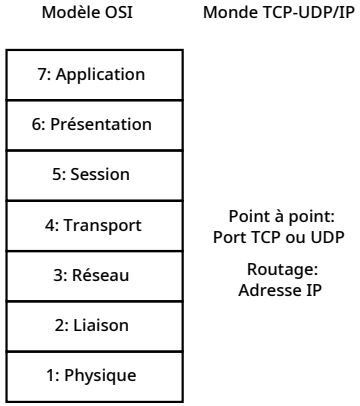
CSC4509 — TCP-UDP

Éric Lallet
Eric.Lallet@telecom-sudparis.eu

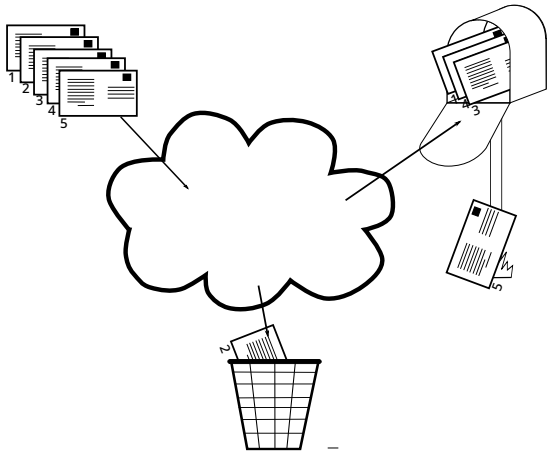
Télécom SudParis

30 avril 2025

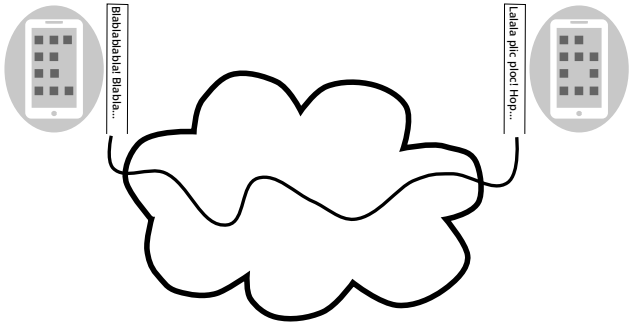
Les couches réseaux



Le service Datagram (UDP)



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]));
```

UDP avec Java NIO : l'émission

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 buffeur.
- Envoyer les données avec la méthode `send()` du canal.

Couches réseaux
Service Datagram
Service Stream
Adresses TCP–UDP/IP avec Java
Java NIO et UDP
Java NIO et TCP

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());
```

Éric Lallet Eric.Lallet@telecom-sudparis.eu CSC4509 — TCP-UDP

UDP avec Java NIO : la réception

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 buffeur.
- Recevoir les données avec la méthode `receive()` du canal.

Couches réseaux
Service Datagram
Service Stream
Adresses TCP–UDP/IP avec Java
Java NIO et UDP
Java NIO et TCP

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());
```

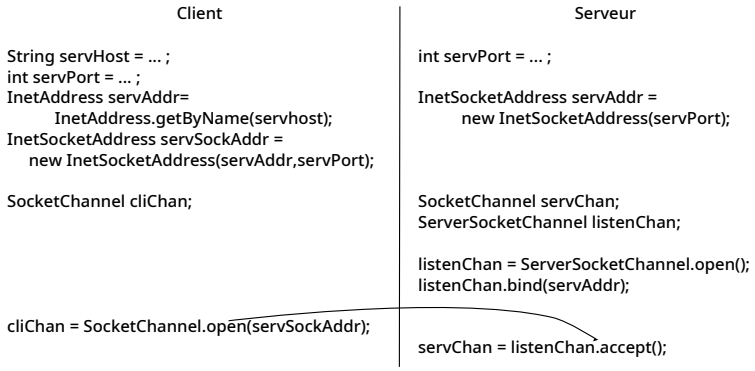
Éric LalletEric.Lallet@telecom-sudparis.eu CSC4509 — TCP-UDP

TCP avec Java NIO

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



Note : Il est possible de lier le canal dès le open() en passant l'adresse en paramètre : *listenChan = ServerSocketChannel.open(servAddr) ;*

Mais nous verrons plus loin que, très souvent, on souhaite modifier le comportement du bind() en modifiant les options par défaut, et il faudra donc placer un appel de méthode entre le open() et le bind(). Il sera donc nécessaire de bien séparer les deux étapes.

TCP avec Java NIO : l'échange de données

Client	Serveur
ByteBuffer dataBuf; // supposé rempli	ByteBuffer dataBuf;
SocketChannel cliChan; // déjà connectée	SocketChannel servChan; // connectée

dataBuf.flip();
cliChan.write(dataBuf); →
	dataBuf.clear();
	servChan.read(dataBuf);
	// ici on peut utiliser les données reçues

	dataBuf.clear(); // les données sont perdues
	// ici on remplit le buffer
	dataBuff.flip();

dataBuf.clear(); // les données sont perdues
cliChan.read(dataBuff); ←
// ici on peut utiliser les données reçues
	servChan.write(dataBuf);

TCP avec Java NIO : la fermeture de connexion

Client	Serveur

SocketChannel cliChan;
// on suppose que la connexion et l'utilisation
// de cliChan ont été réalisées
	SocketChannel servChan;
	ServerSocketChannel listenChan;
	// on suppose que les cannaux ont été
	// ouverts

// possibilité de fermeture partielle
cliChan.shutdownOutput(); // envois terminés
cliChan.shutdownIutput(); //réceptions terminées
	listenChan.close() // plus aucun nouveau
	// client ne pourra être accepté

// généralement on fait directement la fermeture
cliChan.close();
	// possibilité de fermeture partielle
	servChan.shutdownOutput(); // envois terminés
	servChan.shutdownIutput(); //réceptions terminées

	// généralement on fait directement la fermeture
	servChan.close();

TCP avec Java NIO : l'option SO_REUSEADDR

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)));
```


TCP avec Java NIO : détection de la déconnexion

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.

Pour aller plus loin

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