Algorithmique parallèle et distribuée :
Les protocoles UDP et Multicast IP en Java

Julien Rossit

julien.rossit@parisdescartes.fr

IUT Paris Descartes

UNIVERSITÉ PARIS DESCARTES
INSTITUT UNIVERSITAIRE DE TECHNOLOGIE

Motivations

Ce qu'on a déja vu :

- s'adresser à une machine distante ;
- échanger des données avec cette machine en mode *connecté* (TCP).

Mais, pour aller plus loin :

- Comment envoyer des données *à la volée* sans se connecter ?
- Comment améliorer la *vitesse* des échanges ?
- Comment *diffuser* un message à *plusieurs* machine ?
- etc... (une autre fois ?)

Le package java.net, la suite

le protocole UDP

En mode *datagramme* :

- pas de *connexion* entre les machines ;
- pas de controle de la communication ;
- envoie de paquets de données (*datagrammes*) à construire ;
- envoie uniquement des tableaux de *byte* de taille limitée.

Avantages sur le protocole TCP :

- très simple à programmer ;
- communication plus rapide, plus performante ;

La fiabilité peut se progammer au niveau application !

Principe de la connection

Pour échanger un message en mode datagramme :

1. une *machine A* et une *machine B* créent chacune une socket UDP ;
2. la *machine A* attend la réception d'un message ;
3. une *machine B* envoie un message en précisant l'adresse de la *machine A*.

Pas besoin d'une d'architecture "client/serveur" !

La classe *DatagramPacket*

Constructeurs :

- *DatagramPacket(byte[] buf, int length)* ;
- *DatagramPacket(byte[] buf, int length, InetAddress, port)* ;
- etc.

Quelques méthodes :

- *setAddress(InetAddress iaddr)* : modifie l'adresse de destination ;
- *setPort(int iport)* : modifie le port de destination ;
- *setData(byte ibuf[])* : modifie la référence des données ;
- *setLength(int ilength)* : modifie la taille de la zone contenant les données ;
- *getPort()* : retourne le port de l'expéditeur ou de destination ;
- *getAddress()* : retourne l'adresse de l'expéditeur de destination ;
- *getData()* : retourne les données reçues ou à envoyer ;
- *getLength()* : retourne la taille des données reçues ou à envoyer ;
- etc.

La classe *DatagramSocket*

Constructeurs :

- *DatagramSocket(port)* : créé une socket UDP sur un *port* ;
- *DatagramSocket()* : créé une socket UDP sur un port quelconque ;
- etc.

Quelques méthodes :

- *send(DatagramPacket)* : envoi d'un paquet ;
- *receive(DatagramPacket)* : réception d'un paquet.
- etc.

Attention en modifiant le packet !

Ce qu'il faut savoir :

- *send* ne copie pas le paquet ;
- *send* est une primitive asynchrone !

La classe *DatagramSocket*

Constructeurs :

- *DatagramSocket(port)* : créé une socket UDP sur un *port* ;
- *DatagramSocket()* : créé une socket UDP sur un port quelconque ;
- etc.

Quelques méthodes :

- *send(DatagramPacket)* : envoi d'un paquet ;
- *receive(DatagramPacket)* : réception d'un paquet.
- etc.

Attention en modifiant le packet !

Ce qu'il faut savoir :

- *send* ne copie pas le paquet ;
- *send* est une primitive asynchrone !

Un exemple de code pour la machine A

```
import java.io.*;
import java.net.*;

class MachineA
{
  final static int port = 8532;
  final static int taille = 1024;
  final static byte buffer[] = new byte[taille];

  public static void main(String argv[]) throws Exception
  {
    DatagramSocket socket = new DatagramSocket(port);
    while(true)
    {
      DatagramPacket data = new DatagramPacket(buffer, buffer.length);
      socket.receive(data);
      System.out.println(data.getAddress());
      socket.send(data);
    }
  }
}
```

Un exemple de code pour la machine B

```
import java.io.*;
import java.net.*;

public class MachineB
{
  final static int port = 8532;
  final static int taille = 1024;
  final static byte buffer[] = new byte[taille];

  public static void main(String argv[]) throws Exception
  {
    InetAddress serveur = InetAddress.getByName(argv[0]);
    int length = argv[1].length();
    byte buffer[] = argv[1].getBytes();

    DatagramPacket dataSent = new DatagramPacket(buffer, length, serveur, port);
    DatagramSocket socket = new DatagramSocket();

    socket.send(dataSent);

    DatagramPacket dataRecieved = new DatagramPacket(new byte[length], length);
    socket.receive(dataRecieved);

    System.out.println("Data received : " + new String(dataReceived.getData()));
    System.out.println("From : " + dataReceived.getAddress() + ":" + dataReceived.getPort());
  }
}
```

Le multicast

Principe de fonctionnement

Le protocole *multicast* permet la diffusion de messages vers un *groupe* de destinataires :

- les messages sont émis vers une adresse unique ;
- les messages sont reçus par tous les récepteurs *écoutant* sur cette adresse ;
- plusieurs émetteurs sont possibles vers la même adresse ;
- les récepteurs peuvent rejoindre ou quitter le groupe à tout instant.

L'adresse :

- est une IP de classe D,
  (de 224.0.0.1 à 239.255.255.255, les 28 bits les moins significatifs constituent l'adresse du groupe) ;
- est indépendante de la localisation physique des émetteurs/récepteurs.

Principe (suite)

Utilisation : vidéo-conférences, jeux en réseaux, etc.

Caractéristiques intéressantes :

- indépendance entre service et localisation physique (choix adresse IP classe D) ;
- possibilité de doubler/multiplier les instances de service (tolérance aux pannes).

Certaines adresses de classe D sont assignées !
(voir http ://www.iana.org/assignments/multicast-addresses)

Concernant l'émission des données

Pour la multi-diffusions :

- pas d'envoi multiple d'un même paquet ;
- le réseau se charge d'acheminer le paquet aux différents destinataire enregistrés.

Cependant :

- pas de contrôle sur les inscriptions à l'adresse choisie ;
- la sécurité doit s'implémenter au niveau logiciel.

Principe de fonctionnement

Pour échanger des données avec le protocole multicast :

1. chaque émetteur créé une socket Multicast-IP ;
2. les récepteurs rejoignent le(s) groupe(s) de diffusion ;
3. chaque émetteur commence à émettre des messages ;
4. les récepteurs reçoivent les messages émis sur les groupes auxquels ils sont inscrits.

La classe *MulticastSocket*

Constructeurs

- *MulticastSocket(port)* : création sur port ;
- *MulticastSocket()* ; création sur port quelconque
- etc.

Quelques méthodes :

- *send(DatagramPacket)* : envoi ;
- *receive(DatagramPacket)* : réception ;
- *joinGroup(InetAddress)* : se lier à un groupe ;
- *leaveGroup(InetAddress)* : quitter un groupe.

Des questions ?