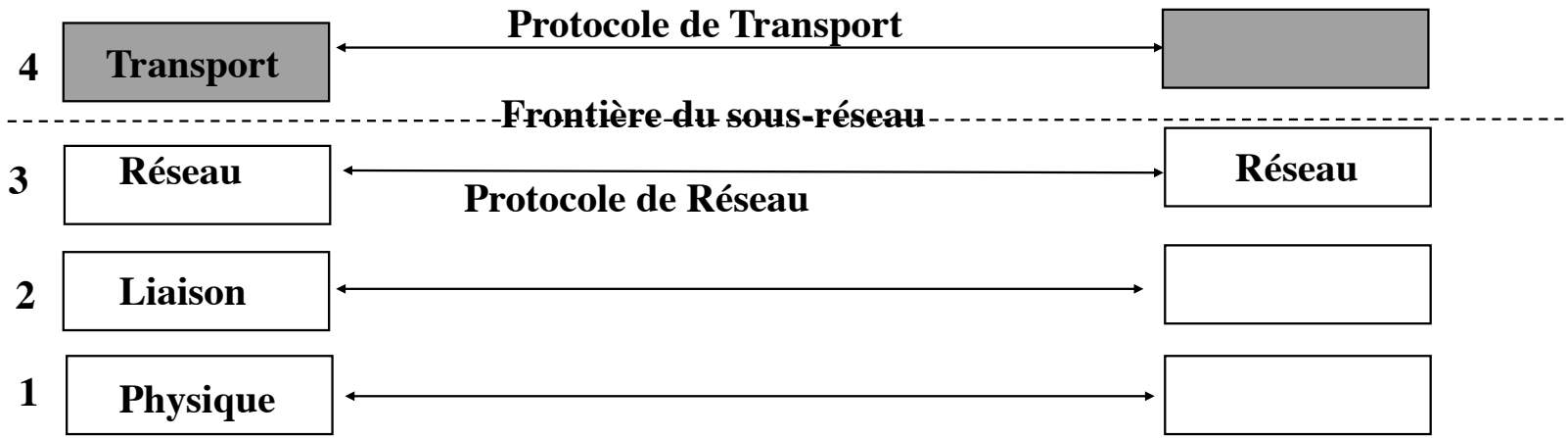
La couche Transport



Services de la couche transport

- **Transfert fiable efficace, sûr et économique d'informations de bout en bout**
- **Dans l'idéal, répondant à des QoS diverses**
- **Dernière couche avant les "applications"**
- **Les qualités de services nécessaires aux applications**
  - **Très variables**
  - **Exemples:**
    - **Transfert de fichier: Taux erreur nul, débit n'est pas primordial, le temps de transit non plus**
    - **Téléphone: Taux d'erreur peu être non nul, débit minimum indispensable, temps de transit minimum (< 0,25 s)**

Les services de la couche transport d'Internet

- Rappel: IP sans connexion non fiable, déséquencement possible, délai de transfert très variable
- Protocoles pensés au départ pour le transfert de fichier et l'utilisation de machine à distance
- Deux types de services suivant les besoins de l'application à développer
  - Sans connexion, aucune QoS: User Datagram Protocol (UDP)
  - Avec connexion: Transport Control Protocol (TCP)
    - » Ouverture et fermeture de connexion
    - » Fragmentation et reassemblage
    - » Contrôle de flux et récupération des erreurs
    - » Données urgentes
- TCP et UDP se sont imposés naturellement. Les normes OSI ont été définies en parallèle mais elles sont arrivées trop tard
- Dans les norme OSI il existe différents niveaux de QoS pouvant être utilisés suivant le réseau sous jacent

Fonctionnalités des procoles transport d'Internet

- **Protocoles client/serveur:**
  - » Le serveur se met en attente de demandes
  - » Le client initie le dialogue par une demande
- **Interface des "sockets"**
- **1 serveur : 1 numéro de port fixé**
  - – Côté serveur: réservés pour applications standards
    - – Fichier /etc/services
    - – Exemple: HTTP port 80 en TCP
  - - Côté client: alloués dynamiquement
- **Multiplexage vers les applications:**
  - - Adresses Internet source et destination, numéros de ports source et destination
  - - Entêtes réseau (IP) et transport (TCP ou UDP)

QoS dans Internet

- **Application temps-réel : la QoS de TCP est insuffisant**
- **Niveau Réseau**
  - **RSVP: Ressource Reservation Protocol**
    - **Permet de réserver dans les routeurs des ressources permettant de garantir un débit, délai de traversée...**
    - **C'est le destinataire qui est à l'origine des réservations en fonction de la QoS qu'il espère**
    - **Nécessite des échanges de paquets de signalisation**
  - **Différentiation de service (DiffServ):**
    - **Définition de classes de services de qualités différentes**
    - **Marquage des paquets et traitement prioritaire dans les routeurs**
  - **Ces techniques sont peu utilisées à grande échelle car difficile à mettre en oeuvre**

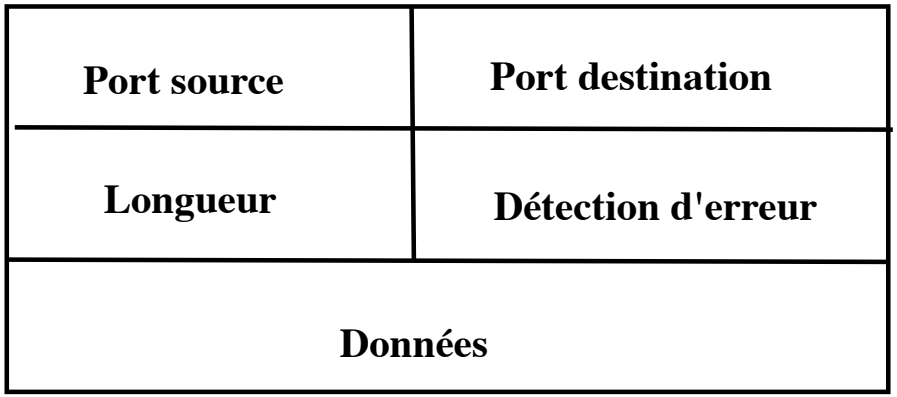
Garanties de QoS

- **Niveau Tansport TCP inutile pour beaucoup d'application "temps-réel"**
  - **Exemple: le téléphone**
    - **Les paquets perdus et réemis arrivent trop tard**
    - **Un taux d'erreur non nul est possible**
  - **Utilisation de UDP**

- **Une couche supplémentaire entre UDP et l'application: RTP (Real Time Protocol) normalisé**
  - **Numérotation des paquets, estampillage temporel, rapports du récepteur à l'émetteur pour signaler: la QoS courante (délai de transit, taux d'erreur, débit…)**
  - **Adaptation par l'application (changement de taux de compression, correction à l'arrivée, tampon d'amortissement à l'arrivée …)**

Les services du protocole UDP

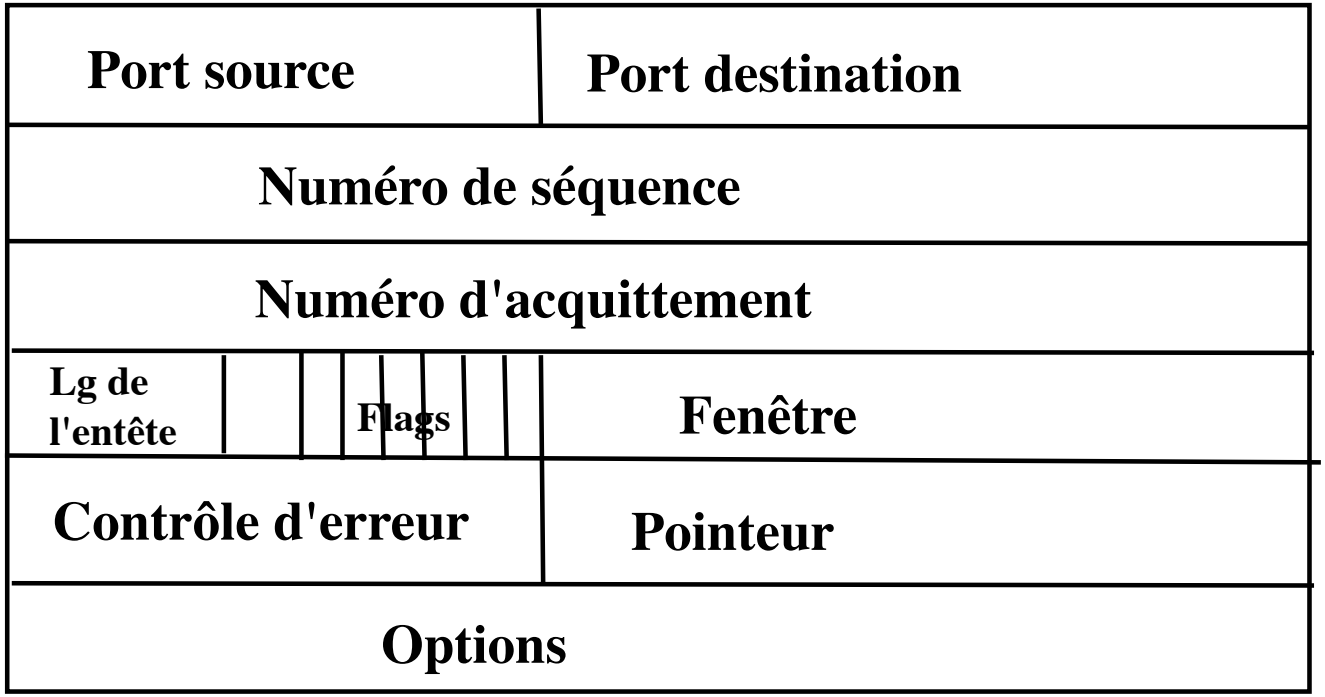
- Mode sans connexion (Datagram)
- Multiplexage applications -> transport
- Numéro de port sur 1 octets
- Longueur en nombre d'octets
- Détection d'erreur optionnelle (par "checksum")
- Aucun contrôle de flux et récupération d'erreur
- Entête UDP:



Services du protocole TCP

- **Segmentation re-assemblage des messages**
  - **Concaténation des paquets : flux d'octets**
- **Rétablir l'ordre des paquets**
  - **Numérotation des octets de données**
- **Multiplexage vers plusieurs applications (numéro de port)**
- **Service orienté connexion**
  - **Ouverture et libération de la connexion**
- **Transfert de données exprès**
- **Contrôle de flux**
- **Détection des paquets erronés et perdus**
- **Récupération des erreurs par réémission**
- **Détection d'inactivité**

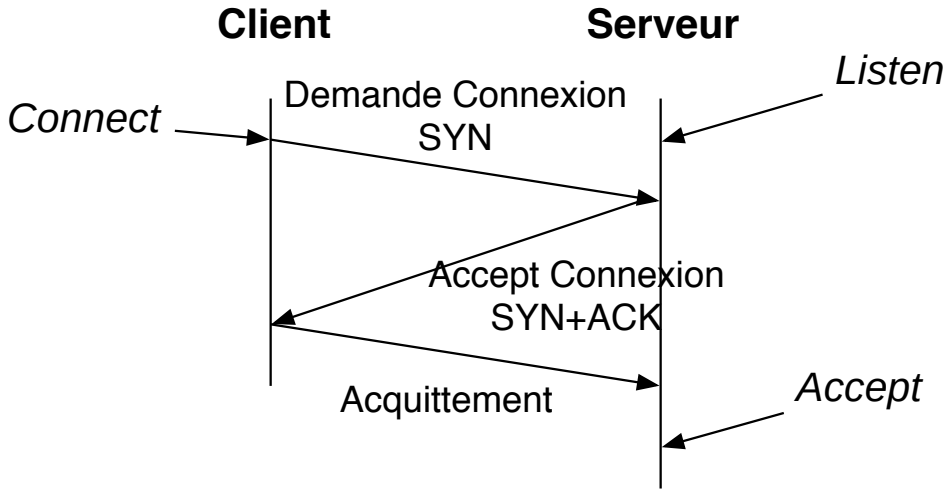
Le format du paquet TCP



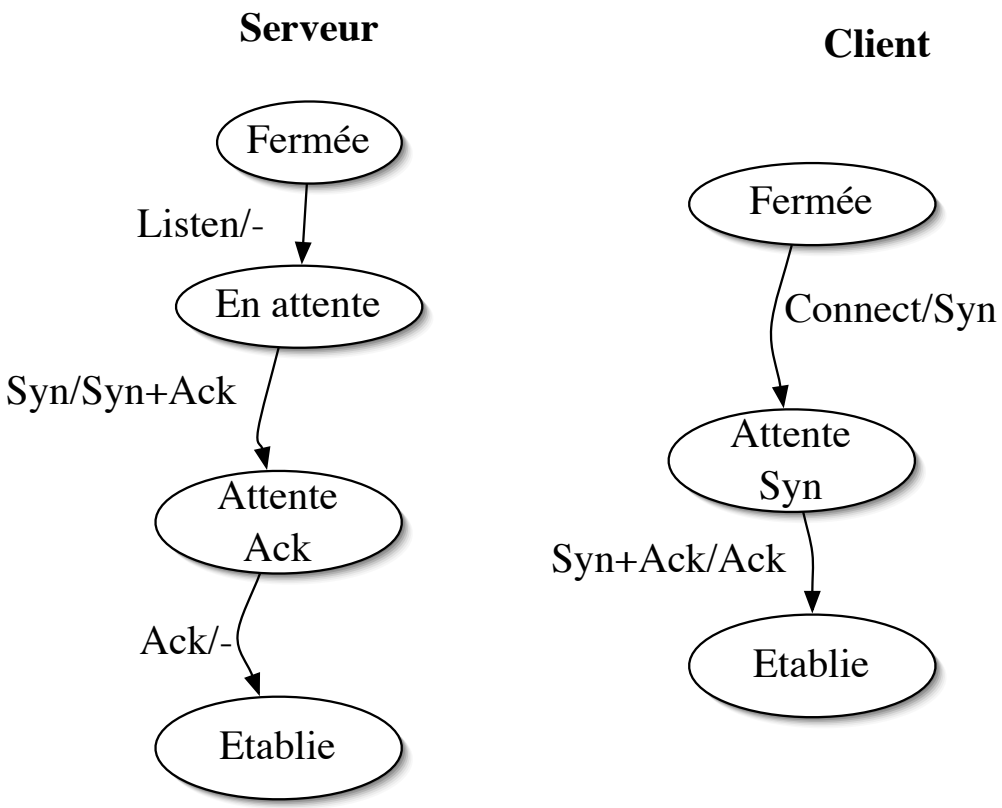
- Flags: Urgent, Ack, Psh, Rst, Syn, Fin

Etablissement d'une connexion TCP

- **Des paquets particuliers pour ouvrir la connexion**
  - **Demande de connexion: Flag SYN=1 et ACK =0**
  - **Acceptation de connexion : Flags SYN= 1 et ACK=1**
    - » **Des options sont possibles**
      - **Taille maximale des paquets**
      - **Convention pour le contrôle de flux sur réseau haut débit ...**
- **Ouverture**
  - **Le protocole est un protocole à trois phases:**



Automates de Mealy

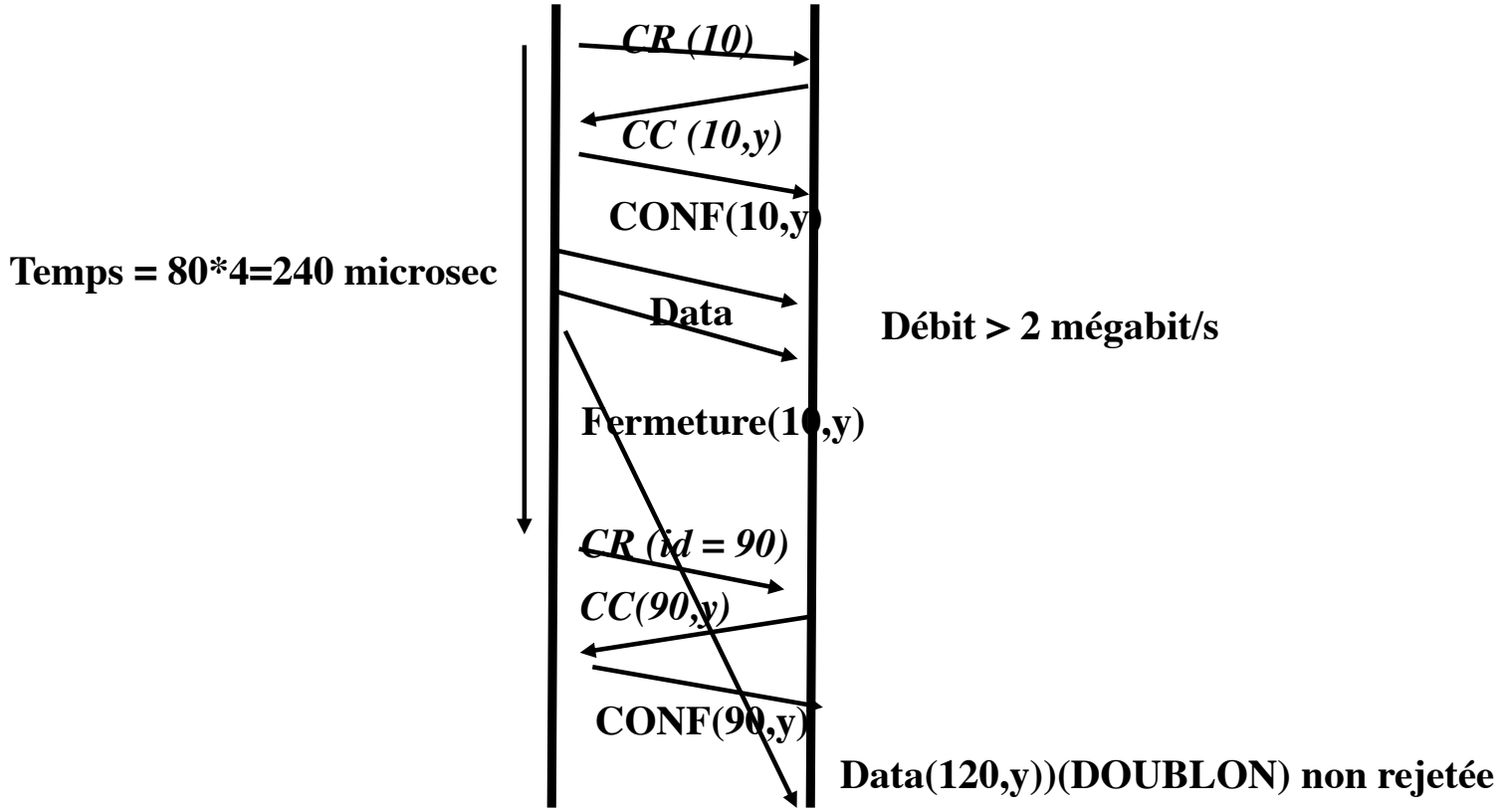


Etablissement d'une connexion(2)

- Trois paquets sont nécessaires pour garantir qu'il n'y ait pas d'ambiguité sur des demandes de connexions (doublon, timers de réémission)
- Le numéro de séquence initiaux permettent de différencier des demandes de connexions dupliquées
- Flag Reset en cas de duplication (ou autre incohérence)
- Les numéros de séquence initiaux sont calculés à partir d'une horloge système de période 4 microsecondes. Cela garantit l'unicité d'un paquet pendant 4 heures
- Problème: l'évolution des NoSeq des données dépend du débit de la connexion.
- On peut arriver à des cas d'ambiguité par rapport aux id initiaux (zone interdite)
- Solution : on attend la durée de vie maximale sur Internet avant de ré-ouvrir une connexion sur la même paire de ports (2 minutes)

Zone interdite

- **Débit pemettant d'arriver dans une zone interdite.**
  1 octet/4.10 $^{-6}$ sec= 8bits/4 10^{-6}= 2 Mégabit/s
- **Si le débit est supérieur à 2 mégabit/s , le No de séquence d'une donnée peut être valable pour une connexion ultérieure**

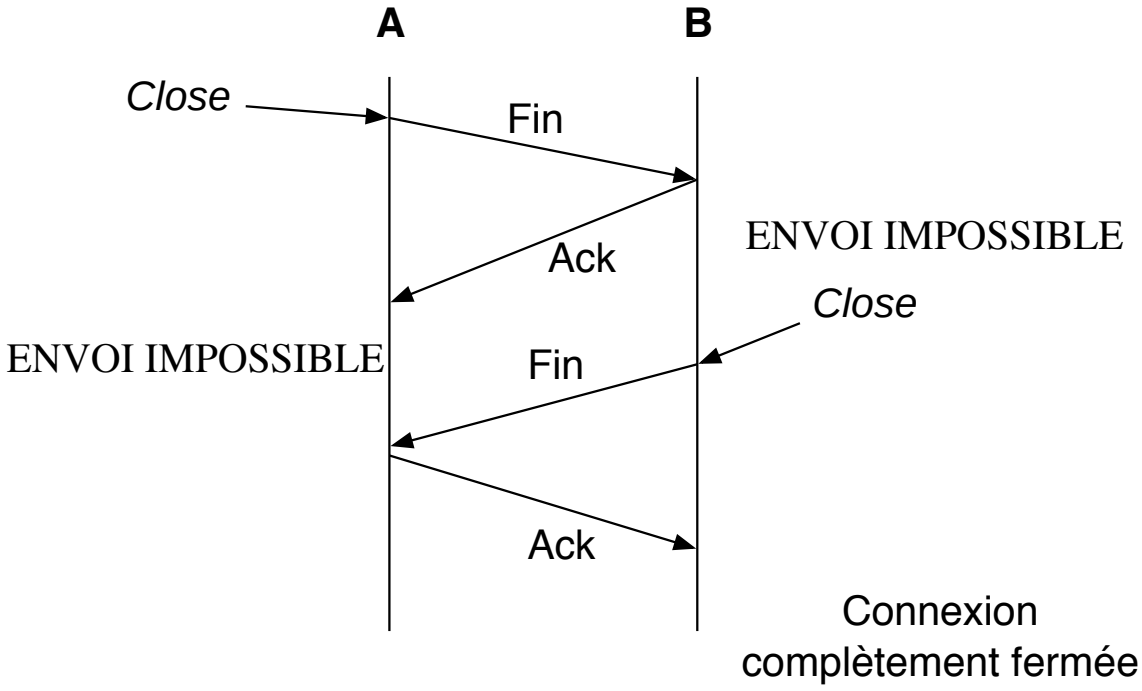


Fermeture de connexion

- **1ère solution:**
  - **Fermeture brutale, quand on veut fermer la connexion d'un coté , on ne se soucie pas de l'état de l'autre. Comme au téléphone, on raccroche.**
  - **Primitive *close* des Sockets**
- **2ème solution**
  - **Il faut éviter que la rupture de la connexion provoque des pertes de messages. Pour cela, il faut essayer que les deux entités de transport aient la même vision de la connexion**
  - **Solution: la connexion est gérée comme deux demi-connexions unidirectionnelles**
  - **On peut fermer en émission, et continuer à recevoir, si l'autre n'a pas fini d'émettre.**
  - **Primitive *shutdown* des sockets**

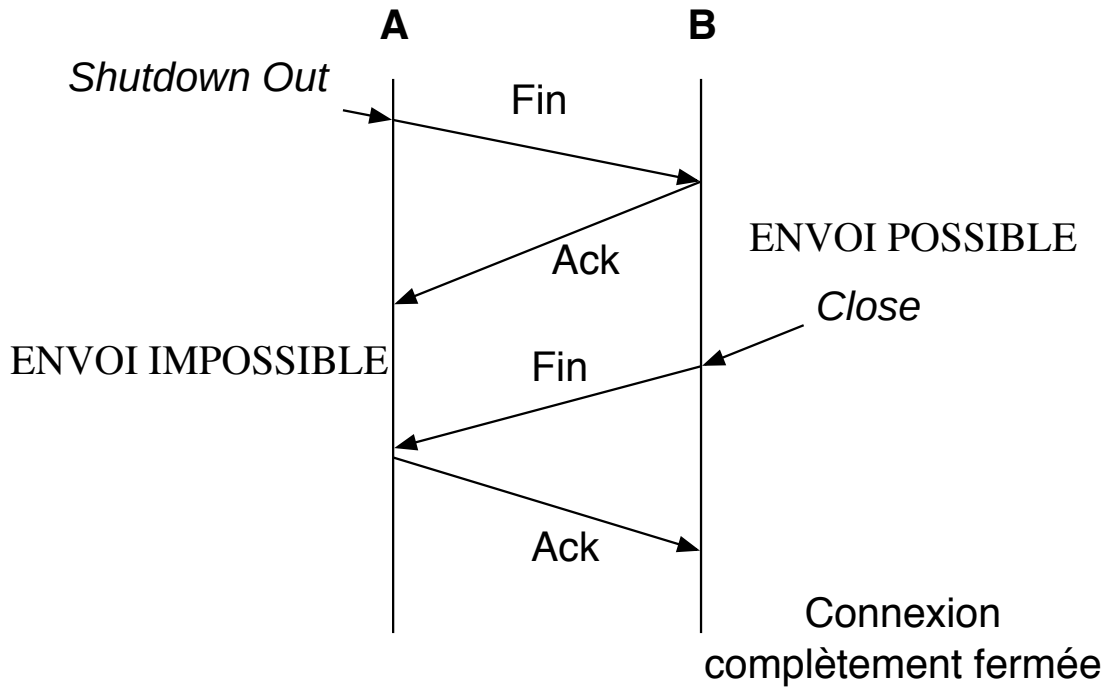
Fermeture de connexion
Le close

- **Flag FIN pour signifier la demande de fermeture**
- **L'entité distante acquitte pour qu'il n'y ai pas d'ambiguité sur l'état de la connexion**



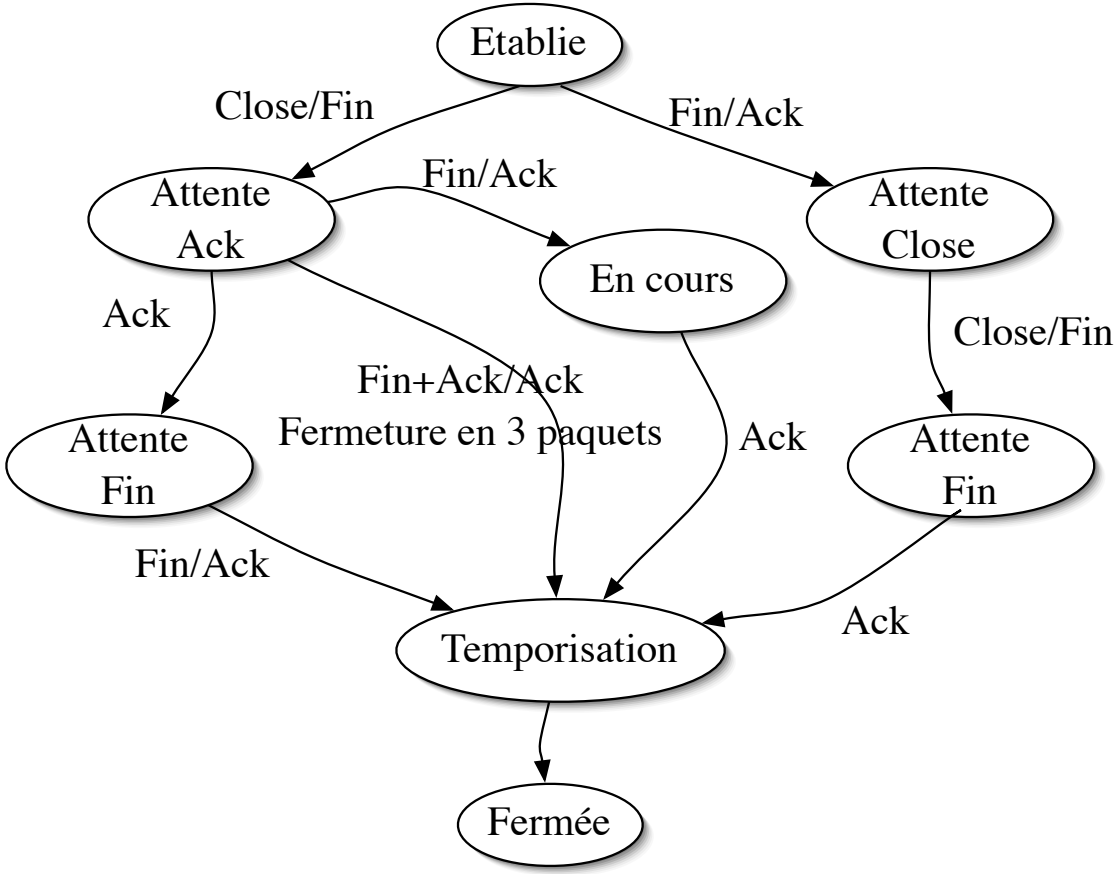
Fermeture de connexion
Le shutdown

- **Shutdown out, in, both**



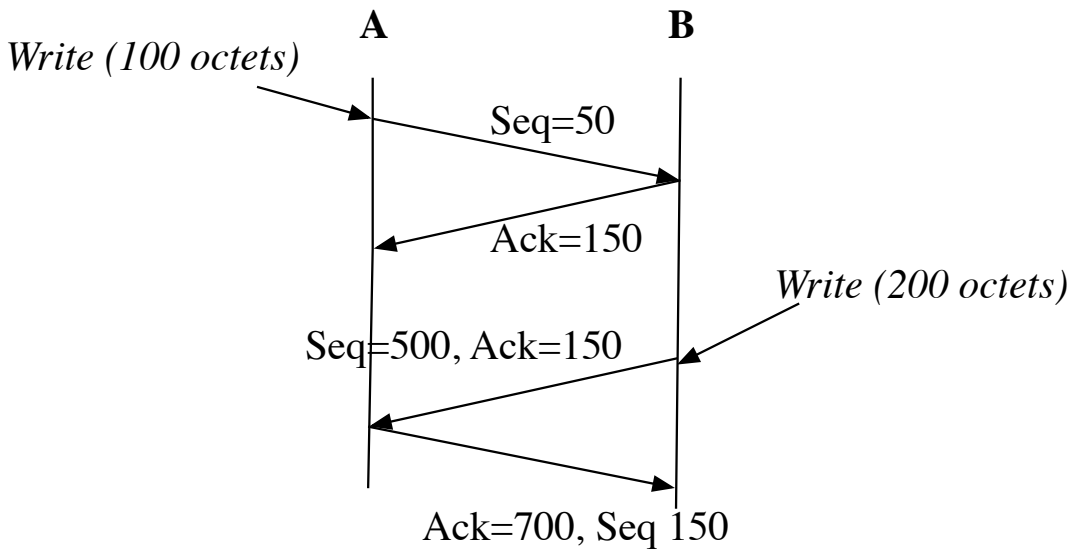
Automate de la fermeture

Serveur et client



Récupération d'erreur

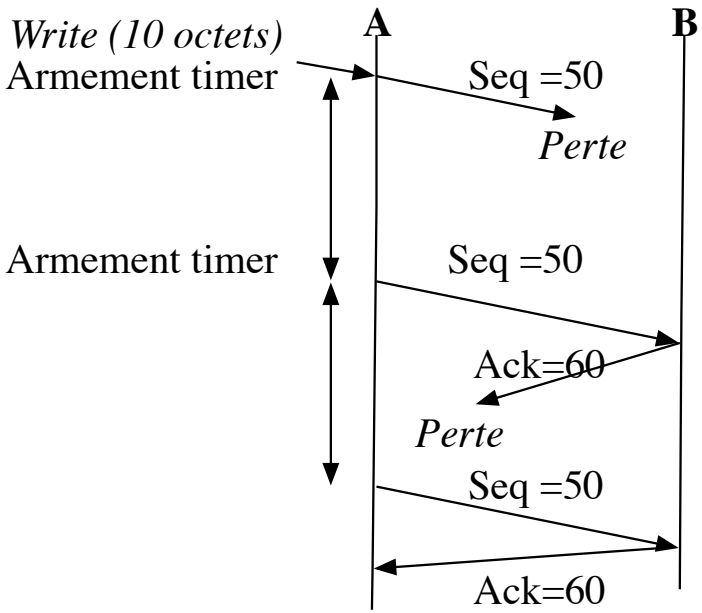
- **Flux d'octets (contrairement à UDP)**
- **Une fois la connexion ouverte, elle est symétrique et le transfert des données est bidirectionnel**
- **Numéro de séquence: Numéro du 1er octet de donnée du message (sert aussi à la segmentation et réassemblage)**
- **Numéro d'acquittement: numéro de séquence + 1 du dernier octet bien arrivé (dans l'autre sens) (flag Ack=1)**

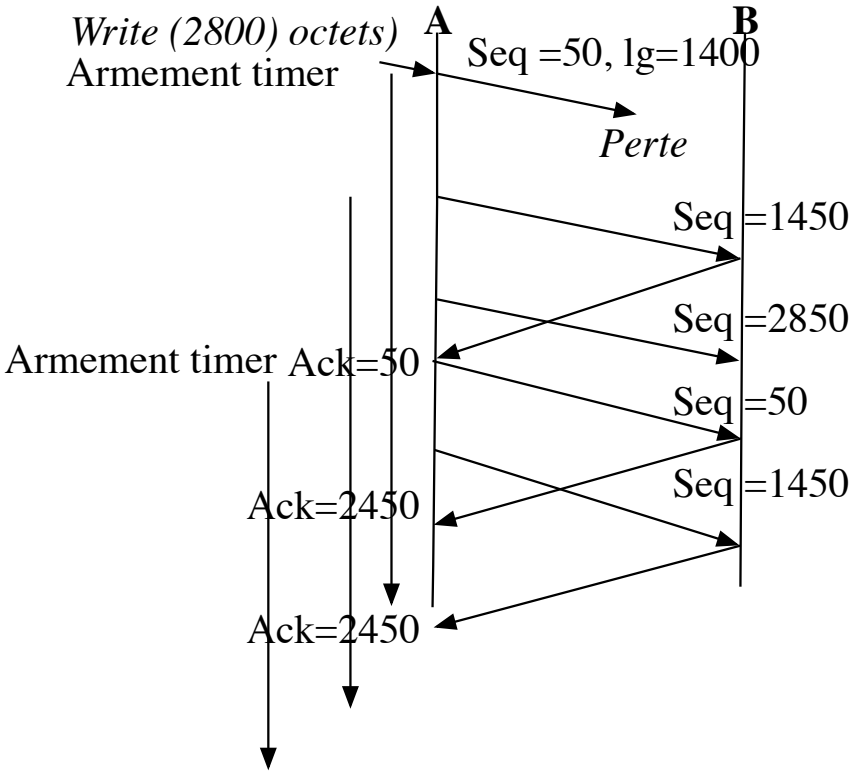


Mécanisme de la récupération d'erreur

- **Côté récepteur:**
  - » **Si Numéro de séquence attendu alors acquittement jusqu'au dernier reçu**
  - » **Sinon : mémorisation du paquet dans un tampon et acquittement jusqu'au dernier reçu sans «trou»**
- **Côté émetteur:**
  - » **Timer associé à un paquet et mémorisation dans un tampon**
  - » **Timer de re-émission**
    - » **Variable estimé en fonction du temps d'aller-retour des paquets de données et ACQ précédents**
    - » **Souvant timer sur-estimé (finesse de l'horloge système)**
    - » **Optimisation :**
      - » **si réception d'un Ack portant le même numéro**
      - » **perte de paquet**
      - » **réémission du paquets à partir du numéro d'Ack**

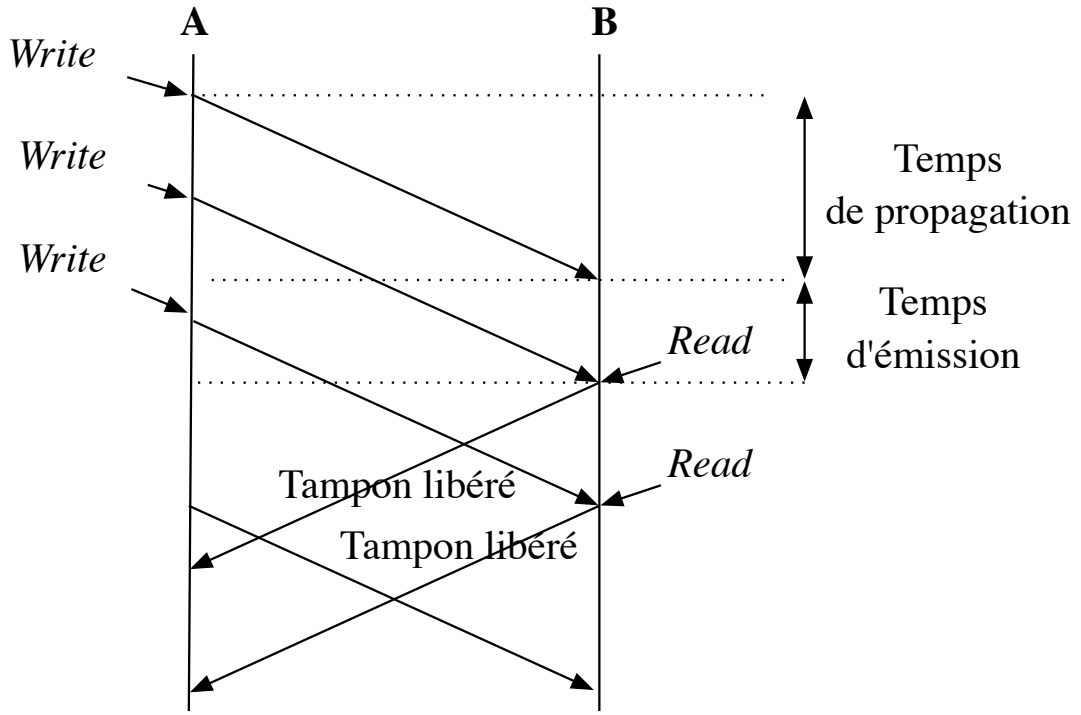
Exemples de récupération d'erreur





Contrôle de flux

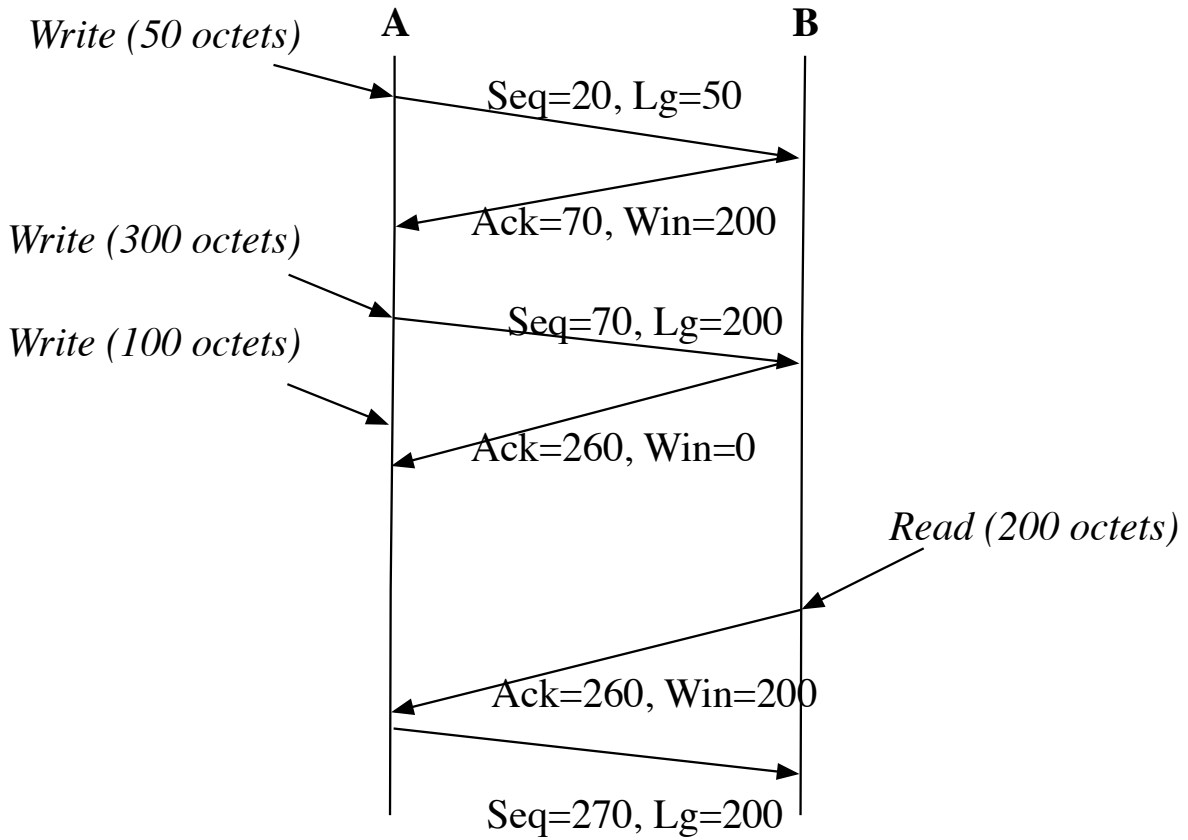
- **Problème: limiter le flux de l'émetteur pour ne pas saturer le récepteur**
- **Un tampon de taille fixe en réception: si le tampon est plein (l'application n'a pas encore récupéré les données) les prochaines données seront perdues**
- **Régler la taille du tampon pour que les données soient émis au débit maximal si le récepteur suit le rythme (dépend du temps de propagation et du temps d'émission)**



Mécanisme à fenêtre d'anticipation variable

- **Permet à l'émetteur d'envoyer des paquets tant que le tampon du recepteur n'est pas plein**
- **Champ Fenêtre (WIN): nombre d'octets pouvant être expédiés après le numéro d'acquittement**
- **Au départ WIN est inférieur ou égal à la taille du tampon de réception**
- **Quand WIN = 0 :**
  - **Le Buffer du récepteur est plein**
  - **L'émetteur est bloqué jusquà la libération du tampon (lecture de l'application côté récepteur)**

Exemple


A
B
Write (50 octets)
Seq=20, Lg=50
Ack=70, Win=200
Write (300 octets)
Write (100 octets)
Seq=70, Lg=200
Ack=260, Win=0
Read (200 octets)
Ack=260, Win=200
Seq=270, Lg=200

Cas de la fenêtre stupide

- **Libération du buffer d'un seul octet**
  - **Déblocage par envoi d'un paquet avec champ Fen= 1**
  - **Envoi d'un paquet de 1 seul octet de donnée**
  - **Charge du réseau inutile**
- **Solution:**
  - **Attendre une libération «importante» du buffer avant de débloquer**
  - **En pratique: Moitié du buffer ou taille maximale des segments**

Cas de blocage

- **Après un remplissage du tampon de réception si le paquet de "déblocage" se perd, on arrive à une situation de blocage**
- **Solution: l'émetteur continue à envoyer le dernier octet de donnée afin de forcer le récepteur à émettre un acquittement**