Utiliser le broadcast UDP pour obtenir l'adresse IP d'un serveur local

par Sylvain Berfini (Mes tutoriels) (Blog)

Date de publication : 27/06/2011

Dernière mise à jour :

L'objectif de ce tutoriel est de vous apprendre à implémenter un détecteur de serveur en attente d'une connexion cliente sur le même réseau Wi-Fi que celui auquel vous êtes connecté, le tout dans une application Android.

I - Prérequis

Ce tutoriel étant clairement orienté développement avancé, je vous conseille fortement d'avoir de bonnes bases de réseau. En effet, la connaissance d'Android ne sera pas forcément nécessaire, mais les termes UDP, TCP, broadcast, etc. doivent vous être familiers.

II - Principe et intérêt

Imaginons que votre application permette à (au moins) deux utilisateurs d'interagir ensemble par le biais d'une connexion Internet dispensée par une connexion à un réseau Wi-Fi. Cela peut être le cas pour un mode multijoueurs d'un jeu, ou de la vidéoconférence par exemple. Cependant, vous n'avez peut-être pas la possibilité d'avoir un serveur externe dont le rôle serait de répertorier les serveurs attendant une connexion cliente. Dans ce cas-là, c'est l'utilisateur qui doit faire le travail de recherche du serveur, puis initier la connexion vers lui. Et cela peut être (c'est très souvent le cas) un problème pour un néophyte en informatique que de rentrer une adresse IP, quand bien même les instructions sont claires. Nous allons donc voir comment remédier à ce problème.

Quel est le principe permettant d'éviter tout ce travail à faire, que ce soit par un serveur tiers ou l'utilisateur ? Eh bien nous allons lancer un message en broadcast (par le biais du protocole UDP, TCP supportant mal cela), c'est-à-dire à toutes les machines présentes sur le même réseau que vous, et attendre une réponse de l'une d'elles confirmant que c'est celle que l'on cherche. Notre travail va donc être de créer un serveur UDP en attente d'une requête d'un côté, et de l'autre implémenter un client UDP balançant son message à tout le monde et attendant un retour de quelqu'un d'autre (le broadcast va faire qu'il va recevoir son propre message). Voyons comment faire ça.

En choisissant UDP on s'expose à une perte de paquet, à savoir que notre requête ou notre réponse ne parviendra pas forcément à son destinataire. Il vous faudra donc penser à prendre cela en compte en cas d'échec de la détection d'un serveur.

Attention, nous allons étudier ici une solution qui ne résoud que le problème de la détection de l'IP d'une application hôte attendant une connexion sur un réseau Wi-Fi (et non 3G) local. A vous d'adapter cette solution pour qu'elle résolve un problème différent.

III - Implémentation du client

III-1 - Récupérer l'adresse de broadcast

Nous allons créer une fonction dont le rôle sera de générer, en fonction de l'adresse IP associée à la connexion Wi-Fi, l'adresse de broadcast du réseau. Cette méthode ne prendra rien en paramètre et retournera un objet de type InetAddress. Voici son code.

Génération de l'adresse de broadcast

```
private InetAddress getAdresseBroadcast()
{
    WifiManager wifiManager = (WifiManager) mContext.getSystemService(Context.WIFI_SERVICE);
    DhcpInfo dhcp = wifiManager.getDhcpInfo();

    int broadcast = (dhcp.ipAddress & dhcp.netmask) | ~dhcp.netmask;
    byte[] quads = new byte[4];
    for (int k = 0; k < 4; k++)
      quads[k] = (byte) ((broadcast >> k * 8) & 0xFF);
    return InetAddress.getByAddress(quads);
}
```

Je ne m'aventurerai pas plus en détail dans l'explication du code, le but ici étant de se focaliser sur l'aspect Android et non sur l'aspect réseau.

Utiliser le service système WIFI_SERVICE requiert l'ajout de d'une permission dans le manifest.xml de votre application : android.permission.ACCESS_WIFI_STATE

III-2 - Envoyer notre requête à tout le monde, en attendant une réponse

Maintenant que nous savons quelle adresse IP donner à notre trame UDP à envoyer, construisons-la.

Génération de la trame UDP

```
private DatagramPacket envoyerTrameUDP(String requete, int port) throws Exception
{
 DatagralSocket socket = new DatagramSocket(mPort);
 socket.setBroadcast(true);
 InetAddress broadcastAdress = getAdresseBroadcast();
 DatagramPacket packet = new DatagramPacket(requete.getBytes(), requete.length(), broadcastAdress,
 port);
 socket.send(packet);

 byte[] buf = new byte[1024];
 packet = new DatagramPacket(buf, buf.length);
 socket.setSoTimeout(TIMEOUT_RECEPTION_REPONSE);

 String monAdresse = getMonAdresseIP();
 socket.receive(packet);
 while (packet.getAddress().getHostAddress().contains(monAdresse))
 {
  socket.receive(packet);
 }

 socket.close();

 return packet;
}
```

Quelques explications sur ce bout de code s'imposent. Tout d'abord, nous créons un objet nommé socket de type DatagramSocket, qui est le type de socket utilisé pour faire de l'UDP. Ensuite nous créons notre trame UDP en fonction de la requête que l'on veut envoyer, et sur le port du serveur.

Le port passé ici en paramètre doit être le même que celui utilisé par le serveur, cf. plus bas.

Puis nous envoyons notre trame en broadcast et attendons une réponse. Quand une réponse nous parvient, on vérifie qu'il ne s'agit pas du message que nous venons d'envoyer, le broadcast faisant que l'expéditeur reçoit également sont message. Si l'adresse IP de l'expéditeur de message reçu est différente de la notre, alors je considère qu'il existe un serveur sur le réseau ayant répondu à ma requête.

Rien ne vous empêche ici de faire une vérification plus poussée, comme par exemple en analysant le message reçu pour qu'il corresponde à la réponse que le serveur est sensé fournir à votre requête, comme Ping - Pong.

J'utilise ici une constante, nommée TIMEOUT_RECEPTION_REPONSE. À vous de la définir comme bon vous semble (en millisecondes). Elle permet de lever une exception après un certain temps passé sans réponse. Concrètement, si vous levez cette exception, vous pouvez admettre qu'il n'y a pas de serveur disponible. J'appelle également une méthode à moi, getMonAdresseIP(), qui me renvoie une chaîne de caractères (mon IP en fait).

Obtention de l'adresse IP de notre terminal

```
public String getMonAdresseIP()
{
    try
    {
        for (Enumeration<NetworkInterface> en = NetworkInterface.getNetworkInterfaces();
 en.hasMoreElements();)
```

Obtention de l'adresse IP de notre terminal

```
        {
            NetworkInterface intf = en.nextElement();
            for (Enumeration<InetAddress> enumIpAddr = intf.getInetAddresses();
 enumIpAddr.hasMoreElements();)
            {
                InetAddress inetAddress = enumIpAddr.nextElement();
                if (!inetAddress.isLoopbackAddress())
                 return inetAddress.getHostAddress();
            }
        }
    }
    catch (SocketException e)
    {
        e.printStackTrace();
    }
    return null;
}
```

Petite explication rapide sur ce code. Pour chaque interface réseau de mon terminal, je récupère son adresse IP associée. Une fois que j'en ai trouvé une qui n'est pas l'adresse de loopback, je la renvoie considérant que j'ai la bonne. Vous pouvez toutefois vérifier qu'il s'agisse bien de l'adresse de la carte Wi-Fi (et non celle de la 3G) le cas échéant.

Vous aurez remarqué que notre fonction envoyerTrameUDP() renvoie un DatagramPacket, celui qu'elle à reçu en réponse à sa requête. Il ne nous reste plus qu'à appeler packet.getAddress().getHostAddress() pour connaître l'adresse IP du serveur qui nous intéresse.

Maintenant que nous avons l'adresse IP du serveur UDP (qui est la même que celle du serveur TCP ou UDP qui nous intéresse), il ne nous reste plus qu'à la passer à notre fonction de connexion habituelle, pour qu'elle tente (sur un autre port bien entendu si vous passez de l'UDP au TCP) de joindre le serveur et de faire son travail.

Puisque notre client est prêt, voyons comment créer notre serveur.

IV - Code du serveur UDP

Sur le même principe, nous allons créer un socket serveur UDP qui attend un message lui étant destiné, l'analyse pour savoir quoi répondre, et renvoie à l'expéditeur une réponse si nécessaire.

En attente d'un évènement

```
public void attendreRequete(int port)
{
 socket = null;
 try
 {
  // On essaye de créer notre socket serveur UDP
  socket = new DatagramSocket(port);
  socket.setSoTimeout(TIMEOUT_RECEPTION_REPONSE);
 }
 catch (SocketException se)
 {
  se.printStackTrace();
 }

 // On initialise les trames qui vont servir à recevoir et envoyer les paquets
 byte[] receiveData = new byte[1024];
 byte[] sendData = new byte[1024];

 // Tant qu'on est connecté, on attend une requête et on y répond
 while (socket != null && !socket.isClosed())
 {
  try
  {
   DatagramPacket paquetRecu = new DatagramPacket(receiveData, receiveData.length);
```

En attente d'un évènement
```
    socket.receive(paquetRecu);
    String requete = new String(paquetRecu.getData());
    InetAddress IPAddress = paquetRecu.getAddress();
    int port = paquetRecu.getPort();
    // Si on reçoit un "ping", on répond "pong" à celui qui nous l'a envoyé
    if (requete.contains("Ping"))
    {
     sendData = "Pong".getBytes();
     DatagramPacket paquetRetour = new DatagramPacket(sendData, sendData.length, IPAddress, port);
     socket.send(paquetRetour);
     socket.close();
    }
   }
   catch (Exception e)
   {
    e.printStackTrace();
   }
  }
 }
```

Rien de bien compliqué ici (du moins je l'espère). On crée notre socket serveur, et lorsque l'on reçoit un message, si celui-ci contient le texte 'Ping', on répond à son expéditeur 'Pong' et on ferme le socket.

Si vous voulez utiliser le même protocole Ping-Pong que moi, votre requête passée à la fonction envoyerTrameUDP() doit être 'Ping'.

Encore une fois, pensez à bien passer le même port en paramètre aux méthodes attendreRequete() et envoyerTrameUDP() !

V - Classe complète

Voici donc le code complet de la classe (je l'utilise à peu près tel quel dans deux de mes applications). Veuillez noter que quelques noms de méthodes ou de variables peuvent avoir changé entre les extraits de code précédents et celui-ci.

En attente d'un évènement
```
import java.io.InterruptedIOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;

import android.content.Context;
import android.net.DhcpInfo;
import android.net.wifi.WifiManager;
import android.util.Log;

public class Discovery
{
 public static final int RECEIVING_TIMEOUT = 10000;
 public static final int RECEIVING_TIMEOUT_SERVER = 30000;
 private DatagramSocket socket;
 private Context mContext;
 private int mPort;

 public Discovery(Context c, int p)
 {
  mContext = c;
  mPort = p;
 }

 public void stop()
 {
  udpNeeded = false;
```

En attente d'un évènement

```
   try
   {
    socket.close();
   }
   catch (Exception e)
   {
    e.printStackTrace();
   }
  }

  public String getServerIp()
  {
   try
   {
    DatagramPacket packet = sendBroadcast("Ping");
    return packet.getAddress().getHostAddress();
   }
   catch (InterruptedIOException ie)
   {
    Log.d("ERROR", "No server found");
    try
    {
     socket.close();
    }
    catch (Exception e2) {}
    ie.printStackTrace();
   }
   catch (Exception e)
   {
    Log.d("ERROR", "Verify your Wifi connection");
    try
    {
     socket.close();
    }
    catch (Exception e2) {}
    e.printStackTrace();
   }

   stop();
   return null;
  }

  public void waitingForDiscover()
  {
   socket = null;
   try
   {
    socket = new DatagramSocket(mPort);
    socket.setSoTimeout(RECEIVING_TIMEOUT_SERVER);
   }
   catch (SocketException se)
   {
    Log.d("ERROR", "Verify your Wifi connection");
    se.printStackTrace();
   }

   byte[] receiveData = new byte[1024];
   byte[] sendData = new byte[1024];

   int i = 0;
   while (socket != null && !socket.isClosed())
   {
    try
    {
     DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
     socket.receive(receivePacket);
     String sentence = new String(receivePacket.getData());
     InetAddress IPAddress = receivePacket.getAddress();
     int port = receivePacket.getPort();
     if (sentence.contains("Ping"))
```

En attente d'un évènement

```
     {
      sendData = "Pong".getBytes();
      DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, port);
      socket.send(sendPacket);
     }
    }
    catch (Exception e)
    {
     e.printStackTrace();
    }
    i++;
   }
   stop();
  }

  private InetAddress getBroadcastAddress() throws Exception
  {
      WifiManager wifi = (WifiManager) mContext.getSystemService(Context.WIFI_SERVICE);
      DhcpInfo dhcp = wifi.getDhcpInfo();

      int broadcast = (dhcp.ipAddress & dhcp.netmask) | ~dhcp.netmask;
      byte[] quads = new byte[4];
      for (int k = 0; k < 4; k++)
        quads[k] = (byte) ((broadcast >> k * 8) & 0xFF);
      return InetAddress.getByAddress(quads);
  }

  private DatagramPacket sendBroadcast(String data) throws Exception
  {
   socket = new DatagramSocket(mPort);
   socket.setBroadcast(true);
   InetAddress broadcastAdress = getBroadcastAddress();
   DatagramPacket packet = new DatagramPacket(data.getBytes(), data.length(), broadcastAdress, mPort);
   socket.send(packet);

   byte[] buf = new byte[1024];
   packet = new DatagramPacket(buf, buf.length);
   socket.setSoTimeout(RECEIVING_TIMEOUT);

   String myAdress = Engine.getIpAddress();

   socket.receive(packet);
   while (packet.getAddress().getHostAddress().contains(myAdress))
    socket.receive(packet);

   stop();
   return packet;
  }
 }
```

VI - Conclusion et remerciements

Et voilà. Rien de bien sorcier, mais il faut avouer que cela peut être pratique. J'espère que ce tutoriel vous aura plu, et s'il vous sert un jour ou l'autre j'en serai très content. Je reste à votre disposition sur le forum pour répondre à vos questions. J'en profite pour remercier **_Max_** pour sa relecture attentive.