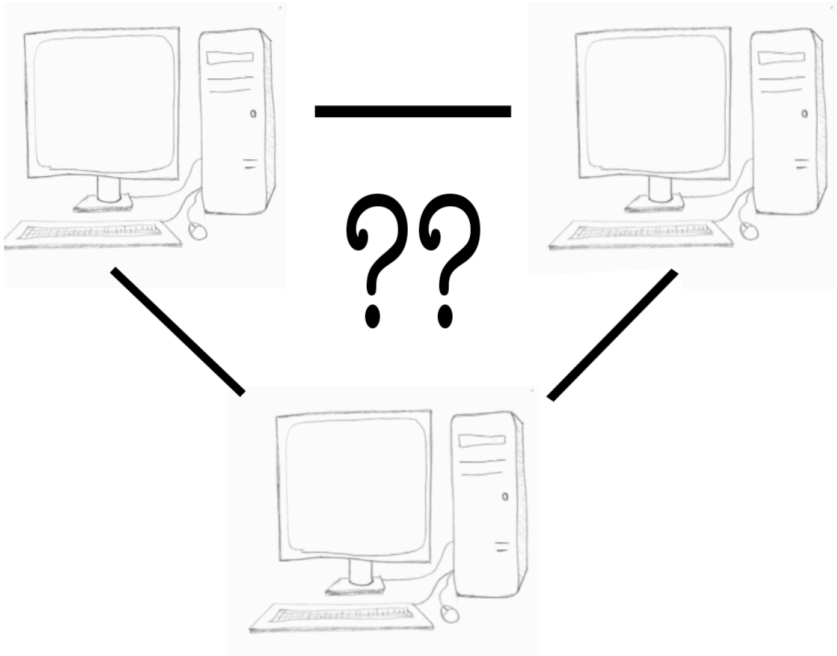
PROGRAMMATION RÉSEAU

Arnaud Sangnier
sangnier@irif.fr

UDP - Mode par paquet



Différence flux et paquet

- Dans la communication **par flux** (comme **TCP**)
  - Les informations sont reçues dans l'ordre de leur émission
  - Il n'y a pas de perte
  - Inconvénient :
    - Établissement d'une connexion
    - Nécessité de ressources supplémentaire pour la gestion
- Dans la communication **par paquet** (comme **UDP**)
  - Pas d'ordre dans la délivrance des paquets
    - Un paquet posté en premier peut arrivé en dernier
  - Pas de fiabilité
    - Un paquet envoyé peut être perdu

Communication par paquet

La communication UDP en java

- Deux classes vont être importantes :
  - La classe **java.net.DatagramSocket**
    - Sert pour créer des socket UDP
    - Fournit les méthodes pour communiquer
  - La classe **java.net.DatagramPacket**
    - Sert à encapsuler les données que l'on souhaite envoyer
    - En pratique, on va envoyer et recevoir des objets de type **DatagramPacket**

La classe DatagramSocket

- Elle représente :
  - Un point de réception de paquets, ou,
  - Un point d'envoi de paquets
- **Pour la réception de paquets** :
  - On doit nécessairement attaché la Socket à un port et à une adresse (ou à toutes les adresses d'une machine donnée)
  - C'est le point de communication où les autres entités du réseau enverront leurs données
  - Comme pour les serveurs en TCP, on précisera souvent le port
- **Pour l'envoi de paquets :**
  - Il n'est pas nécessaire d'attacher la Socket à un port

La classe DatagramSocket (2)

- **public DatagramSocket()**
  - Crée une socket sans l'attacher à un port
  - Utilisée si on ne souhaite qu'envoyer un message
- **public DatagramSocket(int port)**
  - Crée une socket et l'attache à port de la machine où elle est appelée
- **public DatagramSocket(SocketAddress sa)**
  - Dans la classe SocketAddress on peut donner un port et une adresse
- **On servira surtout des deux premiers constructeurs !**

Des nouvelles classes

- La classe **InetAddress** représente une adresse
- La classe **SocketAddress** caractérise un point de communication
  - C'est une classe abstraite, en pratique on utilisera sa sous-classe **InetSocketAddress**

La classe InetAddress

- **Pas de constructeur**
- En pratique on la récupère grâce à des méthodes statiques
  - **static InetAddress getByAddress(byte[] addr)**
    - ici addr est un tableau d'octets contenant une adresse IP
    - Tableau de taille 4 pour IPv4 et de taille 16 pour IPv6
  - **static InetAddress getLocalHost()**
    - Retourne l'InetAddress correspondant à la machine locale
  - **static InetAddress getByName(String host)**
    - Ici host peut-être soit le nom de la machine, soit l'adresse IP en chaîne de caractères
- On peut ensuite récupérer ces infos, avec les méthodes :
  - **String getHostName()**, **String toString()**, **byte[] getAddress()**

Exemple

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

public class InetTest {
    public static void main(String[] args){
        try{
            InetAddress ia=InetAddress.getByName("lampe");
            System.out.println(ia.toString());

        } catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

La classe InetSocketAddress

- Cette classe représente les caractéristiques d'une socket
  - (la machine + le port)
- Constructeurs :
  - **InetSocketAddress(InetAddress addr, int port)**
  - **InetSocketAddress(int port)**
    - L'adresse ici est quelconque
  - **InetSocketAddress(String hostname, int port)**
- Méthodes utiles :
  - **String getHostName()**
  - **int getPort()**
  - **InetAddress getAddress()**
- On verra que nos paquets contiennent une **InetSocketAddress**
  - On pourra donc exploiter ces informations

Recevoir des données en UDP

- Créer une socket UDP (avec la classe **DatagramSocket**)
- Attacher la socket à un port
  - Soit on fournit le port au moment de la construction
    - **public DatagramSocket(int port)**
  - Soit on la lie après
    - méthode **void bind(SocketAddress addr) de DatagramSocket**
- Créer un paquet vide où stocker le paquet reçu (classe **DatagramPacket**)
  - Il faut donc connaître la taille du paquet
- Recevoir les paquets avec la méthode
  - **void receive(DatagramPacket p)**

La classe DatagramPacket

- Cette classe représente les paquets qui seront envoyés/reçus
- Constructeurs principaux
  - **DatagramPacket(byte[] buf, int length)**
    - Ici on mettra la longueur du paquet que l'on s'attend à recevoir
    - buf devra être assez grand pour recevoir ces paquet
    - Ce constructeur est pour recevoir
  - **DatagramPacket(byte[] buf, int length, InetAddress address, int port)**
    - On précise en plus l'adresse de la machine et le port où le paquet doit être envoyé
    - Ce constructeur est donc pour envoyer
  - **DatagramPacket(byte[] buf, int length, SocketAddress address)**
    - Comme au-dessus mais avec SocketAddress

Recevoir un paquet

- On commence donc par déclarer un tableau d'octets d'une certaine taille
- On associe ensuite ce tableau d'octets à un paquet
- Au moment de la réception, le **receive** remplit le tableau d'octets

```
DatagramSocket dso=new DatagramSocket(5555);
byte[]data=new byte[100];
DatagramPacket paquet=new DatagramPacket(data,data.length);
dso.receive(paquet);
```

- On peut ensuite récupérer le contenu et la longueur du paquet avec les deux méthodes suivante de **DatagramPacket** :
  - **byte[] getData()**
  - **int getLength()**
- Pour passer des chaînes de caractères aux tableaux d'octets on peut utiliser :
  - **String(byte[] bytes, int offset, int length)** -> constructeur
  - **byte[] getBytes()** -> méthode de la classe String

Un receveur UDP

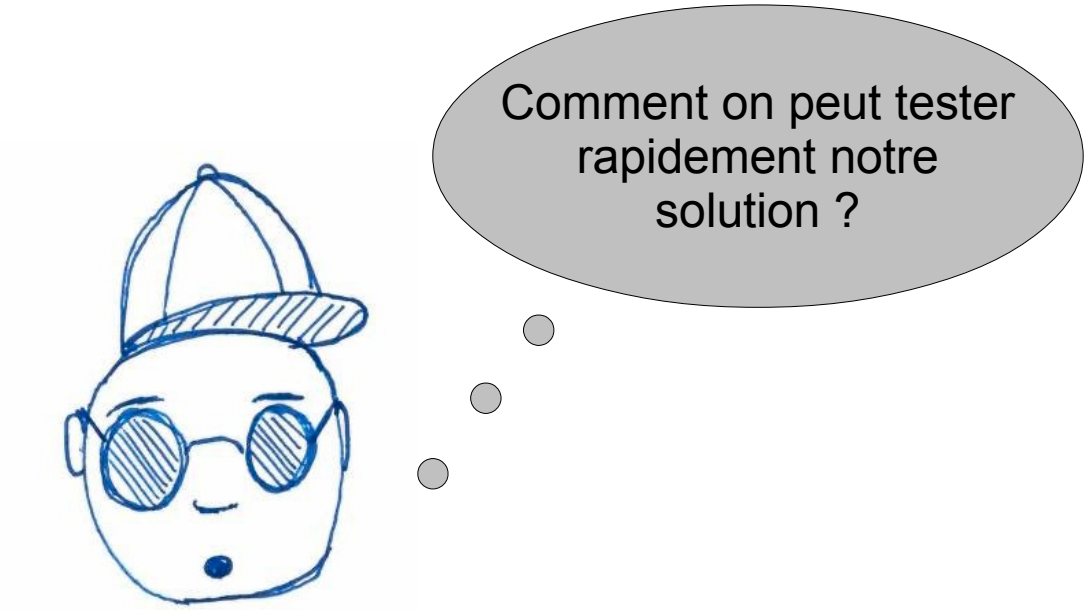
- On va faire un programme qui :
  - Écoute sur le port 5555
  - Attend un paquet (de moins de 100 caractères) correspondant à une chaîne de caractères
  - Affiche le contenu de ce paquet

Solution

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

public class ReceiveUDP {
    public  static void main(String[] args){
        try{
            DatagramSocket dso=new DatagramSocket(5555);
            byte[]data=new byte[100];
            DatagramPacket paquet=new DatagramPacket(data,data.length);
            while(true){
                dso.receive(paquet);
                String st=new
                   String(paquet.getData(),0,paquet.getLength());
                System.out.println("J'ai reçu :"+st);
            }
        } catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

Tester notre solution



- En TCP, on utilisait **telnet**
- Mais **telnet** ne permet pas de faire UDP
- On va utiliser **netcat**

Netcat (nc)

- **netcat** permet
  - de se connecter en mode TCP
  - d'envoyer des messages en UDP
  - d'écouter sur un port en TCP ou UDP
  - Comme pour telnet
    - Les messages tapés au clavier sont envoyés
    - Les messages reçus sont affichés sur le terminal
- Pour UDP, on utiliser l'option **-u**
- Pour écouter on utilise l'option **-l**
- Pour notre exemple précédent :
  - Dans un terminal on lance ReceiveUDP
  - Dans un autre terminal **nc -u localhost 5555**

Méthodes utiles de DatagramPacket

- **InetAddress getAddress()**
  - Renvoie l'adresse de la machine vers laquelle le paquet est envoyé ou de la machine qui a envoyé le paquet
- **int getPort()**
  - Renvoie le port vers lequel le paquet est envoyé ou depuis lequel la paquet a été envoyé
- **SocketAddress getSocketAddress()**
  - Comme les deux méthodes ci-dessus avec SocketAddress

Exemple 1

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

public class ReceiveUDPPlus {
    public  static void main(String[] args){
        try{
            DatagramSocket dso=new DatagramSocket(5555);
            byte[]data=new byte[100];
            DatagramPacket paquet=new DatagramPacket(data,data.length);
            while(true){
                dso.receive(paquet);
                String st=new
                    String(paquet.getData(),0,paquet.getLength());
                System.out.println("J'ai reçu :"+st);
                System.out.println(
                     "De la machine  "+paquet.getAddress().toString());
                System.out.println("Depuis le port "+paquet.getPort());
            }
        } catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

Exemple 2

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

public class ReceiveUDPPlus2 {
    public  static void main(String[] args){
        try{
            DatagramSocket dso=new DatagramSocket(5555);
            byte[]data=new byte[100];
            DatagramPacket paquet=new DatagramPacket(data,data.length);
            while(true){
                dso.receive(paquet);
                String st=new
                   String(paquet.getData(),0,paquet.getLength());
                System.out.println("J'ai reçu :"+st);
                System.out.println
                   ("De la machine "+paquet.getAddress().getHostName());
                System.out.println("Depuis le port "+paquet.getPort());
            }
        } catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

Exemple 3

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

public class ReceiveUDPPlus3 {
    public  static void main(String[] args){
        try{
            DatagramSocket dso=new DatagramSocket(5555);
            byte[]data=new byte[100];
            DatagramPacket paquet=new DatagramPacket(data,data.length);
            while(true){
                dso.receive(paquet);
                String st=new
                    String(paquet.getData(),0,paquet.getLength());
                System.out.println("J'ai reçu :"+st);
                InetSocketAddress ia=(InetSocketAddress)
                                            paquet.getSocketAddress();
                System.out.println("De la machine "+ia.getHostName());
                System.out.println("Depuis le port "+ia.getPort());
            }
        } catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

Envoi de paquets

- Pour envoyer des paquets, on n'a pas besoin d'attacher la socket à un port
- On met l'adresse et le port du destinataire dans le paquet

```
String s="MESSAGE "+i+" \n";
byte[]data = s.getBytes();
InetSocketAddress ia=new InetSocketAddress("localhost",5555);
DatagramPacket paquet=new DatagramPacket(data,data.length,ia);
```

- Ou encore :

```
String s="MESSAGE "+i+" \n";
byte[]data = s.getBytes();
DatagramPacket paquet=new DatagramPacket(data,data.length,
                              InetAddress.getByName("localhost"),5555);
```

- **Pensez à un colis où on écrit l'adresse sur le paquet !**
- Ensuite on fait **send** sur une **DatagramSocket**
- **ATTENTION : EN UDP, c'est pas parce que l'on envoie un paquet qu'il est reçu !!!**

Exemple d'envoi

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

public class EnvoiUDP {
    public  static void main(String[] args){
        try{
            DatagramSocket dso=new DatagramSocket();
            byte[]data;
            for(int i=0;i <= 60; i++){

                String s="MESSAGE "+i+" \n";
                data=s.getBytes();
                DatagramPacket paquet=new
                    DatagramPacket(data,data.length,
                       InetAddress.getByName("localhost"),5555);
                dso.send(paquet);
            }
        } catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

Exemple d'envoi 2

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

public class EnvoiUDP2 {
    public  static void main(String[] args){
        try{
            DatagramSocket dso=new DatagramSocket();
            byte[]data;
            for(int i=0;i <= 10; i++){
                //Thread.sleep(1000);
                String s="MESSAGE "+i+" \n";
                data=s.getBytes();
                InetSocketAddress ia=new
                                InetSocketAddress("localhost",5555);
                DatagramPacket paquet=
                            new DatagramPacket(data,data.length,ia);
                dso.send(paquet);
            }
        } catch(Exception e){
            e.printStackTrace();
        }
    }
}
```