

# Mise au propre des TDs d'application réseaux

Yann Miguel

3 mars 2020

## Table des matières

1	TD1	3
1.1	Exercice 1 . . . . .	3
1.1.1	Question 1 . . . . .	3
1.1.2	Question 2 . . . . .	3
1.1.3	Question 3 . . . . .	3
1.1.4	Question 4 . . . . .	3
1.2	Exercice 2 . . . . .	3
1.2.1	Question 1 . . . . .	3
1.2.2	Question 2 . . . . .	3
1.3	Exercice 3 . . . . .	4
1.3.1	Question 1 . . . . .	4
1.3.2	Question 2 . . . . .	4
1.3.3	Question 3 . . . . .	4
1.3.4	Question 4 . . . . .	4
1.4	Exercice 4 . . . . .	4
2	TD2	5
2.1	Exercice 1 . . . . .	5
2.1.1	Question 1 . . . . .	5
2.1.2	Question 2 . . . . .	5
2.2	Exercice 2 . . . . .	5
2.2.1	Question 1 . . . . .	5
2.2.2	Question 2 . . . . .	5
2.2.3	Question 3 . . . . .	5
2.3	Exercice 3 . . . . .	5
2.3.1	Question 1 . . . . .	5
2.3.2	Question 2 . . . . .	5
2.3.3	Question 3 . . . . .	6
2.3.4	Question 4 . . . . .	6
3	TD3	7
3.0.1	Question 1 . . . . .	7
3.0.2	Question 2 . . . . .	7
3.0.3	Question 3 . . . . .	8
3.0.4	Question 5 . . . . .	8

4	TD 4	9
4.1	Exercice 1 . . . . .	9
4.1.1	Question 1 . . . . .	9
4.1.2	Question 2 . . . . .	9
4.2	Exercice 2 . . . . .	9
4.2.1	Question 1 . . . . .	9
4.2.2	Question 2 . . . . .	10
4.2.3	Question 3 . . . . .	10
4.2.4	Question 4 . . . . .	10
4.3	Exercice 3 . . . . .	10
4.3.1	Question 1 . . . . .	10
4.3.2	Question 2 . . . . .	10
4.3.3	Question 3 . . . . .	11
4.3.4	Question 4 . . . . .	11

# 1 TD1

## 1.1 Exercice 1

### 1.1.1 Question 1

L'intérêt du modèle en couche est la facilité d'abstraction et l'isolation des couches.

### 1.1.2 Question 2

Le modèle de couche TCP/IP contient 4 couches:

1. Application
2. Transport
3. Internet
4. NetworkAccess

### 1.1.3 Question 3

Les deux modèles sont des modèles à couches, mais le TCP/IP en contient moins.

Couches du modèle OSI:

1. Application
2. Présentation
3. Session
4. Transport
5. Network
6. DataLink
7. Physical

### 1.1.4 Question 4

La couche permettant de faire abstraction de la manière dont les différents réseaux locaux sont interconnectés est la couche Network pour le modèle OSI et la couche Internet pour le modèle TCP/IP.

## 1.2 Exercice 2

### 1.2.1 Question 1

Protocoles de la couche DataLink:

- WiFi
- Ethernet
- Bluetooth

### 1.2.2 Question 2

Le rôle de la couche DataLink est d'assurer la communication entre plusieurs machines connectées sur un même réseau local.

### 1.3 Exercice 3

#### 1.3.1 Question 1

Le protocole Ip appartient à la couche Internet.

#### 1.3.2 Question 2

Afin de connaître la destination d'un paquet, on regarde l'adresse dans la paquet.

#### 1.3.3 Question 3

La principale amélioration de L'IPv6 sur l'IPv4 est la quantité d'adresses disponibles.

#### 1.3.4 Question 4

On a choisi des adresses de 128 bits pour deux choses:

1. La quantité d'adresses disponibles.
2. La généralisation des adresses à partir de l'adresse Mac.

### 1.4 Exercice 4

$$\frac{vc}{d} < t \Rightarrow \frac{36}{d} < 1,2 \Rightarrow d < \frac{5 \times 16000}{2,4} \Rightarrow d < \frac{400000}{12} \Rightarrow d < 33333,333 \text{ m}$$

Cette solution prends en compte le retour du disque vers le point de départ.

Si on compte un allé simple, on fait  $33,333 \times 2 = 66,67km$ , ou bien 66666,67 m.

## 2 TD2

### 2.1 Exercice 1

#### 2.1.1 Question 1

Afin de représenter l'adresse de ce serveur, on utilise la classe `InetAddress` de Java.

#### 2.1.2 Question 2

```
1 public static InetAddress getInetAddressByName(String hostname){
2     return InetAddress.getByAddress(hostname);
3 }
```

Listing 1 – code pour récupérer une adresse IP depuis un String

### 2.2 Exercice 2

#### 2.2.1 Question 1

Les classes importantes pour UDP sont `DatagramSocket` et `DatagramPacket`.

#### 2.2.2 Question 2

Afin de créer un `DatagramPacket`, un client à besoin de:

- Le buffer.
- La taille du buffer.
- L'adresse du serveur.
- Le port d'écriture.

#### 2.2.3 Question 3

```
1 public static void receiveTextFromUdp(){
2     DatagramSocket s = new DatagramSocket(ssss);
3     byte msg[] = new byte[10];
4     DatagramPacket n = new DatagramPacket(msg, 40);
5     s.receive(n);
6     System.out.println(new String(msg));
7     s.close();
8 }
```

Listing 2 – code pour recevoir du texte via protocole UDP

### 2.3 Exercice 3

#### 2.3.1 Question 1

La différence entre les transmission UDP et TCP est que UDP envoie un flux unidirectionnel et plus rapides, mais a des pertes, alors que le TCP est plus lent mais n'a aucune perte.

#### 2.3.2 Question 2

Les données UDP se transmettent plus vite.

### 2.3.3 Question 3

```
1 Socket s = new Socket(adresse, port);
2 ObjectOutputStream output = new ObjectOutputStream(s.
  getOutputStream());
3 ObjectInputStream input = new ObjectInputStream(s.getInputStream)
  ;
4 input.read();
5 output.write();
6 s.close();
```

Listing 3 – code pour récupérer des données d'un socket

### 2.3.4 Question 4

```
1 public static void receiveMessageTCP(){
2     Socket s = new Socket(myServer, ssss);
3     InputStream in = s.getInputStream();
4     BufferedReader b = new BufferedReader(new InputStreamReader(in)
5     );
6     System.out.println(b.readLine());
7     s.close();
8 }
```

Listing 4 – code pour recevoir du texte via protocole TCP

### 3 TD3

#### 3.0.1 Question 1

L'unité consommée n'est pas nécessaire pour le SYN, onn le fait plus par habitude, mais elle l'est pour le FIN.

#### 3.0.2 Question 2

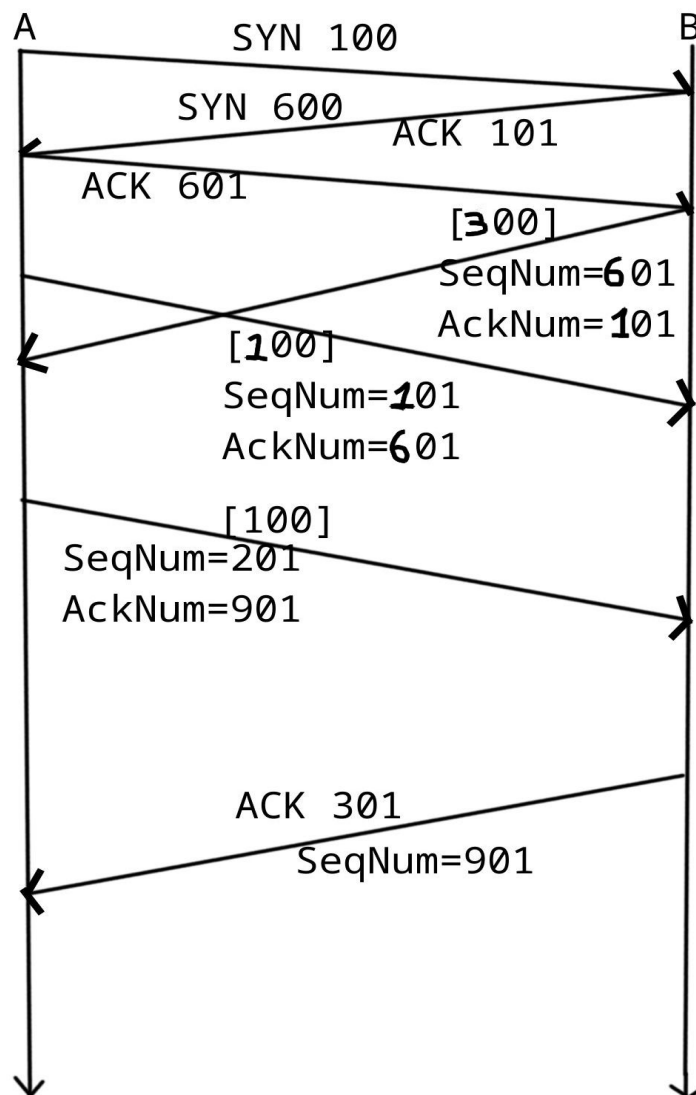


FIGURE 1 – Chronogramme d'échange entre A et B

Si le premier segment de données de A se perd, alors l'AckNum de B contiendra 100 de moins que le SeqNum de A, ce qui pousse B à envoyer un ACK pour indiquer qu'il manque le packet à partir de 101.

### 3.0.3 Question 3

Résumé du chronogramme:

- Envoie d'abord de 'a', attente d'ACK/tampon plein.
- Envoie "bcde", attente d'ACK/tampon plein.
- Envoie "fghi".

Si la connection est de type "TelNet full duplex", les observations sont les mêmes.

### 3.0.4 Question 5

Elle va grandir exponentiellement jusqu'à la moitié de la taille de la fenêtre avant le crash, 9 Ko dans ce cas.

Elle sera donc de la taille du seuil au bout de 4 transmissions acquitées, si tout se passe bien.

Après le seuil, au lieu de faire une croissance exponentielle, on incrémente seulement la quantité d'octets qui passent.



## 4 TD 4

### 4.1 Exercice 1

#### 4.1.1 Question 1

Afin d'utiliser les threads en Java, il faut créer une classe que étend la classe Thread. La méthode principale pour un thread est start.

#### 4.1.2 Question 2

```
1 public class SocketHandler implements Runnable{
2     private Socket socket;
3     private List<String> messages;
4
5     public SocketHandler(Socket socket){
6         this.socket = socket;
7         messages = new ArrayList<String>();
8     }
9
10    // TODO: Ajouter la gestion d'exceptions
11    @Override
12    void run(){
13        BufferedReader input = new BufferedReader(new
14        InputStreamReader(this.socket.getInputStream()));
15        String line;
16        while((line = input.readLine()) != null) this.messages.add(
17        line);
18
19        socket.close();
20    }
21
22    public Socket getSocket(){
23        return this.socket;
24    }
25
26    public List<String> getMessages(){
27        return this.messages;
28    }
29
30    public void setSocket(Socket socket){
31        this.socket = socket;
32    }
33 }
```

Listing 5 – code de la classe SocketHandler

### 4.2 Exercice 2

#### 4.2.1 Question 1

On peut lui faire créer un thread par client TCP lu.

```
1 ServerSocket server = new ServerSocket(1234);
2 while(true)
3     new Thread(new SocketHandler(server.accept())).start();
```

Listing 6 – Serveur TCP multi-thread en trois lignes

### 4.2.2 Question 2

Comportement erroné (duplication potentielles) et imprévisible.

### 4.2.3 Question 3

La structure ne sera modifiée que par un seul à la fois via un système de sémaphores et de section critiques, si la structure est sérialisée.

```
1  class Data{
2      public int data=0;
3  }
4
5  class Job implements Runnable{
6      Data d;
7
8      Job(Data d){
9          this.data = data;
10     }
11
12     void run(){
13         int newData = d.data+1;
14         d.data = newData;
15     }
16 }
17
18 public static void main(String[] args){
19     Data d = new Data();
20
21     Thread A = new Thread(new Job(d).start());
22     Thread B = new Thread(new Job(d).start());
23
24     System.out.println(d.data);
25 }
```

Listing 7 – Exemple d'écriture sur la même structure de données par deux threads

La donnée affichée sera soit 1 soit 2.

### 4.2.4 Question 4

Il n'y a aucun moyen.

## 4.3 Exercice 3

### 4.3.1 Question 1

Pour la communication entre un socket server et un socket client, il faut utiliser les classes `ServerSocketChannel` et `SocketChannel` de `java.nio.channels`.

### 4.3.2 Question 2

Utiliser Java NIO pour la communication entre sockets a comme avantages:

- non-bloquant
- pas de threads
- plus optimisé

Il est globalement très efficace.

### 4.3.3 Question 3

On utilise un Selector pour les communications entre sockets car il trouve ce qu'il faut faire dans l'instant. C'est-à-dire, il dit quel channel est prêt à travailler, et comment.

```
1 selector.select();
2 Set<SelectionKey> selectedKeys = selector.selectedKeys();
3 Iterator<SelectionKey> iterator = selectedKeys.iterator();
4 while(iterator.hasNext()){
5     SelectionKey key = iterator.next();
6 }
```

Listing 8 – Exemple selector

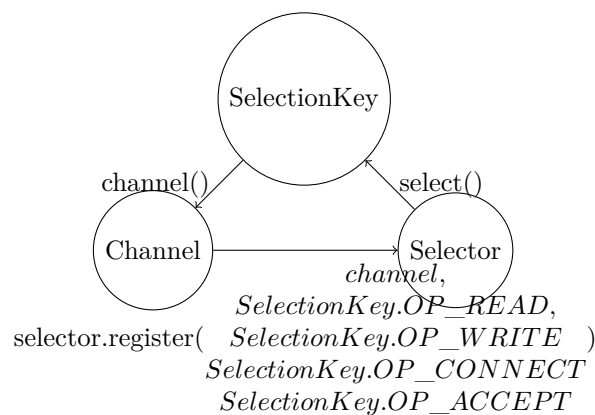


FIGURE 2 – Schéma classes

Les clés en majuscules sont à choisir en fonction de ce dont on a besoin.

### 4.3.4 Question 4

```
1 Selector selector = Selector.open();
2 ServerSocketChannel server = ServerSocketChannel().open();
3 server.configureBlocking(false);
4 selector.register(server, SelectionKey.OP_ACCEPT);
5 // Exemple selector, code ensuite dans la boucle while
6 if(key.isAcceptable()){
7     SocketChannel client = server.accept();
8     selector.register(client, (SelectionKey.OP_WRITE|SelectionKey.OP_READ));
9     client.configureBlocking(false);
10 }
11 if(key.isReadable()){
12     SocketChannel client = key.channel();
13     // lecture
14 }
```

Listing 9 – Serveur Mono-Thread qui gère jusqu'à 10 clients en simultanés