

## Architecture des Réseaux (ARES) 3/5 : Transport

Olivier Fourmaux (olivier.fourmaux@upmc.fr)

Version 6.2

### Couche transport

Compréhension des principes de base de la couche transport<sup>1</sup>

- multiplexage
- transfert fiable
- contrôle de flux
- contrôle de congestion

Etude des protocoles de transport dans l'Internet

- UDP : transport sans connexion
- TCP : transport orienté-connexion
- contrôle de congestion de TCP

<sup>1</sup>Nombreux emprunts au livre de J. F. Kurose et K. W. Ross, *Computer Networking : A Top Down Approach Featuring the Internet*, 3e édition (Addison-Wesley)

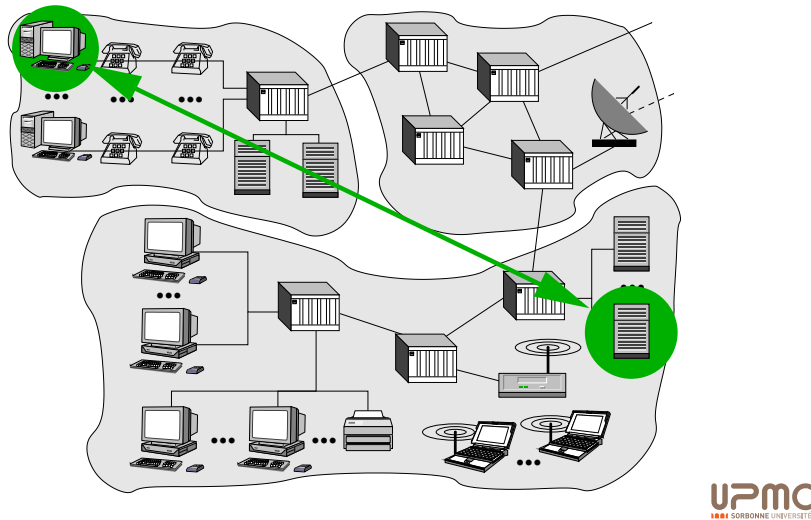
## ARES : Plan du cours 3/5

- 1 Service de base
  - Rappels sur la couche transport
  - Multiplexage et démultiplexage
  - UDP : un protocole en mode non connecté
- 2 Service fiable
  - Principes de transfert de données fiable
  - TCP : un protocole en mode orienté connexion
  - TCP : mécanismes de fiabilisation
- 3 Contrôle de congestion
  - Principes généraux
  - Mécanismes de TCP

### ARES : Plan du cours 3/5

- 1 Service de base
  - Rappels sur la couche transport
  - Multiplexage et démultiplexage
  - UDP : un protocole en mode non connecté
- 2 Service fiable
  - Principes de transfert de données fiable
  - TCP : un protocole en mode orienté connexion
  - TCP : mécanismes de fiabilisation
- 3 Contrôle de congestion
  - Principes généraux
  - Mécanismes de TCP

## Couche transport



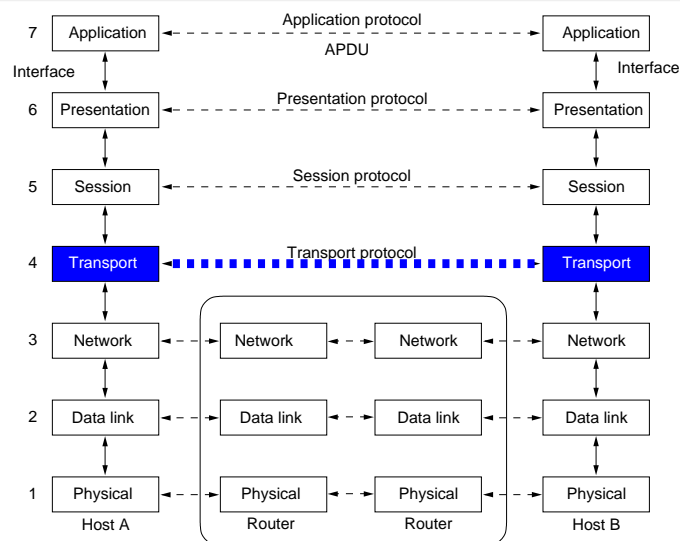
## Couche transport

La **Couche transport** permet de faire **communiquer directement** deux ou plusieurs entités sans avoir à se préoccuper des différents éléments de réseaux traversés :

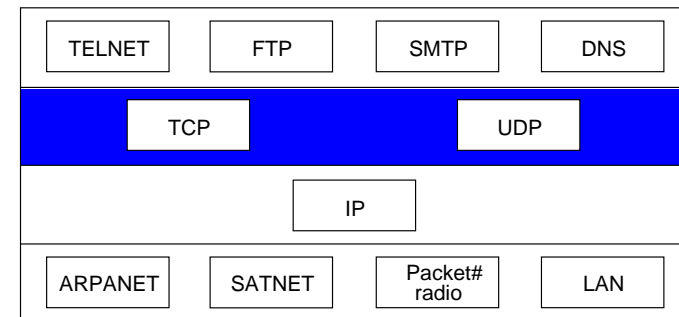
- associations virtuelles entre **processus**
- communication de bout-en-bout (*end-to-end*)
  - abstraction de la **topologie** et des **technologies** sous-jacentes
  - fonctionne dans les machines d'extrémité
    - **émetteur** : découpe les messages de la couche applicative en segments et les "descend" à la couche réseau
    - **récepteur** : réassemble les segments en messages et les "remonte" à la couche application

⇒ 2 **modèles** définissent les fonctionnalités associés à chaque couche...

## Couche transport : OSI



## Couche transport : TCP/IP



## Couche transport : Internet

2 protocoles de transport standard : TCP et UDP

- transfert fiable et ordonné : **TCP**
  - gestion de la connexion
  - contrôle de flux
  - contrôle de congestion
- transfert non fiable non ordonné : **UDP**
  - service *best effort* ("au mieux") d'IP
  - très léger
- non disponible :
  - garanties de débit
  - garanties temporelles
    - délais non bornés
    - jigue imprévisible

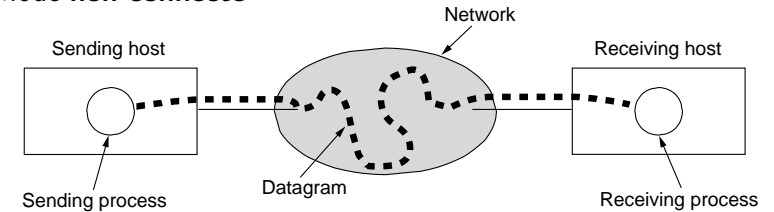
## Couche transport : primitives

Interface de **programmation** (applications ou développeurs)

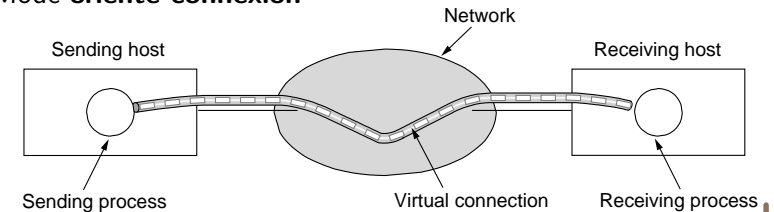
- exemple de primitives de base en mode connecté :
  - LISTEN
  - CONNECT
  - SEND
  - RECEIVE
  - DISCONNECT

## Couche transport : 2 approches

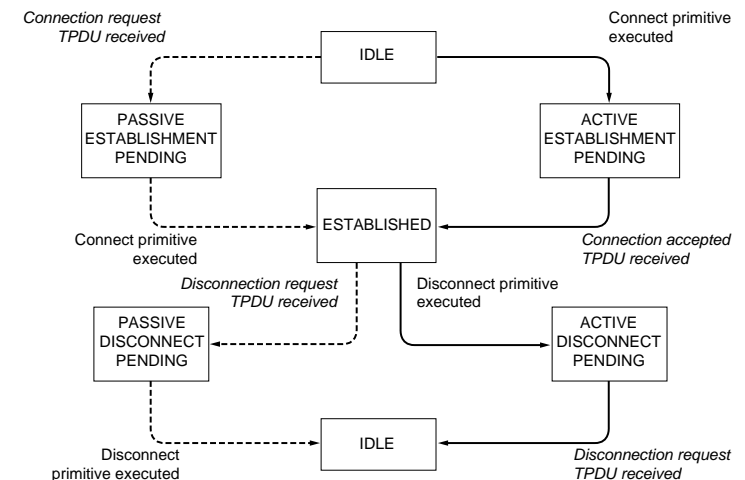
Mode **non connecté**



Mode **orienté connexion**

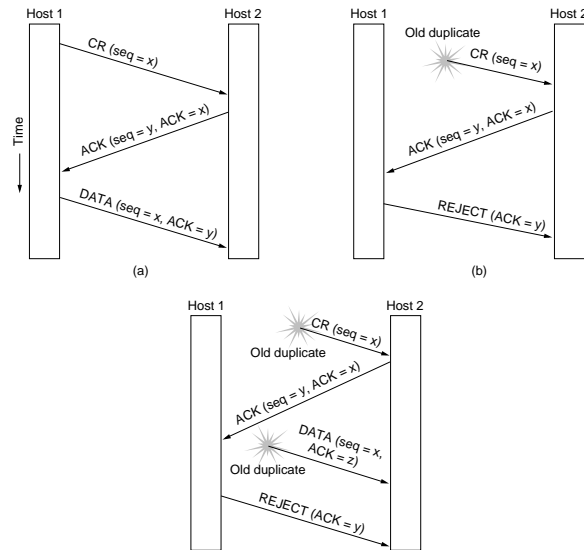


## Couche transport : automate de connexion



pictures from TANENBAUM A. S. *Computer Networks 3rd edition*

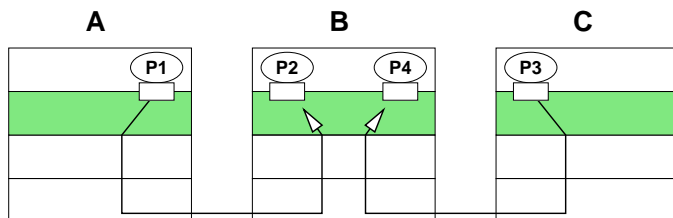
## Couche transport : établissement (call setup)



## Multiplexage/Démultiplexage

Les **processus** applicatifs transmettent leurs données au système à travers des **sockets** : Le **multiplexage** consiste à regrouper ces données.

- **mux** (à l'émetteur) :
  - ajout d'un entête à chaque bloc de données d'un socket
  - collecte les données de plusieurs socket
- **demux** (au récepteur) :
  - fourniture des données au socket correspondant



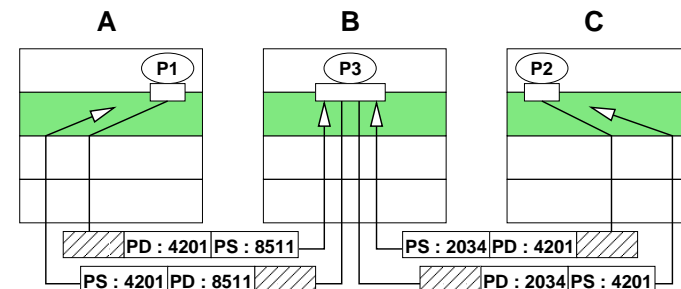
## ARES : Plan du cours 3/5

- 1 Service de base
  - Rappels sur la couche transport
  - Multiplexage et démultiplexage
  - UDP : un protocole en mode non connecté
- 2 Service fiable
  - Principes de transfert de données fiable
  - TCP : un protocole en mode orienté connexion
  - TCP : mécanismes de fiabilisation
- 3 Contrôle de congestion
  - Principes généraux
  - Mécanismes de TCP

## Démultiplexage en mode non connecté

Association d'un socket avec un numéro de port

- identification du DatagramSocket : (@IPdest, numPortDest)
- réception d'un **datagramme** à un hôte :
  - vérification du numPortDest contenu
  - envoi au socket correspondant à numPortDest
    - $\forall \text{@IPsource}, \forall \text{numPortSource}$



## Multiplexage en mode orienté connexion

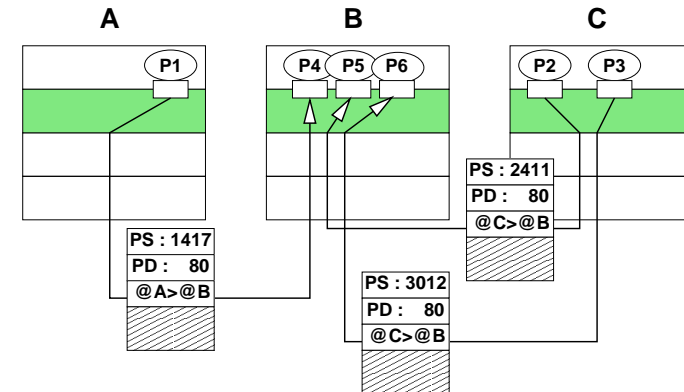
Association relative à une **connexion** entre deux processus

- identification du StreamSocket par le quadruplet :
  - adresse source : @IPsource
  - port source : numPortSource
  - adresse destination : @IPdest
  - port destination : numPortDest
- réception d'un **segment** à un hôte :
  - vérification du quadruplet contenu
  - envoi au socket correspondant au quadruplet
    - un serveur web peut avoir plusieurs connexions simultanée

## Démultiplexage en mode orienté connexion (1)

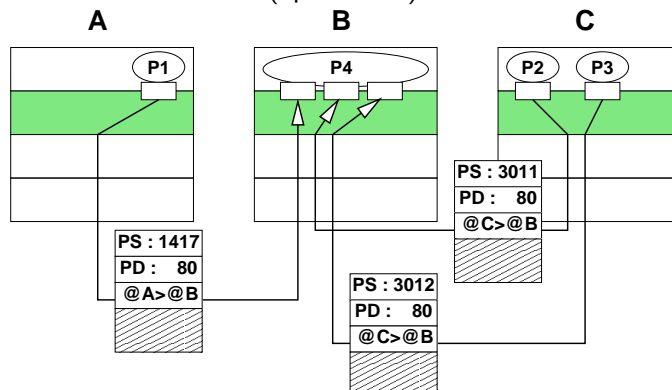
Serveur web classique (apache 1.x)

- un socket par connexion
  - HTTP en mode non persistant : un socket par requête !

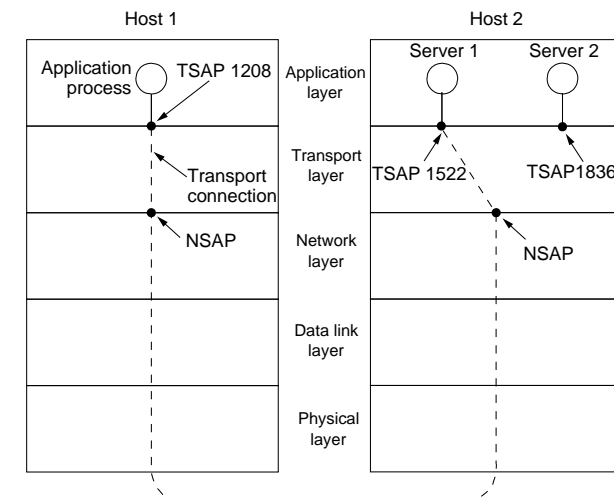


## Démultiplexage en mode orienté connexion (2)

Serveur web multi-threadé (apache 2.x)



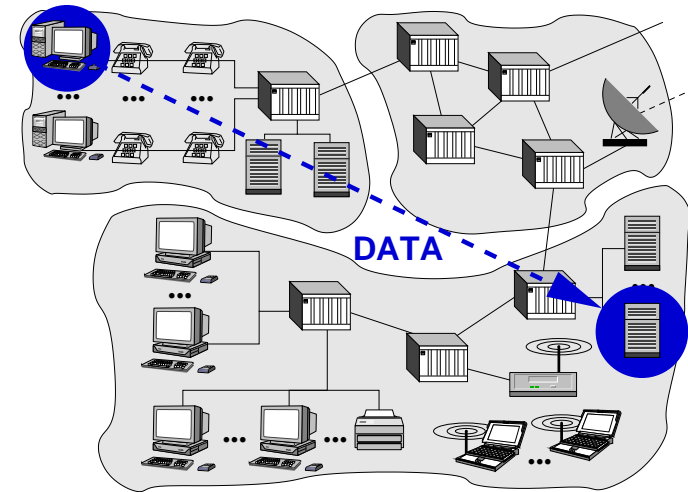
## Multiplexage : dénominations OSI



## ARES : Plan du cours 3/5

- 1 Service de base
  - Rappels sur la couche transport
  - Multiplexage et démultiplexage
  - UDP : un protocole en mode non connecté
- 2 Service fiable
  - Principes de transfert de données fiable
  - TCP : un protocole en mode orienté connexion
  - TCP : mécanismes de fiabilisation
- 3 Contrôle de congestion
  - Principes généraux
  - Mécanismes de TCP

## UDP

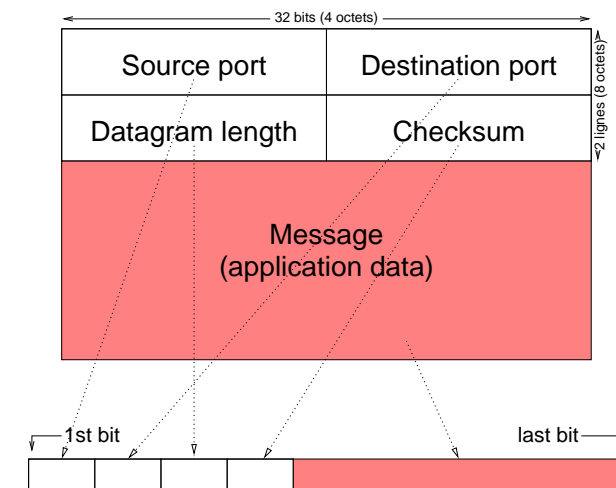


## UDP

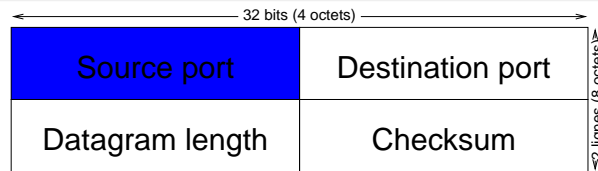
### User Datagram Protocol [RFC 768]

- protocole de transport Internet basique (sans fioriture)
- service *best effort* :
  - les datagrammes transférés peuvent être...
    - perdus
    - dupliqués
    - désordonnés
- service sans connexion :
  - pas d'échange préalable
  - pas d'information d'état aux extrémités
    - chaque datagramme géré indépendamment

## Datagramme UDP

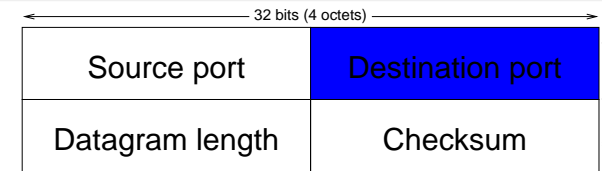


## UDP : port source



- 16 bits (65535 ports)
- **multiplexage** à la source
- identification du socket pour un retour potentiel
- allocation fixe ou dynamique (généralement dans le cas d'un client)
- répartition de l'espace des ports :
  - $0 \leq \text{numPort} \leq 1023$  : accessible à l'administrateur
    - socket serveurs (généralement)
  - $1024 \leq \text{numPort}$  : accessible aux utilisateurs
    - socket clients (généralement)

## UDP : port destination



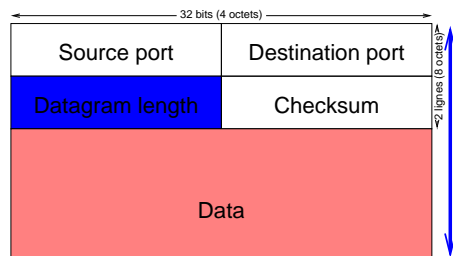
- 16 bits (65535 ports)
- **démultiplexage** à la destination
- le destinataire doit être à l'écoute sur ce port
- négociation du port ou *well-known ports* (port réservés) :
 

```

Unix> cat /etc/services |grep udp
domain          53/udp
echo            7/udp
discard         9/udp
daytime        13/udp
chargen        19/udp
ssh            22/udp
time           37/udp
..

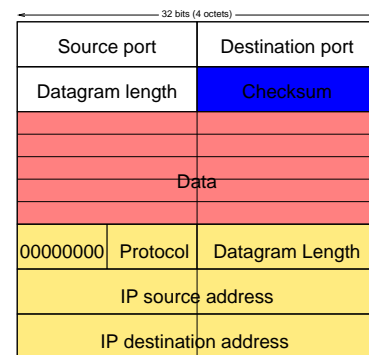
```

## UDP : longueur du datagramme



- 16 bits (64 Koctets maximum)
- longueur totale avec les données exprimée en **octets**

## UDP : contrôle d'erreur



- 16 bits
- contrôle d'erreur **facultatif**
- émetteur :
  - ajout *pseudo-header*
  - $\text{checksum}^a = \sum \text{mot}_{16\text{bits}}$
- récepteur :
  - ajout *pseudo-header*
  - recalcul de  $\sum \text{mot}_{16\text{bits}}$ 
    - $= 0$  : pas d'erreur détectée toujours possible...
    - $\neq 0$  : erreur (destruction silencieuse)

<sup>a</sup>Somme binaire sur 16 bits avec report de la retenue débordante ajoutée au bit de poids faible

## UDP : arguments pour un transport sans connexion

Motivation pour le choix d'un service transport non connecté :

- ressources limitées aux extrémités
  - pile TCP/IP limitée
  - absence d'**état** dans les hôtes
  - capacité de traitement limitée
- besoin d'échange rapide
  - pas d'**établissement** de connexion
- besoin d'efficacité
  - entête réduit**
- contraintes temporelles
  - retransmission** inadapté
  - pas de **contrôle** du débit d'émission
- besoin de nouvelles fonctionnalités
  - remontés dans la couche application...

## UDP : Interface socket

```
#include <sys/types.h>
#include <sys/socket.h>

# Create a descriptor
int socket(int domain, int type, int protocol);
# domain : PF_INET for IPv4 Internet Protocols
# type : SOCK_DGRAM Supports datagrams (connectionless, unreliable msg of a fixed max length)
# protocol : UDP (/etc/protocols)

# Bind local IP and port
int bind(int s, struct sockaddr *my_addr, socklen_t addrlen);

# Send an outgoing datagram to a destination address
int sendto(int s, const void *msg, size_t len, int flags,
           const struct sockaddr *to, socklen_t tolen);

# Receive the next incoming datagram and record its source address
int recvfrom(int s, void *buf, size_t len, int flags,
             struct sockaddr *from, socklen_t *fromlen);

# End : deallocate
int close(int s);
```

## UDP : exemples d'applications

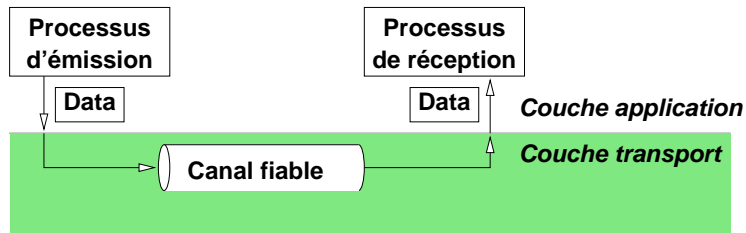
- applications classiques :
  - résolution de noms (DNS)
  - administration du réseau (SNMP)
  - protocole de routage (RIP)
  - protocole de synchronisation d'horloge (NTP)
  - serveur de fichiers distants (NFS)
    - fiabilisation implicite par redondance temporelle
    - fiabilisation explicite par des mécanismes ajoutés dans la couche application
- applications *multicast* ➡ U.E. **ING**
- applications multimédia ➡ U.E. **MMQOS**
  - diffusion multimédia, *streaming* audio ou vidéo
  - téléphonie sur Internet
  - visioconférence
    - contraintes temporelles
    - tolérance aux pertes

## ARES : Plan du cours 3/5

- Service de base
  - Rappels sur la couche transport
  - Multiplexage et démultiplexage
  - UDP : un protocole en mode non connecté
- Service fiable
  - Principes de transfert de données fiable
  - TCP : un protocole en mode orienté connexion
  - TCP : mécanismes de fiabilisation
- Contrôle de congestion
  - Principes généraux
  - Mécanismes de TCP



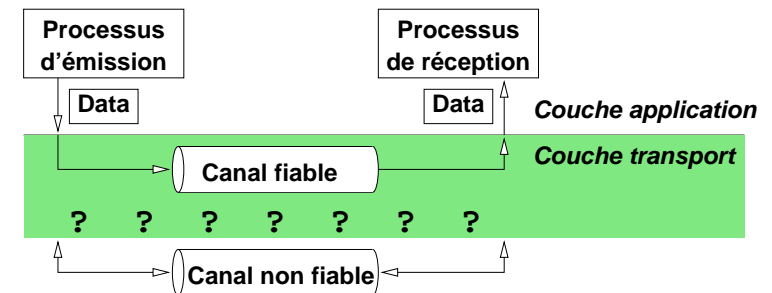
## Couche transport et fiabilité (1)



Problématique multi-couche :

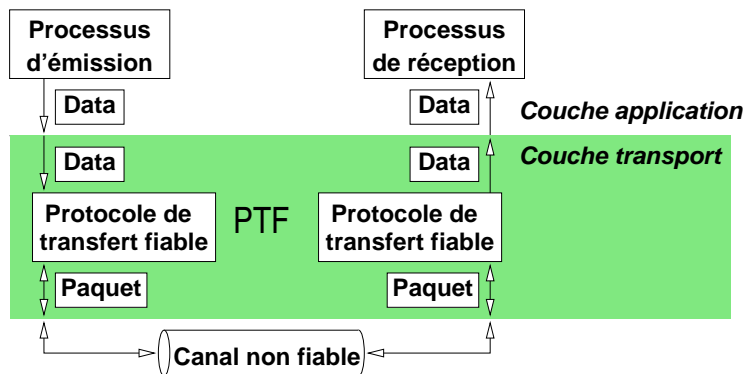
- couche application
- couche transport
- couche liaison

## Couche transport et fiabilité (2)

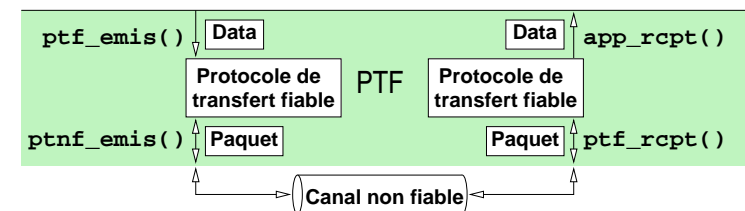


Les caractéristiques du **canal non fiable** déterminent la complexité du **protocole de transfert fiable (PTF)**.

## Couche transport et fiabilité (3)



## Protocole de Transfert Fiable (PTF)



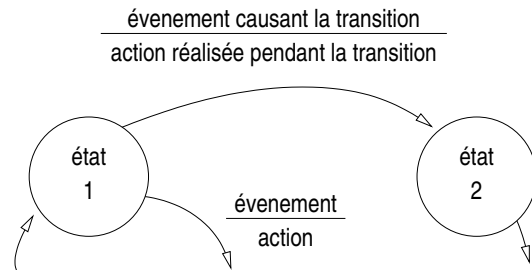
Primitives de base du PTF :

- `ptf_emis()` : appelée par la couche supérieure (application) pour envoyer des données à la couche correspondante du récepteur
- `ptfn_emis()` : appelée par le PTF transférer un paquet sur le canal non fiable vers le récepteur
- `ptf_rcpt()` : appelée lorsqu'un paquet arrive au récepteur
- `app_rcpt()` : appelée par le PTF pour livrer les données

## PTF et AEF

Nous allons construire progressivement le **PTF**

- transfert de données dans un seul sens
  - information de contrôle dans les 2 directions
- spécification de l'émetteur et du récepteur par des Automates à Etats Finis (AEF) :



## PTF v2.0

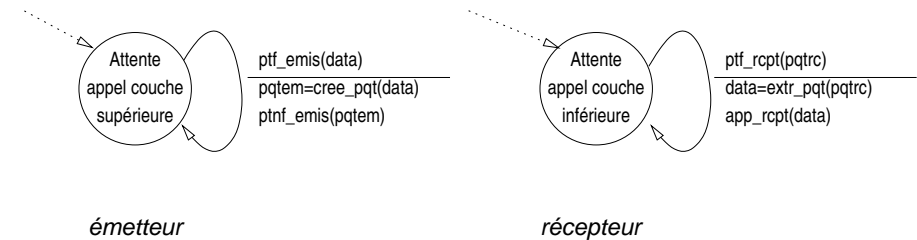
Transfert fiable sur un **canal avec des erreurs**

- canal sous-jacent pouvant changer la valeur des bits d'un paquet
  - introduction de contrôle d'erreur :
    - ctrlerr : redondance rajoutée au paquet
- Comment récupérer les erreurs ?
  - acquiescement (ACK)** : le récepteur indique explicitement la réception correcte d'un paquet
  - acquiescement négatif (NAK)** : le récepteur indique explicitement la réception incorrecte d'un paquet
    - l'émetteur ré-émet le paquet sur réception d'un NAK
- nouveau mécanisme dans PTF v2.0 :
  - détection d'erreur
    - valide(pqt) : vrai si le contrôle d'erreur de pqt est correct
    - erreur(pqt) : vrai si le contrôle d'erreur de pqt est incorrect
  - retour d'information (feedback) du récepteur (ACK et NAK)

## PTF v1.0

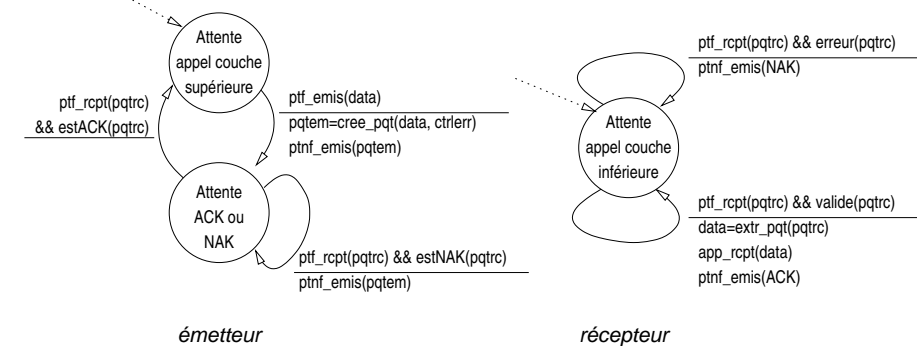
Transfert fiable sur un canal sans erreur

- canal sous-jacent complètement fiable
  - pas de bits en erreur
  - pas de perte de paquets
- automates séparés pour l'émetteur et le récepteur :



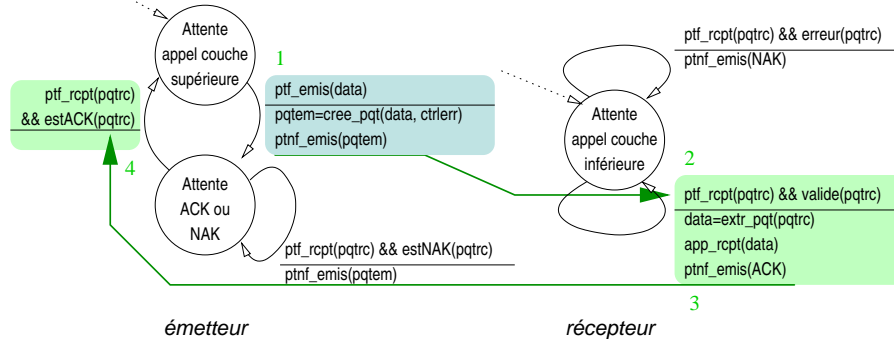
## PTF v2.0

Transfert fiable sur un **canal avec des erreurs** :



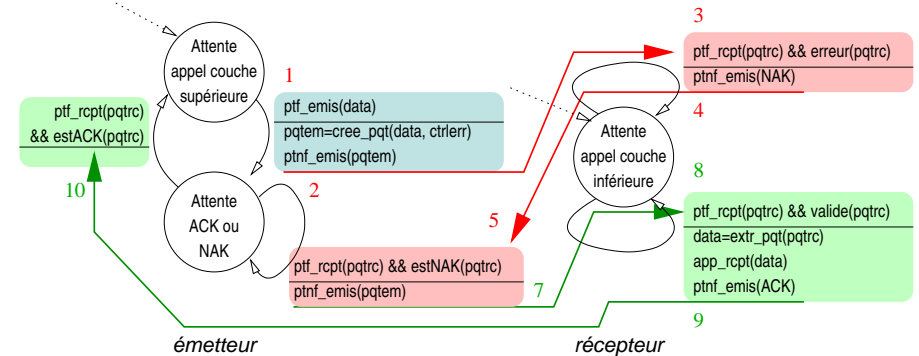
## PTF v2.0 : ACK

Transfert fiable lorsqu'il n'y a pas d'erreur :



## PTF v2.0 : NAK

Transfert fiable lorsqu'il y a une erreur :



## PTF v2.0 : discussion

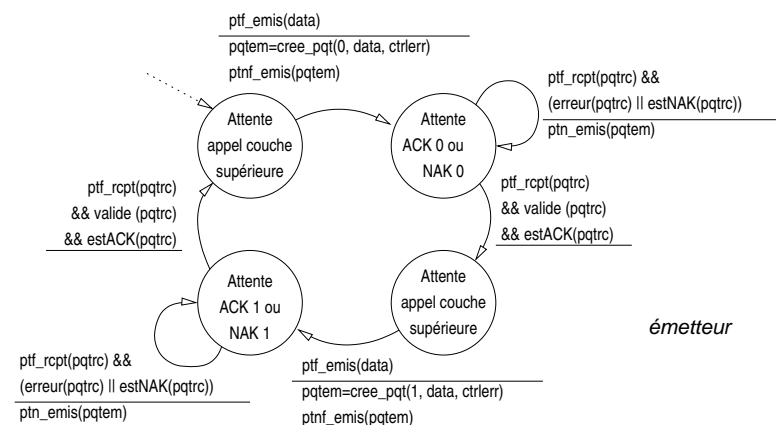
PTF v2.0 est un protocole *stop and wait* :

- émetteur envoie un paquet et attend la réponse du récepteur
- peu performant...

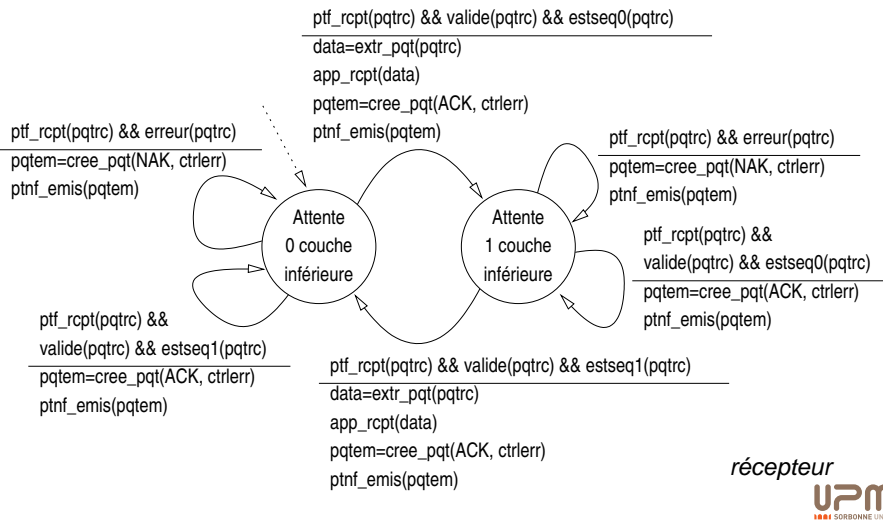
PTF v2.0 à un défaut majeur !

- *Que se passe-t-il si les ACK ou NAK sont incorrect ?*
  - pas d'information sur l'état du récepteur
  - une retransmission simple risque de dupliquer les données
- gestion des duplicats :
  - émetteur **retransmet** le paquet courant si ACK/NAK incorrect
  - émetteur insère un **numéro** de séquence à chaque paquet
  - récepteur **supprime** les paquets dupliqués
    - inclu dans **PTF v2.1**

## PTF v2.1 : émetteur



## PTF v2.1 : récepteur



## PTF v2.1 : discussion

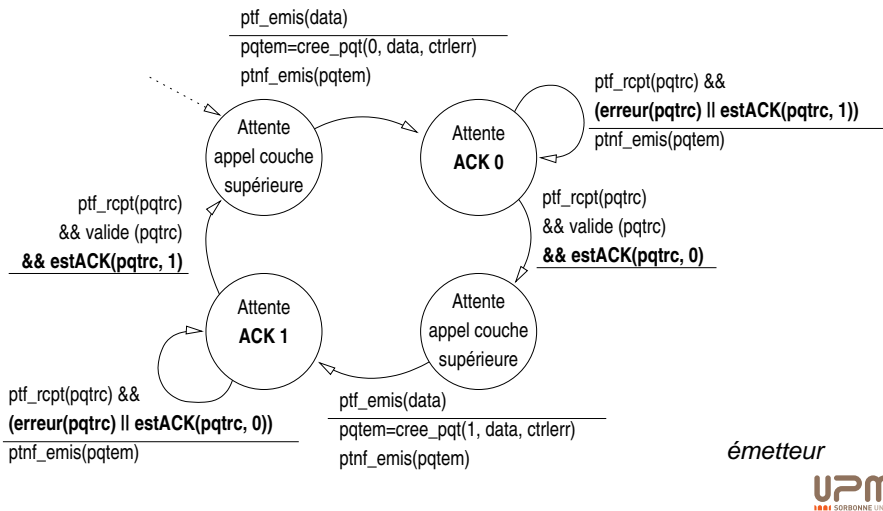
### Comportement des extrémités avec PTF v2.1

- **émetteur**
  - ajout de numéro de séquence à chaque paquet
    - 2 suffisent (0 et 1)
  - contrôle d'erreur sur les ACK et NAK
    - 2 fois plus d'états
- **récepteur**
  - vérification que le paquet n'est pas dupliqué
    - l'état où l'on se trouve indique le numéro de séquence attendu

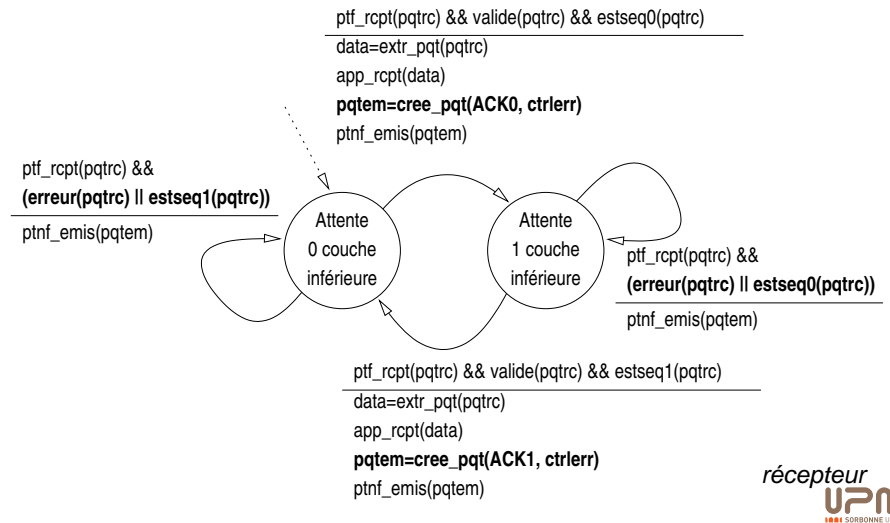
### Peut-on éliminer les NAK ?

- remplacement des NAK par **ACK du dernier paquet valide** reçu
  - récepteur inclut le numéro de séquence correspondant dans le ACK
  - ACK dupliqué au récepteur = NAK reçu au récepteur
    - intégré dans **PFT v2.2**

## PTF v2.2 : émetteur



## PTF v2.2 : récepteur



## PTF v3.0

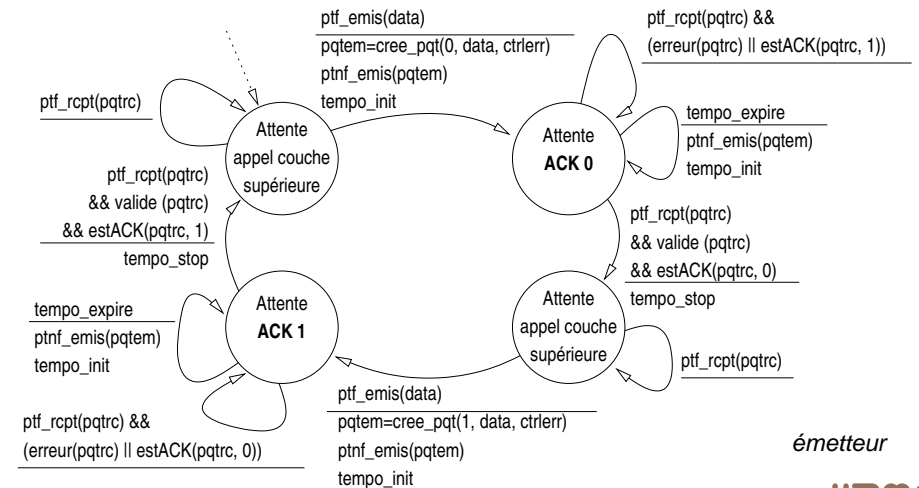
### Transfert fiable sur un canal avec erreurs et pertes

- canal sous-jacent peut aussi perdre des paquets (data ou ACK)
  - $ctrlerr + numSeq + ACK + retransmission$ 
    - insuffisant : l'absence d'un paquet bloque l'automate !

### Temporisation des retransmission

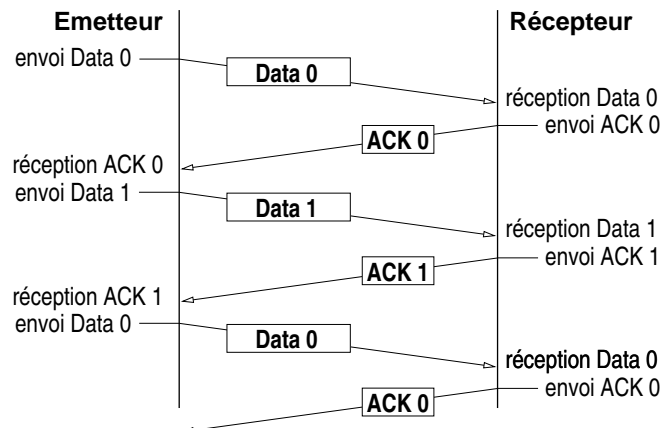
- estimation d'un temps de retour de ACK raisonnable
  - déclenchement d'une temporisation à l'émission d'un paquet
    - $tempo\_init$
  - ACK avant l'expiration de la temporisation ➡ rien
    - $tempo\_stop$
  - pas de ACK à l'expiration de la temporisation
    - retransmission
      - $tempo\_expire$
- si le ACK est seulement en retard...
  - retransmission = duplication
    - détectée grâce au numéro de séquence

## PTF v3.0 : émetteur

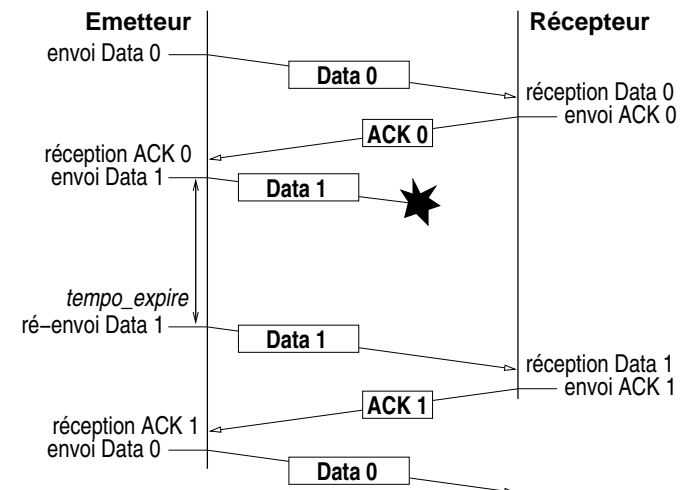


émetteur

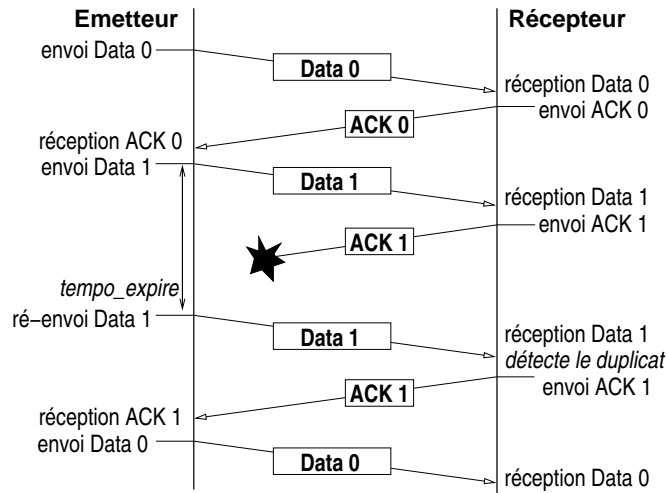
## PTF v3.0 : sans perte



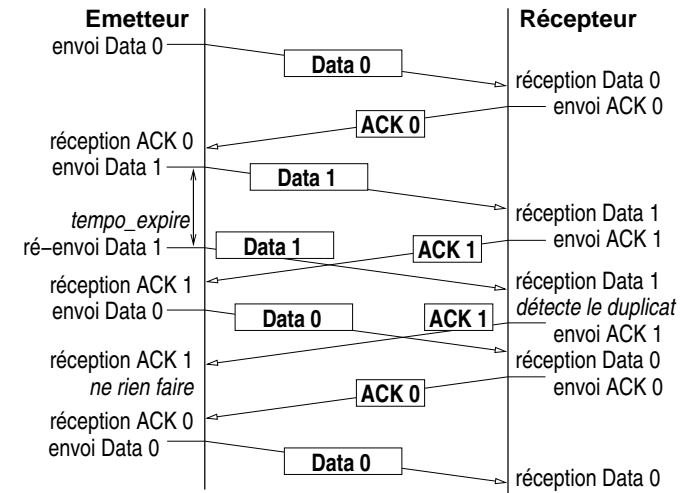
## PTF v3.0 : perte d'un paquet de données



## PTF v3.0 : perte d'un ACK



## PTF v3.0 : fin de temporisation prématurée

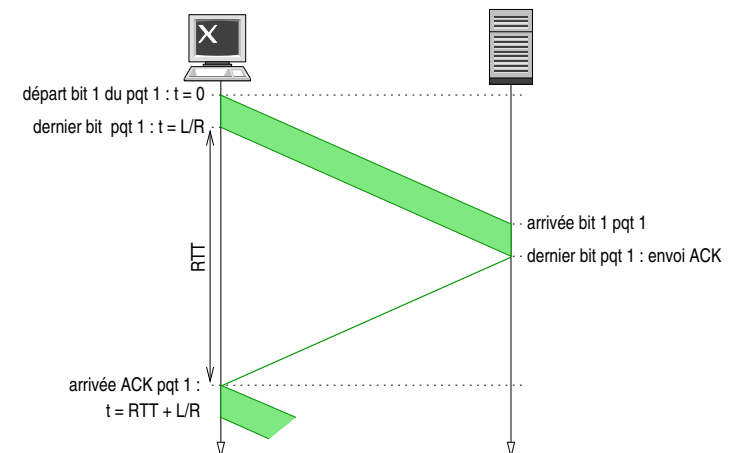


## PTF v3.0 : performance

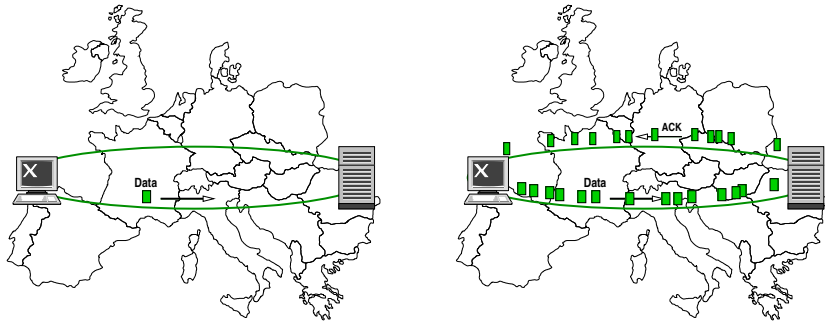
PFT v3.0 fonctionne mais quelles sont ses performances ?

- exemple de communication :
  - débit du lien :  $D_{\text{reseau}} = 1 \text{ Gbps}$ ,
  - délais de bout-en-bout :  $d = 40 \text{ ms}$  ( $d_{AR} = 80 \text{ ms}$ )
  - paquets de longueur 1000 octets ( $L_{\text{paquet}} = 8000 \text{ b}$ )
- $T_{\text{transmission}} = L_{\text{paquet}} / D_{\text{reseau}} = 8.10^3 / 10^9 = 8 \mu\text{s}$
- efficacité émetteur ( $E_{\text{emis}}$ ) : fraction de temps en émission
  - $E_{\text{emis}} = \frac{L_{\text{paquet}} / D_{\text{reseau}}}{L_{\text{paquet}} / D_{\text{reseau}} + d_{AR}} = \frac{8.10^{-6}}{8.10^{-6} + 8.10^{-2}} = \frac{1}{10000}$
  - $D_{\text{transport}} = L_{\text{paquet}} / d_{AR} = 8.10^3 / 8.10^{-2} = 100 \text{ Kbps}$ 
    - le protocole limite l'utilisation des ressources disponibles

## PTF v3.0 : stop and wait



## Protocole pipeline



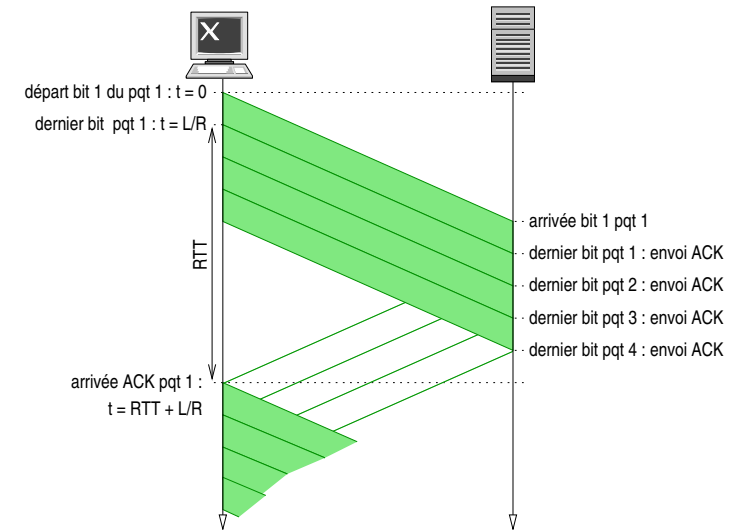
- l'émetteur autorise plusieurs paquets en attente d'acquittement
  - numéro de séquences étendus
  - tampons d'émission et de réception
    - 2 types de protocole pipeliné : **Go-Back-N** et **Retransmissions sélectives**

## Go-Back-N : émetteur

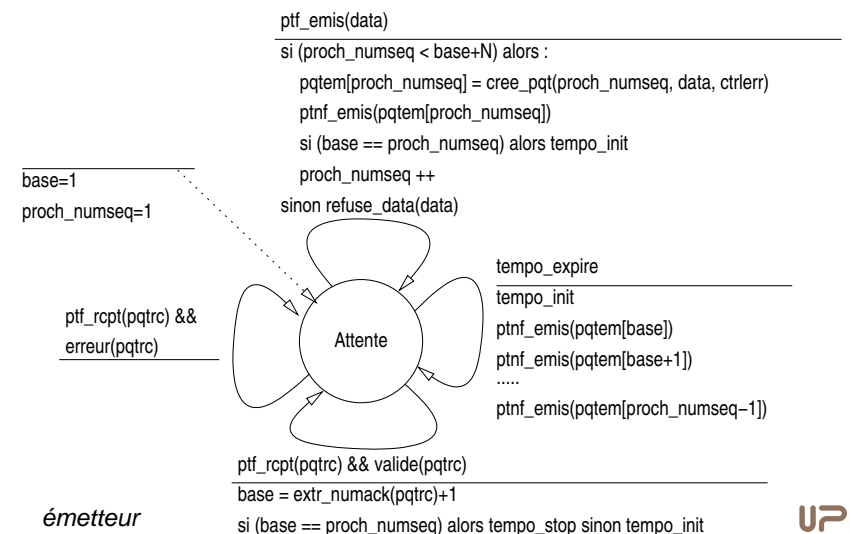
Emetteur avec gestion *Go-Back-N* (retour arrière).

- entête des paquets avec k bits de numéro de séquence
- acquittements **cumulatifs**
  - ACK(n) acquitte tous les paquets jusqu'au numéro de séquence n
- fenêtre d'au maximum N paquets non acquités :
- une temporisation pour les paquets en attente (*in-flight*)
  - tempo\_expire(n) : retransmission du paquet n et des suivants avec numéro de séquence supérieur

## Performance pipeline



## PTF v4.0 : émetteur



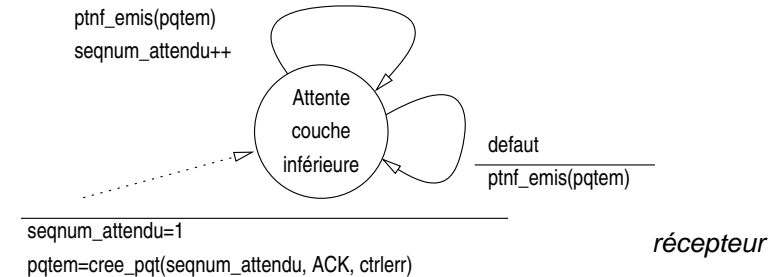
## Go-Back-N : récepteur

Récepteur avec gestion *Go-Back-N* (retour arrière).

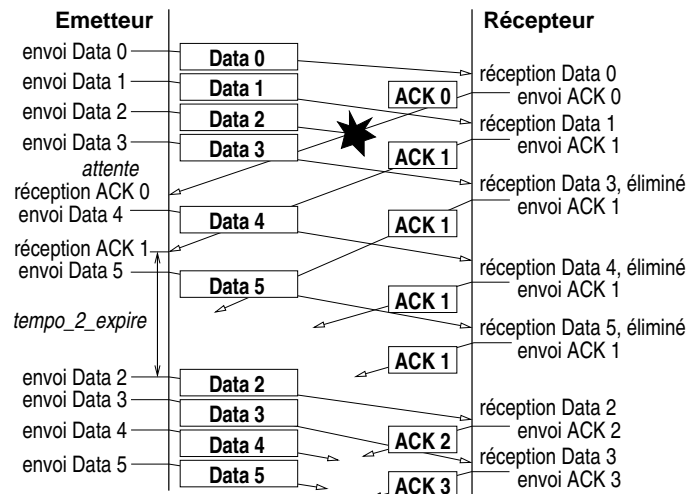
- **seulement des ACK :**
  - envoie toujours des ACK avec le plus élevé des *seqnum* de paquets valides **ordonnés**
    - peut générer des ACK dupliqués
    - seul *seqnum\_attendu* est mémorisé
- **déséquencement :**
  - élimine les paquets déséquencés
    - pas de tampon au niveau du récepteur
  - ré-émet le ACK avec le plus élevé des *seqnum* de paquets valides **ordonnés**

## PTF v4.0 : récepteur

```
ptf_rcpt(pqtrc) && valide(pqtrc) && estseqnum(pqtrc, seqnum_attendu)
data=extr_pqt(rcpqt)
app_rcpt(data)
pqtem=cree_pqt(seqnum_attendu, ACK, ctrlerr)
ptnf_emis(pqtem)
seqnum_attendu++
```



## PTF v4.0 : exemple



## Retransmissions sélectives : émetteur

Emetteur avec gestion des retransmissions sélectives :

- retransmet **seulement** les paquets non acquittés
- fenêtre d'émission limitée à *N* paquets consécutifs
- algo :
  - *pft\_emis(data)*
    - envoie un paquet si *seqnum* dans la fenêtre
  - *tempo\_expire(n)*
    - retransmet paquet *n*
    - *tempo\_init(n)*
  - *ACK(n)*
    - marque le paquet *n* reçu
    - si *n* est le plus petit paquet non acquitté, décale la fenêtre

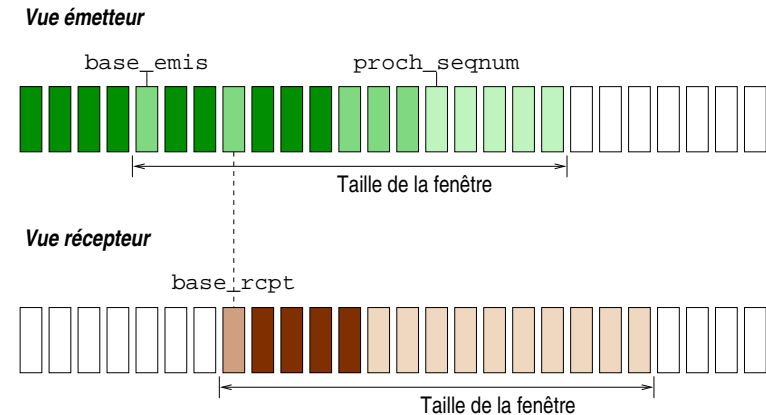


## Retransmissions sélectives : récepteur

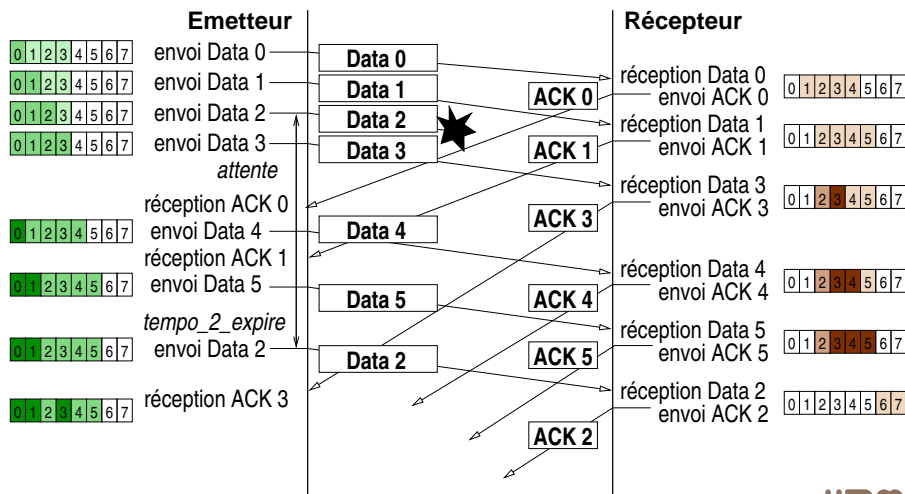
Récepteur avec gestion des retransmissions sélectives :

- acquitte **explicitement** chaque paquet valide reçu
- tampon de réception pour re-séquencement
- algo :
  - $\text{ptf\_rcpt}(n)$   
( $\text{base\_rcpt} \leq n \leq \text{base\_rcpt} + N - 1$ )
    - ACK(n)
    - si déséquenté : tampon
    - si séquenté :  $\text{app\_emis}(\text{data})$ , est le plus petit paquet non acquitté, décale la fenêtre
  - $\text{ptf\_rcpt}(n)$   
( $\text{base\_rcpt} - N \leq n \leq \text{base\_rcpt} - 1$ )
    - ACK(n)
  - autre
    - ignore

## Retransmissions sélectives : visualisation



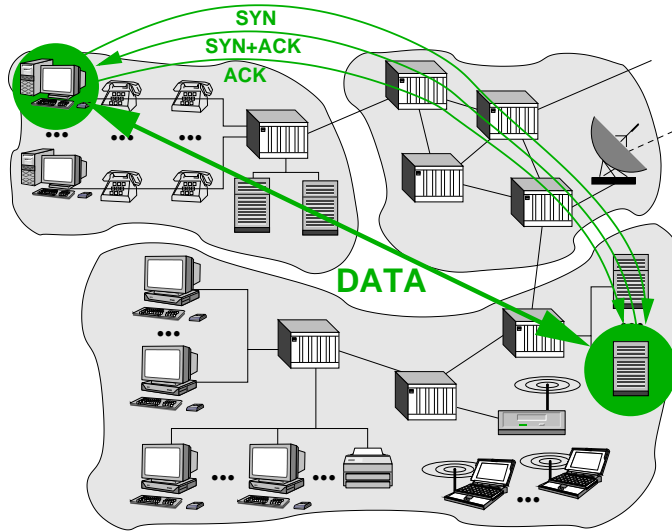
## Retransmissions sélectives : exemple



## ARES : Plan du cours 3/5

- 1 Service de base
  - Rappels sur la couche transport
  - Multiplexage et démultiplexage
  - UDP : un protocole en mode non connecté
- 2 Service fiable
  - Principes de transfert de données fiable
  - TCP : un protocole en mode orienté connexion
  - TCP : mécanismes de fiabilisation
- 3 Contrôle de congestion
  - Principes généraux
  - Mécanismes de TCP

## TCP

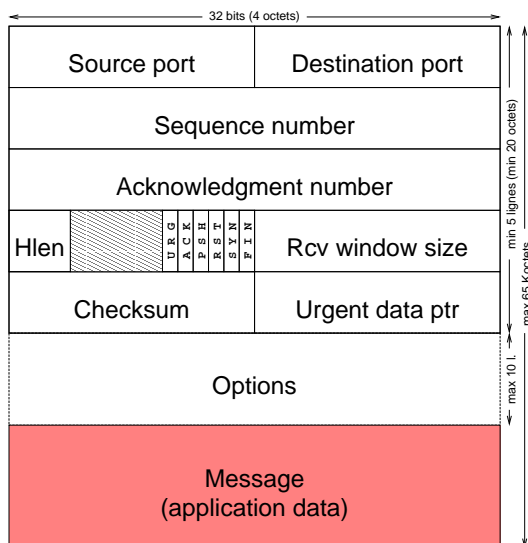


## TCP (*Transmission Control Protocol*)

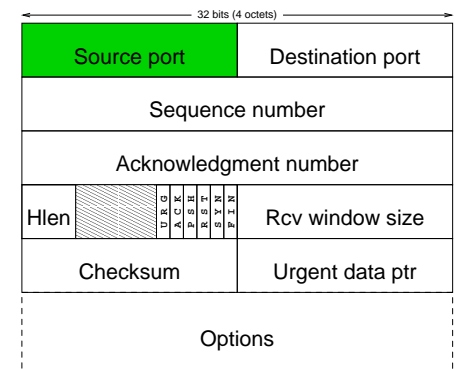
[RFCs : 793, 1122-1123, 2474, 3168, 3260, 4379, 5462 et 5681]

- **service fiable**
  - mécanismes ARQ
- **point-à-point**
  - deux processus (généralement un client et un serveur)
- **flot d'octet continu**
  - pas de frontières de messages
- **orienté connexion**
  - ouverture en trois échanges (*three-way handshake*)
    - initiation des états aux extrémité avant l'échanges de données
  - fermetures courtoise ou brutale
- **connexion bidirectionnelle (*full duplex*)**
  - flux de données dans chaque sens
  - taille maximum du segment : MSS (*Maximun Segment Size*)
- **pipeline**
  - tampons d'émission et de réception
  - double fenêtre asservie aux contrôles de flux et de congestion

## Segment TCP

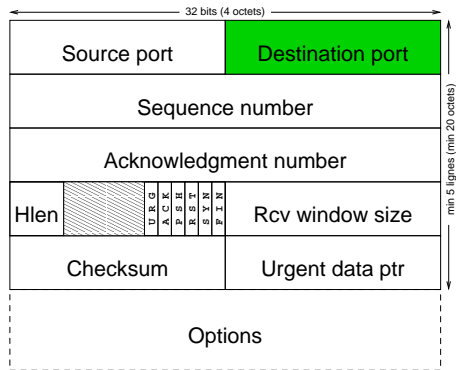


## TCP : Port source



- 16 bits (65535 ports)
- **multiplexage** à la source
- identification partielle du socket (demi-association locale)
- allocation généralement dynamique (dans le cas d'un client)
- répartition espace des ports :
  - $0 \leq \text{numPort} \leq 1023$  : accessible à l'administrateur
    - socket usuel des serveurs
  - $1024 \leq \text{numPort}$  : accessible aux utilisateurs
    - socket usuel des clients

## TCP : Port destination



- 16 bits (65535 ports)
- **démultiplexage** au niveau de la destination
- identification partielle du socket (demi-association distante)
- destinataire à l'écoute sur ce port lors de la création
- négociation du port ou *well-known ports* (réservés) :

```

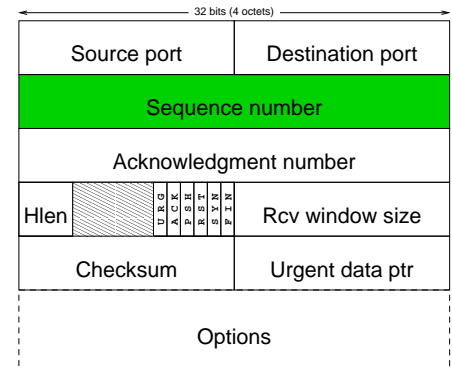
Unix> cat /etc/services|grep tcp
tcpmux      1/tcp
chargen     19/tcp
ftp-data    20/tcp
ftp         21/tcp
ssh         22/tcp ..
  
```

```

telnet      23/tcp
smtp        25/tcp
gopher     70/tcp
finger     79/tcp
www         80/tcp
kerberos   88/tcp
  
```



## TCP : Numéro de séquence (1)

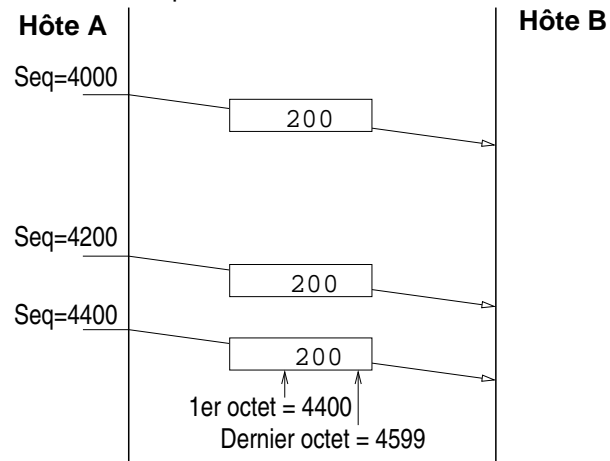


- 32 bits
- associé à chaque **octet** (et non pas à un segment)
  - numérote le **premier** octet des *data*
  - numérotation implicite des octets suivants
  - boucle au bout de 4 Goctets
- détection des **pertes**
- **ordonnancement**

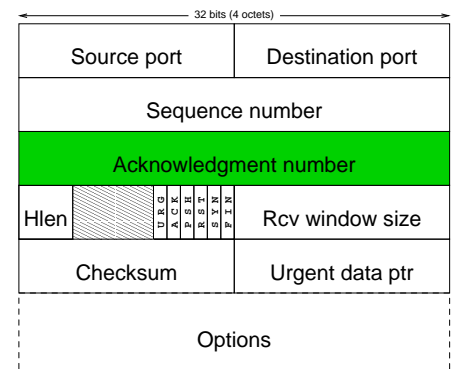


## TCP : Numéro de séquence (2)

Numérotation de chaque **octet** du flot continu de données



## TCP : Numéro d'acquittement (1)

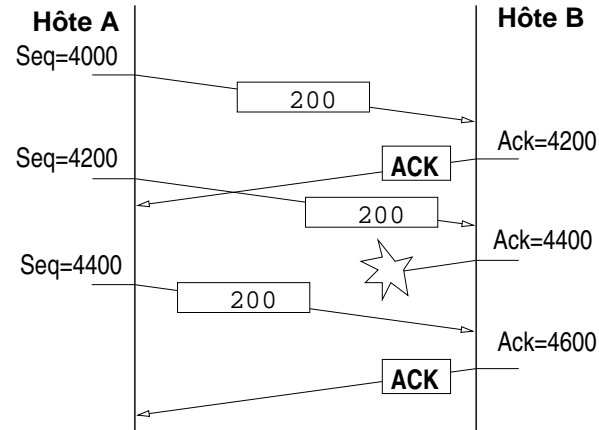


- 32 bits
- *piggybacking*
- indique le numéro du **prochain** octet attendu
- **cumulatif**, indique le premier octet non reçu (d'autres peuvent avoir été reçus avec des numéros de séquence supérieurs)



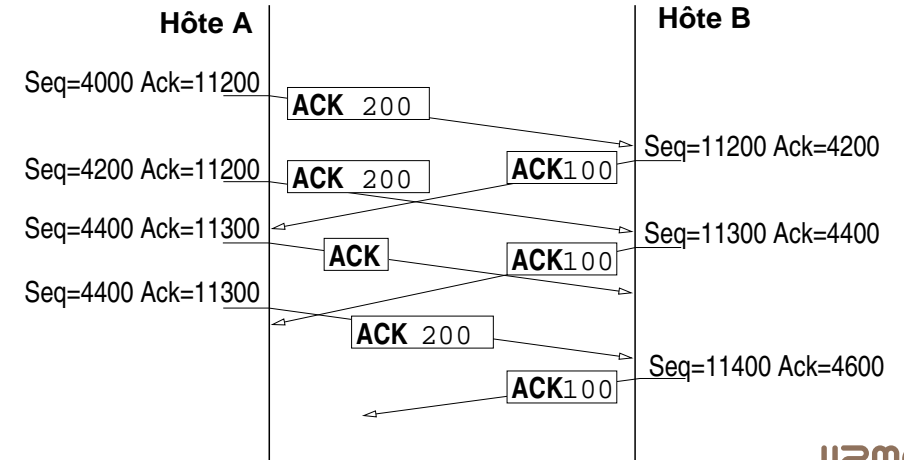
## TCP : Numéro d'acquittement (2)

Acquittement de chaque **octet** du flot continu de données



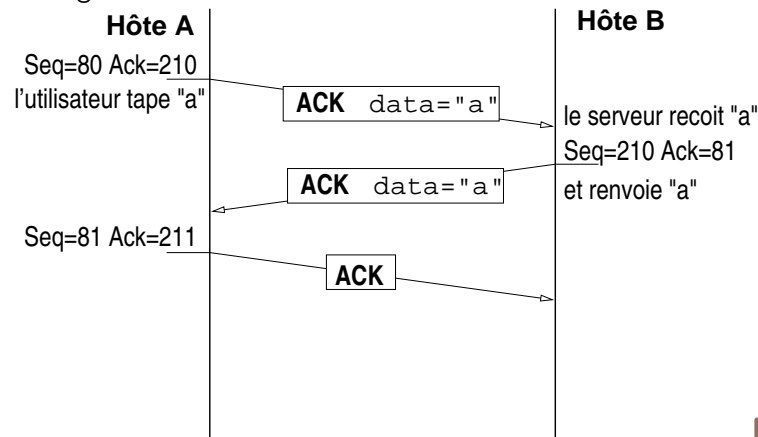
## TCP : Numéro d'acquittement (3)

*Piggybacking*



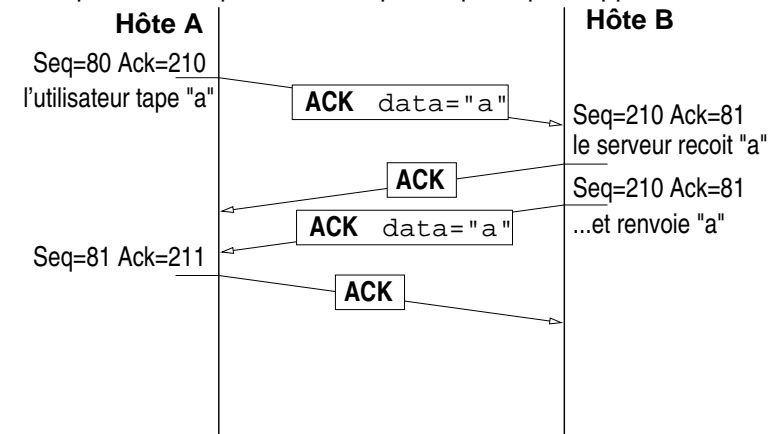
## TCP : Exemple TELNET (1)

Emission d'un caractère frappé et renvoi par le serveur pour l'affichage



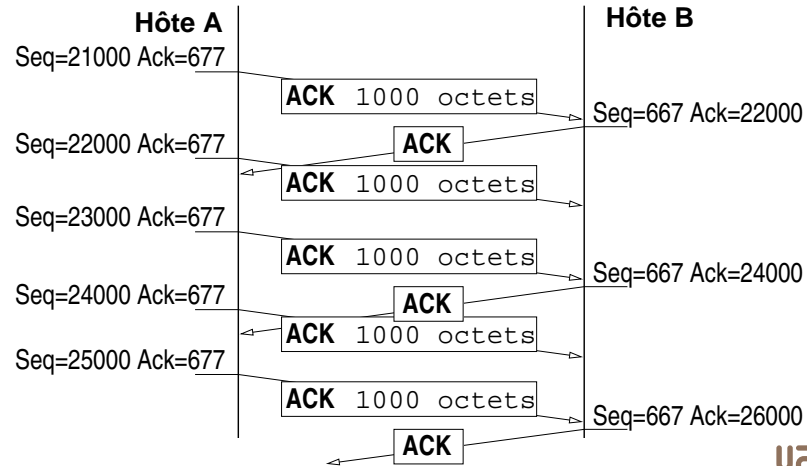
## TCP : Exemple TELNET (2)

Les acquittements peuvent être plus rapide que l'application

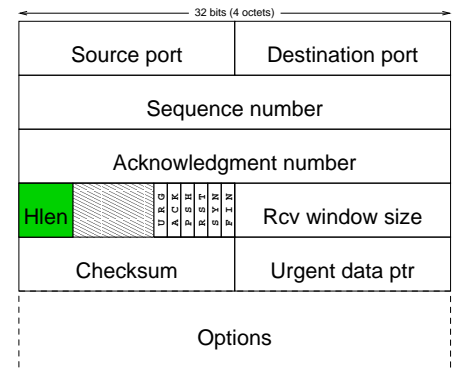


## TCP : Acquittements temporisés

Delayed ACK (attente de deux segments ou 500 ms max.)

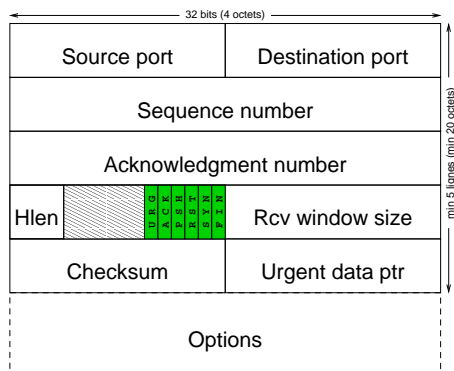


## TCP : Longueur de l'entête



- 4 bits (valeur 15 max)
- nombre de lignes de 32 bits dans l'entête TCP
- nécessaire car le champ option est de longueur variable
  - valeur 5...
    - pas d'options
    - entête TCP de 20 octets minimum
  - ... à 15
    - 10 lignes d'options
    - 40 octets d'options max
    - entête TCP de 60 octets maximum

## TCP : Indicateurs (*flags*)

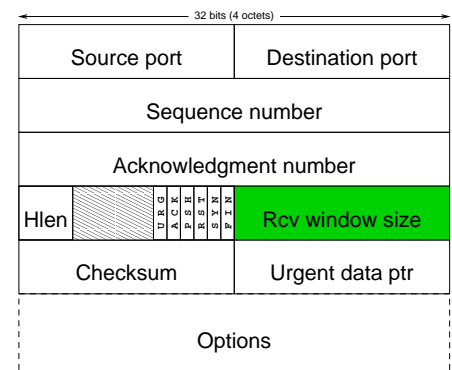


Chacun sur 1 bit indique :

- URG : données **urgentes**
- ACK : le champ **acquittement** est valide
- PSH : envoi **immédiat** avec vidage des tampons
- RST : **terminaison** brutale de la connexion
- SYN : synchronisation lors de l'**ouverture**
- FIN : échanges terminaux lors d'une **fermeture** courtoise

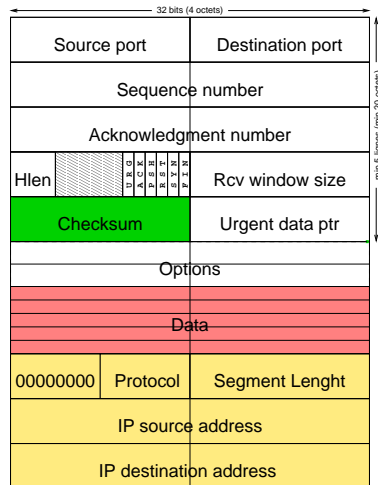
il y en a d'autres récents ➡ U.E. ING

## TCP : Taille de la fenêtre de réception



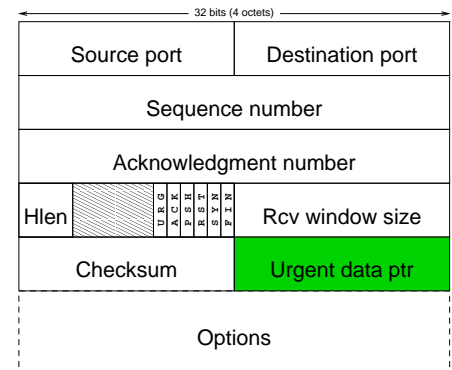
- 16 bits
  - le récepteur peut annoncer jusqu'à 64 Koctets
- piggybacking
- contrôle de flux
  - indique le nombre d'octets disponibles du côté du récepteur
  - dimensionne la taille de la fenêtre d'anticipation de l'émetteur

## TCP : Somme de contrôle du segment



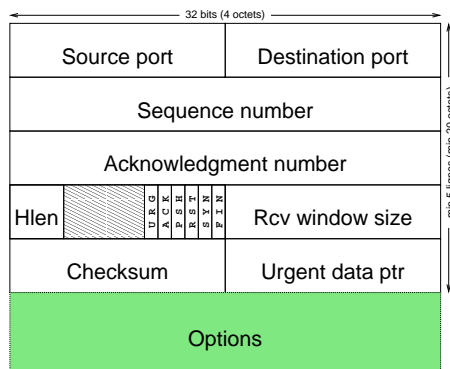
- 16 bits
- idem UDP
- émetteur :
  - ajout *pseudo-header*
  - *checksum*
- récepteur :
  - ajout *pseudo-header*
  - recalcul de  $\sum \text{mot}_{16\text{bits}}$ 
    - = 0 : Ok
    - $\neq 0$  : destruction

## TCP : Pointeur sur les données urgentes



- 16 bits
- permet l'envoi de données spéciales (et non **hors bande**)
- délimite des données traitées en priorité
- indique la fin des données urgentes
  - interprétation de la quantité de données et de leur rôle par l'application

## TCP : Options



Les options sont de la forme TLV ou *Type, Length (octets), Value* :

- END : fin de liste d'options (T=0, non obligatoire)
- NOOP : pas d'opération (T=1, bourrage)
- MSS : négociation du MSS (T=2, L=4, V=MSS)
- WSIZE : mise à l'échelle de la fenêtre par le facteur  $2^N$  (T=3, L=3, V=N)
- SACK : demande d'acquit. sélectif (T=4, L=2, à l'ouverture)
- SACK : acquittement sélectif de  $n$  blocs (T=5, L=2 + 8n, 2n numéros de séquences) ...

## TCP : Gestion de la connexion

Ouverture de la **connexion** préalable à l'échange des données :

- initialisation des variables TCP
  - synchronisation des numéros de séquence
  - création des tampons
  - initiation du contrôle de flot
- **client** : initiateur de la connexion
- **serveur** : en attente de la demande de connexion

Fermeture de la connexion après l'échange des données :

- attente ou non de l'émission des données restantes
- libération des tampons

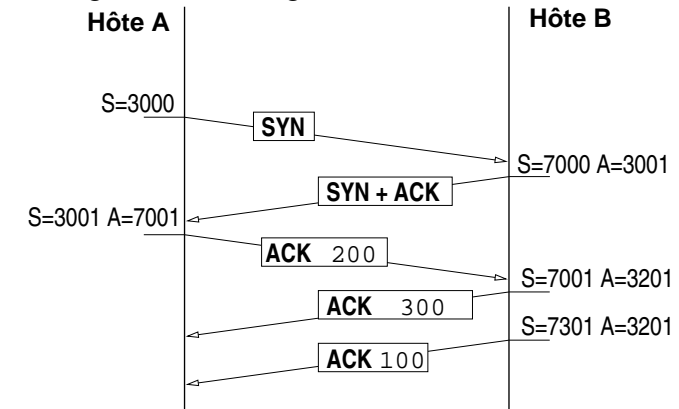
## TCP : Three-Way Handshake (1)

### Echange initial en 3 segments (*Three-Way Handshake*)

- **1** client → serveur : segment TCP avec le bit SYN
  - indique le numéro de séquence initial (ISN) choisi par le client
  - l'émission du SYN incrémentera le futur numéro de séquence
  - pas de données
- **2** serveur → client : segment TCP avec les bits SYN + ACK
  - la réception du SYN à incrémenté le numéro de d'aquittement
  - indique le numéro de séquence initial (ISN) choisi par le serveur
  - l'émission du SYN incrémentera le futur numéro de séquence
  - allocation des tampons du serveur
- **3** client → serveur : segment TCP avec le bit ACK
  - la réception du SYN à incrémenté le numéro de d'aquittement
  - peut contenir des données

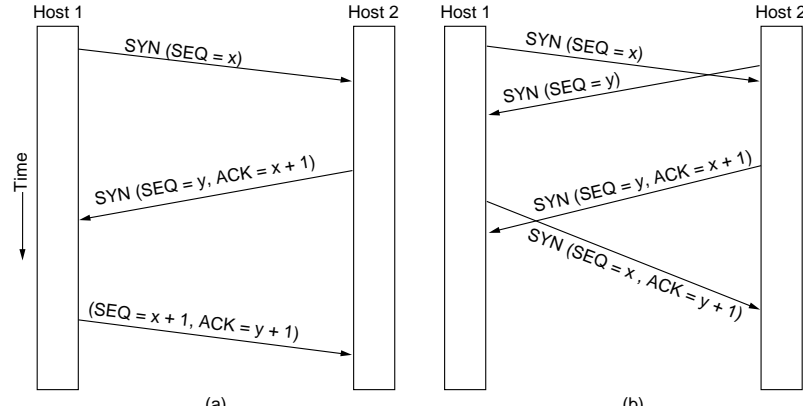
## TCP : Three-Way Handshake (2)

### Echange initial en 3 segments



## TCP : Three-Way Handshake (3)

### Gestion des ouvertures simultanées

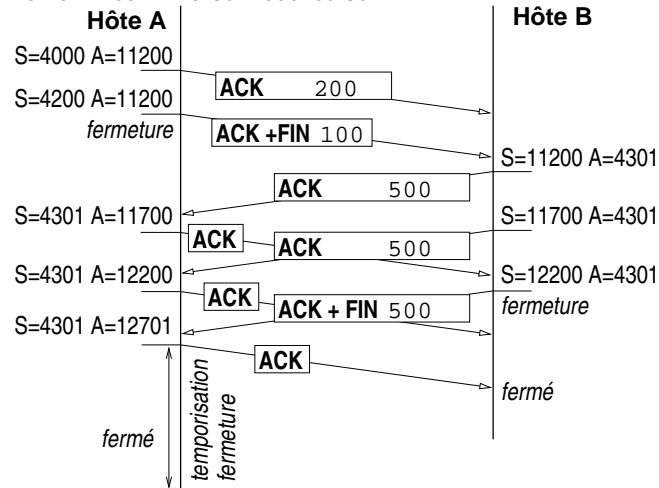


## TCP : Gracefull Release (1)

- **1** le **client** émet un segment TCP avec FIN
  - l'émission du FIN incrémentera le futur numéro de séquence
  - peut contenir des données
- **2** le **serveur** reçoit le segment avec FIN
  - la réception du FIN incrémente le numéro d'aquittement
  - émet un segment TCP avec ACK
  - termine la connexion (**envoie les données restantes**)
  - émet un segment TCP avec FIN
  - l'émission du FIN incrémentera le futur numéro de séquence
- **3** le **client** reçoit le segment avec FIN
  - la réception du FIN incrémente le numéro d'aquittement
  - émet un segment TCP avec ACK
  - termine la connexion
    - déclenche une temporisation d'attente (FIN dupliquées)
- **4** le **serveur** reçoit le segment avec FIN

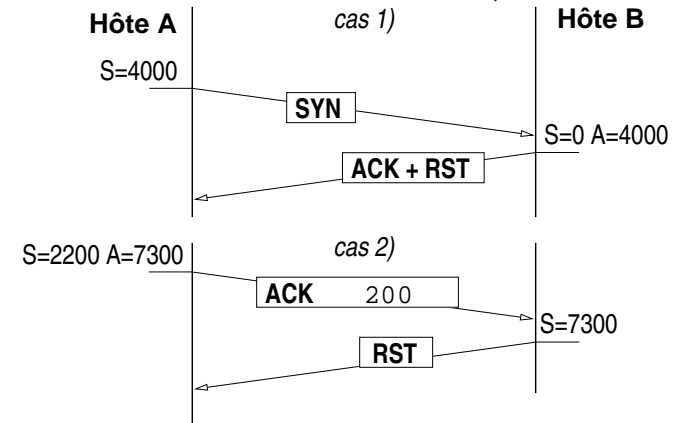
## TCP : Gracefull Release (2)

Déconnexion : terminaison courtoise

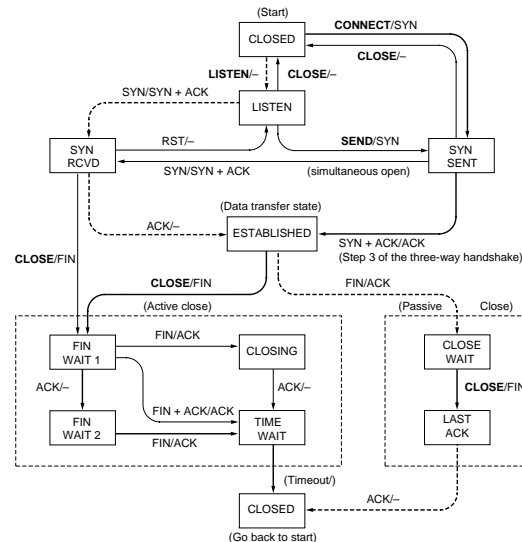


## TCP : Shutdown

Déconnexion : terminaison unilatérale  
(pour tout comportement anormal ou indésiré)



## TCP : Automate d'états finis



## ARES : Plan du cours 3/5

- 1 Service de base
  - Rappels sur la couche transport
  - Multiplexage et démultiplexage
  - UDP : un protocole en mode non connecté
- 2 Service fiable
  - Principes de transfert de données fiable
  - TCP : un protocole en mode orienté connexion
  - TCP : mécanismes de fiabilisation
- 3 Contrôle de congestion
  - Principes généraux
  - Mécanismes de TCP



## Transmission fiable de TCP

TCP est un protocole fiable de transfert sur le service IP non fiable

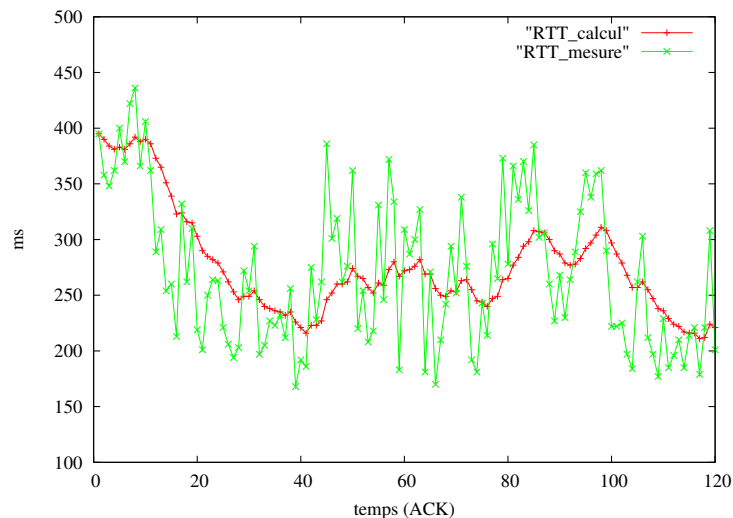
- mécanismes de base :
  - **pipeline**
  - **ACK cumulatifs**
  - temporisateur de retransmission **unique**
  - retransmissions déclenchées par :
    - expiration de temporisation (*timeout*)
    - duplication d'ACK
- dans la suite...
  - émetteur TCP simplifié :
    - pas d'ACK dupliqué
    - pas de contrôle de flux
    - pas de contrôle de congestion

## TCP : Calcul du RTT

### Round Trip Time

- Estimation de la temporisation de retransmission :
  - supérieure au RTT... mais le RTT varie !
    - trop petit : retransmissions inutiles
    - trop grand : réaction lente aux pertes
- Estimation du RTT :
  - $RTT_{mesure} = \Delta$  (envoi segment, réception ACK correspondant)
  - $RTT_{mesure}$  peut varier rapidement ➡ lissage
    - $RTT = \alpha RTT_{mesure} + (1 - \alpha) RTT_{ancien}$   
avec  $\alpha$  usuel = 1/8
  - moyenne glissante à décroissance exponentielle

## TCP : Exemple de calcul de RTT



## TCP : Temporisations

Gestion de multiples temporisations (*timers*) :

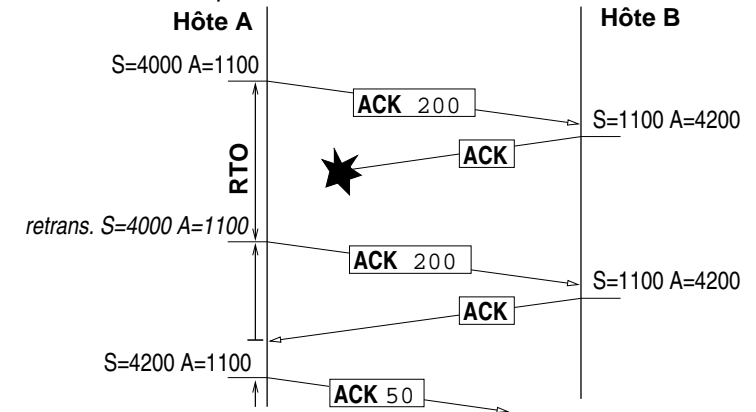
- *retransmission timer* (détecte les pertes)
  - $RTO = RTT + \delta D$ 
    - avec  $\delta = 4$  et une valeur initiale du  $RTT$  élevée (3 secondes)
  - $D = \beta(|RTT_{mesure} - RTT_{ancien}|) + (1 - \beta)D_{ancien}$ 
    - calcul de l'écart moyen avec  $\beta$  usuel = 1/4
  - **algorithme de Karn**
    - ne pas tenir compte des paquets retransmis et doubler le  $RTO$  à chaque échec (*exponential backoff*)
- *persistence timer* (évite les blocages)
  - envoi d'un acquittement avec une fenêtre à 0
- *keep alive timer* (vérifie s'il y a toujours un destinataire)
- *closing timer* (terminaison)

## Évènements émetteur TCP

- **réception des données de la couche supérieure**
  - **création** d'un segment avec numSeq
    - numSeq est le numéro dans le flux d'octet du premier octet de donnée du segment
  - démarrer la **temporisation** si elle n'est pas déjà en cours
    - la temporisation correspond au segment non acquitté le plus ancien
- **expiration de temporisation** (*timeout*)
  - **retransmission** du segment associé à la temporisation
  - redémarrer la **temporisation**
- **réception d'acquiescement** (ACK)
  - si acquiesce des segments non acquies :
    - actualiser la base de la fenêtre de transmission (*base\_emis*)
    - redémarrer la **temporisation** si d'autres ACK sont attendus

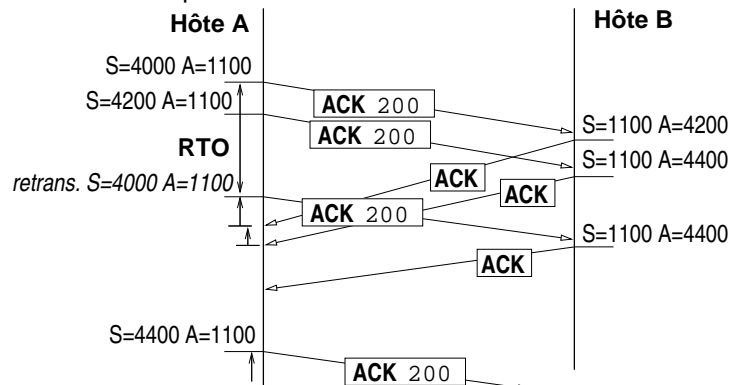
## Retransmission TCP (1)

Scénario avec ACK perdu



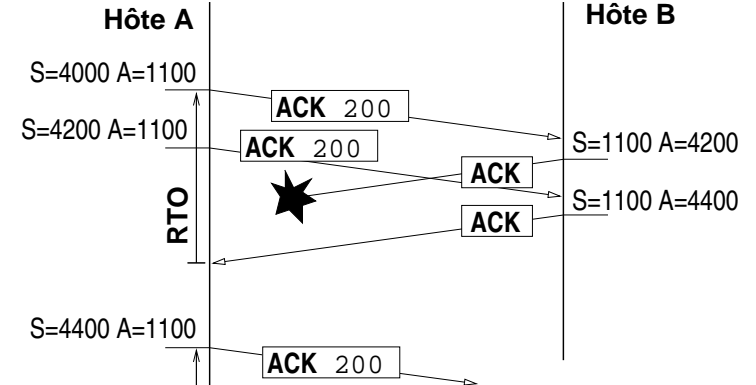
## Retransmission TCP (2)

Scénario avec temporisation sous-estimée



## Retransmission TCP (3)

Scénario avec ACK cumulatifs



## Évènement récepteur TCP

### Génération d'ACKs (actions du récepteur)

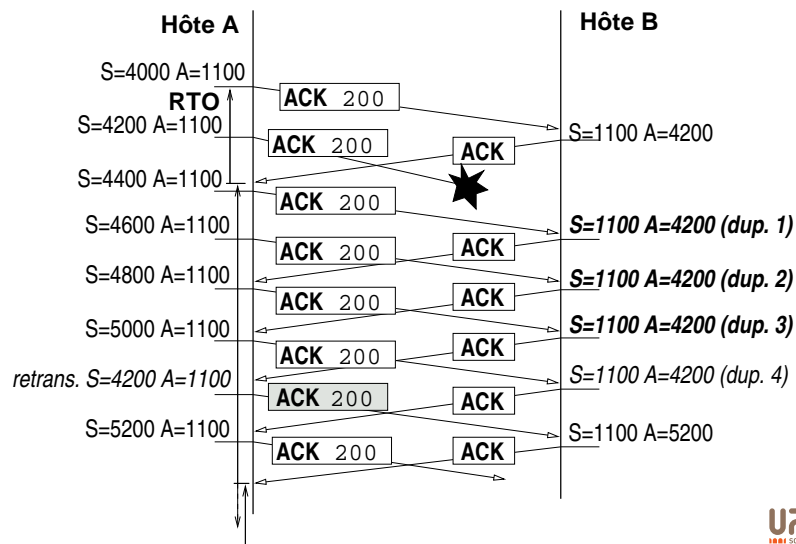
- arrivée d'un segment **dans l'ordre** avec le numSeq attendu :
  - les segments précédents sont déjà acquittés
    - ACK **retardé** (*delayed ACK*), attente jusqu'à 500 ms
    - si pas d'autre segments, envoi d'un ACK
  - un autre segment est en attente d'acquiescement
    - envoi immédiat d'un ACK **cumulatif** pour ces deux segments dans l'ordre
- arrivée d'un segment **dans le désordre** :
  - numSeq supérieur à celui attendu (intervalle détecté)
    - envoi immédiat d'un ACK **dupliqué**
    - rappel du prochain numSeq attendu
  - rempli partiellement ou totalement un intervalle
    - envoi immédiat d'un ACK
    - nouveau numSeq attendu suite au remplissage de l'intervalle

## TCP : Fast Retransmit (1)

### Optimisation du mécanisme de retransmission

- temporisation souvent relativement élevée
  - délai important avant une retransmission
- détection des segments perdus grâce aux ACKs **dupliqués**
  - ensemble de segments souvent envoyés côte-à-côte
  - si un segment est perdu ➡ nombreux ACKs dupliqués
- si l'émetteur reçoit 3 ACK dupliqués (4 ACKs identiques)
  - TCP suppose que le segment suivant celui acquité est perdu
    - **fast retransmit** : retransmission du segment avant l'expiration de la temporisation

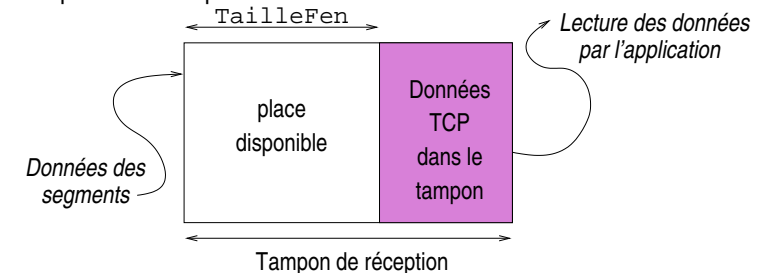
## TCP : Fast Retransmit (2)



## TCP : Asservissement au récepteur

### contrôle de flux

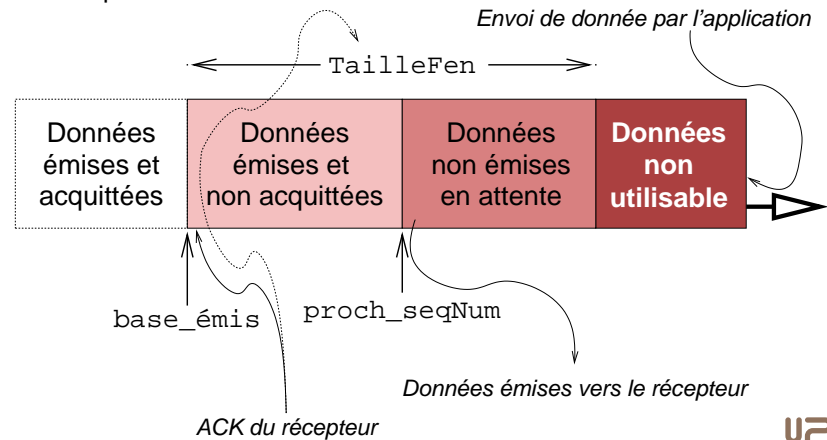
- l'émetteur ne doit pas dépasser les capacités du récepteur
- récupération de la taille de la place disponible du tampon de réception du récepteur :



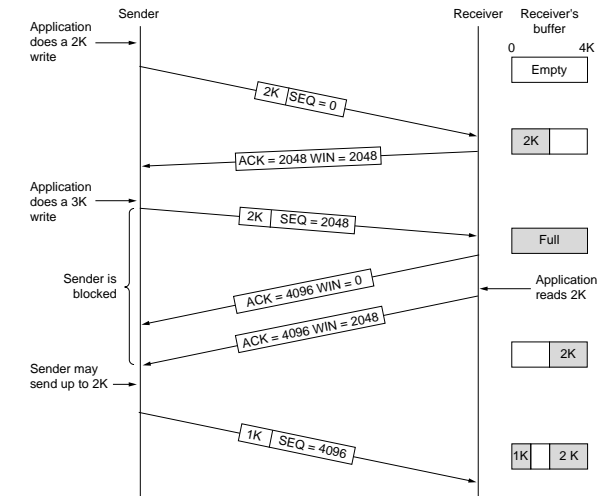
- $\text{TailleFen} = \text{TailleTampon} - \text{DernierOctetRecu} + \text{DernierOctetLu}$

## TCP : Limitation de l'émetteur

*Sliding window* : l'émetteur limite la transmission de ses données non acquittées



## TCP : Contrôle de flux



pictures from TANENBAUM A. S. *Computer Networks 3rd edition*

## TCP : temporisation de ré-ouverture de la fenêtre

### Persistence timer

- évite que la taille de la fenêtre reste à 0
  - possible si perte du ACK annonçant une fenêtre non nulle
  - évité grâce à l'envoi d'un paquet sonde après une temporisation
    - tempo. initiée à RTT puis double à chaque expiration jusqu'à 60s (puis reste à 60s)
    - le paquet sonde est un segment avec 1 octet de données

## TCP : Optimisation du contrôle de flux

### Send-side silly window syndrome

- Algorithme de Nagle (RFC 896)
  - agrégation de petits paquets (*nagling*)
  - attente d'un acquittement ou d'un MSS avant d'envoyer un segment
    - TELNET : évite d'envoyer un paquet par caractère tapé
    - désactivable avec l'option TCP\_NODELAY des sockets

### Receiver silly window syndrome

- Algorithme de Clark
- limiter les annonces de fenêtre trop petites
  - fermeture de la fenêtre en attendant d'avoir suffisamment de place pour un segment complet

## TCP : exemples d'applications

Les applications suivantes reposent typiquement sur TCP :

- connexion à distance (TELNET, rlogin et ssh)
- transfert de fichiers (FTP, rcp, scp et sftp)
- protocole de routage externe (BGP)
- messageries instantanées (IRC, ICQ, AIM...)
- web (HTTP)
  - nouvelles applications utilisent HTTP comme service d'accès au réseau
  - permet de passer les *firewalls*



## TCP : Interface socket

```
#include <sys/types.h>
#include <sys/socket.h>

# create a descriptor and bind local IP and port
int socket(int domain, int type, int protocol);
# domain : PF_INET for IPv4 Internet Protocols
# type : SOCK_STREAM Provides sequenced, reliable, 2-way, connection-based byte streams.
# An out-of-band data transmission mechanism may be supported.
# protocol : TCP (/etc/protocols)
int bind(int s, struct sockaddr *my_addr, socklen_t addrlen);

# Server : passive queuing mode and connection acceptance
int listen(int s, int backlog);
int accept(int s, struct sockaddr *addr, socklen_t *addrlen);

# Client : active connection
int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);

# Send and receive data
int send(int s, const void *msg, size_t len, int flags);
int recv(int s, void *buf, size_t len, int flags);

# End : deallocate
int close(int s);
```



## TCP : utilisation spécifiques

TCP doit s'adapter à des flots de qqs **bps** à plusieurs **Gbps** :

- LFN (*Long Fat Network*)
  - capacité du réseau = **bande passante \* délai de propagation**
    - limitation de taille de la fenêtre (option WSIZE, jusqu'à un facteur  $2^{14}$ )
    - rebouclage des numéros de séquence (PAWS, *Protect Against Wrapped Sequence*, utilise l'option TIMESTAMP)
    - acquittements sélectifs pour éviter des retransmissions importantes inutiles (option SACK)
  - satellites
  - fibres transatlantiques
- réseaux asymétriques (ADSL, Cable)
  - sous-utilisation du lien rapide



## ARES : Plan du cours 3/5

- 1 Service de base
  - Rappels sur la couche transport
  - Multiplexage et démultiplexage
  - UDP : un protocole en mode non connecté
- 2 Service fiable
  - Principes de transfert de données fiable
  - TCP : un protocole en mode orienté connexion
  - TCP : mécanismes de fiabilisation
- 3 Contrôle de congestion
  - Principes généraux
  - Mécanismes de TCP

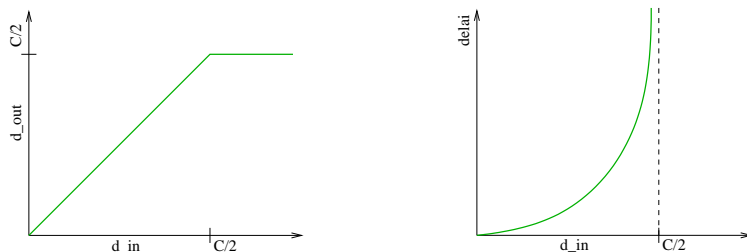


## Contrôle de congestion

## Congestion

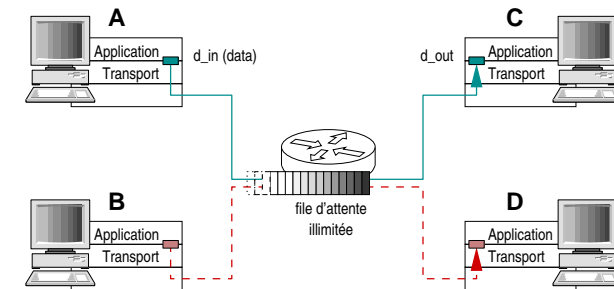
- trop de flots de données saturent un ou plusieurs éléments du réseau
- différent du contrôle de flux
  - TCP n'a pas accès à l'intérieur du réseau
- manifestation :
  - longs délais
    - attente dans les tampons des routeurs
  - pertes de paquets
    - saturation des tampons des routeurs

## Congestion : scénario 1b



- **coût de la congestion :**
  - débit maximum atteignable
    - $d_{in} = C/2$
  - délai très élevé proche du maximum
    - croissance infinie des tampons

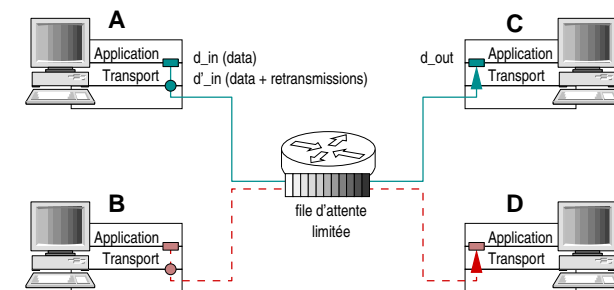
## Congestion : scénario 1a



- 2 émetteurs, 2 récepteurs
- 1 routeur
  - tampons infinis
- pas de retransmission

➡ Que ce passe-t-il quand  $d_{in}$  augmente ?

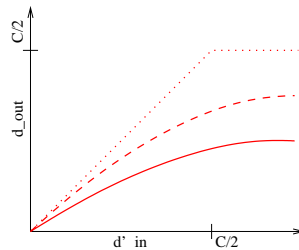
## Congestion : scénario 2a



- 2 émetteurs, 2 récepteurs
- 1 routeur
  - **tampons finis**
- **retransmission** des segments perdus

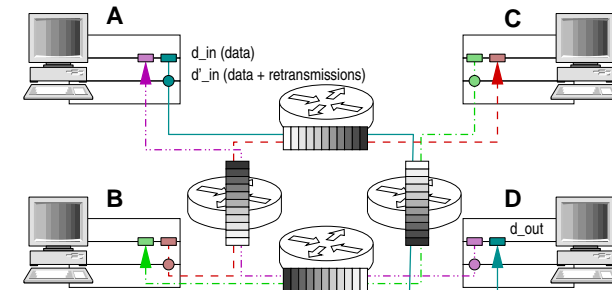
➡ Que ce passe-t-il quand  $d'_{in}$  augmente ?

## Congestion : scénario 2b



- toujours  $d_{in} = d_{out}$  (*goodput*)
- coût des retransmissions
  - **retransmissions utiles** : seulement pour des pertes
    - $d'_{in}$  supérieur à  $d_{out}$
  - **retransmissions inutiles** : segments en retard
    - $d'_{in}$  encore plus supérieur à  $d_{out}$
- **coût de la congestion** :
  - beaucoup plus de trafic pour un  $d_{out}$  donné
  - duplications de segment inutile

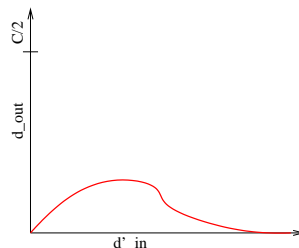
## Congestion : scénario 3a



- 4 émetteurs, 4 récepteurs
- 4 routeurs
  - chemins multi-saut
  - tampons finis
- retransmission

➡ Que ce passe-t-il quand  $d'_{in}$  augmente ?

## Congestion : scénario 3b



- **coût supplémentaire de la congestion** :
  - lors de la perte d'un paquet, toute la capacité amont est gachée

## Contrôle de congestion

Deux approches :

- contrôle de congestion géré par le **réseau**
  - les routeurs informent les extrémités
    - bit d'indication de la congestion (SNA, DECbit, ATM, TCP/IP ECN...)
    - indication explicite du débit disponible (ATM ABR, TCP/IP RSVP + IntServ...)
- contrôle de congestion aux **extrémités** (*end-to-end*)
  - aucune indication explicite du réseau
  - **inférence** à partir des observations faites aux extrémités
    - pertes
    - délais
- **approche choisie dans TCP**

## ARES : Plan du cours 3/5

- ① Service de base
  - Rappels sur la couche transport
  - Multiplexage et démultiplexage
  - UDP : un protocole en mode non connecté
- ② Service fiable
  - Principes de transfert de données fiable
  - TCP : un protocole en mode orienté connexion
  - TCP : mécanismes de fiabilisation
- ③ Contrôle de congestion
  - Principes généraux
  - Mécanismes de TCP

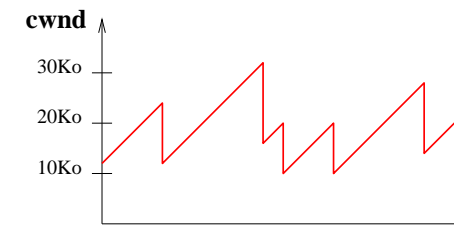
## TCP : Contrôle de congestion

- basé sur la limitation de l'émission de l'émetteur
  - $\text{dernierOctetEmis} - \text{dernierOctetAcq} \leq \text{cwnd}$
  - approximation du débit :
    - $d_{TCP} = \frac{\text{cwnd}}{RTT}$
- $\text{cwnd}$  = fonction dynamique de la congestion perçue
  - perception de la congestion par le récepteur :
    - expiration de temporisation (RTO)
    - triple ACK dupliqués
  - 3 mécanismes :
    - AIMD
    - *Slow Start*
    - prudence après l'expiration d'une temporisation

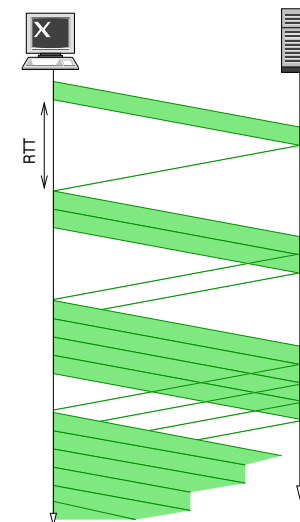
## TCP : Contrôle AIMD

*Additive Increase, Multiplicative Decrease*

- augmentation progressive du débit de transmission ( $\text{cwnd}$ ) tant qu'il n'y a pas de perte
  - *Additive Increase*
    - augmenter  $\text{cwnd}$  de 1 MSS à chaque RTT tant qu'il n'y a pas de perte détectée
  - *Multiplicative Decrease*
    - diviser  $\text{cwnd}$  par 2 après une perte
- comportement en dent de scie :



## TCP : Slow Start

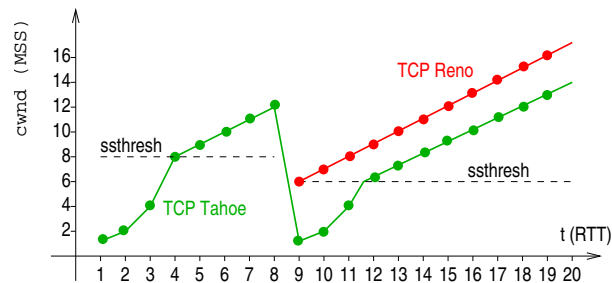
Démarré lentement (*slow start*)

mais croît très vite !!

- au démarrage de la connexion
  - $\text{cwnd} = 2 \text{ à } 4 \text{ MSS}$
- au redémarrage (après perte ou inactivité)
  - $\text{cwnd} = 1 \text{ MSS}$  ( $d_{init} = \frac{\text{MSS}}{RTT}$ )
- puis croissance exponentielle jusqu'à la première perte
  - $\text{cwnd}$  double / RTT
  - implémenté par :  $\text{cwnd}++$  / ACK
- $d_{potentiel} \gg \frac{\text{MSS}}{RTT}$



## TCP : Optimisation



Passage de la croissance exponentielle à linéaire

- $cwnd \geq$  ancienne valeur de  $cwnd$  juste avant la perte
  - implémenté par une limite variable :  
 $ssthresh = cwnd_{avant\ la\ dernière\ perte} / 2$
  - plus précisément calculé avec les segments non acquittés :  
 $ssthresh = qtt\_données\_non\_acquit / 2$  (ou  $flightsize/2$ )

## Contrôle de congestion TCP : synthèse

## RFC 5681

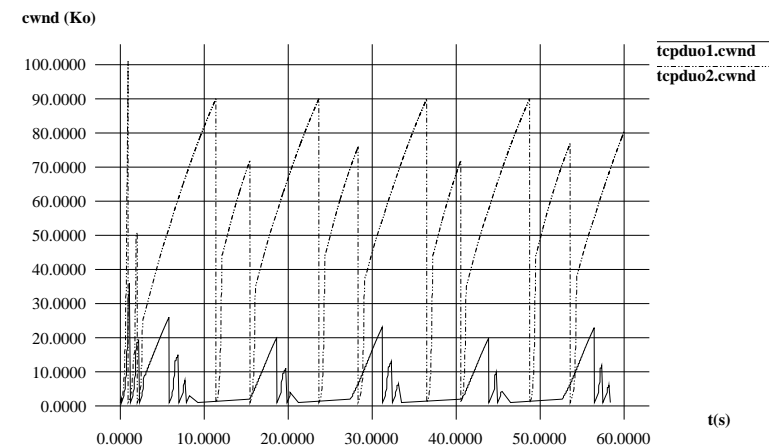
- quand  $cwnd < ssthresh$  :
  - émetteur en phase *Slow Start*
  - $cwnd$  croît exponentiellement
- quand  $cwnd \geq ssthresh$  :
  - émetteur en phase *Congestion Avoidance*
  - $cwnd$  croît linéairement
- quand 3 ACK dupliqués apparaissent :
  - $ssthresh = dernière\ cwnd / 2$
  - $cwnd = ssthresh$
- quand la temporisation expire :
  - $ssthresh = dernière\ cwnd / 2$
  - $cwnd = 1\ MSS$

## TCP : Inférence des pertes

Les ACK dupliqués sont moins graves que les expirations de temporisation

- suite **3 ACK dupliqués** :
  - indique que le réseau continue à transmettre des segments
    - $cwnd$  divisé par 2
    - $cwnd$  croît ensuite linéairement
- suite **expiration temporisation** :
  - indique que le réseau se bloque
    - $cwnd = 1\ MSS$
    - *Slow Start* (croissance exponentielle)
    - à  $ssthresh = cwnd/2$  (croissance linéaire)

## TCP : équité entre flots ?



- oscillation de deux flots en phase de congestion

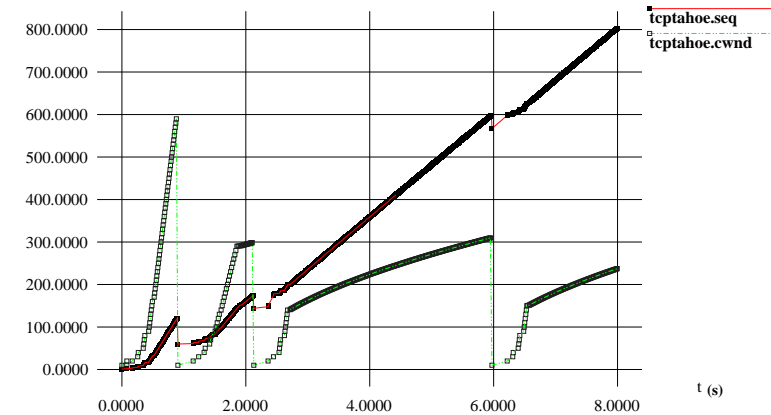
## Implémentations

A trip to Nevada :

- **TCP Tahoe 1988**
  - *slow start + congestion avoidance + multiplicative decrease*
  - **fast retransmit** (déclenche la retransmission d'un segment après 3 acquit. dupliqués, avant l'expiration de la tempo.)
  - décrit précédemment... pb lorsque seulement 1 seg. est perdu
- **TCP Reno 1990 (RFC 2582)**
  - idem TCP Tahoe
  - **fast recovery** (pas de *slow start* après un *fast retransmit*)
- **TCP newReno 1996 (RFC 3782)**
  - idem TCP Reno
  - pas de *slow start* à la première congestion et ajustement de *cwnd*
  - SACK (RFC 2018)

## TCP : Tahoe

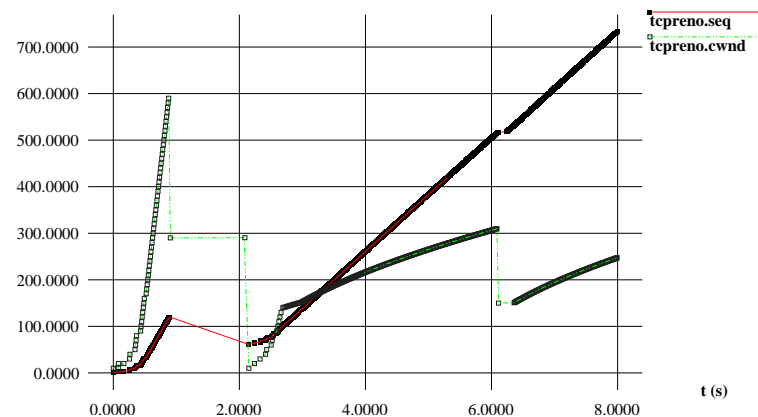
seq (Ko) / cwnd (Ko/10)



- *slow start + congestion avoidance + multiplicative decrease + fast retransmit*

## TCP : Reno

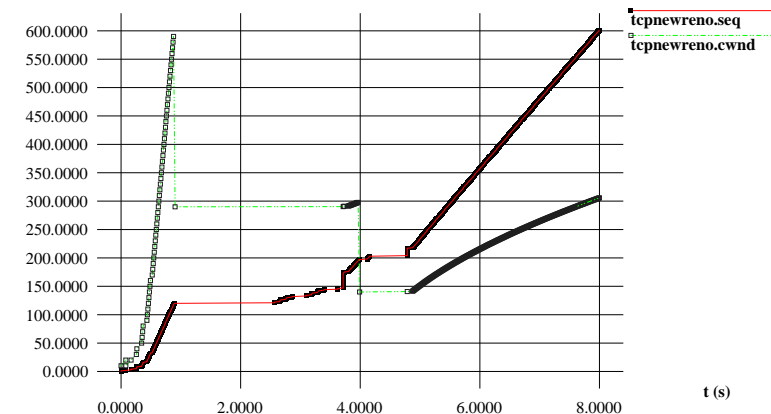
seq (Ko) / cwnd (Ko/10)



- *fast recovery* (pas de *slow start* après un *fast retransmit*)

## TCP : newReno

seq (Ko) / cwnd (Ko/10)



- pas de *slow start* à la première congestion et ajustement de *cwnd*