

P2P sous Dot Net avec les sockets

Il n'existe pas, pour l'instant, d'API stable pour Dot Net spécialement dédiée au P2P, mais cela n'empêchera pas un développeur de programmer son application peer-to-peer en utilisant les sockets

Par Xavier Leclercq
Xavier.Leclercq@programmez.com

[1] Comment programmer les sockets ?

Un socket est une interface logicielle à partir de laquelle un développeur élaborera un service sur base d'un protocole réseau. Cette programmation par socket est universelle et standardisée (on le doit à l'université de Berkeley qui a inventé ce système il y a plus de 20 ans).

==> INSEREZ ICI SVP Communication_Socket.JPG

Légende : Les sockets ont été mis au point par l'université de Berkeley il y a déjà quelques décennies. Les fonctions C écrites à l'époque sont devenues une norme pour la programmation réseau de bas niveau.

Les espaces de noms du framework dot net impliqués dans cette programmation des sockets sont :

```
using System.Net;  
using System.Net.Sockets;
```

Le peer qui est à la fois client et serveur créera un objet socket de la classe du même nom en spécifiant le type d'adresse (Appletalk, Ipx, DecNet, NetBios, Sna, Unix, ou InterNetwork (adresses IP codées sur 4 octets)), le type de socket (Dgram, Raw, Stream) et le protocole utilisé (Icmp, IP, UDP, TCP).

```
Socket sock;  
sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp)
```

Le programme serveur doit ensuite exécuter Bind en spécifiant l'adresse ip de sa machine ainsi que le numéro de port :

```
n = sock.Bind(new IPEndPoint(IPAddress.Parse('192.168.0.24'),6000));
```

L'objet IPEndPoint regroupe une adresse IP et un numéro de port et forme un objet dit «de connexion».

Ensuite le serveur doit se mettre à l'écoute :

```
sock.Listen(1);  
Et se place en attente d'une demande de connexion :  
Socket sockClient = sock.Accept();
```

Le mécanisme est alors le suivant : le serveur se bloque, et ne réagit plus tant qu'il ne reçoit pas une demande de connexion d'un client. La demande reçue, le socket sockClient sera utilisé pour communiquer avec le client.

Du côté du client, cette demande a été initiée par une ligne :
`sock.Connect(new IPEndPoint(IPAddress.Parse('192.168.0.24'), 6000));`

Que peut faire le serveur ? Par exemple envoyer des données au client :
`string sdonnees = 'mes donnees';
byte[] tb = ASCIIEncoding.Default.GetBytes(sdonnees);
n = sockClient.Send(tb);`

Attention, pour que cette chaîne de données soit bien réceptionnée (par le client) il faut qu'il se mette à lire celles-ci :

```
byte[] donnees = new byte[100];  
int n = sock.Receive(donnees);  
string chainerecue = ASCIIEncoding.Default.GetString(donnees);
```

==> INSEREZ ICI SVP Communication_Socket.JPG

Légende : Résumé du mécanisme de communication par socket.

[2] Comment expédier un fichier de sauvegarde vers un peer

Mais la programmation d'un logiciel P2P ne s'arrête pas à la programmation des sockets. D'une part nous vous conseillons de travailler avec des règles car il ne faut pas oublier que vous devez impérativement prévoir le blocage de paquets anormaux. En effet au départ de la communication le client demandera au serveur d'établir une connexion (avec un paquet TCP sans données et avec le flag SYN activé). Durant une connexion c'est le seul paquet TCP sans flag ACK qui sera impliqué. Ce type de paquet est toujours envoyé d'un client vers un serveur et jamais l'inverse. Si c'est le cas alors ce paquet est "anormal" et devra être bloqué. Si les pairs de votre logiciel P2P communiquent sur Internet (ne se limitant pas au seul intranet) c'est important d'y penser. Ou du moins de penser à mettre au point un mécanisme qui sécuriserait les échanges.

D'autre part, il est bon de prévoir un serveur central. Celui-ci maintiendra à jour les informations à propos des clients qui sont connectés au réseau. C'est à dire qu'il pourra vous fournir leurs adresses IP, et la liste des fichiers qu'ils possèdent (par exemple dans le cadre d'une application de sauvegarde P2P). Ce serveur central pourra aussi permettre les recherches dans le cas d'une demande de restauration d'un fichier.

Ceci dit voici un extrait de code qui résume un peu ce que nous avons expliqué sur les sockets et qui permet d'envoyer un fichier à un peer. L'adresse IP est ici résolue par DNS car c'est toujours plus simple de travailler avec un nom plutôt qu'avec une adresse IP :

```
public void Envoyer_Fichier_Peer(String Nom_Du_Serveur, string Fichier_Distant, string  
Fichier_Local)  
{  
    try  
    {  
        TcpClient tcpc = new TcpClient();  
        Byte[] read = new Byte[1024];  
  
        int port = 6000;  
  
        IPEndPoint IPHost = Dns.Resolve(Nom_Du_Serveur);  
        string []aliases = IPHost.Aliases;
```

```

IPAddress[] addr = IPHost.AddressList;

IPEndPoint ep = new IPEndPoint(addr[0], port);
tcpc.Connect(ep);

Stream s = tcpc.GetStream();
Byte[] b = Encoding.ASCII.GetBytes(Fichier_Distant.ToCharArray());
s.Write( b, 0, b.Length );
int bytes;
FileStream fs = new FileStream(Fichier_Local, FileMode.OpenOrCreate);
BinaryWriter w = new BinaryWriter(fs);

while( (bytes = s.Read(read, 0, read.Length)) != 0)
{
    w.Write(read, 0, bytes);
    read = new Byte[1024];
}

tcpc.Close();
w.Close();
fs.Close();
}
catch(Exception ex)
{
    throw new Exception(ex.ToString());
}
}

```