Couche Transport

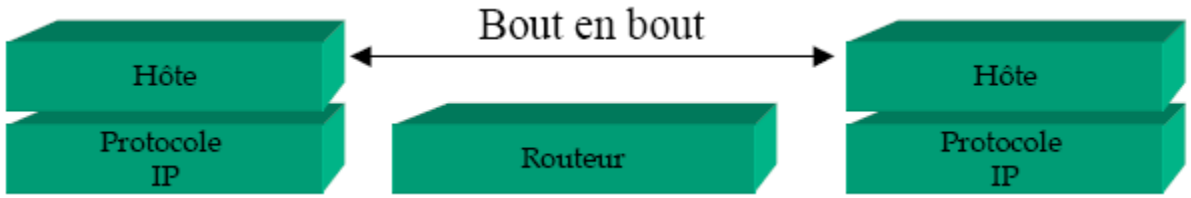
Prof. Rami Langar
LIGM/UPEM
Rami.Langar@u-pem.fr
http://perso.u-pem.fr/~langar

Plan

- Architecture TCP/IP
  - Les protocoles de transport
  - Fonctions
  - Multiplexage/démultiplexage
- UDP
- Mécanismes de transport fiable des données
  - Go-Back-N (GBN)
  - Répétition sélective
- TCP
  - Eléments de base de TCP
  - Contrôle de flux
  - Algorithmes de contrôle de congestion

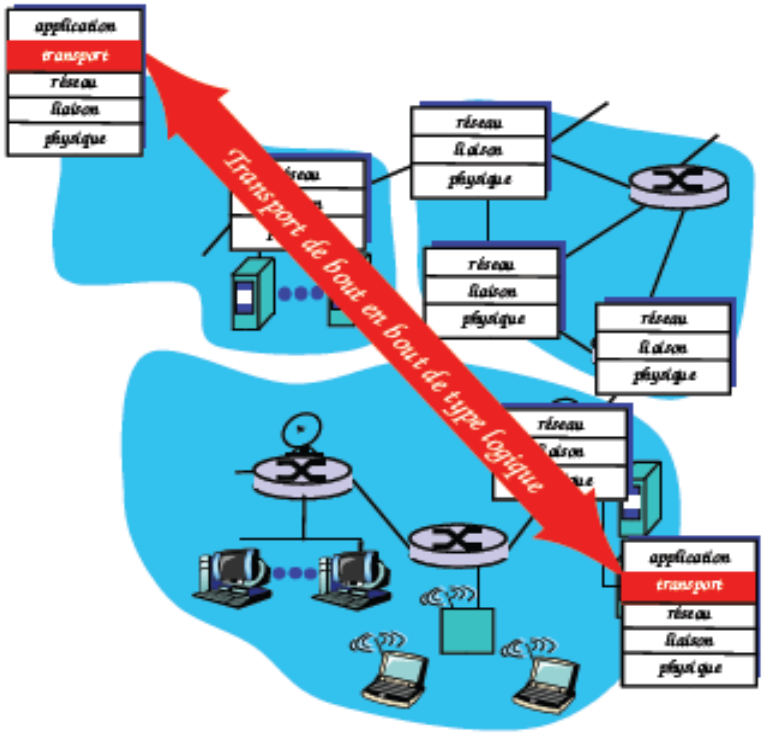
Principe de bout en bout

- Pas d'intelligence dans le réseau
  - Traitement simple dans le réseau => haut débit
  - Pas de redondance de contrôle
- Robustesse:
  - Séparation des fonctions : Le fonctionnement du système n'est pas lié à celui du réseau:
    - Contrôle : hôte
    - Acheminement : routeur



Fonctions des protocoles de transport

- Fournissent une communication logique entre des processus applicatifs s'exécutant sur des hôtes différents
- Logique = connexion directe pour l'application, même si ce n'est pas le cas physiquement (ex : telnet google.fr).
  - Couche transport : communication logique entre processus sur des serveurs différents
  - Couche réseau : communication logique entre différents serveurs



Fonctions des protocoles de transport

- Protocoles de bout en bout (pas comme à la couche réseau par exemple)
  - Service connecté ou non
  - Fiabilité
  - Performance
- S'appuie sur des protocoles de couches basses pour l'acheminement des données.
- Deux protocoles principaux:
  - UDP: non fiable, sans connexion
  - TCP: fiabilité, contrôle d'erreur, de flux, de congestion, d'ordre :
    - Convertit IP (service non fiable de la couche réseau) en un service fiable.

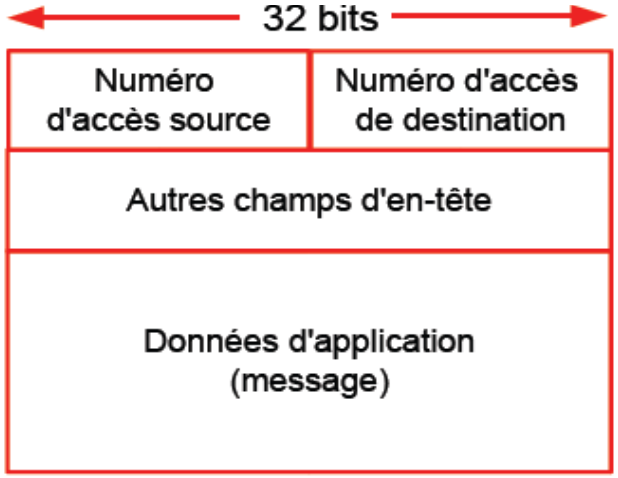
Multiplexage / démultiplexage

- But de la couche Transport :
  - Transmettre les données des segments en provenance de la couche réseau au processus d'application.
  - Utilisation des ports (socket) pour identifier une cible plus spécifique que l'adresse IP car les datagrammes sont destinés à des process et pas au système.
- Problème du démultiplexage :

  Sur un même terminal on peut avoir en parallèle : 2 sessions telnet, 2 sessions ftp et 1 processus http
  - Comment la couche transport de votre ordinateur transmet l'information au bon processus ?
  - Votre processus dispose d'une interface de connexion (socket): identifiant unique qui dépend du protocole
  - La couche transport transmet l'information à travers cette interface de connexion

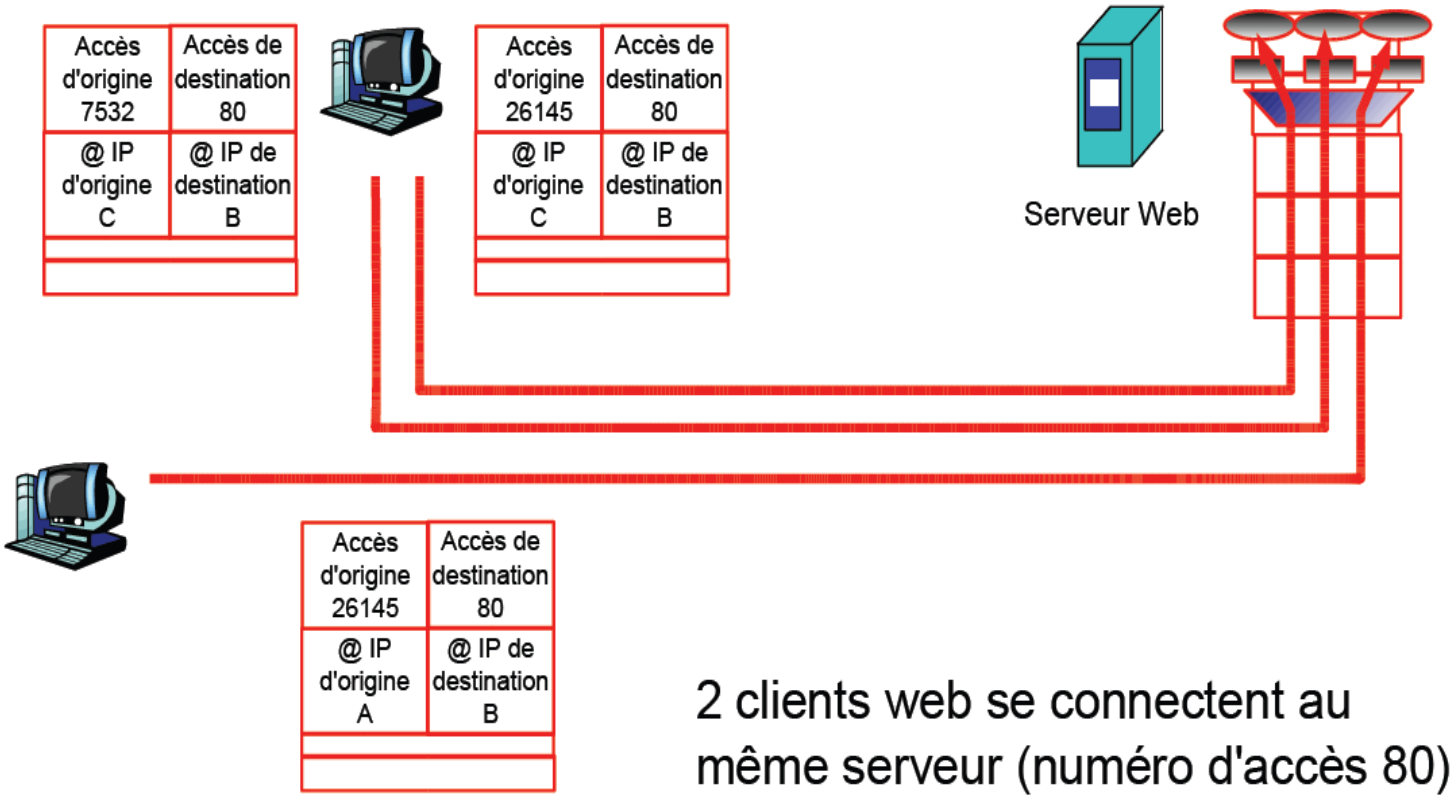
Multiplexage / démultiplexage

- Multiplexage : Rassemble les données en provenance de différentes interfaces pour créer des segments.
  - Chaque segment porteur de champs spéciaux pour l'interface de source et destination.
  - Numéro de port (numéro d'accès) d'origine (16 bits)
  - Numéro de port (numéro d'accès) de destination (16bits)
    - Nombre compris entre 0 et 65535 :
    - 0-1023 : ports connus et d'usage limité (http: 80, telnet: 23, ftp: 21)
    - RFC1700 : liste les numéro d'accès connus (mise a jour régulière sur http://www.iana.org



Format d'un segment TCP/UDP

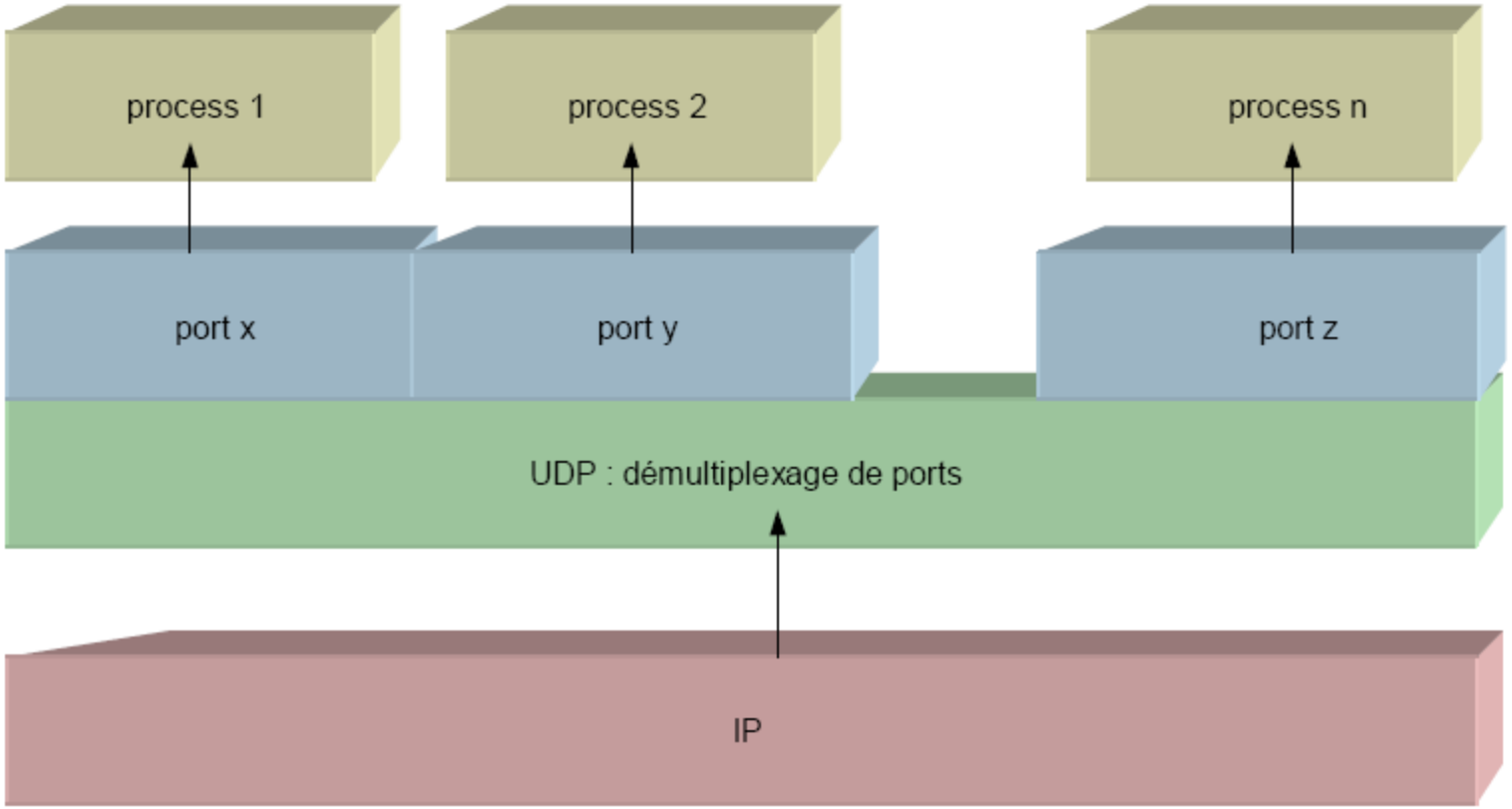
Multiplexage/démultiplexage : exemple



Transport sans connexion : UDP

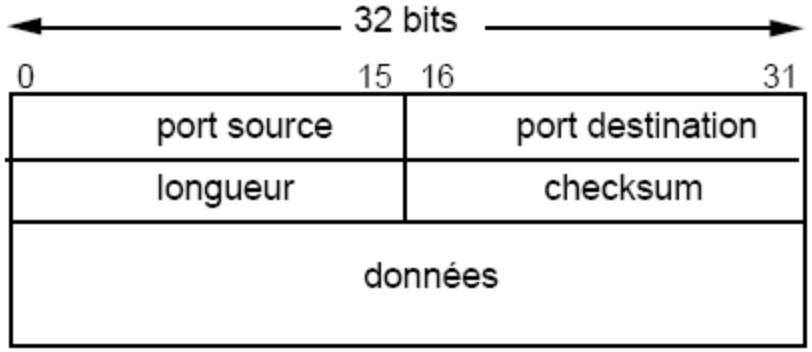
- User Datagram Protocol (RFC 768)
  - Numéro de protocole 17 quand transporté par IP
  - Pas de garantie sur l'ordre d'arrivée, segment perdu,
  - Chaque segment est indépendant des autres
- Mais alors pourquoi utilise-t-on UDP ?
  - Pas de surcoût lié à une connexion
  - Petit segment d'en-tête (TCP : 20 octets, UDP : 8 octets)
  - Orienté non connexion (réduire le délai)
  - Pas de contrôle de congestion (UDP peut envoyer des paquets très rapidement sur le réseau) : création d'un paquet et transmission immédiate au niveau IP
  - Qualité « Best Effort »

Communication entre les couches



Datagrammes UDP

- Port source
  - Indique le numéro de port du processus émetteur
  - Toute réponse y est dirigée
- Port destinataire
- Longueur:
  - Nombre d'octets dans le datagramme entier (avec en-tête)
  - > 8



Checksum d'UDP

- But : détecter d'éventuelles erreurs.
- Expéditeur :
  - Contenu découpé en morceaux de 16 bits
  - Basé sur complément à 1 de la somme
  - Le résultat est placé dans le champs « somme de contrôle »
- Destinataire :
  - Calcul le complément à 1 de la somme du segment reçu
  - Compare le résultat avec celui stocké dans le message

Checksum d'UDP

- Expéditeur

```
  0110011001100110
+ 0101010101010101
+ 0000111100001111
= 1100101011001010
  0011010100110101 (complément à 1)
```

Destinataire

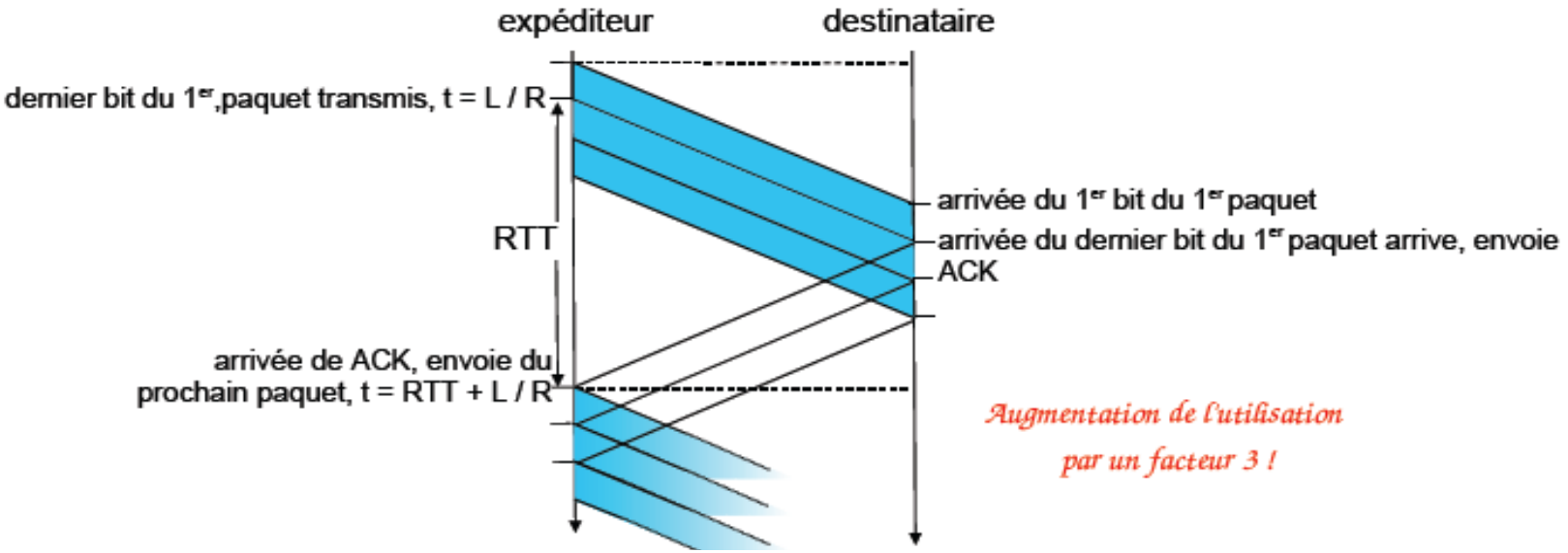
```
  0110011001100110
+ 0101010101010101
+ 0000111100001111
= 1100101011001010
+ 0011010100110101
= 1111111111111111 (<=Ok)

= 1111111101111111 (<=Erreur)
```

Applications d'UDP

- La signalisation de certains protocoles
  - Trivial File Transfer Protocol (TFTP)
  - Domain Name System (DNS) name server
  - Simple Network Management Protocol (SNMP)
- Applications temps réel, Visio et audioconférences

Protocoles de transfert pipelinés

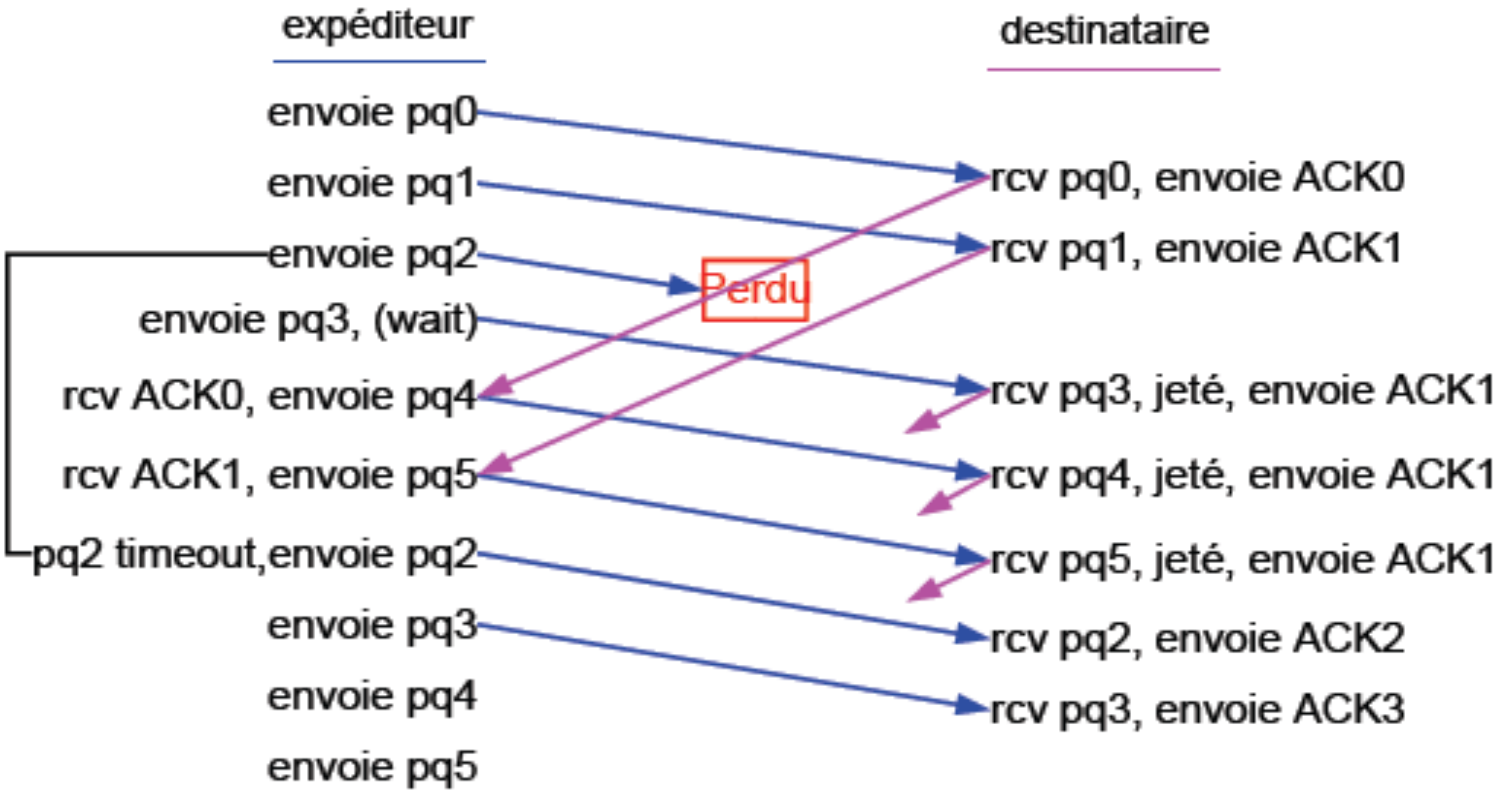
- Le pipeline :
  - Au lieu d'attendre inutilement, l'expéditeur envoie les prochains paquets
  - Mise en place de numéro de séquence plus grand
  - Mémoire tampon plus grande



Correction d’erreur en mode pipeliné

- Go-Back-N ou protocole à fenêtre glissante
  - L'expéditeur peut retransmettre plusieurs paquets à la suite dans la limite de N paquets
- Utilisation d'accusé de réception cumulatif, d'un compte à rebours, et d'un système à gestion de fenêtre (vide/pleine)
- Tampon très simple pour le destinataire:
  - Il jette lorsqu’il reçoit un paquet non attendu.

GBN : fonctionnement

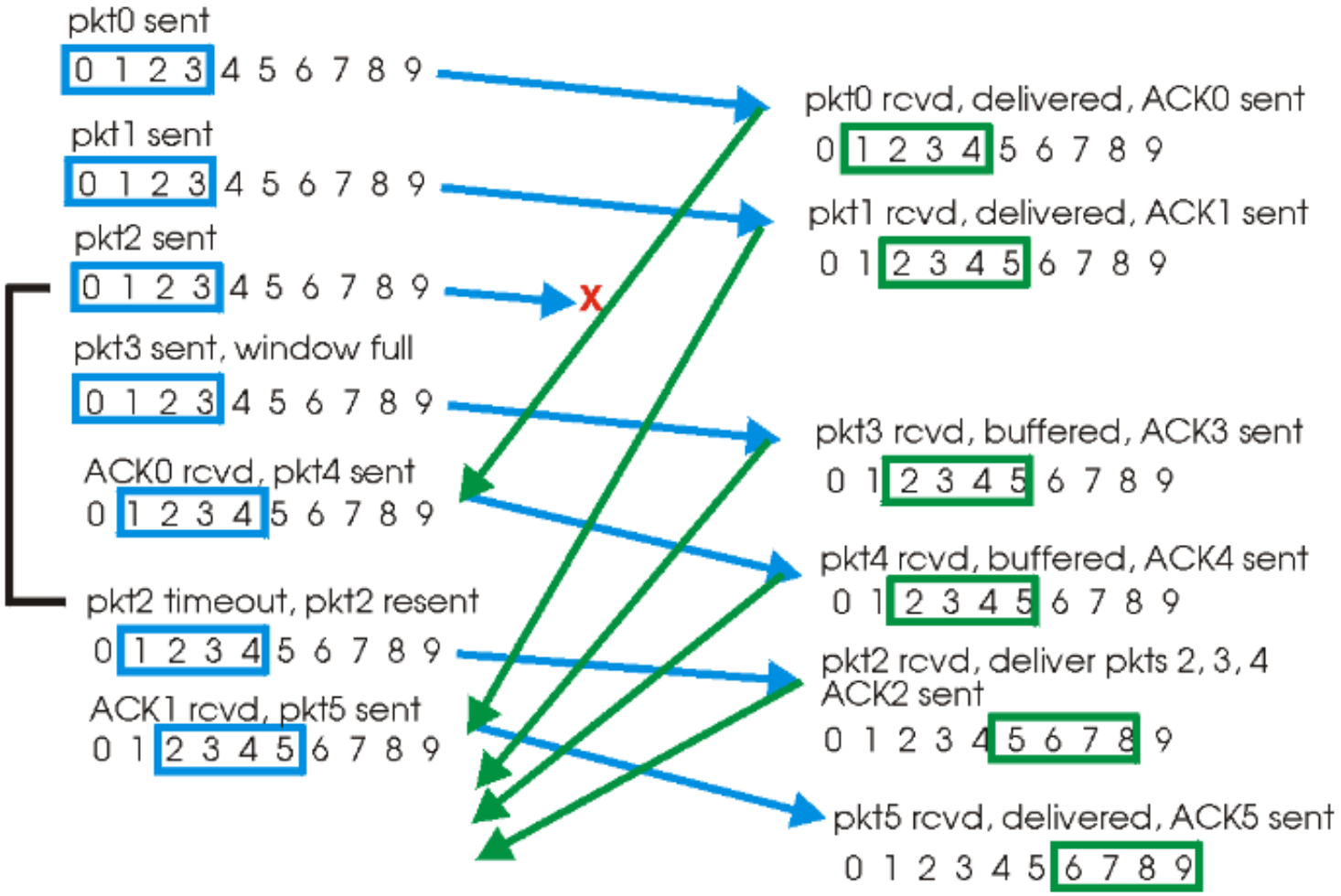


Problème : si trop grande fenêtre et/ou débit trop élevé

Correction d'erreur en mode pipeliné

- Répétition sélective
  - L'expéditeur ne retransmet que les paquets perdus ou corrompus
- Destinataire :
  - Transmet un ACK pour tout paquet reçu
  - Mise en place d'un tampon de taille N
- Expéditeur :
  - Ne retransmet que les paquets pour lesquels ACK non reçu
  - Mise en place d'un tampon de taille N

Répétition sélective : fonctionnement

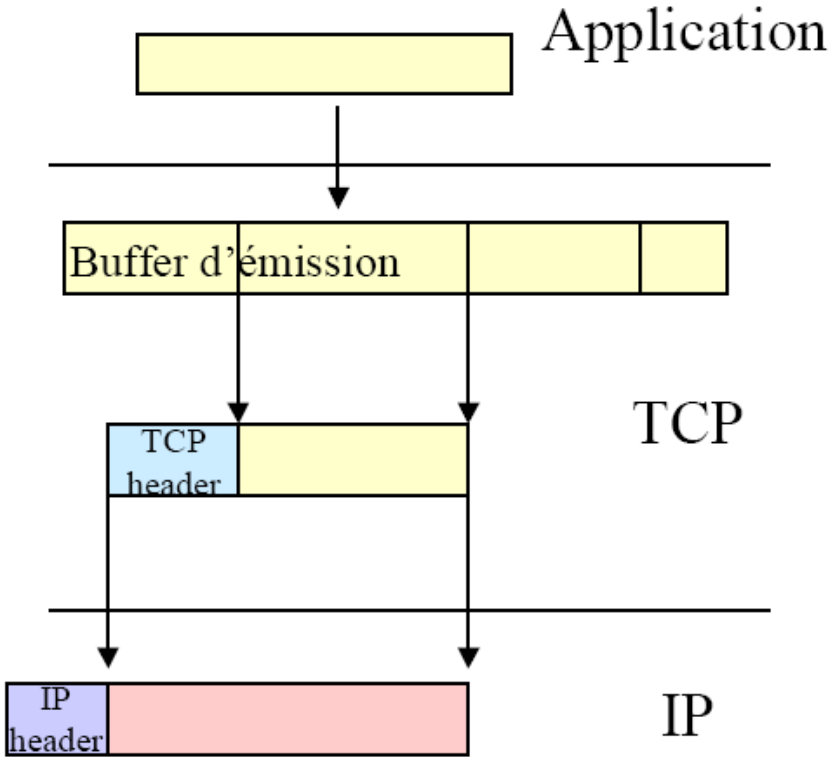


TCP

- Transmission Control Protocol (RFC 793)
- 90% du trafic
- Buts :
  - Bonne utilisation des ressources du réseau
  - Transfert fiable de données (≠ UDP)
    - Contrôle de perte
    - Contrôle de flux
    - Contrôle de congestion: interaction hôte-réseau réactive et agréssive
    - TCP suppose que les couches de communication qui lui sont inférieures lui procurent un service de transmission de paquet simple, dont la qualité n'est pas garantie.
  - Mode connecté et commutation de paquets
    - La connexion est établie et considérée comme un circuit physique
    - Fiabilité
- Utilisé par TELNET, FTP, HTTP …

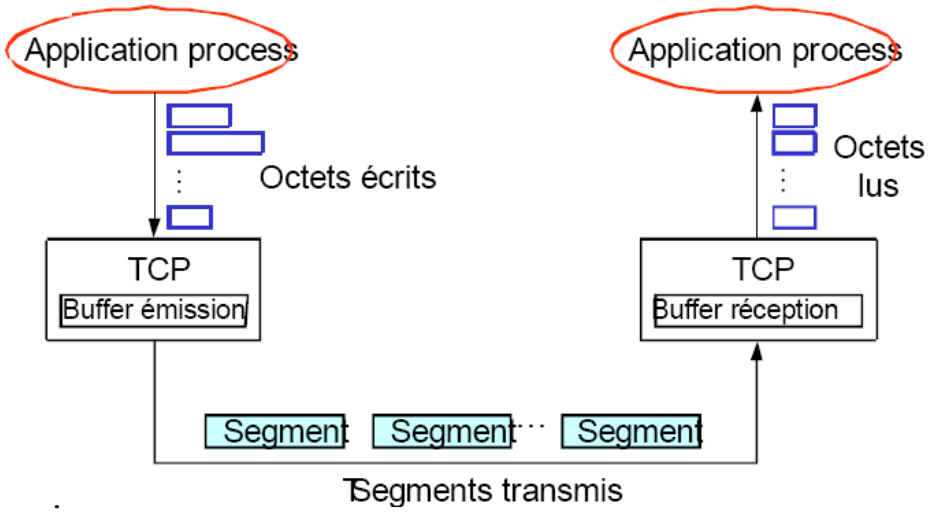
Segments TCP

- Les octets de données sont accumulés jusqu'au moment où TCP décide d'envoyer un segment
  - Découpage en segment indépendant du découpage au niveau application
- MSS = longueur maximale d'un segment
  - Dépend du type de TCP (déterminé par le système d'exploitation du terminal)
  - Valeurs les plus courantes: 1500 octets, 536 octets, 512 octets.



Caractéristiques de TCP

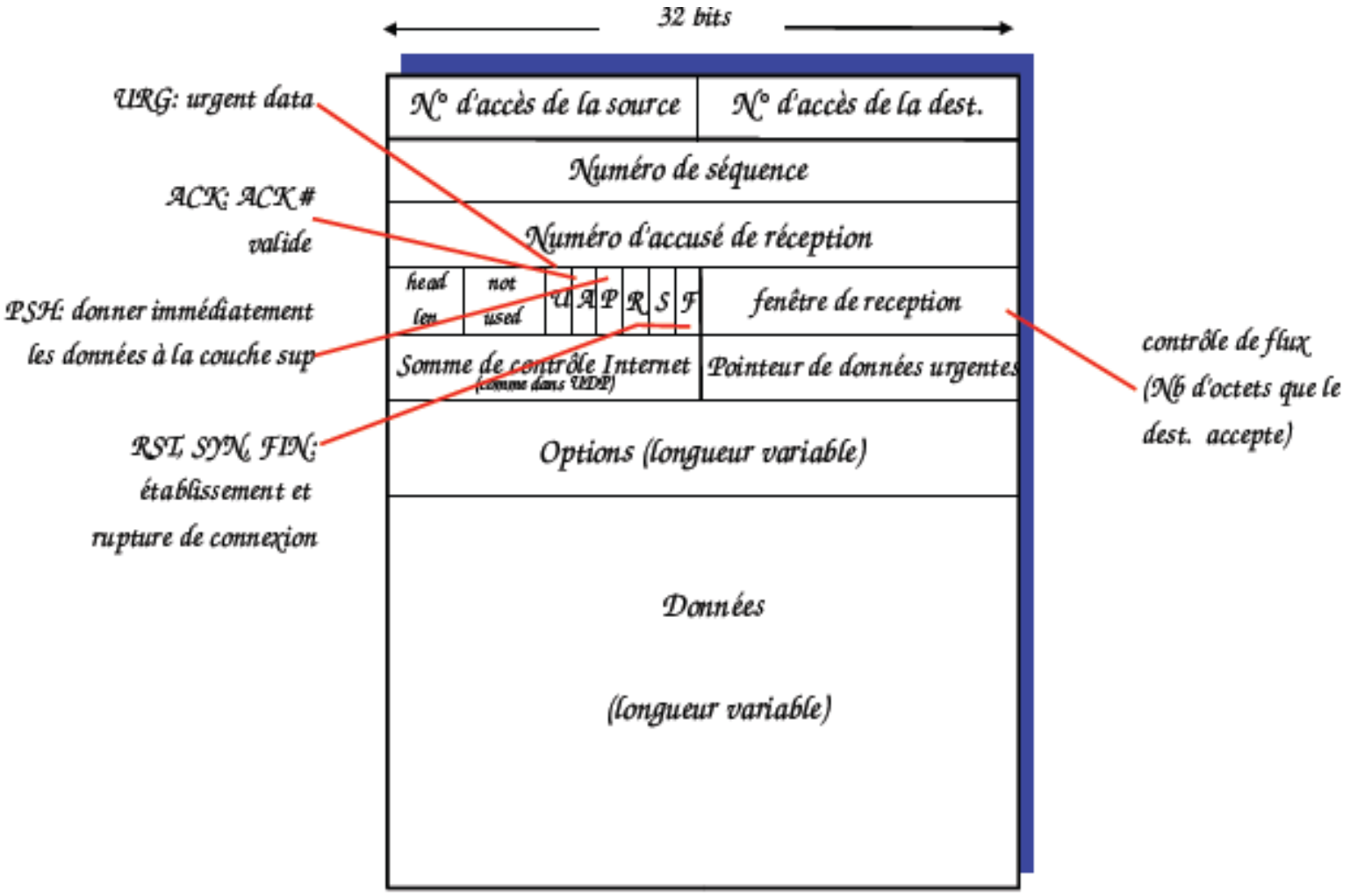
- Orienté flot d'octets :
  - Application écrit des octets
  - TCP émet des segments séquencés
  - Segmentation des données au niveau TCP
- Transfert bufférisé
  - Attendre d'avoir assez de données pour émettre (performance)
- Full duplex
  - Pour les ACKs
  - Mode pipeliné
- Numéros de séquence (de 32 bits) indépendants dans les 2 directions.
  - Associés à chaque octets
    - Indique le numéro du premier octet transmis
    - Si N octets sont délivrés du paquet avec le numéro de séquence X, l'acquittement aura la valeur X+N (soit le numéro du prochain octet à recevoir)
  - Piggybacking: les deux numéros sont présents dans les mêmes paquets



Caractéristiques de TCP

- Fiabilité
  - Numéro de séquence
  - Détection des pertes :
    - Acquittements positifs (ACK) du récepteur -> OK
    - Pas d'ACK -> timeout (temporisation) -> retransmission
    - ACK dupliqué
  - Ré-ordonnancement des paquets au récepteur
  - Elimination des paquets dupliqués
  - Checksum
  - Retransmissions :
    - Go back N (ACK cumulatif).
    - Répétition sélective (retransmission du segment N perdu).

Structure d'un segment TCP



Structure d'un segment TCP

- **Port source** (16 bits) : le numéro de port de la source.
- **Port Destination** (16 bits) : Le numéro de port du destinataire.
- **Numéro de séquence** (32 bits) : Le numéro du premier octet de données par rapport au début de la transmission (sauf si SYN est marqué). Si SYN est marqué, le numéro de séquence est le numéro de séquence initial (ISN) et le premier octet à pour numéro ISN+1.
- **Accusé de réception** (32 bits) : si ACK est marqué ce champ contient le numéro de séquence du prochain octet que le récepteur s'attend à recevoir. Une fois la connexion établie, ce champ est toujours renseigné.

Structure d'un segment TCP

- **Header length** (4 bits): La taille de l'en-tête TCP en nombre de mots de 32 bits. Il indique là ou commence les données. Généralement de 20 octets.
- **Réservé** (6 bits) : réservés pour usage futur. Doivent nécessairement être à 0.
- **Bits de contrôle** (6 bits - de gauche à droite):
  - URG: Pointeur de données urgentes significatif
  - ACK: Accusé de réception significatif
  - PSH: Fonction Push
  - RST: Réinitialisation de la connexion
  - SYN: Synchronisation des numéros de séquence
  - FIN: Fin de transmission
- **Fenêtre** (16 bit): le nombre d'octets à partir de la position marquée dans l'accusé de réception que le récepteur est capable de recevoir. Sert pour le contrôle de flux.

Connexion TCP

- Trois phases :
  - Etablissement de la connexion
  - Transfert des informations avec
    - Contrôle de flux
    - Contrôle de congestion
  - Libération de la connexion

Etablissement de la connexion TCP

- Procédure à trois échanges
  - Three way handshake
    - TCP client envoie un segment spécial (segment SYN), sans donnée de la couche application: bit SYN =1, N° de séquence initial (Client_ISN)
    - Réception du datagramme par le serveur.

      Allocation des tampons et variables TCP.

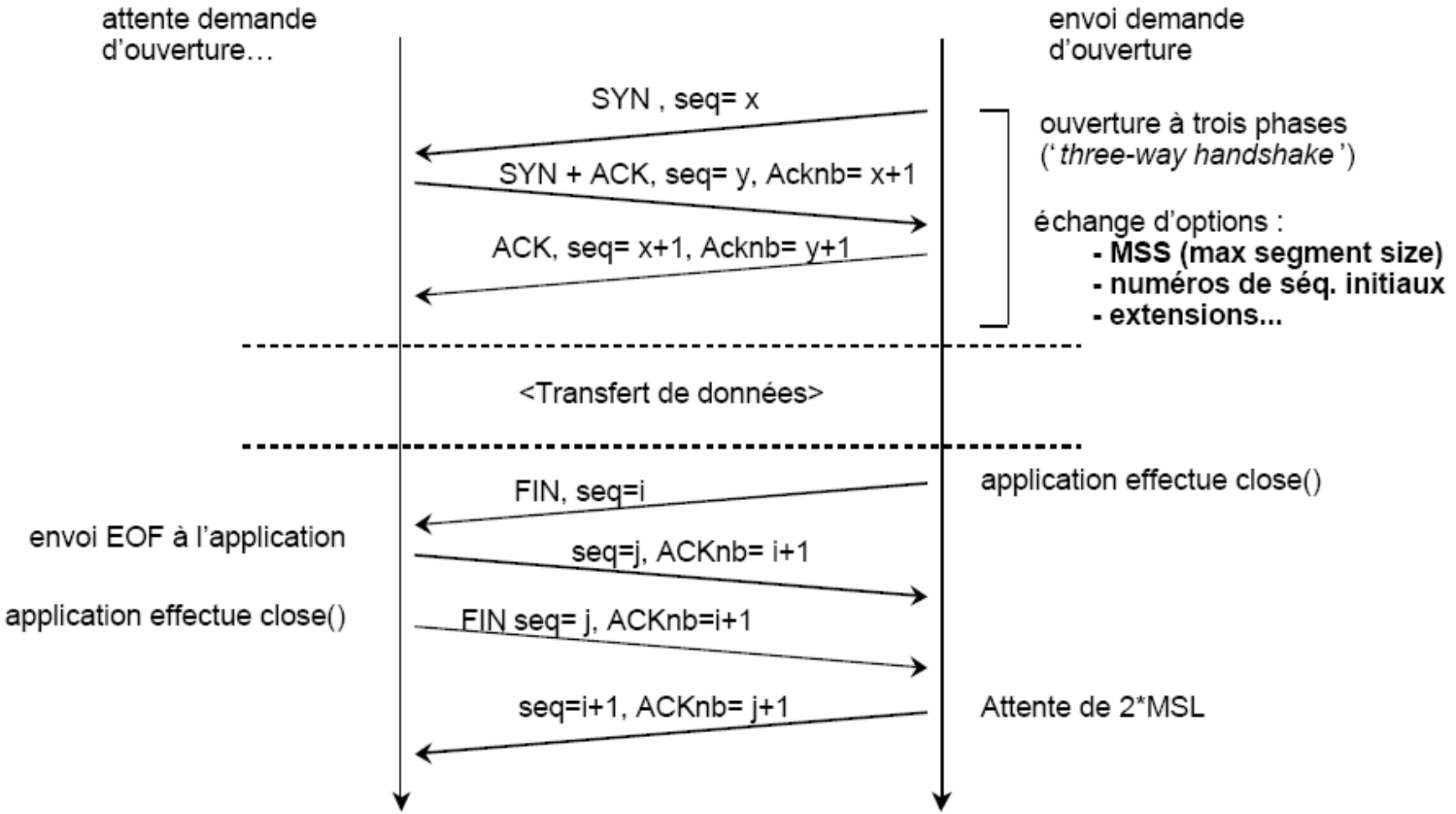
      Définition d'un N° de séquence initial (Server_ISN)

      Envoi d'un segment d'autorisation de connexion(seg. SYNACK) (SYN=1, A =1, Client_ISN+1, Server_ISN)
    - Réception du datagramme par le client.

      Allocation des tampons et variables TCP.

      Envoi d'un segment accusant réception de cette autorisation.
- ISN (Initial Sequence Number)
  - Numéro de séquence du premier octet d'information transporté
  - Le premier octet transporté est alors ISN+1
  - ISN est généralement dérivé de l'horloge de l'hôte

Exemple

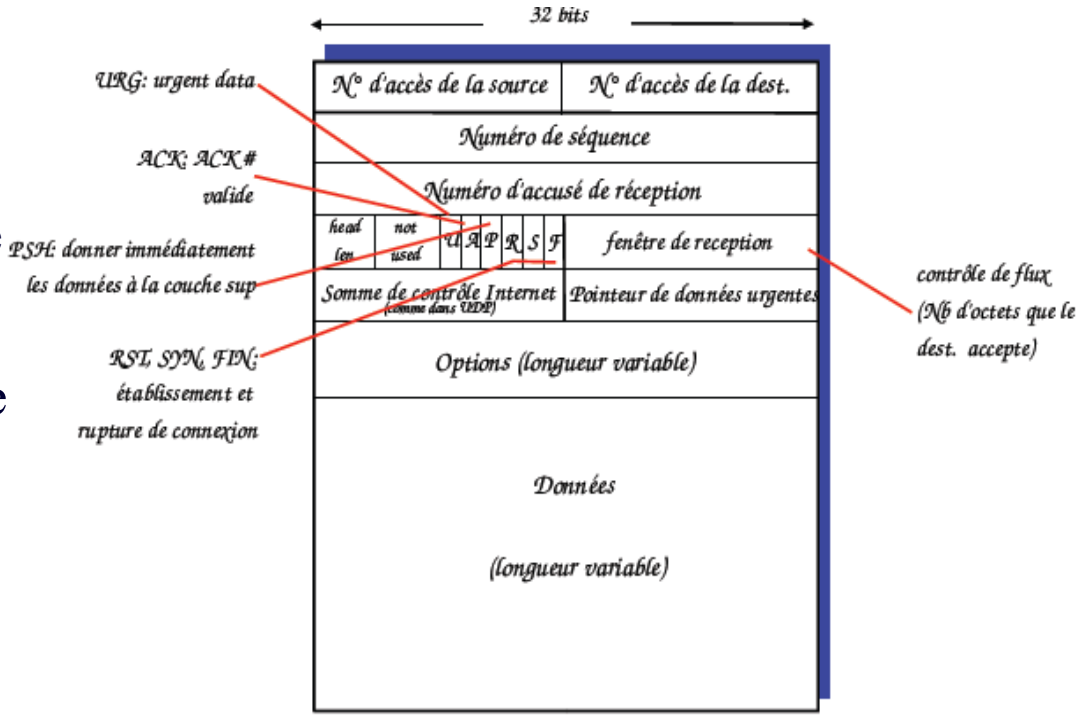

attente demande d'ouverture…
envoi demande d'ouverture
SYN , seq= x
SYN + ACK, seq= y, Acknb= x+1
ACK, seq= x+1, Acknb= y+1
ouverture à trois phases
(' three-way handshake ')
échange d'options :
- MSS (max segment size)
- numéros de séq. initiaux
- extensions...
<Transfert de données>
FIN, seq=i
application effectue close()
envoi EOF à l'application
seq=j, ACKnb= i+1
application effectue close()
FIN seq= j, ACKnb=i+1
seq=i+1, ACKnb= j+1
Attente de 2*MSL

Libération de la connexion

- Normale
  - Fin demandée par l'une des extrémités (flag FIN=1)
- Terminaison brutale
  - Envoi de la primitive ABORT (flag RST=1)
  - Toutes les transmissions ou réceptions sont interrompues et les tampons sont vidés

Contrôle de flux TCP

- Utilisation d'un tampon aux 2 bouts de la connexion TCP
  - Le processus application communique avec ce tampon.
  - Saturation du tampon si l'application est occupée à autre chose.
- Mise en place d'un contrôle de flux par l'utilisation d'une fenêtre de réception
  - Fenêtre de réception réglée sur les envois de l'expéditeur.
  - Le destinataire informe l'expéditeur de l'espace disponible dans les segments qu'il lui envoie.



Contrôle de flux TCP

- LastByteRead: numéro du dernier octet dans le flux de données retiré du tampon par le processus d'application destinataire.
- LastByteRcvd: numéro du dernier octet dans le flux de données reçu par le destinataire et placé dans sa mémoire tampon.
- Fenêtre de réception (RcvWindow) est adaptée à l'espace disponible dans le tampon:
  - RcvWindow = RcvBuffer – (LastByteRcvd – LastByteRead).

Fenêtre d'émission

- Emetteur:
  - LastByteSent – LastByteAcked ≤ RcvWindow

- LastByteSent – LastByteAcked = min (Cwnd, W)
  - Cwnd est une variable maintenue par la source
    - Tient compte de la congestion du réseau
  - W = RecvWindow: fixé par la destination, champ fenêtre annoncé
    - Tient compte de la capacité du récepteur

Contrôle de congestion de TCP

- Contrôle de congestion de bout en bout
- Ce sont les expéditeurs qui régulent leur vitesse
  - Comment assurent-ils cette régulation ?
  - Comment détecter les phénomènes de congestion ?
  - Quel algorithme utiliser pour réguler le taux d'envoi ?
- Nous avons vu que chaque pôle TCP dispose de :
  - Tampon de réception
  - Tampon d'envoi
  - Différentes variables
- + fenêtre de congestion ***Congwin***

Contrôle de congestion de TCP

- Principe
  - Trouver le point d'équilibre: "additive increase"
    - Augmenter la fenêtre de contrôle de congestion
  - Détection de la congestion par l'indication de la perte d'un paquet
  - Suppression de congestion en réduisant la fenêtre
- Algorithme
  - phase 1: slow start
  - phase 2: congestion avoidance
  - En cas de perte: réduction de la taille de la fenêtre et récupération (possibilité de Fast retransmit)

Phase 1: Slow Start

- Fenêtre glissante et taille variable de la fenêtre
  - Croissance exponentielle de la taille de la fenêtre (x2 à chaque fois que les paquets sont transmis correctement)
    - Augmentation de 1 segment à chaque acquittement.
  - Chaque RTT, doubler la taille de la fenêtre Cwnd.

Phase 2: Congestion Avoidance

- Pour stopper l'augmentation trop rapide (le slow start est exponentiel)
  - À partir d'un seuil (ssthresh): augmentation de 1 segment à chaque RTT (et non pas à chaque ACK!).
    - Augmentation linéaire de la taille Cwnd.
  - Mise à jour du seuil et il vaut la moitié de la fenêtre courante lors de la dernière congestion.

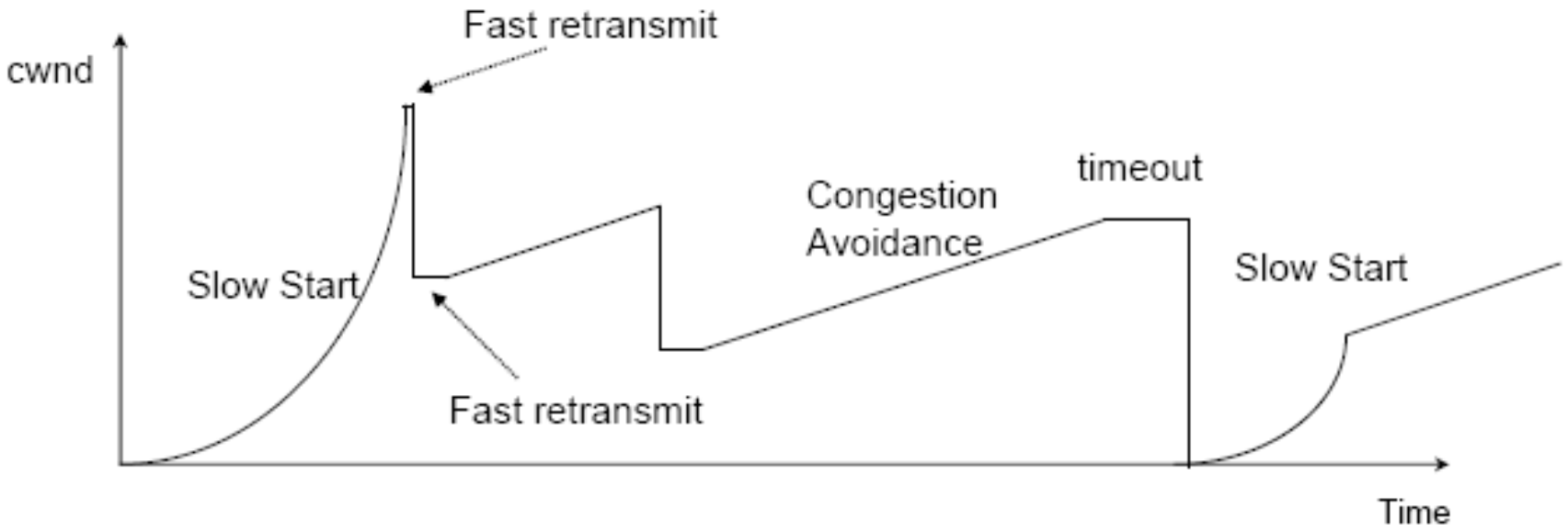
Algorithme de contrôle de congestion

- Initialisation avec Cwnd=1 et ssthresh=65536
- Nouvel ACK => grandir Cwnd selon:
  - Si Cwnd <= ssthresh alors slow start (phase 1) jusqu'à ssthresh.
  - Sinon Congestion avoidance (phase 2).
- Congestion => ssthresh = max (Cwnd avant congestion /2, 2).
- Si timeout (c.a.d perte), alors Cwnd=1
  - Slow start + Congestion avoidance.

Fast retransmit

- Si réception de 3 ACK pour le même segment (3 DUPACKs)
  - Paquet supposé perdu et donc retransmission
  - ssthresh = ½ min (Cwnd, fenêtre récepteur)
  - Augmenter linéairement la taille Cwnd (phase 2)
  - Pas de slow-start (pas de phase 1).

Contrôle de congestion



Conclusion

- Protocoles de transport => de bout-en-bout
  - multiplexing/demultiplexing
  - Principes de transfert fiable de données
  - Contrôle de flux
  - Contrôle de congestion
  - Principalement deux protocoles de transport
    - UDP : non fiable, seulement vérification des erreurs
    - TCP : fiable, contrôle de congestion, reprise sur erreurs (ACK + timeout)…
  - Implémentés partout
- Cours suivant :
  - On quitte la bordure du réseau (couches application et transport)...
  - ... pour entrer au coeur du réseau.