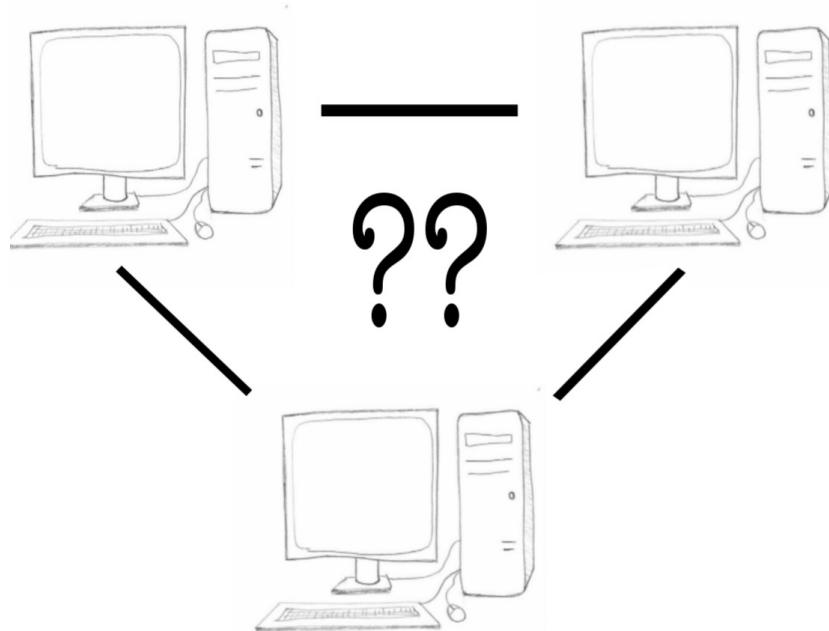


PROGRAMMATION RÉSEAU

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
 - Cette socket sera associée à toutes les adresses de la machine
- **public DatagramSocket(int port, InetAddress ia)**
 - Comme au-dessus, mais on peut préciser l'adresse
- **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



À quoi correspondent
les classes `InetAddress`
et `SocketAddress`

- 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("monjetas");
            System.out.println(ia.toString());

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



The image shows a terminal window titled "Terminal - ssh - 80x24". The prompt is "sangnier@lucien:~/ProgReseaux/Cours7\$". The user has entered "java InetTest". The output is "monjetas/194.254.199.95". The prompt is now "sangnier@lucien:~/ProgReseaux/Cours7\$".

```
Terminal - ssh - 80x24
sangnier@lucien:~/ProgReseaux/Cours7$ java InetTest
monjetas/194.254.199.95
sangnier@lucien:~/ProgReseaux/Cours7$
```

La classe InetAddress

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



Comment on peut tester rapidement 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 utilise l'option **-u**
- Pour écouter on utilise l'option **-l**

Un client-serveur TCP avec nc

```
Terminal — nc — 80x24
bash-3.2$ nc localhost 4444
HELLO
HALLO
█
```

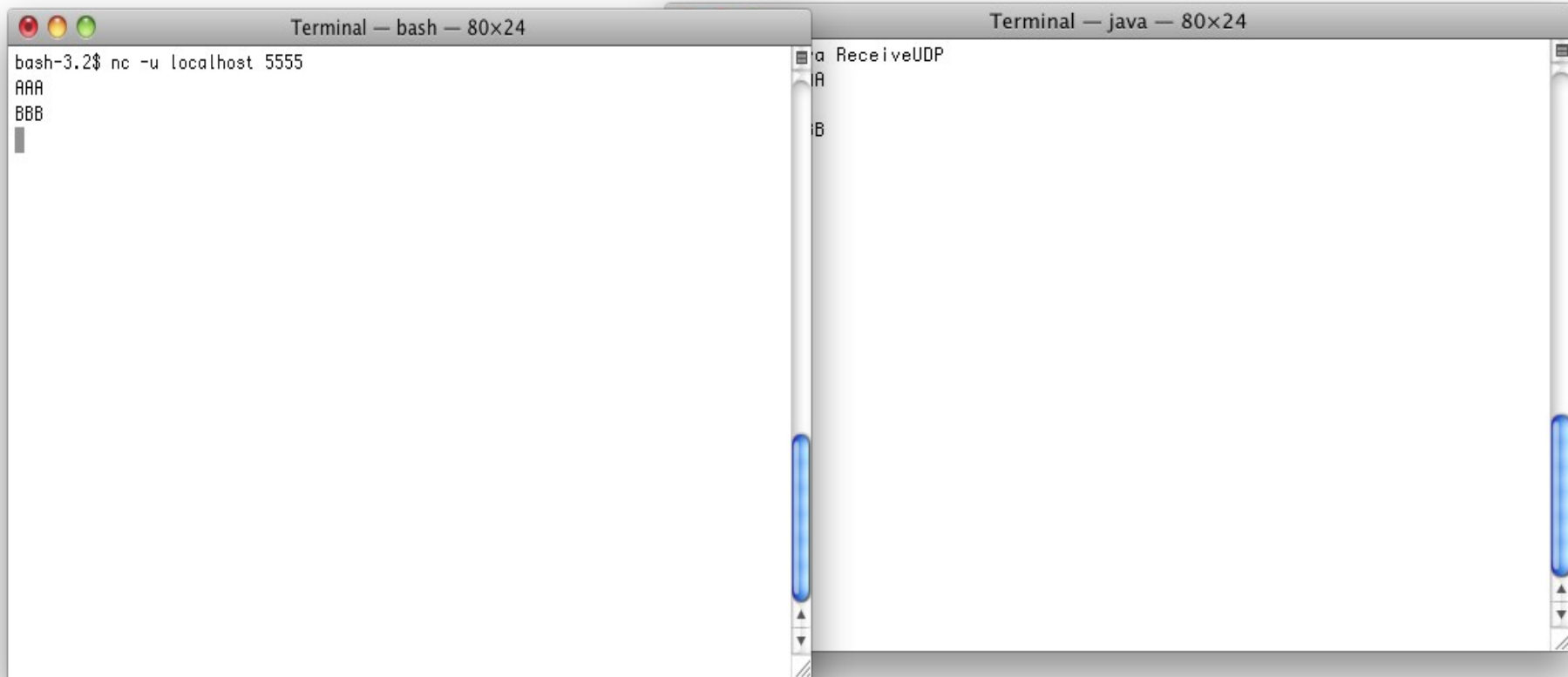
CLIENT

```
Terminal — bash — 80x24
bash-3.2$ nc -l 4444
HELLO
HALLO
█
```

*On précise pas
la machine*

SERVEUR

Pour tester notre application



The image shows two terminal windows side-by-side. The left window, titled "Terminal — bash — 80x24", contains the following text: `bash-3.2$ nc -u localhost 5555`, `AAA`, and `BBB`. The right window, titled "Terminal — java — 80x24", contains the text `ReceiveUDP`, `A`, and `B`. Both windows have a scroll bar on the right side.

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 le 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);
                InetAddress ia=(InetAddress)
                    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();
                InetAddress ia=new
                    InetAddress("localhost",5555);
                DatagramPacket paquet=
                    new DatagramPacket(data,data.length,ia);
                dso.send(paquet);
            }
        } catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

La communication UDP en C

- Pour le C, il faut changer le type des sockets quand on appelle socket
 - **SOCK_STREAM** : socket TCP
 - **SOCK_DGRAM** : socket UDP

```
int sock=socket(PF_INET,SOCK_DGRAM,0);
```

- On n'a plus besoin d'utiliser **connect** (vu qu'on n'est pas en mode connecté)
- Si on veut écouter sur une socket, il faudra bien faire le **bind**
- On n'aura plus de **listen** ou d'**accept**
- On utilisera **sendto**, **recv**, **recvfrom** pour l'envoi et la réception des paquets

Envoi en UDP

- Comme en Java, on va préciser le destinataire au moment de l'envoi du paquet
- On va utiliser la fonction suivante :
 - **ssize_t sendto(
int socket, // le numéro de socket
const void *buffer, // le message à envoyer
size_t length, // la taille de ce message
int flags, // pour les options
const struct sockaddr *dest_addr, //infos destinataire
socklen_t dest_len); //taille de la struct sockaddr**

Envoi en UDP (2)

- On va donc commencer par récupérer les infos du destinaire par exemple avec **getaddrinfo**
- **int getaddrinfo(const char *hostname, const char *servname ,const struct addrinfo *hints, struct addrinfo **res);**
- Rappel :

```
struct addrinfo hints;  
bzero(&hints, sizeof(struct addrinfo));  
hints.ai_family = AF_INET;  
hints.ai_socktype=SOCK_DGRAM;  
struct addrinfo *first_info;  
int r=getaddrinfo("localhost", "5555", &hints, &first_info);
```

- Ici, dans les hints, on précise SOCK_DGRAM car on veut des sockets UDP
- Ensuite le struct sockaddr correspondant est dans
 - first_info->ai_addr; (de type struct sockaddr*)

Exemple d'envoi

```
int main() {
    int sock=socket(PF_INET,SOCK_DGRAM,0);
    struct addrinfo *first_info;
    struct addrinfo hints;
    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET;
    hints.ai_socktype=SOCK_DGRAM;
    int r=getaddrinfo("localhost","5555",&hints,&first_info);
    if(r==0){
        if(first_info!=NULL){
            struct sockaddr *saddr=first_info->ai_addr;
            char tampon[100];
            int i=0;
            for(i=0;i<=10;i++){
                strcpy(tampon,"MESSAGE ");
                char entier[3];
                sprintf(entier,"%d",i);
                strcat(tampon,entier);
                sendto(sock,tampon,strlen(tampon),0,saddr
                    (socklen_t)sizeof(struct sockaddr_in));
            }
        }
    }
    return 0;
}
```

Pour la réception

- Là encore on doit préciser que l'on utilise des socket UDP
- On fait un bind pour écouter sur le bon port
- **int bind(int sockfd, struct sockaddr *my_addr, int addrlen);**
- Rappel :

```
sock=socket(PF_INET,SOCK_DGRAM,0);  
struct sockaddr_in address_sock;  
address_sock.sin_family=AF_INET;  
address_sock.sin_port=htons(5555);  
address_sock.sin_addr.s_addr=htonl(INADDR_ANY);  
int r=bind(sock,(struct sockaddr *)&address_sock,sizeof(struct  
sockaddr_in);
```

- On utilise la fonction suivante pour recevoir :
 - **ssize_t recv(int socket, void *buffer, size_t length, int flags);**

Exemple

```
int main() {
    int sock=socket(PF_INET,SOCK_DGRAM,0);
    sock=socket(PF_INET,SOCK_DGRAM,0);
    struct sockaddr_in address_sock;
    address_sock.sin_family=AF_INET;
    address_sock.sin_port=htons(5555);
    address_sock.sin_addr.s_addr=htonl(INADDR_ANY);
    int r=bind(sock,(struct sockaddr *)&address_sock,sizeof(struct
sockaddr_in));
    if(r==0){
        char tampon[100];
        while(1){
            int rec=recv(sock,tampon,100,0);
            tampon[rec]='\0';
            printf("Message recu : %s\n",tampon);
        }
    }
    return 0;
}
```


D'autres informations ?



Est-ce-qu'on peut savoir
qui nous envoie des
paquets ?

- OUI !!!!
- Avec la méthode :
 - **ssize_t recvfrom(int socket, void *restrict buffer, size_t length, int flags, struct sockaddr *restrict address, socklen_t *restrict address_len);**

Exemple

```
int main() {
    int sock=socket(PF_INET,SOCK_DGRAM,0);
    sock=socket(PF_INET,SOCK_DGRAM,0);
    struct sockaddr_in address_sock;
    address_sock.sin_family=AF_INET;
    address_sock.sin_port=htons(5555);
    address_sock.sin_addr.s_addr=htonl(INADDR_ANY);
    int r=bind(sock,(struct sockaddr *)&address_sock,sizeof(struct
sockaddr_in));
    struct sockaddr_in emet;
    socklen_t a=sizeof(emet);
    if(r==0){
        char tampon[100];
        while(1){
            int rec=recvfrom(sock,tampon,100,0,(struct sockaddr *)&emet,&a);
            tampon[rec]='\0';
            printf("Message recu : %s\n",tampon);
            printf("Port de l'emetteur: %d\n",ntohs(emet.sin_port));
            printf("Adresse de l'emetteur: %s\n",inet_ntoa(emet.sin_addr));
        }
    }
    return 0;
}
```