1 - Préambule

Contenu du chapitre

- *Une lettre ou un appel ?*
  - Transport de données entre un client et un serveur à travers UDP et TCP avec le modèle datagramme, et les approches connecté et non connecté.
    Gestion et utilisation de l'API socket

2 - Objectifs de la couche transport

Introduction

C'est une **couche clé** du modèle en couche, car à la frontière entre les **fournisseurs** et les **utilisateurs** de la transmission de données

Couche transport

Couche réseau

Client (utilisateur)

Opérateur réseau (transporteur)

Fig 5.1 - Interface Couche transport / couche réseau

2 - Objectifs de la couche transport

Introduction

Les couches inférieures agissent sur les machines intermédiaires, alors que la couche transport est une vrai couche de bout en bout

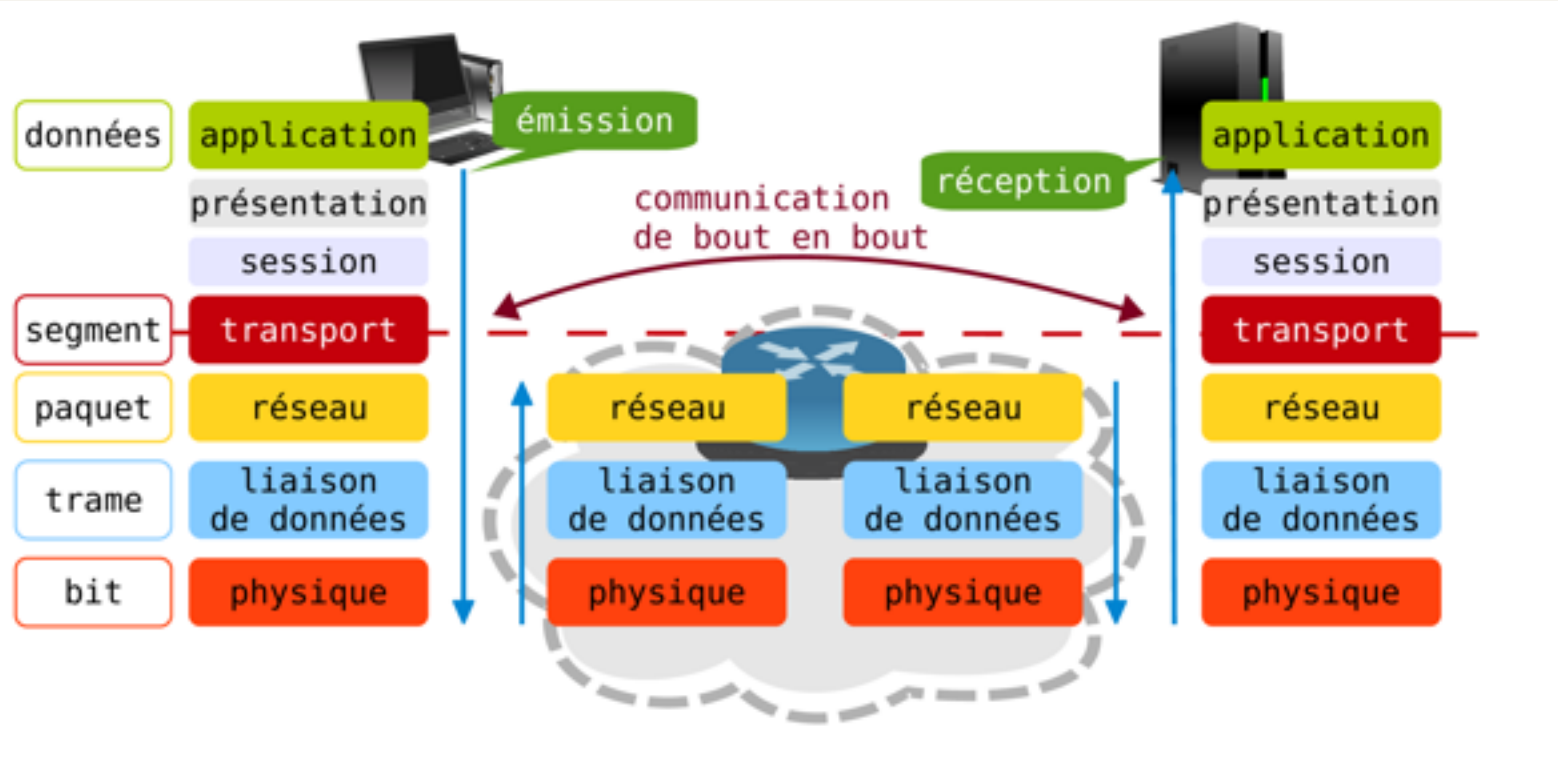


Fig 5.2 - Place de la couche transport dans le modèle OSI

Le service de transport

L'objectif de la couche transport est de fournir à l'utilisateur un service de transport

- efficace
- fiable ou économique
- Indépendamment de la nature du ou des réseaux utilisés

L'entité de transport est la partie logicielle qui assure ces services. Elle est intégrée au système d'exploitation (ou dans un paquetage de bibliothèque).

Types de services

- Services **avec connexion**
- Services **sans connexion**

La qualité de service

QoS (*Quality of Service*) ; elle est liée à différents paramètres :

- temps d'établissement de la connexion
- probabilité d'échec de la connexion
- débit de la liaison
- temps de transit (latence)
- taux d'erreur résiduel
- protection (contre écoutes, intrusions, etc.)
- priorité
- résiliation (probabilité que la couche transport provoque elle même une déconnexion)

Négociation d'options

- L'utilisateur indique les valeurs souhaitées et minimales
- la couche transport propose ces options à la machine distante qui indique ses contre-propositions.

2 - Objectifs de la couche transport

Les primitives de service

Elles permettent l'accès aux services de transport.

- Il s'agit des primitives d'une API (*Application Programming Interface*)
- Elles sont utilisées par les développeurs d'applications afin de, par ex. :
  - Établir une connexion
  - Utiliser et exploiter cette connexion
  - Libérer la connexion

Exemple de l'interface de connexion par **socket** pour TCP dans l'UNIX de **Berkeley**. Voir par ex.

- Package java.net
- inetdoc.net/pdf/socket-c.pdf

Primitive	Description
`socket`	Créer une nouvelle prise de communication
`bind`	Attacher une adresse locale au socket
`listen`	Annoncer l'acceptation de connexion et la taille de la file d'attente
`accept`	Bloquer l'appelant jusqu'à une tentative de connexion
`connect`	Tenter activement d'établir une connexion sur une socket distante
`send`	Envoyer des données via la connexion
`receive`	Recevoir des données de la connexion
`close`	Fermer la connexion

Fig 5.3 - Primitives de socket

3 - Les protocoles de transport

Généralités

Comme pour la liaison de données, le protocole de transport fiable a un rôle de **contrôle** d'**erreurs**, de **séquencement** et de **flux**. Mais l'environnement est très différent.

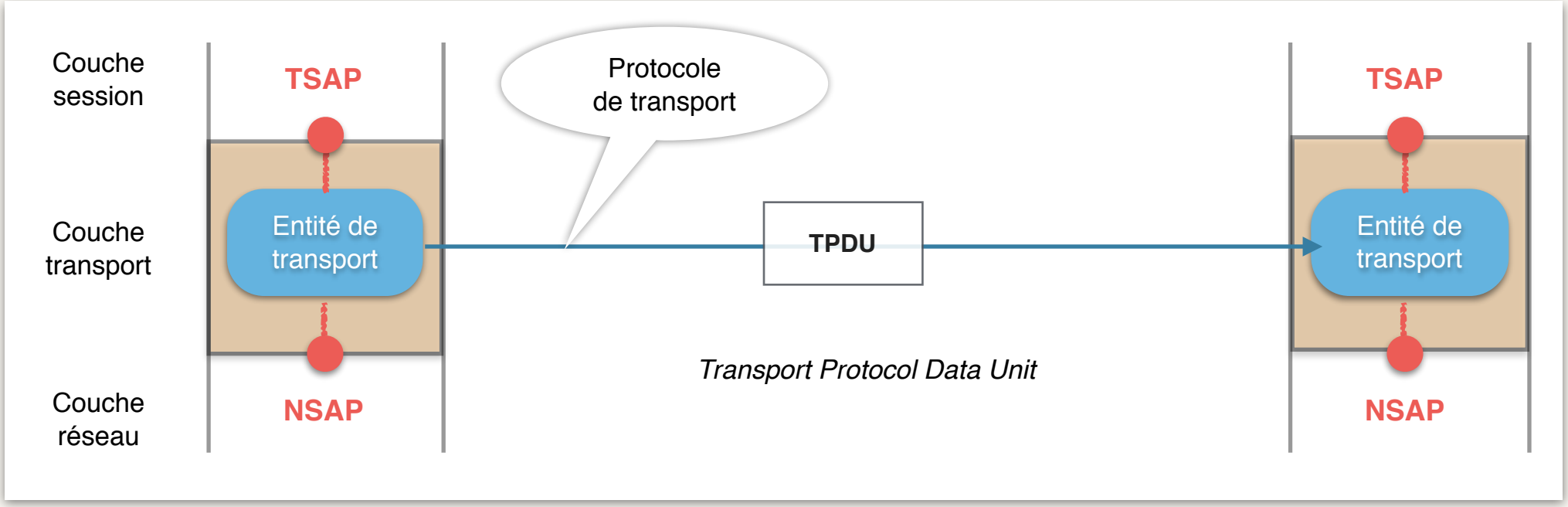


Fig 5.4 - Protocole de transport

Généralités

En liaison de données, l'entité paire est située de l'autre côté de la ligne physique

Pour la couche transport :

- L'adresse de l'entité destinataire doit être spécifié (TSAP)
- Le sous-réseau peut emmagasiner des paquets qui réapparaissent subitement après (mode datagramme)
- La gestion du flux et celle du séquencement seront adaptées au fait que le couche transport gère un nombre important et fluctuant de connexions.

Adressage

Pour l'envoi d'un message ou pour établir une connexion, on doit spécifier à qui s'adresser.

Il s'agit de définir les points d'accès au service de transport

- **TSAP,** *Transport Service Access Point*
- Pour internet, un TSAP est une **adresse de socket** (adresse IP + n° de port)

En général, le TSAP est lié à un service (fourni par un serveur)

Comment le déterminer ?

- Le TSAP est fixe et connu de la source
- La source appelle un annuaire (un service de noms) pour connaitre l'adresse du service à partir de son nom
- Avec un protocole de pré-connexion :
  - Un serveur de processus agit comme délégué (proxy) des serveurs peu utilisées
  - Il écoute un ensemble de ports en attente de connexion
  - Suivant le service demandé par la source lors de la connexion le délégué active le serveur et lui transmet la connexion

Établissement d'une connexion

Une connexion entre deux hôtes est réalisée en **trois étapes** ; la méthode se nomme ***three-way handshake***

- Les numéros de séquence x et y sont choisis de façon aléatoire
- A envoie **CR**, *Connection Request*
- B envoie **Ack**, *Acknowledgement* (accusé de réception)
- A envoie des données

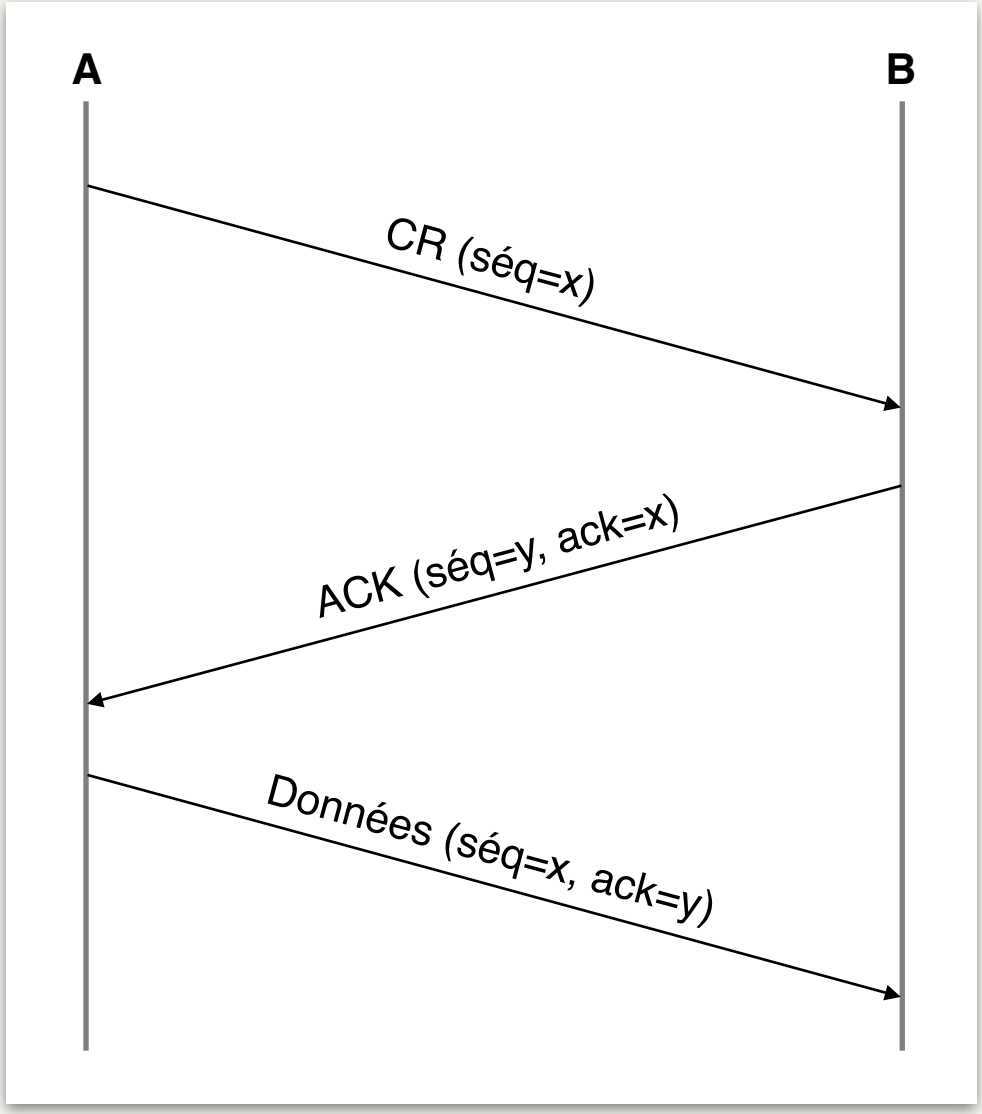


Fig 5.5 - Three-way handshake

Libération d'une connexion

Une libération asymétrique implique un risque de perte de données

On préfère donc une libération symétrique

- Chaque hôte envoi un « *Disconnection request* » et arme un temporisateur
- La figure correspond à une libération normale
- Une déconnexion sera, au pire, réalisé à l'expiration d'un temporisateur (en cas de perte d'un des messages)

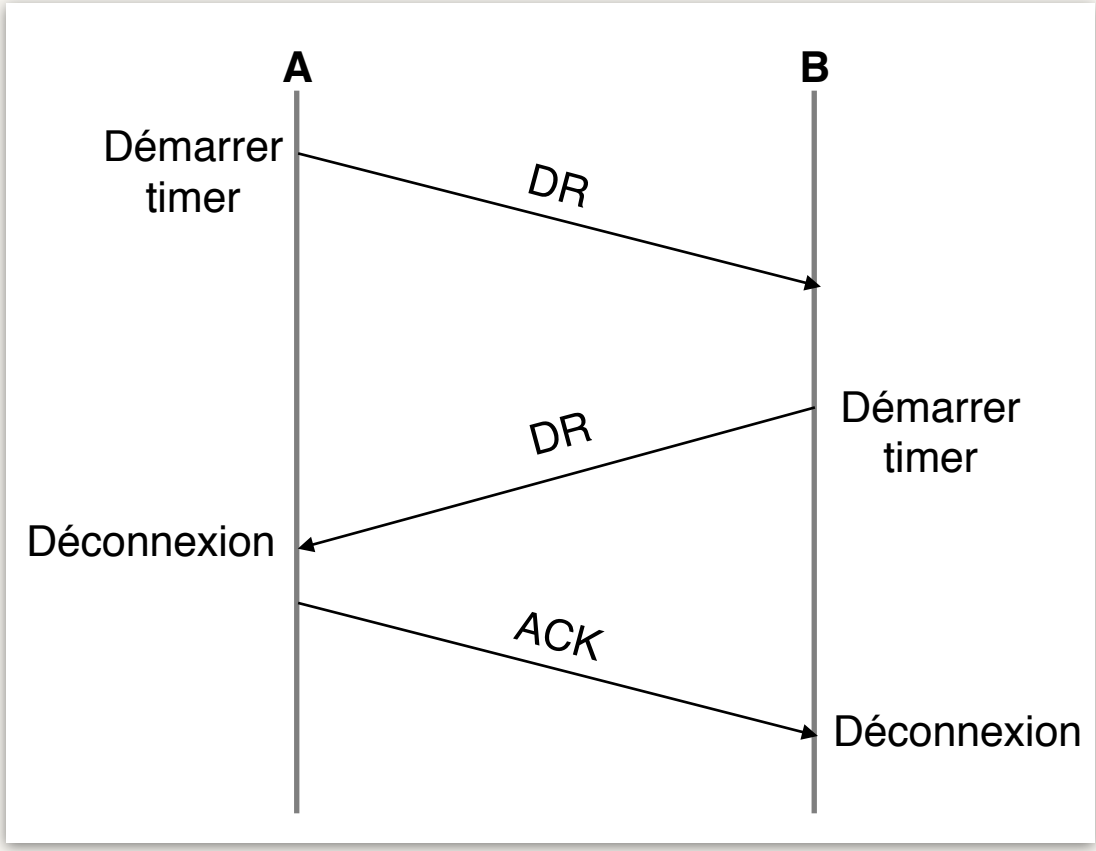


Fig 5.6 - Libération de connexion

Préambule

- Les deux principaux protocoles de la couche transport de l'architecture TCP/IP sont :
  - **TCP** (*Transmission Control Protocol*) propose un service fiable orienté connexion
  - **UDP** (*User Datagram Protocol*) est un protocole plus simple, non fiable et sans connexion
- On trouve également :
  - QUIC : nouveau protocole de transport utilisant TLS/SSL et UDP.
  - RTP (*Real-Time Transport Protocol*) permet le transport de données soumises à des contraintes de temps réel, tels que des flux média audio ou vidéo.
    - Il utilise en fait UDP et il permet le transport de média pour les services de la **voix sur IP**, de **vidéo conférence** et de **streaming**.
    - RTP, protocole de transport, est toujours associé à un protocole de niveau Application comme SIP, *Session Initiation Protocol* (VoIP ou vidéo conférence) ou **RTSP**, *Real Time Streaming Protocol*.

TCP - Introduction

- Conçu pour traiter des flots de données de bout-en-bout, de manière fiable, sur un ensemble de réseaux non fiables.

- TCP s'adapte dynamiquement aux variations de paramètres des réseaux (topologie, largeur de bande, délais de transmission, taille de paquets…)

- Résistant aux pannes de la couche réseau et aux pertes de datagrammes.

Le service TCP

- **Notion de socket**
- les connexions TCP sont bidirectionnelles, en mode point à point entre deux adresses de sockets

- Un **socket** est définie par l'association de :
  - Protocole (UDP ou TCP)
  - Adresse IP de la source
  - Port de la source
  - Adresse IP de destination
  - Port de destination

- On peut considérer un **socket** comme une « *prise réseau* »

Le service TCP

- Un numéro de port fait 16 bits
- Certains numéros (< 1024) sont réservés et liés à des services (*Well known ports*)
  - 20 : FTP (data)
  - 21 : FTP (control)
  - 22 : SSH
  - 23 : Telnet
  - 25 : SMTP
  - 80 : HTTP
  - 993 : IMAPS

- Voir le fichier unix **/etc/services**

- Les ports référencés permettent à une application cliente d'identifier une application et un service sur un système distant (un serveur).

Le service TCP

- Groupement de données.
  - La délimitation des messages délivrés par un processus n'est généralement pas conservée.
  - TCP traite des flots d'octets et attends, avant de transmettre des données, que le buffer d'émission soit plein.

  - Dans l'en-tête de TCP un drapeau, PUSH, peut-être utilisé par une application afin que les données soient délivrées immédiatement
  - Cela est notamment utilisé avec l'action de la touche « Entrée » d'un terminal virtuel

Le protocole TCP

- Les échanges sont sous forme de **segments**
- Un segment :
  - Un en-tête de 20 octets
  - Un en-tête optionnel
  - Des données optionnelles

- La taille maximum d'un segment est de 65 535 octets (charge utile max. pour IP)

- Les données traitées par TCP sont fragmentées en segment de la taille de MTU (*Maximum Transfer Unit*),
  - unité de transfert maximale
  - caractéristique d'un réseau

Le protocole TCP

- TCP utilise un mécanisme de fenêtre d'anticipation pour optimiser la communication
  - À la transmission d'un segment S, l'émetteur arme un temporisateur
  - l'entité TCP réceptrice renvoie un segment avec une n° d'accusé de réception égal au n° de séquence suivant attendue
  - l'émetteur retransmet S si le temporisateur expire avant réception de l'accusé de réception

- Certaine difficultés proviennent du fait que les segments peuvent être fragmentés, et ces fragments peuvent arriver dans un ordre différent de celui d'émission
- TCP sait traiter ces types de difficultés

L'en-tête TCP

- Port source (16 bits)
- Port destination (16 bits)
- N° de séquence (32 bits) ; n° du premier octet envoyé
- N° d'accusé de réception (32 bits) ; n° du prochain octet attendu)
- Longueur d'en-tête (4 bits) ; nombre de mots de 32 bits de l'en-tête
- Réservé (6 bits) ; non utilisés

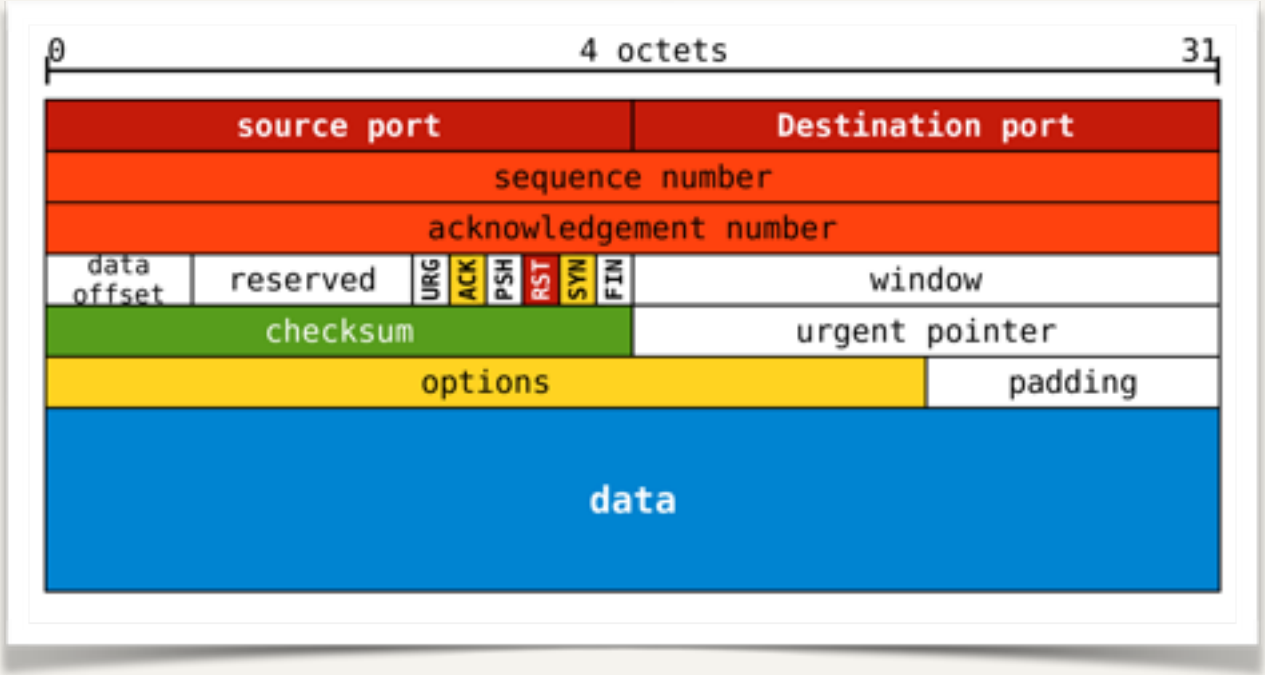


Fig 5.7 - Format d'un segment TCP

L'en-tête TCP

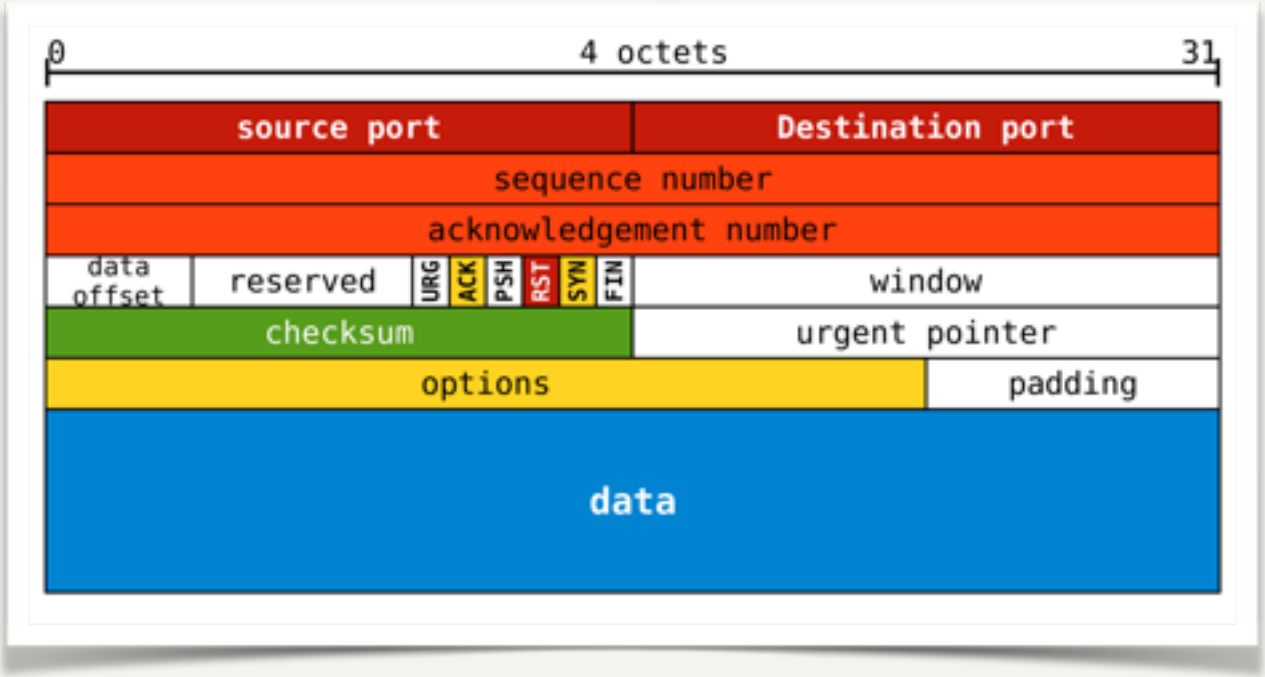


Fig 5.7 - Format d'un segment TCP

- 6 drapeaux d'un bit :
  - URGent
  - ACK à 1 si n° d'AR valide (à 0 => AR négatif)
  - PUSH ne pas bufferiser les données
  - RST à 1 si problème et reset de la connexion nécessaire ou si refus de tentative de connexion
  - SYN pour l'établissement de la connexion
    - SYN = 1 et ACK = 0 => CONNECTION REQUEST
    - SYN = 1 et ACK = 1 => CONNECTION ACCEPTED
  - FIN à 1 pour libérer la connexion

L'en-tête TCP

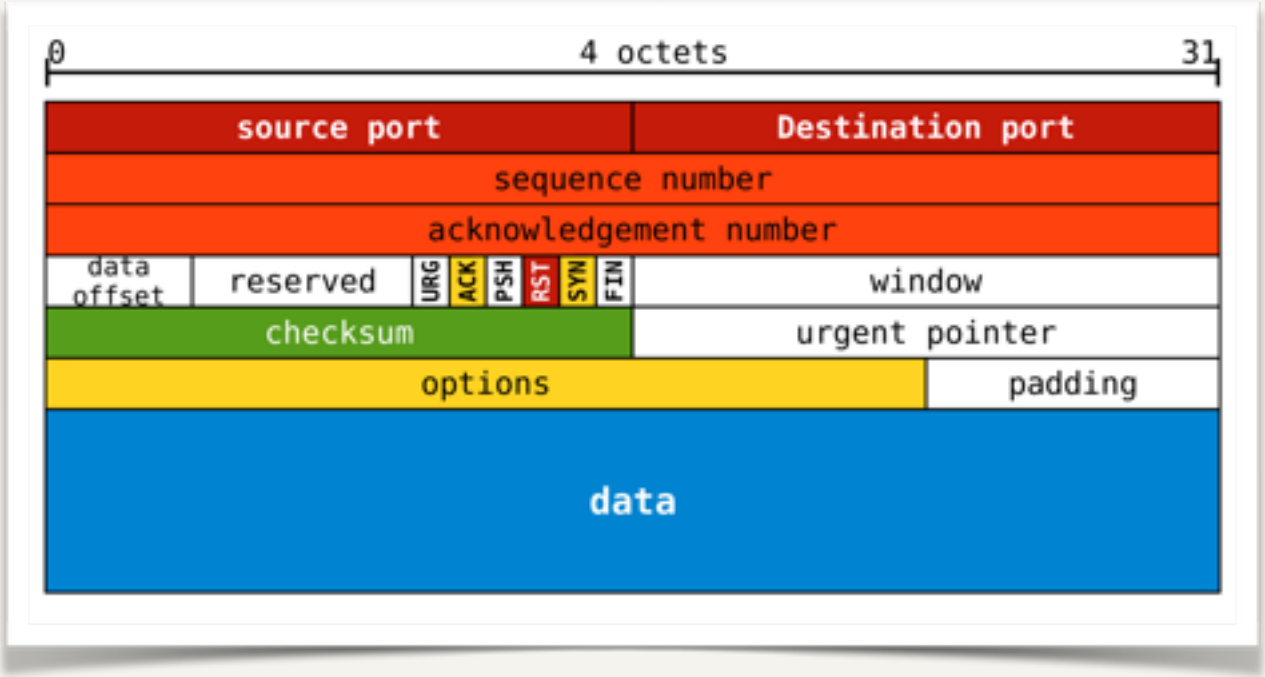


Fig 5.7 - Format d'un segment TCP

- Taille fenêtre (16 bits) ; taille de la fenêtre d'anticipation = nombre d'octets autorisés à être reçu
- Total de contrôle (16 bits) ; sur l'en-tête, les données et un pseudo en-tête
- Pointeur d'urgence (16 bits) ; utilisé si URG=1 pour indiquer le décalage en octets par rapport au n° de séquence
- Options + bourrage (0 ou un multiple de 32 bits)
  - Négociation de la taille du MTU en phase de connexion (536 par défaut)
  - Négociation d'un facteur d'échelle de la taille de la fenêtre d'anticipation
  - etc.

Gestion de la connexion TCP

- Établissement de connexion avec une poignée de main en trois étapes (*Three ways handshake*)
  - Le serveur réalise une ouverture passive d'un port, avec la primitive LISTEN
  - Un client réalise une ouverture TCP active via la primitive CONNECT, en indiquant :
    - l'adresse socket du serveur (@IP + n° de port)
    - le MTU, taille maximum des segments TCP
    - quelques données utilisateurs (ex. identifiant + mot de passe)
  - CONNECT => SYN=1 ; ACK=0 ; Seq=x

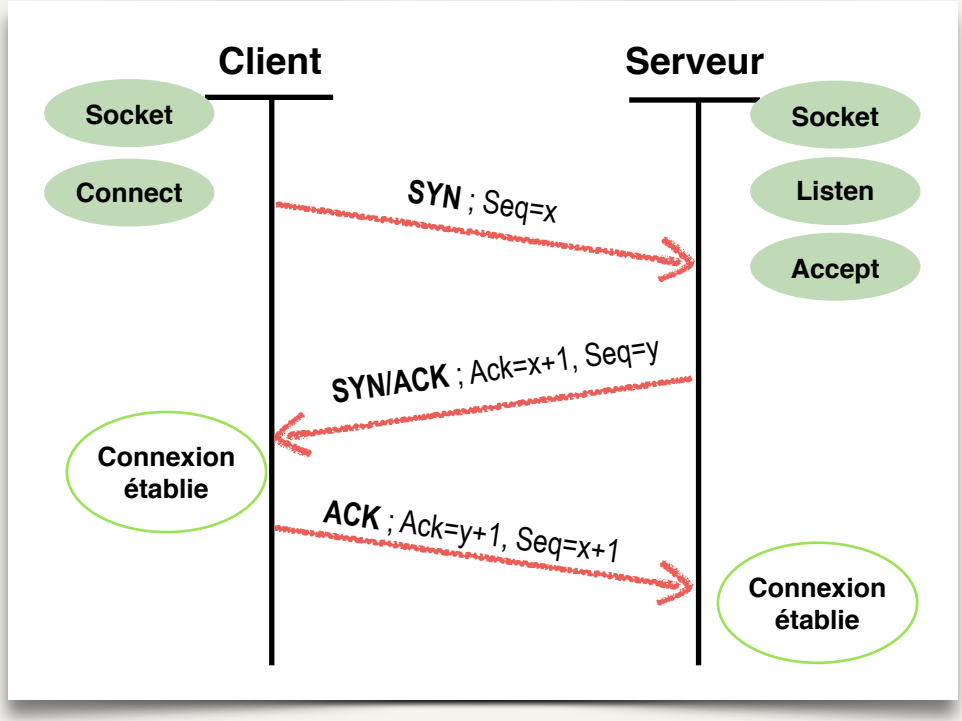


Fig 5.8 - Connexion TCP en 3 étapes

Gestion de la connexion TCP

- Le serveur vérifie que le port demandé est en écoute ; dans ce cas, TCP passe le segment entrant et l'application :
  - **accepte** la demande de connexion (Primitive *Accept*)
    => retour d'un AR
    avec SYN=1 ; ACK=1 ; Seq=y
  - ou **refuse** la demande de connexion
    Rejet => RST=1
- Suite à *Accept* du serveur, le client à sa connexion établie et il retourne un AR
  - => retour d'un AR
    avec ACK=1 ; ack=y+1
- À réception du ACK, la connexion est établie sur le serveur.

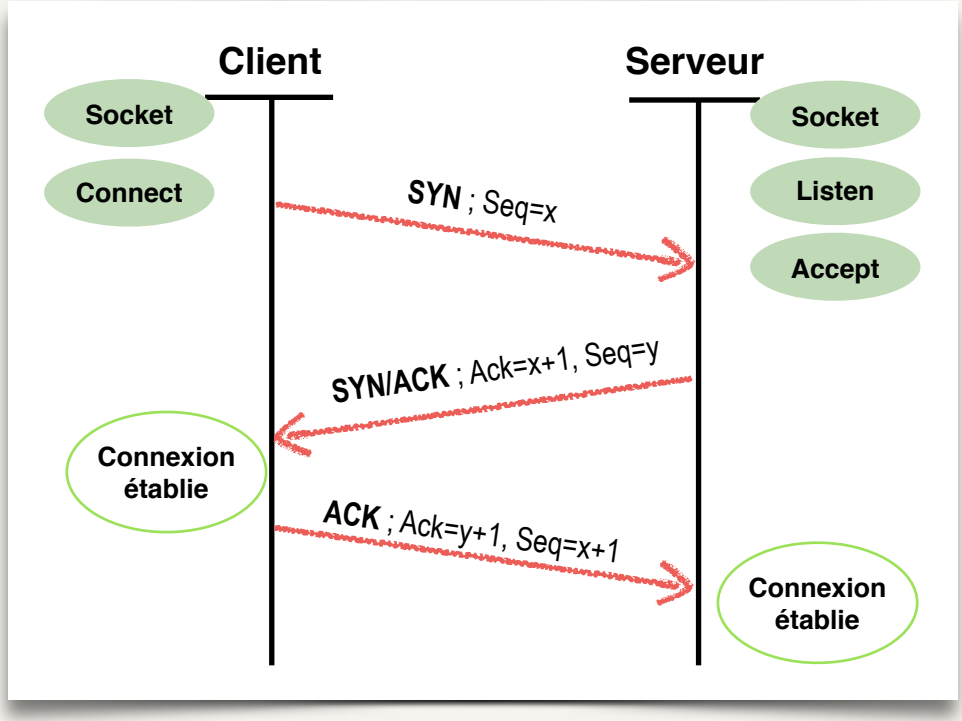


Fig 5.8 - Connexion TCP en 3 étapes

Gestion de la connexion TCP

- La libération de la connexion nécessite 2 FIN et 2 ACK
  - Une partie, A, exécute une primitive CLOSE
    - Il envoie FIN à B et arme un temporisateur
  - B reçoit FIN ; il acquitte (ACK=1) et arme un temporisateur
  - B envoie FIN à son tour
  - A acquitte le FIN de B et ferme la connexion
  - B reçoit le ACK de A et ferme la connexion

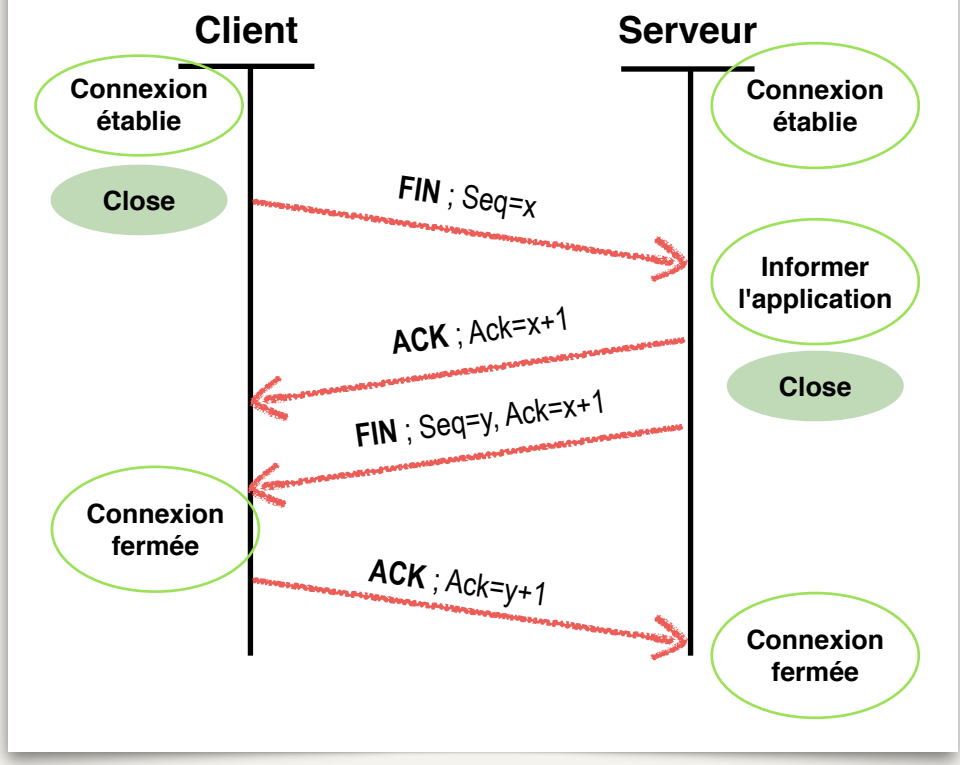


Fig 5.9 - Libération de connexion TCP

- Une des parties qui trouve un temporisateur expiré ferme sa connexion

Fenêtre d'anticipation TCP

- Lié aux contrôle de flux et de congestion
- TCP gère un mécanisme de fenêtre d'anticipation de taille variable
- Tant que le champ taille de fenêtre Win = 0
  - Le récepteur indique que son tampon de réception est plein
  - Cela bloque l'émetteur
- Le récepteur indique avec ce champ combien d'octets, Win, il peut accepter
  - La taille augmente de N_{lu} lorsque l'application a lu N_{lu} octets
  - La taille diminue de $N_{émis}$, nombre d'octet émis par l'émetteur, avec $N_{émis} <= Win$.

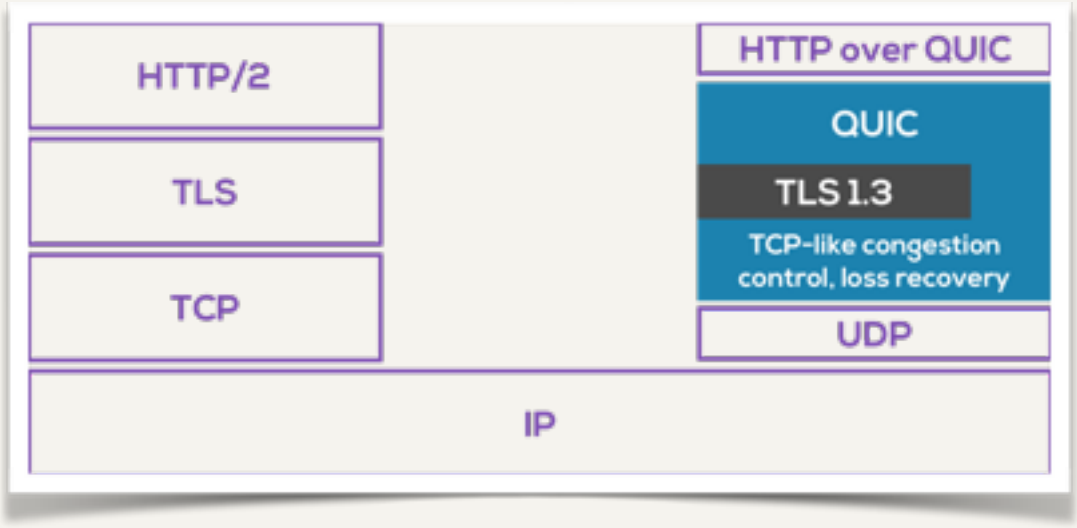
Fenêtre d'anticipation TCP ; gestion des congestions

- Pour réduire la charge sur le réseau et afin que des modifications de fenêtre peu significatives ne soient envoyées :
  - L'émetteur peut regrouper des données de l'application pour éviter d'envoyer des segments trop petits
  - Le récepteur peut attendre un nombre suffisant de données de son application plutôt que de renvoyer des AR sans donnée
- La détection des congestions passe par la surveillance des temporisateurs
- Le choix et le réglage de la taille maximum des fenêtres est très important vis à vis des congestions
- Dans la pratique, une deuxième fenêtre est utilisée : la fenêtre de congestion et l'émetteur n'envoie que le minimum permis par ces deux fenêtres.

UDP : User Datagram Protocol

- Service en mode non connecté
- Les applications utilisent UDP pour encapsuler des données et pour les envoyer sans établir de connexion
- UDP est dédié à des applications :
  - client-serveur ; ex. DNS (*Domain Name System*)
  - en mode dialogue, avec des couples de demande / réponse
  - de streaming (lecture en continu)
  - jeux en réseau…

- QUIC est un protocole de transport qui utilise UDP.
  - QUIC intègre TLS 1.3, *Transport Layer Security* 1.3
  - HTTP/3 s'appuie sur QUIC



UDP : User Datagram Protocol

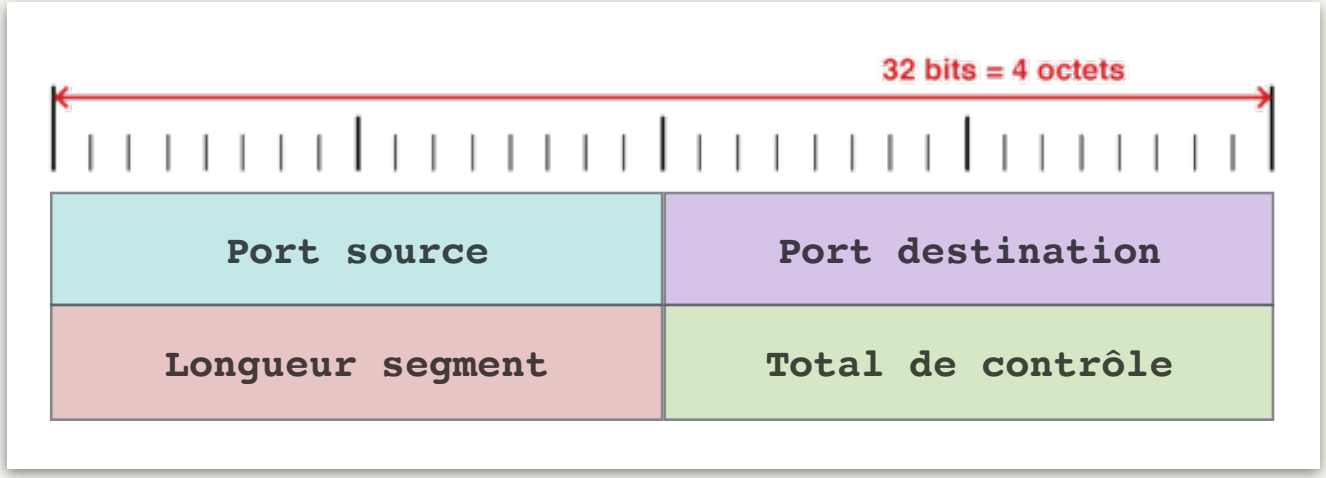


Fig 5.10 - L'en-tête UDP

- En-tête de **8 octets**
- 4 champs de 16 bits
- Port source et destination
  - idem TCP
- Longueur du segment
  en octets, en-tête inclus.
- Total de contrôle (ou 0 s'il n'est pas calculé)
  comme TCP, sur :
  - Pseudo en-tête IP (adresse source, adresse destination, protocole, taille UDP)
  - En-tête UDP
  - Données