendre :

```
import android.content.Context;

import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttMessage;

public class MqttCallbackHandler implements MqttCallback {

    /** {@link Context} for the application used to format and import external
strings**/
    private Context context;
```

```
    /** Client handle to reference the connection that this handler is attached
to**/
    private String clientHandle;

    public MqttCallbackHandler()
    {
    }

    /**
     * @see
org.eclipse.paho.client.mqttv3.MqttCallback#connectionLost(java.lang.Throwable)
     */
    @Override
    public void connectionLost(Throwable cause) {
    }

    /**
     * @see
org.eclipse.paho.client.mqttv3.MqttCallback#messageArrived(java.lang.String,
|org.eclipse.paho.client.mqttv3.MqttMessage)
     */
    @Override
    public void messageArrived(String topic, MqttMessage message) throws Exception
{
        String message_str = new String(message.getPayload(), "UTF-8");
        System.out.println("message arrivé str " + topic + " " + message_str);
    }

    /**
     * @see
|org.eclipse.paho.client.mqttv3.MqttCallback#deliveryComplete(org.eclipse.paho.cli
ent.mqttv3.IMqttDeliveryTok|en)
     */
    @Override
    public void deliveryComplete(IMqttDeliveryToken token) {
        // Do nothing
    }
}
```

Créez votre application

Nous avons maintenant toutes les bases qu'il nous faut pour créer notre application qui va communiquer avec notre Arduino. Il ne nous reste plus qu'à créer deux boutons : un bouton qui va permettre d'allumer la LED et un bouton pour l'éteindre. Si on clique sur le bouton qui doit allumer la LED, on va envoyer le message "ON" en appelant simplement la méthode `sendMsg` précédemment écrite. Si on clique sur le bouton pour éteindre la LED, on va envoyer le message "OFF". Simple non ? Je vous laisse le faire, et si besoin je vous donne mon code Java et XML :

```
public void AllumerLed(View v) {
    sendMsg("ON");
}
public void EteindreLed(View v) {
    sendMsg("OFF");
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:orientation="vertical">

        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:onClick="AllumerLed"
            android:text="Que la lumière soit !" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="EteindreLed"
            android:text="J'ai eu assez de lumière, merci !" />

    </LinearLayout>

</RelativeLayout>
```

Une fois votre application terminée, elle devrait pouvoir se connecter au broker MQTT (pensez à lancer le broker sur votre ordinateur et à avoir votre appareil Android en réseau local avec votre ordinateur, par exemple en étant sur le même réseau wifi, et envoyer des messages. Lorsque vous cliquez sur un bouton sur Android, vous devriez voir dans la console que le broker MQTT a reçu un message (voir image ci-dessous).

```
C:\Users\Sébastien>"C:\Program Files (x86)\mosquitto\mosquitto.exe" -v -p 1883
1501603843: mosquitto version 1.4.9 (build date 08/06/2016 11:59:29.51) starting
1501603843: Using default config.
1501603843: Opening ipv6 listen socket on port 1883.
1501603843: Opening ipv4 listen socket on port 1883.
1501603846: New connection from 192.168.1.11 on port 1883.
1501603846: New client connected from 192.168.1.11 as paho3054682707210 (c1, k60).
1501603846: Sending CONNACK to paho3054682707210 (0, 0)
1501603846: Received SUBSCRIBE from paho3054682707210
1501603846:     LEDArduino (QoS 0)
1501603846: paho3054682707210 0 LEDArduino
1501603846: Sending SUBACK to paho3054682707210
1501603850: Received PUBLISH from paho3054682707210 (d0, q1, r0, m2, 'LEDArduino', ... (2 bytes))
1501603850: Sending PUBACK to paho3054682707210 (Mid: 2)
1501603850: Sending PUBLISH to paho3054682707210 (d0, q0, r0, m0, 'LEDArduino', ... (2 bytes))
1501603852: Received PUBLISH from paho3054682707210 (d0, q1, r0, m3, 'LEDArduino', ... (3 bytes))
1501603852: Sending PUBACK to paho3054682707210 (Mid: 3)
1501603852: Sending PUBLISH to paho3054682707210 (d0, q0, r0, m0, 'LEDArduino', ... (3 bytes))
```

Console Mosquitto

Le message reçu par le broker qui fait 2 bytes est le message "ON" et le message reçu qui fait 3 bytes est le message "OFF".

Connectez votre Arduino au broker MQTT

Initialiser votre projet Arduino

Comme pour Android, il vous faut d'abord télécharger la librairie qui va vous permettre de communiquer avec le protocole MQTT. Rendez-vous à l'adresse suivante : https://eclipse.org/paho/clients/c/embedded/ et descendez jusqu'à la section Arduino. Vous allez avoir un lien pour télécharger la librairie. Pour rappel, pour ajouter la librairie dans Arduino (téléchargée au format ZIP), il vous faut cliquer (dans l'IDE d'Arduino) sur "Croquis->Inclure une bibliothèque->Ajouter la bibliothèque .ZIP".

Une fois la librairie ajoutée, vous pouvez utiliser le protocole MQTT ! Sachez qu'un exemple très complet vous est fourni avec la librairie. Vous le trouverez dans votre dossier Arduino->libraries->MQTTClient->example. Vous ne devriez pas avoir de mal à le lire car il reprends ce que nous avons vu avec Android (connect, subscribe, …), mais façon Arduino. Sachez d'ailleurs que je me suis grandement inspiré de cet exemple pour écrire le code que nous allons voir ensemble.

Le câblage

Pour pouvoir utiliser le protocole MQTT sur votre Arduino, il vous faut un shield ethernet (ou wifi). En effet, le protocole MQTT étant basé sur le protocole TCP/IP, le shield ethernet (ou wifi) est requis. Il vous faudra également relier votre Arduino à votre réseau local en connectant le shield

à votre box via un câble ethernet (ou via wifi). Une fois ceci fait, nous allons pouvoir brancher notre LED.

Concernant la LED, je l'ai connecté au PIN 2 de l'Arduino, en série avec sa résistance (ça serait bête de la griller !). J'ai connecté l'anode sur le pin 5V de l'Arduino et la cathode sur le PIN 2. Ainsi, quand le PIN 2 sera à 0V, la LED s'allumera, et quand le PIN sera à 5V, la LED s'éteindra.

Communiquez avec le protocole MQTT

Nous allons à présent passer au code pour connecter l'Arduino au broker MQTT et pour recevoir les messages. Je ne vous montrerai pas comment envoyer un message. Vous trouverez, si besoin, les quelques lignes qui le permettent dans la fonction loop() de l'exemple fourni avec la librairie.

Les includes nécessaires sont les suivants

```
#include <SPI.h>         // Pour communiquer avec le shield Ethernet
#include <Ethernet.h>    // Pour la partie Ethernet, evidemment !
#include <IPStack.h>     // Permet de gérer la couche IP
#include <Countdown.h>   // Timer utilisé par le protocole MQTT
#include <MQTTClient.h> // Permet de gérer le protocole MQTT
```

Les variables nécessaires

Dont le pin de la LED et le topic auquel on veut souscrire (qui sera le même que celui sur lequel on envoie les messages côté Android) :

```
const int led = 2; // pin de la LED

EthernetClient c; // remplacez par un YunClient si vous utilisez Yun
IPStack ipstack(c);
MQTT::Client<IPStack, Countdown, 50, 1> client = MQTT::Client<IPStack, Countdown,
50, 1>(ipstack);

byte mac[] = { 0x00, 0x11, 0x22, 0x33, 0x44, 0x55 };  // remplacer par l'adresse
MAC de votre shield ethernet
const char* topic = "LEDArduino"; // le topic utilisé pour communiquer
```

Les fonctions `connect` et `messageArrived`

Nous allons créer la fonction qui sera appelée quand un message sera reçu. On indiquera plus tard au client MQTT que c'est cette fonction qu'il faudra qu'il appelle quand on recevra un message appartenant au topic*LEDArduino* . C'est dans cette fonction que nous gérons l'allumage et l'extinction de la LED en fonction du message reçu.

```
void messageArrived(MQTT::MessageData& md)
{
```

```
  MQTT::Message &message = md.message;

  char* msg = (char*) message.payload; // on recupere le message
  msg[message.payloadlen] = 0; // indique la fin du char*

  if (strcmp(msg, "ON") == 0) // on a reçu le message "ON"
    digitalWrite(led, LOW);   // on allume la LED
  else if (strcmp(msg, "OFF") == 0) // on a reçu le message "OFF"
    digitalWrite(led, HIGH);        // on éteint la LED
}
```

Maintenant il va falloir nous connecter au broker MQTT et souscrire au topic qui nous intéresse. Pour cela, nous allons créer la méthode `connect` , exactement de la même façon que nous avons fait pour Android. Pensez à remplacer la variable hostname par votre propre adresse IP (comme nous avons fait pour l'application Android).

```
void connect()
{
  char hostname[] = "192.168.1.17"; // IP où se trouve le broker MQTT
  int port = 1883; // port utilisé par le broker

  int rc = ipstack.connect(hostname, port);
  if (rc != 1)
  {
    Serial.print("rc from TCP connect is ");
    Serial.println(rc);
  }

  Serial.println("MQTT connecting");
  MQTTPacket_connectData data = MQTTPacket_connectData_initializer;
  data.MQTTVersion = 3;
  data.clientID.cstring = (char*)"arduino-id";
  rc = client.connect(data);
  if (rc != 0)
  {
    Serial.print("rc from MQTT connect is ");
    Serial.println(rc);
  }
  Serial.println("MQTT connected");

  rc = client.subscribe(topic, MQTT::QOS0, messageArrived);    // le client
souscrit au topic
  if (rc != 0)
  {
    Serial.print("rc from MQTT subscribe is ");
    Serial.println(rc);
  }
  Serial.println("MQTT subscribed");
}
```

Finalisation du programme

Voilà ! Toutes nos fonctions sont prêtes, il ne nous reste plus qu'à implémenter les fonctions setup() et loop() ! Pensez à appeler `client.yield(1000)` dans la fonction loop() sinon votre Arduino ne recevra aucun message !

```
void setup() {
  pinMode(led, OUTPUT);
  digitalWrite(led, HIGH); // on initialise la LED en l'éteignant

  Serial.begin(9600);
  Ethernet.begin(mac);

  connect();
}

void loop() {
   if (!client.isConnected())
     connect();

   delay(500);
   client.yield(1000); // permet la reception des messages
}
```

Le résultat !

Nous allons à présent voir le résultat tant attendu après un travail acharné entre le monde d'Android, d'Arduino et sans oublier le protocole MQTT géré par le broker qui tourne sur votre ordinateur.

- Lancez le broker sur votre ordinateur
- Lancez l'application Android (soyez sûr que vous ayez toujours la bonne adresse IP pour le broker MQTT et que vous soyez connecté en réseau local avec le broker)
- Démarrez votre Arduino connecté à votre réseau local

Amusez-vous à cliquer sur les boutons qui permettent d'allumer et d'éteindre la LED de votre Arduino !

Nous avons vu dans ce tutoriel comment faire communiquer un appareil Android avec un Arduino.

Nous nous sommes contenté, pour illustrer ce tuto, d'envoyer un message à l'Arduino pour qu'il allume ou éteigne une LED. Mais les possibilités sont infinies ! Vous pouvez envoyer des messages d'un Arduino à un appareil Android. Vous pouvez aussi faire communiquer autant d'Arduino et d'appareil d'Android que vous le voulez simultanément !

J'espère que ce tutoriel vous a plu ! Je vous laisse libre imagination pour intégrer la communication Android/Arduino dans vos futurs projets !