heig-vd

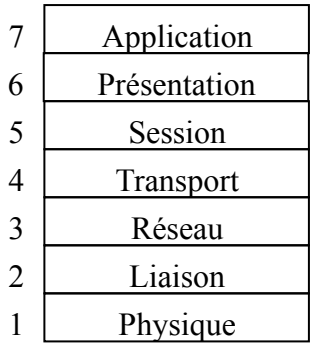
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Chapitre VII

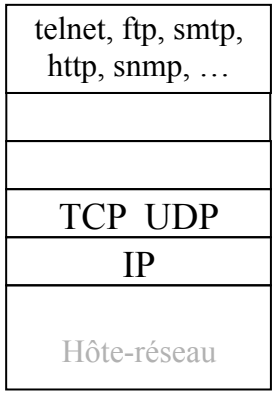
La couche transport

Rappel : la couche Réseau

- Fonctionnalités de la couche Réseau / IP
  - Adressage des nœuds du réseaux
  - Routage et acheminement
  - Interconnexion de réseaux hétérogènes
    - Fragmentation des paquets
    - Conversion d'adresses IP -> MAC
- Service offert à la couche Transport : « Best Effort »
  - Paquets perdus
  - Paquets ré-ordonnés
  - Paquets dupliqués
  - Limite la taille des paquets

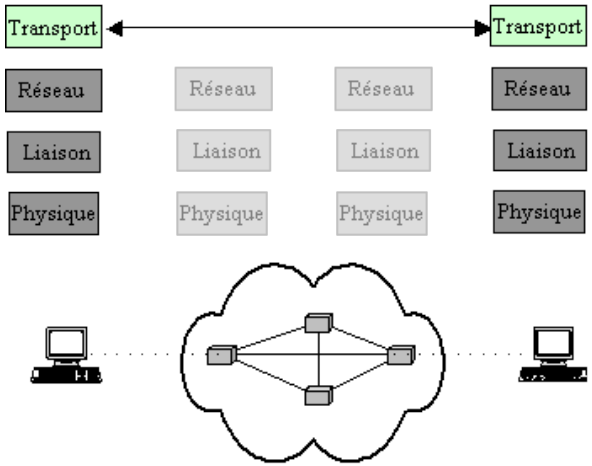


OSI



TCP/IP

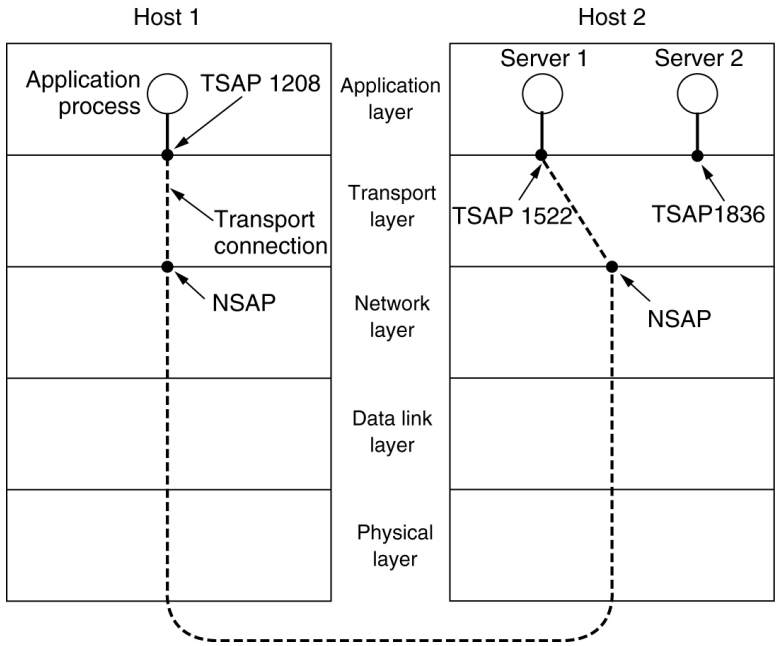
La couche Transport



- **Communication de bout en bout**
- Services offerts à la couche supérieure
  - Service sans connexion
  - Service orienté connexion
    - Fiable (garantie de délivrance)
    - Délivrance dans l'ordre d'émission
  - Transmission de messages de longueur arbitraire
- Éléments d'un protocole de transport
  1. Adressage
  2. Établissement et terminaison de connexions
  3. Transmission fiable (séquencement, retransmission, …)
  4. Contrôle de flux et de congestion

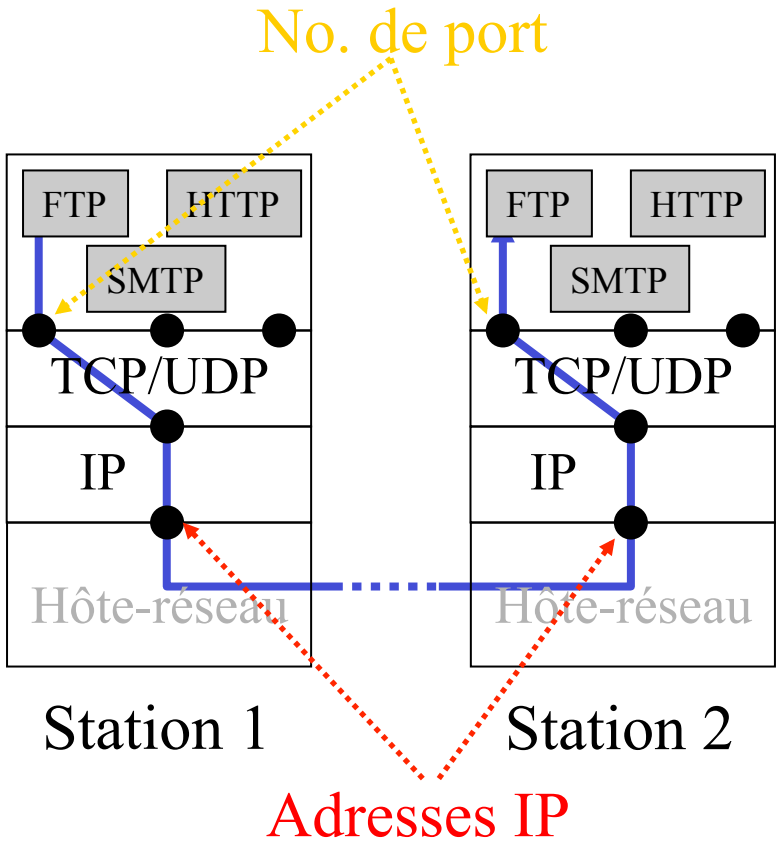
L’adressage

- Indique l’entité de la couche supérieur avec laquelle on veut communiquer
  - Modèle OSI
    - Couche réseau: NSAP (Network Service Access Point)
    - Couche transport: TSAP (Transport Service Access Point)



L'adressage dans le modèle TCP/IP

- Adressage de l'interface
  - Adresse IP
- Adresse d'un service
  - Port (= TSAP)
    - Permet de démultiplexer les transmissions
    - Entier sur 16 bits
    - Utilisés par TCP et UDP
    - TCP et UDP peuvent réutiliser les mêmes ports



Les ports TCP / UDP

- Deux types de numéros de port
  - « Ports bien connus »
    - Définis dans des RFC
    - Configurés dans le fichier `/etc/services` dans Unix

Service	Port	Protocole utilisé
ftp (données)	20	TCP
ftp (contrôle)	21	TCP
telnet	23	TCP
smtp	25	TCP
snmp	161	UDP
portmap	111	TCP

  - Ports éphémères
    - Assignés dynamiquement par le protocole PORTMAP
      - Un serveur s'enregistre auprès de PORTMAP de sa machine
      - Un client contacte le PORTMAP la machine éloignée pour demander le no de port correspondant à un nom d'une application

Interface vers la couche application: Sockets TCP/UDP

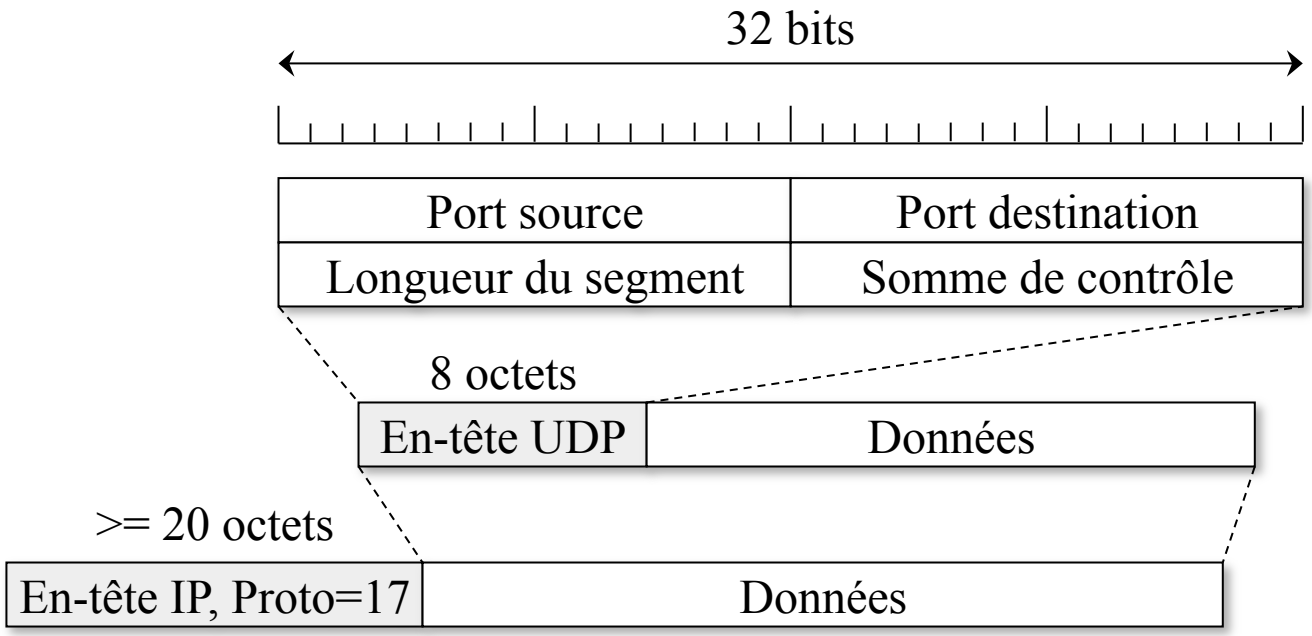
- La communication des applications à travers un réseau TCP/IP repose sur l'utilisation de sockets
- Socket: représente une extrémité d'une connexion
  - Identifié par
    1. L'adresse IP de la machine locale
    2. Le type du protocole utilisé: TCP ou UDP
    3. Un numéro de port
- Connexion
  - Association de deux sockets

Protocole de démultiplexage simple

- Protocole UDP: *User Datagram Protocol* (RFC 768)
- Fonctionnalités
  - Démultiplexage entre applications en utilisant des port
  - Contrôle d'erreur optionnel (obligatoire dans IPv6)
- Transmission non fiable
  - Sans acquittement ou retransmission
  - Sans contrôle de flux
  - Sans connexion

➢ Service de transmission similaire à IP

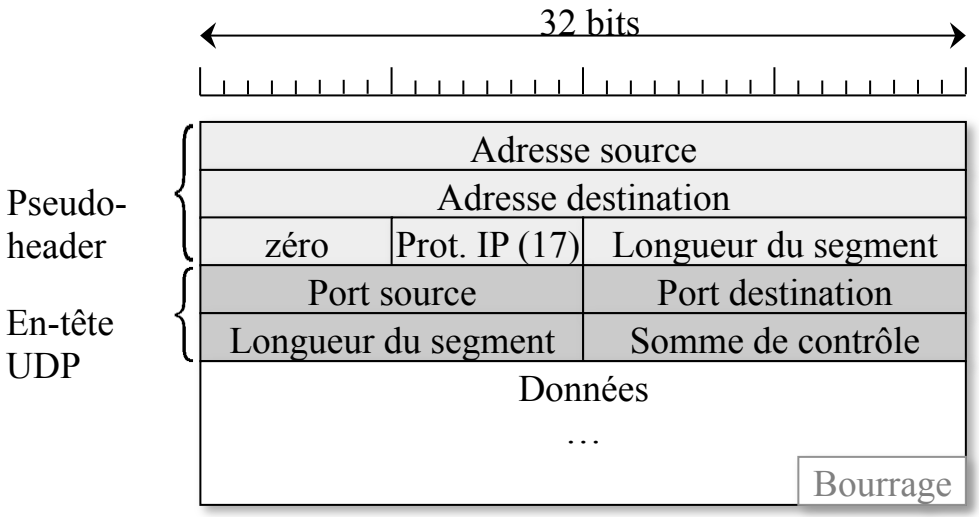
Segment UDP



- Longueur de l'en-tête: 8 octets
- Port source: optionnel, pour la réponse
- Longueur maximum d'un segment: 65'535 octets

Somme de contrôle

- Inclut
  - Le 'pseudo-header' (informations de l'en-tête IP)
  - L'en-tête UDP
  - Les données (+bourrage)



- Calcul
  - « The checksum is the 16-bit one's complement of the one's complement sum of the 16-bit words »

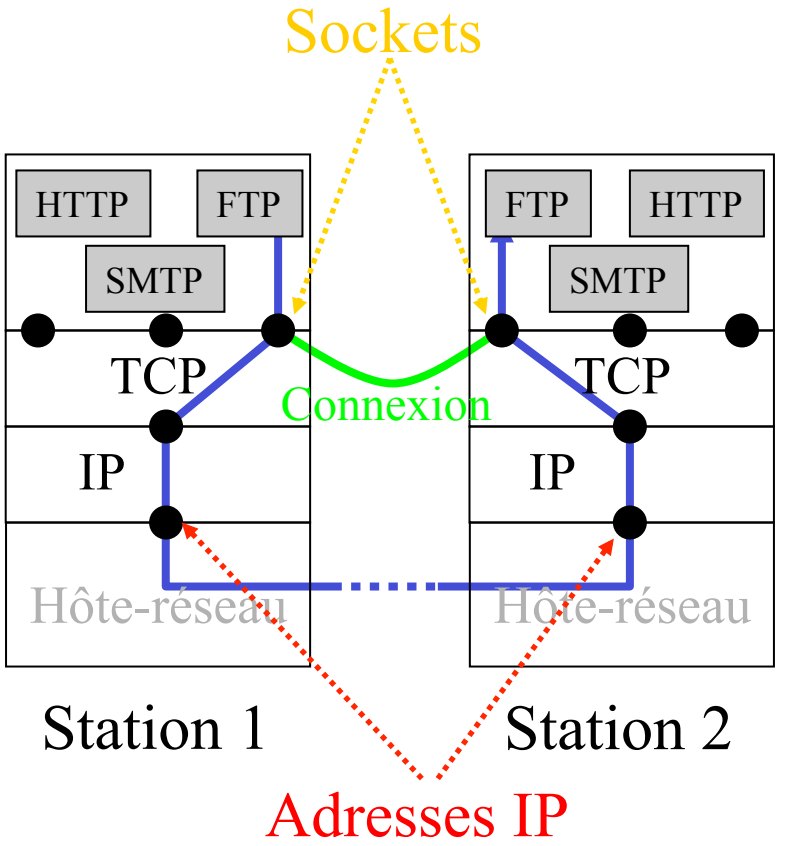
Exercices 5,6,7

Le protocole TCP

- *Transmission Control Protocol*, RFC 793
- Fonctionnalité principale :
  - ➢ Transmission fiable de bout en bout
- Fonctionnalités supplémentaires importantes
  - Contrôle de flux entre les systèmes terminaux
  - Contrôle de gestion du réseau
- Implémentations (interopérables !)
  - TCP Tahoe, TCP Reno, TCP NewReno, TCP Sack, TCP Vegas

Modèle de Service

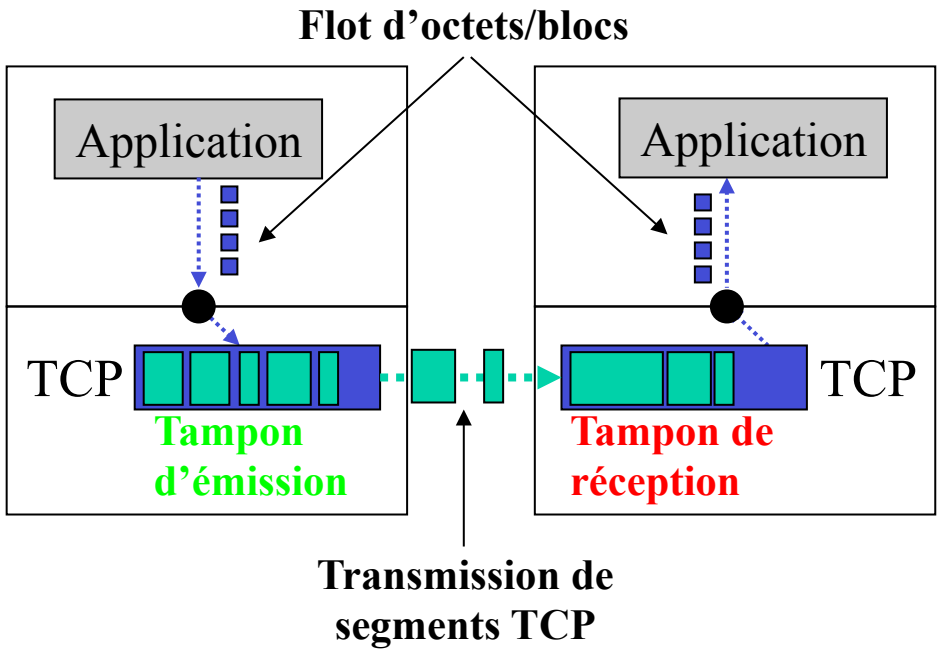
- Interface vers la couche d'application : sockets
- Connexions TCP
  - Connexion : liaison entre deux sockets
  - Transmission bidirectionnelle
  - Transmission point-à-point
    - Le multicast / broadcast n'est pas possible



Transfert tamponné à flot d'octets

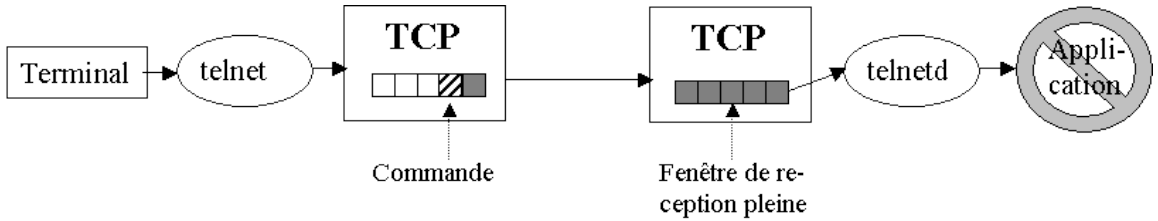
- TCP offre à la couche supérieur un service à flot d'octets
  1. L'application passe des blocs de données à TCP
  2. TCP met les données dans un tampon d'émission
  3. TCP regroupe les données en segments qui sont transmis
  4. Le récepteur TCP place les segments dans un tampon de réception
  5. TCP passe des données en bloc à l'application

➢ La délimitation des messages de l'application n'est pas préservée



Forcer la délivrance

- Le drapeau PUSH
  - L'application peut ordonner à TCP d'envoyer et délivrer les données immédiatement
  - Exemple : terminal à distance (Telnet)
    - Transmission après chaque retour chariot
  - ➢ L'émetteur TCP envoie les données sans attendre
  - ➢ Le récepteur TCP remet les données immédiatement à l'application
- Le drapeau URGENT (signalisation hors bande)
  - Permet de signaler au récepteur qu'il doit lire les données stockées par TCP
  - Exemple :
    - Application distante est bloqué et n'accepte pas les données de TCP
    - Permet de « tuer » l'application distante en envoyant un « Ctrl-C »
  - ➢ Le processus d'application est interrompu et lit les données de TCP



➢ L'interface TCP à la couche application permet de spécifier ces drapeaux

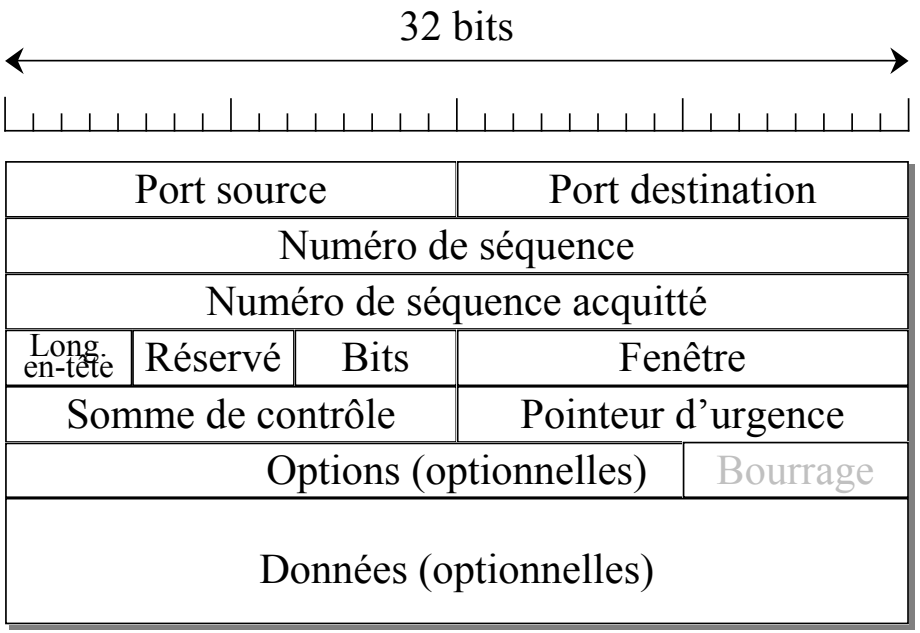
Transmission des données - Survol

Éléments de la transmission fiable :

1. Numéros de séquence (sur 32 bits)
   - TCP numérote les octets transmis et non pas les segments
     - Les données dans d'un segment peuvent changer lors d'un retransmission
2. Segments d'acquittement : confirmation la réception correcte d'un segment
   - Indique le numéro de séquence du prochain octet attendu
   - Dans une transmission bidirectionnelle, l'acquittement peut être transporté 'gratuitement' dans un segment de données normal ('piggybacking')
3. Temporisateur de retransmission
   - L'émetteur arme un temporisateur lors de la transmission de chaque segment
   - Si le temporisateur expire avec la réception d'un acquittement, le segment est retransmis

Format du segment TCP

- Ports et somme de contrôle
  - Comme dans UDP
- Numéro de séquence
  - Du premier octet des données
- Acquittement (optionnel):
  - Prochain no. de séquence attendu
- Longueur de l'en-tête
  - En mots de 32 bits
- Fenêtre : contrôle de flux
  - Espace libre du tampon de réception
- Pointeur d'urgence
  - Indique la fin des données urgentes
- Options : p. ex. MSS
  - Taille maximale acceptée d'un segment

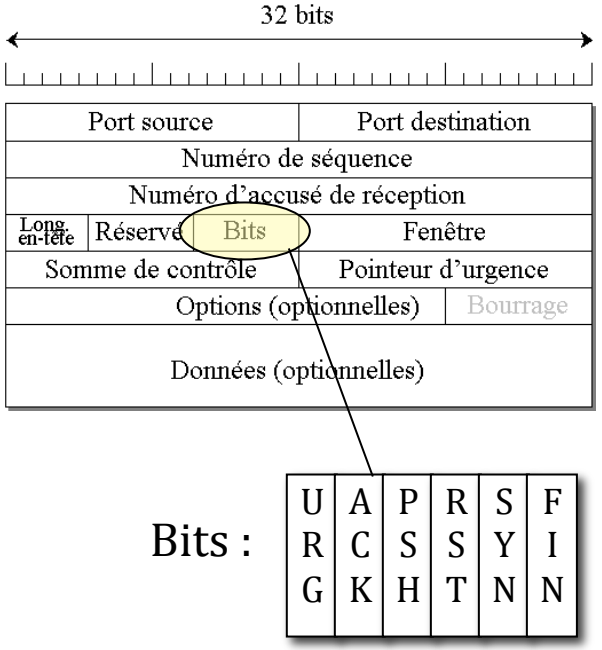


Bits :

U	A	P	R	S	F
R	C	S	S	Y	I
G	K	H	T	N	N

Bits de signalisation

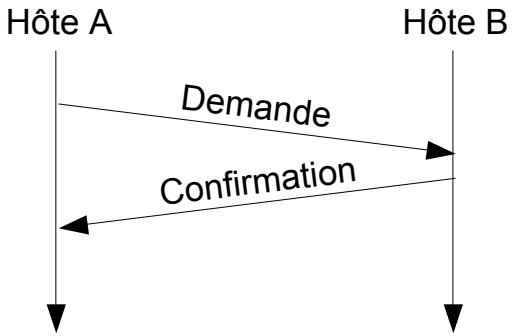
- URG : drapeau URGENT
- PSH : drapeau PUSH
- ACK :
  - Indique si le segment contient un acquittement
- SYN (synchronisation) :
  - Signale un segment SYN
  - Utilisé lors de l'établissement d'une connexion
- FIN (fin de connexion) :
  - Signale un segment FIN
  - Utilisé lors de la libération d'une connexion
- RST (reset)
  - Utilisé pour réinitialiser une connexion



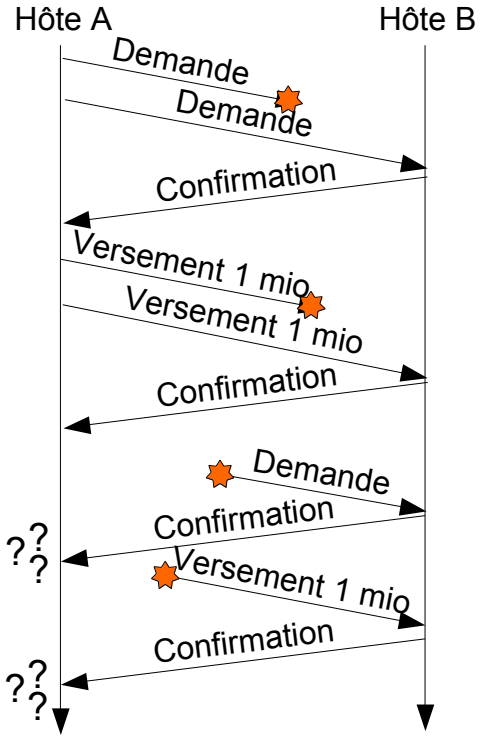
Exercice 9

Établissement fiable d'une connexion

- C'est simple ? Non !
- Protocole simpliste :
  - Établissement de connexion en deux temps
    1. Demande d'établissement
    2. Confirmation

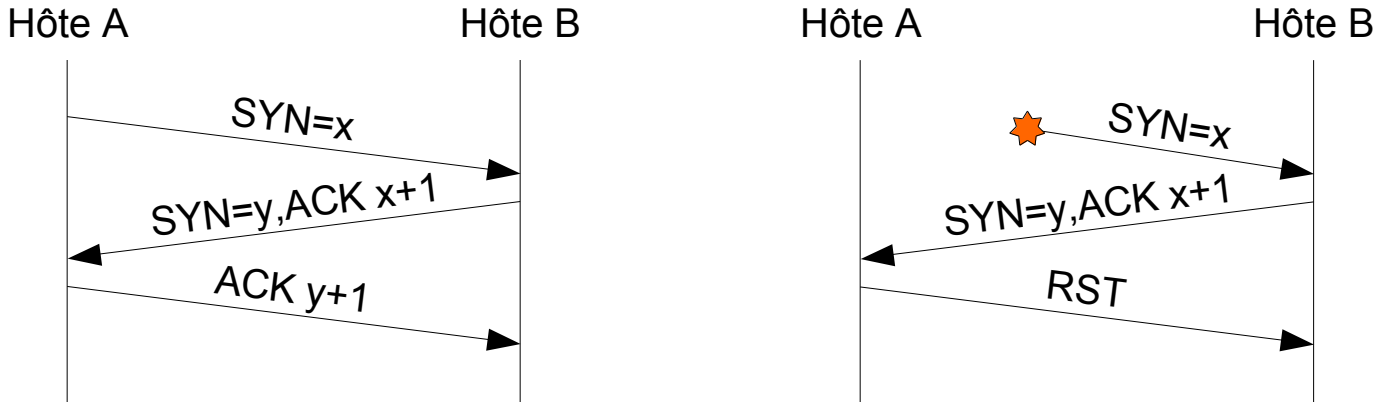


- Scénario
  - Ordre de versement à une banque
  - Tous les paquets du client sont retardés et retransmis



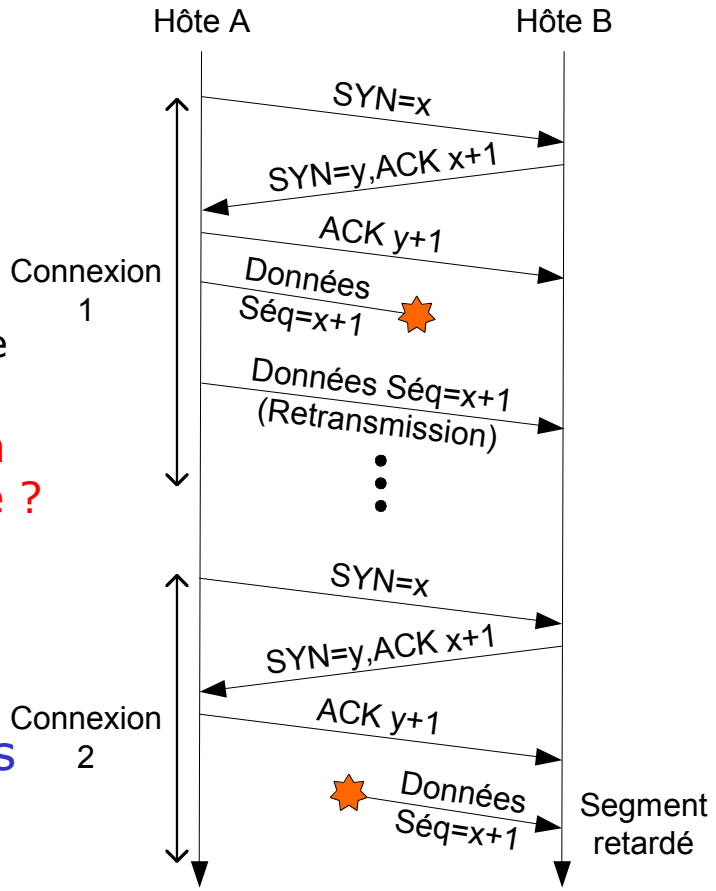
Établissement de connexion dans TCP

- Difficulté principale :
  - Tenir compte de la possibilité de doublons
- Mécanismes de TCP
  - Les segments SYN ont également un numéro de séquence
  - Un ordinateur qui reçoit un SYN demande à l'émetteur s'il est valable
  - ➢ *Three way handshake* (établissement de connexion en trois étapes)
    - ➢ Les hôtes préparent la transmission des données
    - ➢ Les hôtes se mettent d'accord sur les numéros de séquence initiaux



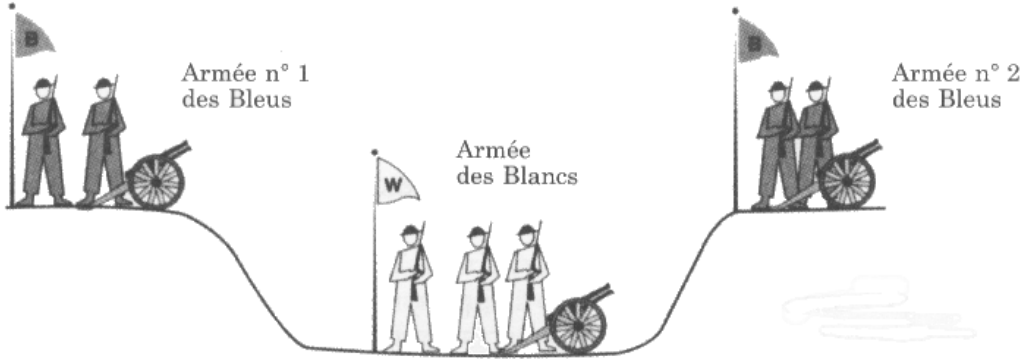
Durée de vie maximale d'un segment

- Problème des nouvelles incarnations d'une connexion
  - Deux hôtes peuvent établir et libérer des incarnations de la même connexion en succession rapide
    - Nouvelle incarnation : utilise la même paire de sockets
  - Comment éviter des confusions à cause d'un doublon retardé d'une connexion précédente ?
- ➢ On suppose une durée de vie limitée des segments
  - MSL : *Maximum segment lifetime* (2 min)
- ➢ Interdiction de réutiliser les mêmes sockets pendant 2 * MSL



Libération de connexion

- C'est simple ? Non plus !
- Protocole simpliste
  - Libération en deux temp
    - J'ai terminé ! Et vous ?
    - J'ai également terminé

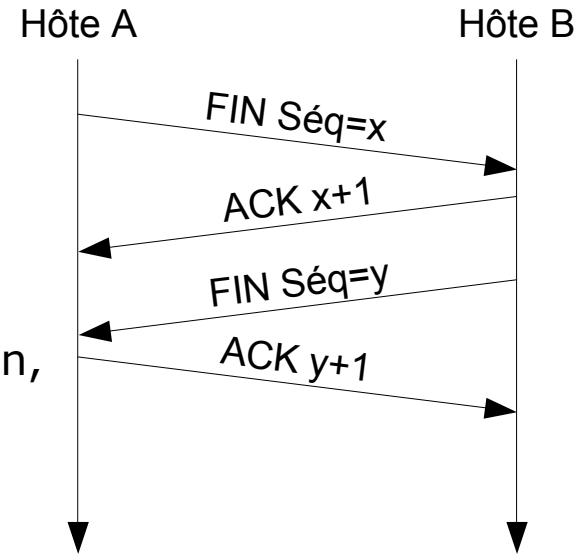


« Problème des deux armée

  - Les deux armée bleues doivent se mettre d'accord sur l'heure de l'attaque
    - Armée 1 : « Attaque à l'aube »
    - Armée 2 : « Confirmé »
  - Comment l'armée 2 sait-elle que la confirmation a été reçue ?
    - L'armée 1 pourrait confirmer la réception de la confirmation
      - Comment l'armée 1 sait-elle que cette confirmation a été reçue ?
  - …

Libération d'une connexion dans TCP

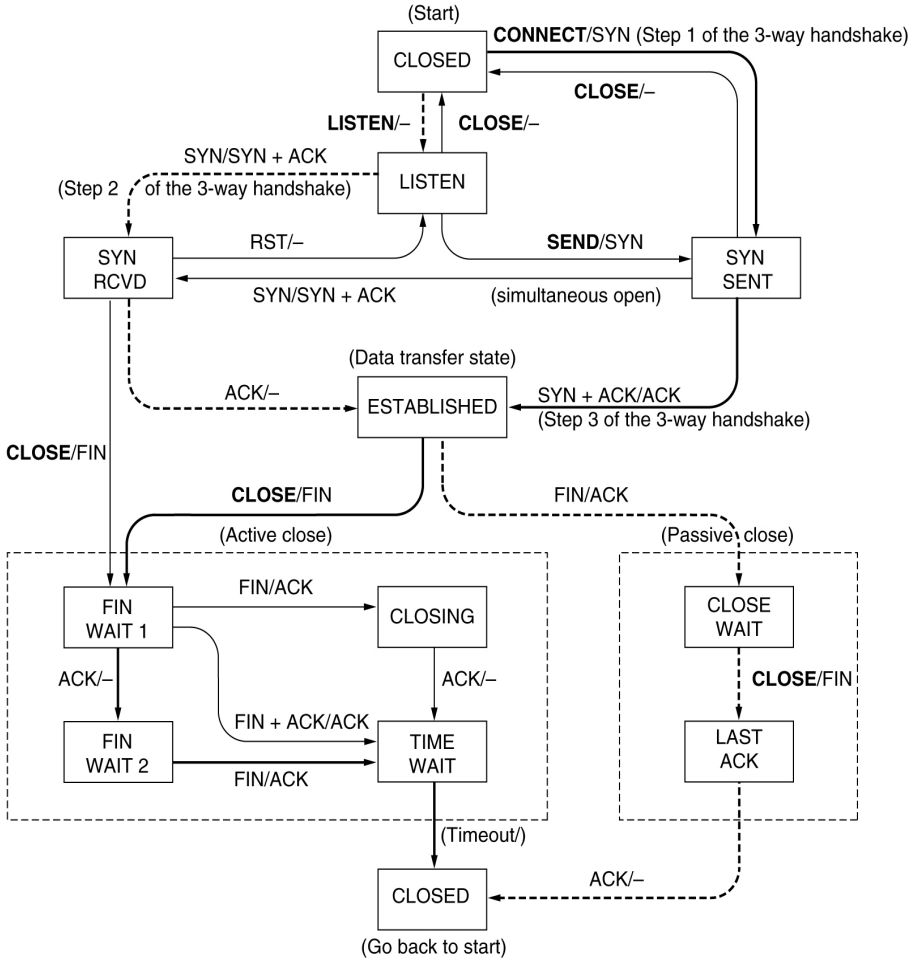
- Les connexions TCP sont bidirectionnelles
  - Libération séparément dans les deux sens
    - Un hôte peut demander la libération d'un sens de la connexion
    - L'autre hôte peut continuer à transmettre
    - ➢ Segments FIN et ACK pour libérer la connexion dans un sens



- Ne résout pas le problème des deux armées
- Solution pratique :
  - Retransmission du premier FIN
  - La connexion est libérée par une temporisation, même si le deuxième ACK n'arrive pas

Automate à nombre d'états finis de TCP

- Résume le comportement de TCP lors de l'établissement et de libération de connexions



indique les transitions normales pour le client

indique les transitions normales pour le serveur

appl : *indique les états de transitions pris lorsque l'application fait une opération*

recv : *indique les états de transitions pris lorsque le segment est reçu*

send : *indique ce qui est envoyé pour cette transition*

Transmission fiable

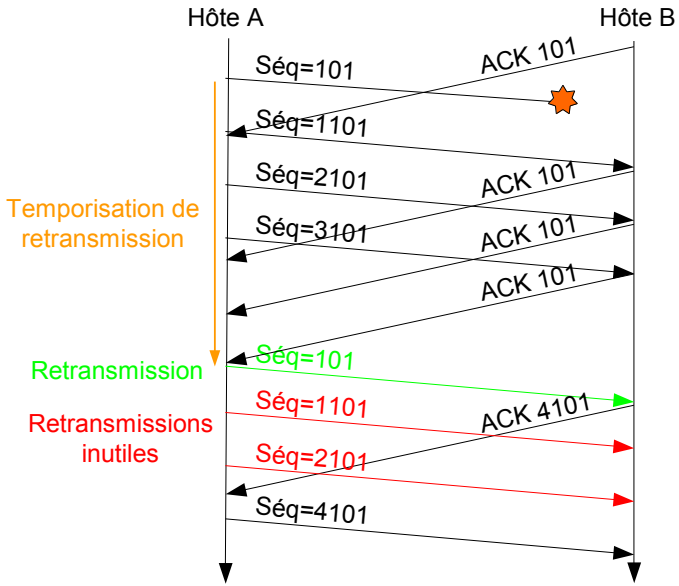
- Garantie de la délivrance des données, dans l'ordre de l'émission
- Doit être réalisée sur une infrastructure non-fiable (service best-effort d'IP)

Éléments de la réalisation

1. Numéros de séquence
   - Les octets transmis sont numérotés, non pas les segments
   - SYN et FIN ont également des numéros de séquence
2. Acquittement
3. Retransmission
   - Estimation de la temporisation de retransmission

Acquittements cumulatifs

- L'acquittement d'un numéro de séquence confirme la réception de tous les octets avant ce numéro de séquence
- Avantages
  - Il n'est pas nécessaire d'acquitter chaque segment séparément
    - Les implémentations actuelles acquittent chaque deuxième segment, sauf lors d'un délai trop grand
  - Un acquittement perdu n'implique pas nécessairement une retransmission
- Inconvénient principal
  - Si un segment intermédiaire a été perdu, le récepteur ne peut pas signaler la réception correcte des segments suivants
  - L'émetteur retransmettra probablement tous les segments à partir du segment perdu (Méthode Go-back-n)

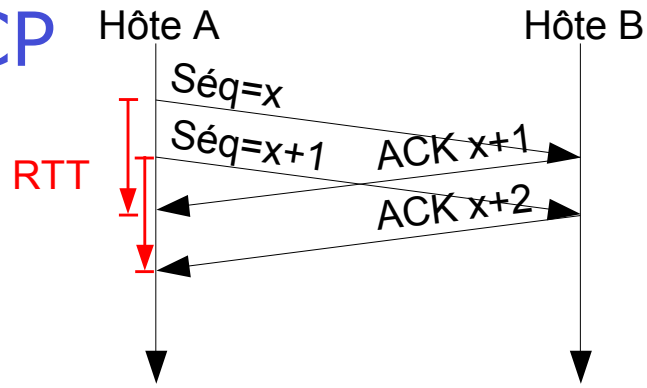


Temporisateur de retransmission

- L'expiration d'un temporisateur déclenche la retransmission d'un segment
- Problème : quelle valeur choisir pour le temporisateur ?
  - Valeur trop petite : retransmissions inutiles
  - Valeur trop grande : attente inutile lors d'une perte
- La bonne valeur doit dépendre du temps aller retour normal entre l'émission du segment et la réception de l'acquittement

➢ Paramètres importants de TCP

- RTT : *Round Trip Time*
- RTO : *Retransmission Timeout*



Estimation du RTT

- Méthode originale préconisée dans la RFC 793
  1. Mesurer le temps RTT entre l'émission d'un segment et la réception de l'ACK correspondant
     - La plupart des implémentations TCP ne mesurent qu'un segment à la fois et non pas tous
  2. Lissage exponentielle des mesures : SRTT (*Smoothed RTT*)

$$SRTT = \alpha \cdot SRTT + (1 - \alpha) \cdot RTT$$

     - alpha : Coefficient de lissage (recommandé : alpha=0,9)
       - Détermine la vitesse de l'adaptation aux variations du RTT
  3. Timeout de retransmission

$$RTO = \beta \cdot SRTT$$

     - beta: Coefficient de variance du RTT (recommandé : beta=2)

Estimation améliorée du RTT

- En 1986 Internet a souffert plusieurs effondrements
  - Chute du débit effectif d'un facteur 1000 sur quelques liens importants
- Constatation
  - TCP était incapable de s'adapter aux charges élevées
    - Une augmentation brusque du RTT provoquait beaucoup de retransmissions

➢ TCP injectait du trafic supplémentaire dans un réseau déjà congestionné

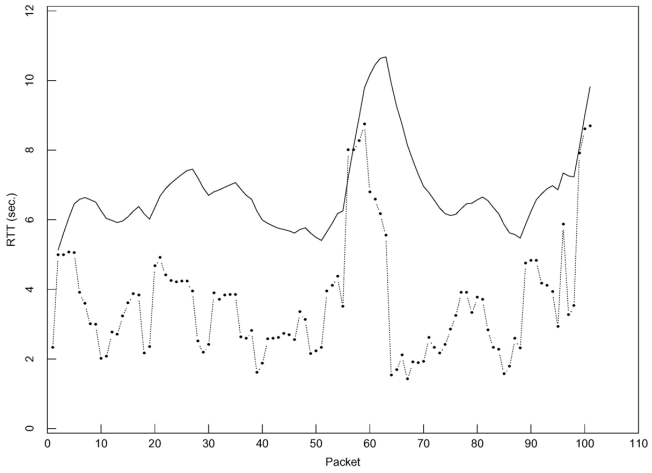
Analyse du problème

- Théorie des files d'attente
  - Charge rho = Fréquence des arrivées * Temps de services des paquets
- La variance du RTT est inversement proportionnelle à rho
  - Exemple :
    - Charge rho=75% peut provoquer des variations du RTT d'un facteur 16
  - Le calcul $RTO = \beta \cdot SRTT$ avec beta=2 tolère des variations du RTT d'un facteur 2
  - ➢ Applicable pour une charge maximale du réseau de 30 % !
- Solution :
  - Estimation non seulement du RTT mais aussi de sa variance

Méthode améliorée de calculer RTO

$$Err = RTT - SRTT$$
$$SRTT = SRTT + g \cdot Err$$
$$D = D + h \cdot (|Err| - D)$$
$$RTO = SRTT + 4D.$$

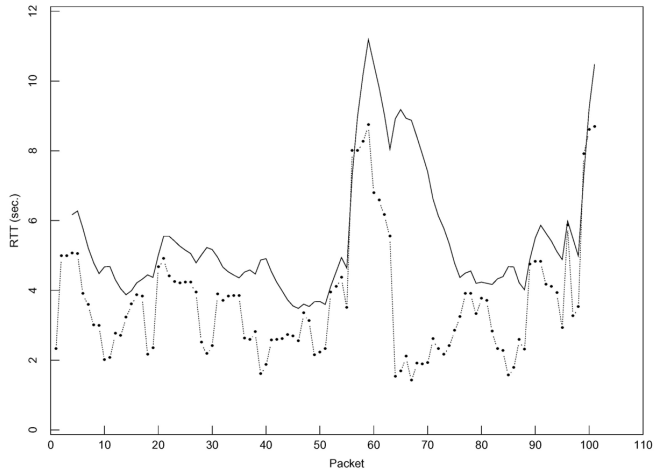
- Err : différence entre SRTT et nouvelle mesure
- D : écart moyen du RTT
- g, h : contrôlent la vitesse de l'adaptation à des variations du RTT
  - Recommandé: g = 1/8, h=1/4



Méthode originale

RTO

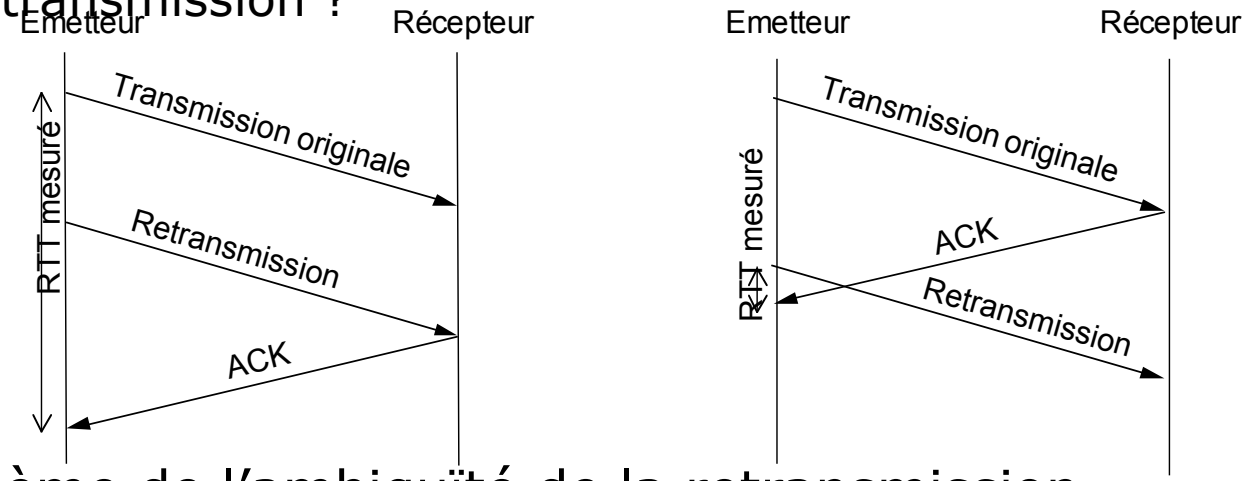
RTT mesuré



Méthode améliorée

Algorithme de Karn

- Comment mesurer le RTT si un segment a été retransmis ?
  - Est-ce que l'ACK concerne le segment original ou la retransmission ?

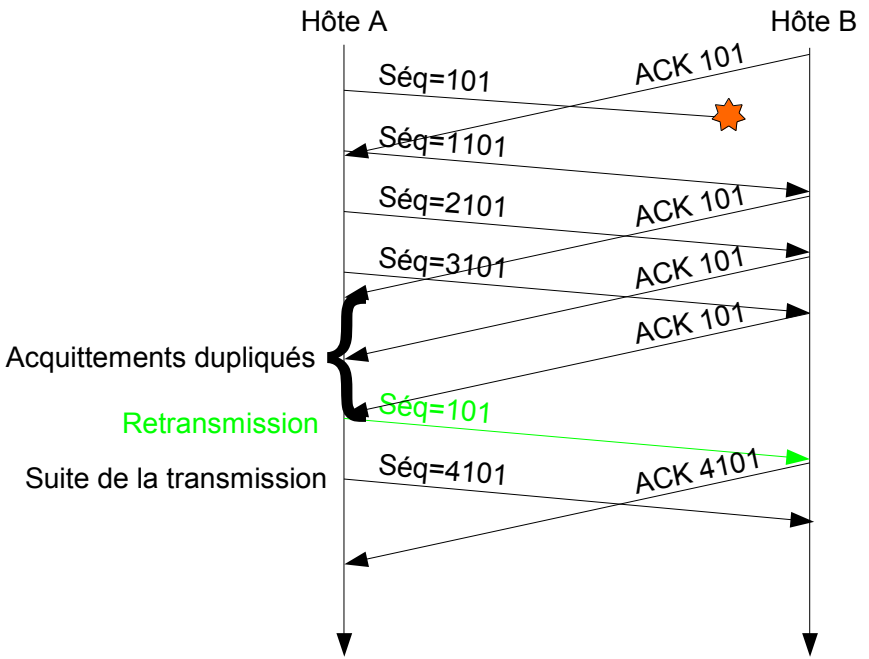


- Problème de l'ambiguïté de la retransmission
- Solution (Algorithme de Karn) :
  - ➢ Ignorer les mesures du RTT concernant des segments qui ont été retransmis

Retransmission rapide

- Problème des acquittements cumulatifs
  - Un segment intermédiaire a été perdu
  - Comment signaler que les segments suivants sont arrivés ?

- Solution :
  - Lors de la réception d'un segment en désordre, le récepteur répète le dernier ACK ('*dup ACK*')
  - Si l'émetteur reçoit 3 dup-ACKs, il retransmet le segment sans attendre un timeout



Résumé: service de TCP

- Connexions bidirectionnelles
- Établissement de connexion
  - *Three-way handshake*
- Libération de connexion
  - Séparément pour chaque sens
- Service de transmission fiable
  - Numéro de séquence pour chaque octet transmis
  - Acquittements cumulatifs
  - Retransmission déclenchée par un temporisateur
    - Estimation adaptative du RTT

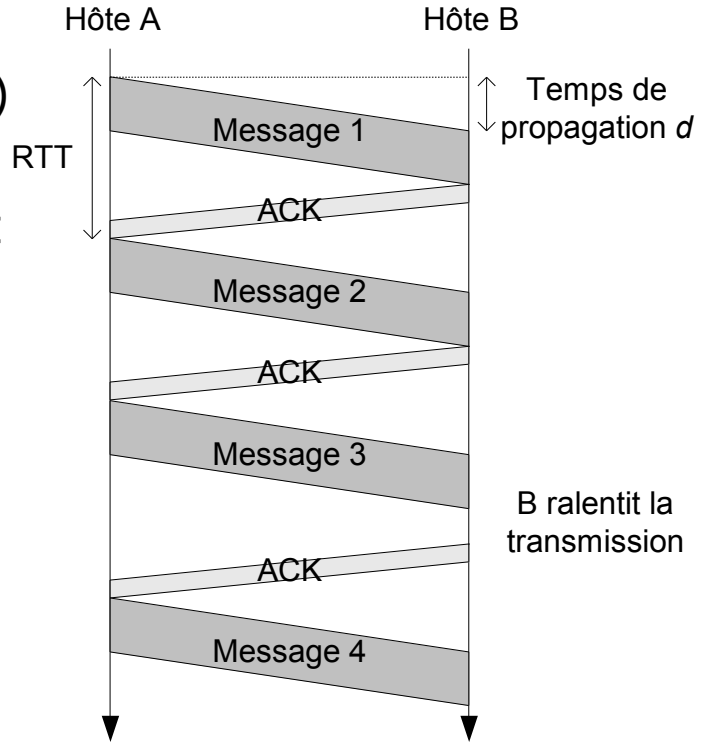
Exercices 8,15,16,18,19

Contrôle de flux et de congestion

- Objectif
  - ➢ Régulation de la vitesse de transmission
- Contrôle de flux
  - – Adaptation à la vitesse du récepteur
  - – Évite qu'un émetteur rapide surcharge un récepteur lent
- Contrôle de congestion
  - – Adaptation à la vitesse du réseau
  - – Évite des pertes excessives de paquets à cause d'une surcharge du réseau

Rappel : contrôle de flux

- Méthode la plus simple :
  - « Stop and Go » (Envoyer et attendre)
- Algorithme
  - Le récepteur acquitte chaque segment séparément
  - L'émetteur ne peut envoyer un nouveau segment qu'après la réception de l'acquittement du segment précédent
  - ➢ Le récepteur peut ralentir la transmission en retardant les acquittements
- ➢ Simple mais faible utilisation du réseau

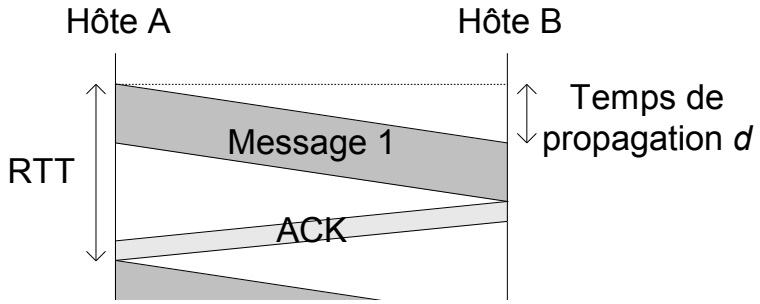


Performances du protocole « Stop and Go »

- Taux d'utilisation du réseau U :
  - Rapport entre le débit effectif obtenu D et la capacité du réseau C
- Calcul
  - L_{mess}: longueur d'un message en bits
  - L_{ack}: longueur d'un ACK en bits

$$RTT = (L_{Mess} + L_{ACK}) / C + 2d$$

$$U = \frac{L_{Mess}}{RTT \cdot C} = \frac{L_{Mess}}{L_{Mess} + L_{ACK} + 2dC}$$

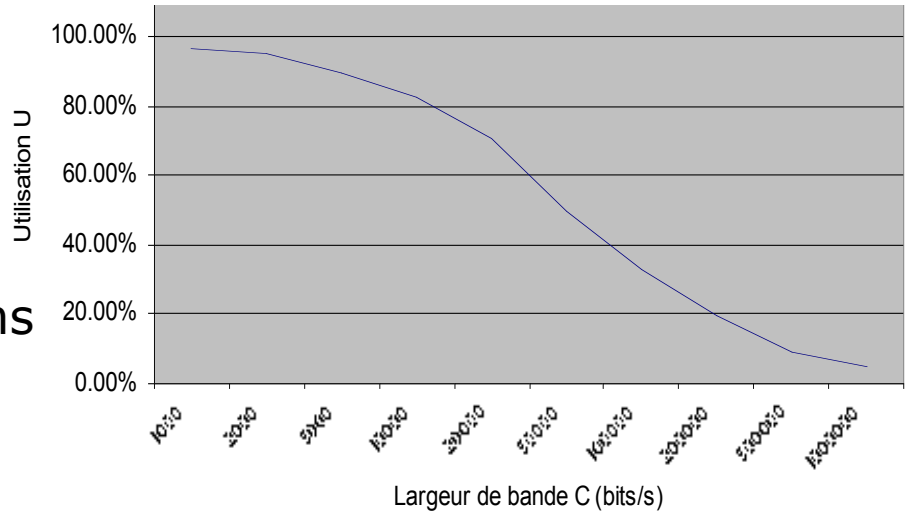


- Utilisation est inversement proportionnelle au « produit largeur de bande–délai » (*bandwidth delay product*)

$$BWD = RTT \cdot C$$

Exemple numérique

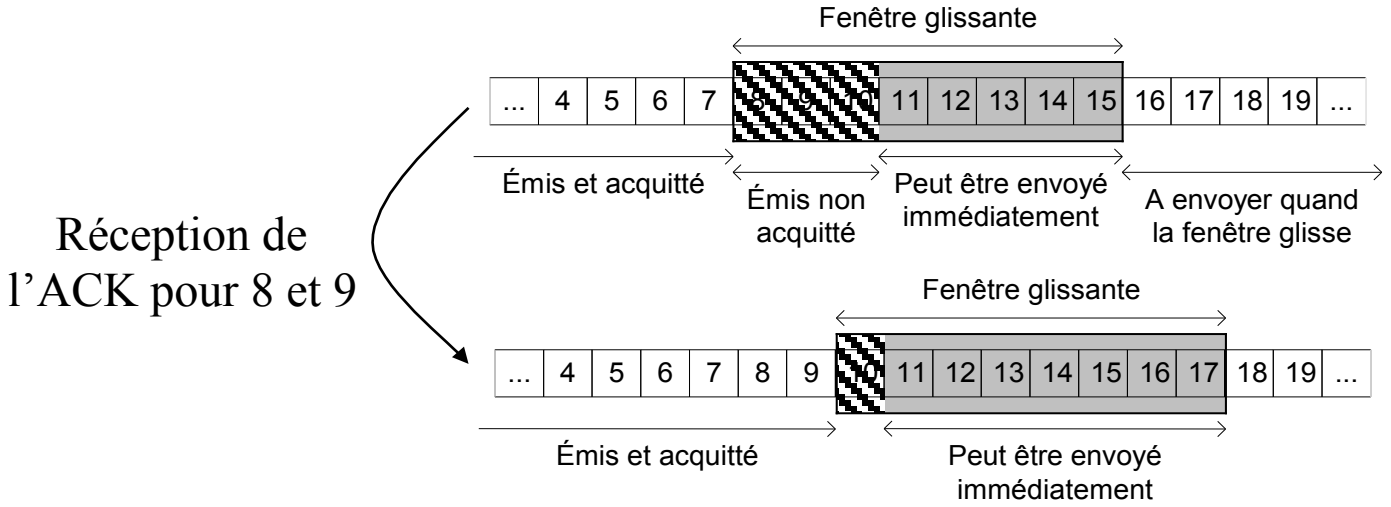
- Paramètres:
  - L_{mess}=1000 bit
  - L_{ack}: 10 bit
  - Délai de propagation d = 10ms
  - Largeur de bande C entre 1 kb/s et 1 Mb/s



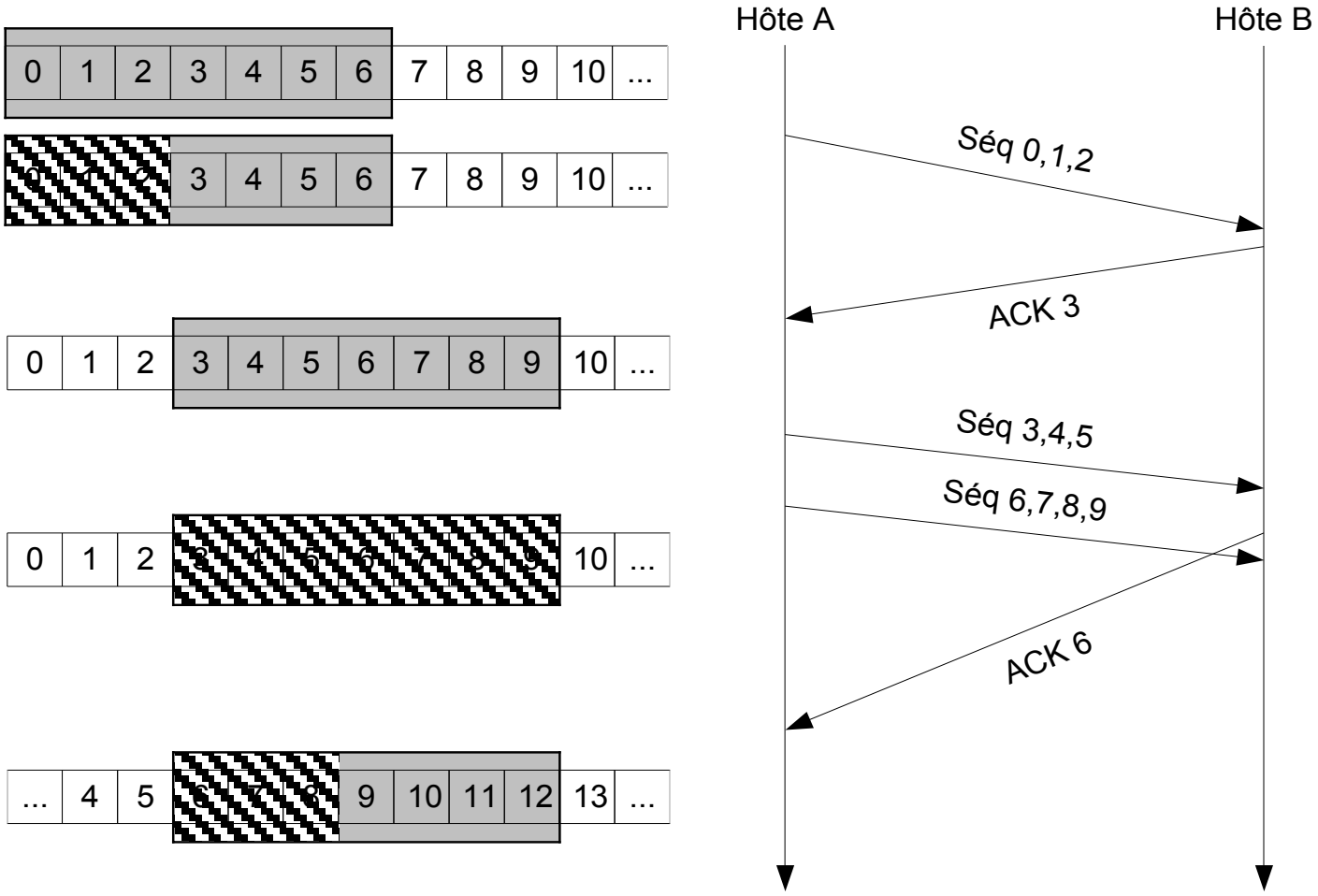
- Utilisation inférieure à 5% à C = 1 Mb/s
- Stop and Go n'est approprié que pour les réseaux à faible produit largeur de bande – délai

Protocole de la fenêtre glissante

- Méthode de contrôle de flux plus élaborée
- Améliore le taux d'utilisation
  - Permettant à l'émetteur d'envoyer plusieurs paquets avant de devoir attendre un accusé de réception
- Principe
  - Les données dans la fenêtre peuvent être envoyées sans attendre d'acquittement
  - La réception d'un acquittement permet de glisser la fenêtre à droite



Gestion de la fenêtre glissante



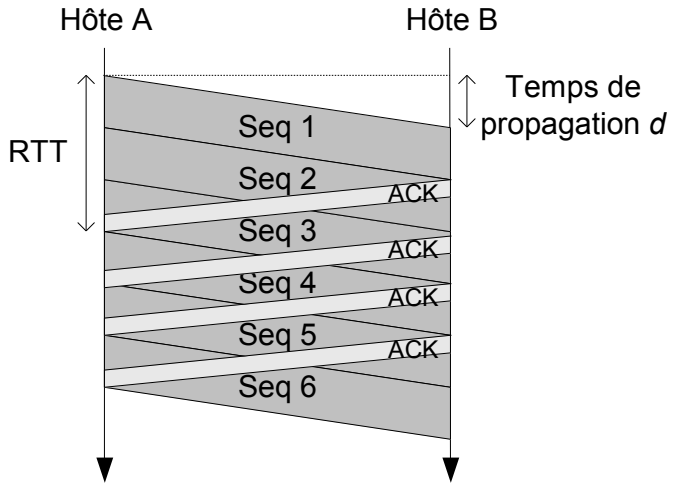
Performances du protocole de la fenêtre glissante

- Une fenêtre suffisamment grande permet d'exploiter le canal de transmission à 100 %
- Taille optimale de la fenêtre :

$$RTT = (L_{Mess} + L_{ACK}) / C + 2d$$

$$U = 1 = \frac{W}{RTT \cdot C}$$

$$W = RTT \cdot C$$



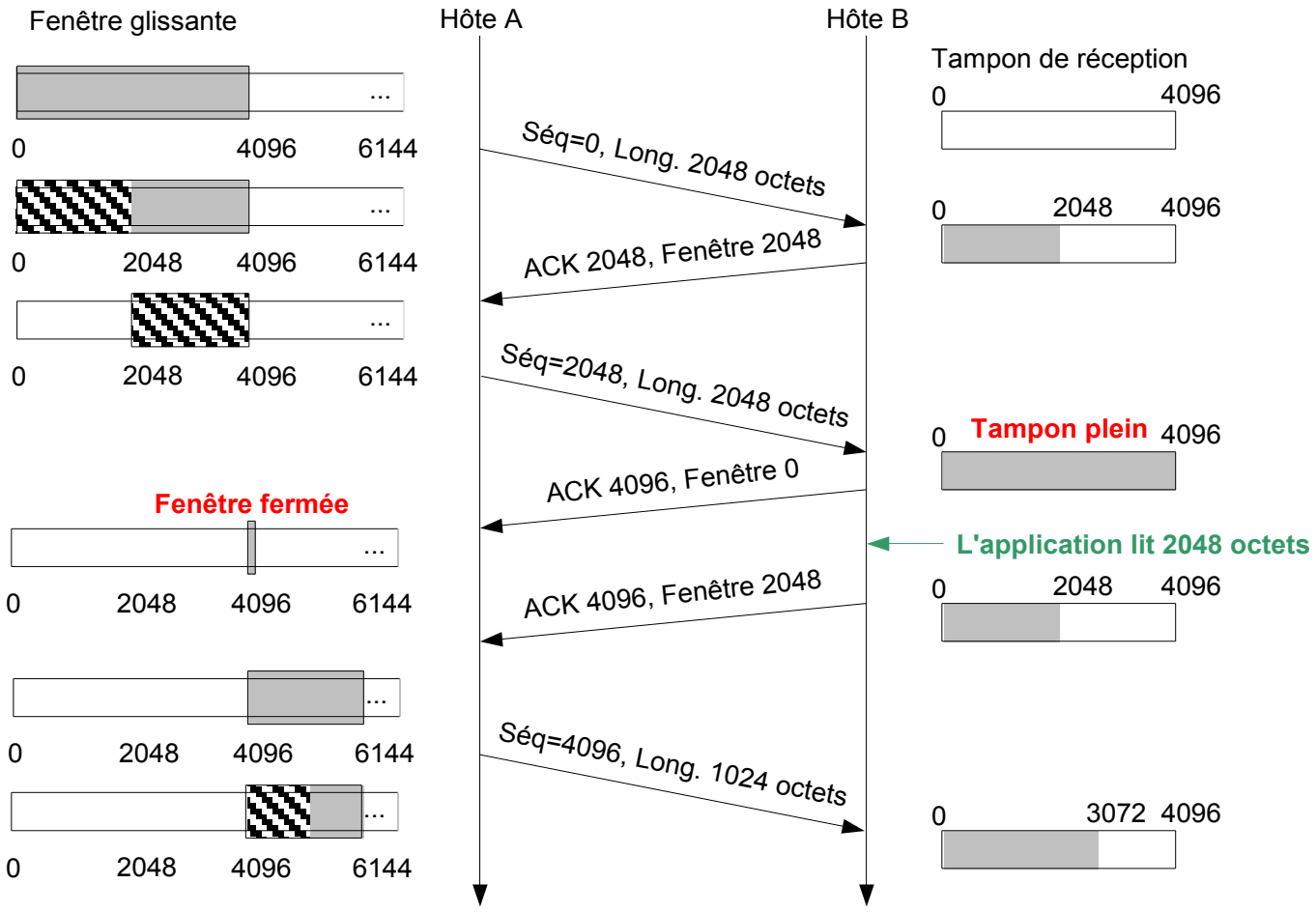
➢ La taille optimale de la fenêtre glissante est égale au produit largeur de bande – délai du canal

Contrôle de flux dans TCP

- Basé sur un protocole à fenêtre glissante mais avec une taille de fenêtre variable
- La fenêtre utilisable correspond à la place libre dans le tampon du récepteur
- Chaque accusé de réception indique en plus la taille de la fenêtre (*window advertisement*)

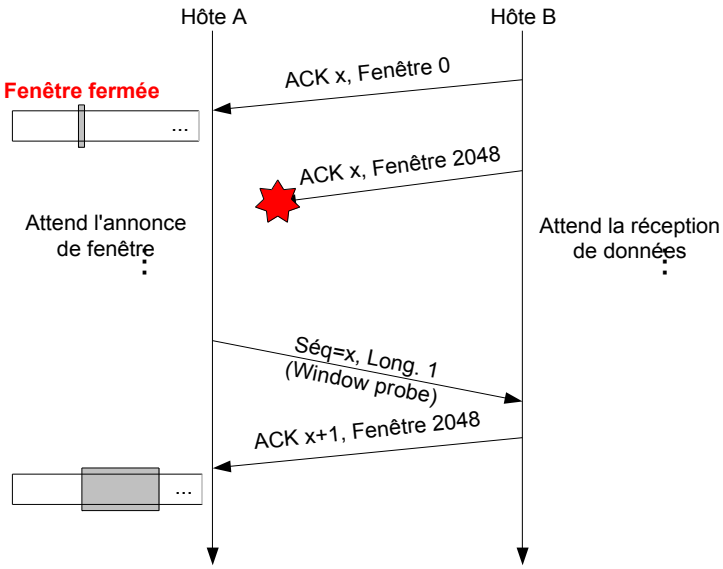
➢ En variant la taille de la fenêtre, le récepteur peut contrôler la vitesse de transmission de l'émetteur

Exemple du contrôle de flux dans TCP



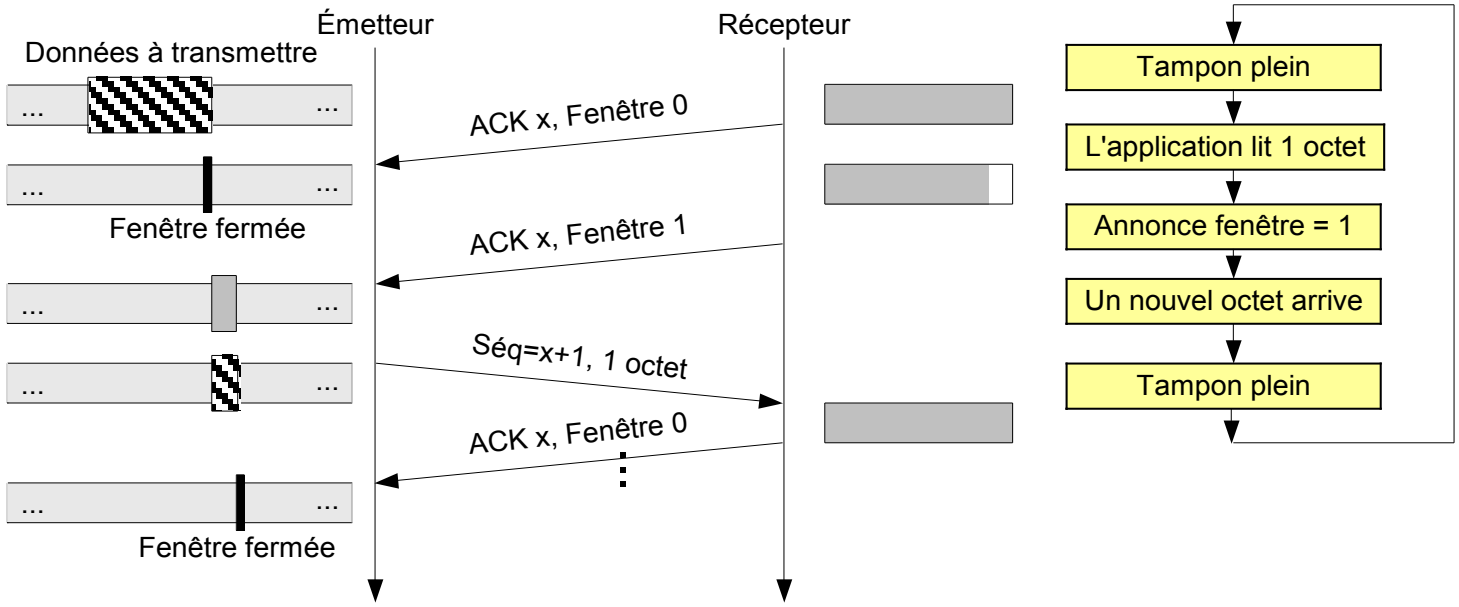
Fenêtre fermée

- Si le tampon de réception est plein, le récepteur arrête la transmission en indiquant une fenêtre nulle
- Quand la fenêtre est nulle, l'émetteur ne peut transmettre des données que dans deux cas
  - Données urgentes (avec le drapeau URG)
  - Sondes de fenêtre (*window probes*)
    - Lors d'une fenêtre nulle, la perte de l'annonce d'une fenêtre non nulle causerait un interblocage
    - ➢ L'émetteur envoie périodiquement de petits segments d'un octet qui obligent le récepteur de ré-annoncer la fenêtre



Le syndrome de la « fenêtre stupide »

- Silly Window Syndrome
  - Problème de performances lorsque l'application réceptrice lit les données octet par octet
  - ➢ Chaque accusé de réception annonce un petit espace disponible et chaque segment ne transporte qu'une petite quantité de données.



Éviter la fenêtre stupide

- L'émetteur et le récepteur contribuent à ce problème, donc les deux côtés doivent contribuer à le résoudre
- Côté récepteur : ne pas annoncer de petites fenêtres
  - Ne pas annoncer de nouvelle taille de fenêtre jusqu'à ce que puisse être agrandie
    - soit par un segment de pleine taille (c'est-à-dire le MSS),
    - soit par la moitié de l'espace du tampon du récepteur
- Côté émetteur : ne pas transmettre de petits segments
  - Ne pas transmettre un segment que si
    - un segment de pleine taille peut être envoyé, ou
    - un segment peut être envoyé qui a au moins la moitié de la taille du tampon de réception
    - L'ACK du segment précédent a déjà été reçu et toutes les données du tampon d'émission peuvent être envoyées

Exercices 25, 27, 28

Contrôle de congestion

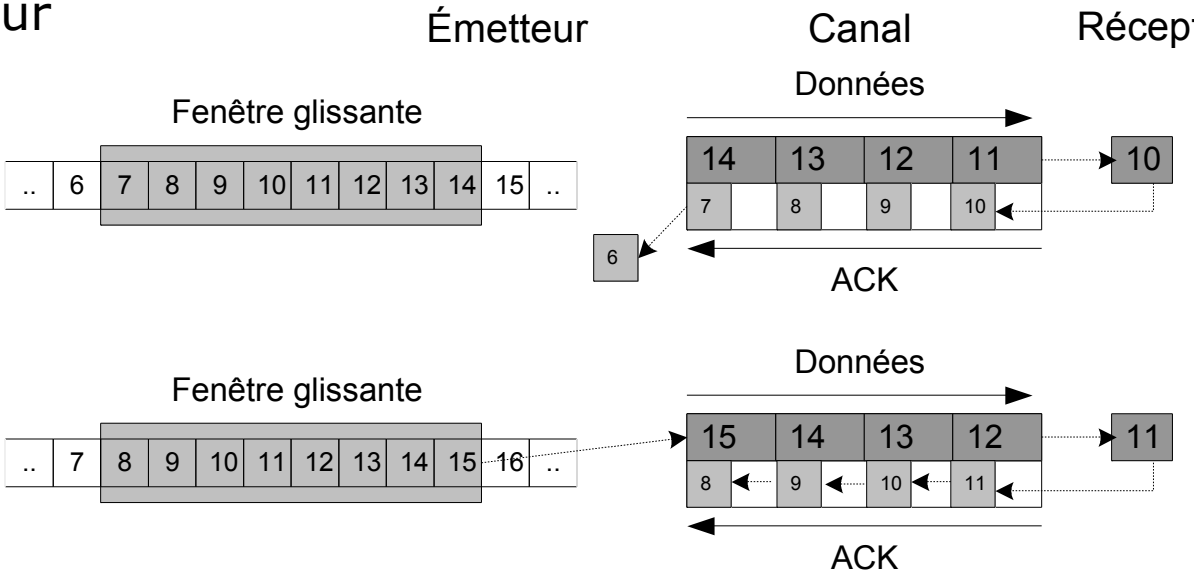
- État de congestion
  - Le réseau n'est plus en mesure de transporter tout le trafic injecté et supprime des paquets
- Danger de l'amplification d'une congestion par TCP
  - TCP réagit à une perte avec une retransmission ce qui peut augmenter la charge du réseau

➢ Le contrôle de congestion de TCP doit optimiser le débit de transmission sans mettre en danger la stabilité du réseau

- Défis
  1. Déterminer la capacité disponible sur le réseau
  2. Ajuster le débit pour obtenir le débit optimal qui permet un régime de transmission stable et en équilibre

Comportement stationnaire de TCP

- Débit optimal : équilibre stable de la transmission
  - Taille de la fenêtre = Produit largeur de bande – délai
  - Un nouveau paquet n'est injecté dans le réseau que si un autre est sorti
  - Toute une fenêtre de données est en transit entre l'émetteur et le récepteur
  - La vitesse de transmission est autorégulée par le débit aller-retour



Bases du contrôle de congestion de TCP

- La fenêtre de congestion (cwnd, *congestion window*)
  - Gérée par l'émetteur pour limiter le débit d'émission
  - TCP adapte la taille de cwnd au niveau de congestion détecté
- Combinaison du contrôle de flux et de congestion

$$\text{Fenêtre effective} = \min\left(\text{Annonce de fenêtre}, cwnd\right)$$

- Principe du contrôle de congestion
  - Pour détecter le débit optimal, TCP commence avec une petite fenêtre de congestion et augmente rapidement le débit (→ Démarrage lent)
  - Dès que TCP s'approche à la zone de congestion, TCP augment le débit plus lentement (→ Évitement de congestion)
  - Dès qu'une perte est détecté TCP ralentit rapidement et augmente à nouveau lentement

Démarrage lent (*Slow Start*)

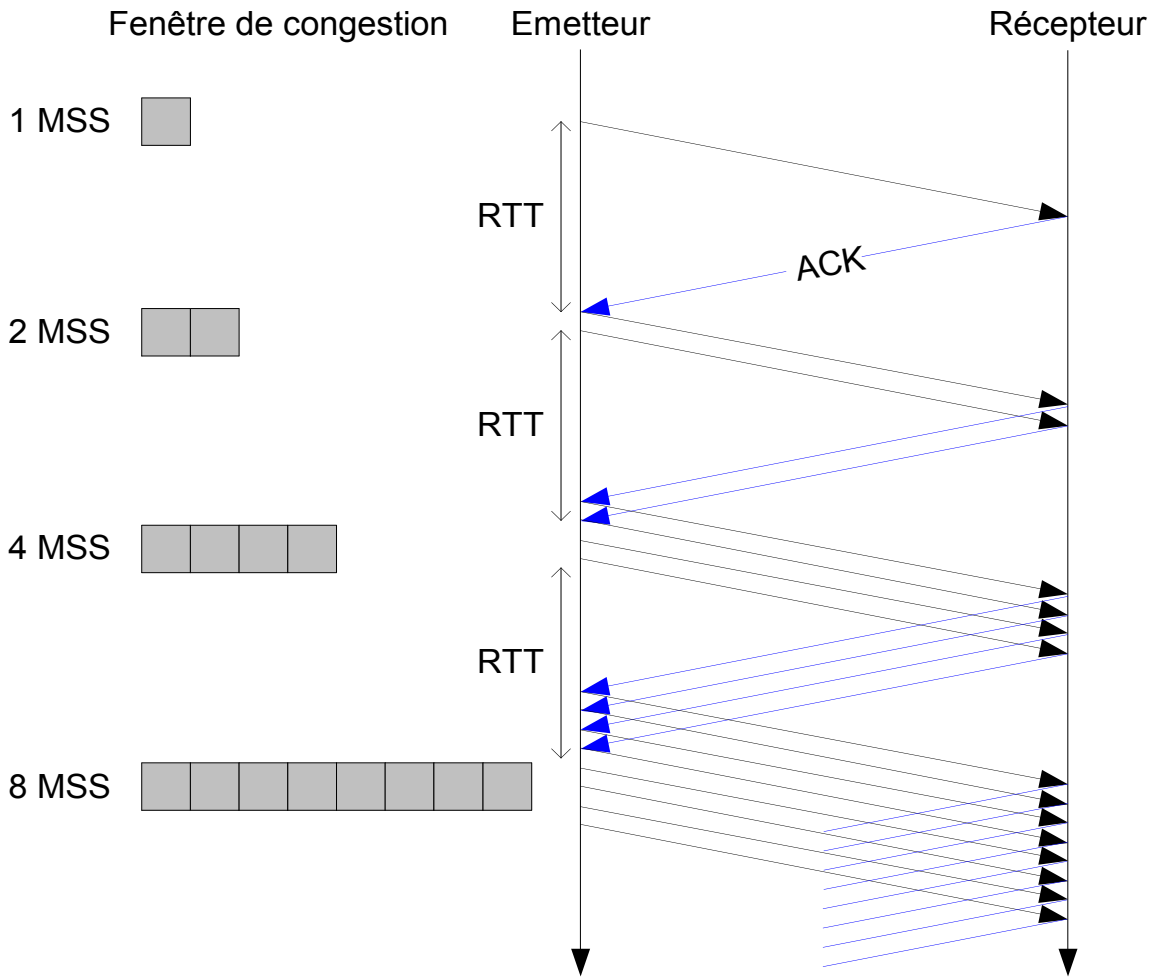
- Motivation
  - TCP doit fonctionner correctement pour n'importe quelle capacité du réseau (entre bits/s et Gb/s)
  - Il faut éviter de surcharger un lien lent dès le début
    - ➢ Petite fenêtre cwnd au début
  - Il faut rapidement arriver à exploiter la capacité de liens importants
    - ➢ Augmentation rapide de cwnd

Algorithme Slow Start

- Initialement cwnd = 1 MSS
- Ensuite cwnd est incrémenté de 1 MSS par acquittement reçu

➢ Cwnd double chaque RTT

Exemple de Slow Start



Évitement de congestion (*Congestion Avoidance*)

- Dans Slow Start, la taille de cwnd augmente de manière exponentielle
- Augmentation doit ralentir quand TCP s'approche au débit optimal
- Un seuil d'évitement de congestion indique quand on s'approche à une congestion
  - Paramètre ssthresh de TCP

Algorithme Congestion Avoidance

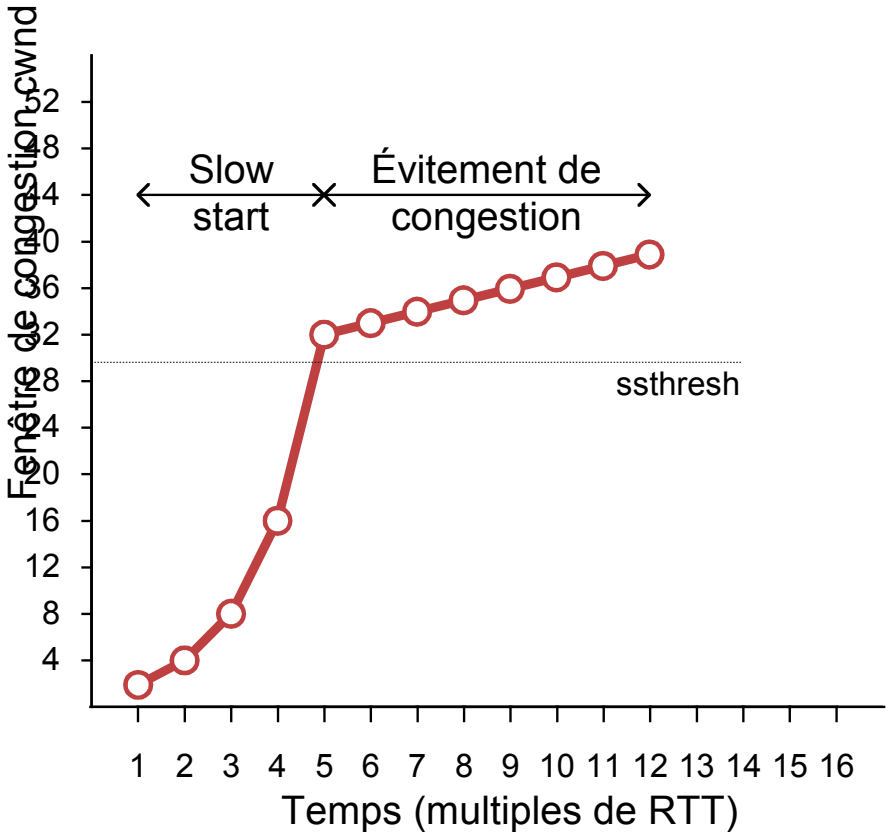
- Dès que cwnd > ssthresh, cwnd est agrandie linéairement
- Pour chaque acquittement reçu :

$$cwnd \leftarrow cwnd + \frac{MSS \cdot MSS}{cwnd}$$

Exemple
Slow Start et Congestion Avoidance

- Au-dessous de ssthresh:
  Slow Start
  - Cwnd double chaque RTT

- Au-dessus de ssthresh:
  Congestion Avoidance
  - Cwnd croît d'un segment chaque RTT



Variation du seuil d'évitement de congestion

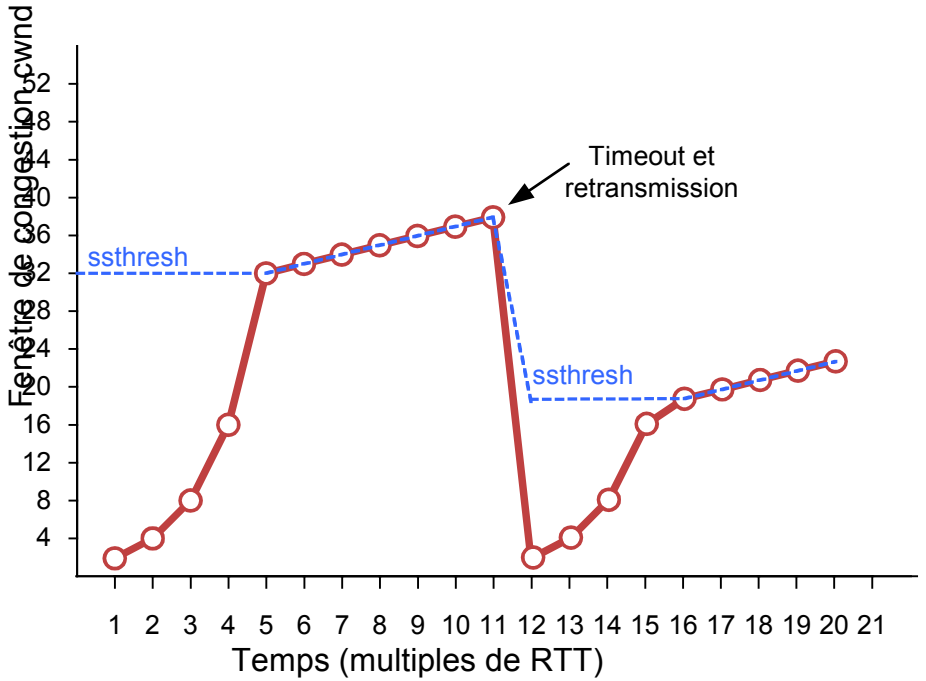
Comment déterminer la valeur du seuil de congestion ssthresh ?

- Ssthresh doit représenter la taille 'optimale' de la fenêtre de congestion

- Lorsqu'il n'y a pas de congestion, ssthresh croît linéairement avec cwnd → accroissement additif
- Les signal que le débit optimal a été dépassé est la perte d'un paquet
  - Théorie des files d'attente
    - Lors d'une congestion, la longueur des files d'attente peut croître de manière exponentielle
    - Pour garantir la stabilité du réseau, le débit doit diminuer de manière exponentielle
- Lorsqu'une perte a été détectée, ssthresh est diminué à la moitié → décroissance multiplicative
- TCP recommence avec Slow Start (après un timeout)
  - Pour éviter des rafales de retransmissions

Comportement de cwnd et ssthresh

- Trois phases
  - Slow Start avec une croissance exponentielle de cwnd
  - Congestion avoidance (accroissement additif de ssthresh)
  - Diminution de ssthresh lors d'une perte (décroissance multiplicative de ssthresh)



Recouvrement rapide (*Fast Recovery*)

- L'utilisation de Slow Start après des pertes isolées n'est pas efficace
  - Si des segments intermédiaires ont été perdus mais les segments suivants sont arrivés, cela indique une congestion légère et ne justifie pas Slow Start
- Recouvrement rapide pour résoudre rapidement la perte de segments isolés
  - Utilisé en combinaison avec Retransmission Rapide

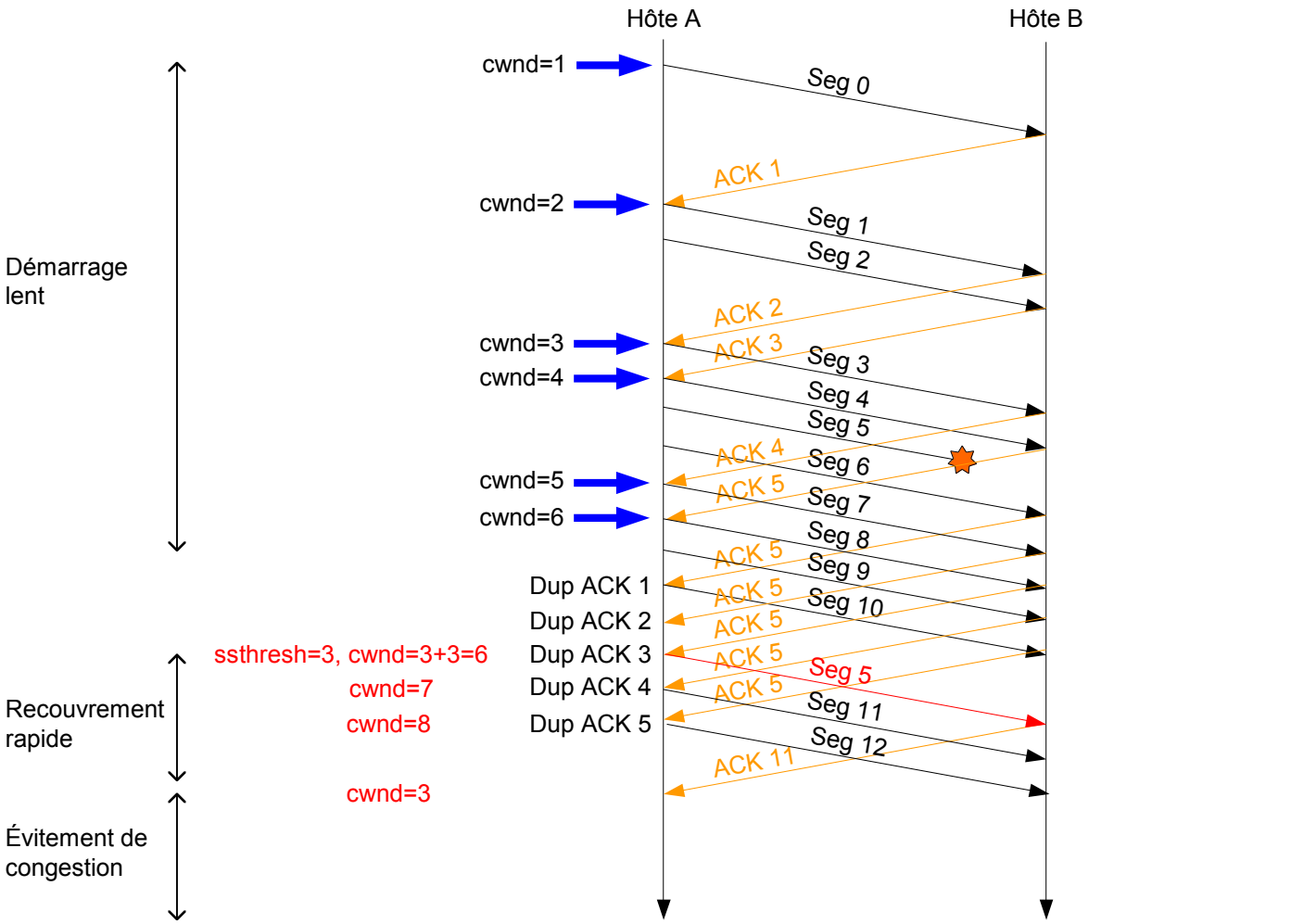
Algorithme Recouvrement Rapide

–Retransmettre rapidement le segment manquant

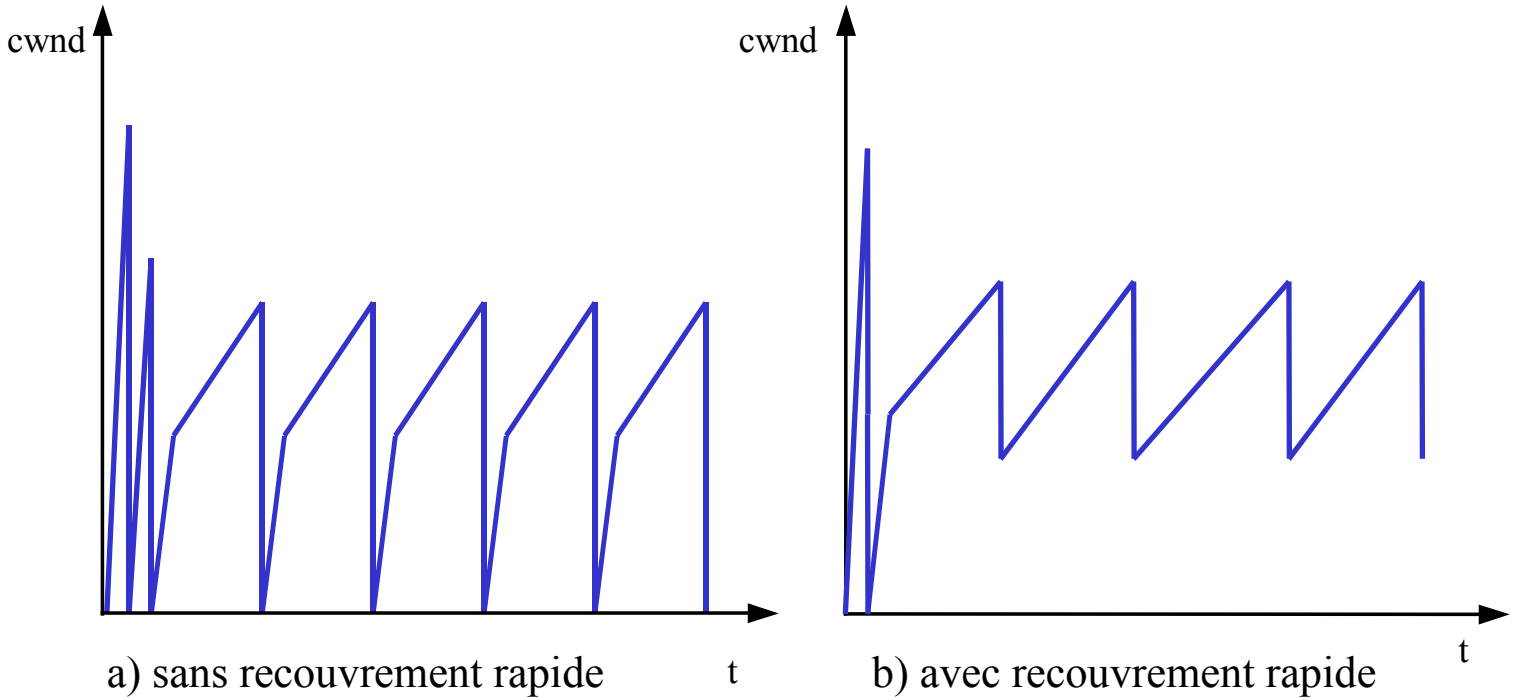
–Diminuer la fenêtre à la moitié

–Continuer avec Congestion Avoidance

Exemple de Recouvrement Rapide



Effet de recouvrement rapide sur le débit



Résumé du contrôle de congestion dans TCP

- La taille de la fenêtre de congestion est variée en fonction de la congestion du réseau
  - Une congestion est détectée à cause de pertes de paquets
- Slow Start : croissance exponentielle du débit
  - Au début de la connexion et après un timeout de retransmission (→ congestion sévère)
  - Permet d'augmenter rapidement le débit
- Congestion avoidance: croissance linéaire du débit
  - Dès que le débit s'approche à la capacité disponible
- Seuil d'évitement de congestion ssthresh
  - Indique la zone critique où une congestion est possible
  - Accroissement additif et décroissance multiplicative de ssthresh
  - Ssthresh oscille entre le débit maximum et la moitié de ce débit

Exercices 31, 33, 34, 35, 36, 37, 38

Gestion active des files d'attente

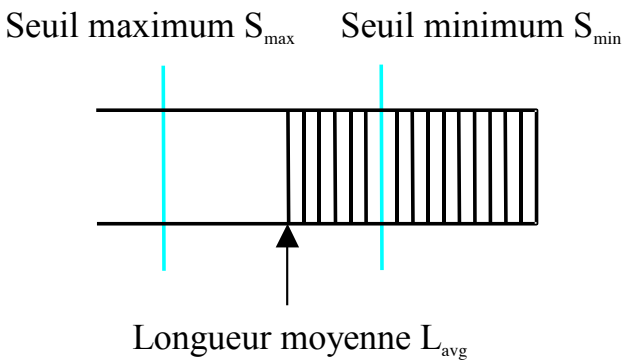
- Le contrôle de congestion de TCP est effectué par les systèmes terminaux
  - Les routeurs ne doivent pas être modifiés pour cette méthode
  - TCP réagit tard, quand la congestion s'est produite
    - Niveau élevé de la longueur des files d'attente
- Meilleure solution
  - Le réseau avertit TCP d'une congestion avant qu'elle ne se produise
  - Méthodes de gestion active des files d'attente

Random Early Detection (RED)

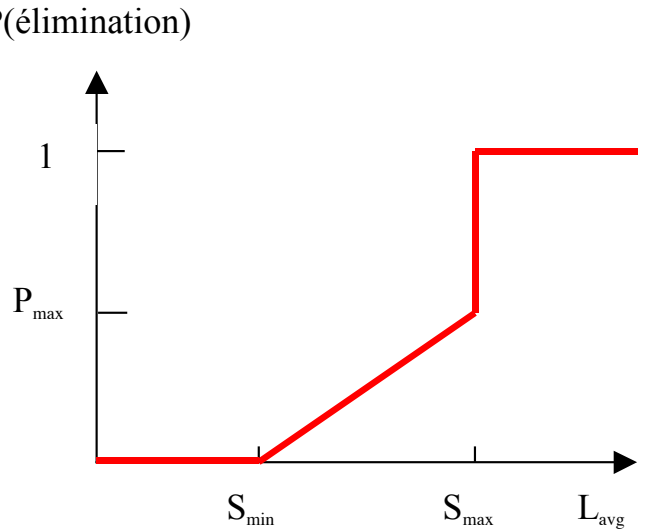
- Implémenté sur des routeurs
- Principe
  - Le routeur mesure la longueur moyenne de la file d'attente
  - Dès qu'une congestion s'annonce, le routeur avertit TCP en supprimant de manière aléatoire des paquets
  - La probabilité d'élimination est une fonction de la longueur de la file d'attente

Fonctionnement de RED

- L_{avg} : longueur moyenne de la file d'attente
- S_{min} : Si $L_{avg} > S_{min}$, RED commence à supprimer des paquets au hasard
- S_{max} : Si $L_{avg} > S_{max}$, RED supprime tous les paquets entrant



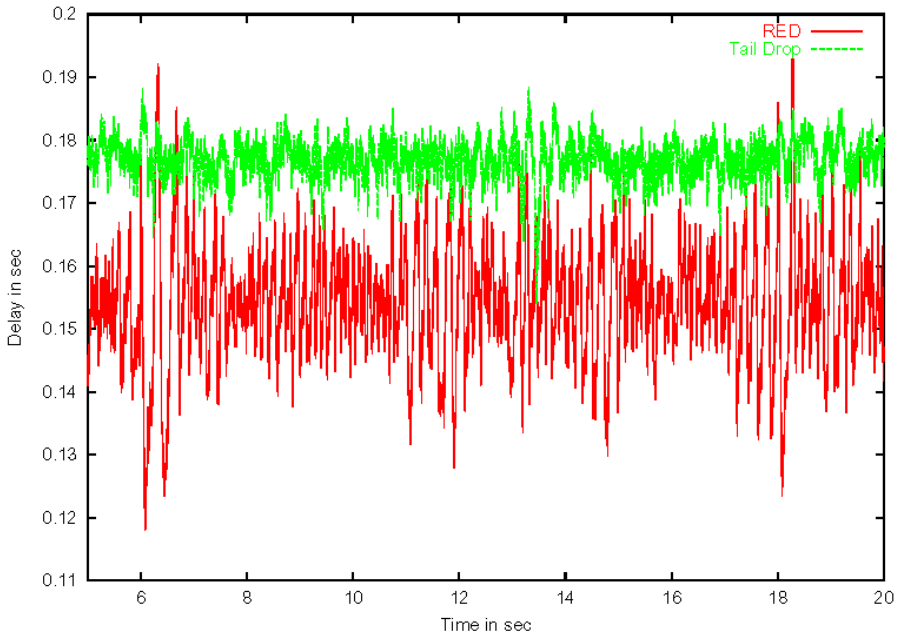
a) Seuils de la longueur de la file d'attente



b) Probabilité d'élimination d'un paquet en fonction de l'occupation moyenne de la file d'attente

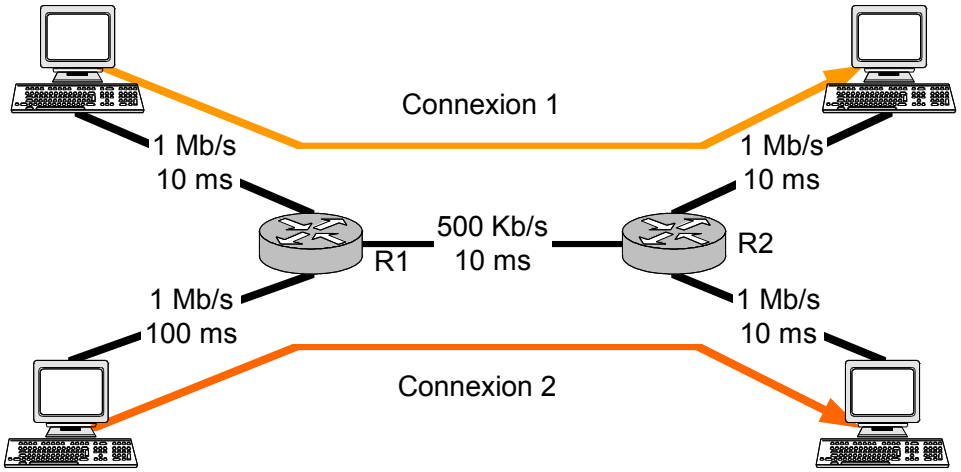
Comportement de RED

- RED permet de réduire la longueur moyenne des files d'attente en offrant le même throughput
  - Réduction des délais de transfert
- Problèmes
  - Choix des paramètres S_{min}, , S_{max}, P_{max}
  - Oscillation de la longueur d'une file d'attente



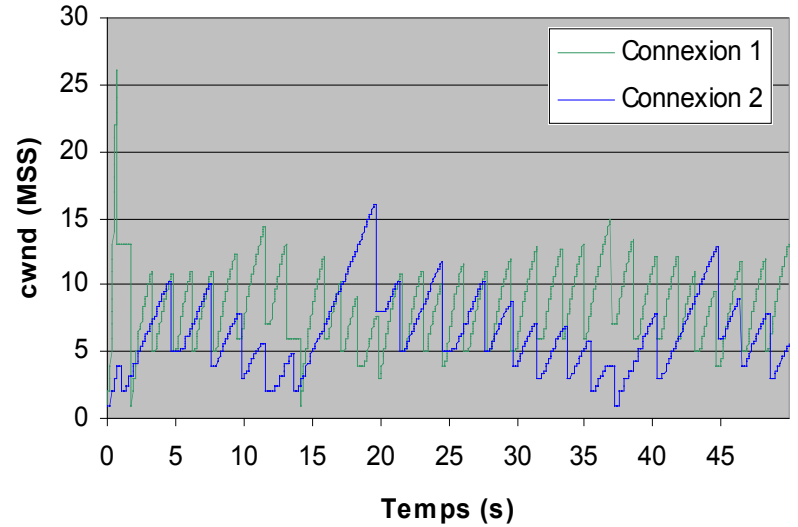
Comportement de flux dans un réseau

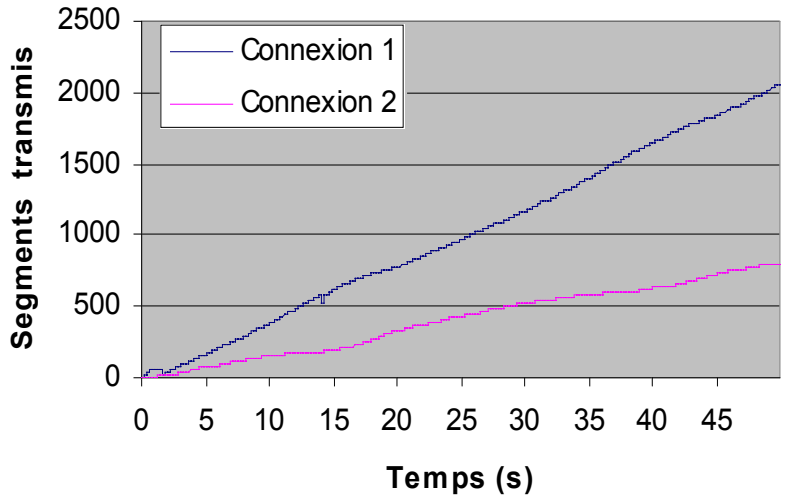
- Plusieurs connexions TCP
  - Problème principal : équité
    - Chaque connexion doit recevoir une partie équitable de la capacité du réseau
  - Difficile à obtenir (et à définir !)
    - Le comportement d'une connexion TCP dépend
      - De la capacité des liens traversés
      - Du délai aller retour du chemin
- Exemple



Résultats de simulation

- La connexion 'rapide' a une CWND plus grande pendant la plupart du temps
  - Connexion 1 réagit plus rapidement aux pertes et agrandit la fenêtre plus rapidement
- La connexion rapide obtient 2,5 fois le débit de la connexion lente
  - Un RTT plus petit signifie que avec même fenêtre, plus de données sont transmis par unité de temps

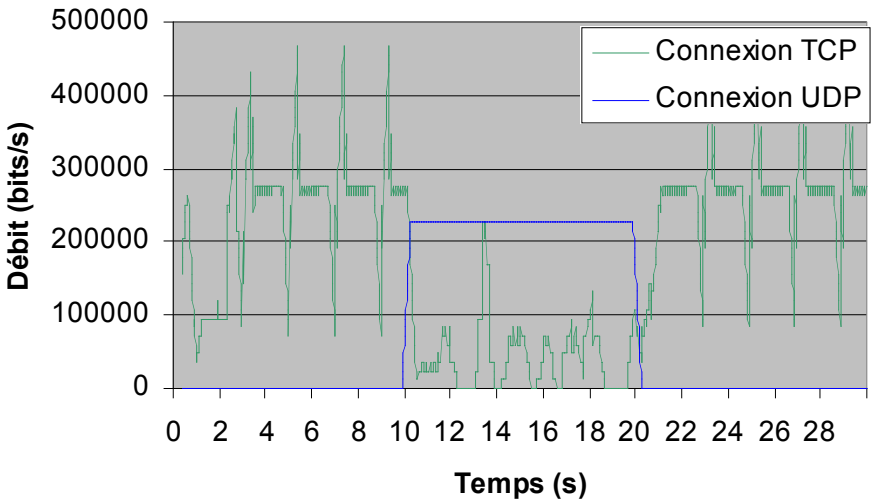




Flux TCP et UDP

- TCP adapte le débit aux conditions dans le réseau
  - Flux élastiques
  - Approprié pour le transfert de données
- UDP transmet avec le débit imposé par l'application
  - Flux non-élastiques
  - Adapté aux flux multimédia, qui nécessitent souvent un débit fixe
    - Exemple : voix sur IP : 64 kb/

➢ Le trafic TCP doit être protégé contre le trafic UDP
  - Contrôle d'accès pour le trafic UDP
  - Applications adaptatives qui simulent le comportement de



Exercice 42