CSC4509 — communication sous TCP

Éric Lallet
Eric.Lallet@telecom-sudparis.eu

Télécom SudParis

26 avril 2021

Le but de cette présentation est de résoudre trois problèmes :

1. Découper les messages d'un flux TCP
2. Gérer la représentation des données dans le message
3. Transmettre des objets en Java (sérialisation)

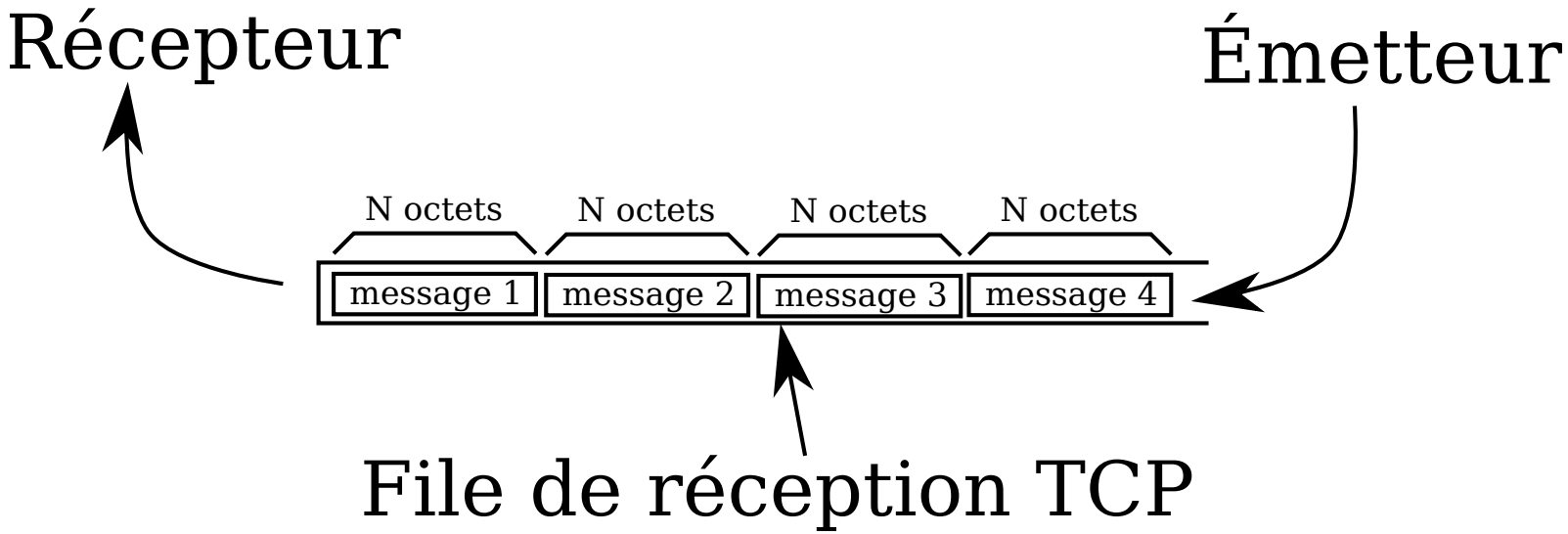
La file de réception de TCP ne contient pas des paquets, mais un flot continu d'octets. TCP ne permet pas de :

- Savoir si les octets arrivés forment un message cohérent et terminé
- Connaître la frontière en les octets concernant un message et ceux concernant le message suivant

Pour résoudre ce problème, il existe plusieurs solutions :

- Échanger des paquets de taille fixe
- Utiliser un délimiteur de fin de paquet (comme un caractère réservé)
- Échanger des trames de taille variable débutant avec une entête de format fixe indiquant la taille de la trame
- etc.

Messages de taille fixe

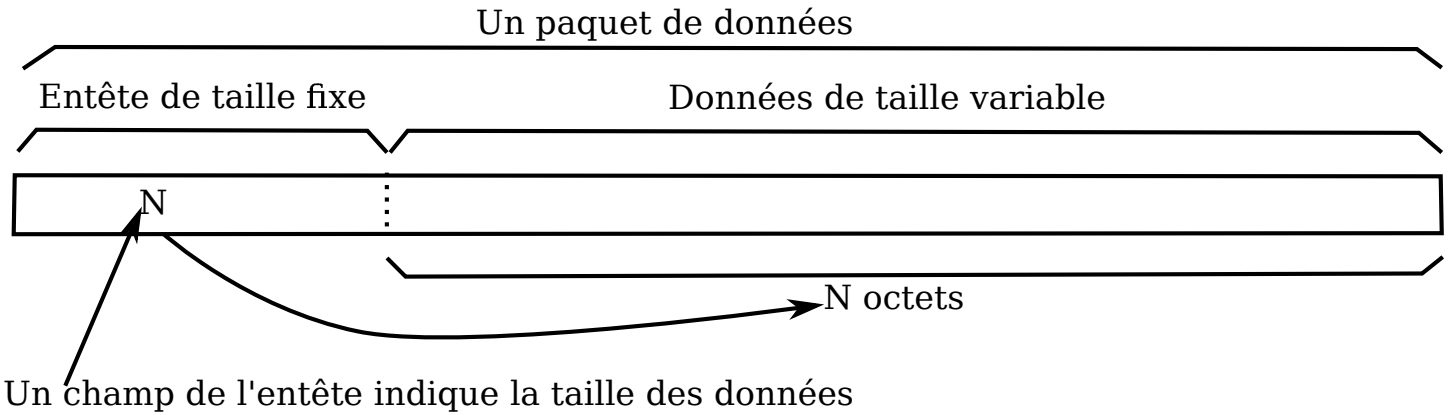


Délimiteur de fin de paquet

De nombreux protocoles textuels utilisent le passage à la ligne (le caractère « \n ») comme délimiteur.
Par exemple un échange SMTP ressemble à cela :

```
HELO b06-13.int-evry.fr\n
MAIL FROM: nom.prenom@mon.adresse.net\n
RCPT TO: destinataire@son.adresse.net\n
DATA\n
Subject: sujet du courrier\n
Texte du courrier\n
Texte du courrier\n
Texte du courrier\n
.\n
```

Messages avec entête de taille fixe



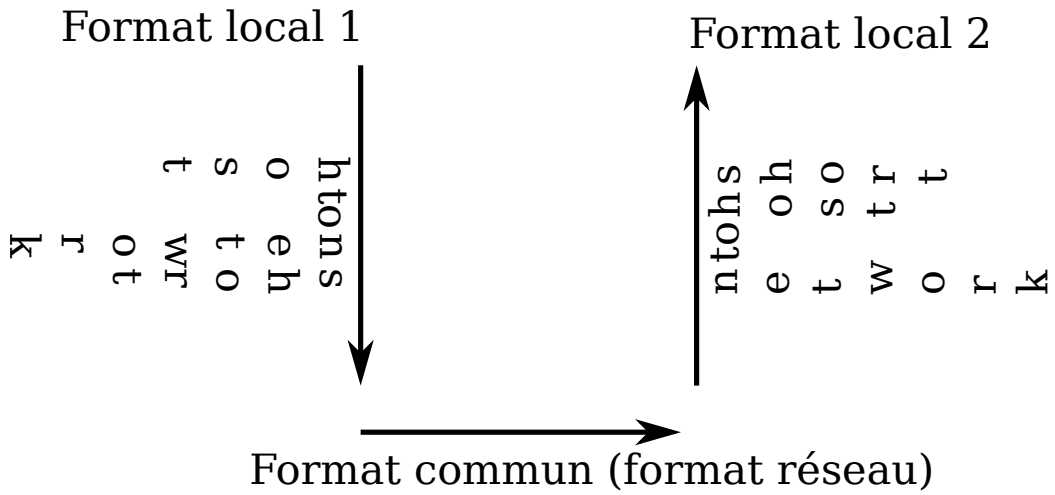
Présentation des données

Un réel, un entier ou même un caractère peuvent être codés différemment pour une même valeur. Il faut se mettre d'accord avec l'interlocuteur.

- Utiliser un langage qui spécifie le format du codage des types (comme Java)
- Transformer dans une représentation binaire commune pour les échanges (les fonctions *htons()*, *ntohs()* du `C`)
- Utiliser un formalisme qui décrit la donnée (`JSON`, `XML`, `ASN 1`...)
- Utiliser une représentation textuelle en `ASCII` 7bits (solution de `FTP`, `HTTP`...)

Présentation des données : exemple du C ou C++

Les langages style C utilisent une bibliothèque de fonctions qui transforment le format local des données (format « host ») dans un format commun (format « network »), et réciproquement : *htons()*, *ntohs()*, *htonl()*, *ntohl()*...

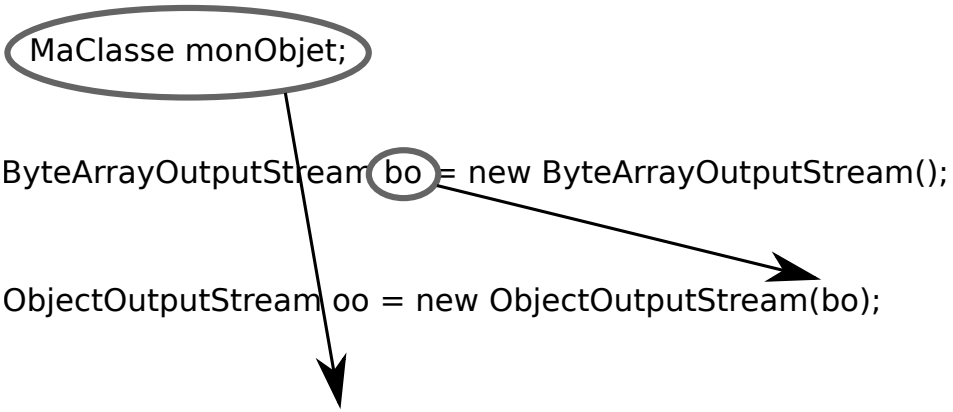


Sérialisation

En Java, les types primitifs ne dépendent pas de l'architecture de la machine : aucun problème pour les transmettre.
Mais si on veut transmettre un objet complet ?
Il faut sérialiser :

- Une classe pour transformer un objet en un flux d'octets : ObjectOutputStream
- Une classe pour retransformer ce flux d'octets en objet : ObjectInputStream

Sérialisation en Java



Un objet créé et initialisé d'une classe Serializable

Création du flux où va être envoyée
la sortie de la sérialisation

Création de l'usine à sérialiser.
Le flux de sortie est fourni en paramètre

Sérialisation de l'objet. Le flux de sortie contient
l'objet sous sa forme sérialisée.

Buffeur où l'on va stocker l'objet sérialisé

Écriture de l'objet sérialisé dans le buffeur
(prêt à être envoyé).

Désérialisation en Java

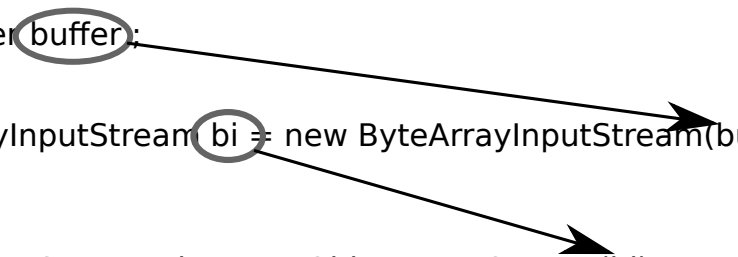
```
MaClasse monObjetRestitué = null;

ByteBuffer buffer;

ByteArrayInputStream bi = new ByteArrayInputStream(buffer.array());

ObjectInputStream oi = new ObjectInputStream(bi);

MonObjetRestitué = (MaClasse) oi.readObject();
oi.close();
bi.close();
```



Référence (pas encore instanciée) pour l'objet à restituer.

Buffeur déjà créé, et contenant les octets de l'objet sous sa forme sérialisée.

Création du flux où va être lu l'objet à désérialiser.

Création de l'usine à désérialiser.
Le flux d'entrée est fourni en paramètre.

Désérialisation de l'objet. L'objet restitué est une copie exacte de l'objet de départ.