Les protocoles de transport dans INTERNET : TCP et UDP

Session de Remise à Niveau, Télécom Paris

Août-septembre 2021

Ken CHEN

ken.chen@univ-paris13.fr

Programme

+ La couche transport dans Internet
+ UDP
+ TCP
+ TCP et contrôle de congestion

La couche transport

+ La couche transport : contrôle de bout-en-bout
+ Niveau 3 de Internet = IP : datagram
+ Besoin d'un service fiable : TCP
+ Besoin d'un service datagram : UDP
+ La couche transport de INTERNET
  - TCP : communications orientées connexion
  - UDP : communications sans connexion

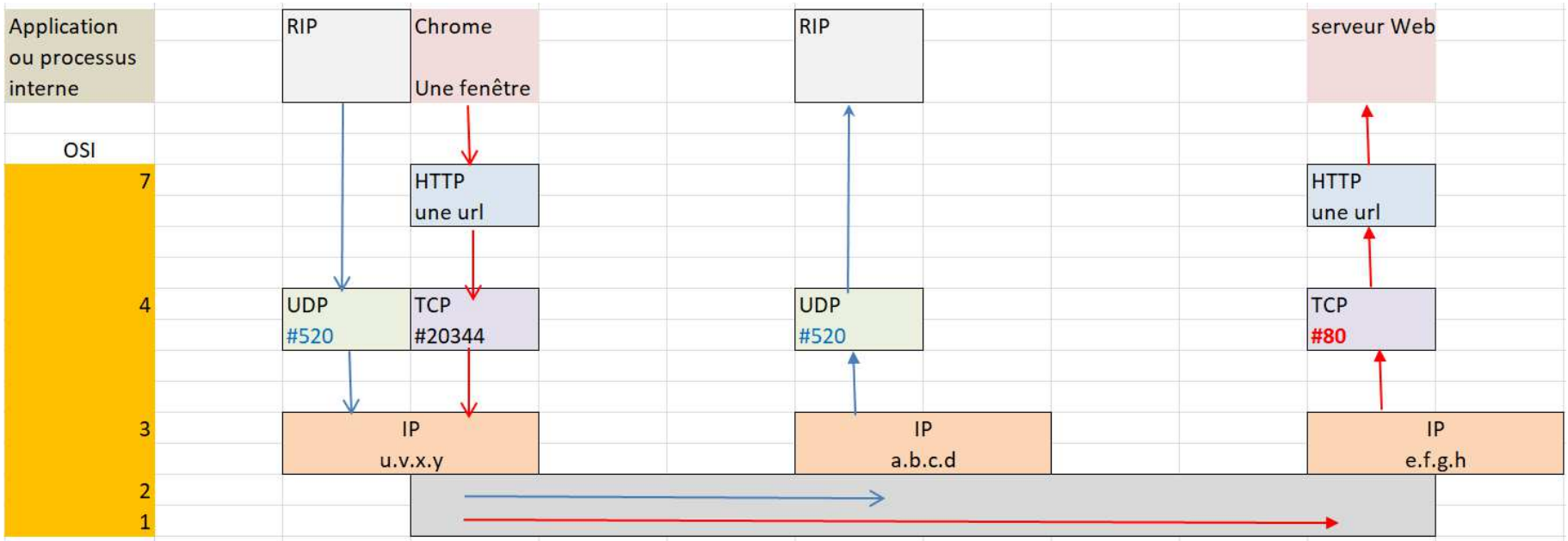
Identification d'une communication

+ Sur un même **terminal** (smart phone, PC, etc.), il y de nombreux processus communicants (Chrome, Edge, etc.)
  - Ils co-existent (1 fenêtre Chrome, 2 fenêtres Edge)
  - Ils communiquent avec différentes destinations
  - De nombreuses **sessions** de communications se déroulent en **parallèle**
+ Chaque session de communications est identifiée par un numéro de **port**
  - Une application (type une fenêtre Web)
  - Un processus interne

Identification d'une communication

+ Une **communication** Internet est **entièrement identifiée** par le **5-tuples**
  - **<port source, adresse source, protocole, port destination, adresse destination>**
    + Appli/processus côté émetteur
    + Appli/processus côté émetteur
    + Le protocole de la couche transport
+ Toute communication Internet : **unidirectionnelle**
+ Certains ports sont pré-définis (***Well-Known Port***)
  - Quelques exemples :
    + 20 : TCP/FTP Port de donnée ; 21 : TCP/FTP Port de contrôle
    + 23 : TCP/Telnet
    + 25 : TCP/SMTP
    + 80 : TCP-UDP/ HTTP

Identification d'une communication



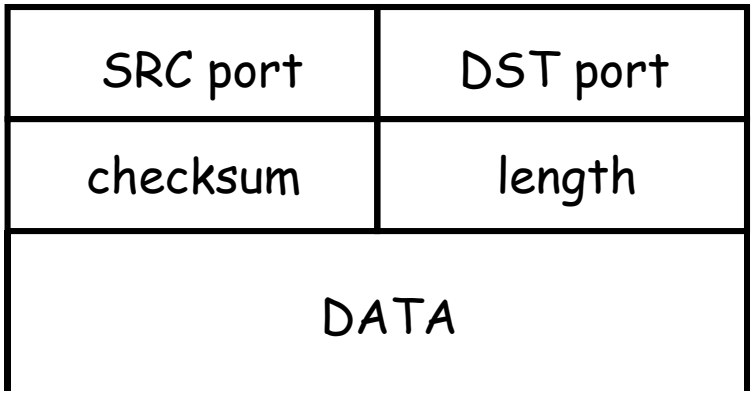
+ Une communication entre deux processus RIP (*hypothètiquement*, on suppose que l'équipement u.v.x.y se charge d'un rôle de routeur..)
+ Une communication entre un terminal et un serveur

UDP (1/2)

+ User Data Protocol (RFC 768)
+ Service datagramme = sans connexion
  - Transfert non fiabilisé
+ Convient à certaines applications qui n'ont pas à gérer une connexion (appli. transactionnelles) ou des applications (multimédia en particulière) qui ont leurs propres logiques et gestionnaires de fiabilité
  - **Attention** : le ***streaming video*** est géré, très majoritairement, par-dessus HTTP/TCP (norme **DASH**)

UDP (2/2)

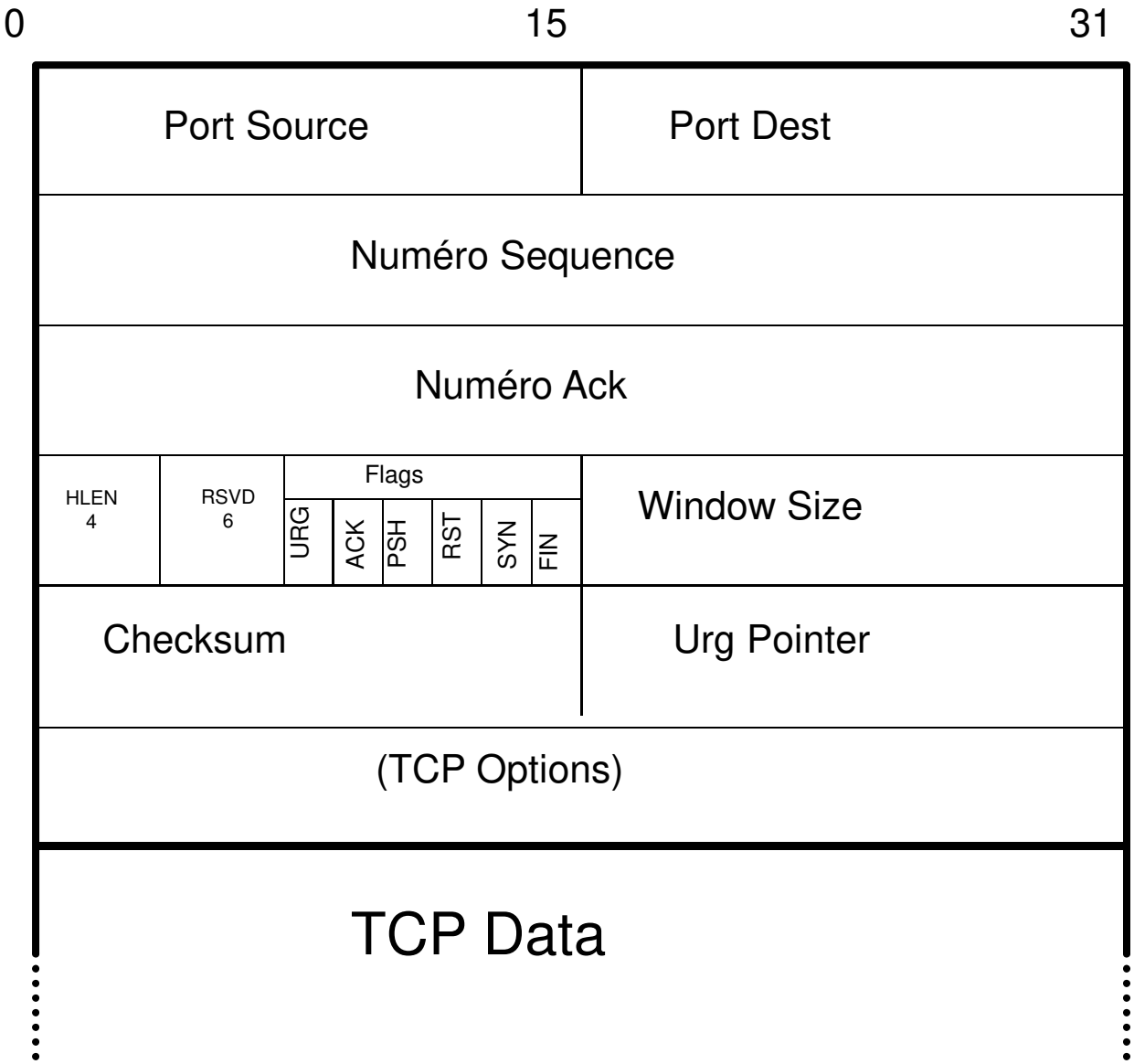
+ Le format de UDP est très simple. Le chechsum couvre l'entête



TCP

+ TCP = Transport Control Protocol (RFC 793)
+ Service orienté connexion (bidirectionnelle)
  - Garantie d'un transfert fiable de données
+ Connexion = flux d'**octets**
  - Flux divisés en **segments**
+ Fiabilité acquise par ACK
+ Perte détectée par TimeOut
+ Contrôle de flux
+ Contrôle de congestion

Format TCP


0
15
31
Port Source
Port Dest
Numéro Sequence
Numéro Ack
HLEN
4
RSVD
6
Flags
URG
ACK
PSH
RST
SYN
FIN
Window Size
Checksum
Urg Pointer
(TCP Options)
TCP Data

Entête TCP

+ Port Source/Dest : 16 bits chacun
+ Numéro séquence **:** Numéro de séquence du premier octet du segment
+ Numéro ACK : prochain octet attendu
+ HLEN : taille de l'entête en unité de 32 bits
+ Les 6 drapeaux (cf transparent suivant)
+ Window size : taille de la fenêtre d'anticipation
+ Checksum : protection de l'entête
+ Urgent pointeur : cf drapeau « URG »

En-tête TCP : les drapeaux

+ SYN= 1: l'établissement de connexion
+ FIN =1 : libération de connexion dans un sens
+ ACK =1 : validation du numéro ACK
+ RST = 1 : re-synchronisation
+ PSH (push) =1 : demande au récepteur de remettre immédiatement les données
+ URG =1 : valide le Urgent Pointer (UP), le récepteur doit remettre les données jusqu'à UP (dernier octet contenant des données urgentes)

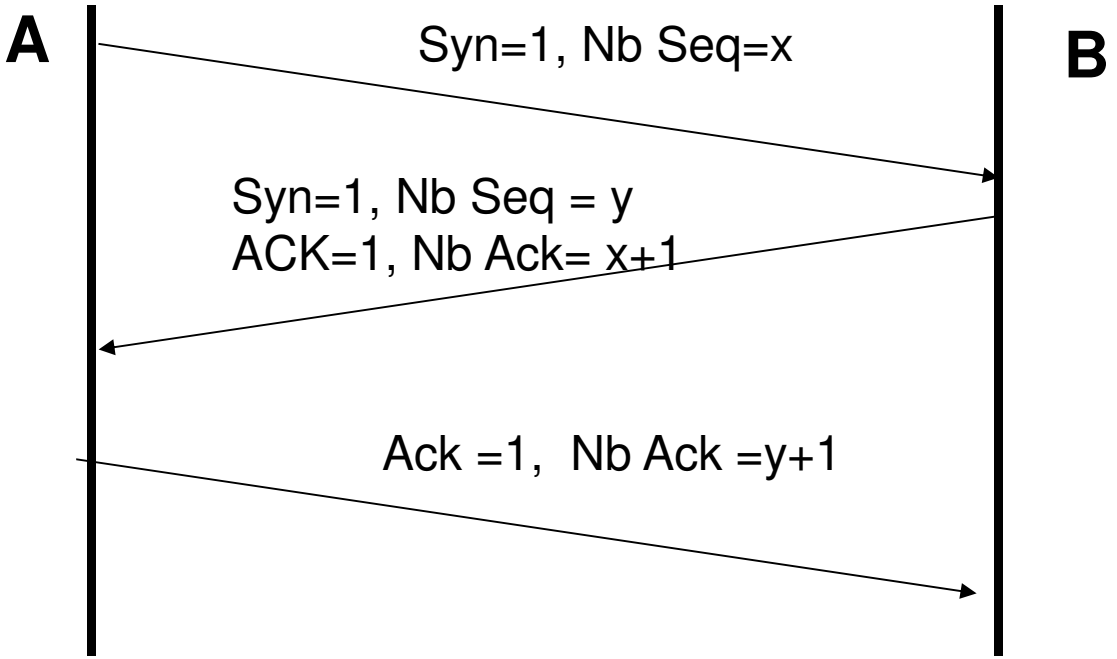
En-tête TCP

No.	Time	Length	Source	Destination	Protocol	SrcPort	DestPort	SrcPortNI
4	0.911310	533	dialin-145-254-160-237.pools.arcor-ip…	65.208.228.223	HTTP	tip2	http	33

```
Transmission Control Protocol, Src Port: tip2 (3372), Dst Port: http (80), Seq: 1, Ack: 1, Len: 479
    Source Port: tip2 (3372)
    Destination Port: http (80)
    [Stream index: 0]
    [TCP Segment Len: 479]
    Sequence number: 1    (relative sequence number)
    Sequence number (raw): 951057940
    [Next sequence number: 480    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    Acknowledgment number (raw): 290218380
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window size value: 9660
    [Calculated window size: 9660]
    [Window size scaling factor: -2 (no window scaling used)]
    Checksum: 0xa958 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  > [SEQ/ACK analysis]
  > [Timestamps]
    TCP payload (479 bytes)
```

Connexion (1/2)

+ Procédure en 3 étapes (*3-way handshake*)
+ Scénario : A demande une connexion avec B



Connexion (2/2)

+ A envoie un 1er segment sans data avec numéro de séquence = X (choisi **aléatoirement**)
  - Raison d'être : éviter de « tomber » sur une connexion existante
+ B accepte la connexion A->B avec ACK=1 et Numéro ACK = x+1
  - Comme si A avait envoyé un octet de donnée
+ B utilise le même segment pour établir la connexion dans le sens B->A avec n. séquence y
+ A accepte la connexion avec ACK=1 et numéro ACK = y+1

Libération

+ Procédure par sens de connexion en 2 étapes
+ A envoie un segment avec FIN=1 pour fermer la connexion A-> B, B répond avec un ACK
+ A ne peut alors plus envoyer des données à B!
+ Plus tard, B effectue le même type d'opération
  - La connexion B-> A est close également
  - B ne peut plus envoyer des données à A!
+ La connexion TCP entre A et B est alors définitivement close.

+ *Remarque: TCP=Full duplex*

Fiabilisation de transfert

+ Connexion orientée connexion
  - Segments livrés dans l'ordre, sans perte ni double
+ Numéro de séquence : permet la détection des anomalie
+ Acquittement positif
+ En cas de perte :
  - Détection via TimeOut
  - *Retransmission GoBack-N*
+ Principe similaire à celui de HDLC
+ Mise en œuvre plus complexe car « lien » beaucoup plus complexe (« lien » à travers Internet)

TimeOut

+ Acquittement positif
  - pas de message REJ comme pour HDLC
+ Détection de perte via l'absence de ACK
  - Un timer TimeOut (RTO) pour chaque segment
+ Quel horizon pour conclure à une perte ?
  - Un compromis : ni trop prompt ni trop lent
  - Un défi : délai variable dans Internet
+ Estimation de RTT pour estimer le RTO
  - RTT = Round Trip Time = temps d'aller-retour

Estimation RTT et RTO (1/2)

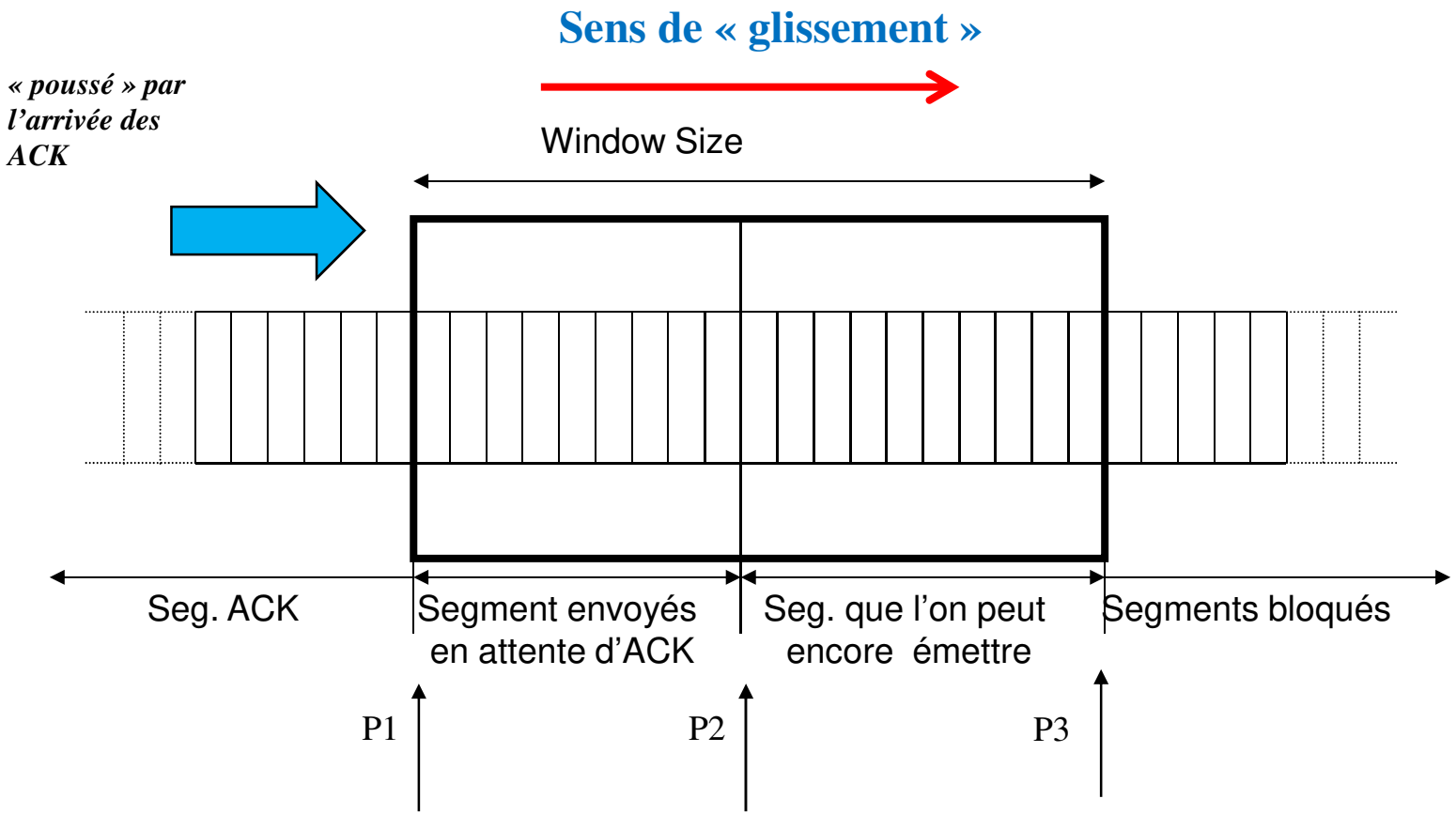
+ Algorithme de Van Jacobson
+ Variables :
  - RTTs = RTT courant (sampled)
    + Mesuré avec l'arrivée de chaque ACK)
  - RTTe = RTT estimé
  - DEV = estimation courante de la variance
+ Initialisation
  - RTTe = RTTs + 500 ms
  - DEV = RTTe /2

Estimation RTT et RTO (2/2)

+ Evolution (à chaque RTTs, c-à-d chaque ACK)
  - Err = RTTs - RTTe
  - RTTe = RTTe + Err/8
  - Dev = Dev + ¼ (|Err| - Dev)
+ RTO = RTTe + 4 Dev
+ Remarque :
  - RTTe=(1-1/8) RTTe + RTTs/8 (Filtre passe-bas)
  - Dev = ¾ Dev + ¼ |Err|

Fenêtre d'anticipation (1/2)

+ Fenêtre d'anticipation = fenêtre glissante (*sliding window*)
+ Avancé (glissement activé) par l'arrivé de ACK



Fenêtre d'anticipation (2/2)

+ Trois repères
  - P1 : Premier segment en attente d'ACK
  - P2 : Premier segment prêt pour l'émission
  - P3 : Premier segment/octets bloqué
+ Taille fenêtre F évolue dynamiquement en fonction de
  - Capacité d'accueil du récepteur indiqué par le champ « window size »
  - Contrôle de congestion (cwnd)
  - F = min (cwnd, w)

Contrôle de congestion

+ Problème
  - Perte : cause principale = Congestion
    + charge élevée sur une partie du chemin
  - En cas de perte : retransmission => débit réel plus élevé que le débit de l'appli.
  - Risque d'amplification de phénomène
+ Remède
  - Utiliser Perte comme signal de congestion
  - Adapter le débit en fonction des pertes

Contrôle de congestion

+ Algorithmes de base (TCP Tahoe, 1988)
  - Démarrage lent (slow start)
  - Evitement de congestion (congestion avoidance)
  - Retransmission rapide (Fast retransmission)
+ Variables
  - Cwnd = congestion window = nb de segments que l'on peut transmettre par antcipation
  - Awnd = fenêtre annoncée par le récepteur
  - ssthresh = seuil de basculement
+ Action sur fenêtre glissante
  - Rappel : F = min (cwnd, window size)

Slow start

+ Appliquer au début d'une connexion
+ Idée : au démarrage, on ne connaît pas la situation et on démarre doucement
+ Cwnd est initialisé à 1 puis est augmenté d'un avec chaque ACK
  - Chaque ACK donne lieu à 2 ACK
  - Croissance exponentiel, cwnd*2 par cycle de RTT
+ Slow start s'arrête si l'on atteint un seuil de basculement ssthresh, on applique alors « congestion avoidance »

Congestion avoidance

+ Slow start = croissance exponentielle
  - Trop rapide à partir d'une certaine taille
+ Algo congestion avoidance :
  - A partir de ssthresh, **croissance linéaire**
  - Chaque ACK : cwnd = cwnd + 1/cwnd
  - Par cycle RTT : cwnd = cwnd +1

Fast retransmission

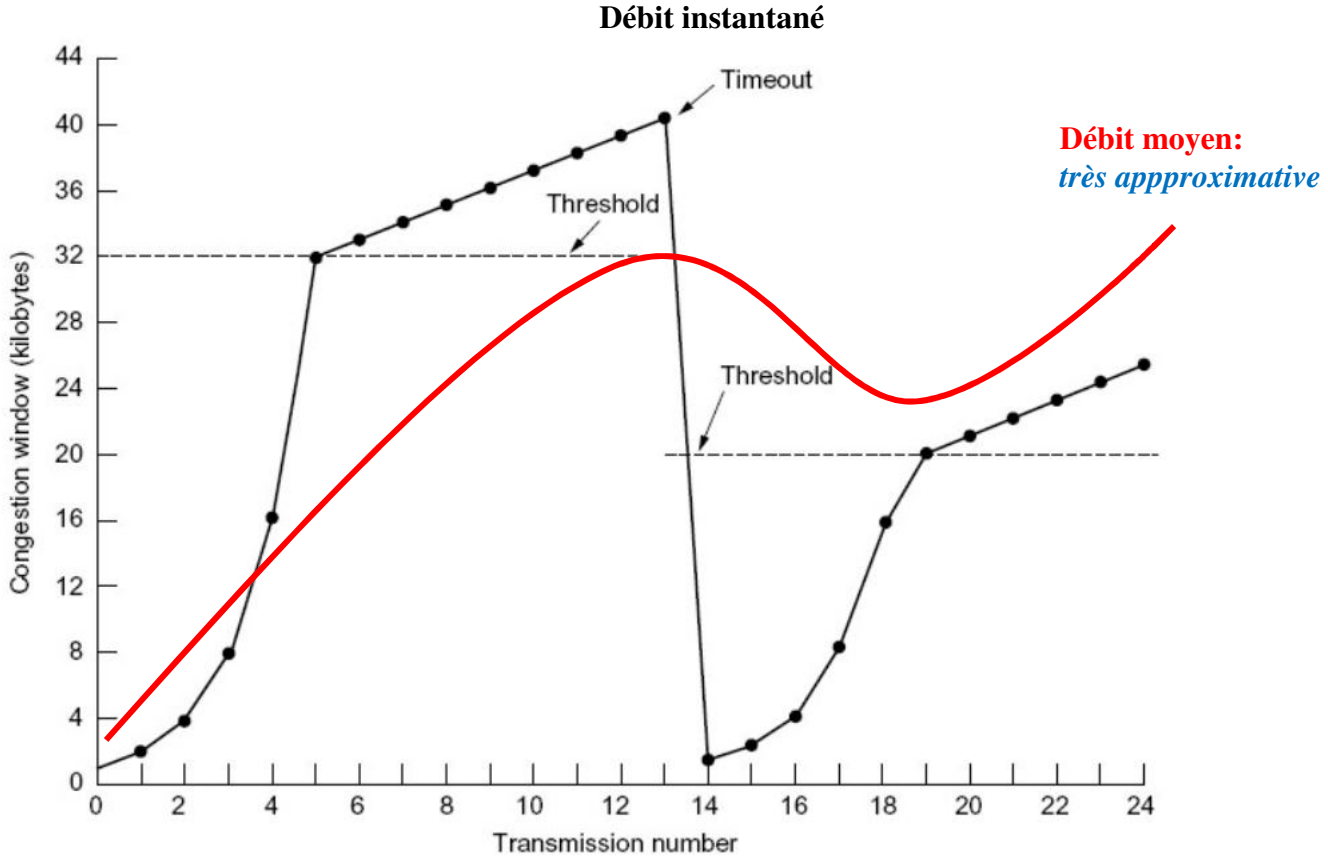
+ Perte d'un segment = détection avec RTO
+ Assez lent
+ Récepteur envoie ACK pour segments suivants
  - Ces ACK ne sont pas celui **attendu**
+ Idée de « Fast retransmission »:
  - Réception de 3 ACK non attendu est assimilée au signalement de la perte du segment dont le ACK est attendu
  - Retransmission du segment plus tôt que le RTO

En cas de perte

+ Perte = signal de congestion
  - Perte = RTO
  - Perte = 3 ACK dup (Fast Retransmission)
+ Calcul ssthresh
  - Ssthresh = fenetre actuelle /2
+ Recommencer avec slow start

Effet combiné

Trafic en « dent-de-scie »
TRES sensible aux pertes



Débit moyen:
très appproximative

Point-à-Point vs bout-en-bout

+ Communication orienté connexion
  - « liaison » entre deux **entités logiques (protocoles)**
  - niveau 2 ou 4 (ou niveau applicatif)
+ Niveau 2 : point-à-point
  - **Liaison** physique dont les caractéristiques sont (quasiment) figées (Ethernet) ou au moins connues (Wifi)
    + Souvent de courte distance
  - Exemple : HDLC
+ Niveau 4 : bout-en-bout
  - Souvent, les caractéristiques de la « **liaison** » sont fluctuante
  - Peut emprunter un long parcours
  - Exemple : TCP