

CSC4509 — TCP-UDP

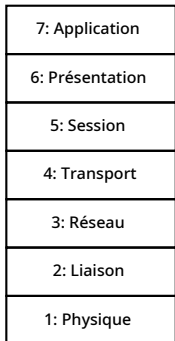
Éric Lallet

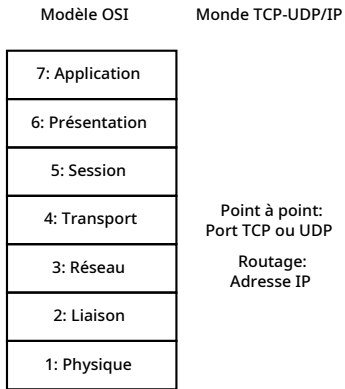
`Eric.Lallet@telecom-sudparis.eu`

Télécom SudParis

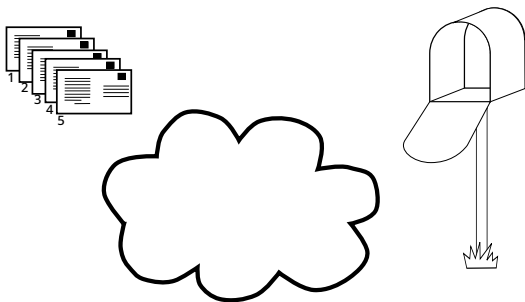
30 avril 2025

Modèle OSI



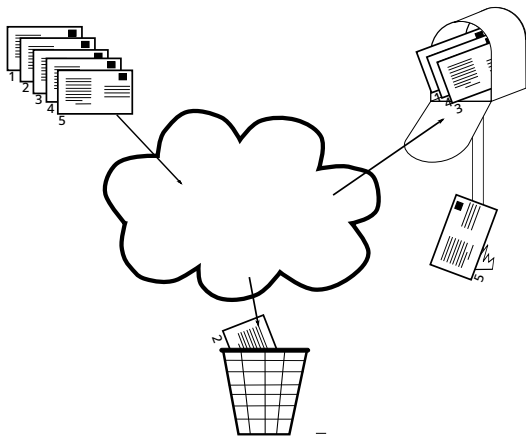


Le service Datagram (UDP)

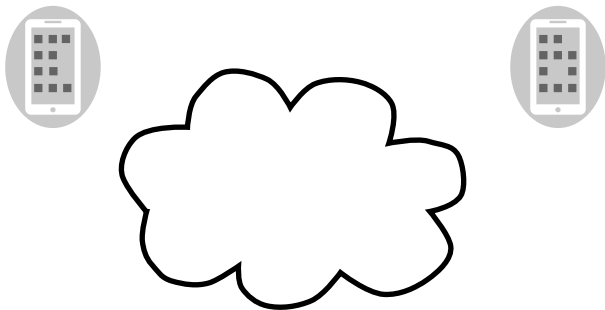


-

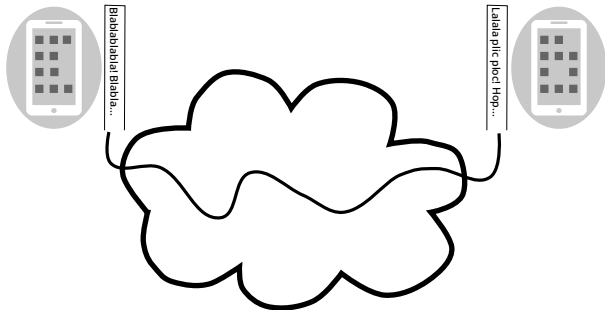
Le service Datagram (UDP)



Le service Stream (TCP)



Le service Stream (TCP)



La classe java.net.InetAddress

Les adresses IP sont gérées par la classe `java.net.InetAddress`

Quelques exemples d'usages :

```
import java.net.InetAddress;
(...)
InetAddress destAddr;

destAddr = InetAddress.getByName("localhost");
destAddr = InetAddress.getByName("157.159.100.105");
System.out.println("message pour : " + destAddr.getHostName());
destAddr = InetAddress.getByName("ssh.int-evry.fr");
System.out.println("message pour : " +
                   destAddr.getHostAddress());
```


La classe java.net.InetSocketAddress

Les adresses des sockets TCT-UDP/IP sont gérées par la classe

`java.net.InetSocketAddress`

Quelques exemples d'usages :

```
import java.net.InetAddress;
import java.net.InetSocketAddress;
(...)
InetAddress destAddr;
InetSocketAddress destSockAddr;
// ... on suppose que destAddr contient l'adresse IP

rcvAddress = new InetSocketAddress(destAddr,5000);

// ... on suppose que argv[1] contient le port
rcvAddress = new InetSocketAddress(destAddr,
                                   Integer.parseInt(argv[1]));
```

Pour envoyer des paquets UDP avec Java NIO il faut :

- Ouvrir un canal avec la méthode `DatagramChannel.open()`
- Remplir un *ByteBuffer* avec les données.
- Faire le `flip()` sur le buffer.
- Envoyer les données avec la méthode `send()` du canal.

UDP avec Java NIO : l'émission

```
import java.nio.ByteBuffer;
import java.nio.channels.DatagramChannel;
(... plus les autres imports)
InetAddress destAddr = InetAddress.getByName("machine");
InetSocketAddress destSockAddr =
    new InetSocketAddress(destAddr, 5000);
DatagramChannel sndChan = DatagramChannel.open();
ByteBuffer dataBuf;
(... on suppose dataBuf rempli ...)

dataBuf.flip();
int sent = sndChan.send(dataBuf, destSockAddr);

System.out.println("Un paquet de " + sent + " octets pour "
    + destAddr.getHostName());
```

Pour recevoir des paquets UDP avec Java NIO il faut :

- Ouvrir un canal avec la méthode `DatagramChannel.open()`
- Lier ce canal à l'adresse UDP/IP
- Avoir un *ByteBuffer* pour recevoir données.
- Faire le `clear()` sur le buffer.
- Recevoir les données avec la méthode `receive()` du canal.

UDP avec Java NIO : la réception

```
import java.nio.ByteBuffer;
import java.nio.channels.DatagramChannel;
(... plus les autres imports)
InetSocketAddress sndSockAddr;
InetSocketAddress rcvSockAddr =
    new InetSocketAddress(5000);
ByteBuffer dataBuf;
DatagramChannel rcvChan = DatagramChannel.open();
rcvChan.bind(rcvSockAddr);

dataBuf.clear();
sndSockAddr=(InetSocketAddress)rcvChan.receive(dataBuf);
System.out.println("Un paquet de " +
    sndSockAddr.getAddress().getHostName());
```

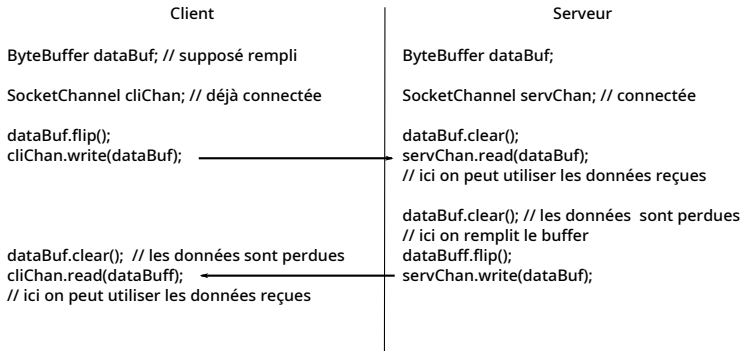
Pour faire des échanges sur une connexion TCP avec Java NIO, il y a trois étapes :

- 1 L'établissement de la connexion (dissymétrie client/serveur)
- 2 L'échange entre client et serveur (symétrie)
- 3 La fermeture de la connexion

TCP avec Java NIO : l'ouverture de connexion

Client	Serveur
<pre>String servHost = ... ; int servPort = ... ; InetAddress servAddr= InetAddress.getByName(servhost); InetSocketAddress servSockAddr = new InetSocketAddress(servAddr,servPort); SocketChannel cliChan;</pre>	<pre>int servPort = ... ; InetSocketAddress servAddr = new InetSocketAddress(servPort); SocketChannel servChan; ServerSocketChannel listenChan; listenChan = ServerSocketChannel.open(); listenChan.bind(servAddr); servChan = listenChan.accept();</pre>
<pre>cliChan = SocketChannel.open(servSockAddr);</pre>	

TCP avec Java NIO : l'échange de données



TCP avec Java NIO : la fermeture de connexion

Client

```
SocketChannel cliChan;  
// on suppose que la connexion et l'utilisation  
// de cliChan ont été réalisées  
  
// possibilité de fermeture partielle  
cliChan.shutdownOutput(); // envois terminés  
cliChan.shutdownInput(); //réceptions terminées  
  
// généralement on fait directement la fermeture  
cliChan.close();
```

Serveur

```
SocketChannel servChan;  
ServerSocketChannel listenChan;  
// on suppose que les canaux ont été  
// ouverts  
  
listenChan.close() // plus aucun nouveau  
// client ne pourra être accepté  
  
// possibilité de fermeture partielle  
servChan.shutdownOutput(); // envois terminés  
servChan.shutdownInput(); //réceptions terminées  
  
// généralement on fait directement la fermeture  
servChan.close();
```

Une option bien pratique, pour éviter d'attendre la fermeture du contexte TCP lors du relancement du serveur :

```
import java.net.StandardSocketOptions;  
(...)  
ServerSocketChannel listenChannel;  
  
listenChannel = ServerSocketChannel.open();  
listenChannel.setOption(StandardSocketOptions.SO_REUSEADDR, true);  
listenChannel.bind(new InetSocketAddress(5000));
```

Il existe deux moyens de détecter une déconnexion TCP :

- 1 la méthode `read()` retourne `-1`
- 2 la méthode `write()` lève une `IOException`

La méthode `isConnected()` de la classe `SocketChannel` retourne `true` à partir du moment où la connexion a été établie (par un `open()`, un `connect()` ou un `accept()`), et continue de retourner `true` même après la déconnexion TCP tant que le canal n'a pas été explicitement fermé (par un `close()`). Il ne faut donc pas l'utiliser pour détecter une déconnexion.

Quelques liens vers des tutoriels qui présentent les mêmes sujets :

- tutoriel JAVA NIO sur les DatagramChannel :
<http://tutorials.jenkov.com/java-nio/datagram-channel.html>
- tutoriel JAVA NIO sur les SocketChannel :
<http://tutorials.jenkov.com/java-nio/socketchannel.html>
- tutoriel JAVA NIO sur les ServerSocketChannel :
<http://tutorials.jenkov.com/java-nio/server-socket-channel.html>